Министерство образования Республики Беларусь
УО «Полесский государственный университет»

Л. П. ВОЛОДЬКО

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Часть 1

Учебно-методическое пособие
для студентов экономических и технических специальностей различных форм обучения
и слушателей факультета дополнительного образования

Пинск
ПолесГУ
2023

УДК 004.42(075.8)
ББК 32.973.26я73
В68

Р е ц е н з е н т ы:
кандидат экономических наук, доцент,
доцент кафедры промышленного маркетинга и коммуникаций БГЭУ
О. А. Синявская;
кандидат технических наук, доцент кафедры информационных технологий
и интеллектуальных систем ПолесГУ
Ю. М. Вишняков

У т в е р ж д е н о
научно-методическим советом ПолесГУ

Володько, Л. П.

В68 Основы алгоритмизации и программирования : учебно-методическое пособие для студентов экономических и технических специальностей различных форм обучения и слушателей факультета дополнительного образования : в 2 ч. / Л. П. Володько. – Пинск : ПолесГУ, 2023. – Ч. 1. – 151 с.

ISBN 978-985-516-754-0 (Ч. 1)
ISBN 978-985-516-753-3

Учебно-методическое пособие содержит материалы по описанию алгоритмов различными способами: словесным, графическим, на языке программирования C#. В пособии подробно описаны различные варианты структуры программы на языке программирования C#, предложена единая методика по выполнению лабораторной работы и написанию отчета по ней, приведены различные способы написания программ одной и той же задачи, описан программный код форматированного вывода данных на консоль, а также предлагается большое количество различных вариантов заданий к лабораторным работам.

Издание предназначено для студентов специальностей «Информационные системы и технологии», «Программное обеспечение информационных систем», «Программное обеспечение информационных технологий», «Искусственный интеллект», а также для других технических и экономических специальностей, предусматривающих изучение алгоритмизации и программирования.

УДК 004.42(075.8)
ББК 32.973.26я73

ISBN 978-985-516-754-0 (Ч. 1)
ISBN 978-985-516-753-3

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Важнейшей составляющей учебной работы студентов является выполнение лабораторных работ. Лабораторные работы позволяют не только закрепить и расширить теоретические знания, приобретенные в процессе изучения преподаваемых дисциплин, но и овладеть навыками самостоятельной работы, выработать умение формулировать суждения и выводы, логически, последовательно их излагать, более глубоко усвоить материал, получаемый на лекциях [10].

При подготовке предусмотренных учебным планом курсового проекта (курсовой работы), дипломного проекта (дипломной работы), магистерской диссертации, лабораторной работы, практической работы необходимо соблюдать общие требования, предъявляемые к научному исследованию: актуальность; научная новизна; лабораторная значимость; четкость и логическая последовательность изложения материала; краткость и точность формулировок; конкретность изложения результатов работы; аргументация выводов и обоснованность рекомендаций; правильность оформления.

При написании отчета по лабораторной работе особое внимание следует уделить языку и стилю изложения материала. Язык и стиль работы как часть письменной научной речи сложились под влиянием так называемого академического этикета, суть которого состоит в интерпретации собственной и привлекаемых точек зрения с целью обоснования истины. Для текста характерна смысловая законченность, целостность и связность, логический переход от одной мысли к другой.

Письменная речь имеет и чисто стилистические особенности. Основная стилевая черта такой речи – объективность изложения, которая вытекает из специфики познания, стремящегося установить истину. Обязательным условием объективности изложения материала является также указание на то, каков источник сообщения, кем высказана та или иная мысль, кому конкретно принадлежит определенное выражение. Стиль письменной речи – это безличный монолог. Поэтому изложение обычно ведется от третьего лица, т. к. внимание сосредоточено на содержании и логической последовательности сообщения, а не на субъекте. Такой подход устраняет необходимость в фиксации субъекта действия и тем самым избавляет от необходимости вводить в текст личные местоимения [10].

ГЛАВА 1
ОБЩИЕ ПОНЯТИЯ АЛГОРИТМА И СПОСОБЫ ИХ ОПИСАНИЯ

1.1 Свойства алгоритмов и способы их описания

Алгоритм – это строгая последовательность арифметических и логических действий, которая однозначно определяет процесс вычисления результата в зависимости от исходных данных. Слово «алгоритм» произошло от арабского слова «algoritmi», возникшего из имени хорезмийского математика IX века аль-Хорезми.

Основными свойствами алгоритма являются:

– **результативность –** алгоритм должен обеспечивать получение результата за конечное число шагов;

– **определенность** – применение алгоритма к однотипным исходным данным должно приводить к одному и тому же результату, независимо от пользователя;

– **массовость** – возможность применения алгоритма к целому классу однотипных задач, различающихся исходными данными.

Наиболее распространенными способами описания алгоритма являются:

– **словесно-формульный** – алгоритм записывается в виде текста с формулами по пунктам, определяющими последовательность действий;

– **аналитическая запись** – алгоритм записывается с использованием формул или специальных символов, например стенографических;

– **графический в виде блок-схемы** – алгоритм изображается специальными геометрическими фигурами (блоками), связанными направляющими стрелками;

– **алгоритмический язык** – это специальное средство, предназначенное для записи алгоритмов в аналитическом виде. Алгоритмические языки близки к математическим выражениям и к естественным языкам. Каждый алгоритмический язык имеет свой словарь. Алгоритм, записанный на алгоритмическом языке, выполняется по строгим правилам этого конкретного языка.

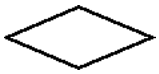
Образцом словесно-формульного описания алгоритма может служить следующий пример (таблица 1.1).

Таблица 1.1 – Словесное описание алгоритма

Условие задачи:	**Словесное описание алгоритма**
Вычислить значение выражения: **$y = 2 \cdot a - (x + 6)$**	1) ввести значения величин a и x; 2) выполнить сложение x и 6$\rightarrow$ x + 6; 3) выполнить умножение a на 2$\rightarrow$ 2a; 4) вычесть из полученного произведения значение полученной суммы $\rightarrow$ 2a – (x + 6); 5) вывести значение полученного результата y

Однако наиболее удобным и наглядным способом представления алгоритма является графический способ в виде блок-схемы. При этом каждый логически завершенный этап вычислительного процесса изображается в виде специального геометрического символа – **блока**. Наиболее часто используемые графические символы представлены в таблице 1.2.

Таблица 1.2 – Графические символы, применяемые при составлении блок-схем

Наименование блока	Обозначение	Выполняемое действие
1. Начало или конец алгоритма		Показывает начало или конец алгоритма
2. Ввод или вывод		Обеспечивает ввод или вывод данных в алгоритме
3. Арифметический		Выполняет арифметические вычисления
4. Логический		Выполняет проверку заданного логического условия
5. Модификации		Заголовок цикла «Для»
6. Предопределенный процесс		Вызов подпрограммы
7. Линии потока		Указывают связь и направление движения между блоками
8. Соединитель		Указывает связь между прерванными линиями потока
9. Межстраничный соединитель		Указывает связь между частями блок-схемы, которые расположены на разных страницах
10. Комментарий	 текст	Запись пояснения к блоку или к линии потока

При составлении блок-схемы алгоритма блоки записываются последовательно друг за другом и соединяются линиями потока информации, которые показывают направление движения по блок-схеме. Каждый блок алгоритма должен иметь вход и выход (исключение составляют блоки начала и конца алгоритма). При этом может быть несколько входящих в блок линий потока информации и только один выходящий поток (исключения составляют логический блок и блок модификации).

Несколько линий потока могут объединяться в одну линию, но одна линия потока информации не может разветвляться на несколько потоков. В блок-схеме любой путь движения из блока «Начало» алгоритма должен привести в блок «Конец» алгоритма. В общем случае любой алгоритм может состоять из трех частей: ввод исходных данных, вычисление требуемых величин и вывод полученных результатов. При этом каждый блок в блок-схеме должен быть пронумерован (кроме первого и последнего).

1.2 Базовые структуры схем алгоритмов

Существует три основные типовые структуры алгоритма:

- линейный вычислительный процесс;
- разветвляющийся вычислительный процесс;
- циклический вычислительный процесс.

Любой алгоритм сложной структуры может быть получен путем комбинированного использования типовых структур. Рассмотрим алгоритм линейной структуры (рисунок 1.1).

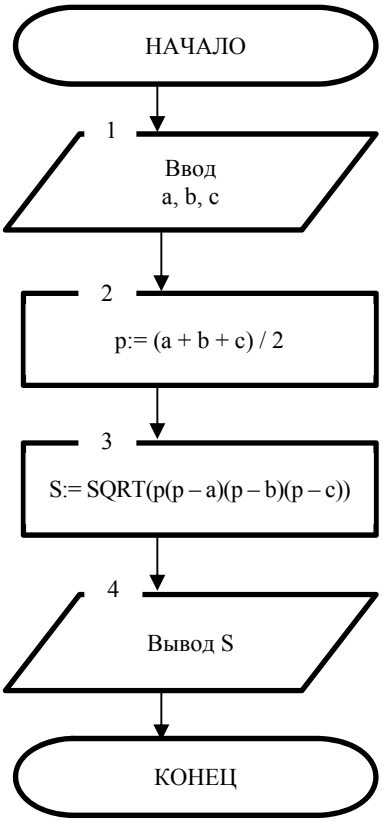


Рисунок 1.1 – Линейный алгоритм вычисления площади треугольника

В линейном вычислительном процессе все действия выполняются в строгой последовательности друг за другом. Таким образом, существует только один путь, по которому можно пройти из блока «Начало» в блок «Конец» алгоритма, т. е. выполнить алгоритм.

Пример. Составить алгоритм вычисления площади треугольника с заданными сторонами **a**, **b** и **c**. Как известно из курса геометрии, площадь треугольника, заданного длинами сторон, можно вычислить, используя формулу Герона. Разрабатываемый алгоритм (рисунок 1.1) должен обеспечивать ввод исходных данных, т. е. значений длин сторон треугольника a, b и c (блок 2). Затем, используя введенные значения, вычислять полупериметр p (блок 3) и значение площади треугольника S (блок 4). SQRT – это имя функции извлечения квадратного корня. После завершения вычислений необходимо вывести результат, т. е. полученное значение площади треугольника S (блок 5). Однако решение абсолютного большинства инженерных задач невозможно свести к алгоритмам линейной структуры. В блоке 2 и 3 используется операция присваивания, которая записывается в виде последовательности двух символов (двоеточия и равно – :=) и состоит из двух частей (левой и правой). Операция присваивания выполняется следующим образом: выбирается свободная ячейка памяти, ей присваивается имя (левая часть), в ячейку памяти с этим именем записывается число (вычисленное по формуле или указанное), находящееся в правой части. Теперь перейдем к рассмотрению разветвляющихся алгоритмов.

Разветвляющийся вычислительный процесс позволяет выбрать один из нескольких вариантов решения поставленной задачи в зависимости от выполнения некоторых условий. Таким образом, существует несколько различных путей, по которым можно пройти из блока «Начало» в блок «Конец» алгоритма. Для реализации процесса выбора одного из двух вариантов решения используется логический блок (блок проверки условий), приведенный на рисунке 1.2.

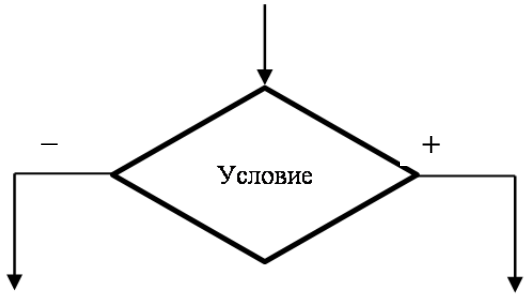


Рисунок 1.2 – Изображение логического блока

При входе в этот блок выполняется проверка логического условия (обычно математического неравенства). Если результат проверки условия «Истина», т. е. условие выполняется, то происходит переход к выполнению блоков, стоящих по ветви «+» (Да). В противном случае, т. е. когда проверяемое условие не выполняется, происходит переход к выполнению блоков, стоящих по ветви «–» (Нет) (рисунок 1.2). Если требуется выбрать один из трех и более вариантов решения, то необходимо использовать вложенные логиче-

ские блоки. В случае, когда действия должны выполняться только по одной из ветвей логического блока, их необходимо реализовывать по ветви «+», подобрав соответствующее условие (рисунок 1.5). Обратите внимание на неправильное и правильное использование логического блока (рисунок 1.3).

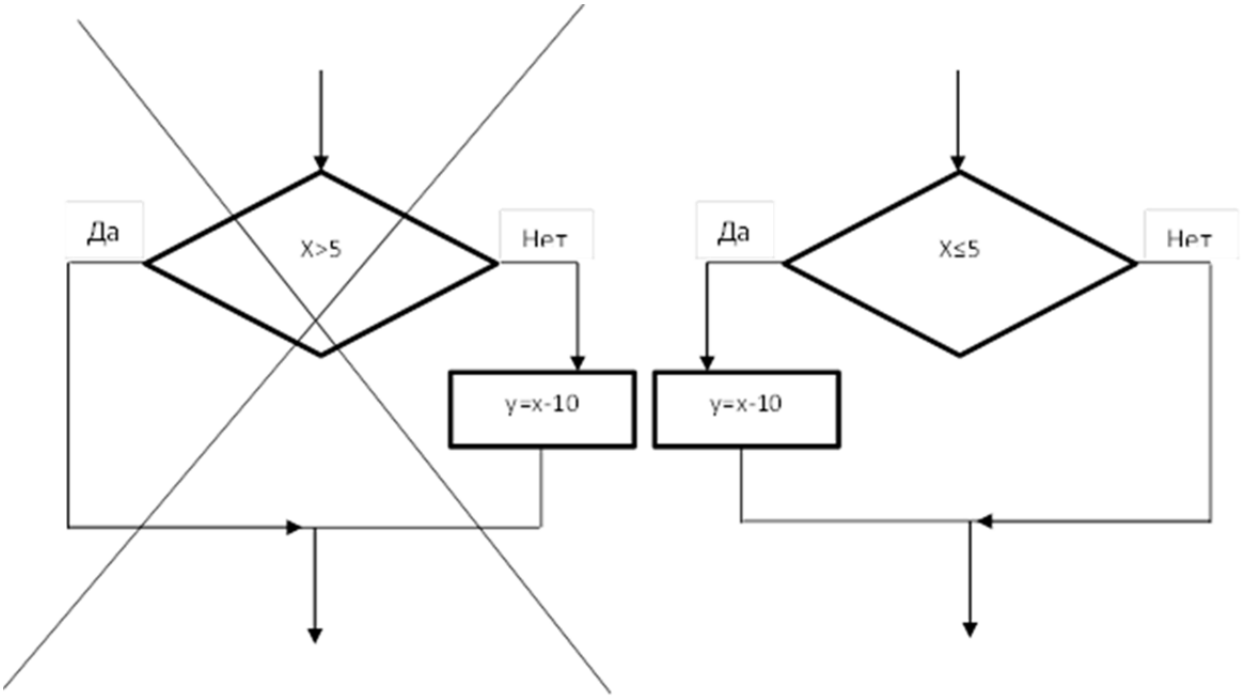


Рисунок 1.3 – Варианты неправильного и правильного использования логического блока

При выполнении вычислений необходимо учитывать область определения математических функций. Следовательно, вначале необходимо проверить возможность вычисления данного математического выражения при текущих значениях исходных данных, т. е. проверить «**аномалию**».

К наиболее часто встречающимся «аномалиям» относятся: операция деления (на 0 делить нельзя), вычисление квадратного корня (подкоренное выражение должно быть ≥ 0), вычисление логарифма (выражение под знаком логарифма должно быть > 0), вычисление tg, ctg.

В случае возникновения «аномалии» (невозможно выполнить вычисления) необходимо пропустить все действия, которые зависят от вычисляемой величины, и перейти в ту часть алгоритма, где можно продолжить вычисления.

Пример. Составить блок-схему алгоритма, вычисляющего значение y по одной из трех формул (рисунок 1.4.), в зависимости от значения x.

$$y = \begin{cases} x^2 - 4x, \text{если } x \geq 4 \\ \sqrt{x - 1.5}, \text{если } 2 \leq x < 4 \\ \dfrac{3x}{x+1}, \text{если } x < 2 \end{cases}$$

Рисунок 1.4 – Формула вычисления сложной функции

Блок-схема алгоритма приведена на рисунке 1.5.

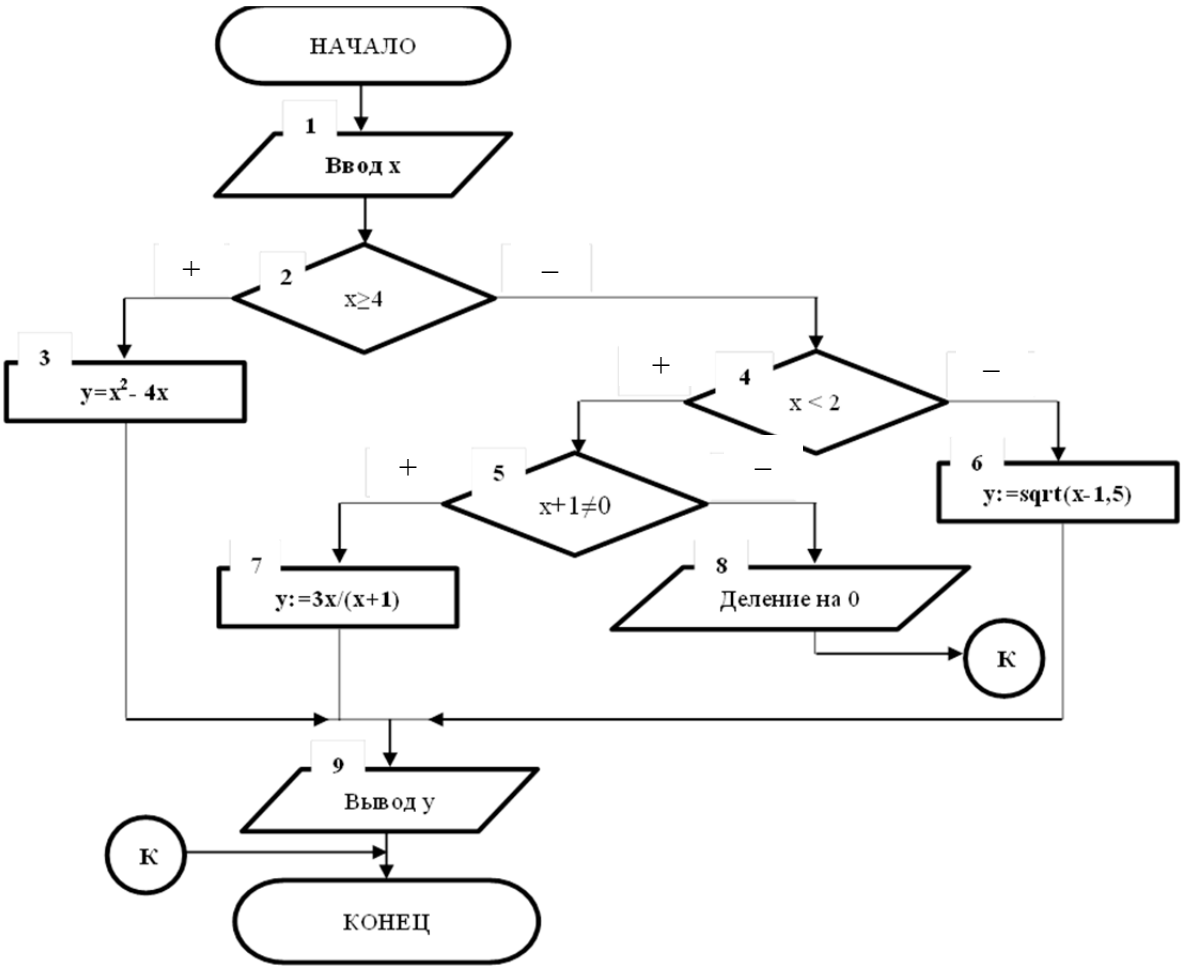


Рисунок 1.5 – Алгоритм разветвляющейся структуры

Для выбора формул, по которым будет вычисляться значение ***y***, необходимо проверить три условия. Такие ограничения являются взаимоисключающими условиями, достаточно проверить только два условия, т. е. если не выполняются два условия, то третье выполняется автоматически, его не надо проверять. Исходными данными для решения поставленной задачи является значение ***x***, которое вводится в блоке 1. Затем в блоке 2 проверяется первое ограничение ($x \geq 4$), в случае его выполнения происходит переход к блоку 3, в котором вычисляется значение ***y*** по первой формуле. Если результатом проверки первого условия является слово «ЛОЖЬ» (это означает, что $x < 4$), то необходимо проверить одно из двух оставшихся условий. Обычно выбирается более «короткое» ограничение (в блоке 4 проверяется третье ограничение). Если $x < 2$, то необходимо вычислять ***y*** по третьей формуле, в которой может возникнуть «аномалия» – деление на 0, поэтому в блоке 5 проверяется неравенство знаменателя дроби нулю. И только после этого в блоке 7 вычисляется значение ***y***. В том случае, когда невозможно вычислить значение ***y*** по третьей формуле, выводится сообщение о возникновении ошибки (блок 9) и выполняется переход на конец алгоритма. И, наконец, если значение ***x*** не удовлетворяет ни первому, ни третьему ограничению, значит, выполняется второе ограничение, ***y*** вычисляется по второй формуле (блок 6). Эта формула также содержит «аномалию» – подкоренное выражение должно быть ≥ 0. Однако проверять её не надо, т. к. при любом значении ***x,*** попадающем в интервал [2; 4], подкоренное выражение всегда > 0. Независимо от того, по какой формуле будет вычислено значение ***y***, его надо вывести на экран, поэтому выходы блоков 3, 6 и 7

объединяются и происходит переход к блоку 9, в котором выводится ***y***. За одно выполнение алгоритма может быть вычислено только одно значение ***y*** в зависимости от введенного значения ***x***. Всего же существует четыре варианта работы алгоритма в зависимости от значения ***x***. При этом если возникнет «аномалия», ***y*** вычислен не будет.

Рассмотрим циклические алгоритмы.

Алгоритм называется циклическим, если в нем имеются действия или наборы действий, которые необходимо выполнить более одного раза. Повторяющиеся алгоритмические действия являются телом цикла.

Дополнительно каждый цикл имеет условие, по которому выполнение циклического алгоритма заканчивается. Каждый циклический алгоритм имеет в своем составе условие цикла, т. е. логическое выражение, результат проверки которого определяет, будет выполняться тело цикла еще раз или цикл будет завершен. По способу обработки все циклические алгоритмы делятся на три группы: цикл с предусловием, цикл с постусловием и безусловный цикл. В циклических алгоритмах с постусловием условие продолжения проверяется до обработки тела цикла, т. е. наличествует необходимость повторения обработки цикла. Рассмотрим вывод на печать чисел от –5 до 0 как пример циклических алгоритмов с предусловием (рисунок 1.6).

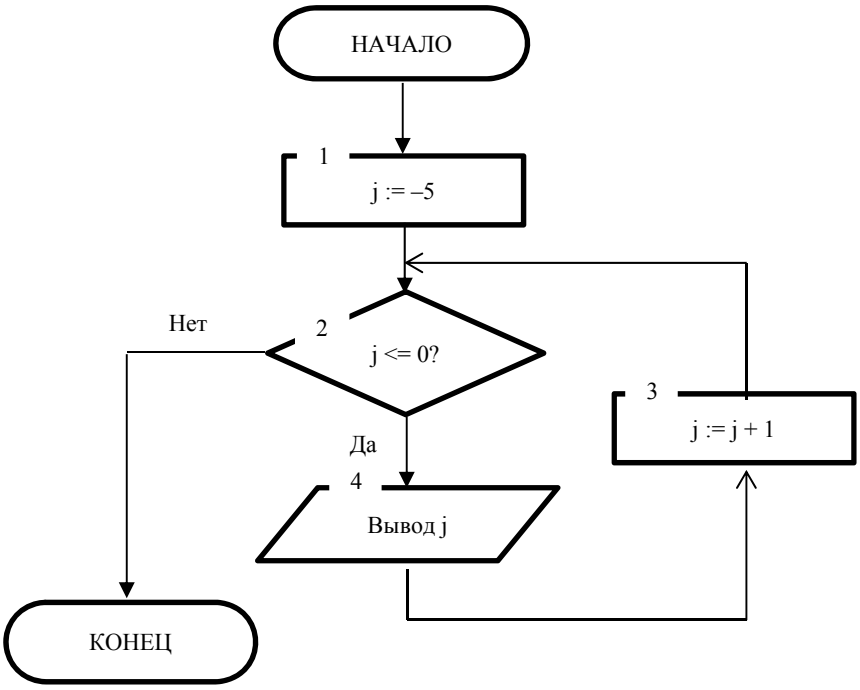


Рисунок 1.6 – Алгоритм вывода пяти чисел

Элементы алгоритма, изображенного на рисунке 1.6:

1. Задаем начальное значение базовой переменной *j*, равное –5.
2. Проверяем условие цикла, если условие положительное, то переходим по ветке «Да» и тело цикла выполняется первый раз.
3. Далее прибавляем к переменной *j* единицу, снова проверяем условие цикла.
4. Цикл продолжает выполняться, пока значение *j* меньше нуля или равно ему, в противном случае выходим из цикла по ветке «Нет».

В цикле с постусловием проверка условия выполняется после первой обработки тела цикла и контролирует выход из него. Разберем расчет суммы от 1 до числа **N** как пример циклических алгоритмов, в которых используются постусловие (рисунок 1.7).

1. Вводим конечное число расчета суммы **N** и задаем нулевые начальные значения итоговой суммы **SUM** и счетчика цикла *i*.

2. Цикл выполняется до первой проверки условия.
3. Проверяем условие цикла, т. е. значение счетчика *i* меньше или равно **N**.
4. Если результат условия положительный, выполняем цикл еще раз, иначе заканчиваем цикл и выводим сумму на дисплей или печать.

Безусловный цикл обычно используется в алгоритмах, когда нужное количество выполнений цикла заранее известно, и очень часто применяется при работе с массивами. Такой алгоритм содержит три обязательных элемента: стартовое значение, которое называют параметром цикла, т. к. именно эта переменная изменяется после каждого выполнения цикла и определяет момент его завершения; значение, при котором цикл завершается; шаг цикла (рисунок 1.8).

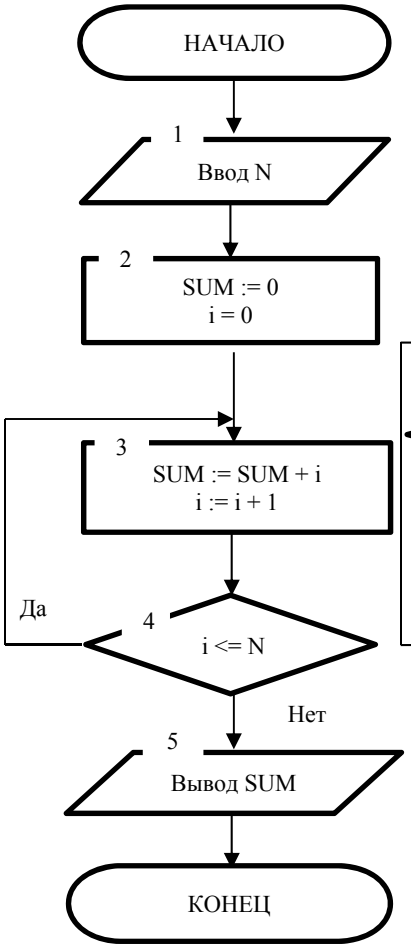


Рисунок 1.7 – Алгоритм расчета суммы элементов от 1 до N

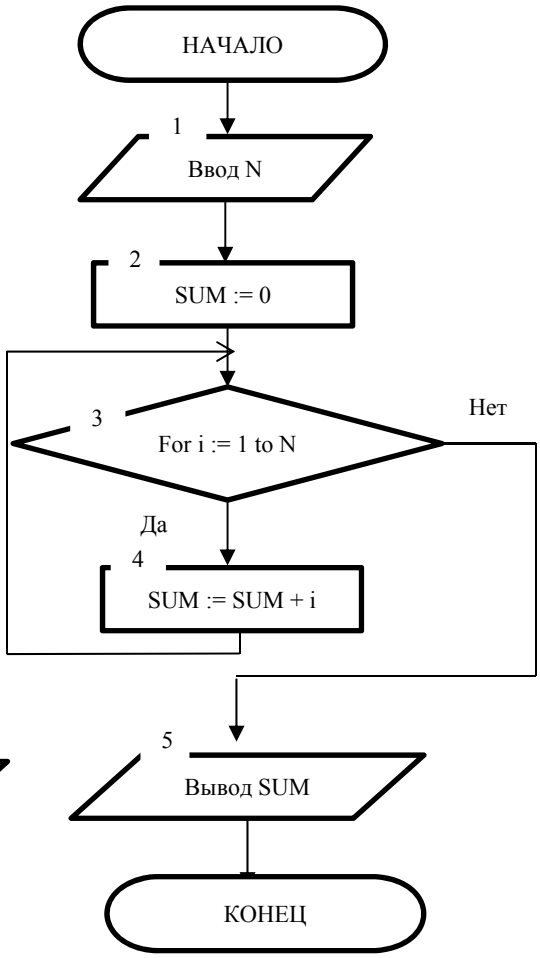


Рисунок 1.8 – Алгоритм расчета суммы элементов от 1 до N

На каждом шаге программа проверяет, не превосходит ли стартовое значение конечное. И если да, то цикл завершается. В противном случае к стартовому значению прибавляем величину шага и цикл повторяется. Особо следует отметить, что любой безусловный цикл можно заменить условным с пред- или постусловием. При составлении циклических алгоритмов необходимо придерживаться двух обязательных условий. *Первое*: для окончания цикла необходимо, чтобы содержимое тела влияло на пост- или предусловие, иначе мы в итоге можем получить бесконечный цикл. Но для некоторых программных задач такие циклы применяются. Как пример циклических алгоритмов, выполняющихся бесконечно, можно привести операционную систему Windows, где используется бесконечный цикл опроса положения мыши для определения действий пользователя. *Второе*: переменные, передаваемые в цикл, должны обеспечивать хотя бы одно его выполнение.

ГЛАВА 2
ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ C#

2.1 Состав языка

Любой язык программирования можно уподобить очень примитивному иностранному языку с жесткими правилами без исключений. Изучение иностранного языка обычно начинают с алфавита, затем переходят к словам и законам построения фраз, и только в результате длительной практики и накопления словарного запаса появляется возможность свободно выражать на этом языке свои мысли. Примерно так же необходимо изучать и язык программирования C #.

2.1.1 Алфавит и лексемы

Алфавит – совокупность допустимых в языке символов. Алфавит языка C# включает:

1) прописные и строчные латинские буквы и буквы национальных алфавитов (включая кириллицу);

2) арабские цифры от 0 до 9, шестнадцатеричные цифры от A до F;

3) специальные знаки: " { } , | ; [ ] ( ) + - / % * . \ ' : ? < = > ! & ~ ^ @ _;

4) пробельные символы: пробел, символ табуляции, символ перехода на новую строку.

Из символов алфавита формируются лексемы языка: идентификаторы, ключевые (зарезервированные) слова, знаки операций, константы, разделители (скобки, точка, запятая, пробельные символы). Границы лексем определяются другими лексемами, такими как разделители или знаки операций. В свою очередь лексемы входят в состав выражений (выражение задает правило вычисления некоторого значения) и операторов (оператор задает законченное описание некоторого действия).

2.1.2 Идентификаторы

Идентификатор (имя) – это имя программного элемента: константы, переменной, метки, типа, класса, объекта, метода, пространства и т. д.

Требования к выбору идентификаторов (имен):

1) имя должно начинаться с буквы или знака подчеркивания (_);
2) имя должно содержать только буквы, знак подчеркивания и цифры;
3) прописные и строчные буквы различаются;
4) длина имени практически не ограничена;
5) имена не должны совпадать с ключевыми словами, однако допускается: @if, @float;
6) идентификатор может включать латинские буквы и буквы национальных алфавитов;
7) пробелы и другие разделители внутри имен не допускаются;
8) в именах можно использовать управляющие последовательности Unicode.

Язык C# не налагает никаких ограничений на длину имен, однако для удобства чтения и записи кода не стоит делать их слишком длинными. Для улучшения читабельности кода программным элементам следует давать осмысленные имена, составленные в соответствии с определенными правилами. Существует несколько видов нотаций – соглашений о правилах создания имен. В нотации Pascal каждое слово, входящее в идентификатор, начинается с заглавной буквы. Например: Age, LastName, TimeOfDeath. Венгерская нотация отличается от предыдущей наличием префикса, соответствующего типу величины. Например: fAge, sName, iTime. В нотации Camel с заглавной буквы начинается каждое слово идентификатора, кроме первого. Например: age, lastName, timeOfDeath. Наиболее часто используются нотации Pascal или Camel. Однако в простых программах будут использоваться однобуквенные переменные.

2.1.3 Ключевые слова, знаки операций и разделители

Ключевые слова – это зарезервированные идентификаторы, которые имеют специальное значение для компилятора (например, static, int и т. д.). Ключевые слова можно использовать только по прямому назначению. Однако если перед ключевым словом поставить символ @, например @int, @static, то полученное имя можно использовать в качестве идентификатора. С полным перечнем ключевых слов и их назначением можно ознакомиться в справочной системе C#.

Таблица 2.1 – Ключевые слова C#

Имя	Имя	Имя	Имя	Имя
abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit	in	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	void
volatile	while			

Знак операции – один или более символов, определяющих действие над операндами. Внутри знака операции пробелы не допускаются.

Операции делятся на *унарные* (с одним операндом) (таблица 2.2), *бинарные* (с двумя операндами) (таблица 2.3) и *тернарную* (с тремя операндами).

Таблица 2.2 – Унарные операции

Знак операции	Действие
+	Унарный плюс
–	Унарный минус
!	Логическое отрицание НЕ
~	Поразрядное НЕ
++	Инкремент
--	Декремент
true	Истина
false	Ложь

Таблица 2.3 – Бинарные операции

Знак операции	Действие
=	Присваивание (базовая операция)
+	Сложение
–	Вычитание
*	Умножение
/	Деление
%	Деление по модулю
&	Логическое И
|	Логическое ИЛИ
^	Логическое исключающее ИЛИ
<<	Сдвиг влево
>>	Сдвиг вправо
= =	Равно
!=	Не равно
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно

Работу базовой операции присваивания рассмотрим на примере оператора присваивания (X = 5). Операторы языка C# будут рассмотрены в следующих разделах. Этот оператор состоит из двух операндов: левого, правого и операции присваивания. Этот оператор выполняется следующим способом: выбирается свободная ячейка памяти, ей присваивается имя X и в эту ячейку записывается число, равное 5 (пяти), то есть содержимое правой части записывается в ячейку с именем левой части.

Тернарная условная операция (от лат. ternarius – «тройной») – реализованная во многих языках программирования **операция**, возвращающая свой второй или третий операнд в зависимости от значения логического выражения, заданного первым операндом. В C# только одна тернарная операция, которую рассмотрим ниже.

Разделители используются для разделения или, наоборот, группирования элементов. Примеры разделителей: скобки, точка, запятая.

Большинство операций в языке C#, их приоритет и порядок наследованы из языка C++. Однако есть и различия: например, нет операции « , », позволяющей вычислять список выражений, добавлены уже упоминавшиеся операции checking и unchecking, применимые к выражениям.

Операции при выполнении имеют приоритеты. Приведем **таблицу приоритетов** операций, в каждой строке которой собраны операции одного приоритета, а строки следуют в порядке приоритетов, от высшего к низшему (таблица 2.4).

Таблица 2.4 – Приоритеты операций языка C#

Приоритет	Категория	Операции	Порядок
0	Первичные	(expr) x.y f(x) a[x] x++ x-- new sizeof(t) typeof(t) checked(expr) unchecked(expr)	Слева направо
1	Унарные	+ - ! ~ ++x --x (T)x	Слева направо
2	Мультипликативные (Умножение, деление)	* / %	Слева направо
3	Аддитивные (Сложение, вычитание)	+ -	Слева направо
4	Сдвиг	<< >>	Слева направо
5	Отношения, проверка типов	< > <= >= is as	Слева направо
6	Эквивалентность	= = !=	Слева направо
7	Логическое И (AND)	&	Слева направо
8	Логическое исключающее ИЛИ (XOR)	^	Слева направо
9	Логическое ИЛИ (OR)	|	Слева направо
10	Условное И	&&	Слева направо
11	Условное ИЛИ	||	Слева направо
12	Условное выражение	? :	Справа налево
13	Присваивание	= *= /= %= += -= <<= >>= &= ^= |=	Справа налево

2.1.4 Константы

Константа (литерал) – это лексема, представляющая изображение фиксированного, числового, строкового или символьного значения. Константы – это постоянные значения, которые известны во время компиляции и не изменяются во время выполнения программы. Константы бывают двух видов: константы-литералы и именованные константы. Константы-литералы представляют собой сами значения и приведены в таблице 2.5, а именованные константы задаются с помощью ключевого слова **const: const double Pi = 3.14; const int c1 = 9090, c2 = 7099; const char point = '.'**.

Константы делятся на 5 групп: логические, целые, вещественные, символьные, строковые (таблица 2.5).

Таблица 2.5 – Константы языка C#

Константа	Описание	Примеры		
		Первый	Второй	Третий
1	2	3	4	5
Логическая	true (истина) или false (ложь)	true	false	–
Целая	Десятичная: последовательность десятичных цифр (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), за которой может следовать суффикс (U, u, L, 1, UL, Ul, uL, ul, LU, Lu, 1U, lu)	8 8u	0 0Lu	175999 175999L
	Восьмеричная: символ 0 (ноль), за которым следуют восьмеричные цифры (0, 1, 2, 3, 4, 5, 6, 7)	016	0177	099
	Шестнадцатеричная: символы 0x или 0X, за которыми следуют шестнадцатеричные цифры (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F), а за цифрами, в свою очередь, может следовать суффикс (U, u, L, 1, UL, Ul, uL, ul, LU, Lu, 1U, lu)	0xA 0xAU	0x199 0x199LU	0X11FF 0X11FFl

Окончание таблицы 2.5

1	2	3	4	5
Вещественная	С фиксированной точкой: [цифры] [.] [цифры] [суффикс] Суффикс – один из символов F, f, D, d, M, m С порядком: [цифры][.][цифры]{E|e}[+|-][цифры] [суффикс] Суффикс – один из символов F, f, D,d, M, m	5.7 5.7F 5F 0.2E6 0.2E6D	.001 .001d .001f .11e+3 .11E-3	35 35 35m 5E-10 5E10
Символьная	Символ, заключенный в апострофы	'A' '\0' '\xF' '\uA81B'	'Ю' '\n' '\x74' -	'* ' - - -
Строковая	Последовательность символов, заключенных в кавычки	"Здесь был Vasia" "\tЗначение r = \0xF5 \n" "Здесь был \u0056\u0061" "C:\\ temp\\ filel.txt" @"C:\temp\filel.txt"		
Константа null	Ссылка, которая не указывает ни на какой объект	null		

Более подробно остановимся на таблице 2.5. Логических констант всего две. Они широко используются в качестве признаков наличия или отсутствия чего-либо.

Целые константы могут быть представлены в десятичной, восьмеричной либо в шестнадцатеричной системе счисления; а вещественные – только в десятичной системе счисления, но в двух формах: с фиксированной точкой и с порядком. Вещественная константа с порядком представляется в виде мантиссы и порядка. Мантисса записывается в двух формах: с фиксированной точкой и с порядком. Мантисса записывается слева от знака экспоненты (E или e), порядок – справа от знака. Значение константы определяется как произведение мантиссы и возведенного в указанную в порядке степень числа 10 (например, $1.3e2 = 1,3 \cdot 10^2 = 130$). При записи вещественного числа могут быть опущены либо целая часть, либо дробная, но, конечно, не обе сразу.

Примечание. Пробелы внутри числа не допускаются. Для отделения целой части от дробной используется не запятая, а точка. Символ E не представляет собой знакомое всем из математики число e, а указывает, что далее располагается степень, в которую нужно возвести число 10. Если требуется сформировать отрицательную целую или вещественную константу, то перед ней ставится знак унарной операции изменения знака (-), например: -218, -022, -0x3C, -4.8, -0.1e4.

Когда компилятор распознает константу, он отводит ей место в памяти в соответствии с ее видом и значением. Если по каким-либо причинам требуется явным образом задать, сколько памяти следует отвести под константу, используются суффиксы, описания которых приведены в таблице 2.6.

Таблица 2.6 – Суффиксы целых и вещественных констант

Суффикс	Значение
L, l	Длинное целое (long)
U, u	Беззнаковое целое (unsigned)
F, f	Вещественное с одинарной точностью (float)
D, d	Вещественное с двойной точностью (double)
M, m	Финансовое десятичного типа (decimal)

Символьная константа представляет собой любой символ в кодировке Unicode. Символьные константы записываются в одной из четырех форм, представленных в таблице 2.5:

– «обычный» символ, имеющий графическое представление (кроме апострофа и символа перевода строки), – 'A', 'ю', ' * ' ;

– управляющая последовательность – '\0', '\n';

– символ в виде шестнадцатеричного кода – '\xF', '\x74';

– символ в виде escape-последовательности Unicode – '\uA81B'.

Управляющей последовательностью, или простой escape-последовательностью, называют определенный символ, предваряемый обратной косой чертой. Управляющая последовательность интерпретируется как одиночный символ и используется для представления:

– кодов, не имеющих графического изображения (например, \n – переход в начало следующей строки);

– символов, имеющих специальное значение в строковых и символьных константах (например, апострофа ').

В таблице 2.7 приведены допустимые значения последовательностей. Если непосредственно за обратной косой чертой следует символ, не предусмотренный таблицей, возникает ошибка компиляции.

Таблица 2.7 – Управляющие последовательности в C#

Вид	Наименование
\a	Звуковой сигнал
\b	Возврат на шаг
\f	Перевод страницы (формата)
\n	Перевод строки
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\\	Обратная косая черта
\’	Апостроф
\”	Кавычка
\0	Нуль-символ

Символ, представленный в виде шестнадцатеричного кода, начинается с префикса \0x, за которым следует код символа. Числовое значение должно находиться в диапазоне от 0 до 2^{16}- 1, иначе возникает ошибка компиляции. Escape-последовательности Unicode служат для представления символа в кодировке Unicode с помощью его кода в шестнадцатеричном виде с префиксом \u или \U (например, \u00F2, \U00010011). Коды в диапазоне от \U10000 до \U10FFFF представляются в виде двух последовательных символов; коды, превышающие \U10FFFF, не поддерживаются.

Управляющие последовательности обоих видов могут использоваться и в строковых константах, называемых иначе строковыми литералами. Например, если требуется вывести несколько строк, можно объединить их в один литерал, отделив одну строку от другой символами \n:

"Никто не доволен своей \nвнешностью, но каждый доволен\nсвоим умом"

Этот литерал при выводе будет выглядеть так:

Никто не доволен своей
внешностью, но каждый доволен
своим умом

Другой пример: если внутри строки требуется использовать кавычку, ее предваряют косой чертой, по которой компилятор отличает ее от кавычки, ограничивающей строку:

"Издательский дом \"Питер\""

Как видите, строковые литералы с управляющими символами несколько теряют в читабельности, поэтому в C# введен второй вид литералов – дословные литералы (verbatim strings). Эти литералы предваряются символом @, который отключает обработку управляющих последовательностей и позволяет получать строки в том виде, в котором они записаны. Например, два приведенных выше литерала в дословном виде выглядят так:

@"Никто не доволен своей
внешностью, но каждый доволен
своим умом"
@"Издательский дом "Питер""

Чаще всего дословные литералы применяются в регулярных выражениях и при задании полного пути файла, поскольку в нем присутствуют символы обратной косой черты, которые в обычном литерале пришлось бы представлять с помощью управляющей последовательности. Сравните два варианта записи одного и того же пути:

"C:\\app\\bin\\debug\\a.exe"
@"C:\app\bin\debug\a.exe"

Строка может быть пустой (записывается парой смежных двойных кавычек " "), пустая символьная константа недопустима. Константа *nul1* представляет собой значение, задаваемое по умолчанию для величин так называемых ссылочных типов.

2.1.5 Комментарии

Комментарии – это текстовые пометки в исходном коде приложения, которые не влияют на сам исходный код, а служат только для повышения удобочитаемости и понятности кода. В них чаще всего содержатся пояснения к сложным участкам кода.

Несмотря на то, что комментарии не несут никакой полезной нагрузки для самого приложения, они просто невероятно полезны для долгосрочного поддержания проекта, потому что бывает очень сложно разобраться в чужом коде. А если принять за правило утверждение, что любой написанный самим тобой код при прошествии полугода становиться «чужим», то писать комментарии полезно и нужно. Да, есть противоположное мнение, что код должен быть самодокументированным (то есть понятным для чтения без комментариев), но даже к такому коду лучше оставлять комментарии.

Итак, в языке C# используются два вида комментариев:

– однострочные (//) – распространяются до перехода на новую строку;

– ногострочные (/* */) – распространяются на все строки в промежутке между ключевыми символами.

Примеры:

1. // Это однострочный комментарий с начала строки.
2. *int i = 5; //* Это однострочный комментарий после команды.
3. /* Это многострочный комментарий в одной строке. Но зачем?.. */.
4. /*.
   Это
   тоже
   многострочный
   комментарий
   /.

Комментарии в первую очередь нужны, чтобы оставлять заметки, которые могут пригодиться в будущем вам или другому программисту, который будет поддерживать ваш код. Кроме того, комментарии часто используются на этапе отладки приложения, чтобы временно пропускать некоторые команды. По оформлению комментариев существует рекомендация добавлять пробел после //, начинать писать новое предложение комментария

с большой буквы, а длинное перенесенное с новой строки – с маленькой, в конце комментария ставить точку.

Примеры:

1. //некрасивый комментарий без пробела и точки.
2. // Хорошо оформленный длинный комментарий,
// который может быть на несколько строк.

2.2 Типы данных

Данные, с которыми работает программа, хранятся в оперативной памяти. Естественно, что компилятору необходимо точно знать, сколько места они занимают, как именно закодированы и какие действия с ними можно выполнять. Все это задается при описании данных с помощью типа.

Тип данных однозначно определяет внутреннее представление данных, а следовательно, и множество их возможных значений, допустимые действия над данными (операции и функции). Например, целые и вещественные числа, даже если они занимают одинаковый объем памяти, имеют совершенно разные диапазоны возможных значений; целые числа можно умножать друг на друга, а, например, символы – нельзя.

Каждое выражение в программе имеет определенный тип. Величин, не имеющих никакого типа, не существует. Компилятор использует информацию о типе при проверке допустимости описанных в программе действий.

Память, в которой хранятся данные во время выполнения программы, делится на две области: стек (stack) и динамическая область, или хип (heap). Стек используется для хранения величин, память под которые выделяет компилятор, в динамической области память резервируется и освобождается во время выполнения программы с помощью специальных команд. Основным местом для хранения данных в C# является хип (в русскоязычной литературе называется «куча»).

2.2.1 Классификация типов

Любая информация легче усваивается, если она «разложена по полочкам». Поэтому, прежде чем перейти к изучению конкретных типов языка C#, рассмотрим классификацию. Типы можно классифицировать по разным признакам. Если принять за основу строение элемента, то все типы можно разделить на простые (не имеют внутренней структуры) и структурированные (состоят из элементов других типов). По своему «создателю» типы данных можно разделить на встроенные (стандартные) и определяемые программистом. Для данных статического типа память выделяется в момент объявления, при этом ее требуемый объем известен. Для данных динамического типа размер данных в момент объявления может быть неизвестен, и память под них выделяется по запросу в процессе выполнения программы.

2.2.2 Встроенные типы

Встроенные типы не требуют предварительного определения. Для каждого типа существует ключевое слово, которое используется при описании переменных, констант и т. д. Если же программист определяет собственный тип данных, он описывает его характеристики и сам дает ему имя, которое затем применяется точно так же, как имена стандартных типов. Описание собственного типа данных должно включать всю информацию, необходимую для его использования, а именно внутреннее представление и допустимые действия.

Встроенные типы C# приведены в таблице 2.8. Они однозначно соответствуют стандартным классам библиотеки .NET, определенным в пространстве имен System (в библиотеке). Как видно из таблицы, существуют несколько вариантов представления целых

и вещественных величин. Программист выбирает тип каждой величины, используемой в программе, с учетом необходимого ему диапазона и точности представления данных.

Целые типы, а также символьный, вещественные и финансовый можно объединить под названием арифметических типов. Внутреннее представление величины целого типа – целое число в двоичном коде. В знаковых типах старший бит числа интерпретируется как знаковый (0 – положительное число, 1 – отрицательное). Отрицательные числа чаще все представляются в так называемом дополнительном коде. Для преобразования числа в дополнительный код все разряды числа, за исключением знакового, инвертируются, затем к числу прибавляется единица, знаковому биту тоже присваивается единица. Беззнаковые типы позволяют представлять только положительные числа, поскольку старший разряд рассматривается как часть кода числа.

Таблица 2.8 – Встроенные типы данных в C#

Название типа данных	Ключевое слово	Тип .NET	Диапазон значений	Описание	Размер, битов
Логический	bool	Boolean	true, false		
Целые	sbyte	SByte	От-128 до 127	Со знаком	8
	byte	Byte	От 0 до 255	Без знака	8
	short	Intl6	От -32 768 до 32 767	Со знаком	16
	ushort	UIntl6	От 0 до 65 535	Без знака	16
	int	Int32	От $-2 \cdot 10^9$ до $2 \cdot 10^9$	Со знаком	32
	uint	UInt32	От 0 до 4 • 10»	Без знака	32
	long	Int64	От $-9 \cdot 10^{18}$ до $9 \cdot 10^{18}$	Со знаком	64
	ulong	UInt64	От 0 до $18 \cdot 10^{18}$	Без знака	64
Символьный	char	Char	От U+0000 до U+ffff	Unicode-символ	16
Вещественные	float	Single	От $1.5 \cdot 10^{-45}$ до $3.4 \cdot 10^{38}$	7 цифр	32
	double	Double	От $5.0 \cdot 10^{-324}$ до $1.7 \cdot 10^{308}$	15–16 цифр	64
Финансовый	decimal	Decimal	От $1.0 \cdot 10^{-28}$ до $7.9 \cdot 10^{28}$	28–29 цифр	128
Строковый	string	String	Длина ограничена объемом доступной памяти	Строка из Unicode-символов	
Object	object	Object	Можно хранить все что угодно	Всеобщий	

Вещественные типы, или типы данных с плавающей точкой, хранятся в памяти иначе, чем целочисленные. Внутреннее представление вещественного числа состоит из двух частей – мантиссы и порядка. Каждая часть имеет знак. Длина мантиссы определяет точность числа, а длина порядка – его диапазон. В первом приближении это можно представить себе так: например, для числа $0{,}381 \cdot 10^4$ цифры мантиссы 381 и порядок 4, а для числа $560{,}3 \cdot 10^2$ – мантисса 5603 и порядок 5 (мантисса нормализуется), а число 0,012 представлено как 12 и 1. Конечно, в этом примере не учтены система счисления и другие особенности.

Все вещественные типы могут представлять как положительные, так и отрицательные числа. Чаще всего в программах используется тип *double*, поскольку его диапазон и точность покрывают большинство потребностей. Этот тип имеет вещественные литералы и многие стандартные математические функции.

Тип *decimal* предназначен для денежных вычислений, в которых критичны ошибки округления. Как видно из таблицы 2.8, тип *float* позволяет хранить одновременно всего 7 значащих десятичных цифр, тип *double* – 15–16. При вычислениях ошибки округления накапливаются, и при определенном сочетании значений это даже может привести к ре-

зультату, в котором не будет ни одной верной значащей цифры! Величины типа *decimal* позволяют хранить 28–29 десятичных разрядов. Тип *decimal* не относится к вещественным типам, у него различное внутреннее представление. Величины денежного типа даже нельзя использовать в одном выражении с вещественными типами без явного преобразования типа. Использование величин финансового типа в одном выражении с целыми допускается.

Любой встроенный тип C# соответствует стандартному классу библиотеки .NET, определенному в пространстве имен System. Везде, где используется имя встроенного типа, его можно заменить именем класса библиотеки. Это значит, что у встроенных типов данных C# есть методы и поля.

2.2.3 Типы констант

Как уже говорилось, величин, не имеющих типа, не существует. Поэтому константы тоже имеют тип. Если значение целого литерала находится внутри диапазона допустимых значений типа *int*, константа рассматривается как *int*, иначе она относится к наименьшему из типов *uint*, *long* или *ulong*, в диапазон значений которого она входит. Вещественные константы по умолчанию относятся типу double.

Например, константа 10 относится к типу *int* (хотя для ее хранения достаточно байта), а константа 2147483648 будет определена как *uint*. Для явного задания типа константы служит суффикс, например 1.1f, 1UL, 1000m (все суффиксы перечислены в таблице 2.6). Явное задание применяется в основном для уменьшения количества неявных преобразований типа, выполняемых компилятором.

2.2.4 Типы-значения и ссылочные типы

Чаще всего типы C# разделяют по способу хранения элементов на типы-значения и ссылочные типы (рисунок 2.1). Встроенные типы на рисунке выделены полужирным шрифтом, простые типы подчеркнуты. Элементы типов-значений, или значимых типов (value types), представляют собой просто последовательность битов в памяти, необходимый объем которой выделяет компилятор. Иными словами, величины значимых типов хранят свои значения непосредственно. Величина ссылочного типа хранит не сами данные, а ссылку на них (адрес, по которому расположены данные). Сами данные хранятся в хипе.

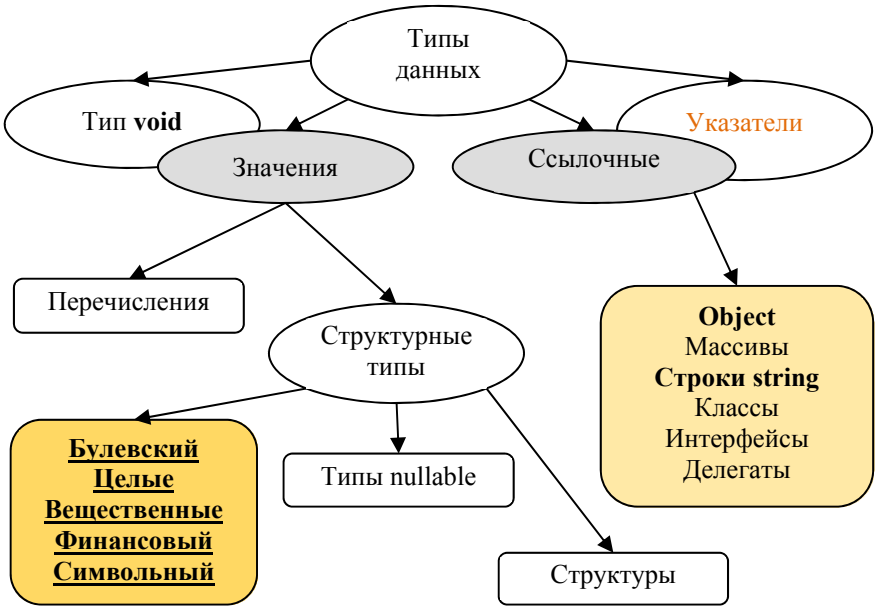


Рисунок 2.1 – Классификация типов данных C# по способу хранения

Рисунок 2.2 иллюстрирует разницу между величинами значимого и ссылочного типов. Одни и те же действия над ними выполняются по-разному. Рассмотрим в качестве примера проверку на равенство. Величины значимого типа равны, если равны их значения. Величины ссылочного типа равны, если они ссылаются на одни и те же данные (на рисунке *в* и *с* равны, но *а* не равно *в* даже при одинаковых значениях). Из этого следует, что если изменить значение одной величины ссылочного типа, это может отразиться на другой.

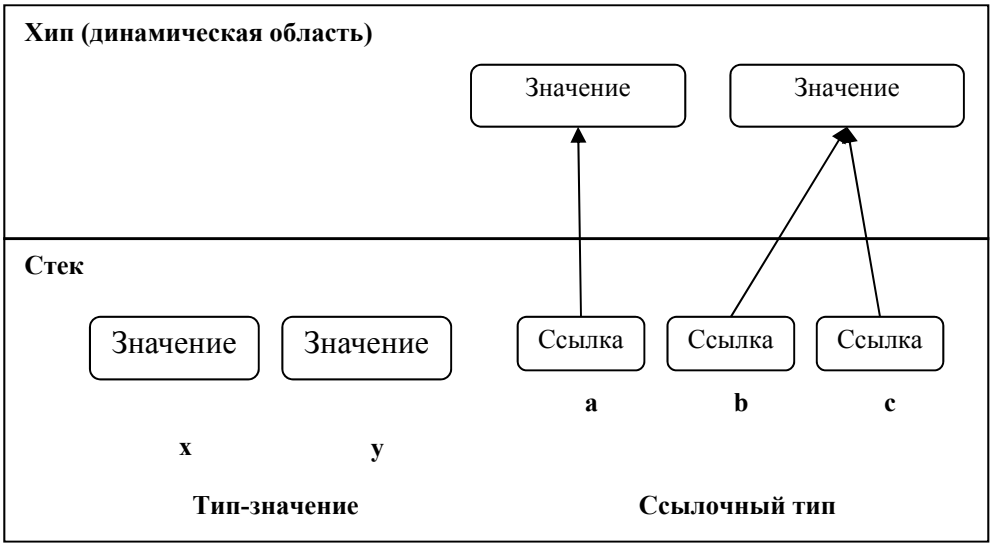


Рисунок 2.2 – Хранение в памяти величин значимого и ссылочного типов

Обратите внимание на то, что не все значимые типы являются простыми. По другой классификации структуры и перечисления относятся к структурированным типам, определяемым программистом.

2.2.5 Преобразование типов

Преобразования типов существуют практически во всех языках программирования. В некоторых оно происходит автоматически, в других – вручную, а в третьих – частично автоматически, частично вручную. Преобразованием типов в C# – это операция назначения переменной одного типа переменной другого типа.

В C# есть два вида преобразований:

– неявное преобразование (автоматическое) – преобразованием меньше по размеру типа данных в больший: ***char -> int -> long -> float -> double;***

– явное преобразование (ручное) – обратное предыдущему преобразование большего типа в меньший: ***double -> float -> long -> int -> char.***

Неявное преобразование осуществляется автоматически при назначении переменной с меньшим типом данных большему (рисунок 2.3). Рисунок 2.3 представляет собой часть программного кода на языке C#, который более детально будет рассмотрен в следующих разделах. Общая схема неявных арифметических преобразований представлена на рисунке 2.4.

```
int num = 5;
double dnum = num;       // Автоматическое преобразование: int в double
Console.WriteLine(num);    // Значение num  будет равно 5
Console.WriteLine(dnum);   // Значение dnum  будет равно 5
```

Рисунок 2.3 – Часть программного кода на языке C# с неявным преобразованием типов и выводом результатов на консоль

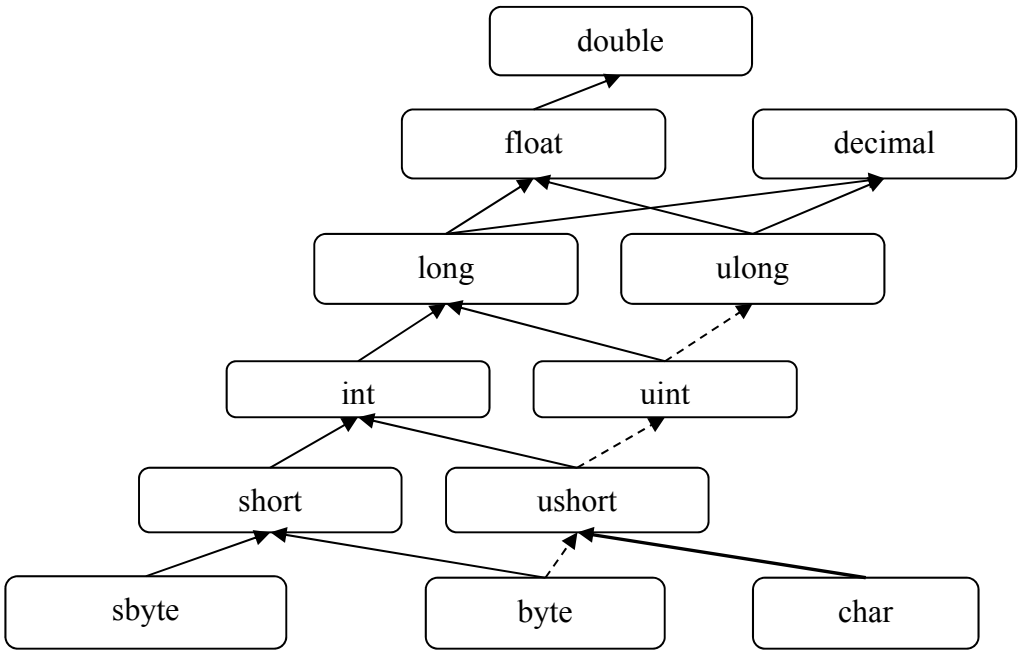


Рисунок 2.4 – Неявные арифметические преобразования типов в C#

Как тип преобразования должен быть выполнен вручную. Для этого перед преобразуемой переменной в круглых скобках указываем желаемый тип, в который должна быть преобразована переменная исходного типа (рисунок 2.5):

```
double dnum = 5.23;
int inum = (int) dnum; // Ручное преобразование: double в int
Console.WriteLine(dnum);   // Значение dnum равно 5.23
Console.WriteLine(inum);   // Значение inum равно 5
```

Рисунок 2.5 – Часть программного кода на языке C# с явным преобразованием типов и выводом результатов на консоль

В последнем примере (рисунок 2.5) показан один из типов преобразования, но кроме него в C# также существуют и другие способы явного преобразования типов, например с помощью методов встроенного класса **Convert** (рисунок 2.6). Понятие класса в C# будет рассмотрено в следующих разделах.

```
// Преобразования
Convert.ToInt32 (int)    // в тип int
Convert.ToInt64 (long) // в тип long
Convert.ToDouble       // в тип double
Convert.ToBoolean      // в тип boolean
Convert.ToString       // в строку
```

Рисунок 2.6 – Методы класса Convert преобразования типов

На рисунке 2.7 приведены примеры использования класса Convert

```
int inum = 10;
double dnum = 5.25;
```

```
bool boolValue = true;
Console.WriteLine(Convert.ToString(inum));       // int в string
Console.WriteLine(Convert.ToDouble(inum));       // int в double
Console.WriteLine(Convert.ToInt32(dnum));        // double в int
Console.WriteLine(Convert.ToString(boolValue));  // bool в string
```

Рисунок 2.7 – Примеры использования методов класса Convert

Таким образом, преобразования типов помогают получить нужный в конкретной ситуации тип данных. Некоторые из них происходят неявно, некоторые необходимо выполнить явно, но в любом случае хорошее понимание данной особенности необходимо, так как позволит быстрее решать проблемы типов в C# в дальнейшем.

2.3 Переменные, операнды, выражения и операторы

Для хранения данных в программе применяются переменные. Переменная представляет именованную область памяти, в которой хранится значение определенного типа. Переменная имеет тип, имя и значение. Имя переменной должно отражать смысл хранимой величины. Тип определяет, какого рода информацию может хранить переменная. Перед использованием любую переменную надо объявить (в языке C# это обязательно). Использование переменных означает определение значений этих переменных (то есть занесение в эти переменные конкретных данных), извлечение данных из этих переменных и для других операций. Синтаксис объявления переменной выглядит следующим образом:

Тип имя_переменной.

Определять переменные можно различными способами: с помощью операции присваивания, с помощью операций ввода и другими возможностями языка программирования. В языке C# переменные можно объявлять и определять одновременно. Синтаксис объявления и определения переменной одновременно выглядит следующим образом:

Тип имя_переменной = Значение переменной.

Примеры объявления и инициализации (определения) переменных представлены на рисунке 2.8. Инициализация – это начальное определение переменных. Из примеров видно, что вначале объявляются переменные с необходимыми типами (строки 1–5), а затем с помощью операторов присваивания определяются (строки 7–13). А примеры объявления и описания переменных одновременно представлены на рисунке 2.9.

```
1.  double d, f; // Объявление 2-х переменных типа double
2.  int i, j; // Объявление 2-х переменных типа int
3.  bool answer; // Объявление переменной типа bool
4.  char c; // Объявление переменной типа char
5.  string s; // Объявление переменной типа string
6.  // Инициализация (определение) переменных описанных выше
7.  d = 2.939;
8.  f = -29309.39;
9.  i = 50;
10. j = -300;
11. c = '=';
12. s = "This is a text";
13. answer = false;
```

Рисунок 2.8 – Примеры объявления и определения переменных

```
1.  double d = 2.939,f = -29309.39; // Объявление и определение 2-х переменных типа
double
2.  int i = 50, j = -300; // Объявление и определение 2-х переменных типа int
3.  bool answer = false; // Объявление и определение переменной типа bool
4.  char c = '='; // Объявление переменной типа char
5.  string s= "This is a text"; // Объявление и определение переменной типа string
```

Рисунок 2.9 – Примеры объявления и определения переменных одновременно

Каждая переменная доступна в рамках определенного контекста или области видимости (действия). Вне этого контекста переменная уже не существует.

Существуют различные контексты:

– контекст класса. Переменные, определенные на уровне класса, доступны в любом методе этого класса. Их еще называют глобальными переменными или полями;

– контекст метода. Переменные, определенные на уровне метода, являются локальными и доступны только в рамках данного метода. В других методах они недоступны;

– контекст блока кода. Переменные, определенные на уровне блока кода, также являются локальными и доступны только в рамках данного блока. Вне своего блока кода они не доступны.

Выражение – правило вычисления значения. В выражении участвуют **операнды**, объединенные знаками операций. Операндами выражения могут быть константы, переменные и вызовы функций. Например, *b* + 2 – это выражение, в котором + является знаком операции, а *b* и 2 операндами. Операции выполняются в соответствии с приоритетами (таблица 2.4). Для изменения порядка выполнения операций используются круглые скобки. Результатом выражения всегда является значение определенного типа, который определяется типами операндов. Величины, участвующие в выражении, должны быть совместимых типов.

Оператор – это элемент языка, задающий полное описание действия, которое необходимо выполнить. Каждый оператор представляет собой законченную фразу языка программирования и определяет некоторый вполне законченный этап обработки данных. В состав операторов могут входить служебные слова, данные, выражения и другие операторы. В английском языке данное понятие обозначается словом «statement», означающим также «предложение или инструкция».

Каждый оператор в любом языке программирования имеет определенный **синтаксис** и **семантику**. Под *синтаксисом* оператора понимается система правил (грамматика), определяющая его запись с помощью элементов алфавита данного языка, в который наряду с различными символами входят, например, и служебные слова. Под *семантикой* оператора понимают его смысл, т. е. те действия, которым соответствует запись того или иного оператора. Например, запись *i = i + 1* является примером синтаксически корректной записи *оператора присваивания* в языке C#, семантика которого в данном случае такова: извлечь значение ячейки памяти, соответствующей переменной *i*, сложить его с единицей, результат записать в ту же ячейку памяти.

В большинстве процедурных языков программирования набор операторов практически одинаков и состоит из оператора присваивания, операторов выбора, операторов цикла, оператора вызова процедуры, операторов перехода. Иногда выделяют также пустой (отсутствие действия) и составной операторы. Многие операторы являются способом представления определенных алгоритмических конструкций. Программа на языке программирования представляет собой совокупность логически взаимосвязанных операторов, приводящую к решению конкретной задачи. Для написания программы на любом языке программирования необходимо знать структуру программы этого языка. Структуру программы на языке программирования C# рассмотрим в следующем разделе 2.5.

2.4 Массивы

До настоящего момента в примерах программного кода использовались простые переменные. При этом каждой области памяти, выделенной для хранения одной переменной, соответствует свое имя. Если переменных много, программа, предназначенная для их обработки, получается длинной и однообразной. Поэтому в любом процедурном языке есть понятие массива – ограниченной совокупности однотипных величин. Элементы массива имеют одно и то же имя, а различаются порядковым номером (индексом). Это позволяет компактно записывать множество операций с помощью циклов. Массив относится к ссылочным типам данных, то есть располагается в динамической области памяти, поэтому создание массива начинается с выделения памяти под его элементы. Элементами массива могут быть величины как значимых, так и ссылочных типов (в том числе массивы). Массив значимых типов хранит значения, массив ссылочных типов – ссылки на элементы. Всем элементам при создании массива присваиваются значения по умолчанию: нули для значимых типов и **null** для ссылочных.

Вот, например, как выглядят операторы создания массива из 10 целых чисел и массива из 100 строк:

```
int[] w = new int[10];
string[] z = new string[100];
```

В первом операторе описан массив **w** типа int[]. Операция **new** выделяет память под 10 целых элементов (чисел), и они заполняются нулями.

Во втором операторе описан массив **z** типа string[]. Операция **new** выделяет память под 100 ссылок на строки, и эти ссылки заполняются значением **null**. Память под сами строки, составляющие массив, не выделяется, это будет необходимо сделать перед заполнением массива. Количество элементов в массиве (размерность) не является частью его типа, это количество задается при выделении памяти и не может быть изменено впоследствии. Размерность может задаваться не только константой, но и выражением. Результат вычисления этого выражения должен быть неотрицательным, а его тип должен иметь неявное преобразование к int, uint, long или ulong.

Пример размерности массива, заданной выражением:

```
short n = . . .;
string[] z = new string[n + 1];
```

Элементы массива нумеруются с нуля, поэтому максимальный номер элемента всегда на единицу меньше размерности (например, в описанном выше массиве **w** элементы имеют индексы от 0 до 9). Для обращения к элементу массива после имени массива указывается номер элемента в квадратных скобках, например: z[4], z [i].

С элементом массива можно делать все, что допустимо для переменных того же типа. При работе с массивом автоматически выполняется контроль выхода за его границы: если значение индекса выходит за границы массива, генерируется исключение IndexOutOfRangeException.

Массивы одного типа можно присваивать друг другу. При этом происходит присваивание ссылок, а не элементов, как и для любого другого объекта ссылочного типа, например:

```
int[] a = new int[10];
int[] b = a; // b и a указывают на один и тот же массив
```

Все массивы в C# имеют общий базовый класс **Array**, определенный в пространстве имен **System**. В нем есть несколько полезных методов, упрощающих работу с массивами, например методы получения размерности, сортировки и поиска. В этой части методического пособия эти методы рассматриваться не будут.

Массивы, являющиеся полями класса, могут иметь те же спецификаторы, что и поля, представляющие собой простые переменные.

В C# существуют три разновидности массивов: одномерные, прямоугольные (двумерные), ступенчатые (невыровненные) и многомерные.

Одномерные массивы используются в программах чаще всего. Варианты описания одномерного массива:

```
тип[] имя;
тип[] имя = new тип [размерность];
тип[] имя = {список инициализаторов};
тип[] имя = new тип [] {список инициализаторов};
тип[] имя = new тип [ размерность ] {список инициализаторов};
```

Примечание. При описании массивов квадратные скобки являются элементом синтаксиса, а не указанием на необязательность конструкции.

Примеры некоторых способов описания и определения одномерных массивов представлены ниже, но существуют и другие способы их описания и определения (ввод массива с экрана монитора, ввод массива из файла, ввод массива из базы данных):

```
1. int [] a;                          // Элементов нет
2. int [] b = new int [4];            // Элементы равны 0
3. int [] c = { 61,  2,  5, -9 };     // new подразумевается
4. int [] d = new int [] { 61. 2, 5. -9 };   // Размерность вычисляется
5. int [] e = new int [4] { 61, 2, 5, -9 };  // Избыточное описание
6. string [] z = new string [100];    // Элементы равны null
```

В примерах описано шесть массивов. Отличие первого оператора от остальных состоит в том, что в нем, фактически, описана только ссылка на массив, а память под элементы массива не выделена. Если список инициализации не задан, размерность может быть не только константой, но и выражением типа, приводимого к целому. В каждом из остальных массивов по четыре элемента целого типа. Как видно из операторов 3–6, массив при описании можно инициализировать. Если при этом не задана размерность (оператор 4), количество элементов вычисляется по количеству инициализирующих значений. Для полей объектов и локальных переменных можно опускать операцию ***new***, она будет выполнена по умолчанию (оператор 3). Если присутствует и размерность, и список инициализаторов, размерность должна быть константой (оператор 5). Операция ***new*** выделяет память под 100 ссылок на строки, и эти ссылки заполняются значением ***null*** (оператор 6).

Прямоугольный массив имеет два измерения. Чаще всего в программах используются двумерные массивы. Варианты описания двумерного массива:

```
тип[ , ] имя;
```

```
тип[ , ] имя = new тип [разм_1, разм_2];
тип[ , ] имя = {список инициализаторов};
тип[ , ] имя = new тип [,] {список инициализаторов};
тип[ , ] имя = new тип [разм_1, разм_2] {список инициализаторов};
```

Примеры описаний (один пример для каждого варианта описания):

```
int[ , ] a ; // 1 Элементов нет
int[ , ] b = new int[2, 3] ; // 2 Элементы равны 0
int[ , ] c = { { 1 , 2, 3}, {4, 5, 6 } } ; // 3 new подразумевается
int[ , ] c = new int[ , ] { { 1, 2, 3}, {4, 5, 6 } } ; // 4 Размерность вычисляется
int[ , ] d = new int[2, 3] { { 1, 2, 3}, {4, 5, 6} } ; // 5 Избыточное описание
```

Если список инициализации не задан, размерности могут быть не только константами, но и выражениями типа, приводимого к целому. К элементу двумерного массива обращаются, указывая номера строки и столбца, на пересечении которых он расположен, например: a[1, 3], b[i , j], b[j , i].

Примечание. Необходимо помнить, что компилятор воспринимает как номер строки первый индекс, как бы он ни был обозначен в программе.

В ступенчатых массивах количество элементов в разных строках может различаться. В памяти ступенчатый массив хранится иначе, чем прямоугольный: в виде нескольких внутренних массивов, каждый из которых имеет свой размер. Кроме того, выделяется отдельная область памяти для хранения ссылок на каждый из внутренних массивов.

Описание ступенчатого массива имеет вид:

```
тип[][] имя;
```

Под каждый из массивов, составляющих ступенчатый массив, память требуется выделять явным образом, например:

```
int[][] a = new int[3] [ ];        // Выделение памяти под ссылки на три строки
    a[0] = new int[5];             // Выделение памяти под 0-ю строку (5 элементов)
    a[1] = new  int[3];            // Выделение памяти под 1-ю строку (3 элемента)
    a[2] = new int[4];             // Выделение памяти под 2-ю строку (4 элемента)
```

Здесь a[0], a[1] и a[2] – это отдельные массивы, к которым можно обращаться по имени. Другой способ выделения памяти: int[ ] [ ] a = {new int[5], new int [3] , new int[4]}.

К элементу ступенчатого массива обращаются, указывая каждую размерность в своих квадратных скобках, например: = a[1] [2], a[i] [j], a[j] [i].

В остальном использование ступенчатых массивов не отличается от использования прямоугольных. Невыровненные массивы удобно применять, например, для работы с треугольными матрицами большого объема.

Многомерные массивы имеют три измерения и более. Описание трехмерного массива имеет вид:

```
int[, ,] array1 = new int[2, 2, 3];
```

Примеры описания и определения трехмерных массивов:

```
int[,,] array2 = new int[2, 2, 3] { { { 1, 2, 3 }, { 4, 5, 6 } },
```

```
                                    { { 7, 8, 9 }, { 10, 11, 12 } } };
int[,,] array3 = new int[,,]      { { { 1, 2, 3 }, { 4, 5, 6 } },
                                    { { 7, 8, 9 }, { 10, 11, 12 } } };
```

2.5 Структура программы

Структура программы на языке C# может быть представлена с различной степенью детализации, наглядности и восприятия. Приведем различные способы представления структур.

Структуру программы на C# можно представить состоящей из следующих блоков:

- объявление пространства имен (своего рода контейнера);
- объявление класса (основная сущность программы);
- методы класса (подпрограммы), как минимум метод **Main**;
- операторы и выражения;
- комментарии.

Приведем пример простейшей программы, написанной на C#. Это будет консольное приложение, выводящее строку «Привет Мир» (своего рода классика для первой программы в практике программиста). Код такой программы приведен на рисунке 2.10.

```
1.  // Подключение пространства имен System
2.  using System;
3.  //  пространства имен
4.  namespace ProgramStructure
5.  {
6.      // Объявление класса
7.      class Program
8.      {
9.          // Главный метод программы
10.         static void Main(string[] args)
11.         {
12.         // Вывод строки
13.          Console.WriteLine("Привет Мир!");
14.         // Вспомогательный оператор
15.          Console.ReadKey();
16.         }
17.     }
18. }
```

Рисунок 2.10 – Код простейшей программы на языке C#

Опишем этот код более подробно. Первая строка данной программы – это комментарий. Комментарии никак не влияют на работу программы, они нужны для специалиста, который будет сопровождать код программы (дорабатывать её, исправлять ошибки и т. п.).

Вторая строка программы (**using System;**) является оператором, который подключает стандартное пространство имен **System**. Это дает возможность получить доступ к набору классов, имеющихся в «контейнере» **System**. Как видно, данная строка состоит из двух слов, первое (ключевое слово **using**) означает, что необходимо подключить пространство имен, а второе **System** – название нужного пространства имен. В конце второй строки

стоит символ «;», который обозначает завершение оператора. Каждый оператор программы должен заканчиваться таким символом.

Третья строка программы является комментарием так же, как и строки 6, 9, 12, 14. Комментарии в C# начинаются с символов «//» (две косые черты, два слэша) и действуют только до конца строки.

В C# есть и многострочные комментарии, иногда удобнее использовать их, они рассматривались более подробно в подразделе 2.1.5.

В четвертой строке (**namespace** ProgramStructure) объявляется своё пространство имен, оно называется «ProgramStructure». Пространство имен является своего рода контейнером, и оно ограничивается фигурными скобками (открывающей – строка 5 и закрывающей – строка 18), следующими за его названием. Таким образом, все, что находится между строками 5 и 18, принадлежит пространству имен ProgramStructure.

В строке 7 объявляется класс с именем «Program» – это основной и единственный класс этой программы. Как можно заметить, для объявления класса служит ключевое слово **class**, за которым следует имя класса. В программе может быть и не один, а несколько классов. Как правило, класс состоит из набора методов, которые определяют так называемое поведение класса (если хотите, функциональность). Границы класса, так же как и пространства имен, обозначаются фигурными скобками (строки 8 и 17). В нашем случае класс имеет только один метод – это метод **Main**.

В строке 10 как раз и объявляется метод **Main**. Этот метод является главным в этой программе, это так называемая точка входа в программу. И это означает, что при запуске программы первым будет выполняться именно метод **Main**. Каждый метод тоже имеет границы, которые также обозначаются фигурными скобками (строки 11 и 16).

Метод **Main** программы содержит только два оператора. Эти операторы значатся в строках 13 и 15. Первый выводит сообщение «Привет Мир!». А второй является вспомогательным, он заставляет программу ждать нажатие клавиши на клавиатуре и не дает ей до этого момента завершить свое выполнение (без этого оператора программа бы вывела строку и быстро закрылась, так что программист или пользователь даже бы не успели прочитать, что она вывела).

Второй способ представления структуры программы на C# можно описать следующим образом:

1. Программа состоит из описаний пользовательских типов (в основном классов).
2. Описания классов состоят из описания полей (переменных) и методов.
3. Описание переменных состоит из указания типа и имени переменной.
4. Описание методов состоит из описания локальных переменных и набора операторов.
5. Оператор состоит из набора ключевых слов и выражений.
6. Выражения состоят из переменных и констант, связанных знаками операций.
7. В программу могут вставляться комментарии.
8. В программу может входить пространство имен программы, каждое из которых представляет собой совокупность классов.
9. В программу могут входить библиотеки (уже разработанные пространства имен, которые будут использоваться в программе).

Прежде всего, сделаем несколько общих пояснений синтаксиса языка C#. В данном языке, как и в других языках, основанных на языке C, большинство операторов заканчивается символом "точка с запятой" (;). Несколько операторов может записываться в одной строке, или один оператор может записываться в нескольких строках. Операторы могут объединяться в блоки с помощью фигурных скобок ({}). На рисунке 2.11 покажем пример кода на C#, использующий 2-й способ представления структуры программы.

```
// Подключаемые библиотеки
using System;
using System.IO;
using System.Text;
// Пространство имен программы
namespace ConsoleApplication
{
    // Основной класс
    class Program
    {
        // Точка входа
        static void Main(string[] args)
        {
           // Операторы (инструкции)
           int a = 1; // Оператор присваивания
           int b = 3; // Оператор присваивания
           Console.WriteLine(a + b); // Вызов метода вывода WriteLine
           Console.ReadKey(); // Вызов метод ввода одиночного символа ReadKey
        }
    }
}
```

Рисунок 2.11 – Код программы на языке C#, осуществляющий сложение двух чисел (1 и 3) и вывод результата на консоль

Опишем код этой программы:

– **подключаемые библиотеки** (здесь указываются пространства имен из библиотек, которые используются в программе);

– **пространство имен программы** (используется для упорядочения классов в проекте, является не обязательным);

– **основной класс** – синтаксис языка C# (требует создания хотя бы одного класса для корректной компиляции программы);

– **точка входа** – метод Main является точкой входа в программу, с него начинается выполнение программ, написанных для платформы .NET (платформа .NET будет описана в следующих разделах);

– **инструкция** (statement) – базовый блок программы, представляет собой некоторое действие: объявление переменной, присвоение значения, вызов метода, арифметическую операцию. Инструкция в C# должна заканчиваться точкой с запятой (;). Несколько инструкций можно объединить в блок кода, выделив его фигурными скобками { }.

Основной класс программы не обязательно должен называться **Program**, однако один из классов программы должен содержать метод *static void Main* для того, чтобы среда CLR смогла идентифицировать, откуда начинать выполнения программы.

Третий способ представления структуры программы на C# можно изобразить графически (рисунок 2.12):

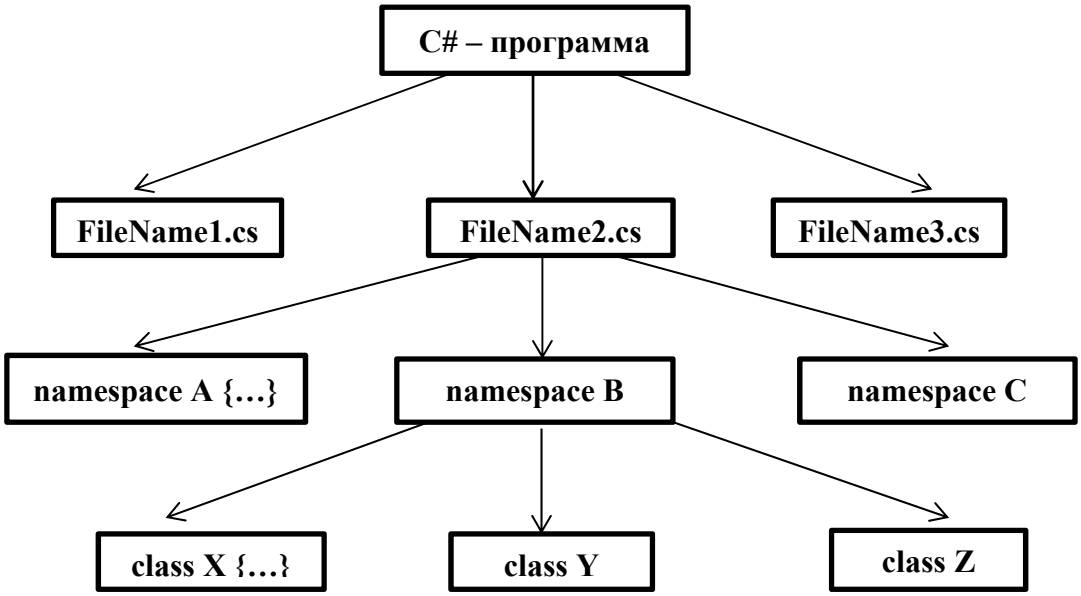


Рисунок 2.12 – Структура программы на языке C#

Структура программы на рисунок 2.12 носит глобальный характер, то есть показывает организацию самой сложной программной системы. Заметим, что программа на C# может состоять как из одного, так и из нескольких файлов, содержащих исходный текст на языке программирования C#. Каждый такой файл имеет расширение .cs (в нашем примере файлы названы FileName1.cs, FileName2.cs и FileName1.cs). Любой файл с исходным текстом на языке программирования C# может как содержать пространства имен, так и не содержать их (в нашем примере файл FileName2.cs содержит три пространства имен (A, B и C), а FileName1.cs и FileName3.cs не содержат пространств имен). Наконец, каждое пространство имен может как содержать описание (одного или нескольких) классов, так и не содержать их (в нашем примере пространство имен B содержит три описания трех классов (X, Y и Z), а пространство имен A и C не содержат ни одного описания классов). Итак, программа на языке C# размещается в одном или нескольких текстовых файлах, стандартное расширение которых – .cs. В программе объявляются пользовательские типы, которые состоят из элементов. Примерами пользовательских типов являются классы и структуры, а примером элемента типа может служить метод класса. Типы могут быть логически сгруппированы в пространства имен. При компиляции программы получается сборка, представляющая собой файл с расширением .exe или .dll.

Исходный текст программы на языке C# содержит операторы и комментарии. Основными видами операторов в C# являются следующие:

– оператор-выражение. Под выражением может пониматься вызов метода, присваивание, а также допустимые комбинации операндов и операций. Оператор-выражение завершается символом “;” (точка с запятой);

– операторы управления ходом выполнения программы, такие как оператор условного перехода, оператор безусловного перехода или операторы циклов;

– блок операторов. Блок – это набор операторов, обрамленных фигурными скобками – ‘{‘ и ‘}’. Блоки необходимо использовать там, где синтаксис языка требует выполнение более одного оператора;

– операторы объявлений пользовательских типов, элементов типов и локальных переменных и констант.

Программа может содержать комментарии, игнорируемые при компиляции. Различают следующие виды комментариев:

1. Строчный комментарий – это комментарий, начинающийся с последовательности символов // и продолжающийся до конца строки.

2. Блочный комментарий – все символы, заключенные между /* и */.

3. Комментарии для документации напоминают строчные комментарии, но начинаются с последовательности символов /// и могут содержать специальные XML-тэги.

В языке C# различаются строчные и прописные символы при записи идентификаторов и ключевых слов.

Количество пробелов в начале строки, в конце строки и между элементами строки значения не имеет. Это позволяет улучшить структуру исходного текста программы.

Язык C# требует, чтобы вся программная логика была заключена в определения типов (под типом подразумеваются классы, интерфейсы, структуры и аналогичные компоненты языка). В отличие от языков C и C++ глобальные функции и глобальные переменные в чистом виде в языке C# использовать нельзя.

2.6 Основные операторы языка

Понятие оператора в языке программирования подробно описано в разделе 2.3. Основные операторы языка программирования C# будем рассматривать следующим образом:

1. Структура оператора (вместо слова структура в литературе может употребляться слово синтаксис, формат, общий вид).
2. Описание каждого элемента структуры.
3. Правила выполнения (работы) оператора.
4. Примеры использования оператора.

2.6.1 Оператор присваивания

Оператор присваивания существует практически в любом языке программирования и является ключевым, в том числе и в C#, и он уже упоминался в предыдущих разделах при объяснении различных способов описания алгоритмов. Структура оператора присваивания имеет вид:

[Тип] имя_переменной = Значение переменной;

Оператор присваивания состоит из имени переменной, ее типа, знака равенства и значения переменной. Тип переменной необязателен, это указывают квадратные скобки. При рассмотрении других операторов те его элементы, которые будут указаны в квадратных скобках, будут считаться необязательными. Оператор присваивания выполняется следующим образом: значение переменной записывается в ячейку памяти с именем переменной, под которую отведен объем памяти в соответствии с ее типом. Если в операторе присваивания отсутствует тип переменной, то он должен быть указан в операторе описания этой переменной заранее (рисунок 2.8). Самое главное – все переменные правой части оператора присваивания должны быть определены. Примеры: *int x = 5; int y = x + 9; int z = x + y;*

2.6.2 Оператор условного перехода: IF

Оператор условного перехода (условный оператор) ***if*** (в переводе с англ. – «если») реализует выполнение определённых команд (операторов) при условии, что некоторое логическое выражение (условие) принимает значение true («истина») или false («ложь»). Можно сказать, что оператор ***if*** обеспечивает передачу управления на одну из двух ветвей вычислений. Структура оператора ***if*** имеет вид:

if (логическое выражение) оператор 1; [else оператор 2;]

If – это имя оператора. Логическое выражение – это простая или сложная условная конструкция, которая используется для управления ходом выполнения программы. Логические выражения могут включать в себя операции сравнения и логические операции. К операциям сравнения относятся: > (больше); >= (больше равно); < (меньше); <= (меньше равно); = = (равно); != (не равно). Логические операции это: && (логическое «И», || (логическое «ИЛИ», ! (логическое «НЕ» и другие. ***Оператор 1*** и ***оператор 2*** – это один или несколько операторов языка C#, а ***else*** (в переводе с англ. – «иначе») – это вторая часть оператора ***if.*** Примеры простых логических выражений: ***a > b; c <= d; b != c.*** Примеры сложных логических выражений: ***a > b && c < d; a <= d || c >= b; !(a >b) || c !=d && d == a.*** Оператор ***if*** применяется в двух формах: в полной (это со второй частью ***else***) и в неполной (без ***else***). Поэтому рассмотрим правила выполнения оператора ***if*** в полной и неполной формах.

*Выполнение оператора **if** в полной форме:* вначале проверяется логическое выражение, если оно имеет значение «истина», то выполнятся ***оператор 1*** и управление передается на выполнение оператора, следующего за оператором ***if***; а если логическое выражение имеет значение «ложь», то выполняется ***оператор 2*** и управление передается на выполнение оператора, следующего за оператором ***if***. Оператор ***if*** в полной форме необходимо использовать для описания фрагмента 1 граф-схемы (блок-схемы) алгоритма (ГСА), представленного на рисунке 2.13. Примеры:

```
if (b > a) max = b; else max = a;
if (a < b && ( a > d || a == 0 ) ) b++; else { b *= a: a = 9; }
   else if (a < b ) if ( a < c ) m = a; else m = c;
if ( b < c ) m = c; else m = b;
```

*Выполнение оператора **if** в неполной форме* происходит следующим образом: вначале проверяется логическое выражение, если оно имеет значение «истина», то выполнятся ***оператор 1*** и управление передается на выполнение оператора, следующего за оператором ***if***; а если логическое выражение имеет значение «ложь», то управление передается на выполнение оператора, следующего за оператором ***if***. Оператор ***if*** в неполной форме необходимо использовать для описания фрагмента 2 граф-схемы алгоритма, представленного на рисунке 2.14. Примеры:

```
if (a < 0 ) b = 1;
if (a <= 10 ) {b = 9; c =15;}
```

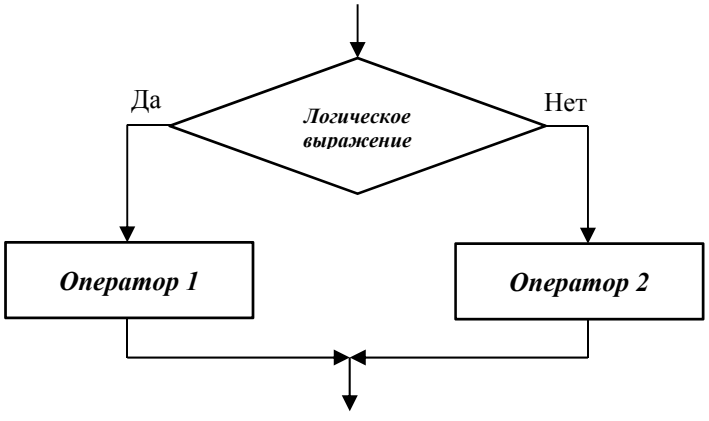


Рисунок 2.13 – Фрагмент 1 граф-схемы алгоритма

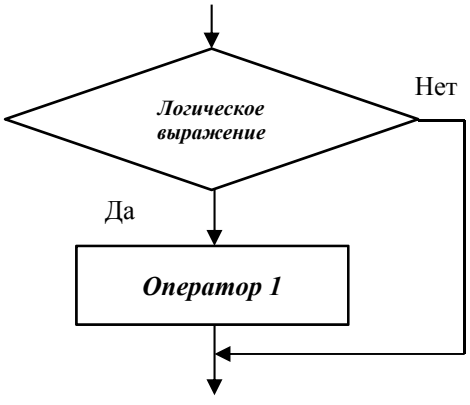


Рисунок 2.14 – Фрагмент 2 граф-схемы алгоритма

2.6.3 Оператор выбора: SWITCH

Оператор ***switch*** (переключатель) предназначен для разветвления процесса вычислений на несколько направлений. Можно сказать, что оператор ***switch*** обеспечивает передачу управления на одну из произвольного числа ветвей или заменяет несколько следующих подряд операторов ***if.*** Структура оператора ***switch*** имеет вид:

switch (выражение)
{
case константное_выражение_1: [список_операторов_1;]
case константное_выражение_2: [список_операторов_2;]
…
case константное_выражение_n: [список_операторов_n;]
[default: операторы;]
}

Switch – это имя оператора. ***Выражение*** – заданное выражение целочисленного (**char, byte, short, int**) типа, перечисления или строчного (**string**) типа; ***константное_выражение_1***, ***константное_выражение_2***, …, ***константое_выражение_n*** – константы выбора, тип которых может быть совместим с типом выражения. Среди констант выбора не должно быть двух с одинаковыми значениями.

Список_операторов_1, ***список_операторов_2***, …, ***список_операторов_n*** – это последовательность операторов, которые выполняются в случае, если значение константы выбора совпадет со значением выражения.

Если ни одна из констант выбора не совпадает с заданным выражением, то выполняются ***операторы***, которые следуют за ключевым словом «***default»***. В случае если блок ***default*** отсутствует в операторе ***switch*** и ни одна из констант не совпадает со значением выражения, то никаких действий не выполняется.

В конце каждого блока ***case*** должен ставиться один из операторов перехода: ***break, goto case, return или throw***. Как правило, используется оператор ***break***. При его применении другие блоки ***case*** выполняться не будут. Операторы ***break, goto case, return*** или ***throw*** будут рассмотрены ниже. Оператор ***switch*** можно использовать для описания фрагмента 3 граф-схемы алгоритма, представленного на рисунке 2.15. Примеры:

```
// Пример 1
string name = "Иван";
 switch (name)
{
    case "Пётр":
        Concole.WriteLine("Ваше имя - Пётр");
        break;
    case "Николай":
        Console.WriteLine("Ваше имя - Николай");
        break;
    case "Иван":
        Console.WriteLine("Ваше имя - Иван");
        break;
}
// Пример 2
string name = "Владимир";
 switch (name)
{
    case "Пётр":
        Console.WriteLine("Ваше имя - Пётр");
```

```
        break;
    case "Николай":
        Console.WriteLine("Ваше имя - Николай");
        break;
    case "Иван":
        Console.WriteLine("Ваше имя - Иван");
        break;
    default:
        Console.WriteLine("Неизвестное имя");
        break;
}
// Пример 3
int number = 1;
switch (number)
{
         case 1:
             Console.WriteLine("case 1");
             goto case 5; // переход к case 5
         case 3:
             Console.WriteLine("case 3");
             break;
         case 5:
             Console.WriteLine("case 5");
             break;
         default:
             Console.WriteLine("default");
             break;
}
```

В примере 1 конструкция ***switch*** последовательно сравнивает значение переменной ***name*** с набором значений, которые указаны после операторов. Поскольку здесь значение переменной ***name –*** строка "Иван", то будет выполняться блок:

```
case "Иван":
        Console.WriteLine("Ваше имя - Иван");
        break;
```

В примере 2 никакое из значений после операторов ***case*** не совпадает со значением переменной ***name***, поэтому будет выполняться блок ***default***. Однако если необходимо, чтобы, наоборот, после выполнения текущего блока ***case*** выполнялся другой блок ***case***, то можно использовать вместо ***break*** оператор ***goto case*** (пример 3).

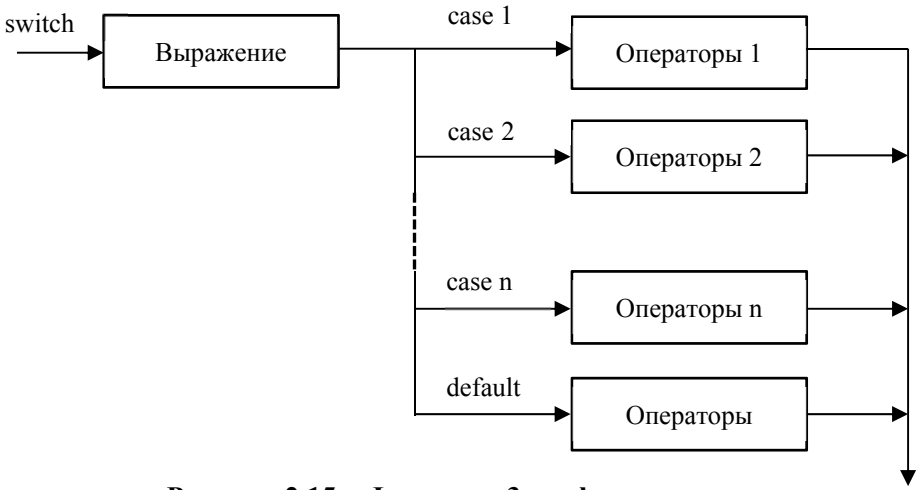


Рисунок 2.15 – Фрагмент 3 граф-схемы алгоритма

2.6.4 Оператор цикла с предусловием: WHILE

Операторы цикла используются для организации многократно повторяющихся вычислений. К операторам цикла относятся: цикл с предусловием ***while***, цикл с постусловием ***do while***, цикл с параметром *for* и цикл перебора ***foreach.***

Оператор цикла ***while*** организует выполнение одного оператора (простого или составного) неизвестное заранее число раз. Структура оператора ***while*** имеет вид:

while (логическое выражение)
{
Оператор (операторы);
Оператор (операторы) модификации параметра цикла;
}

While – это имя оператора цикла с предусловием, логическое выражение подробно рассмотрено в подразделе 2.5.2. ***Оператор (операторы), оператор (операторы) модификации параметра цикла*** – это группа операторов, которая может выполняться многократно (тело цикла)***. Параметр цикла*** – это переменная, которая управляет выполнением цикла. ***Параметр цикла*** обязательно должен присутствовать в ***логическом выражении*** и хотя бы в одном операторе тела цикла, в котором он должен быть обязательно модифицирован. Выполнение оператора ***while***: до выполнения оператора цикла в программе ***параметру цикла*** необходимо присвоить начальное значение, затем проверяется ***логическое выражение***, если оно имеет значение «истина», то выполняются все операторы тела цикла, в противном случае цикл (тело цикла) не выполнится ни разу; если цикл выполнился один раз, то управление передается в начало цикла (на проверку логического выражения); если логическое выражение имеет значение «истина», то опять выполняется тело цикла. Так тело цикла будет выполняться до тех пор, пока логическое выражение не примет значение «ложь».

Оператор цикла с предусловием ***while*** должен быть организован таким образом, чтобы в конечном счете значение логического выражения стало равным «ложь» (false). Иначе программа «зависнет», так как выйдет бесконечный цикл. Оператор ***while*** необходимо использовать для описания фрагмента 4 ГСА, представленного на рисунке 2.16. Примеры:

```
// Пример 1
  static void Main(string[] args)
  {
// Шкала температур Цельсий => Фаренгейт
     int tc; // текущее значение температуры по Цельсию
     double tf; // значение температуры по Фаренгейту
     tc = -50;
     Console.WriteLine("Шкала температур: Цельсий - Фаренгейт");
     while (tc <= 50)
     {
     // Формула преобразования в шкалу по Фаренгейту
        tf = 9.0 / 5.0 * tc + 32;
        tc++; // Переменная tc – это параметр цикла
        Console.WriteLine("{0} C   => {1} F", tc, tf);
     }
  }
 // Пример 2
  static void Main(string[] args)
  {
     // Вычислить значение числа Пи
     double pi; // результат
     int denom; // знаменатель
```

```
    int sign; // переменная, изменяющая знак числа + на -, и наоборот
    double t; // дополнительная переменная - текущая точность
    const double eps = 0.0000001; // точность 7 знаков после запятой
    int k; // количество слагаемых для получения заданной точности
    // Начальные приготовления
    k = 0;
    pi = 0;
    denom = 1;
    sign = 1;
    t = 1.0 / denom * sign;
    // Цикл while - Вычисление
    while (t > eps)
    {
        pi = pi + 4 * t * sign;
        sign = -sign; // Изменить знак
        denom += 2; // Изменить знаменатель
        t = 1.0 / denom; // t - это параметр цикла
        k++; // Увеличить на 1 количество слагаемых
    }
    // Вывод результата
    Console.WriteLine("Pi = {0}", pi);
    Console.WriteLine("k = {0}", k);
}
```

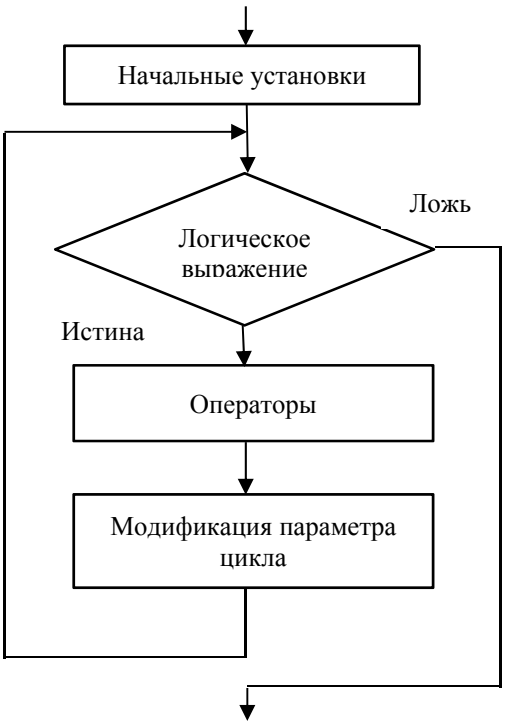


Рисунок 2.16 – Фрагмент 4 граф-схемы алгоритма

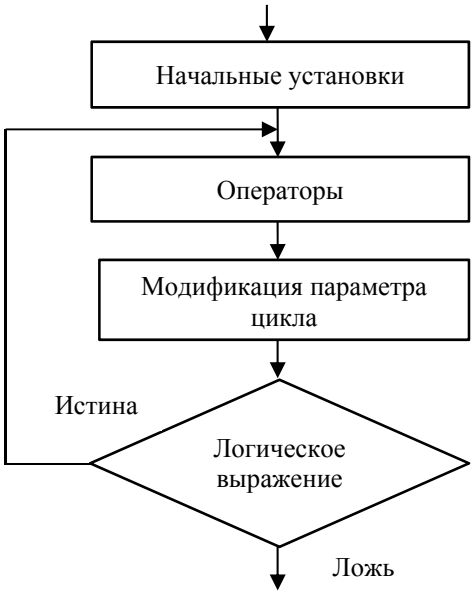


Рисунок 2.17 – Фрагмент 5 граф-схемы алгоритма

2.6.5 Оператор цикла с постусловием: DO…WHILE

Оператор ***do…while*** используется для организации циклического процесса. Отличие этого оператора от оператора ***while*** состоит в том, что тело оператора ***do…while*** будет выполнено как минимум один раз независимо от значения условия выполнения цикла. Структура оператора ***do…while*** имеет вид:

do
{
Оператор (операторы);
Оператор (операторы) модификации параметра цикла;
} while *(логическое выражение);*

Do и ***while*** – это ключевые слова для обозначения имени оператора цикла с постусловием. Остальные элементы структуры подробно описаны в подразделе 2.5.4. Выполнение оператора ***do…while***: до выполнения оператора цикла в программе ***параметру цикла*** необходимо присвоить начальное значение, затем выполняется тело цикла, далее проверяется ***логическое выражение***, если оно имеет значение «истина», то опять выполняется тело цикла; в противном случае происходит выход из цикла, то есть управление передается оператору, следующему за оператором ***do…while***. Оператор ***do…while*** необходимо использовать для описания фрагмента 5 ГСА, представленного на рисунке 2.17. Примеры:

```
// Пример 1
  static void Main(string[] args)
  {
      int number; // Вводимое число
      int summ = 0; // Сумма чисел
      // Цикл ввода чисел, 0 – это выход из программы
      do
      {
        // Ввести число с клавиатуры
        Console.Write("Введите число, если число 0, то выход из программы ");
        number = Int32.Parse(Console.ReadLine());
        // Вычислить сумму
        summ += number; // Переменная number – это параметр цикла
      } while (number != 0);
      // Вывести сумму
      Console.WriteLine("sum = {0}", summ);
      Console.ReadKey();
  }
// Пример 2
  static void Main(string[] args)
  {
      // Определение количества цифр 7 в числе
      int number;// Переменная для записи введенного числа
      int t;     // Переменная для хранения копии введенного числа
      int k;     // Количество цифр, равных 7
      // Ввод числа
      Console.Write("Введите положительное число:");
      number = Convert.ToInt32(Console.ReadLine());
      if (number<0)
      {
        Console.WriteLine("Ошибка, некорректное число.");
        Console.ReadKey();
        return;
      }
      // Вычисление количества цифр 7
      t = number; // сделать копию из number
      k = 0;
      // цикл вычисления k
      do
      {
        if (t % 10 == 7) k++;
        t = t / 10;
      } while (t > 0); // Переменная t – это параметр цикла
```

```
        Console.WriteLine("k = {0}", k);
        Console.ReadKey();
    }
```

2.6.6 Оператор цикла с параметром: FOR

Цикл ***for*** или оператор цикла с параметром предназначен для организации циклического процесса. С помощью цикла ***for*** можно организовывать циклический процесс любого типа, в котором:

– количество итераций цикла заведомо известно;

– количество итераций цикла неизвестно и определяется на основе выполнения некоторого условия. Структура оператора ***for*** имеет вид:

for ([инициализация]; [логическое выражение]; [модификация]) // Заголовок цикла
{
    Оператор (операторы); // Тело цикла
}

For – это имя оператора цикла с параметром (***for*** является ключевым словом для C#). Оператор ***for*** состоит из двух частей: заголовка цикла и тела цикла. Заголовок оператора цикла может состоять из трех элементов: ***инициализации, логического выражения, модификации.*** Исходя из структуры оператора, все три элемента могут и отсутствовать в заголовке цикла, но в этом случае точки с запятой должны присутствовать.

Инициализация – выражения (выражение), которые описывают и инициализируют переменные (переменную), использующиеся в цикле. Эта переменная (переменные) определяет количество итераций, которое должно выполняться в цикле. Эту переменную еще можно называть переменной-счетчиком или параметром цикла.

Логическое выражение определяет необходимость выполнения следующей итерации цикла. Если ***логическое выражение*** = «истина», то выполняется следующая итерация цикла. Если ***логическое выражение*** = «ложь», то происходит прекращение выполнения цикла и осуществляется переход к следующему оператору, который следует за оператором ***for***.

Модификация – некоторое выражение, изменяющее значение переменной (переменных) или параметра (параметров) цикла, определяющее (определяющих) количество итераций цикла. Необязательно выражение может изменять значение переменной (переменных). Также переменная (переменные) может изменять свое значение и в теле цикла. Цикл с параметром ***for*** реализован как цикл с предусловием ***while.***

Оператор ***for*** необходимо использовать для описания фрагмента 6 ГСА, представленного на рисунке 2.18. Примеры:

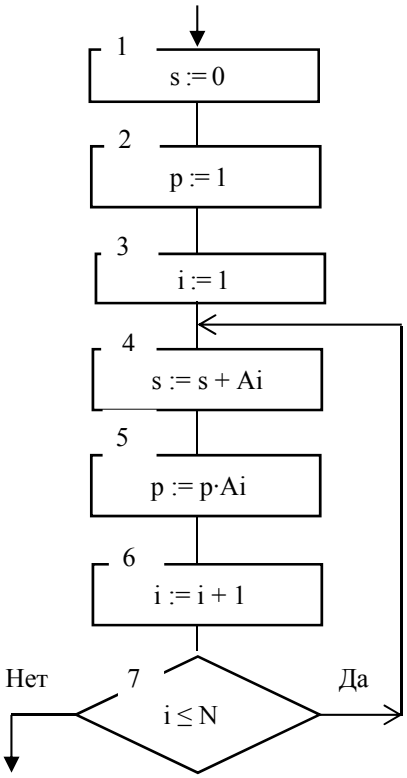


Рисунок 2.18 – Фрагмент 6 граф-схемы алгоритма

```
// Пример 1 – Код программы соответствует ГСА на рисунке 2.18
  static void Main(string[] args)
  {
      // Вычислить сумму и произведение элементов массива A
      int [] a = { 2, 1, 7, 3, 4 }; // Описание и определение массива A
      int s = 0;// Переменная для расчета суммы
      int p = 1;// Переменная для расчета произведения
      int n = 5;// Количество элементов массива A
      // i – параметр (переменная) цикла
      for (int i = 0; i < n; i++)// Заголовок оператора цикла
      {
          // Тело цикла
          S = s + a[i];
          P = p * a[i];
      }
  }
  // Ответ: сумма = 17, произведение = 168

// Пример 2
static void Main(string[] args)
  {
     // Вычислить произведение выражения a*(a+1)*...*(a+n-1)
     int proiz = 1; // Описание и определение переменной для вычисления произведения
     int i; // Описание переменной-счетчика цикла

     int n = 5;
     int a = 4;
     // Отсутствует выражение прироста переменной-счетчика i
     for (i = 0; i < n; )
     {
         proiz = proiz * (a + i);
```

```
          i++;
      }
  }
  // Ответ: proiz = 6720

// Пример 3
static void Main(string[] args)
{
            // Определить максимальную цифру числа n
            int n;
            int t, d;
            int max;
            // Ввод n
            Console.Write(" Введите n:");
            n = Convert.ToInt32(Console.ReadLine());
            t = n;
            max = 0; // Максимальная цифра
            // Цикл не содержит инициализации и прироста счетчика
            for (; t > 0; )            {
                d = t % 10;
                if (max < d) max = d;
                    t = t / 10;
            }
            Console.WriteLine("Max = {0}", max);
}
// Пример 4
/* Задано число a (1<a≤1.5). Найти такое наименьшее n, что в последовательно-
сти чисел 1+1/2, 1+1/3,…, 1+1/n последнее число есть меньше, чем a. */
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication4
{
    class Program
    {
        static void Main(string[] args)
        {
            // Найти наименьшее n
            int n;
            double a;
            double t;
            Console.Write("Введите число a:");
            a = Convert.ToDouble(Console.ReadLine());
            // В части инициализации помещаются два выражения
            for (n = 2, t = 1 + 1.0 / n; t >= a; n++)
                t = 1 + 1.0 / n;
            Console.WriteLine("n = {0}", n-1);
        }
    }
}
// Ответ: для a = 1.3 n = 4
// Пример 5
static void Main(string[] args)
  {
            // Найти все числа от 1 до 300, в которых ровно 5 делителей
            int num;
            int i, k;
            num = 1;
            while (num <= 300)
            {
```

```
                k = 0;
                for (i = 1; i <= num; i++) // Цикл for вложен в цикл while
                if (num % i == 0) k++;
                if (k == 5)
                    Console.WriteLine("{0}", num);
                num++;
            }
  }
// Ответ: 16 и 81
// Пример 6
static void Main(string[] args)
  {
  /* Найти позицию (номер) pos первого элемента массива, значение которого
     лежит в диапазоне от -5 до +5. */
            float[] B = { 12.4f, -11.9f, 7.5f, 2.3f, 1.8f };
            int pos;
           // Заголовок оператора цикла содержит только первый элемент
            for (pos = 0; ; )
            {
                if (pos == B.Length)
                    break;
                if ((B[pos] >= -5) && (B[pos] <= 5))
                    break;
                pos++;
            }
            if (pos == B.Length)
                Console.WriteLine("Искомого элемента нет в массиве.");
            else
                Console.WriteLine("pos = {0}", pos);
  }

Ответ: pos = 3
```

В примере 1 заголовок оператора цикла ***for*** описывает три вершины (3, 6, 7) ГСА, изображенной на рисунке 2.18. Пример 2: в заголовке оператора цикла опущено выражение, которое изменяет переменную-счетчик. В примере 3 заголовок оператора цикла не содержит выражений инициализации и прироста переменной счетчика. Обратите внимание (пример 4), что в заголовке цикла элемент инициализации представлен двумя выражениями. В примерах 1 и 2 количество итераций цикла заведомо известно, в примерах 3 и 4 – неизвестно. Пример 5 показывает вложенность цикла ***for*** в цикл ***while***. Обратите особое внимание на пример 6: количество элементов в массиве **B** вычисляется при помощи метода ***Length*** класса ***String*** (методы и классы будут рассмотрены во 2-й части учебного пособия), используется оператор ***break*** (будет рассмотрен в следующих подразделах) для выхода из цикла.

2.6.7 Оператор цикла для перебора элементов: FOREACH

В C# есть простая в использовании и элегантная альтернатива циклу ***for.*** Цикл ***foreach*** во время работы с массивами и коллекциями выполняет итерацию по их элементам, то есть последовательный перебор. **Итерация** – это общий термин, который описывает процедуру взятия элементов чего-то по очереди. Цикл ***foreach*** выполняет итерацию по каждому элементу – именно поэтому он называется ***foreach***. Структура оператора ***foreach*** имеет вид:

foreach (*тип идентификатор* ***in*** *контейнер*) // Заголовок цикла
{
    Оператор (операторы); // Тело цикла
}

Foreach – это имя оператора цикла (имя является ключевым словом C#). *Идентификатор* – имя переменной, которая используется в качестве итератора. Переменная *идентификатор* приобретает значение следующего элемента цикла на каждом шаге выполнения цикла ***foreach.*** Тип переменной *идентификатор* должен совпадать с типом массива или коллекции *контейнер*. Связь между *идентификатором* и *контейнером* реализуется с помощью ключевого слова ***in***.

Оператор цикла ***foreach*** работает следующим образом. При вхождении в цикл переменной *идентификатор* присваивается первый элемент массива (коллекции) – *контейнер*. На каждом следующем шаге итерации выбирается следующий элемент из *контейнера*, который сохраняется в переменной *идентификатор*. Цикл завершается, когда будут пересмотрены все элементы массива (коллекции) *контейнер*. **Коллекция в** C# – это группа связанных объектов, хранящихся в структуре. Существует много типов коллекций, определенных в .NET framework, каждый из которых предоставляет различную структуру объектов. К ним относятся коллекции для простых списков, очередей и стеков. Программирование с использованием коллекций в этом пособии рассматриваться не будет. Пример:

```
static void Main(string[] args)
 {
      // Объявление массива из 10 целых чисел
      int[] A = new int[10];
      // Заполнение произвольными значениями
      for (int i = 0; i < 10; i++)
            A[i] = i + 2 * i; // B[i] - элемент в позиции i массива B
       // Вывести массив для проверки
      Console.WriteLine("Массив A:");
      for (int i = 0; i < A.Length; i++)
      {
            Console.Write("{0 } ", A[i]);
      }
      Console.WriteLine();
      // Использование цикла foreach для вычисления суммы элементов массива
      double summ = 0;
      foreach (double item in A)
      {
            summ += item;
      }
      // Вывод суммы элементов массива
      Console.WriteLine("summ = {0}", summ);
      Console.ReadKey();
   }
Ответ:
Массив A:
0 3 6 9 12 15 18 21 24 27
summ = 135
```

2.6.8 Операторы передачи управления

2.6.8.1 Оператор безусловного перехода: GOTO

Оператор ***goto*** предназначен для выполнения безусловного перехода в программе. Структура оператора ***goto*** имеет вид:

goto *метка*;

Goto – это имя оператора безусловного перехода (имя является ключевым словом C#). *Метка* – это обычный идентификатор языка, который предназначен для организации

перехода в программе. Оператор ***goto*** передает управление оператору, помеченному меткой. В программировании на C# оператор ***goto*** может применяться в двух аспектах:

– в теле программы (некоторого метода) для организации перехода между операторами;

– в операторе ***switch*** для перехода к одной из ветвей ***case***.

Использование оператора ***goto*** в программе считается плохим тоном в программировании. Это связано с тем, что весьма частое использование оператора ***goto*** в программе путает программный код и усложняет его восприятие. Поэтому, по возможности, рекомендуется заменять ***goto*** операторами цикла. Однако это только рекомендация. Примеры:

```
// Пример 1
int s=0, n=8;
metka:
s += n; n++;
if (n < 10) goto metka;
Console.WriteLine(s);
using System;
// Пример 2 – практическое применение оператора goto
class Use_goto {
  static void Main() {
    int i = 0, j = 0, k = 0;
    for (i = 0; i < 10; i++) {
      for (j = 0; j < 10; j++) {
        for (k = 0; k < 10; k++) {
          Console.WriteLine("i, j, k: " + i + " " + j + " " + k);
          if (k == 3)
            goto stop;
        }
      }
    }
    stop:
      Console.WriteLine("Остановлено!  i, j, k: " + i + ", " + j
                        + " " + k);
  }
}
Ответ:
i, j, k: 0 0 0
i, j, k: 0 0 1
i, j, k: 0 0 2
i, j, k: 0 0 3
Остановлено!  i, j, k: 0, 0 3
// Пример 3 – Применение оператора goto в операторе switch
static void Main(string[] args)
{
       for (int i = 1; i < 5; i++)
       {
           switch (i)
           {
               case 1:
                   Console.WriteLine("В ветви case 1");
                   goto case 3;
               case 2:
                   Console.WriteLine("В ветви case 2");
                   goto case 1;
               case 3:
                   Console.WriteLine("В ветви case 3");
                   goto default;
               default:
                   Console.WriteLine("В ветви default");
```

```
                    break;
                }
                Console.WriteLine();
                Console.ReadKey();
            }
        // goto case 1;  // Ошибка! Безусловный переход к оператору switch недопустим.
    }
```

Ответ:

```
В ветви case 1
В ветви case 3
В ветви default

В ветви case 2
В ветви case 1
В ветви case 3
В ветви default

В ветви case 3
В ветви default

В ветви default
```

Комментарий к **примеру 2**: если бы не оператор ***goto***, то в приведенной выше программе пришлось бы прибегнуть к трем операторам ***if*** и ***break***, чтобы выйти из глубоко вложенной части этой программы. В данном случае оператор ***goto*** действительно упрощает код. И хотя приведенный выше пример служит лишь для демонстрации применения оператора ***goto***, вполне возможны ситуации, в которых этот оператор может на самом деле оказаться полезным. И последнее замечание: как следует из приведенного выше примера, из кодового блока можно выйти непосредственно, но войти в него так же непосредственно нельзя. Обратите внимание на то (**пример 3**), как оператор ***goto*** используется в операторе ***switch*** для перехода к другим его ветвям ***case*** или к ветви ***default***. Обратите также внимание на то, что ветви ***case*** не оканчиваются оператором ***break***. Благодаря тому, что оператор ***goto*** препятствует последовательному переходу от одной ветви ***case*** к другой, упоминавшееся ранее правило недопущения «провалов» не нарушается, а, следовательно, необходимость в применении оператора ***break*** в данном случае отпадает. Но как пояснялось выше, оператор ***goto*** нельзя использовать как внешнее средство для безусловного перехода к оператору ***switch***. Так, если удалить символы комментария в начале последней строки примера 3, приведенная выше программа не будет скомпилирована. Откровенно говоря, применение оператора ***goto*** в операторе ***switch***, в общем, не рекомендуется как стиль программирования, хотя в ряде особых случаев это может принести определенную пользу.

2.6.8.2 Оператор выхода из цикла: BREAK

Оператор ***break*** применяется для прерывания текущей итерации (break (broke, broken) – ломать, разрывать). С его помощью происходит выход из блока фигурных скобок оператора цикла либо оператора ***switch*** с дальнейшей передачей управления следующему оператору. Если задействуются вложенные операторы, ***break*** обеспечивает выход из самого внутреннего оператора. В языке C# оператор ***break*** имеет два основных направления применения:

– в операторах цикла оператор ***break*** используется для завершения циклического процесса (прерывание работы цикла). Такое действие необходимо, кода нужно прервать выполнение цикла в зависимости от некоторого условия;

– в операторе выбора ***switch*** применение оператора ***break*** нужно для реализации выхода из данного оператора. Структура оператора ***break*** имеет вид:

break;

Оператор ***break*** состоит из одного ключевого слова. Примеры:

```
// Пример 1
static void Main(string[] args)
{
      // Вычисление суммы элементов последовательности
      int num;
      int sum;
      Console.WriteLine("Enter numbers:");
      // Цикл ввода чисел, использован оператор do…while
      sum = 0; // искомая сумма
      do
      {
        num = Convert.ToInt32(Console.ReadLine());
        if (num == 15) break; // Выход из цикла, если введено число 15
        sum = sum + num;
      } while (true); // Бесконечный цикл
      Console.WriteLine("sum = {0}", sum);
      Console.ReadKey();
}
// Пример 2 – это пример 2 в примерах оператора switch и пример 6 в примерах
оператора for
```

2.6.8.3 Оператор перехода к следующей итерации: CONTINUE

Оператор ***continue*** применяется внутри тела цикла. Оператор прекращает выполнение текущей итерации и переходит к следующей итерации (следующему шагу цикла). Использование оператора ***continue*** в цикле эффективно, если нужно пропустить некоторые итерации в зависимости от условия. Структура оператора ***continue*** имеет вид:

continue;

Оператор ***continue*** состоит из одного ключевого слова. Примеры:

```
Пример 1
/* Данный оператор позволяет перейти к следующей итерации, не завершив до
конца текущую */
// Пример программы, которая находит сумму нечетных элементов массив
static void Main(string[] args)
{
   int[] numbers = { 4, 7, 13, 20, 33, 23, 54 };
   int s = 0;
   for (int i = 0; i < numbers.Length; i++)
   {
      if (numbers[i] % 2 == 0)
         continue; //переход к следующей итерации
      s += numbers[i];
      }
   Console.WriteLine(s);
   Console.ReadKey();
}
Пример 2
var sum = 0;
for(int i = 0; i < n; i++)
{
    if (i - a == 0)
```

```
    {
        continue;
    }
    sum+= 1 / (i - a);
}
```

Как видно из **примера 2**, при i = a будет получена ошибка «Деление на ноль». В данном случае необходимо пропускать значение счетчика, которое приводит к ошибке. Ключевое слово **var** ссылается на тип неявным способом. Это псевдоним любого типа. Реальный тип определит компилятор **C#**. Использование var никак не ухудшает производительность. Var – это отличный (и вкусный) синтаксический «сахар». Он делает программы короче и проще для чтения. Var может использоваться в коде методов и в теле циклов. Ключевое слово var может представлять любой тип, и какой это будет тип – определяется во время компиляции. После компиляции результат получится тот же самый, как если бы тип был точно указан.

2.6.8.4 Оператор возврата из функции: RETURN

Оператор ***return*** организует возврат из метода. Его можно также использовать для возврата значения. Имеются две формы оператора ***return***: одна – для методов типа ***void***, т. е. тех методов, которые не возвращают значения, а другая – для методов, возвращающих конкретные значения.

Структура оператора ***return*** имеет вид:

return;

Оператор ***return*** состоит из одного ключевого слова. Пример:

Пример 1:
```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int result = Sum(230);
            Console.WriteLine("Сумма четных чисел от 1 до 230 равна: " + re-
sult);
            Console.ReadLine();
        }
        // Метод, возвращающий сумму всех четных чисел
        // от 1 до s
        static int Sum(int s)
        {
            int mySum = 0;
            for (int i = 1; i <= s; i++)
                if (i % 2 == 0)
                    mySum += i;
            return mySum;
        }
    }
}
```
Ответ: Сумма четных чисел от 1 до 230 равна: 13340

Пример 2: это **пример 2** в примерах оператора **do…while**

Более подробно оператор ***return*** буден рассмотрен при изучении методов.

2.6.9 Консольный ввод и вывод

Консоль (Console) − характерная особенность ранних операционных систем (например, MS DOS), использующих интерфейс командной строки для интерактивного обмена информацией с пользователем. Консольные приложения используются и сейчас. По сравнению с графическим интерфейсом, интерфейс командной строки требует меньше системных ресурсов и предоставляет инструменты автоматизации для повторяющихся задач.

Наиболее яркими примерами интерфейсов командной строки (англ. *Command line interface, CLI*) являются: командная оболочка *Windows*, *PowerShell*, а также *Bash*, доступная на всех платформах (наибольшее распространение *Bash* получил в *Unix*-системах и *Mac*, присутствует также в компонентах подсистема *Windows* для *Linux* (англ. *Windows Subsystem for Linux, WSL*)).

В операционной системе *Windows* консоль называется окном командной строки, для вызова которой Вы можете пройти в меню *Пуск -> Командная строка*.

В языке C# нет операторов ввода и вывода, поэтому ввод с консоли и вывод на консоль реализованы в виде методов (подпрограмм). Такие методы реализованы на платформе .NET.

На платформе ***.NET*** такие операции в виде методов находятся в классе ***Console*** пространства имен ***System***, предоставляющем базовую поддержку для приложений, считывающих и записывающих символы в консоль стандартных входных и выходных потоков.

Для вывода информации на консоль применяются 2 ключевых метода класса ***Console*** – ***Console.WriteLine*** и ***Console.Write.*** Метод ***Write()*** выводит одно или несколько значений на экране без символа новой строки. Это означает, что любой последующий вывод будет напечатан в той же строке. Метод ***WriteLine()*** печатает один или несколько объектов в одной строке с символом новой строки, вставленным в конце. Это означает, что любой последующий вывод будет напечатан в новой строке.

Разница между ***Write()*** и ***WriteLine()*****: м**етод ***Write()*** используется для печати одного или нескольких объектов в одной строке без вставки символа новой строки в конце. Метод ***WriteLine()*** вставляет символ новой строки после печати вывода. В методе ***Write()*** курсор остается на той же строке, а в ***WriteLine()*** перемещается на следующую строку. Примеры:

Пример 1
```
static void Main(string[] args)
{
   Console.Write("Сегодня"); Console.Write(""); Console.Write("хорошая");
   Console.Write(""); Console.Write("погода.");
   Console.ReadKey();
}
```
Ответ: Сегодняхорошаяпогода.

Пример 2
```
static void Main(string[] args)
{
   Console.Write("Сегодня"); Console.Write(" "); Console.Write("хорошая");
   Console.Write(" "); Console.Write("погода.");
   Console.ReadKey();
}
```
Ответ: Сегодня хорошая погода.

Пример 3
```
static void Main(string[] args)
{
   Console.Write("Сегодня "); Console.Write("хорошая ");
   Console.Write("погода.");
   Console.ReadKey();
```

```
}
Ответ: Сегодня хорошая погода
Пример 4
static void Main(string[] args)
{
   Console.WriteLine("Сегодня"); Console.WriteLine("хорошая");
   Console.WriteLine("погода.");
   Console.ReadKey();
}
Ответ: Сегодня
       хорошая
       погода.
Пример 5
static void Main(string[] args)
{
   int x = 9;
   double h = 3.39;
   Console.WriteLine(x); Console.WriteLine(h);
   Console.ReadKey();
}
Ответ:
9
3,39
```

В примере 1 добавлено два оператора ***Console.Write("")***, которые ничего не выводят, и выводимые слова отображаются на консоли слитно. Для разделения слов одним пробелом в операторе вывода между двойными кавычками необходимо вставить пробел (**пример 2** – ***Console.Write(" ")***). А можно упростить программный код следующим образом: убрать 2 оператора ***Console.Write(" ")***, а в операторах вывода после первых двух слов в двойных кавычках поставить по одному пробелу (**пример 3**). Обратите внимание, что текст между двумя двойными кавычками означает вывод его на консоль в том виде, в котором он написан в программном коде. В примере 4 показано использование метода ***WriteLine().*** В **примере 5** показан вывод значений переменных ***x, h.*** Обратите внимание, что вещественное число в коде программы пишется через точку (3.39), а выводится через запятую (3,39). В **примерах 1–5** метод ***WriteLine()*** используется для вывода одной величины (одного параметра).

Рассмотрим примеры, когда необходимо вывести две величины и более.

```
Пример 6
static void Main()
{
            int     i = 9;
            double  y = 3.39;
            decimal d = 900m;
            string  s = "Ольга";
            Console.WriteLine( "i = " + i );                              // 1
            Console.WriteLine( "s = " + s );                              // 2
            Console.WriteLine( "y = {0} \nd = {1}", y, d );        // 3
}
Ответ:
i = 9
s = Ольга
y = 3,39
d = 900
```

В строках, помеченными комментариями "1" и "2", показан вывод в каждой строке не одной, а двух величин, поэтому прежде чем передавать их для вывода, их требуется "склеить" в одну строку с помощью операции +.

Оператор 3 иллюстрирует форматный (форматируемый) вывод. В этом случае используется другой вариант метода ***WriteLine()***. Первым параметром методу передается строковый литерал, содержащий, помимо обычных символов, предназначенных для вывода на консоль, *параметры* в фигурных скобках, а также *управляющие последовательности*. Параметры нумеруются с нуля, перед выводом они заменяются значениями соответствующих переменных в списке вывода.

Из управляющих последовательностей чаще всего используются символы перевода строки (\n) и горизонтальной табуляции (\t).

Методы ***Write() и WriteLine()*** также позволяют форматировать данные перед выводом. Форматирование позволяет вывести целые, вещественные числа, а также числа в денежном формате. Пусть в программе определена переменная ***myInt*** типа ***int:***

```
int myInt = 12345;
```

В предыдущих примерах вывод чисел осуществлялся без указания формата:

```
Console.Write("myInt = " + myInt);
Результат такой строки:
myInt = 12345;
```

Теперь попробуем применить один из форматов. Следующая строка переменную ***myInt*** выводит шириной в 8 позиций:

```
Console.Write("myInt = {0,8}", myInt);
Результат такой строки:
myInt = _ _ _ 12345;
```

Здесь символы {0,8} меняют формат выводимого значения. Первая цифра 0 – значение первой переменной, 8 – ширина значения (число позиций). Если ширина значения меньше, чем 8 символов, то перед числом ставятся пробелы.

Поэтому результат будет в виде: ***myInt*** = _ _ _ 12345, перед числом появятся 3 знака пробела. Если значение в формате числа меньше, чем указанная ширина, то число выводится полностью. С помощью фигурных скобок можно вывести сразу несколько значений:

```
int myInt = 123456, myInt2 = 67890;
Console.Write("myInt = {0,6},myInt2 = {1,5}", myInt, myInt2);
Результат этого фрагмента:
myInt = _ 12345, myInt2 = 67890
```

Вывод вещественного числа чуть сложней. Например, имеется такое определение:

```
double myDouble = 1234.56789;
```

Тогда следующая строка для значения ***myDouble*** резервирует 10 позиций, а для дробной части – 3:

```
Console.Write("myDouble = {0,10:f3}",myDouble);
Результат: myDouble = _ _1234.568
```

Здесь формат f3 указывает на вещественный тип, дробная часть будет иметь 3 цифры. Такой же формат можно использовать для типов ***float*** и ***decimal***:

```
float myFloat = 1234.56789f;
Console.WriteLine("myFloat= {0,10:f3}",
myFloat);
decimal myDecimal = 1234.56789m;
Console.WriteLine("myDecimal = {0,10:f2}", myDecimal);
Результат этих строк:
myFloat = _ _ 1234.568
myDecimal = _ _ 1234.57
```

Всего имеется 9 форматов (таблица 2.9).

Таблица 2.9 – Форматы методов *Write() и WriteLine()*

Символ формата	Описание
f или F	Формат для числа с плавающей точкой (вещественное)
e или E	Формат для экспоненциального числа
p или P	Процентный формат
n или N	Раздельное форматирование для разрядов числа
c или C	Формат для локальной денежной суммы
d или D	Формат для десятичного числа
g или G	Формат экспоненциального числа или числа с плавающей точкой
r или R	Число с плавающей точкой форматируется для вывода заданного значения числа
x или X	x или X целое число преобразуется в шестнадцатеричное

Приведем примеры использования форматов, описанных в таблице 2.9.

```
// Вывод целого числа по формату
int myInt = 12345;int myInt2 = 67890;
Console.WriteLine("myInt = {0,6},myInt2 = {1,5}",myInt, myInt2);
Console.WriteLine("myInt в формате 10:d ={0,10:d}", myInt);
Console.WriteLine("myInt в формате 10:x ={0,10:x}", myInt);
// Форматирование вещественных чисел
double myDouble = 1234.56789;
Console.WriteLine("myDouble в формате 10:f3 = {0,10:f3}", myDouble);
float myFloat = 1234.56789f;
Console.WriteLine("myFloat в формате 10:f3 = {0,10:f3}", myFloat);
decimal myDecimal = 1234.56789m;
Console.WriteLine("myDecimal в формате 10:f3 = {0,10:f3}", myDecimal);
Console.WriteLine("myFloat в формате 10:e3 = {0,10:e3}", myFloat);
Console.WriteLine("myFloat в формате 10:p2 = {0,10:p2}", myFloat);
Console.WriteLine("myFloat в формате 10:n2 = {0,10:n2}", myFloat);
Console.WriteLine("myFloat в формате 10:g2 = {0,10:g2}", myFloat);
Console.WriteLine("myFloat в формате r = {0,10:r2}", myFloat);
// Форматирование денежных значений
decimal myMoney = 15123.45m;
Console.WriteLine("myMoney в формате 10:c2 = {0,10:c2}", myMoney);
// Пользовательский формат
double y = 9.4579;
Console.Write(" y = {0,5:0.### 'руб.'} \n", y);
```

```
Console.WriteLine("10 / 3 = {0:#.###}", 10.0 / 3.0);
Результаты выполнения:
myInt = 12345,myInt2 =     67890
myInt в формате 10:d =     12345
myInt в формате 10:x =      3039
myDecimal в формате 10:f3 =  1234,568
myFloat в формате 10:f3 =  1,235e+003
myFloat в формате 10:f3 =     123456,80%
myFloat в формате 10:e3 =   1 234,57
myFloat в формате 10:g2 =    1,2e+03
myFloat в формате r = 1234,56787
myMoney в формате 10:c2 = 15 123,45р.
Y = 9,458 руб.
10 / 3 = 3,333
```

В предпоследнем примере используется пользовательский формат (выделен подчеркиванием): символ "#" обозначает количество разрядов поле запятой, если число имеет больше разрядов, то оно округляется по правилам арифметики (разряды чисел). В последнем примере используется шаблон, где выделено всего 3 знака после запятой. Символ "#" обозначает разряды чисел. Шаблон **"{#.###}"** указывает методу ***WriteLine()***, что необходимо отобразить три десятичных разряда в дробной части выводимого значения. Символ "#" в целой части числа на разрядность целой части не влияет.

А теперь рассмотрим основные методы ввода информации с консоли ***Console.Read()***, ***Console.ReadLine().*** Для ввода символа используется метод ***Console.Read()***. Этот метод возвращает тип ***int***, но при необходимости можно его преобразовать в символ:

```
char myChar = (char) Console.Read( )
```

Метод ***Console.Read( )*** после ввода символа требует нажатия клавиши **Enter**. Пример:

```
// Пример ввода отдельного символа
static void Main(string[] args)
{
      Console.Write("Введите символ: ");
      char myChar = (char) Console.Read();
      Console.WriteLine("Вы ввели " + myChar);
}
Результаты выполнения:
Введите символ: а
Вы ввели а
Введите символ: Павел
Вы ввели П
Введите символ: Светлана
Вы ввели С
```

Для ввода строки символов используется метод ***Console.ReadLine()***. Здесь применяют тип ***string***. Строка ввода:

```
string myStr = Console.ReadLine( )
```

Метод ***Console.ReadLine()*** после ввода строки символов требует нажатия клавиши **Enter**. Пример:

```
/* Пример ввода строки символов */
static void Main(string[] args)
{
      Console.Write("Введите строку: ");
      string myString = Console.ReadLine();
      Console.WriteLine("Вы ввели " + myString);
}
```
Результаты выполнения:
```
Введите строку: Hello, student!
Вы ввели Hello, student!
Введите символ: Привет, студент!
Вы ввели Привет, студент!
Введите символ: Учим язык C#.
Вы ввели Учим язык C#.
```

Ввод числовых данных производится в два этапа:

– разряды чисел вводятся в символьную переменную;

– строка символов преобразуется в число определенного типа.

Преобразование можно выполнить либо с помощью специального класса ***Convert***, определенного в пространстве имен **System**, либо с помощью метода ***Parse***, имеющегося в каждом стандартном арифметическом классе. Приведем примеры различных методов ввода:

```
static void Main(string[] args)
{
     Console.Write(" Введите строку: ");
     string s = Console.ReadLine();       // 1
     Console.WriteLine("s = " + s);
     Console.Write(" Введите символ: ");
     char c = (char)Console.Read();       // 2
     Console.ReadLine();                  // 3
     Console.WriteLine("c = " + c);
     string buf; // строка – буфер для ввода чисел
     Console.Write(" Введите целое число: ");
     buf = Console.ReadLine();
     int i = Convert.ToInt32(buf);        // 4
     Console.WriteLine(i);
     Console.Write(" Введите вещественное число: ");
     buf = Console.ReadLine();
     double x = Convert.ToDouble(buf);    // 5
     Console.WriteLine(x);
     Console.Write(" Введите вещественное число: ");
     buf = Console.ReadLine();
     double y = double.Parse(buf);        // 6
     Console.WriteLine(y);
     Console.Write(" Введите вещественное число: ");
     buf = Console.ReadLine();
     decimal z = decimal.Parse(buf);      // 7
     Console.WriteLine(z);
     Console.ReadLine();
}
```
Результаты выполнения:
```
Введите строку: Добрый день.
S = Добрый день.
 Введите символ: Ж
C = Ж
Введите целое число: 25369
25369
```

```
 Введите вещественное число: 3455,455
3455,455
 Введите вещественное число: 43455,567
43455,567
Введите вещественное число: 73455,5679
73455,5679
```

Ввод строки выполняет оператор 1. Длина строки не ограничена, ввод выполняется до символа перевода строки. Ввод символа выполняется с помощью метода ***Read()***, который считывает один символ из входного потока (оператор 2). Метод возвращает значение типа ***int***, представляющее собой код символа, или -1, если символов во входном потоке нет (например, пользователь нажал клавишу **Enter**). Поскольку нам требуется не ***int***, а ***char***, а неявного преобразования от ***int*** к ***char*** не существует, приходится применить операцию явного преобразования типа. Оператор 3 считывает остаток строки и никуда его не передает. Это необходимо потому, что ввод выполняется через буфер. Фактически данные сначала заносятся в буфер, а затем считываются оттуда процедурами ввода. Занесение в буфер выполняется по нажатию клавиши **Enter** вместе с ее кодом. Метод ***Read()***, в отличие от ***ReadLine()***, не очищает буфер, поэтому следующий после него ввод будет выполняться с того места, на котором закончился предыдущий. В операторах 4 и 5 используются методы класса ***Convert***, в операторах 6 и 7 – методы ***Parse*** классов ***Double*** и ***Decimal*** библиотеки **.NET**, которые используются здесь через имена типов ***double*** и ***decimal***.

2.7 Платформа .NET

Совокупность средств, с помощью которых программы пишут, корректируют, преобразуют в машинные коды, отлаживают и запускают, называют средой разработки, или оболочкой.

Среда разработки обычно содержит:

– текстовый редактор, предназначенный для ввода и корректировки текста программы;

– компилятор, с помощью которого программа переводится с языка, на котором она написана, в машинные коды;

– средства отладки и запуска программ;

– общие библиотеки, содержащие многократно используемые элементы программ;

– справочную систему и другие элементы.

Под платформой понимается нечто большее, чем среда разработки для одного языка. Платформа **.NET** (произносится «дот нет») включает не только среду разработки для нескольких языков программирования, называемую **Visual Studio.NET**, но и множество других средств (например, механизмы поддержки баз данных, электронной почты и коммерции).

В эпоху стремительного развития Интернета – глобальной информационной сети, объединяющей компьютеры разных архитектур, важнейшими задачами при создании программ становятся:

– переносимость – возможность выполнения на различных типах компьютеров;

– безопасность – невозможность несанкционированных действий;

– надежность – способность выполнять необходимые функции в предопределенных условиях; средний интервал между отказами;

– использование готовых компонентов для ускорения разработки;

– межъязыковое взаимодействие – возможность применять одновременно несколько языков программирования.

Платформа **.NET** позволяет успешно решать все эти задачи. Для обеспечения переносимости компиляторы, входящие в состав платформы, переводят программу не в машинные коды, а в промежуточный язык (**Microsoft Intermediate Language, MSIL**, или просто **IL**), который не содержит команд, зависящих от языка, операционной системы и типа компьютера. Программа на этом языке выполняется не самостоятельно, а под управлением системы, которая называется общеязыковой средой выполнения (**Common Language Runtime, CLR**). Среда **CLR** может быть реализована для любой операционной системы.

Во время работы программы среда **CLR** следит за тем, чтобы выполнялись только разрешенные операции, осуществляет распределение и очистку памяти и обрабатывает возникающие ошибки. Это многократно повышает безопасность и надежность программ. Платформа **.NET** содержит огромную библиотеку классов (кратко будут рассмотрены в следующих главах), которые можно использовать при программировании на любом языке **.NET**. Библиотека имеет несколько уровней. На самом нижнем находятся базовые классы среды, которые используются при создании любой программы: классы ввода-вывода, обработки строк, управления безопасностью, графического интерфейса пользователя, хранения данных и пр.

Подробное изучение библиотеки классов **.NET** – необходимая, но и наиболее трудоемкая задача программиста при освоении этой платформы. Библиотека классов вместе с **CLR** образуют каркас (**framework**), то есть основу платформы. Платформа **.NET** рассчитана на объектно-ориентированную технологию создания программ, поэтому прежде чем начинать изучение языка C#, необходимо познакомиться с основными понятиями объектно-ориентированного программирования (ООП).

Среда разработки **Visual Studio.NET** предоставляет мощные и удобные средства написания, корректировки, компиляции, отладки и запуска приложений, использующих **.NET**-совместимые языки. Корпорация Microsoft включила в платформу средства разработки для четырех языков: C #, VB.NET, C++ и J#. Платформа **.NET** является открытой средой. Это значит, что компиляторы для нее могут поставляться и сторонними разработчиками. К настоящему времени разработаны десятки компиляторов для **.NET**, например Ada, COBOL , Delphi, Eiffel, Fortran, Lisp, Oberon, Perl и Python. Все .**NET**-совместимые языки должны отвечать требованиям общеязыковой спецификации (Common Language Specification, CLS), в которой описывается набор общих для всех языков характеристик. Это позволяет использовать для разработки приложения несколько языков программирования и вести полноценную межъязыковую отладку. Все программы независимо от языка используют одни и те же базовые классы библиотеки **.NET**. Приложение в процессе разработки называется проектом. Проект объединяет все необходимое для создания приложения: файлы, папки, ссылки и прочие ресурсы. Среда **Visual Studio.NET** позволяет создавать проекты различных типов, например:

– Windows-приложение использует элементы интерфейса Windows, включая формы, кнопки, флажки и пр.;

– консольное приложение выполняет вывод «на консоль», то есть в окно командного процессора;

– библиотека классов объединяет классы, которые предназначены для использования в других приложениях;

– веб-приложение – это приложение, доступ к которому выполняется через браузер (например, Internet Explorer), которое по запросу формирует веб-страницу и отправляет ее клиенту по сети;

– веб-сервис – компонент, методы которого могут вызываться через Интернет.

Несколько проектов можно объединить в решение (**solution**). Это облегчает совместную разработку проектов. В настоящее время существуют следующие версии **Visual**

Studio.NET: **Visual Studio 2005, Visual Studio 2008, Visual Studio 2010, Visual Studio 2013, Visual Studio 2015, Visual Studio 2017, Visual Studio 2019, Visual Studio 2022.** Все перечисленные версии существуют и с русскоязычным интерфейсом.

2.8 Консольные приложения

Среда **Visual Studio.NET** работает на платформе **Windows** и ориентирована на создание Windows-приложений и веб-приложений, однако разработчики предусмотрели работу и с консольными приложениями. При запуске консольного приложения операционная система создает так называемое консольное окно, через которое идет весь ввод-вывод программы. Внешне это напоминает работу в операционной системе в режиме командной строки, когда ввод-вывод представляет собой поток символов. Консольные приложения наилучшим образом подходят для изучения языка, так как в них не используется множество стандартных объектов, необходимых для создания графического интерфейса. В данном учебно-методическом пособии предлагается создавать только консольные приложения, чтобы сосредоточить внимание на базовых свойствах языка C#. В следующем подразделе будут рассмотрены самые простые действия в среде **Visual Studio.NET**: создание и запуск на выполнение консольного приложения на C#.

2.8.1 Создание и запуск на выполнение консольного приложения в среде Microsoft Visual Studio 2010

Для запуска среды **Microsoft Visual Studio 2010** на рабочем столе Windows 7 необходимо нажать кнопку «Пуск» (рисунок 2.20), после этого появится окно, изображенное на рисунке 2.21.

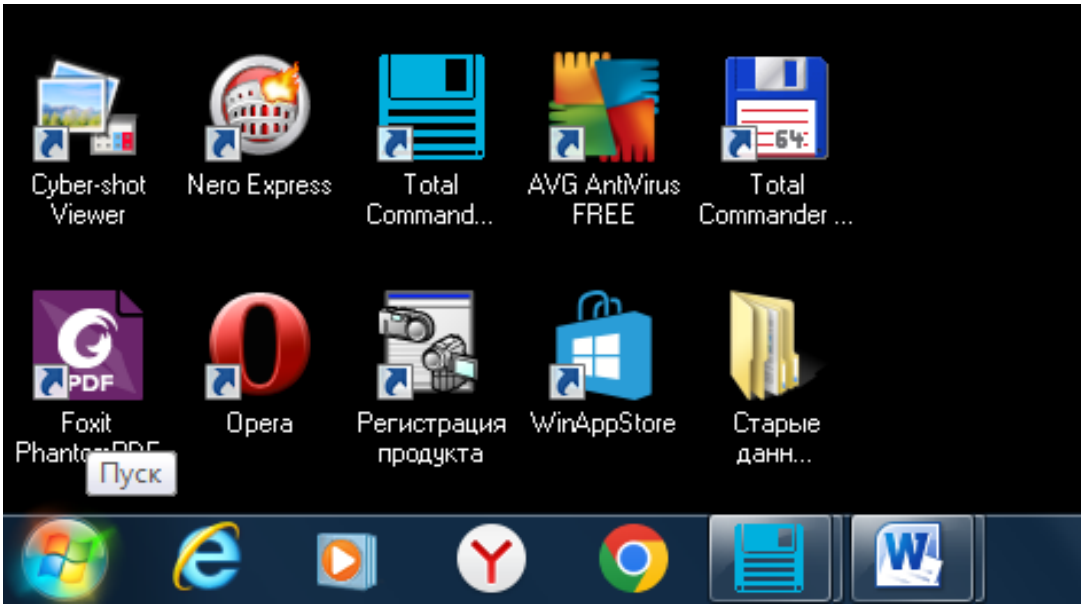


Рисунок 2.20 – Фрагмент рабочего стола ОС Windows 7

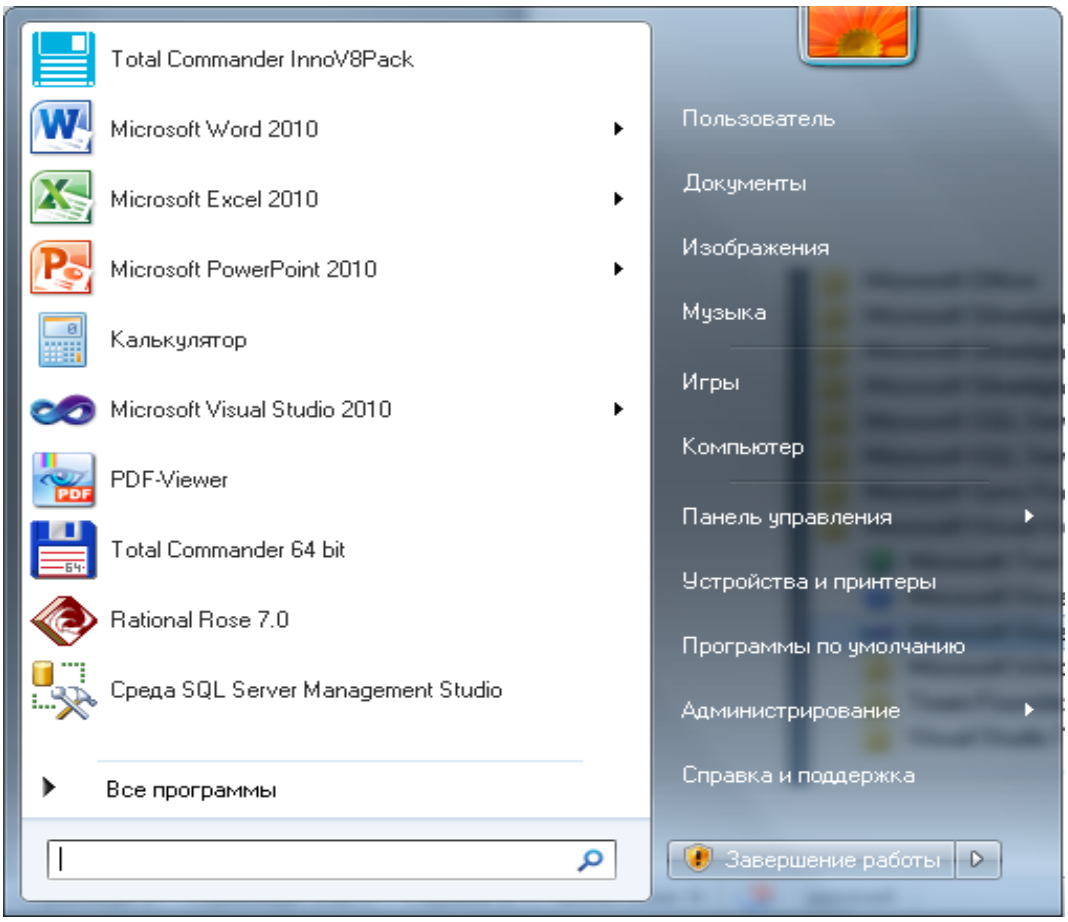


Рисунок 2.21 – Фрагмент окна при нажатии кнопки «Пуск»

Далее выбираем пункт меню «Все программы» (рисунок 2.21), затем пункт меню «Microsoft Visual Studio 2010» (рисунок 2.22), создаем новый проект (рисунок 2.23), выбираем консольное приложение (рисунок 2.24), на экране появится часть программного кода (заготовка) на языке программирования C# в среде редактирования (рисунок 2.25). Набираем две строки программного кода (строки 12, 13) и даем команду начать отладку и выполнение программы (Start debugging), нажав функциональную клавишу «F5» или кнопку «зеленый треугольник» на панели инструментов (рисунок 2.26). Если в программном коде не будет синтаксических ошибок, то программный код откомпилируется, программа выполнится, на экране появится результат работы программы с текстом «Добрый день» (рисунок 2.27). Вывод этого текста в программе обеспечивает оператор в строке 12, оператор в строке 13 дает возможность этот результат увидеть на экране, а далее при нажатии любой клавиши происходит переход в среду редактирования. А если в программе будут синтаксические ошибки, то компилятор укажет, в каких строках программного кода они находятся, а также характер ошибок.

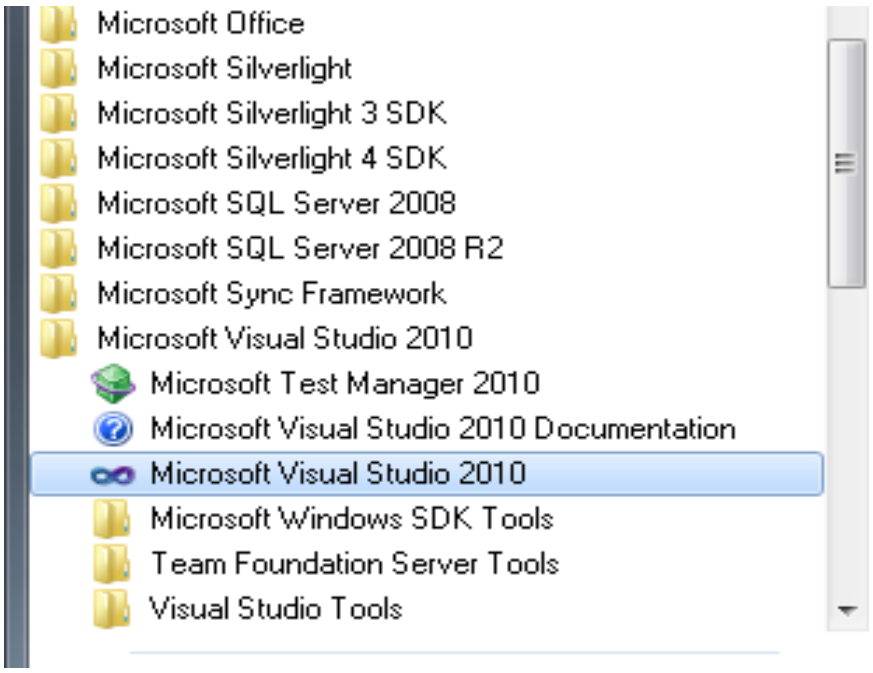


Рисунок 2.22 – Фрагмент окна при выборе пункта меню «Все программы»

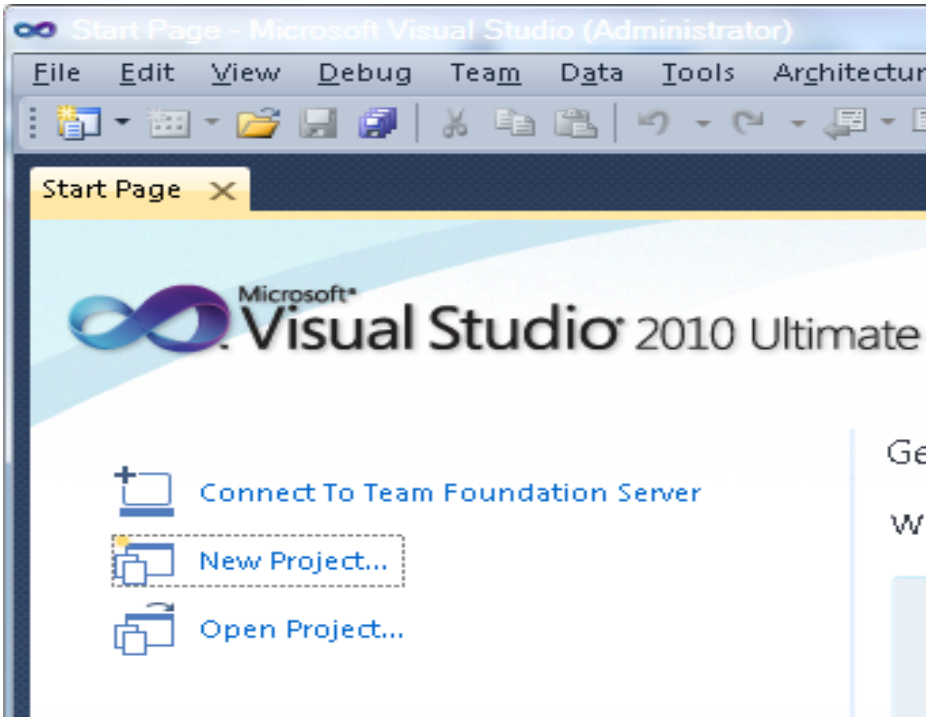


Рисунок 2.23 – Фрагмент окна после запуска Microsoft Visual Studio 2010

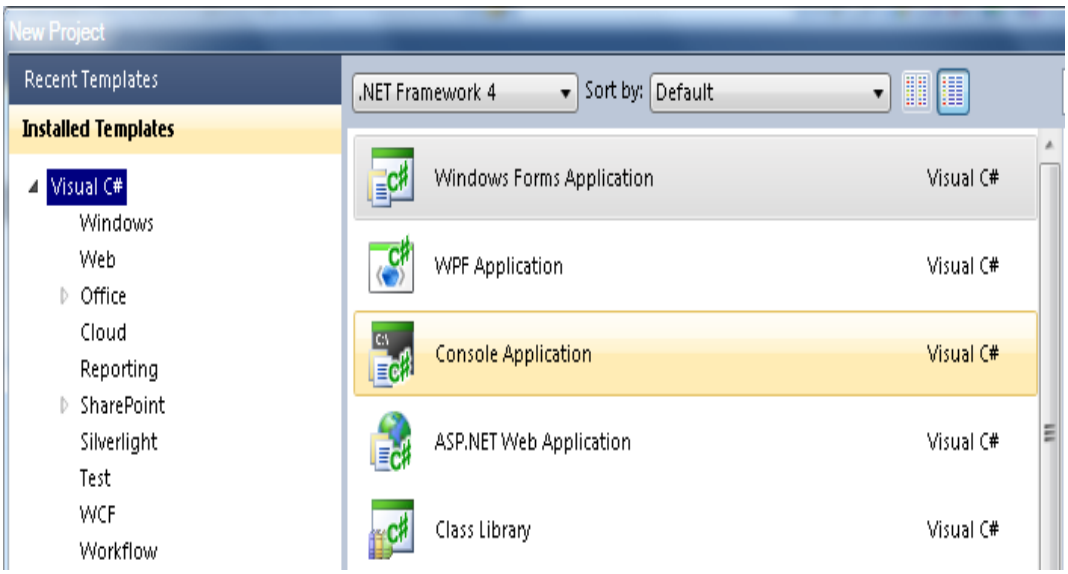


Рисунок 2.24 – Фрагмент окна после выбора пункта меню «New Project»

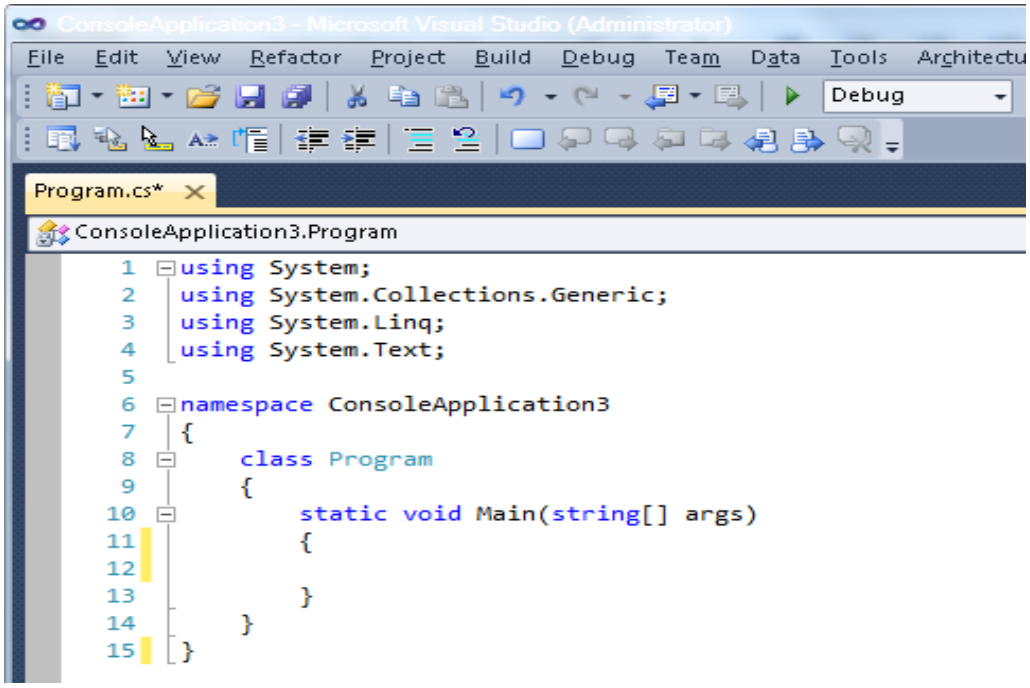


Рисунок 2.25 – Фрагмент окна после выбора пункта меню «Console Application»

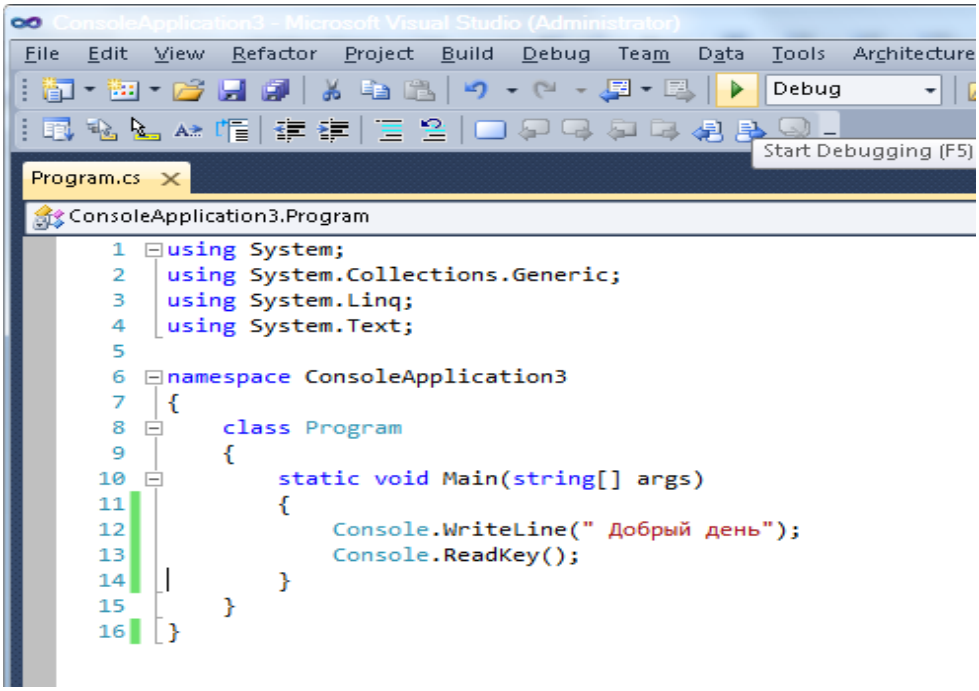


Рисунок 2.26 – Фрагмент окна после включения в программный код двух операторов (строки 12, 13)

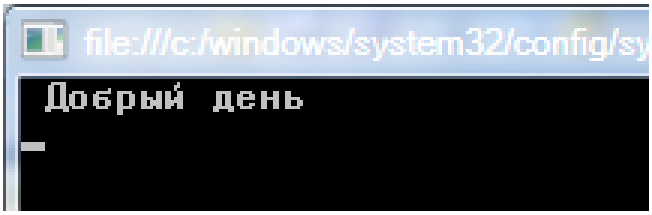


Рисунок 2.27 – Фрагмент окна с результатом работы программы, показанной на рисунке 2.26

2.8.2 Создание и запуск на выполнение консольного приложения в среде Microsoft Visual Studio 2019

После запуска среды **Microsoft Visual Studio 2019** появится окно, изображенное на рисунке 2.28.

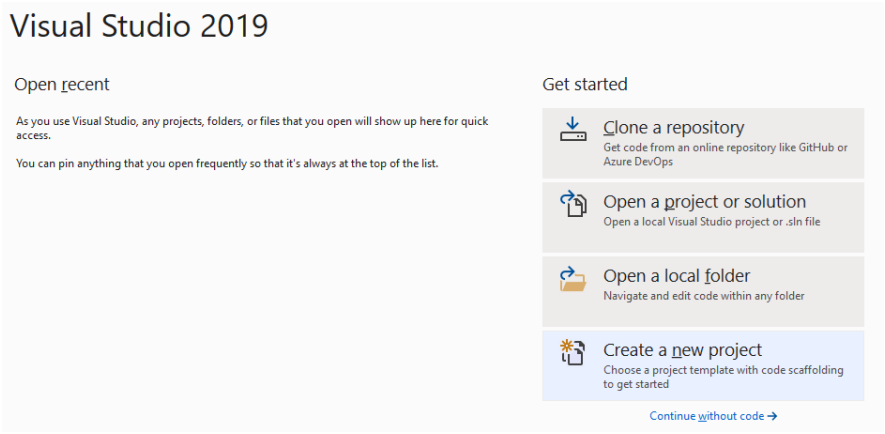


Рисунок 2.28 – Фрагмент окна после запуска Microsoft Visual Studio 2019

Далее выбираем пункт меню «Create a new project» (рисунок 2.28), затем пункт меню «Next» (рисунок 2.29), далее пункт меню «Next» (рисунок 2.30), затем пункт меню «Create» (рисунок 2.31), на экране появится программный код (вывод текста «Hello world» на консоль) на языке программирования C# в среде редактирования (рисунок 2.32). Теперь даем команду начать отладку и выполнение программы, нажав функциональную клавишу «F5» или кнопку «зеленый треугольник» на панели инструментов (рисунок 2.32). Если в программном коде не будет синтаксических ошибок, то программный код откомпилируется, программа выполнится, на экране появится результат работы программы с текстом «Hello world» (рисунок 2.33). Далее при нажатии любой клавиши происходит переход в среду редактирования. Если в программе будут синтаксические ошибки, то компилятор укажет, в каких строках программного кода они находятся, а также характер ошибок. В окне на рисунке 2.30 можно изменить имя проекта и место его хранения.

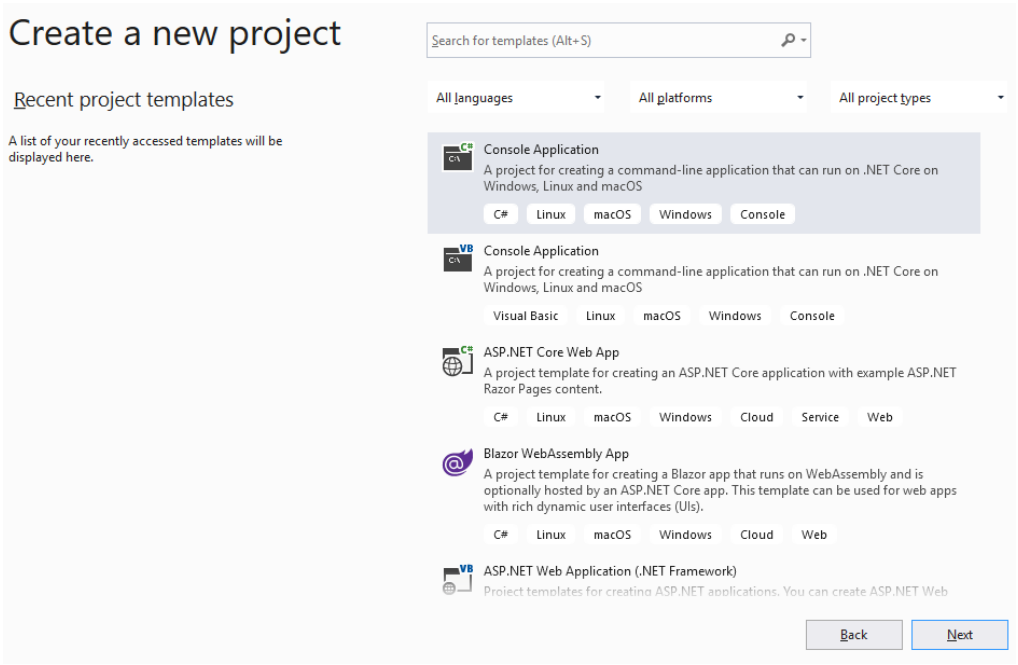


Рисунок 2.29 – Фрагмент окна после выбора пункта меню «Create a new project»

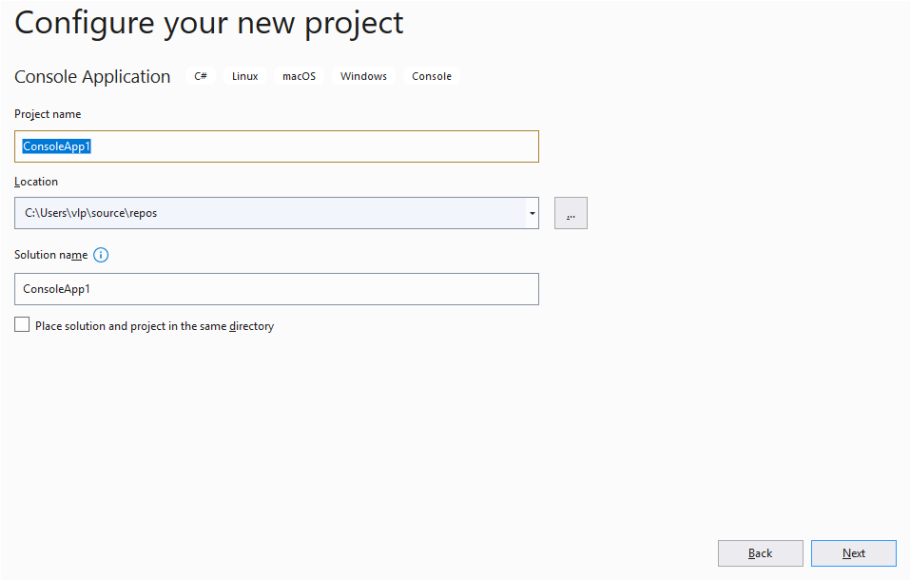


Рисунок 2.30 – Фрагмент окна после выбора пункта меню «Next»

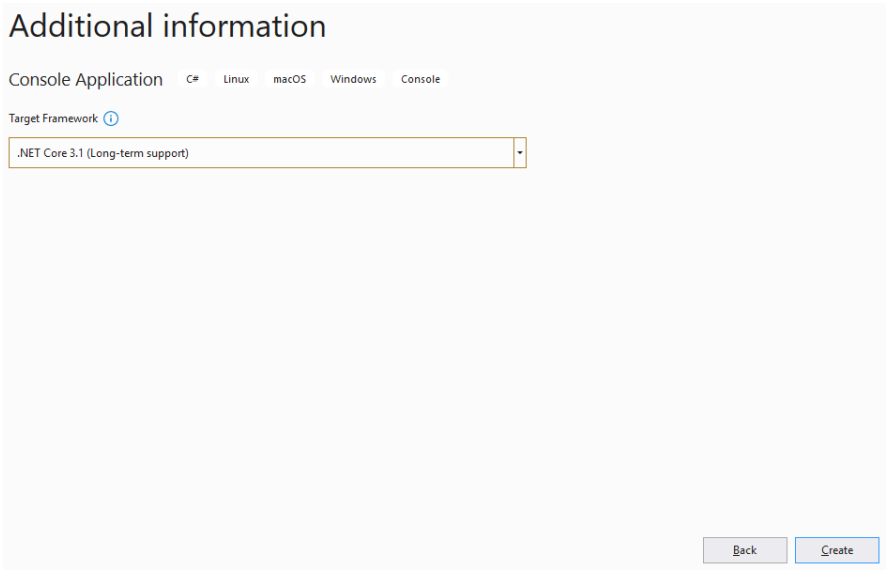


Рисунок 2.31 – Фрагмент окна после выбора пункта меню «Next» на рисунке 2.30

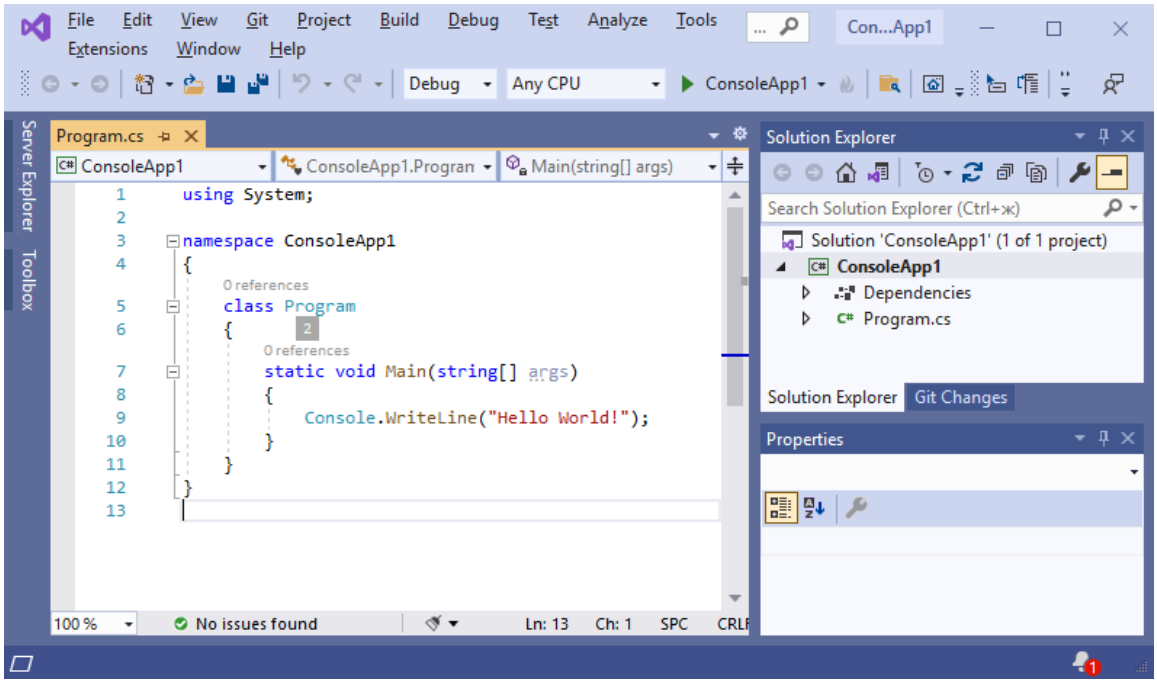


Рисунок 2.32 – Фрагмент окна после выбора пункта меню «Create» на рисунке 2.31

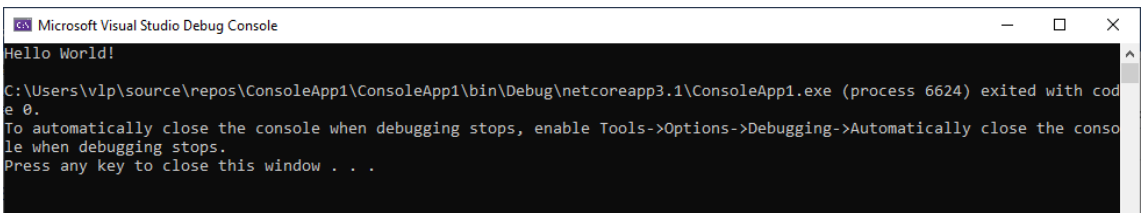


Рисунок 2.33 – Фрагмент окна с результатом работы программы, показанной на рисунке 2.31

ГЛАВА 3
УРОВНИ ПРЕДСТАВЛЕНИЯ ДАННЫХ В АВТОМАТИЗИРОВАННЫХ ИНФОРМАЦИОННЫХ СИСТЕМАХ

Как известно, автоматизированные информационные системы (АИС) хранят и обрабатывают информацию об объектах реального мира. Большие объекты из-за естественной сложности могут быть разделены, используя принцип декомпозиции, на отдельные единицы и узлы.

Некоторый набор информации, описывающей конкретный объект или его часть, называется логической записью или просто записью. Набор записей, описывающих набор объектов определенного класса, называется информационным массивом.

В реальном мире между объектами или их отдельными единицами существуют определенные отношения и взаимосвязи, которые имеют разную степень сложности. Во время разработки систем обработки и хранения информации эти отношения идентифицируются и отображаются путем структурирования записей и информационных массивов. Организация информационного массива, который обеспечивает определенные связи и отношения между данными, называется структурой данных. Чтобы обработка данных в компьютере не теряла своего содержимого, значение отношений, существующих между объектами в реальном мире, не нарушается, структуры должны поддерживаться, т. е. любые манипуляции, выполняемые с данными во время их обработки, не должны уничтожить структуру данных. Структура отображает свойства описываемого объекта, поэтому разрушение структуры приведет к потере ее свойств в результате неадекватного описания объекта.

Существует три уровня представления данных: логический уровень, уровень хранения и физический уровень.

На *логическом уровне* они работают с логическими структурами данных, которые отражают реальные отношения, существующие между объектами и их характеристиками, то есть указывают, как данные представляются пользователю системы. При разработке логических структур данных также учитываются информационные потребности пользователей системы и характер задач, для которых они предназначены. Единицей информации на этом уровне является логическая запись, объект, описываемый соответствующей логической записью, характеризуется определенными признаками – свойствами, которые выражаются как атрибуты записи. На логическом уровне разработчик системы устанавливает список функций, который полностью характеризует описанный класс объектов. Набор признаков и их взаимосвязь определяют внутреннюю структуру логической записи.

Логическая структура данных должна исчерпывающе характеризовать объекты, в отношении которых обрабатывается АИС; адекватно отражать реальные отношения между объектами и их характеристиками; обеспечить удовлетворение информационных потребностей пользователей системы и задач приложения. Какие из свойств объекта должны отражать атрибуты записи, разработчик системы решает на основе известных принципов достаточности.

На логическом уровне представления данных техническое и математическое программное обеспечение системы (тип компьютера, тип памяти, язык программирования, операционная система) не учитывается.

На *уровне хранения* они работают со структурами хранения – представлениями логической структуры данных в памяти компьютера. Структура хранения должна полностью отображать логическую структуру данных и поддерживать ее в процессе функционирования АИС. Единица информации на этом уровне также является логической записью. АИС должна преобразовывать логический уровень в уровень хранения, избегая искажений.

Операционная память машины и внешняя память имеют разные возможности, поэтому средства и методы организации данных в ОЗУ (оперативное запоминающее устройство) и на ВЗУ (внешнее запоминающее устройство) различны. При разработке или выборе структуры хранения принимается во внимание тип хранилища, в котором будут храниться данные, устанавливается тип и формат данных и определяется метод поддержания логической структуры.

Существуют различные способы представления данных в ОЗУ и ВЗУ. Одна и та же логическая структура данных может быть реализована в памяти компьютера различными структурами хранения. Каждая структура хранения обеспечивает определенный способ доступа к данным и определенные возможности для манипулирования данными. Он характеризуется количеством памяти, необходимой для хранения данных. Эффективность обработки данных напрямую зависит от выбора структуры хранения. Правильно подобранная структура хранилища обеспечивает быстрый поиск необходимых данных, возможность добавления новых и удаления устаревших записей без разрушения логической структуры, а также возможность корректировки записей, минимальное потребление памяти компьютера.

Структура хранилища поддерживается программным обеспечением. Для реализации ряда структур хранения требуются определенные языки программирования, поэтому при разработке или выборе структуры хранения следует учитывать возможности языка программирования, на котором будут написаны программы данных.

На *физическом уровне* представления данных работают с физическими структурами данных. На этом уровне решается задача реализации структуры хранения непосредственно в определенной памяти конкретного компьютера. Единицей информации на этом уровне является физическая запись, представляющая часть носителя, на которой размещена одна или несколько логических записей. При разработке структур памяти анализируются параметры конкретных технических средств: тип и объем памяти, метод адресации, метод и время доступа к данным. На этом уровне решаются задачи организации обмена данными между основной и внешней памятью компьютера.

При разработке структур данных всех уровней должен быть обеспечен принцип независимости данных. Физическая независимость данных означает, что изменения в физическом расположении данных и в технической поддержке системы не должны влиять на логические структуры и прикладные программы, то есть не должны вызывать изменения в них. Независимость логических данных означает, что изменения в структурах хранения не должны вызывать изменений в логических структурах данных и прикладных программах. Кроме того, изменения, внесенные в логические структуры данных из-за появления новых пользователей и новых запросов, не должны влиять на прикладные программы других пользователей системы.

Соблюдение принципа независимости данных позволяет использовать определенные типы данных: виртуальные и прозрачные данные.

Виртуальные данные существуют только на логическом уровне. Кажется, что эти данные действительно существуют у программиста, который использует их в своих программах. Каждый раз, когда к данным обращаются, операционная система определенным образом генерирует их на основе других данных, физически существующих в системе. Объявление некоторых данных виртуальными экономит память машины.

Кажется, что прозрачные данные не существуют на логическом уровне. Это позволяет скрыть от программиста или пользователя многие сложные механизмы, используемые при преобразовании логических структур данных в физические, и упростить прикладные программы.

ГЛАВА 4
СТРУКТУРА И СОДЕРЖАНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

Лабораторная работа должна быть логичной, учебно-исследовательской по своему содержанию. В ней в систематизированной форме должны быть изложены материалы проведенного исследования и его результаты.

Этапами подготовки лабораторной работы являются:

– выбор темы (варианта);
– составление плана;
– подбор литературы и ее изучение;
– написание основного текста отчета по лабораторной работе;
– оформление лабораторной работы по действующим ГОСТам, передача преподавателю для получения рецензии;
– защита лабораторной работы.

Лабораторная работа должна состоять из следующих структурных частей:

– титульный лист;
– оглавление;
– перечень условных обозначений и сокращений (при необходимости);
– введение;
– основная часть (главы, разделы и подразделы), включающая теоретическую и практическую часть. Содержание лабораторной части определяется кафедрой, за которой закреплена дисциплина;
– заключение;
– список использованных источников;
– приложения (при необходимости).

Титульный лист оформляется в соответствии с образцом оформления титульного листа лабораторной работы (ПРИЛОЖЕНИЕ А).

Оглавление включает в себя названия структурных частей лабораторной работы, названия всех глав, разделов и подразделов с указанием номеров страниц, на которых помещается начало материала соответствующих частей лабораторной работы, и должно соответствовать образцу оформления оглавления (ПРИЛОЖЕНИЕ Б).

Во введении (объем – 1–3 страницы) раскрывается актуальность и новизна темы, ее научная и практическая значимость, основные направления исследования, формулируются цели и задачи исследования, указывается предмет и объект исследования, а также характеризуются источники и материалы, использованные в процессе исследования.

Основная часть лабораторной работы делится на разделы. Разделы могут включать в себя главы, пункты или подразделы и пункты. Пункты при необходимости могут делиться на подпункты. Каждый подпункт должен содержать законченную информацию.

Первая глава (раздел) носит теоретический характер. В ней на основе изучения и анализа литературных источников раскрывается сущность исследуемого явления, его структура, рассматриваются различные подходы к решению поставленной проблемы, дается их оценка. Здесь же выделяются и раскрываются критерии и показатели изменений, происходящих в исследуемом явлении. Путем критической оценки и сопоставления мнений различных авторов характеризуются теоретические основы исследования вопросов темы, основных проблем, степень их разработки. Высказывается собственная точка зрения автора, которая может совпадать или не совпадать со взглядами ученых на эту проблему, или быть отличной от них. В основной части необходимо отразить научность, новизну и значимость полученных результатов, показать отличие полученных результатов от уже известных, описать степень новизны, объективно оценить личный вклад в разработку исследуемой проблемы.

Основная часть должна содержать данные, отражающие сущность, методику и основные результаты выполненного исследования:

– выбор направления исследования, включающий обоснование принятого направления исследования, метода решения задач и их сравнительную оценку, разработку общей методики исследования;

– теоретические и (или) экспериментальные исследования, включающие определение характера и содержания теоретических исследований, методов исследований;

– обобщение и оценку результатов исследования, включающих оценку полноты решения поставленной задачи и предложения по дальнейшим направлениям работы, оценку достоверности полученных результатов и их сравнение с аналогичными результатами отечественных и зарубежных работ. Рекомендуемый объем основной части лабораторной работы – 10–15 страниц.

В заключении (объем – не менее двух страниц) подводятся итоги работы, делаются выводы, разрабатываются рекомендации по конкретному использованию результатов лабораторной работы. Заключение должно быть кратким, обстоятельным и соответствовать поставленным целям и задачам.

Список использованных источников составляется с учетом требований библиографического описания изданий (ПРИЛОЖЕНИЕ В).

В основном тексте следует применять научно-технические термины, обозначения и определения, установленные действующими стандартами, а при их отсутствии – принятые в научно-технической литературе. Запрещается применять иностранные термины при наличии равнозначных слов и терминов в русском языке.

Приложения включают в себя вспомогательный материал (таблицы, иллюстрации вспомогательного характера, методики, инструкции, заполненные формы отчетности и т. п.), который при включении в основную часть лабораторной работы загромождает текст.

Примерная структура отчета лабораторной работы по дисциплине «Основы алгоритмизации и программирования» должна состоять из следующих элементов:

– титульный лист;

– ОГЛАВЛЕНИЕ, в свою очередь, включает:

– ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ (при необходимости);

– ВВЕДЕНИЕ;

– ГЛАВА 1 ПОСТАНОВКА ЗАДАЧИ;

– ГЛАВА 2 МАТЕМАТИЧЕСКОЕ ОПИСАНИЕ РЕШЕНИЯ ЗАДАЧИ (при необходимости);

– ГЛАВА 3 ОПРЕДЕЛЕНИЕ ВХОДНЫХ, ВЫХОДНЫХ И ВРЕМЕННЫХ ДАННЫХ;

– ГЛАВА 4 РАЗРАБОТКА СЛОВЕСНОГО И ГРАФИЧЕСКОГО ОПИСАНИЯ АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ;

– ГЛАВА 5 ТЕСТИРОВАНИЕ ГРАФИЧЕСКОГО ОПИСАНИЯ АЛГОРИТМА ДВУМЯ СПОСОБАМИ: ПОСЛЕДОВАТЕЛЬНЫЙ МЕТОД, ТРАССИРОВОЧНАЯ ТАБЛИЦА;

– ГЛАВА 6 РАЗРАБОТКА ОПИСАНИЯ АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ ВЫСОКОГО УРОВНЯ – ПРОГРАММНОГО ПРИЛОЖЕНИЯ;

6.1 Выбор, описание идентификаторов программного приложения и внутреннее представление входных, выходных и временных данных;

6.2 Разработка программного приложения и описание его работы;

6.3 Тестирование программного приложения и описание тестовых случаев;

– ЗАКЛЮЧЕНИЕ;

– СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ;

– ПРИЛОЖЕНИЯ.

Примечание. В Оглавлении для более глубокого понимания процесса, начиная с главы 4, используется слово *описание.* Это слово можно опустить, и тогда Оглавление будет иметь вид:

– ОГЛАВЛЕНИЕ, в свою очередь, включает:

– ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ (при необходимости);

– ВВЕДЕНИЕ;

– ГЛАВА 1 ПОСТАНОВКА ЗАДАЧИ;

– ГЛАВА 2 МАТЕМАТИЧЕСКОЕ ОПИСАНИЕ РЕШЕНИЯ ЗАДАЧИ (при необходимости);

– ГЛАВА 3 ОПРЕДЕЛЕНИЕ ВХОДНЫХ, ВЫХОДНЫХ И ВРЕМЕННЫХ ДАННЫХ;

– **ГЛАВА 4 РАЗРАБОТКА СЛОВЕСНОГО И ГРАФИЧЕСКОГО АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ;**

– **ГЛАВА 5 ТЕСТИРОВАНИЕ ГРАФИЧЕСКОГО АЛГОРИТМА ДВУМЯ СПОСОБАМИ: ПОСЛЕДОВАТЕЛЬНЫЙ МЕТОД, ТРАССИРОВОЧНАЯ ТАБЛИЦА;**

– **ГЛАВА 6 РАЗРАБОТКА АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ ВЫСОКОГО УРОВНЯ – ПРОГРАММНОГО ПРИЛОЖЕНИЯ;**

6.1 Выбор, описание идентификаторов программного приложения и внутреннее представление входных, выходных и временных данных;

6.2 Разработка программного приложения и описание его работы;

6.3 Тестирование программного приложения и описание тестовых случаев;

– ЗАКЛЮЧЕНИЕ;

– СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ;

– ПРИЛОЖЕНИЯ.

ГЛАВА 5
ОФОРМЛЕНИЕ ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

Отчет студенческой лабораторной работы должен быть грамотно написан и правильно оформлен. В тексте должны применяться технические и научные термины, обозначения и определения, установленные стандартами. Рекомендуется использовать повествовательную форму изложения текста (например, «разрабатываю», «проектируют» и т. д.).

Отчет по лабораторной работе выполняется с применением устройств ввода ПЭВМ и печатается на одной стороне листа белой бумаги формата А4 (210 × 297 мм) с использованием устройства вывода.

При наборе текста следует использовать текстовый редактор Microsoft Word со следующими параметрами: шрифт – Times New Roman, 14 пунктов; выравнивание текста – по ширине; междустрочный интервал – полуторный (можно одинарный); отступ для первой строки абзаца – 1,25 мм (пять пробелов); поля: левое – 30 мм, правое – 10 мм, верхнее и нижнее – 20 мм. Это составляет около 1 800 знаков на странице, включая пробелы, знаки препинания, т. е. 60–64 знака в строке, 28–30 строк на странице.

Текст основной части отчета по лабораторной работе делят структурные элементы: главы, разделы, подразделы, пункты.

Заголовки структурных частей отчета лабораторной работы: «ОГЛАВЛЕНИЕ», «ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ», «ВВЕДЕНИЕ», «ГЛАВА», «ЗАКЛЮЧЕНИЕ», «СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ», «ПРИЛОЖЕНИЯ» печатают прописными буквами по центру строк, используя полужирный шрифт на один-два пункта больше, чем шрифт основного текста. Так же печатают заголовки глав.

Заголовки разделов печатают строчными буквами (кроме первой прописной) с абзацного отступа полужирным шрифтом на один-два пункта больше, чем шрифт основного текста. Заголовки подразделов печатают с абзацного отступа строчными буквами (кроме первой прописной) полужирным шрифтом основного текста.

Пункты, как правило, заголовков не имеют. При необходимости заголовок пункта печатают с абзацного отступа полужирным шрифтом основного текста в подбор к тексту.

В конце заголовков глав, разделов и подразделов точку не ставят. Если заголовок состоит из двух или более предложений, их разделяют двоеточием или точкой (точками). В конце заголовка пункта ставят точку.

Расстояние между заголовком (за исключением заголовка пункта) и текстом должно составлять один межстрочный интервал. Если между двумя заголовками текст отсутствует, то расстояние между ними устанавливается в один межстрочный интервал. Расстояние между заголовками раздела и подраздела, а также между названием раздела (главы) и текстом (если отсутствуют подразделы в структуре раздела) – 1 межстрочный интервал, между названием подраздела (параграфа) и текстом – 1 межстрочный интервал.

Каждую структурную часть отчета по лабораторной работе следует начинать с нового листа. Нумерация страниц дается арабскими цифрами. Первой страницей лабораторной работы является титульный лист, который включают в общую нумерацию страниц. На титульном листе номер страницы не ставят, на последующих листах номер проставляют в центре нижней части листа без точки в конце.

Нумерация глав, разделов, подразделов, пунктов, рисунков, таблиц, формул, уравнений дается арабскими цифрами без знака «№». Номер главы ставят после слова «ГЛАВА». Разделы «ОГЛАВЛЕНИЕ», «ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ», «ВВЕДЕНИЕ», «ЗАКЛЮЧЕНИЕ», «СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ», «ПРИЛОЖЕНИЯ» не имеют номеров.

Разделы нумеруют в пределах каждой главы. Номер раздела состоит из номера главы и порядкового номера раздела, разделенных точкой, например: «2.3» (третий раздел второй главы).

Подразделы нумеруют в пределах каждого раздела. Номер подраздела состоит из порядковых номеров главы, раздела, подраздела, разделенных точками, например: «1.3.2» (второй подраздел третьего раздела первой главы).

Пункты нумеруют арабскими цифрами в пределах каждого подраздела. Номер пункта состоит из порядковых номеров главы, раздела, подраздела, пункта, разделенных точками, например: «4.1.3.2» (второй пункт третьего подраздела первого раздела четвертой главы). Главы, разделы, подразделы и пункты в тексте выделяют полужирным шрифтом.

Заголовок главы в тесте печатают с новой строки, следующей за номером главы. Заголовки разделов, подразделов, пунктов приводят после их номеров через пробел. Пункт может не иметь заголовка.

В конце нумерации глав, разделов, подразделов, пунктов, а также их заголовков точку не ставят.

В тексте обязательно делаются ссылки на авторов и источники, из которых взяты цитаты, таблицы, схемы и т. д. и результаты тех или иных исследований. Ссылки в тексте на тот или иной источник осуществляются путем приведения номера по списку использованных источников. Номер заключается в квадратные скобки, причем первая цифра соответствует номеру материала в списке использованных источников, а вторая (через запятую и пробел) – номеру страницы, с которой взята цитата. Различные источники отделяются друг от друга точкой с запятой.

Примеры оформления ссылок на «список использованных источников»:

1) А. Н. Леонтьев отмечал: «Предпосылкой всякой деятельности является та или иная потребность» [18, 20, 25].

2) Критический анализ теории «проб и ошибок» был дан К. Коффкой, одним из представителей гештальтпсихологии, работавшим в области обучения и психического развития ребенка [15].

3) В исследованиях ряда авторов [1–5, 17] установлено, что...

4) Если затекстовую ссылку приводят на конкретный фрагмент текста документа, в отсылке указывают порядковый номер и страницы, разделенные запятой. Например, в тексте: [10, с. 81], [10, с. 101] или [10, с. 81; 10, с. 101].

5) Как видно из исследований последних лет [12, 34, 52, с. 14; 64, с. 21].

Иллюстрации (фотографии, рисунки, чертежи, схемы, диаграммы, графики, карты и др.) и таблицы служат для наглядного представления в отчете по лабораторной работе характеристик объектов исследования, полученных теоретических и (или) экспериментальных данных и выявленных закономерностей.

Иллюстрации и таблицы следует располагать в лабораторной работе непосредственно на странице с текстом после абзаца, в котором они упоминаются впервые, или отдельно на следующей странице. Они должны быть расположены так, чтобы их было удобно рассматривать без поворота отчета по лабораторной работе или с поворотом по часовой стрелке. Листы, полностью заполненные иллюстрациями или таблицами, включают в общую нумерацию страниц. Если размеры иллюстраций и таблиц больше формата А4, их размещают на листе формата А3, такой лист учитывают как одну страницу.

Иллюстрации и таблицы обозначают соответственно словами «рисунок» и «таблица», нумеруют последовательно в пределах каждой главы. На все таблицы и иллюстрации должны быть ссылки в тексте отчета по лабораторной работе. Слова «рисунок», «таблица» в подписях к рисунку, таблице и в ссылках на них не сокращают (ПРИЛОЖЕНИЕ Г). Номер иллюстрации (таблицы) должен состоять из номера главы и порядкового номера иллюстрации (таблицы), разделённых точкой. Например: «рисунок 1.2» (второй рисунок первой главы);

«таблица 2.5» (пятая таблица второй главы). Если в отчете по лабораторной работе приведено лишь по одной иллюстрации (таблице) в каждой главе, то их нумеруют последовательно в пределах работы в целом, например: «рисунок 1», «рисунок 2»; «таблица 1», «таблица 2».

Цифровой материал отчета по лабораторной работе оформляют в виде таблиц. Каждая таблица должна иметь краткий заголовок, который состоит из слова «Таблица», ее порядкового номера и названия, отделенного от номера знаком тире. Заголовок следует помещать над таблицей слева, без абзацного отступа (ПРИЛОЖЕНИЕ Г). Например:

Таблица 5.1 – Площадь заболоченных и болотных земель Беларуси

Головка	**Географические провинции Беларуси**	**Площадь (тыс. га) со слоем торфа**		*Заголовки граф*
		менее 30 см	более 30 см	*Подзаголовки граф*
	Северная	37,5	21,3	*Строки (горизонтальные ряды)*
	Центральная	19,2	2,65	
	Южная	42,5	38,2	
	Боковик (графа для заголовков)	*Графы (колонки)*		

При оформлении таблиц необходимо руководствоваться следующими правилами:

– основной шрифт таблицы – на один-два пункта меньше, чем текст работы;

– не следует включать в таблицу графу «Номер по порядку». При необходимости нумерации показателей, включенных в таблицу, порядковые номера указывают в боковике таблицы непосредственно перед их наименованием;

– таблицу с большим количеством строк допускается переносить на следующий лист. При переносе части таблицы на другой лист ее заголовок указывают один раз над первой частью, слева над другими частями пишут слово «Продолжение». Если в работе несколько таблиц, то после слова «Продолжение» указывают номер таблицы, например: «Продолжение таблицы 1.2»;

– таблицу с большим количеством граф допускается делить на части и помещать одну часть под другой в пределах одной страницы, повторяя в каждой части таблицы боковик. Заголовок таблицы помещают только над первой частью таблицы, а над остальными пишут «Продолжение таблицы» или «Окончание таблицы» с указанием ее номера;

– таблицу с небольшим количеством граф допускается делить на части и помещать одну часть рядом с другой на одной странице, отделяя их друг от друга двойной линией и повторяя в каждой части головку таблицы. При большом размере головки допускается не повторять ее во второй и последующих частях, заменяя ее соответствующими номерами граф. При этом графы нумеруют арабскими цифрами;

– если повторяющийся в разных строках графы таблицы текст состоит из одного слова, то его после первого написания допускается заменять кавычками; если из двух или более слов, то его заменяют словами «То же» при первом повторении, а далее – кавычками. Ставить кавычки вместо повторяющихся цифр, марок, знаков, математических, физических и химических символов не допускается. Если цифровые или иные данные в какой-либо строке таблицы не приводят, то в ней ставят прочерк;

– заголовки граф и строк следует писать с прописной буквы в единственном числе, а подзаголовки граф – со строчной, если они составляют одно предложение с заголовком, и с прописной, если они имеют самостоятельное значение. Допускается нумеровать графы арабскими цифрами, если необходимо давать ссылки на них по тексту работы;

– заголовки граф, как правило, записывают параллельно строкам таблицы. При необходимости допускается располагать заголовки граф параллельно графам таблицы.

– головка таблицы отделяется линией от остальной части таблицы. Слева, справа и снизу таблица также ограничивается линиями. Горизонтальные и вертикальные линии,

разграничивающие строки и графы таблицы, могут не проводиться, если это не затрудняет ее чтение;

– не допускается разделять заголовки и подзаголовки боковика и граф диагональными линиями;

– в случае прерывания таблицы и переноса ее части на следующую страницу в конце первой части таблицы нижняя, ограничивающая ее черта, не проводится.

Формулы и уравнения в работе (если их более одной) нумеруют в пределах главы. Номер формулы (уравнения) состоит из номера главы и порядкового номера формулы (уравнения) в главе, разделенных точкой. Номера формул (уравнений) пишут в круглых скобках у правого поля листа на уровне формулы (уравнения), например: «(3.1)» – первая формула третьей главы.

При оформлении формул и уравнений необходимо соблюдать следующие правила:

– формулы и уравнения следует выделять из текста в отдельную строку. Выше и ниже каждой формулы и уравнения оставляется по одной свободной строке;

– если формула или уравнение не умещаются в одну строку, они должны быть перенесены после знака равенства (=) или после знаков плюс (+), минус (–), умножения (×) и деления (:). При этом повторяют знак в начале следующей строки;

– ссылки на формулы по тексту работы дают в скобках;

– пояснение значений символов и числовых коэффициентов, входящих в формулу или уравнение, следует приводить непосредственно под формулой или уравнением в той же последовательности, в какой они даны в формуле (уравнении). Значение каждого символа и числового коэффициента следует давать с новой строки. Первую строку пояснения начинают со слов «где» без двоеточия (ПРИЛОЖЕНИЕ Е).

Раздел «ПРИЛОЖЕНИЯ» оформляют в конце рукописи (отчета) либо в виде отдельной части (книги). Располагают приложения в порядке появления ссылок на них в тексте лабораторной работы. Не допускается включение в приложение материалов, на которые отсутствуют ссылки в тексте работы. Каждое приложение следует начинать с нового листа с указанием в правом верхнем углу слова «ПРИЛОЖЕНИЕ», напечатанного прописными буквами. Приложение должно иметь содержательный заголовок, который печатается с прописной буквы по центру строки. Приложения обозначают заглавными буквами русского алфавита, начиная с А (за исключением букв Ё, З, И, О, Ч, Ь, Ы, Ъ), например: «ПРИЛОЖЕНИЕ А», «ПРИЛОЖЕНИЕ Б», «ПРИЛОЖЕНИЕ В». Допускается обозначать приложения буквами латинского алфавита, за исключением букв I и O. При оформлении приложений отдельной частью (книгой) на титульном листе под названием отчета по лабораторной работе печатают прописными буквами слово «ПРИЛОЖЕНИЯ».

Текст каждого приложения при необходимости может быть разделен на разделы и подразделы, которые нумеруются в пределах каждого приложения, при этом перед номером раздела (подраздела) ставится буква, соответствующая обозначению приложения (например, «А1.2» – второй подраздел первого раздела приложения А). Так же нумеруются в приложении иллюстрации, таблицы, формулы и уравнения (ПРИЛОЖЕНИЕ Ж). Ссылки на приложения и элементы приложений (иллюстрации, таблицы, формулы и уравнения) в тексте лабораторной работы даются в круглых скобках, например (ПРИЛОЖЕНИЕ Ж), (рисунок Ж.1), (таблица Ж.4), или в тексте, не сокращая название элемента приложения (например, удельный вес тарифных окладов кафедры финансов отображен на рисунке Ж.2).

Графическое представление хода решения задачи – самый наглядный способ записи алгоритма. Блок-схемы – наиболее распространенный способ графического изображения алгоритма. Блок-схемы строятся по определенным правилам и включают в себя геометрические фигуры (блочные символы), соединенные между собой стрелками, указывающими порядок выполнения операций. Блочные символы стандартизованы и различаются по типу выполняемых действий (ГОСТ 19.002-80 и 19.003-80, международные стандарты ISO 2636-73

или ISO 1028-73). В таблице Н.1 представлены наиболее часто используемые блочные символы. Контуры символов и их размеры должны соответствовать ГОСТ 19.701-90. Символы должны быть, по возможности, одного размера. Символы в схеме должны быть расположены равномерно. Следует придерживаться разумной длины соединений и минимального числа длинных линий. Минимальное количество текста, необходимого для понимания функции данного символа, следует помещать внутри символа. Текст должен быть записан слева направо и сверху вниз. Для текста следует использовать чертежный шрифт по ГОСТ 2.304-81 с высотой букв не менее 2,5 мм. Сокращение слов в записях не допускается, за исключением установленных государственными стандартами. Если объем текста, помещенного внутри символа, превышает его размеры, следует использовать символ «комментарий». Комментарий помещается на свободном поле схемы алгоритма, по возможности вблизи поясняемого символа, и соединяется с ним штриховой линией (таблица Н.2).

Линии показывают потоки данных или управление. Направление потока слева направо и сверху вниз считается стандартным. Если поток имеет направление, отличное от стандартного, то применяется стрелка – указатель направления потока – по ГОСТ 2.307-68. Линии в схемах должны подходить к символу либо слева, либо сверху, а исходить либо справа, либо снизу. Линии должны быть направлены к центру символа. Толщина линий для вычерчивания символов и связей между ними должна быть одинаковой. Рекомендуется использовать толщину от 0,6 до 0,8 мм.

В схемах предусмотрено использование двух типов линий: сплошной тонкой для вычерчивания символов и потоков и штриховой для изображения связей символа с комментарием или выделения группы символов. В схемах следует избегать пересечений линий. В исключительных случаях допускается изображение пересекающихся линий. Если две и более линии объединяются в одну, то место их объединения должно быть смещено. Разрывы линий в схемах возникают при большой насыщенности символами, при длинных линиях потоков или размещении схемы на нескольких страницах. В этих случаях следует применить специальный символ «страничный соединитель». Внутри этого символа должна быть нанесена буква латинского алфавита, например А, с указанием порядкового номера соединения.

Если схема размещается на нескольких страницах, то следует применять межстраничный соединитель. Соединитель в начале разрыва называется внешним, а в конце разрыва – внутренним соединителем (ПРИЛОЖЕНИЕ К).

Отчеты лабораторных работ по техническим специальностям и дисциплинам оформляются в соответствии с требованиями стандарта ГОСТ 2.105-95 ЕСКД. Общие требования к текстовым документам и СТБ 1.5-96. Отчет по лабораторной работе должен быть сброшюрован с помощью пластиковой пружины.

Схемы, формулы, рисунки, таблицы выполняются чёрными чернилами (если работа отпечатана). Опечатки и неточности, обнаруженные при оформлении работы, также могут аккуратно исправляться чёрными чернилами.

ГЛАВА 6
МЕТОДИКА ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ

6.1 Лабораторная работа 1
Линейные алгоритмы

В этой лабораторной работе необходимо разработать линейный алгоритм решения задачи согласно варианту (таблица 6.4) и описать его тремя способами: словесное описание, графическое и на языке программирования высокого уровня. Алгоритм состоит из шагов (подходит для словесного описания), геометрических фигур (подходит для графического описания), операторов (подходит для языка программирования). В нашем случае будем понимать слова шаг, геометрическая фигура, оператор как определенную команду, то есть, другими словами, любой способ описания алгоритма представляет собой совокупность команд.

В описании методики выполнения лабораторной работы необходимо придерживаться структуры ОГЛАВЛЕНИЯ, предложенной в главе 4.

Опишем методику выполнения одной из задач.

1. Постановка задачи. Разработать алгоритм вычисления гипотенузы прямоугольного треугольника по известным значениям длин его катетов.

2. Математическое описание решения задачи. Математическим решением задачи является известная формула:

$$c = \sqrt{a * a + b * b},$$

где c – длина гипотенузы;
a, b – длины катетов.

3. Определение входных, выходных и временных данных. Входными данными являются значения катетов **a** и **b**. Выходными данными является длина гипотенузы **c**. Совокупность входных, выходных и временных данных относится к переменным алгоритма. Входными и выходными данными могут быть и константы различных групп (подраздел 2.1.4). Временные данные для решения этой задачи не нужны.

4. Разработка словесного и графического описания алгоритма решения задачи.

Словесное описание представим последовательностью шагов:

Шаг 1. Определить значение переменной **a,** то есть в ячейку памяти с именем **a** записать какое-нибудь число, например 3.

Шаг 2. Определить значение переменной **b,** то есть в ячейку памяти с именем **b** записать какое-нибудь число, например 4.

Шаг 3. Вычислить значение гипотенузы **c,** то есть в ячейку памяти с именем **c** записать значение выражения $\sqrt{a * a + b * b}$.

Шаг 4. Вывод вычисленного значения длины гипотенузы, то есть значение ячейки памяти с именем **c** или значение переменной **c.**

4′. Разработка словесного описания алгоритма решения задачи (2-й способ).

Словесное описание представим последовательностью шагов:

Шаг 1. Ввод переменных **a** и **b**.

Шаг 2. Вычислить значение гипотенузы **c,** то есть в ячейку **c** записать значение выражения $\sqrt{a * a + b * b}$.

Шаг 3. Вывод длины гипотенузы, то есть значение переменной **c.**

Команда **вывод** понимается как вывод содержимого переменной (переменных) на печатающее устройство (например, принтер), на экран монитора, в файл, в базу данных и любое другое устройство. Команда **ввод** понимается как ввод содержимого переменной (переменных) с экрана монитора, из файла, из базы данных и любого другого устройства.

Необходимо помнить, что при вводе переменных они автоматически и определяются, то есть получают конкретные значения.

Первый способ словесного описания алгоритма носит частный характер, то есть значение гипотенузы вычисляется только для одного прямоугольного треугольника. Такой алгоритм можно назвать алгоритмом одноразового использования. А для того чтобы алгоритм носил общий характер и являлся универсальным, то есть вычислял значение гипотенузы для любого прямоугольного треугольника, необходимо разделить алгоритм и входные данные. Разделить алгоритм и входные данные – это означает, что в алгоритме исходные данные представляются переменными, то есть в общем виде, а конкретные значения входных данных вводятся с экрана или из файла. Для ввода входных данных с экрана или файла необходимо использовать операцию (команду) **Ввод**. Поэтому 2-й способ словесного описания алгоритма позволяет решить задачу в общем виде и является универсальным.

Графическое описание алгоритма представляет собой совокупность графических символов или геометрических фигур (прямоугольников, параллелограммов, ромбов и других), соединенных между собой линиями связи, то есть схематичное (структурное) представление. Поэтому для разработки графического описания алгоритма двух вышеописанных способов решения задачи воспользуемся графическими символами, представленными в таблице 1.2. Для того чтобы графическое описание алгоритма соответствовало ГОСТам, необходимо обратиться к Приложению К. В дальнейшем любой графический символ алгоритма будем называть вершиной (можно блоком) алгоритма, а все вершины алгоритма (кроме первой и последней) нумеруются арабскими цифрами, что дает существенные преимущества в описании и использовании алгоритма (в тексте можно делать ссылку на номер вершины, переносить фрагменты алгоритма на другие страницы и т. п.). На рисунках 6.1 и 6.2 представлены графические описания алгоритмов 1-го и 2-го способов решения задачи соответственно. Обратите особое внимание, что первому шагу словесного описания алгоритма соответствует 1-я вершина графического описания алгоритма, второму шагу – 2-я вершина и так далее. Не всегда количество шагов совпадет с количеством вершин. Это сделано для того, чтобы лучше понять взаимосвязь двух способов описания алгоритма. Графическое описание алгоритма обладает следующими преимуществами: наглядностью, компактностью, возможностью быстро оценивать правильность решения задачи, а основным недостатком является трудность описания некоторых операций. Лучший вариант решения задачи – это использование двух способов описания алгоритма, так словесный способ позволяет раскрыть нюансы (детали) решения задачи.

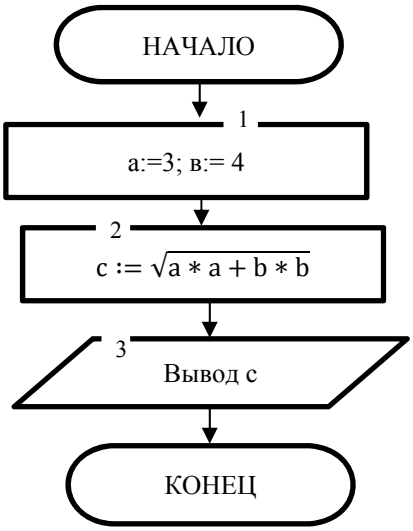


Рисунок 6.1 – Графическое описание алгоритма вычисления гипотенузы (1-й способ)

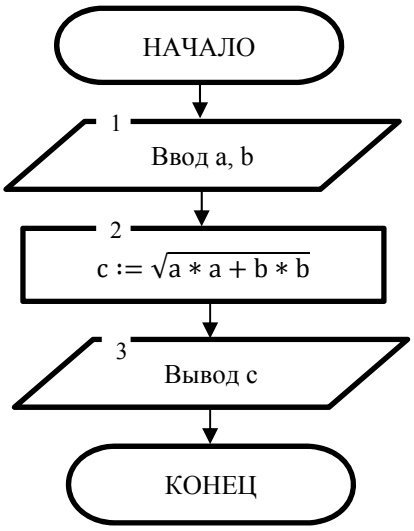


Рисунок 6.2 – Графическое описание алгоритма вычисления гипотенузы (2-й способ)

5. Тестирование графического описания алгоритма двумя способами: последовательный метод, трассировочная таблица.

Тестировать можно любой способ описания алгоритма, но, используя графический способ, это можно сделать быстрее. Последовательный метод тестирования представляет собой последовательность шагов. Если шаг представляет собой операцию присваивания, то в правой части операции необходимо указывать конкретное число, а если шаг представляет операцию сравнения, то в левой и правой части операции указываются числа. В шагах ввода и вывода также указываются числа. Опишем последовательный метод тестирования алгоритма, представленного на рисунке 6.1.

1. a:=3;
2. b:=4;
3. c:= $\sqrt{3 * 3 + 4 * 4}$;
4. Вывод 5.

Опишем последовательный метод тестирования алгоритма, представленного на рисунке 6.2.

1. Ввод a, b;
2. c:= $\sqrt{3 * 3 + 4 * 4}$;
3. Вывод 5.

Результатом решения этой задачи должно быть число, равное 5 (вычислить любыми инструментальными средствами, в этом случае можно и вручную). В результате тестирования получилось число, также равное 5. Делаем вывод: алгоритм разработан верно.

Протестируем алгоритм, представленный на рисунке 6.1, с использование трассировочной таблицы (таблица 6.1).

Таблица 6.1 – Трассировочная таблица тестирования алгоритма вычисления гипотенузы (1-й способ)

Номер шага	Команда алгоритма	Значение переменных			Выполняемое действие
		a	b	c	
1	a := 3	3			a := 3
2	b := 4		4		b := 4
3	c:=$\sqrt{a * a + b * b}$			5	c:=$\sqrt{3 * 3 + 4 * 4}$
4	Вывод c				Вывод 5

Протестируем алгоритм, представленный на рисунке 6.2, с использование трассировочной таблицы (таблица 6.2).

Таблица 6.2 – Трассировочная таблица тестирования алгоритма вычисления гипотенузы (2-й способ)

Номер шага	Команда алгоритма	Значение переменных			Выполняемое действие
		a	b	c	
1	Ввод a, b	3			Ввод 3, 4
2	c:=$\sqrt{a * a + b * b}$			5	c:=$\sqrt{3 * 3 + 4 * 4}$
3	Вывод c				Вывод 5

6. Разработка описания алгоритма решения задачи на языке программирования высокого уровня (C#) – программного приложения.

6.1 Выбор, описание идентификаторов программного приложения и внутреннее представление входных, выходных и временных данных.

Как было отмечено в главе 3 существует три уровня представления данных: логический уровень, уровень хранения и физический уровень. Поэтому внутреннее представление входных, выходных и временных данных представим на уровне хранения, так как на этом уровне необходимо учитывать возможности и особенности языка программирования, на котором будут написаны программные приложения. Для наглядности и краткости приведем в таблице 6.3 выбранные идентификаторы, их описание и их внутреннее.

Таблица 6.3 – Характеристики идентификаторов программного приложения

Наименование идентификатора	Описание	Название типа данных	Ключевое слово	Диапазон значений	Размер, байт	Класс типа		Назначение данных		
						Простой	Структурированный	Входные	Выходные	Временные
a	Катет	Целый	int	От $-2 \cdot 10^9$ до $2 \cdot 10^9$	32	+		+		
b	Катет	Целый	int	От $-2 \cdot 10^9$ до $2 \cdot 10^9$	32	+		+		
c	Гипотенуза	Вещественный	double	От $5.0 \cdot 10^{-324}$ до $1.7 \cdot 10^{308}$	64	+			+	

6.2 Разработка программного приложения и описание его работы.

Разработанное программное приложение на языке C# для вычисления гипотенузы (1-й способ) представлено на рисунке 6.3.

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using System.Text;
5.  namespace ЛинейныеПрограммы1
6.  {
7.      class Program
8.      {
9.          static void Main(string[] args)
10.             {
11.              // Описание переменных
12.             // Внутренне представление входных данных
13.             int a;
14.             int b;
15.             // Внутренне представление выходных данных
16.             double c;
17.             // Временных данных эта программа не использует
18.             // Определение переменных
19.             a = 3;
20.             b = 4;
21.             // Вычисление квадратного корня
22.             c = Math.Sqrt(a * a + b * b);
23.             // Вывод результата
24.             Console.WriteLine("c=" + c);
25.             Console.ReadLine();
26.             }
27.     }
28. }
```

Рисунок 6.3 – Код программы вычисления гипотенузы на языке C# (1-й способ)

Кратко опишем работу этой программы. Строки 1–4 подключают пространства имен (**System, System.Collections.Generic, System.Linq, System.Text**), суть которых описана в разделе 2.5. Строка 5 объявляет свое пространство имен «ЛинейныеПрограммы1», которое ограничивается фигурными скобками (строки 6, 28). В строке 7 объявляется класс с именем **«Program»**, это основной и единственный класс этой программы. Границы класса, так же как и пространства имен, обозначаются фигурными скобками (строки 7 и 27). В этой программе класс имеет только один метод – это метод **Main**. В строке 9 как раз и объявляется метод **Main**. Этот метод является главным в этой программе, это так называемая точка входа в программу. И это означает, что при запуске программы первым будет выполняться именно метод **Main**. Каждый метод тоже имеет границы, которые также обозначаются фигурными скобками (строки 10 и 26). Метод **Main** программы содержит следующие операторы: строки 13, 14 и 16 описывают переменные программы **a, b**, и **c** соответственно. В строках 19, 20 и 22 с помощью операторов присваивания определяются переменные программы **a, b**, и **c** соответственно. В строке 24 для вывода на экран монитора значения гипотенузы используется метод **WriteLine** с двумя параметрами: 1-й параметр "**c=**" (строковый литерал для вывода на экран того, что заключено в кавычки); 2-й параметр – значение переменной **c**. Для вывода значений этих параметров в одну строку их необходимо «склеить» с помощью операции +. А метод **ReadLine**, применяемый в строке 25, является вспомогательным, так как он заставляет программу ждать нажатия клавиши на клавиатуре и не дает ей до этого момента завершить свое выполнение (без этого оператора программа бы вывела строку и быстро закрылась, так что программист или пользователь даже бы не успели прочитать, что она вывела). Строки 11, 12, 15, 17, 18, 21 и 23 – это комментарии.

Разработанное программное приложение на языке C# для вычисления гипотенузы (2-й способ) представлено на рисунке 6.4.

```
1.      using System;
2.      using System.Collections.Generic;
3.      using System.Linq;
4.      using System.Text;
5.      namespace ЛинейныеПрограммы1
6.       {
7.              class Program
8.              {
9.              static void Main(string[] args)
10.                     {
11.                     // Описание переменных
12.                     // Внутренне представление входных данных
13.                     int a;
14.                     int b;
15.                     // Внутренне представление выходных данных
16.                     double c;
17.                     // Временных данных эта программа не использует
18.                     // Определение переменных
19.                     Console.Write("Введите значение катета a: ");       // Вывод на консоль
20.             a = int.Parse(Console.ReadLine());                  // Ввод катета a
21.             Console.Write("Введите значение катета b: ");       // Вывод на консоль
22.             b = int.Parse(Console.ReadLine());                  // Ввод катета b
23.                     // Вычисление квадратного корня
24.                     c = Math.Sqrt(a * a + b * b);
25.                     // Вывод результата
26.                     Console.WriteLine("c=" + c);
27.                     Console.ReadLine();
28.                     }
29.             }
30.     }
```

Рисунок 6.4 – Код программы вычисления гипотенузы на языке C# (2-й способ)

Код программы на рисунке 6.4 отличается от кода программы на рисунке 6.3 только механизмом определения входных данных. Для определения входных данных используется метод **ReadLine** (строки 20 и 22). **СТРОКУ 20 − Int a=int.Parse(Console.ReadLine())** необходимо понимать следующим образом: создается целочисленная переменная **a**, в которую заносится число, введенное с **консоли**. Все, что вводится с **консоли**, по умолчанию имеет тип **string** (строка), и чтобы преобразовать строку в целое число, используется метод **int.Parse**. Строки 19 и 21 необходимы для наглядности и понятности ввода данных, без них программа будет работать.

6.3 Тестирование программного приложения и описание тестовых случаев.

Существуют различные виды и методы тестирования. Далее будет описано тестирование методом «Черного ящика». Этот метод также известен как тестирование с закрытым ящиком, тестирование с использованием данных или функциональное тестирование. Для отладки и выполнения программы необходимо дать команду − **Start debugging**, нажав функциональную клавишу **F5** или кнопку «зеленый треугольник» на панели инструментов (рисунок 2.26). И если в программе нет синтаксических и семантических (логических) ошибок, то на экране появится следующее сообщение (рисунок 6.5).

Рисунок 6.5 – Фрагмент окна с результатом работы программы, показанной на рисунке 6.3

Так как значение гипотенузы **c**=5, делаем вывод, что программа работает правильно. Для того чтобы проверить эту программу при других исходных данных, необходимо поменять код в строках 19 или (и) 20. Далее откомпилировать эту программу и получить результат. Например, в строку 19 запишем a=5, а в строку 20 – b=6. Получим результат − рисунок 6.6.

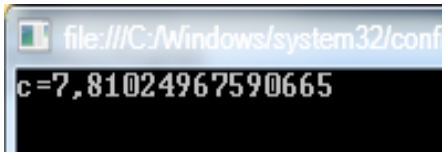


Рисунок 6.6 – Фрагмент окна с другим результатом работы программы, показанной на рисунке 6.3

После применения команды отладки и выполнения программы (рисунок 6.4) на экране монитора появится сообщение (рисунок 6.7). После этого необходимо ввести значение катета **a** и нажать клавишу **Enter,** затем ввести значение катета **b** и нажать клавишу **Enter** (пример экрана монитора – рисунок 6.8).

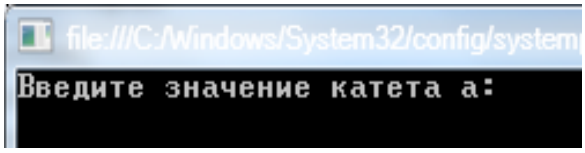


Рисунок 6.7 – Фрагмент окна с результатом работы программы, показанной на рисунке 6.4

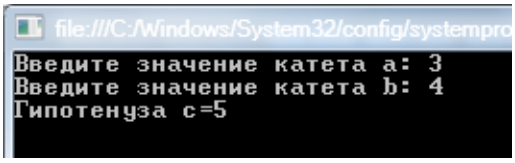


Рисунок 6.8 – Фрагмент окна с введенными входными данными и результатом работы программы, показанной на рисунке 6.4

После просмотра и анализа результата на рисунке 6.8 можно вернуться в среду **Visual Studio**, нажав любую клавишу. Эта программа отличается от предыдущей универсальностью, так как она работает при любых входных данных, ее не нужно каждый раз перекомпилировать.

Таблица 6.4 – Варианты заданий 1-й лабораторной работы

Вариант	Вид функции	Вариант	Вид функции
1	**2**	**3**	**4**
1	$b=\frac{1+\cos^2(x+z)}{\lvert x^3-2y^2\rvert}$	16	$b=x+\frac{\sqrt[3]{zy}}{y+\cos x}$
2	$b=\frac{\ln^2\lvert z\rvert}{\sqrt[3]{\lvert x\rvert+\lvert y\rvert}}$	17	$b=\lg\left(\sqrt{e^{x-y}+x^{\lvert y\rvert}+z}\right)$
3	$b=\frac{y^3}{x+y^3\cos^2 z}$	18	$b=1+\frac{x^2+1}{3+y^2}+\sin 2z$
4	$b=\sqrt{x+\sqrt[4]{\lvert y\rvert}}+\cos^2 z$	19	$b=\lvert\cos x+\cos y\rvert+2\sin^2 z$
5	$b=\frac{\sqrt[3]{e^{\sin x}}\cdot\cos y}{z^2+1}$	20	$b=\frac{\ln(y^3)(z-x/2)}{2\cos^2 x}$
6	$b=z(tg\ y-e^{-(x+3)})$	21	$b=\sqrt{10\left(\sqrt[3]{z}+x^{(y+2)}\right)}$
7	$b=\lvert x-y\rvert(\sin^2 z+tg\ z)$	22	$b=(\sin z)^2+\lvert x+y\rvert$
8	$b=\sqrt{y+\sqrt[3]{x}}-1+2z$	23	$b=e^{2z}-\sqrt[3]{y\lvert x\rvert}$
9	$b=x(tg\ z+\cos^2 y)$	24	$b=e^{(x-1)}+\sin y$

Окончание таблицы 6.4

1	2	3	4
10	$b = e^{\lvert x-y\rvert}\left(tg^{\,2} z + 1\right)^{x}$	25	$b = \sqrt{\lvert z\rvert e^{-(y+x/2)}}$
11	$b = \cos^{2} z + tg\ 2x + \lvert y\rvert$	26	$b = \dfrac{4y^{2}e^{2x}\sin^{2} z}{3z^{3} + \ln x}$
12	$b = 5\,tg\ z - 4y^{2} + \lvert xy\rvert$	27	$b = \dfrac{\sqrt{y\ln x} - z x^{2}}{1 + tg^{2} x^{2}} x$
13	$b = (z-x)\dfrac{y - \ln z}{1 + (y-x)^{2}}$	28	$b = \dfrac{\lg\left(y + \sqrt{z + x^{2}}\right)}{y + x^{2}}$
14	$b = y^{z} + \sqrt{\lvert x\rvert + \lvert y\rvert}$	29	$b = \dfrac{x^{2} + 4}{\sin^{2} z^{2} + x/2} y$
15	$b = \dfrac{\lg\left(\sqrt{x} + \sqrt{y} + 2\right)}{\lvert 2z\rvert}$	30	$b = \dfrac{\sin x + \sqrt{\lvert z - y\rvert}}{y(x-2) + x^{2}}$

Контрольные вопросы и задания к защите лабораторной работы:

1. Какой алгоритм называется линейным?
2. Как создаются и для чего используются тесты?
3. Как в программе описываются константы и переменные?
4. Какие вы знаете типы данных?
5. Что определяет тип данных?
6. Для чего используются и как записываются арифметические выражения?
7. Какие операции используются в арифметических выражениях?
8. Какие функции применяются в арифметических выражениях?
9. Для чего используется и как записывается оператор присваивания?

6.2 Лабораторная работа 2
Разветвляющиеся алгоритмы

В этой лабораторной работе необходимо разработать разветвляющиеся алгоритмы решения задачи согласно вариантам (таблица 6.8–6.9) и описать их тремя способами – словесное описание, графическое и на языке программирования высокого уровня. В описании методики выполнения лабораторной работы необходимо придерживаться структуры ОГЛАВЛЕНИЯ, предложенной в главе 4.

Опишем методику выполнения одной из задач.

1. Постановка задачи. Разработать алгоритм вычисления функции в зависимости от диапазона изменения ее аргумента.

2. Математическое описание решения задачи. Математическим решением задачи является известная формула:

$$y = \begin{cases} x^2 - 4x, \text{если } x \geq 4 \\ \sqrt{x - 1.5}, \text{если } 2 \leq x < 4 \\ \frac{3x}{x+1}, \text{если } x < 2\ . \end{cases}$$

Примечание. Пример этой задачи взят из раздела 1.2 (рисунок 1.4).

3. Определение входных, выходных и временных данных. Входными данными является только значение аргумента функции **x**. Выходными данными является только значение функции **y**. Временные данные для решения этой задачи не нужны.

4. Разработка словесного и графического описания алгоритма решения задачи.

Словесное описание представим последовательностью шагов:

Шаг 1. Начать работу алгоритма.

Шаг 2. Ввод переменной **x**.

Шаг 3. Проверить, если **x** больше четырех (**x** > = 4), то перейти к шагу 4, в противном случае перейти к шагу 5.

Шаг 4. **y:= x^2 – 4x** (вычислить значение функции по 1-й формуле), далее перейти к шагу 10.

Шаг 5. Проверить, если **x < 2**, то перейти к шагу 6, а если нет, то перейти к шагу 7.

Шаг 6. Проверить, если **(x + 1) ≠ 0**, то перейти к шагу 8, в противном случае – к шагу 9.

Шаг 7. **y:= sqrt(x – 1,5)** (вычислить значение функции по 2-й формуле), далее перейти к шагу 10, где **sqrt –** это имя функции извлечения квадратного корня.

Шаг 8. **y:= 3x / (x + 1)** (вычислить значение функции по 3-й формуле), далее перейти к шагу 10.

Шаг 9. Вывод «Деление на ноль», далее перейти к шагу 11.

Шаг 10. Вывод значения функции **y**.

Шаг 11. Завершить работу алгоритма.

Графическое описание алгоритма представлено на рисунке 1.5.

5. Тестирование графического описания алгоритма двумя способами: последовательный метод, трассировочная таблица.

Словесное описание алгоритма, описанного выше, носит избыточный характер (все разветвляющиеся алгоритмы носят избыточный характер), то есть в зависимости от введенного значения аргумента вычислительный процесс может развиваться по 4-м направлениям, которые все необходимо протестировать. Опишем последовательный метод тестирования алгоритма, представленного на рисунке выше, только по 2-м направлениям.

Направление 1.

Шаг 1. Начать работу алгоритма.

Шаг 2. Ввод 7.

Шаг 3. 7 > = 4, да.

Шаг 4. y:= 7 * 7 – 4 * 7.

Шаг 5. Вывод 21.

Шаг 6. Завершить работу алгоритма.

Вывод: **y** получился равным 21, проверив результат альтернативным способом (в уме, вручную, на калькуляторе, в электронной таблице), получаем также 21. Значит это направление (эта ветка) алгоритма работает верно.

Направление 2.

Шаг 1. Начать работу алгоритма.

Шаг 2. Ввод -1.

Шаг 3. -1 > = 4, нет.

Шаг 4. -1 < 2, да.

Шаг 5. (-1 + 1) ≠ 0, нет.

Шаг 6. Вывод «Деление на ноль».

Шаг 7. Завершить работу алгоритма.

Протестируем этот алгоритм с использование трассировочных таблиц по оставшимся 2-м направлениям работы (таблица 6.5–6.6).

Таблица 6.5 – Трассировочная таблица тестирования алгоритма (3-е направление)

Номер шага	Команда алгоритма	Значение переменных		Выполняемое действие
		x	y	
1.	Начало			Запустить алгоритм
2.	Ввод x	2		Ввод 2
3.	x > = 4			2 > = 4, нет
4.	x < 2			2 < 2, нет
5.	y:= sqrt (x – 1,5)		0,71	y:= sqrt (2 – 1,5)
6.	Вывод y			Вывод 0,71
7.	Конец			Остановить алгоритм

Таблица 6.6 – Трассировочная таблица тестирования алгоритма (4-е направление)

Номер шага	Команда алгоритма	Значение переменных		Выполняемое действие
		x	y	
1.	Начало			Запустить алгоритм
2.	Ввод x	1		Ввод 1
3.	x > = 4			1 > = 4, нет
4.	x < 2			1 < 2, да
5.	(x + 1) ≠ 0			(1 + 1) ≠ 0, да
6.	y:= 3x / (x + 1)		1,5	y:= 3 * 1 / (1 + 1)
7.	Вывод y			Вывод 1,5
8.	Конец			Остановить алгоритм

6. Разработка описания алгоритма решения задачи на языке программирования высокого уровня (C#) – программного приложения.

6.1 Выбор, описание идентификаторов программного приложения и внутреннее представление входных, выходных и временных данных.

Для наглядности и краткости в таблице 6.7 приведем выбранные идентификаторы, их описание и их внутреннее.

Таблица 6.7 – Характеристики идентификаторов программного приложения

Наименование идентификатора	Описание	Название типа данных	Ключевое слово	Диапазон значений	Размер, байт	Класс типа		Назначение данных		
						Простой	Структурированный	Входные	Выходные	Временные
x	Аргумент	Вещественный	double	От $5.0 \cdot 10^{-324}$ до $1.7 \cdot 10^{308}$	64	+		+		
y	Функция	Вещественный	double	От $5.0 \cdot 10^{-324}$ до $1.7 \cdot 10^{308}$	64	+			+	

6.2 Разработка программного приложения и описание его работы.

Разработанное программное приложение на языке C# для вычисления функции представлено на рисунке 6.9.

```
1.      using System;
2.      using System.Collections.Generic;
3.      using System.Linq;
4.      using System.Text;
5.      namespace РазветвляющиесяПрограммы1
6.      {
7.          class Program
8.          {
9.              static void Main(string[] args)
10.             {
11.                 // Описание переменных
12.                 // Внутренне представление входных данных
13.                 double x;
14.                 // Внутренне представление выходных данных
15.                 double y=0;
16.                 // Временных данных эта программа не использует
17.                 // Определение переменных
18.                 Console.Write("Введите значение аргумента x: ");   // Вывод на консоль
19.                 x = int.Parse(Console.ReadLine());                                  // Ввод аргумента x
20.                 if (x >= 4) y = x * x - 4 * x;
21.                 else if (x < 2) if ((x + 1) != 0) y = 3 * x / (x + 1);
22.                                 else  Console.WriteLine("Деление на ноль");
23.                     else y = Math.Sqrt(x-1.5);
24.                 Console.WriteLine("y=" + y);
25.                 Console.ReadLine();
26.             }
27.         }
28.     }
```

Рисунок 6.9 – Код программы вычисления функции

К описанию кода программы на рисунке 6.9, по сравнению с описанием кода программы на рисунке 6.4, можно добавить следующее:

1. В строке 15 инициализируется (начальное определение) переменная **y**, хотя она определяется операторами присваивания в строках 20, 21 и 23 (в других языках програм-

мирования этого можно и не делать). Это связано с тем, что оператор вывода (строка 24) работать не будет, так как он находится в другом блоке по отношению к блокам оператора ***if*** (оператор ***if*** состоит из 2-х блоков – блока ***if*** и блока ***else***). Переменные в C# по умолчанию определены только в пределах одного блока (блок определяется операторами, заключенными в фигурные скобки, или другим контекстом – операторами, методами, классами). А для того чтобы не использовать строку 15, переменную **y** необходимо сделать глобальной, добавив строку с ключевым словом **public** между строками 6 и 7 (рисунок 6.10).

2. Строки с 20 по 23 представляют собой три оператора условного перехода ***if*** в полной форме каждый (подраздел 2.6.2), которые описывают блоки графического описания алгоритма (рисунок 1.5) со 2-го по 8-й, то есть одной вершине алгоритма в виде ромба соответствует один оператор ***if*** (три ромба – три оператора ***if***).

```
5.      namespace РазветвляющиесяПрограммы1
6.      {
            public static double y;
7.          class Program
8.          {
9.              static void Main(string[] args)
10.             {
11.                 // Описание переменных
12.                 // Внутренне представление входных данных
13.                 double x;
14.                 // Внутренне представление выходных данных
15.
16.                 // Временных данных эта программа не использует
```

Рисунок 6.10 – Фрагмент кода программы вычисления функции

6.2 Тестирование программного приложения и описание тестовых случаев.

После запуска программы на выполнение на экране монитора появится сообщение, изображенное на рисунке 6.11.

Рисунок 6.11 – Окно после запуска программы, показанной на рисунке 6.9

После ввода значения аргумента, равного 7, получим результат, изображенный на рисунке 6.12.

Рисунок 6.12 – Окно с результатом работы программы, показанной на рисунке 6.9

Далее программу необходимо запустить несколько раз (в нашем случае достаточно еще 3), вводя другие значения аргумента и получая другие значения функции (другие значения аргументов можно взять из пункта 5).

Таблица 6.8 – Варианты заданий 2-й лабораторной работы (задание 1)

Вариант	Вид функции
1	**2**
1	$y=\begin{cases}1/x, & \text{если } x\ge -5,\ x\ne 0 \quad (1)\\ x^2, & \text{если } x\le -10 \quad (2)\\ \sqrt{\lvert x+1\rvert} & \text{в ост. случаях} \quad (3)\end{cases}$
2	$y=\begin{cases}x^2, & \text{если } x\le 0,\ x\ne -10 \quad (1)\\ \sqrt{x+1}, & \text{если } x>1 \quad (2)\\ 1/x & \text{в ост. случаях} \quad (3)\end{cases}$
3	$y=\begin{cases}x+e^{2x}, & \text{если } x\le 0,\quad x\ne -1 \quad (1)\\ \cos^2 x, & \text{если } 0<x\le 3{,}14 \quad (2)\\ x & \text{в ост. случаях} \quad (3)\end{cases}$
4	$y=\begin{cases}x^3, & \text{если } x>1,\quad x\ne 20 \quad (1)\\ x^2, & \text{если } -5\le x\le 5 \quad (2)\\ \lg\lvert x\rvert & \text{в ост. случаях} \quad (3)\end{cases}$
5	$y=\begin{cases}\sqrt{x}, & \text{если } x\ge 100,\ x\ne 105 \quad (1)\\ \sqrt[3]{x}, & \text{если } x=20 \text{ или } x=40 \quad (2)\\ x^2+1 & \text{в ост. случаях} \quad (3)\end{cases}$
6	$y=\begin{cases}\sqrt[3]{x}, & \text{если } x>2 \quad (1)\\ 1/x, & \text{если } x\le 2 \text{ и } x\ne 0 \quad (2)\\ x^2-1 & \text{в ост. случаях} \quad (3)\end{cases}$
7	$y=\begin{cases}8x+1, & \text{если } x\ge 5,\ x\ne 9 \quad (1)\\ x^2+\lvert x\rvert, & \text{если } x\le 1 \quad (2)\\ x^3+\sqrt{x} & \text{в ост. случаях} \quad (3)\end{cases}$
8	$y=\begin{cases}1-3x, & \text{если } x>0,\ x\ne 8 \quad (1)\\ x^2-\sin x, & \text{если } x\le 1 \quad (2)\\ \cos x & \text{в ост. случаях} \quad (3)\end{cases}$
9	$y=\begin{cases}x^3+1, & \text{если } x\ge 8,\ x\ne 10 \quad (1)\\ 2x^2+\sqrt[3]{x}, & \text{если } x\le 1 \quad (2)\\ \sqrt{x} & \text{в ост. случаях} \quad (3)\end{cases}$

Продолжение таблицы 6.8

1	2
10	$y=\begin{cases}\sqrt{x}, & \text{если } x\ge 4 \quad (1)\\ 2x+3, & \text{если } x\le 1 \quad (2)\\ \lvert x^3-4\rvert & \text{в ост. случаях} \quad (3)\end{cases}$
11	$y=\begin{cases}\lg^2 2x, & \text{если } x\ge 5 \quad (1)\\ 2x^2, & \text{если } x<-2 \quad (2)\\ \sin x & \text{в ост. случаях} \quad (3)\end{cases}$
12	$y=\begin{cases}\lg^2 2x, & \text{если } x\ge 7 \quad (1)\\ 2x^2, & \text{если } x<-5 \quad (2)\\ \sin x & \text{в ост. случаях} \quad (3)\end{cases}$
13	$y=\begin{cases}x/3, & \text{если } -3\le x\le 3 \quad (1)\\ \lg(x^2+1), & \text{если } x<-3 \quad (2)\\ \sqrt{x^3-2} & \text{в ост. случаях} \quad (3)\end{cases}$
14	$y=\begin{cases}\lvert x^3+4\rvert, & \text{если } x\le -1 \text{ или } x=0 \quad (1)\\ \sqrt{x/2}, & \text{если } x\ge 8 \quad (2)\\ x^3 & \text{в ост. случаях} \quad (3)\end{cases}$
15	$y=\begin{cases}\sqrt{3x^2+4}, & \text{если } x\ge 2 \quad (1)\\ \ln\lvert x-2\rvert, & \text{если } x<0 \quad (2)\\ \cos x & \text{в ост. случаях} \quad (3)\end{cases}$
16	$y=\begin{cases}tg\ x/2, & \text{если } 0<x\le 2 \quad (1)\\ x^2+1, & \text{если } x\le 0 \quad (2)\\ \cos^2 x & \text{в ост. случаях} \quad (3)\end{cases}$
17	$y=\begin{cases}\sqrt{x^2-2x}, & \text{если } x\ge 10 \quad (1)\\ e^{x/2}, & \text{если } x\le 1 \quad (2)\\ \ln x+x^3/4 & \text{в ост. случаях} \quad (3)\end{cases}$
18	$y=\begin{cases}e^{2x}, & \text{если } x\le 0 \quad (1)\\ \sqrt{\lvert x^2-2\rvert}, & \text{если } 0<x<7 \quad (2)\\ x/2-x^2 & \text{в ост. случаях} \quad (3)\end{cases}$
19	$y=\begin{cases}\sqrt{e^{2x}}, & \text{если } x\ge 0 \quad (1)\\ \cos x/3, & \text{если } x<-1 \quad (2)\\ \lvert x+1\rvert & \text{в ост. случаях} \quad (3)\end{cases}$

Продолжение таблицы 6.8

1	2
20	$y=\begin{cases} x/3+x^2, & \text{если } 0\le x\le 3 \quad (1)\\ \lvert x\rvert+3, & \text{если } x<0 \quad (2)\\ \sqrt{2x} & \text{в ост. случая} \quad (3)\end{cases}$
21	$y=\begin{cases} \sqrt{x+1}, & \text{если } x\ge 8,\ x\ne 10 \quad (1)\\ 0{,}6x, & \text{если } 0<x<8 \quad (2)\\ \lg\lvert x\rvert+3 & \text{в ост. случаях} \quad (3)\end{cases}$
22	$y=\begin{cases} 2x^2, & \text{если } x>0, x\ne 3 \quad (1)\\ \sqrt{x^2+1}, & \text{если } x\le -2 \quad (2)\\ \lvert x+5\rvert & \text{в ост. случаях} \quad (3)\end{cases}$
23	$y=\begin{cases} \sqrt{x-1}, & \text{если } x\ge 10,\ x\ne 20 \quad (1)\\ 1/x+e^{2x}, & \text{если } x<0 \quad (2)\\ \ln(x+1) & \text{в ост. случаях} \quad (3)\end{cases}$
24	$y=\begin{cases} \sqrt{\lvert 2x-x^2-1\rvert}, & \text{если } x\le -1,\ x\ne -4 \quad (1)\\ \ln(x+3), & \text{если } x>0 \quad (2)\\ x/2 & \text{в ост. случаях} \quad (3)\end{cases}$
25	$y=\begin{cases} \cos^2 x/2, & \text{если } x>3 \quad (1)\\ \lg\lvert 2x+4\rvert, & \text{если } -2{,}5\le x\le 3 \quad (2)\\ 3/x & \text{в ост. случаях} \quad (3)\end{cases}$
26	$y=\begin{cases} e^{-x}+1, & \text{если } x\ge 1 \quad (1)\\ \lg 2x, & \text{если } 1<x\le 5 \quad (2)\\ x^2 & \text{в ост. случаях} \quad (3)\end{cases}$
27	$y=\begin{cases} \lg^2 x/2, & \text{если } x>0,\ x\ne 2 \quad (1)\\ 2e^{x+1}, & \text{если } x\le -1 \quad (2)\\ \sqrt{5+x^2} & \text{в ост. случаях} \quad (3)\end{cases}$
28	$y=\begin{cases} x^3/2, & \text{если } x>0,\ x\ne 2 \quad (1)\\ 2e^{x+1}, & \text{если } x\le -1 \quad (2)\\ \sqrt{5+x^2} & \text{в ост. случаях} \quad (3)\end{cases}$
29	$y=\begin{cases} \sqrt{x+2}, & \text{если } x\ge 7 \text{ или } x=-1 \quad (1)\\ x-1, & \text{если } 0<x<7 \quad (2)\\ \lvert x^2+4\rvert & \text{в ост. случаях} \quad (3)\end{cases}$

1	2
30	$y = \begin{cases} \sqrt{\lg^2 x + 1}, & \text{если } x \geq 1 \quad (1) \\ 1/x + e^x, & \text{если } x \leq -1 \quad (2) \\ 0{,}5x^2 & \text{в ост. случаях} \quad (3) \end{cases}$

Таблица 6.9 – Варианты заданий 2-й лабораторной работы (задание 2)

Вариант	Задание
1	2
1	Определить, в какой четверти или на какой оси координатной плоскости находится точка с координатами *x*, *y*
2	Даны два числа, не равных друг другу. Меньшее из них заменить их полусуммой, большее – их удвоенным произведением
3	Даны три целых положительных числа. Если все они четные, каждое число уменьшить в два раза, если хотя бы одно из них четное, увеличить каждое число на 20 %, если четных чисел нет, оставить числа без изменения
4	Даны три целых числа. Найти минимальное из них и прибавить минимальное значение к числам, отличным от минимального
5	Даны три целых числа. Определить, могут ли они быть сторонами треугольника. Если могут, то определить, какой это треугольник: равнобедренный, равносторонний или разносторонний
6	Даны два угла (в градусах). Определить, существует ли треугольник с такими углами. Если существует, то будет ли он прямоугольным
7	Даны три числа. Если одно из них положительное, то найти площадь квадрата со стороной, равной значению положительного числа. В противном случае вывести соответствующее сообщение
8	Даны числа a, b. Если b = 0, то найти min (a, b), если b < 0, то найти max (a, b), в противном случае каждое число уменьшить на 20 %
9	Даны два целых числа, не равных друг другу. Большее из них увеличить на 50 %, меньшее заменить суммой заданных чисел
10	Даны числа a, b, c, d. Найти min {max(a, b), max(c, d)}
11	Даны два целых числа a, b. Найти вещественные корни уравнения $ax^2 + b = 0$ или вывести сообщение об их отсутствии
12	Даны числа a, b, c, d. Если a > b > c > d, то каждое число заменить наибольшим из всех чисел, если a < b < c < d, то каждое число заменить его квадратом, в противном случае оставить числа без изменения
13	Даны числа a, b, c. Если все они равны нулю, вывести об этом сообщение, если среди чисел нет нулей, найти и вывести их произведение, в противном случае нули заменить суммой двух других чисел
14	Даны числа a, b, c. Вычислить max (a + b + c, abc) * min (a, b, c)
15	Определить, где находится точка с координатами x, y: на окружности радиуса r, внутри круга радиуса r или вне его
16	Даны числа a, b, c, d. Если ни одно из чисел a, b, c не равно d, то найти max (d – a, d – b, d – c)
17	Даны три целых числа a, b, c. Найти вещественные корни уравнения ax2 + bx + c = 0 или вывести сообщение об их отсутствии
18	Определить правильность даты, заданной тремя целыми числами (день, месяц, год)

Окончание таблицы 6.9

1	2
19	Даны три числа. Найти сумму минимального и максимального среди них
20	Даны числа a, b, c, d. Найти max {min(a, b), min(c, d)}
21	Даны три целых положительных числа. Если все они нечетные, каждое число увеличить в два раза, если хотя бы одно из них нечетное, оставить числа без изменения, если нечетных чисел нет, увеличить каждое число на 50 %
22	Даны два числа, не равных друг другу. Большее из них уменьшить на 30 %, меньшее заменить произведением заданных чисел
23	Даны три числа. Если все они отрицательные, каждое число увеличить на 40 %, если хотя бы одно из них отрицательное, уменьшить каждое число в два раза, если отрицательных чисел нет, оставить их без изменения
24	Даны три числа a, b, c. Если все они положительные, вычислить площадь треугольника со сторонами a, b и c. Если среди них есть хотя бы одно отрицательное число, найти сумму чисел
25	Даны числа a, b, c. Вычислить min(a + b + c, abc) * max (a, b, c)
26	Даны числа a, b, c, d. Если a < b < c < d, то каждое число заменить наименьшим, если a > b > c > d, то каждое число уменьшить на 40 %, в противном случае оставить числа без изменения
27	Даны числа a, b. Если a < 0, то найти max (a, b), если a = 0, то найти min (a, b), в противном случае каждое число увеличить на 50 %
28	Даны числа a, b, c. Если среди них нет положительных чисел, вывести об этом сообщение, если все они положительные, найти и вывести их сумму, в противном случае положительные числа уменьшить на 20 %
29	Даны числа a, b, c, d. Если ни одно из чисел a, b, c не равно d, то найти min(a – d, b – d, c – d)
30	Даны числа a, b, c. Найти произведение минимального и максимального из них

Контрольные вопросы и задания к защите:

1. Какой алгоритм называется разветвляющимся?
2. Как создаются тесты для разветвляющихся алгоритмов?
3. Какие операторы используются для описания разветвляющихся алгоритмов?
4. Какие существуют способы тестирования?
5. Структура оператора ***if.***
6. Как работает оператор ***if?***
7. Структура оператора ***switch.***
8. Как работает оператор ***switch?***
9. Для чего используется и как записывается логическое выражение?
10. Какой тип и значение имеют логические выражения?
11. Назовите операции отношения и для чего они используются?
12. Назовите логические операции.
13. От чего зависит результат выполнения логических операций?

6.3 Лабораторная работа 3
Циклические алгоритмы: табулирование функций

В этой лабораторной работе необходимо разработать циклические алгоритмы решения задачи согласно вариантам (таблица 6.12–6.13) и описать их тремя способами – словесное описание, графическое и на языке программирования высокого уровня. В описании ме-

тодики выполнения лабораторной работы необходимо придерживаться структуры ОГЛАВЛЕНИЯ, предложенной в главе 4.

Опишем методику выполнения одной из задач.

1. Постановка задачи. Разработать циклический алгоритм решения задачи табулирования функции. В соответствии с видом функции, приведенном в таблицах 6.12–6.13, вычислить значения функции ***y = f(x,a,b)*** для значений аргумента x, изменяющегося в интервале от $x_{нач}$ до $x_{кон}$ с шагом Δx, и заданных коэффициентов a и b ($x_{нач} < x_{кон}$). Исходные данные для проверки алгоритма ($x_{нач}$, $x_{кон}$, Δx, a, b) выбрать самостоятельно из интервала значений, где заданные функции определены (таблица 6.12–6.13).

2. Математическое описание решения задачи. В качестве примера возьмем квадратную функцию $y = ax^2 – b$.

3. Определение входных, выходных и временных данных. Входными данными являются $x_{нач}$, $x_{кон}$, Δx, a, b. Выходными данными являются значения аргумента **x** и соответствующие значения функции **y**. Временные данные для решения этой задачи не нужны.

4. Разработка словесного и графического описания алгоритма решения задачи.

Словесное описание представим последовательностью шагов:

Шаг 1. Начать работу алгоритма.

Шаг 2. Ввод $x_{нач}$, $x_{кон}$, Δx, a, b.

Шаг 3. x:= $x_{нач}$ (аргументу присвоить начальное значение).

Шаг 4. $y = ax^2 – b$ (вычислить значение функции).

Шаг 5. Вывод x и y.

Шаг 6. x:= x + Δx (аргумент x увеличить на шаг).

Шаг 7. Проверить, если x<= $x_{кон}$, то перейти к шагу 4, в противном случае – к шагу 8.

Шаг 8. Завершить работу алгоритма.

Разработанное графическое описание алгоритма представим на рисунке 6.13.

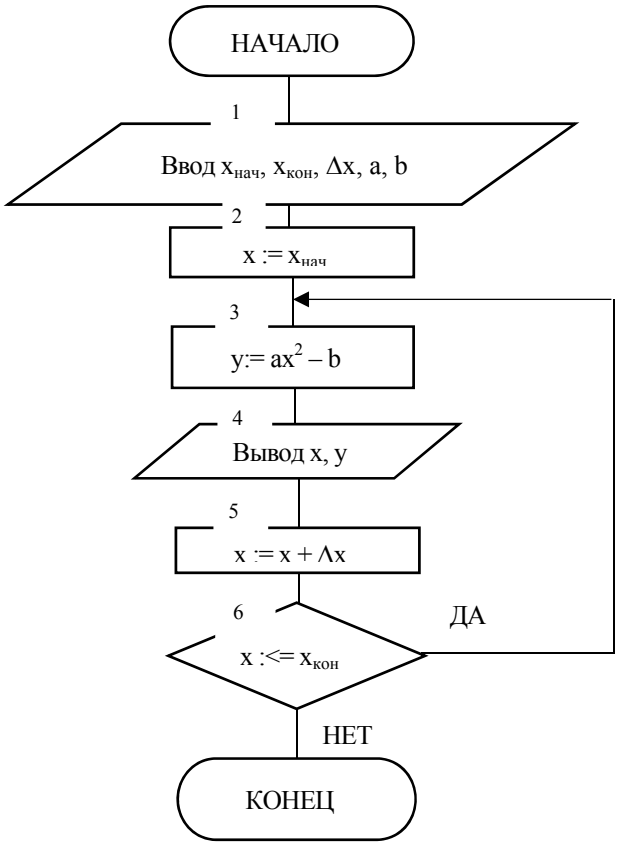


Рисунок 6.13 – Граф-схема табулирования функции $y = ax^2 – b$

5. Тестирование графического описания алгоритма двумя способами: последовательный метод, трассировочная таблица.

Опишем последовательный метод тестирования алгоритма, представленного на рисунке 6.13. Так как этот алгоритм является циклическим, необходимо сократить объем тестирования. Поэтому для тестирования возьмем следующие входные данные: $x_{нач}$ = -1, $x_{кон}$ = 1, Δx = 1, a = 1, b = 3.

Шаг 1. Начать работу алгоритма.

Шаг 2. Ввод –1, 1, 1, 1, 3.

Шаг 3. x:= –1.

Шаг 4. y:= 1 * (–1) * (–1) –3.

Шаг 5. Вывод –1, –2.

Шаг 6. x:= –1 + 1.

Шаг 7. 0<=1, да.

Шаг 8. y:= 1 * (0) * (0) –3.

Шаг 9. Вывод 0, –3.

Шаг 10. x:= 0 + 1.

Шаг 11. 1<=1, да.

Шаг 12. y:= 1 * (1) * (1) –3.

Шаг 13. Вывод 1, –2.

Шаг 14. x:= 1 + 1.

Шаг 15. 2<=1, нет.

Шаг 16. Завершить работу алгоритма.

Вывод: в результате работы алгоритма при указанных входных данных будет выведено 3 значения аргумента и 3 значения функции. Полученный результат можно проверить другим альтернативным способом.

Протестируем этот алгоритм с использование трассировочной таблицы 6.10.

Таблица 6.10 – Трассировочная таблица тестирования алгоритма табулирования функции $y = ax^2 – b$

Номер шага	Команда алгоритма	Значение переменных							Выполняемое действие
		$x_{нач}$	$x_{кон}$	Δx	a	b	**X**	**Y**	
	Начало								Запустить алгоритм
1	Ввод $x_{нач}$, $x_{кон}$, Δx, a, b	–1	1	1	1	3			Ввод –1, 1, 1, 1, 3
2	x := $x_{нач}$						–1		x := –1
3	y := $ax^2 – b$							–2	y := 1 * (–1) *(–1)–3
4	Вывод x, y						–1	–2	Вывод –1, –2
5	x := x + Δx						0		x := –1 + 1
6	x<= $x_{кон}$								0<=1, да
7	y := $ax^2 – b$							–3	y := 1 * (0) * (01) –3
8	Вывод x, y						0	–2	Вывод 0 , –3
9	x := x + Δx						1		x := 0 + 1
10	x<= $x_{кон}$								1<=1, да
11	y := $ax^2 – b$							–2	y := 1 * (1) * (1) –3
12	Вывод x, y						1	–2	Вывод 1, –2
13	x := x + Δx						2		x := 1 + 1
14	x<= $x_{кон}$								2<=1, нет
15	Конец								Остановить алгоритм

6. Разработка описания алгоритма решения задачи на языке программирования высокого уровня (C#) – программного приложения.

6.1 Выбор, описание идентификаторов программного приложения и внутреннее представление входных, выходных и временных данных.

Для наглядности и краткости приведем в таблице 6.11 выбранные идентификаторы, их описание и их внутреннее.

Таблица 6.11 – Характеристики идентификаторов программного приложения

Наименование идентификатора	Описание	Название типа данных	Ключевое слово	Диапазон значений	Размер, байт	Класс типа		Назначение данных		
						Простой	Структурированный	Входные	Выходные	Временные
XN	Начальное значение интервала табулирования	Вещественный	double	От $5.0 \cdot 10^{-324}$ до $1.7 \cdot 10^{308}$	64	+		+		
XK	Конечное значение интервала табулирования	Вещественный	double	От $5.0 \cdot 10^{-324}$ до $1.7 \cdot 10^{308}$	64	+		+		
H	Шаг табулирования	Вещественный	double	От $5.0 \cdot 10^{-324}$ до $1.7 \cdot 10^{308}$	64	+		+		
a	Коэффициент	Целый	int	От $-2 \cdot 10^{9}$ до $2 \cdot 10^{9}$	32	+		+		
b	Коэффициент	Целый	int	От $-2 \cdot 10^{9}$ до $2 \cdot 10^{9}$	32	+		+		
x	Аргумент	Вещественный	double	От $5.0 \cdot 10^{-324}$ до $1.7 \cdot 10^{308}$	64	+			+	
y	Функция	Вещественный	double	От $5.0 \cdot 10^{-324}$ до $1.7 \cdot 10^{308}$	64	+			+	

6.2 Разработка программного приложения и описание его работы.

Алгоритм на рисунке 6.13 является циклическим, и для его описания на языке программирования C# можно использовать оператор передачи управления ***goto*** и 3 оператора цикла: ***while, do..while, for***. Поэтому для более полного и глубокого понимания возможностей языка программирования рассмотрим программные коды этого алгоритма с использованием всех перечисленных операторов.

Разработанное программное приложение на языке C# для табулирования функции с использованием оператора ***goto*** представлено на рисунке 6.14.

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   namespace Циклические_программы_1_табулирование
6   {
7       class Program
8       {
9           static void Main(string[] args)
10          {
11              // Описание и определение переменных
12              Console.Write("Введите начальное значение интервала табулирования XN: ");   // Вывод текста на консоль
```

```
13        // Считывание с консоли символьного представления числа, его преобразование и присваивание переменной XN
14            double XN = Convert.ToDouble(Console.ReadLine());     // Описание и определение входной переменной XN
15            Console.Write("Введите конечное значение интервала табулирования XK: ");
16            double XK = Convert.ToDouble(Console.ReadLine());     // Описание и определение входной переменной XK
17            Console.Write("Введите шаг табулирования H: ");       // Описание и определение входной переменной H
18            double H = Convert.ToDouble(Console.ReadLine());
19            Console.WriteLine("Введите значения коэффициентов a и b: ");
20            double a = Convert.ToDouble(Console.ReadLine());      // Описание и определение входной переменной a
21            double b = Convert.ToDouble(Console.ReadLine());      // Описание и определение входной переменной b
22            double x = XN;                                        // Присваивание переменной x начального значения
23            // Начало цикла
24            cycle: double y = Math.Round(a*Math.Pow(x,2)-b, 2);   // Вычисление значения функции
25            Console.WriteLine("x=" + x + " y=" + y);              // Вывод на консоль переменных x и y
26            x += H;                                               // Изменение переменной x на размер шага
27            if (x <= XK)                          // Сравнение переменной x с конечным значение интервала табулирования
28               goto cycle;                                        // Конец цикла
29            Console.ReadLine();
30          }
31      }
32  }
```

Рисунок 6.14 – Код программы табулирования функции $y = ax^2 - b$ с оператором *goto*

Опишем те строки программного кода, которые еще не рассматривались. В строках 14, 16, 18, 20 и 21 используется метод ***Convert.ToDouble***, который преобразует заданное строковое представление числа в эквивалентное число с плавающей запятой двойной точности. Строки с 25 по 30 – это операторы, которые выполняются циклически или операторы цикла. В первом операторе цикла (строка 25) вычисляется значение функции, и этот оператор помечен меткой **cycle**. В этом операторе используется два метода ***Math.Round*** (округляет количество знаков после запятой до 2) и ***Math.Pow*** (возводит аргумент в степень, равную 2). Строка 27 – это сокращенная форма оператора присваивания x = x + H. В строках 28 и 29 используется оператор ***if*** в неполной форме, который работает следующим образом: если условие истинно, то выполняется оператор ***goto***, который по метке передает управление оператору в строке 25 (то есть организуется цикл); а если условие ложно, то управление передается оператору в строке 30. Использование оператора ***goto*** считается плохим стилем программирования, поэтому его необходимо использовать только в безвыходных ситуациях.

Часть программного кода на языке C# для табулирования функции с использованием оператора ***while*** представлена на рисунке 6.15.

```
23          // Начало цикла
24          while (x <= XK)
25                {
26                double y = Math.Round(a * Math.Pow(x, 2) - b, 2);    // Вычисление значения функции
27                Console.WriteLine("x=" + x + " y=" + y);             // Вывод на консоль переменных x и y
28                x += H;                                              // Изменение переменной x на размер шага
29                 }                                                   //  Конец цикла
30          Console.ReadLine();
```

Рисунок 6.15 – Часть программного кода табулирования функции $y = ax^2 - b$ с оператором *while*

Строки с 23 по 30 на рисунке 6.15 заменяют строки кода с 23 по 29 на рисунке 6.14. Обратите внимание, что строки кода с 26 по 28 заключены в фигурные скобки и составляют тело цикла оператора ***while***. Правила работы оператора ***while*** приведены в подразделе 2.6.4.

Часть программного кода на языке C# для табулирования функции с использованием оператора ***do..while*** представлена на рисунке 6.16.

```
23          // Начало цикла
24          do
25              {
26                  double y = Math.Round(a * Math.Pow(x, 2) - b, 2);    // Вычисление значения функции
27                  Console.WriteLine("x=" + x + " y=" + y);             // Вывод на консоль переменных x и y
28                  x += H;                                              // Изменение переменной x на размер шага
29                  }
30          while (x <= XK)                                              // Конец цикла
31          Console.ReadLine();
```

Рисунок 6.16 – Часть программного кода табулирования функции $y = ax^2 – b$ с оператором *do..while*

Строки кода с 23 по 31 на рисунке 6.16 заменяют строки кода с 23 по 30 на рисунке 6.15. Строки кода с 26 по 28 составляют тело цикла оператора ***do..while***. Правила работы оператора ***do..while*** приведены в подразделе 2.6.5.

Часть программного кода на языке C# для табулирования функции с использованием оператора ***for*** представлена на рисунке 6.17.

```
22          // Начало цикла
23          for (double x = XN; x <= XK; x += H)                         // Заголовок оператора цикла
24              {
25                  double y = Math.Round(a * Math.Pow(x, 2) - b, 2);    // Вычисление значения функции
26                  Console.WriteLine("x=" + x + " y=" + y);             // Вывод на консоль переменных x и y
27                  }                                                    // Конец цикла
28          Console.ReadLine();
```

Рисунок 6.17 – Часть программного кода табулирования функции $y = ax^2 – b$ с оператором *for*

При использовании оператора цикла ***for*** код программы сокращается на 3 строки: не нужна строка 22 (рисунок 6.14), не нужны строки 28 и 30 (рисунок 6.16), все эти операторы описываются в заголовке цикла (рисунок 6.17). Обратите особое внимание, что заголовок ***for*** описывает три вершины (2, 5 и 6) графического описания алгоритма (рисунок 6.13). Поэтому графические описания алгоритмов необходимо разрабатывать с учетом этой возможности, что придает алгоритму красоту, высокую наглядность, понятность и упрощает написание программного кода.

6.3 Тестирование программного приложения и описание тестовых случаев.

После запуска любой программы с кодами, указанными на рисунках 6.14–6.17, на экране монитора появится сообщение, изображенное на рисунке 6.18.

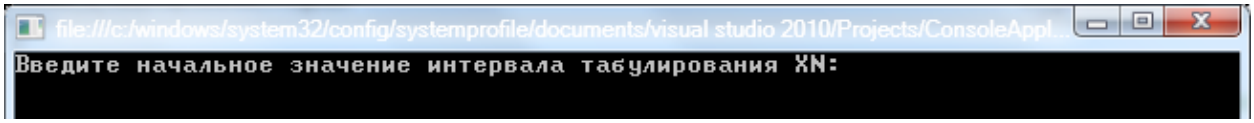


Рисунок 6.18 – Окно после запуска одной из программ, показанных на рисунках 6.14–6.17

После ввода значений XN = –1, XK = 1, H = 1, a = 1, b = 3 получим результат, изображенный на рисунке 6.19.

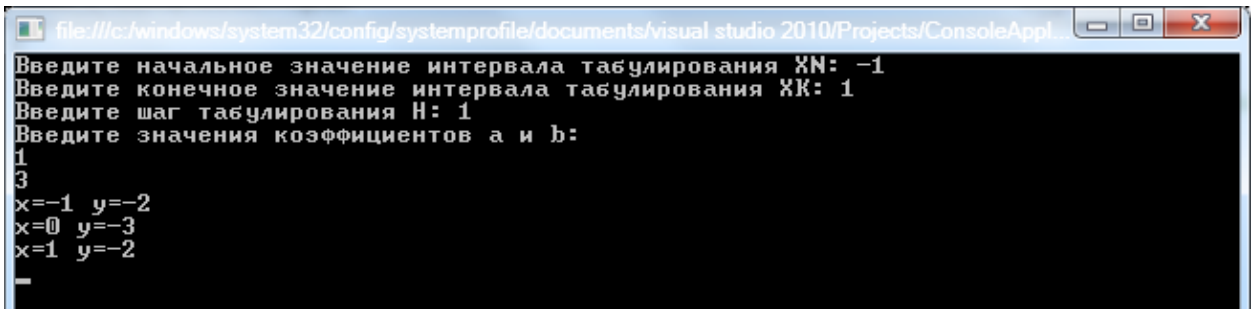


Рисунок 6.19 – Окно с результатом работы программ, показанных на рисунке 6.14–6.17

Далее программу необходимо запустить несколько раз, вводя другие входные данные (границы интервала целые, а шаг – любое вещественное число; все входные данные – вещественные числа). Для качественного и наглядного вывода результатов табулирования необходимо в коде программы использовать форматированный вывод, а также вывод различных строковых литералов или строковых констант (рисунок 6.20). Более детально форматированный вывод описан в подразделе 2.6.9.

```
22      Console.BufferHeight = 60;                                  // Установка высоты консольного окна с добавлением прокрутки
23      // Начало цикла
24      Console.WriteLine("|---------------------|");               // Вывод строковой константы
25      Console.WriteLine("|    x    |    y    |");                 // Вывод строковой константы
26      for (double x = XN; x <= XK; x += H)
27            {
28            double y = Math.Round(a * Math.Pow(x, 2) - b, 2); // Вычисление значения функции
29             Console.WriteLine("|---------------------|");          // Вывод строковой константы
30             Console.WriteLine("|{0,10:f2}|{1,10:f2}|", x, y);    // Форматированный вывод вещественных переменных x и y
31            }
32     Console.WriteLine("|---------------------|");                 // Вывод строковой константы
33     Console.ReadLine();
```

Рисунок 6.20 – Часть программного кода табулирования функции $y = ax^2 – bc$ оператором *for* и форматированным выводом переменных x и y

В строке 22 используется метод ***Console.BufferHeight***, который увеличивает высоту выводимого консольного окна до 60 строк и добавляет справа в окне механизм прокрутки окна. В строках 24, 25, 29 и 30 выводится то, что заключено в двойные кавычки. В строке 30 используется форматированный вывод переменных x и y: символ | – строковая константа; {0,10:f2}: 0 это – это номер первой переменной (x), 10 – число знаков всего для вывода первой переменной, f2: f – вещественное число, 2 – количество цифр после запятой; символ | – строковая константа; {1,10:f2}: 1 это – это номер второй переменной (y), а остальные параметры такие же, как у первой переменной; символ | – строковая константа; x, y – это имена переменных.

После запуска программы с кодом, указанным на рисунке 6.20, на экране монитора появится часть результатов работы (рисунок 6.21). А для просмотра всех результатов необходимо использовать механизм прокрутки экрана. Обратите внимание, что шаг табулирования меньше 1, а вещественное число равно 0,1.

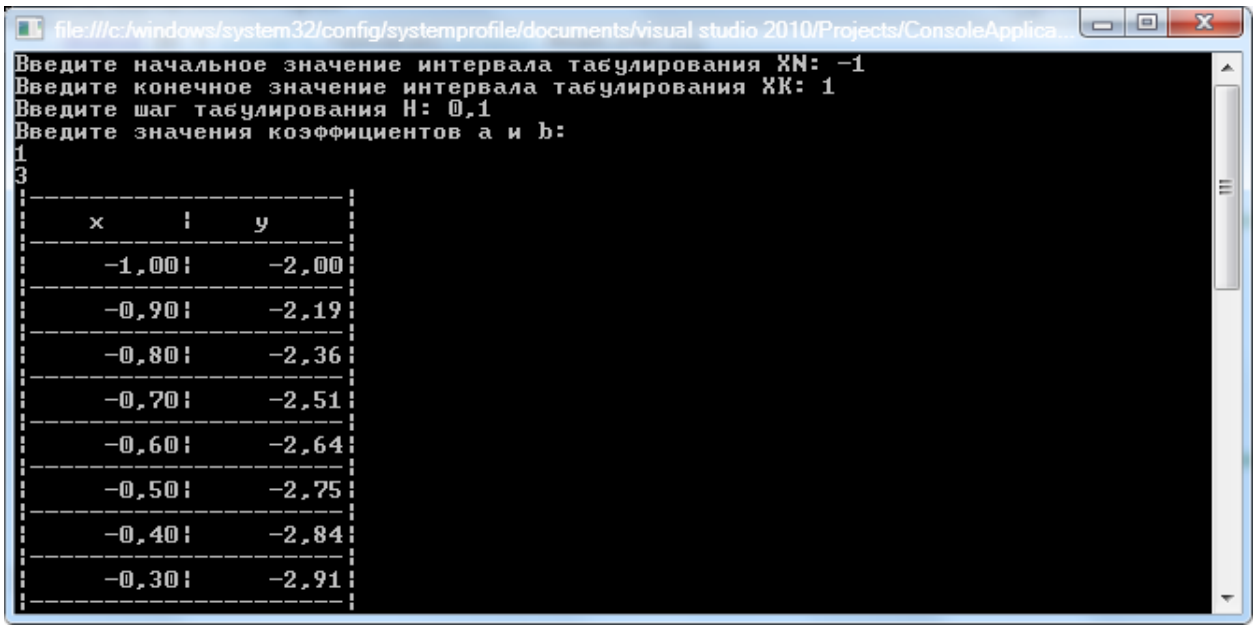


Рисунок 6.21 – Окно с результатами работы программы, показанной на рисунке 6.20

Таблица 6.12 – Варианты заданий 3-й лабораторной работы (задание 1)

Вариант	Вид функции	Вариант	Вид функции
1	2	3	4
1	$y = \frac{arctg\ bx}{1+\sin^2 x}$	16	$y = \frac{arctg\,(a+x)}{\sqrt{a^3+b^3}}$
2	$y = \frac{\sin^2 x + a}{\sqrt{x}+bx}$	17	$y = \frac{1+\sqrt{bx}}{0{,}5+\sin^2 ax}$
3	$y = \sqrt{\frac{a+bx}{\ln^2 x}}$	18	$y = \frac{a - e^{bx}}{\ln\lvert 2x\rvert}$
4	$y = \frac{\ln^2(x-b)}{a\sqrt{x}}$	19	$y = \frac{(a+bx)^2}{1+\cos^3 ax}$
5	$y = \frac{a\ln^2 x}{b+\sqrt{x}}$	20	$y = \frac{b+\sin^2 ax}{e^{-x/2}}$
6	$y = \frac{e^{ax}+b}{1+\cos^2 x}$	21	$y = \frac{\sin^2 x - a}{bx}$
7	$y = \frac{a+\sqrt[3]{x}}{\sin^2 bx}$	22	$y = \frac{arctg^2 ax}{b+0{,}5x}$
8	$y = \frac{a\sqrt{\lvert x\rvert}-bx}{\ln^3 x}$	23	$y = \frac{\ln(a^2-x)}{b\sin^2 x}$
9	$y = \frac{\sqrt{ax}-b}{tg^2 x}$	24	$y = \frac{a-\sqrt{bx}}{1+\cos\lvert 2x\rvert}$
10	$y = e^{-x}\frac{a+bx}{\ln^2\lvert x+1\rvert}$	25	$y = \frac{\ln^2(a+x)}{(b+x)^2}$

Окончание таблицы 6.12

1	2	3	4
11	$y=\frac{tg^2x-b}{e^{ax}}$	26	$y=\frac{\sqrt{\lvert a\ln x\rvert}}{1+tg^2bx}$
12	$y=\frac{arctg\ bx}{1+\sqrt[3]{ax}}$	27	$y=\frac{1+tg^2x}{b+e^{x/a}}$
13	$y=\frac{\sin^3 ax}{ax+b}$	28	$y=\frac{\cos^2 2x+b}{\sqrt{1+e^{ax}}}$
14	$y=\frac{e^{-ab}}{b+\cos^3 ax}$	29	$y=\frac{\sqrt{ax+b}}{\ln^2\lvert x\rvert}$
15	$y=\frac{\ln^2\lvert x\rvert+b}{a\sqrt{x}}$	30	$y=\frac{1+\sin^2 ax}{b^2+x^2}$

Таблица 6.13 – Варианты заданий 3-й лабораторной работы (задание 2)

Вариант	Вид функции	Вариант	Вид функции
1	2	3	4
1	$y=\begin{cases}3x^2, если\ x>10\\3x-1, если\ x\le 2\\\sin x\ в\ остальных\ случаях\end{cases}$	2	$y=\begin{cases}\sqrt[3]{x}, если\ x>2,2\\\frac{1}{x}, если\ -6\le x<0\\x^2-1\ \ в\ остальных\ случаях\end{cases}$
3	$y=\begin{cases}\ln(x^2), если\ x<10\\\sqrt[3]{\lvert\cos(x)\rvert}, если\ x>16\\6\ в\ остальных\ случаях\end{cases}$	4	$y=\begin{cases}\frac{tg\ x}{x-1}, если\ x>3\\\ln(x), если\ 1\le x\le 3\\3x^2\ \ в\ остальных\ случаях\end{cases}$
5	$y=\begin{cases}\sqrt{x+4}, если\ x>16\\\lg x, если\ 1\le x\le 16\\x\ \ в\ остальных\ случаях\end{cases}$	6	$y=\begin{cases}\sqrt{x}, если\ x>20\\\frac{\sin x}{1-x}, если\ 2\le x\le 20\\x-2\ \ в\ остальных\ случаях\end{cases}$
7	$y=\begin{cases}tg\ x, если\ x>9\\\frac{\sin x}{x}, если\ 1\le x\le 9\\x^2\ \ в\ остальных\ случаях\end{cases}$	8	$y=\begin{cases}\frac{\cos x}{2}, если\ -3\le x<3\\\frac{\sin x}{x\sqrt{2}}, если\ x>3\\x^3\ \ в\ остальных\ случаях\end{cases}$
9	$y=\begin{cases}\cos(\sqrt{x}), если\ x>2\\\frac{5}{\lvert x\rvert+1}, если\ -10\le x\le 0\\tg\ x\ \ в\ остальных\ случаях\end{cases}$	10	$y=\begin{cases}\sqrt{x+2}, если\ x>7\\\ln\lvert x\rvert, если\ x<0\\\cos x\ в\ остальных\ случаях\end{cases}$
11	$y=\begin{cases}\sqrt{x}-1,5, если\ x>9\\\frac{1}{x}, если\ 1\le x\le 9\\x^2+2\ \ в\ остальных\ случаях\end{cases}$	12	$y=\begin{cases}\frac{\sqrt{x}}{3}, если\ x>11\\\sqrt[3]{x-1}, если\ 1\le x\le 11\\\cos x\ в\ остальных\ случаях\end{cases}$

Окончание таблицы 6.13

13	$y = \begin{cases} \frac{1}{x}, \text{если } x > 8 \\ \sqrt[3]{x}, \text{если } 1 \le x \le 6 \\ 0,5 \cdot x \text{ в остальных случаях} \end{cases}$	14	$y = \begin{cases} \sqrt{x} + 2, \text{если } 0 \le x \le 3 \\ 3x^2 - 3, \text{если } x \ge 6 \\ 2 \cdot x^{-2} \text{ в остальных случаях} \end{cases}$
15	$y = \begin{cases} \sqrt{x-1}, \text{если } x > 22 \\ \frac{\sin x}{1-x}, \text{если } 3 \le x \le 22 \\ tg\, x \text{ в остальных случаях} \end{cases}$	16	$y = \begin{cases} 0,5 \cdot \cos x, \text{если } -3 \le x \le 3 \\ \frac{\sin x}{x}, \text{если } x > 3 \\ x^3 \text{ в остальных случаях} \end{cases}$
17	$y = \begin{cases} \sqrt{x}, \text{если } x > 15 \\ x^3, \text{если } 1 \le x \le 15 \\ 3x^2 \text{ в остальных случаях} \end{cases}$	18	$y = \begin{cases} 3x^2, \text{если } x > 6 \\ 2x, \text{если } x \le 2 \\ \sin x \text{ в остальных случаях} \end{cases}$
19	$y = \begin{cases} x^3, \text{если } x > 10 \\ x - x^2, \text{если } 1 \le x \\ x^{0,5} \text{ в остальных случаях} \end{cases}$	20	$y = \begin{cases} \sqrt{x+4}, \text{если } x > 5 \\ \lg x, \text{если } 1 \le x \le 5 \\ x \text{ в остальных случаях} \end{cases}$
21	$y = \begin{cases} \sqrt{x} + 25, \text{если } x > 20 \\ -\sin(x^2), \text{если } 1 \le x \le 16 \\ \sin x \text{ в остальных случаях} \end{cases}$	22	$y = \begin{cases} \sqrt{x-4}, \text{если } x > 6 \\ \ln x - \sin x, \text{если } 1 \le x \le 6 \\ x + 2 \text{ в остальных случаях} \end{cases}$
23	$y = \begin{cases} \sqrt{x}, \text{если } x > 3 \\ \sqrt{\lvert x \rvert}, \text{если } -5 \le x \le 3 \\ 2 \cdot x^{-2} \text{ в остальных случаях} \end{cases}$	24	$y = \begin{cases} \sin x, \text{если } x > 5 \\ 4x^2, \text{если } x < 0 \\ \sqrt{x} \text{ в остальных случаях} \end{cases}$
25	$y = \begin{cases} \sqrt{x^3}, \text{если } x > 10 \\ x^{-1}, \text{если } -5 \le x \le -1 \\ \cos x \text{ в остальных случаях} \end{cases}$	26	$y = \begin{cases} \sqrt{x+4}, \text{если } x > 12 \\ \lg x, \text{если } 5 \le x \le 12 \\ x \text{ в остальных случаях} \end{cases}$
27	$y = \begin{cases} 8x, \text{если } x > 6 \\ x^2, \text{если } -5 \le x \le 6 \\ \sin x \text{ в остальных случаях} \end{cases}$	28	$y = \begin{cases} tg\, x, \text{если } x > 5 \\ \frac{\sin x}{x}, \text{если } 1 \le x \le 5 \\ x^2 \text{ в остальных случаях} \end{cases}$
29	$y = \begin{cases} 3x^2, \text{если } x > 10 \\ 3x - 1, \text{если } x \le 2 \\ \sin x \text{ в остальных случаях} \end{cases}$	30	$y = \begin{cases} 0,5 \cdot \cos x, \text{если } -3 \le x \le 3 \\ \frac{\sin x}{x}, \text{если } x > 3 \\ x^3 \text{ в остальных случаях} \end{cases}$

Контрольные вопросы к защите:

1. Какой алгоритм называется циклическим?
2. Что такое табулирование функции?
3. Что такое интервал табулирования?
4. Что такое шаг табулирования?
5. Зачем необходимо табулирование функции?
6. Какие операторы необходимы для описания циклического алгоритма?

6.4 Лабораторная работа 4
Циклические алгоритмы: обработка одномерных массивов

В 4-й лабораторной работе необходимо разработать циклический алгоритм решения задачи обработки одномерного массива согласно варианту (таблица 6.17) и описать его тремя способами – словесное описание, графическое и на языке программирования высокого уровня. В описании методики выполнения лабораторной работы необходимо придерживаться структуры ОГЛАВЛЕНИЯ, предложенной в главе 4.

Опишем методику выполнения одной из задач.

1. Постановка задачи. Дан одномерный массив A(N). Найти сумму и произведение всех элементов массива.

2. Математическое описание решения задачи:

$$S = \sum_{i=1}^{N} A_i \text{ и } P = \prod_{i=1}^{N} A_i,$$

где S – сумма элементов одномерного массива;

Ai – i-й элемент массива;

i – индекс элементов массива;

N – количество элементов массива;

P – произведение элементов одномерного массива, а также графические обозначения суммы и произведения соответственно.

3. Определение входных, выходных и временных данных. Входными данными являются одномерный массив A(N) и количество элементов массива N. Выходными данными являются сумма S и произведение P элементов одномерного массива. Временными данными является индекс элементов одномерного массива i.

4. Разработка словесного и графического описания алгоритма решения задачи.

Словесное описание представим последовательностью шагов.

Шаг 1. Начать работу алгоритма.

Шаг 2. Ввести массив A(N), т. е. все его элементы и его размерность N;

Шаг 3. В ячейку S записать значение 0 (ноль), эта ячейка будет использоваться для накопления суммы элементов массива.

Шаг 4. В ячейку P записать значение 1 (единица), эта ячейка будет использоваться для накопления произведения всех элементов массива.

Шаг 5. В ячейку i записать значение 1, это необходимо для выбора первого элемента массива.

Шаг 6. В ячейку S записать сумму старого значения S плюс значение элемента массива A_i, это будет выглядеть так – S:= S + A_i. И так как S равно нулю и i равно единице, то в ячейку S запишется значение первого элемента массива.

Шаг 7. В ячейку P записать произведение старого значения (P умножить на значение элемента массива A_i), это будет выглядеть так – P:= P * A_i. И так как P равно единице и i равно единице, то в ячейку P запишется значение первого элемента массива.

Шаг 8. В ячейку i записать сумму старого значения i плюс один (это означает переход к следующему элементу массива и будет выглядеть – i:= i + *1*).

Шаг 9. Проверить, если i <= N, то перейти к шагу 6, в противном случае – к шагу 10.

Шаг 10. Вывод S и P.

Шаг 11. Завершить работу алгоритма.

Разработанное графическое описание алгоритма представим на рисунке 6.22.

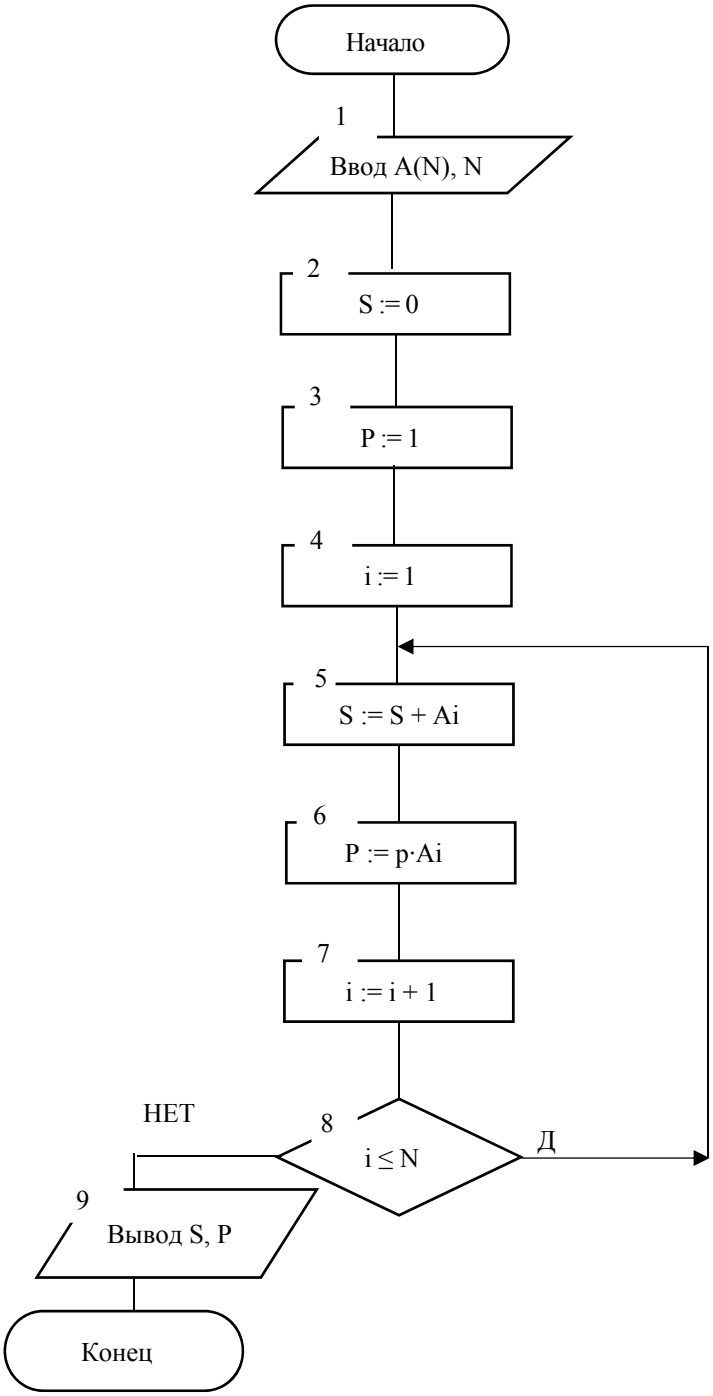


Рисунок 6.22 – Графическое описание алгоритма нахождения суммы и произведения всех элементов массива

5. Тестирование графического описания алгоритма двумя способами: последовательный метод, трассировочная таблица.

Опишем последовательный метод тестирования алгоритма, представленного на рисунке 6.22. Для тестирования возьмем данные из таблицы 6.14.

Таблица 6.14 – Массив A(N) для тестирования

Индекс элемента массива	**1**	**2**	**3**	**4**	**5**
Значение элемента массива	2	4	7	10	11

Шаг 1. Начать работу алгоритма.
Шаг 2. Ввод 2, 4, 7, 10, 11.
Шаг 3. S:= 0.
Шаг 4. P:=1.
Шаг 5. *i*:= 1.
Шаг 6. S:= 0 + 2.
Шаг 7. P:= 1 * 2.
Шаг 8. *i*:= 1 + 1.
Шаг 9. 2 <= 5, да.
Шаг 10. S:= 2 + 4.
Шаг 11. P:= 2 * 4.
Шаг 12. *i*:= 2 + 1.
Шаг 13. 3 <= 5, да.
Шаг 14. S:= 6 + 7;
Шаг 15. P:= 8 * 7.
Шаг 16. *i*:= 3 + 1.
Шаг 17. 4 <= 5, да.
Шаг 18. S:= 13 + 10.
Шаг 19. P:= 56 * 10.
Шаг 20. *i*:= 4 + 1;
Шаг 21. 5 <= 5, да.
Шаг 22. S:= 23 + 11.
Шаг 23. P:= 560 * 11.
Шаг 24. *i*:= 5 + 1.
Шаг 25. 6 <= 5, нет.
Шаг 26. Вывод 34, 6160.
Шаг 27. Завершить работу алгоритма.

Вывод: в результате работы алгоритма при указанных входных данных будет выведено значение суммы элементов одномерного массива, равное 34, и их произведение, равное 6160. Полученный результат можно проверить другим альтернативным способом.

Протестируем этот алгоритм с использование трассировочной таблицы 6.15.

Таблица 6.15 – Трассировочная таблица алгоритма нахождения суммы и произведения элементов одномерного массива A(N)

Номер шага	Команда алгоритма	Значение переменных				Выполняемое действие
		N	S	P	i	
1	Начало					Запустить алгоритм
2	Ввод A(N), N	5				2, 4, 7, 10, 11, 5
3	S := 0		0			S := 0
4	P := 1			1		P := 1
5	i := 1				1	i := 1
6	S := S + Ai		2			S := 0 + 2
7	P := P * Ai			2		P := 1 * 2
8	i := i + 1				2	i := 1 + 1
9	i <= N					2 <= 5, да
10	S := S + Ai		6			S := 2 + 4
11	P := P * Ai			8		P := 2 * 4
12	i := i + 1				3	i := 2 + 1
13	i <= N					3 <= 5, да
14	S := S + Ai		13			S := 6 + 7
15	P := P * Ai			56		P := 8 * 7
16	i := i + 1				4	i := 3 + 1
17	i <= N					4 <= 5, да
18	S := S + Ai		23			S := 13 + 10
19	P := P * Ai			560		P := 56 * 10
20	i := i + 1				5	i := 4 + 1
21	i <= N					5 <= 5, да
22	S := S + Ai		34			S := 23 + 11
23	P := P * Ai			6160		P := 560 * 11
24	i := i + 1				6	i := 5 + 1
25	i <= N					6 <= 5, нет
26	Вывод S, P					S = 34, P = 6160
27	Конец					Остановить алгоритм

6. Разработка описания алгоритма решения задачи на языке программирования высокого уровня (C#) – программного приложения.

6.1 Выбор, описание идентификаторов программного приложения и внутреннее представление входных, выходных и временных данных.

Для наглядности и краткости приведем в таблице 6.16 выбранные идентификаторы, их описание и их внутреннее.

Таблица 6.16 – Характеристики идентификаторов программного приложения

Наименование идентификатора	Описание	Название типа данных	Ключевое слово	Диапазон значений	Размер, байт	Класс типа		Назначение данных		
						Простой	Структурированный	Входные	Выходные	Временные
A	Имя одномерного массива	Целый	int	От $-2 \cdot 10^9$ до $2 \cdot 10^9$	32		+	+		
N	Количество элементов одномерного массива	Целый	int	От $-2 \cdot 10^9$ до $2 \cdot 10^9$	32	+		+		
i	Индекс элементов одномерного массива	Целый	int	От $-2 \cdot 10^9$ до $2 \cdot 10^9$	32	+				+
S	Сумма элементов одномерного массива	Целый	int	От $-2 \cdot 10^9$ до $2 \cdot 10^9$	32	+			+	
P	Сумма элементов одномерного массива	Целый	int	От $-2 \cdot 10^9$ до $2 \cdot 10^9$	32	+			+	

Обратите внимание, что идентификатор A является структурированным типом данных, состоящий из простых типов данных целого типа, а переменная *i* является временной.

6.2 Разработка программного приложения и описание его работы.

Алгоритм на рисунке 6.22 является циклическим, и для его описания на языке программирования C# можно использовать оператор передачи управления ***goto*** и 4 оператора цикла: ***while, do..while, for, foreach***. Поэтому для более полного и глубокого понимания возможностей языка программирования рассмотрим программные коды этого алгоритма с использованием всех перечисленных операторов.

Разработанное программное приложение на языке C# для обработки одномерного массива с использованием оператора ***goto*** представлено на рисунке 6.23.

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   namespace Циклические_программы_1_одномерные_массивы
6   {
7      class Program
8      {
9         static void Main(string[] args)
10        {
11          // Описание и определение переменных
12            // Внутренне представление входных данных
13            int[] A = new int[5] { 2, 4, 7, 10, 11 };        // Описание и определение одномерного массива A
14            int N = 5;                                      // Описание и определение размерности массива A
15            // Внутренне представление выходных данных
16            int S = 0;                                      // Описание и определение переменной S
```

```
17          int P = 1;                                          // Описание и определение переменной P
18          // Внутренне представление временных данных
19          int i = 0;                                          // Описание и определение переменной i
20      M1: S = S + A[i];                                       // Начало цикла и вычисление суммы
21          P = P * A[i];                                       // Вычисление произведения
22          i=i+1;                                              // Изменение индекса
23          if (i <= N - 1) goto M1;                            // Конец цикла
24          Console.WriteLine("S=" + S + " P=" + P);    // Вывод на консоль переменных S и P
25          Console.ReadLine();
26        }
27    }
28 }
```

Рисунок 6.23 – Код программы вычисления суммы и произведения элементов одномерного массива с оператором *goto*

В программном коде на рисунке 6.23 в строке 13 описан и определен одномерный массив целого типа с элементами целого типа, то есть массив определяется в коде программы и поэтому программа носит частный характер. В следующих примерах в коде программы массив и его размерность будут описываться в общем виде и вводится с консоли. Более детально с описаниями и определениями одномерных массивов можно ознакомиться в разделе 2.4. В строке 19 переменной цикла присваивается значение 0, так как в C# нумерация элементов массива начинается с нуля, поэтому цикл должен завершиться при значении, равном 4, правой части логического в операторе ***if*** (строка 23), то есть N-1. Остальные операторы кода этой программы описывались выше.

Часть программного кода на языке C# для обработки одномерного массива с использованием оператора ***while*** представлена на рисунке 6.24.

```
11          // Описание и определение переменных
12          // Внутренне представление входных данных
13          Console.Write("Введите длину массива: ");
14          int N = int.Parse(Console.ReadLine());     // Описание и определение размерности массива A
15          int[] A = new int[N];                      // Описание одномерного массива A
16          // Внутренне представление выходных данных
17          int S = 0;                                 // Описание и определение переменной S
18          int P = 1;                                 // Описание и определение переменной P
19          // Внутренне представление временных данных
20          int i = 0;                                 // Описание и определение переменной i
21          Console.WriteLine("Введите элементы массива поочередно, нажимая клавишу Enter: ");
22          while (i <= N-1)                           // Начало цикла
23          {
24             A[i] = int.Parse(Console.ReadLine());   // Ввод элементов массива
25             S = S + A[i];
26             P = P * A[i];
27             i = i + 1;
28          }                                          // Конец цикла
29          Console.WriteLine("Ответ:");
30          Console.WriteLine("S=" + S + " P=" + P); // Вывод на консоль переменных S и P
31          Console.ReadLine();
```

Рисунок 6.24 – Часть программного кода для вычисления суммы и произведения элементов одномерного массива с оператором *while*

В коде на рисунке 6.24 размерность массива описывается и определяется в общем виде (строка 14). В строке 15 только описывается в общем виде, а для его определения используется цикл (строки с 22 по 28). Обратите внимание, что строка 24 находится в теле цикла. Это означает, что при вводе очередного элемента массива с консоли он сразу обрабатывается (выполняются операции суммирования и умножения), поэтому для ввода элементов массива не

нужно организовывать дополнительный цикл, что сокращает код программы. Строки с 22 по 28 на рисунке 6.24 заменяют строки кода с 20 по 23 на рисунке 6.23.

Часть программного кода на языке C# для обработки одномерного массива с использованием оператора ***do..while*** представлена на рисунке 6.25.

```
11          // Описание и определение переменных
12          // Внутренне представление входных данных
13          Console.Write("Введите длину массива: ");
14          int N = int.Parse(Console.ReadLine());     // Описание и определение размерности массива A
15          int[] A = new int[N];                      // Описание одномерного массива A
16          // Внутренне представление выходных данных
17          int S = 0;                                 // Описание и определение переменной S
18          int P = 1;                                 // Описание и определение переменной P
19          // Внутренне представление временных данных
20          int i = 0;                                 // Описание и определение переменной i
21          Console.WriteLine("Введите элементы массива поочередно, нажимая клавишу Enter: ");
22          do                                         // Начало цикла
23             {
24             A[i] = int.Parse(Console.ReadLine());   // Ввод элементов массива
25             S = S + A[i];
26             P = P * A[i];
27             i = i + 1;
28             }
29          while (i <= N-1) ;                          // Конец цикла
30          Console.WriteLine("Ответ:");
31          Console.WriteLine("S=" + S + " P=" + P); // Вывод на консоль переменных S и P
32          Console.ReadLine();
```

Рисунок 6.25 – Часть программного кода для вычисления суммы и произведения элементов одномерного массива с оператором *do..while*

Строки с 22 по 29 на рисунке 6.25 заменяют строки кода с 22 по 28 на рисунке 6.24.

Часть программного кода на языке C# для обработки одномерного массива с использованием оператора ***for*** представлена на рисунке 6.26.

```
11          // Описание и определение переменных
12          // Внутренне представление входных данных
13          Console.Write("Введите длину массива: ");
14          int N = int.Parse(Console.ReadLine());     // Описание и определение размерности массива A
15          int[] A = new int[N];                      // Описание одномерного массива A
16          // Внутренне представление выходных данных
17          int S = 0;                                 // Описание и определение переменной S
18          int P = 1;                                 // Описание и определение переменной P
19          // Внутренне представление временных данных
20          Console.WriteLine("Введите элементы массива поочередно, нажимая клавишу Enter: ");
21          for (int i = 0; i <= N-1; i += 1)          // Начало цикла
22             {
23                A[i] = int.Parse(Console.ReadLine());// Ввод элементов массива
24                S = S + A[i];
25                P = P * A[i];
26             }                                       // Конец цикла
27          Console.WriteLine("Элементы одномерного массива");
28          // Вывод элементов одномерного массива в строку
29          for (int i = 0; i <= N - 1; i += 1)
30              Console.Write("{0} ",A[i]);            // Вывод элементов массива
31          Console.WriteLine();
32          Console.WriteLine("Ответ:");
33          Console.WriteLine("S=" + S + " P=" + P);   // Вывод на консоль переменных S и P
34          Console.ReadLine();
```

Рисунок 6.26 – Часть программного кода для вычисления суммы и произведения элементов одномерного массива с оператором *for*

Строки с 21 по 26 на рисунке 6.26 заменяют строки кода с 22 по 28 на рисунке 6.25. Кроме этого в программный код добавлен цикл вывода одномерного массива в строку (строки 29, 30).

Часть программного кода на языке C# для обработки одномерного массива с использованием оператора ***foreach*** представлена на рисунке 6.27.

```
11          // Описание и определение переменных
12           // Внутренне представление входных данных
13           Console.Write("Введите длину массива: ");
14           int N = int.Parse(Console.ReadLine());            // Описание и определение размерности массива A
15           int[] A = new int[N];                             // Описание одномерного массива A
16           // Внутренне представление выходных данных
17           int S = 0;                                        // Описание и определение переменной S
18           int P = 1;                                        // Описание и определение переменной P
19           // Внутренне представление временных данных
20           Console.WriteLine("Введите элементы массива поочередно, нажимая клавишу Enter: ");
21           for (int i = 0; i <= N-1; i += 1)                 // Начало цикла
22              A[i] = int.Parse(Console.ReadLine());          // Ввод элементов массива
23           foreach (int j in A)                              // Начало цикла
24              {
25                 S = S + j;
26                 P = P * j;
27              }                                               // Конец цикла
28           // Вывод элементов одномерного массива в строку
29           Console.WriteLine("Элементы одномерного массива");
30           foreach (int j in A)                              // Начало цикла
31              Console.Write("{0} ", j);                      // Вывод элементов массива и конец цикла
32           Console.WriteLine();                              // Переход на следующую строку
33           Console.WriteLine("Ответ:");
34           Console.WriteLine("S=" + S + " P=" + P);          // Вывод на консоль переменных S и P
35           Console.ReadLine();
```

Рисунок 6.27 – Часть программного кода для вычисления суммы и произведения элементов одномерного массива с оператором *foreach*

Строки с 21 по 27 на рисунке 6.27 заменяют строки кода с 21 по 26 на рисунке 6.26. В этом программном коде используются 2 оператора ***foreach***: один для вычисления суммы и произведения (строка 23), другой для вывода массива (строка 30). Обратите внимание на строки 25, 26 и 31, где по-другому записываются элементы массива. Подробно правила работы оператора ***foreach*** описаны в подразделе 2.6.7.

6.3 Тестирование программного приложения и описание тестовых случаев.

После запуска программы с кодом, представленным на рисунке 6.23, на экране монитора появится сообщение, изображенное на рисунке 6.28.

Рисунок 6.28 – Окно с результатом работы программы, показанной на рисунке 6.23

После запуска программ с кодами, представленными на рисунках 6.24 и 6.25, на экране монитора появится сообщение, изображенное на рисунке 6.29. Вначале необходимо ввести длину массива, нажать клавишу ***Enter***, затем поочередно, используя клавишу ***Enter***, ввести 5 элементов массива.

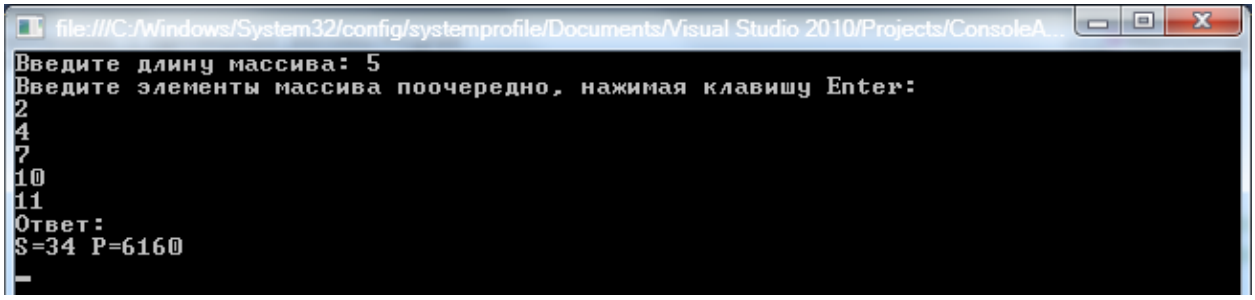


Рисунок 6.29 – Окно с результатом работы программ, представленных на рисунках 6.24 и 6.25

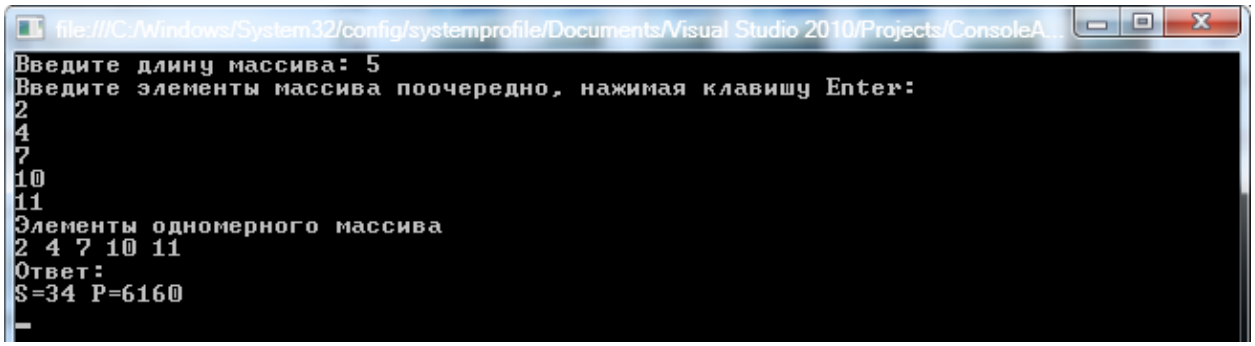


Рисунок 6.30 – Окно с результатом работы программ, представленных на рисунках 6.26 и 6.27

Таблица 6.17 – Варианты заданий 4-й лабораторной работы

Вариант	**Задание**
1	**2**
1	Найти количество чисел, принадлежащих промежутку [a, b], и сумму чисел, стоящих на местах, кратных 3
2	Найти сумму чисел, меньших заданного D, и количество чисел, стоящих на четных местах и больших заданного C
3	Найти произведение всех чисел, стоящих на местах, кратных 4, и количество чисел, не больших заданного A
4	Найти количество чисел, меньших заданного X, и произведение всех отрицательных чисел, стоящих на нечетных местах
5	Найти количество чисел, не принадлежащих промежутку (X,Y], и сумму отрицательных чисел, стоящих на четных местах
6	Найти количество неотрицательных чисел и определить сумму чисел, стоящих на местах, кратных 3 и не равных заданному F
7	Найти среднее арифметическое отрицательных чисел и определить количество чисел, по величине больших A и стоящих на четных местах
8	Найти среднее арифметическое положительных чисел, стоящих на нечетных местах, и количество чисел, меньших заданного B
9	Найти среднее арифметическое чисел, принадлежащих промежутку [A,B), и количество положительных чисел, стоящих на местах, кратных 4
10	Найти среднее арифметической чисел, не равных заданному C, и произведение неположительных чисел, стоящих на четных местах
11	Найти среднее арифметическое чисел, больших заданного D и стоящих на нечетных местах, и определить количество чисел, не больших заданного F
12	Найти среднее арифметическое чисел, не попадающих в промежуток [A,B], и количество положительных чисел, стоящих на местах, кратных 3

Продолжение таблицы 6.17

1	2
13	Найти среднее арифметическое ненулевых чисел и количество чисел, по величине не больших A и стоящих на четных местах
14	Вычислить произведение чисел, принадлежащих промежутку (A,B], и количество отрицательных чисел, стоящих на местах, кратных 3
15	Найти среднее арифметическое положительных чисел, стоящих на нечетных местах, и произведение чисел, меньших заданного C
16	Вычислить сумму квадратов чисел, не принадлежащих промежутку [X,Y), и количество отрицательных чисел, стоящих на четных местах
17	Найти количество нулей во всем массиве и определить сумму квадратов чисел, принадлежащих промежутку (A,B) и стоящих на местах, кратных 4
18	Найти произведение чисел, не равных заданному числу Z, и определить количество чисел, стоящих на нечетных местах и принадлежащих промежутку (A,B]
19	Вычислить сумму неотрицательных чисел, стоящих на местах, кратных 3, и количество чисел, равных заданному T
20	Вычислить сумму квадратов чисел, больших заданного C, и количество неположительных чисел, стоящих на местах, кратных 4
21	Найти количество ненулевых чисел и определить среднее арифметическое чисел, не больших A и стоящих на четных местах
22	Найти произведение положительных чисел и определить количество чисел, принадлежащих промежутку (A,B) и стоящих на нечетных местах
23	Найти сумму квадратов отрицательных чисел, стоящих на местах, кратных 3, и количество чисел, не принадлежащих промежутку [A,B)
24	Найти сумму чисел, принадлежащих промежутку [A,B], и определить количество нулей, стоящих на местах, кратных 4
25	Найти количество чисел, не меньших заданного C, и определить сумму квадратов чисел, принадлежащих промежутку (A,B] и стоящих на четных местах
26	Найти количество чисел, не равных заданному X, и определить произведение чисел, больших заданного A и стоящих на местах, кратных 3
27	Найти количество ненулевых чисел, стоящих на нечетных местах, и среднее арифметическое чисел, меньших заданного T
28	Найти количество чисел, равных W, и определить сумму квадратов чисел, не меньших заданного U и стоящих на местах, кратных 4
29	Найти произведение чисел, принадлежащих промежутку [C,D), и определить количество чисел, больших заданного L и стоящих на местах, кратных 3
30	Найти сумму ненулевых чисел, стоящих на четных местах, и количество чисел, принадлежащих промежутку (M,K)
31	Сформировать массив из элементов исходных массивов, больших второго элемента первого массива и положительных элементов 2-го массива
32	Сформировать массив из отрицательных элементов первого массива и элементов обоих массивов, больших первого элемента второго массива
33	Сформировать массив из элементов исходных массивов, меньших произведения последних элементов заданных массивов
34	Сформировать массив из положительных элементов исходных массивов, меньших 10
35	Сформировать массив из отрицательных элементов исходных массивов, больших –5
36	Сформировать массив из элементов исходных массивов, не превышающих третий элемент каждого из них
37	Сформировать массив из элементов исходных массивов, не превышающих первого элемента первого массива
38	Сформировать массив из элементов исходных массивов, больших первого элемента второго массива

Продолжение таблицы 6.17

1	2
39	Сформировать массив из элементов исходных массивов, не превышающих сумму первых элементов исходных массивов
40	Сформировать массив из положительных элементов первого массива и отрицательных элементов второго массива
41	Сформировать массив из отрицательных элементов первого массива и положительных элементов второго массива
42	Сформировать массив из отрицательных элементов первого массива и элементов второго массива, не больших 3
43	Сформировать массив из элементов первого массива, больших последнего элемента второго массива, и элементов второго массива, меньших последнего элемента первого массива
44	Сформировать массив из тех элементов исходных массивов, которые меньше заданного числа
45	Сформировать массив из элементов первого массива, больших 5, и элементов обоих массивов, меньших –7
46	Сформировать массив из элементов исходных массивов, не принадлежащих промежутку [–4; 6], и из элементов, больших 12, второго массива
47	Сформировать массив из элементов первого массива, которые больше заданного числа D, и элементов второго массива, которые не больше D
48	Сформировать массив из элементов первого массива, не принадлежащих промежутку [2; 7], и элементов второго массива из этого промежутка
49	Сформировать массив из элементов, больших 1, первого массива и элементов второго массива, принадлежащих промежутку (0; 1)
50	Сформировать массив из положительных элементов первого массива и элементов обоих массивов, меньших –4
51	Сформировать массив из элементов обоих массивов, меньших заданного значения, и отрицательных элементов второго массива
52	Сформировать массив из элементов исходных массивов, попадающих в отрезок [–10; 3]
53	Сформировать массив из элементов, больших 10 и меньших –10, исходных массивов
54	Сформировать массив из меньших –3 элементов исходных массивов, стоящих на четных местах
55	Сформировать массив из положительных элементов первого массива и отрицательных, больших –5 элементов второго массива
56	Сформировать массив из положительных элементов первого массива, стоящих на четных местах, и элементов второго массива, не превышающих первый элемент второго массива
57	Сформировать массив из элементов первого массива, больших первого элемента второго массива, и из отрицательных элементов 2-го массива
58	Сформировать массив из элементов исходных массивов, не больших 3 и не меньших 10
59	Сформировать массив из положительных элементов первого массива и отрицательных элементов обоих массивов
60	Сформировать массив из отрицательных элементов первого массива и всех элементов исходных массивов, больших 5
61	Найти максимальный элемент и поменять его местами с последним элементом массива
62	Найти минимальный элемент и поменять его местами с предыдущим элементом массива
63	Найти минимальный элемент и поменять его местами с последующим элементом массива
64	Найти максимальный элемент и поменять его местами с шестым элементом массива
65	Найти максимальный элемент, присвоить его значение последнему элементу массива, а вместо максимального числа записать – 1
66	Найти минимальный элемент, присвоить его значение первому элементу массива, а вместо минимального элемента записать число 9999
67	Найти минимальный элемент и поменять его местами с третьим элементом массива
68	Найти максимальный элемент и поменять его местами с предпоследним элементом массива

Окончание таблицы 6.17

1	2
69	Найти минимальный элемент и присвоить его значение элементу с номером (N – 3), а вместо минимального элемента записать число 101
70	Найти максимальный элемент и поменять его местами с элементом под номером (N – 4)
71	Найти минимальный элемент и записать вместо него число N2 + N
72	Найти максимальный элемент и поменять его местами со вторым элементом массива
73	Найти минимальный элемент и поменять его местами с последним элементом массива
74	Найти максимальный элемент и вместо него записать значение N + 2
75	Найти минимальный элемент и поменять его местами с третьим элементом массива
76	Найти минимальный элемент и вместо него записать N2
77	Найти максимальный элемент и поменять его местами с предпоследним элементом массива
78	Найти минимальный элемент, присвоить его значение последнему элементу массива, а вместо минимального элемента записать значение 3N
79	Найти максимальный элемент и поменять его местами с четвертым элементом массива
80	Найти минимальный элемент и поменять его местами с предпоследним элементом массива
81	Найти максимальный элемент и присвоить его значение элементу с номером (N – 3)
82	Найти минимальный элемент и присвоить его значение второму элементу массива
83	Найти максимальный элемент и поменять его местами со вторым элементом массива
84	Найти минимальный элемент и поменять его местами с элементом массива, номер которого задан
85	Найти максимальный элемент и поменять его местами с последующим элементом массива
86	Найти минимальный элемент, присвоить его значение первому элементу массива, а вместо минимального числа записать – 10
87	Найти минимальный элемент, присвоить его значение второму и четвертому элементам массива, а вместо минимального числа записать сумму второго и четвертого элементов массива
88	Найти максимальный элемент и поменять его местами с элементом, номер которого задан
89	Найти минимальный элемент и заменить его полусуммой первого и последнего элементов
90	Найти максимальный элемент и поменять его местами с предпоследним элементом массива

Контрольные вопросы и задания:

1. Что такое одномерный массив?
2. Зачем нужны одномерные массивы (приведите примеры)?
3. Что такое индекс одномерного массива?
4. Сколько индексов у одномерного массива?
5. Как выполнить описание одномерного массива?
6. Как обратиться к элементу одномерного массива?
7. Опишите массив вещественных чисел, который может содержать не более 10 элементов.
8. Укажите способы определения одномерного массива.
9. Укажите способы описания и определения одномерного массива одновременно.
10. Как выполнить ввод одномерного массива?
11. Нарисуйте и опишите фрагмент блок-схемы алгоритма ввода одномерного массива.
12. Как выполнить вывод одномерного массива?
13. Нарисуйте и опишите фрагмент блок-схемы алгоритма вывода одномерного массива.
14. Что такое элементы одномерного массива с нечетными индексами?
15. Что такое элементы одномерного массива с четными индексами?
16. Нарисуйте фрагмент блок-схемы алгоритма для перебора элементов одномерного массива с нечетными индексами.
17. Нарисуйте фрагмент блок-схемы алгоритма для перебора элементов одномерного массива с четными индексами.

6.5 Лабораторная работа 5
Циклические алгоритмы: обработка двумерных массивов

В 5-й лабораторной работе необходимо разработать циклический алгоритм решения задачи обработки двумерного массива согласно варианту (таблица 6.21) и описать его тремя способами – словесное описание, графическое и на языке программирования высокого уровня. В описании методики выполнения лабораторной работы необходимо придерживаться структуры ОГЛАВЛЕНИЯ, предложенной в главе 4.

Опишем методику выполнения одной из задач.

1. Постановка задачи. Дан двумерный массив A(M, N). Найти количество положительных, отрицательных и нулевых элементов массива. Требования к алгоритму: массив и его размерность вводятся с консоли, на экран монитора выводятся массив, количество положительных, отрицательных и нулевых элементов.

2. Математическое описание решения задачи.

Матрицы принято обозначать заглавными латинскими буквами, а их элементы – соответствующими строчными буквами. Каждый элемент матрицы наделяется двумя индексами, обозначающими номер строки и номер столбца, на пересечении которых находится элемент (первый индекс – номер строки, второй – номер столбца). Например, элемент ***a23*** (читается «а два-три» и находится на пересечении второй строки и третьего столбца матрицы). Общий вид матрицы строения m×n (имеющей m строк и n столбцов):

$$A = \begin{pmatrix} a11 & a21 \cdots & a1n \\ a21 & a22 \cdots & a2n \\ \cdots & \cdots & \cdots \\ am1 & am2 \cdots & amn \end{pmatrix},$$

где A – это имя матрицы;

aij – элемент матрицы (здесь индекс i принимает целочисленные значения от 1 до m, индекс j – от 1 до n);

многоточия обозначают не выписанные в явном виде элементы матрицы.

Матрица, имеющая одинаковое количество строк и столбцов, называется квадратной. Для квадратной матрицы число m = n называется ее порядком.

$$S = \sum_{i=1}^{M} \sum_{j=1}^{N} A_{ij} \text{ и } P = \prod_{i=1}^{M} \prod_{j=1}^{N} A_{ij},$$

где S – сумма элементов двумерного;

Aij – один элемент массива;

i – индекс строки;

j – индекс столбца;

M – количество строк массива;

N – количество столбцов массива

P – произведение элементов двумерного массива, а также графические обозначения суммы и произведения соответственно.

3. Определение входных, выходных и временных данных. Входными данными являются двумерный массив A(M, N), количество строк N и количество столбцов M. Выходными данными являются количество положительных (KP), количество отрицательных (KO) и количество нулевых (KN) элементов массива. Временными данными являются индексы элементов двумерного массива i и j.

4. Разработка словесного и графического описания алгоритма решения задачи.

Словесное описание представим последовательностью шагов.

Шаг 1. Начать работу алгоритма.

Шаг 2. Ввести массив A (M, N), т. е. все его элементы и его размерность M (количество строк) и N (количество столбцов).

Шаг 3. В ячейку KP записать значение 0 (ноль), которая будет использоваться для накопления количества положительных элементов массива.

Шаг 4. В ячейку KO записать значение 0 (ноль), которая будет использоваться для накопления количества отрицательных элементов массива.

Шаг 5. В ячейку KN записать значение 0 (ноль), которая будет использоваться для накопления количества нулевых элементов массива.

Шаг 6. В ячейку *i* записать значение 1, где *i* – это индекс строки элементов массива.

Шаг 7. В ячейку *j* записать значение 1, где *j* – это индекс столбцов элементов массива.

Шаг 8. Проверить, если $A_{ij} > \mathbf{0}$, то перейти к шагу 9, в противном случае – к шагу 10.

Шаг 9. В ячейку KP необходимо добавить единицу, т. е. старое значение ячейки KP сложить с единицей и результат записать в ячейку KP (это будет выглядеть так – KP := KP + 1), затем перейти к шагу 13.

Шаг 10. Проверить, если $A_{ij} = \mathbf{0}$, то перейти к шагу 11, в противном случае перейти к шагу 12.

Шаг 11. В ячейку KN необходимо добавить единицу, т. е. старое значение ячейки KP сложить с единицей и результат записать в ячейку KN (это будет выглядеть так – KN := KN + 1), затем перейти к шагу 13.

Шаг 12. В ячейку KO необходимо добавить единицу, т. е. старое значение ячейки KO сложить с единицей и результат записать в ячейку KO (это будет выглядеть так – KO := KO + 1), затем перейти к шагу 13.

Шаг 13. В ячейку *j* записать сумму старого значения *j* плюс один (это означает переход к следующему элементу массива в строке и будет выглядеть *j* := *j* + 1) и перейти к шагу 14.

Шаг 14. Проверить, если **j <= N**, то перейти к шагу 8, в противном случае перейти к шагу 15.

Шаг 15. В ячейку *i* записать сумму старого значения *i* плюс один (это означает переход к следующему массиву в строке и будет выглядеть *i* := *i* + 1) и перейти к шагу 16.

Шаг 16. Проверить, если i **<= M**, то перейти к шагу 7, в противном случае перейти к шагу 17.

Шаг 17. Вывод KP, KO, KN.

Шаг 18. Завершить работу алгоритма.

Примечание. Описанное словесное описание алгоритма перебирает элементы матрицы по строкам слева направо, начиная с первого элемента. Элементы матрицы можно перебирать и по другому направлению, например по столбцам сверху вниз, начиная с первого элемента (первый элемент – это A_{11}).

Разработанное графическое описание алгоритма представлено на рисунке 6.31. Этот алгоритм состоит из двух циклов, вложенных друг в друга. Цикл с 7 вершины по 13 называется внутренним, а цикл с 6 вершины по 15 называется внешним. Внутренний цикл выполняется M x N раз, а внешний M раз.

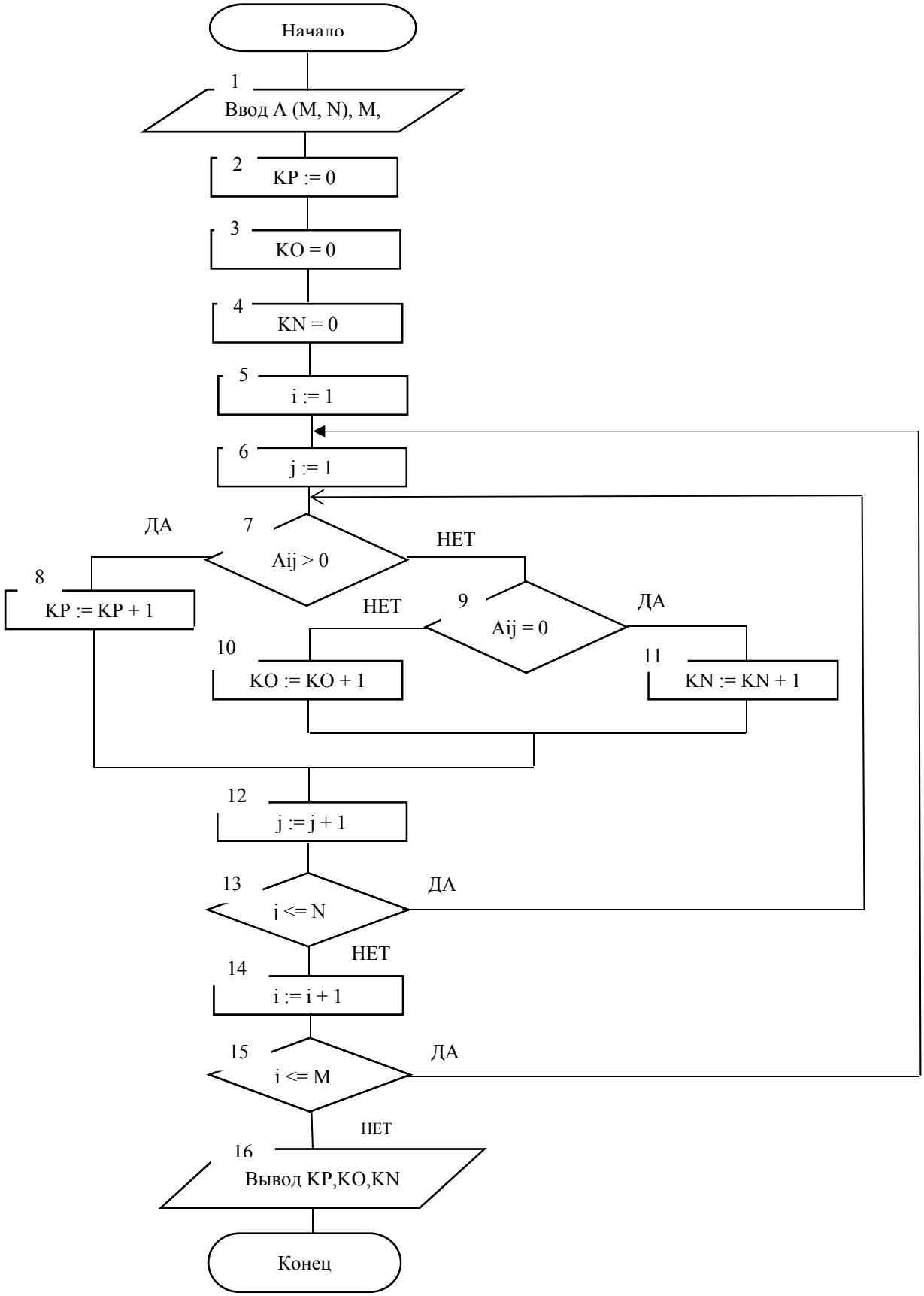


Рисунок 6.31 – Графическое описание алгоритма нахождения количества положительных, отрицательных и нулевых элементов в двумерном массиве A(M, N)

5. Тестирование графического описания алгоритма двумя способами: последовательный метод, трассировочная таблица.

Опишем последовательный метод тестирования алгоритма, представленного на рисунке 6.31. Для тестирования возьмем данные из таблицы 6.18.

Таблица 6.18 – Массив A(M, N)

Индекс строки \ Индекс столбца	1	2	3	4
1	0	–4	7	–10
2	–6	5	0	9

Шаг 1. Начать работу алгоритма.
Шаг 2. Ввод 0, –4, 7, –10, –6, 5, 0, 9, 2,4.
Шаг 3. KP:= 0.
Шаг 4. KO:= 0.
Шаг 5. KN:= 0.
Шаг 6. *i*:= 1.
Шаг 7. *j*:= 1.
Шаг 8. 0 > 0, нет.
Шаг 9. 0 = 0, да.
Шаг 10. KN:= 0 + 1.
Шаг 11. *j*:= 1 + 1.
Шаг 12. 2 <= 4, да.
Шаг 13. –4 > 0, нет.
Шаг 14. –4 = 0, нет.
Шаг 15. KO:= 0 + 1.
Шаг 16. *j*:= 2 + 1.
Шаг 17. 3 <= 4, да.
Шаг 18. 7 > 0, да.
Шаг 19. KP:= 0 + 1.
Шаг 20. *j*:= 3 + 1.
Шаг 21. 4 <= 4, да.
Шаг 22. –10 > 0, нет.
Шаг 23. –10 = 0, нет.
Шаг 24. KO:= 1 + 1.
Шаг 25. *j*:= 4 + 1.
Шаг 26. 5 <= 4, нет.
Шаг 27. *i*:= 1 + 1.
Шаг 28. 2 <= 2 да.
Шаг 29. *j*:= 1.
Шаг 30. –6 > 0, нет.
Шаг 31. –6 = 0, нет.
Шаг 32. KO:= 2 + 1.
Шаг 33. *j*:= 1 + 1.
Шаг 34. 2 <= 4, да.
Шаг 35. 5 > 0, да.
Шаг 36. KP:= 1 + 1.
Шаг 37. *j*:= 2 + 1.
Шаг 38. 3 <= 4, да.
Шаг 39. 0 > 0, нет.
Шаг 40. 0 = 0, да.
Шаг 41. KN:= 1 + 1.
Шаг 42. *j*:= 3 + 1.
Шаг 43. 4 <= 4, да.
Шаг 44. 9 > 0, да.
Шаг 45. KP:= 2 + 1.
Шаг 46. *j*:= 4 + 1.
Шаг 47. 5 <= 4, нет.
Шаг 48. *i*:= 2 + 1.
Шаг 49. 3 <= 2 нет.
Шаг 50. Вывод KP, KO, KN.
Шаг 51. Завершить работу алгоритма.

Вывод: в результате работы алгоритма при указанных входных данных будет выведено количество положительных элементов, равное 3, количество отрицательных элементов, равное 3, количество нулевых элементов, равное 2. Полученный результат можно проверить другим альтернативным способом.

Протестируем этот алгоритм с использование трассировочной таблицы 6.19.

Таблица 6.19 – Трассировочная таблица алгоритма нахождения количества положительных, отрицательных и нулевых элементов двумерного массива A(M, N)

Номер шага	Команда алгоритма	Значение переменных							Выполняемое действие
		M	N	KP	KO	KN	j	i	
1	Начало								Запустить алгоритм
2	Ввод A (M,N), M, N	2	4						Ввод 0, –4, 7, –10, –6, 5, 0, 9, 2, 4
3	KP := 0			0					KP := 0
4	KO := 0				0				KO := 0
5	KN := 0					0			KN := 0
6	*i* := 1						1		*i* := 1
7	j := 1							1	*j* := 1
8	$A_{ij} > 0$								0 > 0, нет
9	$A_{ij} = 0$								0 = 0, да
10	KN := KN + 1					1			KN := 0 + 1
11	j := j + 1						2		j := 1 + 1
12	*j* <= N								2 <= 4, да
13	$A_{ij} > 0$								–4 > 0, нет
14	$A_{ij} = 0$								–4 = 0, нет
15	KO := KO + 1				1				KO := 0 + 1
16	j := j + 1						3		j := 2 + 1
17	*j* <= N								3 <= 4, да
18	$A_{ij} > 0$								7 > 0, да
19	KP := KP + 1			1					KP := 0 + 1
20	j := j + 1						4		*j* := 3 + 1
21	*j* <= N								4 <= 4, да
22	$A_{ij} > 0$								–10 > 0, нет
23	$A_{ij} = 0$								–10 = 0, нет
24	KO := KO + 1				2				KO := 1 + 1
25	j := j + 1						5		j := 4 + 1
26	*j* <= N								5 <= 4, нет
27	i := i + 1							2	*i* := 1 + 1
28	*i* <= M								2 <= 2, да
29	j := 1						1		*j* := 1
30	$A_{ij} > 0$								–6 > 0, нет
31	$A_{ij} = 0$								–6 = 0, нет
32	KO := KO + 1				3				KO := 2 + 1
33	j := j + 1						2		j := 1 + 1
34	*j* <= N								2 <= 4, да
35	$A_{ij} > 0$								5 > 0, да
36	KP := KP + 1			2					KP := 1 + 1
37	j := j + 1						3		j := 2 + 1
38	*j* <= N								3 <= 4, да
39	$A_{ij} > 0$								0 > 0, нет
40	$A_{ij} = 0$								0 = 0, да
41	KN := KN + 1					2			KN := 1 + 1
42	j := j + 1						4		j := 3 + 1
43	*j* <= N								4 <= 4, да
44	$A_{ij} > 0$								9 > 0, да
45	KP := KP + 1			3					KP := 2 + 1
46	j := j + 1						5		j := 4 + 1
47	*j* <= N								5 <= 4, нет
48	i := i + 1							3	*i* := 2 + 1
49	*i* <= M								3 <= 2, нет
50	Вывод KP, KO, KN								Вывод 3, 3, 2
51	Конец								Остановить алгоритм

6. Разработка описания алгоритма решения задачи на языке программирования высокого уровня (C#) – программного приложения.

6.1 Выбор, описание идентификаторов программного приложения и внутреннее представление входных, выходных и временных данных.

Для наглядности и краткости приведем в таблице 6.20 выбранные идентификаторы, их описание и их внутреннее.

Таблица 6.20 – Характеристики идентификаторов программного приложения

Наименование идентификатора	Описание	Название типа данных	Ключевое слово	Диапазон значений	Размер, байт	Класс типа		Назначение данных		
						Простой	Структурированный	Входные	Выходные	Временные
a	Имя двумерного массива	Целый	int	От $-2 \cdot 10^9$ до $2 \cdot 10^9$	32		+	+		
m	Количество строк двумерного массива	Целый	int	От $-2 \cdot 10^9$ до $2 \cdot 10^9$	32	+		+		
n	Количество столбцов двумерного массива	Целый	int	От $-2 \cdot 10^9$ до $2 \cdot 10^9$	32	+		+		
i	Индекс строки двумерного массива	Целый	int	От $-2 \cdot 10^9$ до $2 \cdot 10^9$	32	+				+
j	Индекс столбца двумерного массива	Целый	int	От $-2 \cdot 10^9$ до $2 \cdot 10^9$	32	+				+
kp	Количество положительных элементов двумерного массива	Целый	int	От $-2 \cdot 10^9$ до $2 \cdot 10^9$	32	+			+	
ko	Количество отрицательных элементов двумерного массива	Целый	int	От $-2 \cdot 10^9$ до $2 \cdot 10^9$	32	+			+	
kn	Количество нулевых элементов двумерного массива	Целый	int	От $-2 \cdot 10^9$ до $2 \cdot 10^9$	32	+			+	

Обратите внимание на идентификаторы, которые состоят из строчных букв, что позволяет быстрее и удобнее набирать программный код.

6.2 Разработка программного приложения и описание его работы.

Алгоритм на рисунке 6.31 является циклическим, и для его описания на языке программирования C# можно использовать оператор передачи управления ***goto*** и 4 оператора цикла: ***while, do..while, for, foreach***. Поэтому для более полного и глубокого понимания возможностей языка программирования рассмотрим программные коды этого алгоритма с использованием всех перечисленных операторов.

Разработанное программное приложение на языке C# для обработки двумерного массива с использованием оператора ***goto*** представлено на рисунке 6.32.

```
1    using System;
2    using System.Collections.Generic;
3    using System.Linq;
4    using System.Text;
5    namespace Циклические_программы_1_двумерные_массивы
6    {
7        class Program
8        {
9            static void Main(string[] args)
10           {
```

```
11          // Описание и определение переменных
12          // Внутренне представление входных данных
13            int[,] a = new int[2, 4] { {  0, -4,  7, -10 },          // Описание и определение двумерного массива a
14                                       {-6,   5, 0,     9 }  } ;
15             int m = 2;                                              // Описание и определение количества строк m
16            int n = 4;                                               // Описание и определение количества строк n
17            // Внутренне представление выходных данных
18            int kp = 0;                                              // Описание и определение переменной kp
19            int ko = 0;                                              // Описание и определение переменной ko
20            int kn = 0;                                              // Описание и определение переменной kn
21            // Внутренне представление временных данных
22            // Начало нахождения положительных, отрицательных и нулевых элементов
23            int i = 0;                                                // Описание и определение переменной i
24       M1: int j = 0;                                  // Начало внешнего цикла, описание и определение переменной j
25       M2: if (a[i, j] > 0) kp = kp + 1;                            // Начало внутреннего цикла
26            else if (a[i, j] == 0) kn = kn + 1;
27                else ko = ko + 1;
28            j++;                                                     // Изменение индекса на 1
29            if (j <= n - 1) goto M2                                  // Конец внутреннего цикла
30            i++;                                                     // Изменение индекса на 1
31            if (i <= m - 1) goto M1;                                 // Конец внешнего цикла
32    // Конец нахождения положительных, отрицательных и нулевых элементов
33            // Начало вывода элементов матрицы
34           Console.WriteLine("Элементы матрицы ");
35            i = 0;
36       M3: j = 0;                                                   // Начало внешнего цикла
37       M4: Console.Write("{0} ", a[i, j]);                          // Начало внутреннего цикла и вывод элементов массива
38            j++;                                                     // Изменение индекса на 1
39            if (j < n) goto M4;                                      // Конец внутреннего цикла
40            i++;                                                     // Изменение индекса на 1
41            Console.WriteLine();                                     // Пропуск строки
42            if (i < m) goto M3;                                      // Конец внешнего цикла
43            // Конец вывода элементов матрицы
44            // Вывод на консоль переменных kp, ko, kn
45            Console.WriteLine("kp=" + kp + " ko=" + ko + " kn=" + kn);
46            Console.ReadLine();
47          }
48       }
49    }
```

Рисунок 6.32 – Код программы нахождения количества положительных, отрицательных и нулевых элементов в двумерном массиве A(M, N) с оператором *goto*

В программном коде на рисунке 6.32 в строке 13 описан и определен двумерный массив целого типа с элементами целого типа, то есть массив определяется в коде программы, поэтому эта программа носит частный характер. В следующих примерах по обработке двумерных массивов в коде программы массив и его размерность будут описываться в общем виде и вводиться с консоли. Более детально с описаниями и определениями двумерных массивов можно ознакомиться в разделе 2.4. В строке 21 и 22 переменным цикла присваиваются значения 0, так как в C# нумерация строк и столбцов начинается с нуля, поэтому внутренний цикл должен завершиться при значении параметра цикла, равном 3, а внешний цикл должен завершиться при значении параметра цикла, равном 1. Строки с 23 по 27 – это операторы внутреннего цикла, а строки с 22 по 29 – это операторы внешнего цикла. В строках 26 и 28 вместо операторов присваивания используются унарные операции инкремента (операция ++). Строки с 34 по 42 предназначены для вывода элементов матрицы на экран монитора. Строка 41 используется для вывода матрицы построчно. При повторном переборе элементов массива индексам строки и столбца необходимо заново присваивать нулевые значения (строки 35 и 36). Более подробно с операциями языка C# можно ознакомиться в подразделе 2.1.3. Остальные операторы кода этой программы описывались выше.

Часть программного кода на языке C# для обработки двумерного массива с использованием оператора ***while*** представлена на рисунке 6.33.

```
12          // Описание и определение переменных
13          // Внутренне представление входных данных
14          Console.WriteLine("Введите количество строк: ");
15          int m = int.Parse(Console.ReadLine());          // Описание и определение количества строк m
16          Console.WriteLine("Введите количество столбцов: ");
17          int n = int.Parse(Console.ReadLine());          // Описание и определение количества столбцов n
18          int[,] a = new int[m, n];                       // Описание двумерного массива a
19          // Внутренне представление выходных данных
20          int kp = 0;                                     // Описание и определение переменной kp
21          int ko = 0;                                     // Описание и определение переменной ko
22          int kn = 0;                                     // Описание и определение переменной kn
23          // Внутренне представление временных данных
24          int i = 0;                                      // Описание и определение переменной i
25          int j = 0;                                      // Описание и определение переменной j
26          // Начало ввода элементов матрицы
27          while (i < m)                                   // Начало внешнего цикла
28             {
29             Console.Write("Введите строку матрицы ");
30             Console.WriteLine("{0} :", i + 1);
31             while (j < n)                                // Начало внутреннего цикла
32                {
33                a[i, j] = int.Parse(Console.ReadLine());  // Определение элемента матрицы
34                j++;
35                }                                         // Конец внутреннего цикла
36             i++;
37             j = 0;
38             }                                            // Конец внешнего цикла
39          // Конец ввода элементов матрицы
40          // Начало нахождения положительных, отрицательных и нулевых элементов
41          i = j = 0;                                      // Присваивание 2-м переменным нулевого значения
42          while (i < m)                                   // Начало внешнего цикла
43          {
44             while (j < n)                                // Начало внутреннего цикла
45               {
46                if (a[i, j] > 0) kp = kp + 1;
47                else if (a[i, j] == 0) kn = kn + 1;
48                else ko = ko + 1;
49                j++;                                      // Изменение индекса на 1
50               }                                          // Конец внутреннего цикла
51          i++;                                            // Изменение индекса на 1
52          j = 0;
53          }                                               // Конец внешнего цикла
54          // Конец нахождения положительных, отрицательных и нулевых элементов
55          // Начало вывода элементов матрицы
56             Console.WriteLine("Элементы матрицы ");
57             i = j = 0;                                   // Присваивание 2-м переменным 0
58             while (i < m)                                // Начало внешнего цикла
59             {
60                while (j < n)                             // Начало внутреннего цикла
61                {
62                   Console.Write("{0} ",a[i,j]);          // Вывод элементов массива
63                   j++;
64                }                                         // Конец внутреннего цикла
65                Console.WriteLine();                      // Пропуск строки
66                i++;
67                j = 0;
68             }                                            // Конец внешнего цикла
69          // Вывод на консоль переменных kp, ko, kn
70          // Console.WriteLine();
71          Console.WriteLine("kp=" + kp + " ko=" + ko + " kn=" + kn);
72          Console.ReadLine();
```

Рисунок 6.33 – Часть программного кода для нахождения количества положительных, отрицательных и нулевых элементов в двумерном массиве A(M, N) с оператором *while*

В коде на рисунке 6.33 размерность описывается и определяется в общем виде (строки 15, 17). В строке 18 массив только описывается в общем виде, а определяется с помощью 2-х циклов (строки с 27 по 38). Строка 30 имеет двойное назначение: позволяет вводить массив построчно и выводит номер вводимой строки, но с консоли элементы строки вводятся в столбец с использованием клавиши ***Enter***. Содержательная часть программы (нахождение количества положительных, отрицательных и нулевых элементов) состоит из двух операторов цикла (строки с 41 по 53). Для вывода двумерного массива на экран монитора необходимо также использовать два оператора цикла, вложенных друг в друга (строки с 56 по 68). Строки 37, 52 67 необходимы для выбора первого элемента в строке после очередного выхода из внутреннего цикла. Обратите особое внимание на строки 41 и 57 (при переборе элементов массива с самого начала необходимо заново присвоить индексам строки и столбца нулевые значения). Оператор в строке 66 необходим для вывода двумерного массива построчно.

Часть программного кода на языке C# для обработки двумерного массива с использованием оператора ***do..while*** представлена на рисунке 6.34.

```
11          // Описание и определение переменных
12          // Внутренне представление входных данных
13          Console.WriteLine("Введите количество строк: ");
14          int m = int.Parse(Console.ReadLine());          // Описание и определение количества строк m
15          Console.WriteLine("Введите количество столбцов: ");
16          int n = int.Parse(Console.ReadLine());          // Описание и определение количества столбцов n
17          int[,] a = new int[m, n];                       // Описание и определение двумерного массива a
18          // Внутренне представление выходных данных
19          int kp = 0;                                     // Описание и определение переменной kp
20          int ko = 0;                                     // Описание и определение переменной ko
21          int kn = 0;                                     // Описание и определение переменной kn
22          // Внутренне представление временных данных
23          int i = 0;                                      // Описание и определение переменной i
24          int j = 0;                                      // Описание и определение переменной j
25          // Начало ввода элементов матрицы
26          do                                              // Начало внешнего цикла
27          {
28             Console.Write("Введите строку матрицы ");
29             Console.WriteLine("{0} :", i + 1);
30             do                                           // Начало внутреннего цикла
31             {
32                a[i, j] = int.Parse(Console.ReadLine());
33                j++;
34             } while (j < n);                             // Конец внутреннего цикла
35             i++;
36             j = 0;
37          } while (i < m);                                // Конец внешнего цикла
38          // Конец ввода элементов матрицы
39          // Начало для нахождения положительных, отрицательных и нулевых элементов
40          i = j = 0;
41          do                                              // Начало внешнего цикла
42          {
43             do                                           // Начало внутреннего цикла
44             {
45                if (a[i, j] > 0) kp = kp + 1;
46                else if (a[i, j] == 0) kn = kn + 1;
47                else ko = ko + 1;
48                j++;                                      // Изменение индекса на 1
49             } while (j < n);                             // Конец внутреннего цикла
50          i++;                                            // Изменение индекса на 1
51          j = 0;
52          } while (i < m);                                // Конец внешнего цикла
53          // Конец для нахождения положительных, отрицательных и нулевых элементов
54          // Начало вывода элементов матрицы
55          Console.WriteLine("     Элементы матрицы ");  // Вывод строковой константы
56             Console.WriteLine("|----------------------|");    // Вывод строковой константы
```

```
57              i = j = 0;
58              do                                          // Начало внешнего цикла
59              {
60                 do                                       // Начало внутреннего цикла
61                 {
62                 Console.Write("|{0,5}", a[i, j]);        // Форматированный вывод элементов двумерного массива
63                 j++;
64                 } while (j < n);                         // Конец внутреннего цикла
65                 Console.WriteLine(" |");                 // Вывод строковой константы
66                 Console.WriteLine("|------------------------|");// Вывод строковой константы
67                 i++;
68                 j = 0;
69              } while (i < m);                            // Конец внешнего цикла
70           // Вывод на консоль переменных kp, ko, kn
71           Console.WriteLine("Ответ:");
72           Console.WriteLine("kp=" + kp + " ko=" + ko + " kn=" + kn);
73           Console.ReadLine();
```

Рисунок 6.34 – Часть программного кода для нахождения количества положительных, отрицательных и нулевых элементов в двумерном массиве A(M, N) с оператором *do..while*

В программе на рисунке 6.34 для ввода элементов массива с консоли, нахождения среди них количества положительных, отрицательных и нулевых элементов и вывода элементов массива на консоль используется по два вложенных друг в друга цикла, каждый из которых описывается при помощи оператора ***do..while***. Логика работы этой программы такая же, как и рисунке 6.33, только для вывода массива применяется форматированный вывод (строка 62). Первая цифра в фигурных скобках {0,5}обозначает номер выводимой переменной, а вторая – количество разрядов на экране монитора под эту переменную. Более детально форматированный вывод описан в подразделе 2.6.9.

Часть программного кода на языке C# для обработки двумерного массива с использованием оператора ***for*** представлена на рисунке 6.35.

```
11           // Описание и определение переменных
12           // Внутренне представление входных данных
13           Console.WriteLine("Введите количество строк: ");
14           int m = int.Parse(Console.ReadLine());          // Описание и определение количества строк m
15           Console.WriteLine("Введите количество столбцов: ");
16           int n = int.Parse(Console.ReadLine());          // Описание и определение количества столбцов n
17           int[,] a = new int[m, n];                       // Описание и определение двумерного массива a
18           // Внутренне представление выходных данных
19           int kp = 0;                                     // Описание и определение переменной kp
20           int ko = 0;                                     // Описание и определение переменной ko
21           int kn = 0;                                     // Описание и определение переменной kn
22           // Внутренне представление временных данных
23           // Начало ввода элементов матрицы
24           for (int i = 0; i < m; i++)                     // Заголовок внешнего цикла
25           {
26              Console.Write("Введите строку матрицы ");
27              Console.WriteLine("{0} :", i + 1);
28              for (int j = 0; j < n; j++)                  // Заголовок внутреннего цикла
29                 a[i, j] = int.Parse(Console.ReadLine());  // Конец внутреннего цикла
30           }                                               // Конец внешнего цикла
31           // Конец ввода элементов матрицы
32           // Начало для нахождения положительных, отрицательных и нулевых элементов
33           for (int i = 0; i < m; i++)                     // Заголовок внешнего цикла
34              for (int j = 0; j < n; j++)                  // Заголовок внутреннего цикла
35              if (a[i, j] > 0) kp = kp + 1;
36              else if (a[i, j] == 0) kn = kn + 1;
37                   else ko = ko + 1;                       // Конец внутреннего и внешнего циклов
38           // Конец для нахождения положительных, отрицательных и нулевых элементов
39           // Начало вывода элементов матрицы
40           Console.WriteLine("     Элементы матрицы ");  // Вывод строковой константы
41           Console.WriteLine("|------------------------|");       // Вывод строковой константы
```

```
42          for (int i = 0; i < m; i++)                          // Заголовок внешнего цикла
43          {
44             for (int j = 0; j < n; j++)                       // Заголовок внутреннего цикла
45                Console.Write("|{0,5}", a[i, j]);    // Форматированный вывод двумерного массива и конец внутреннего цикла
46             Console.WriteLine(" |");                          // Вывод строковой константы
47             Console.WriteLine("|------------------------|");  // Вывод строковой константы
48          }                                                    // Конец внешнего цикла
49          // Вывод на консоль переменных kp, ko, kn
50          Console.WriteLine("Ответ:");
51          Console.WriteLine("kp=" + kp + " ko=" + ko + " kn=" + kn);
52          Console.ReadLine();
```

Рисунок 6.35 – Часть программного кода для нахождения количества положительных, отрицательных и нулевых элементов в двумерном массиве A(M, N) с оператором *for*

В программе на рисунке 6.35 для организации ввода двумерного массива с консоли, выполнения содержательной части задачи и для вывода массива на консоль используется оператор ***for***. В теле цикла оператора (строка 28) находится только один оператор, поэтому для выделения тела цикла фигурные скобки можно не использовать. В строках с 33 по 37 внешний оператор цикла состоит из одного оператора (строки 38, 39) и внутренний оператор цикла также состоит из одного оператора (строки 35, 36, 37), поэтому фигурные скобки можно также не использовать. С применением оператора ***for*** существенно сократилось количество строк кода (на 21 строку), а программный код легче читается и понятнее воспринимается. А логика работы этой программы такая же, как и на рисунке 6.34.

Часть программного кода на языке C# для обработки двумерного массива с использованием оператора ***foreach*** представлена на рисунке 6.36.

```
11          // Описание и определение переменных
12          // Внутренне представление входных данных
13          Console.WriteLine("Введите количество строк: ");
14          int m = int.Parse(Console.ReadLine());            // Описание и определение количества строк m
15          Console.WriteLine("Введите количество столбцов: ");
16          int n = int.Parse(Console.ReadLine());            // Описание и определение количества столбцов n
17          int[,] a = new int[m, n];                         // Описание и определение двумерного массива a
18          // Внутренне представление выходных данных
19          int kp = 0;                                       // Описание и определение переменной kp
20          int ko = 0;                                       // Описание и определение переменной ko
21          int kn = 0;                                       // Описание и определение переменной kn
22          // Внутренне представление временных данных
23          // Начало ввода элементов матрицы
24          for (int i = 0; i < m; i++)                       // Заголовок внешнего цикла
25          {
26             Console.Write("Введите строку матрицы ");
27             Console.WriteLine("{0} :", i + 1);
28             for (int j = 0; j < n; j++)                    // Заголовок внутреннего цикла
29                a[i, j] = int.Parse(Console.ReadLine());    // Конец внутреннего цикла
30          }                                                 // Конец внешнего цикла
31          // Конец ввода элементов матрицы
32          // Начало для нахождения положительных, отрицательных и нулевых элементов
33          foreach (int j in a)                              // Начало цикла
34            if (a[i, j] > 0) kp = kp + 1;
35             else if (a[i, j] == 0) kn = kn + 1;
36                   else ko = ko + 1;                        // Конец цикла
37          // Конец для нахождения положительных, отрицательных и нулевых элементов
38          // Начало вывода элементов матрицы
39          Console.WriteLine("    Элементы матрицы ");  // Вывод строковой константы
40          Console.WriteLine("|------------------------|");  // Вывод строковой константы
41          for (int i = 0; i < m; i++)                       // Заголовок внешнего цикла
42          {
43             for (int j = 0; j < n; j++)                    // Заголовок внутреннего цикла
44                Console.Write("|{0,5}", a[i, j]);  // Форматированный вывод двумерного массива и конец внутреннего
45  цикла
46             Console.WriteLine(" |");                       // Вывод строковой константы
```

```
47              Console.WriteLine("|-----------------------|");    // Вывод строковой константы
48          }                                                      // Конец внешнего цикла
49          // Вывод на консоль переменных kp, ko, kn
50          Console.WriteLine("Ответ:");
51          Console.WriteLine("kp=" + kp + " ko=" + ko + " kn=" + kn);
            Console.ReadLine();
```

Рисунок 6.36 – Часть программного кода для нахождения количества положительных, отрицательных и нулевых элементов в двумерном массиве A(M, N) с оператором *foreach*

В программе на рисунке 6.36 для организации ввода и вывода двумерного массива с консоли и на консоль используются операторы ***for***, а для выполнения содержательной части задачи используется оператор ***foreach*** (строки с 33 по 36). Для обработки двумерных массивов лучше использовать операторы цикла ***for***, так как оператор цикла ***foreach*** обладает следующими недостатками: нельзя пересматривать элементы массива или коллекции в обратном порядке, переменная-итератор не может рассматривать выборочные элементы массива или коллекции, например рассматривать элементы коллекции, которые лежат на парных позициях. Наряду с недостатками оператор цикла ***foreach*** имеет и ряд преимуществ: упрощенность синтаксической конструкции цикла; переменной-итератору (переменной цикла) не нужно задавать начальное значение и указывать прирост, не нужно указывать условие завершения цикла. А логика работы этой программы такая же, как и на рисунке 6.35.

6.3 Тестирование программного приложения и описание тестовых случаев.

После запуска программы с кодом, представленным на рисунке 6.32, на экране монитора появится сообщение, изображенное на рисунке 6.37.

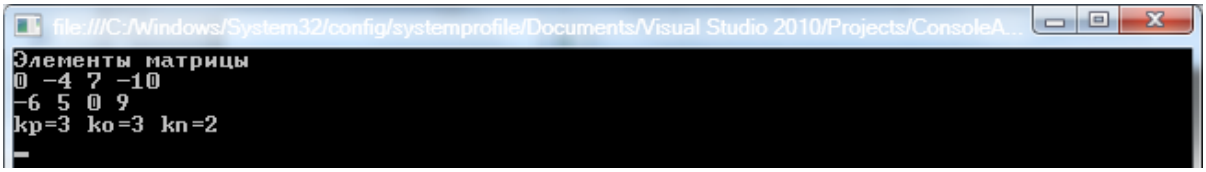


Рисунок 6.38 – Окно с результатом работы программы, показанной на рисунке 6.32

После запуска программы с кодом, представленным на рисунке 6.33, на экране монитора появится сообщение, изображенное на рисунке 6.39. Вначале необходимо ввести количество строк и нажать клавишу ***Enter***, затем ввести количество столбцов и нажать клавишу ***Enter***, а затем поочередно, используя клавишу ***Enter***, ввести по 4 элемента каждой строки.

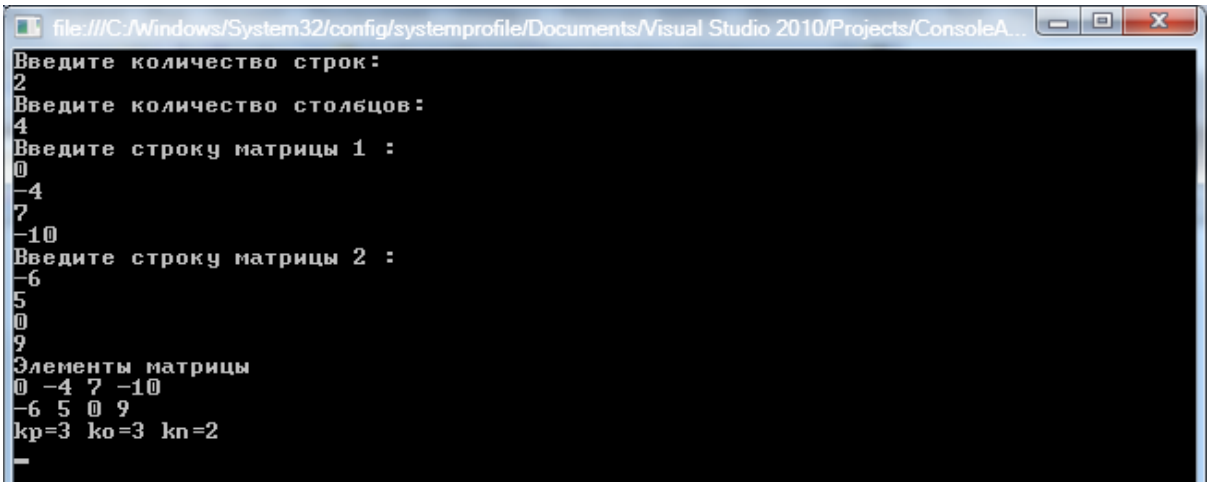


Рисунок 6.39 – Окно с результатом работы программы, показанной на рисунке 6.33

После запуска программ с кодами, представленными на рисунках 6.34, 6.35, 6.36, на экране монитора появится сообщение, изображенное на рисунке 6.40. Входные данные вводятся так же, как и в предыдущей программе. Вывод элементов двумерного массива или прямоугольной матрицы представлен в отформатированном виде.

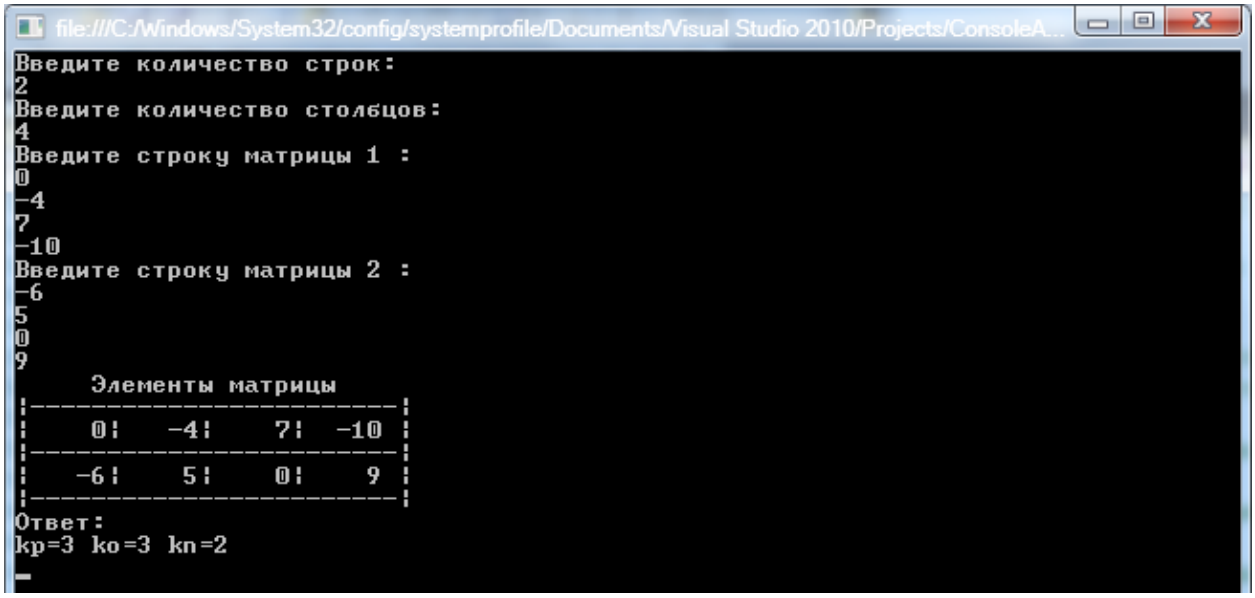


Рисунок 6.40 – Окно с результатом работы программ, показанных на рисунках 6.34, 6.35, 6.36

Таблица 6.21 – Варианты заданий 5-й лабораторной работы

Вариант	**Задание**
1	**2**
1	Вычислить сумму положительных элементов, расположенных в столбцах с четными номерами
2	Вычислить произведение отрицательных элементов, расположенных в строках с нечетными номерами
3	Вычислить сумму квадратов элементов из интервала [A,B], расположенных в строках с четными номерами
4	Определить количество элементов, больших заданного A и расположенных в строках с нечетными номерами
5	Вычислить сумму элементов, меньших заданного B и расположенных в столбцах с номерами, кратными 3
6	Вычислить произведение положительных элементов, расположенных в строках с номерами, кратными 4
7	Вычислить сумму квадратов отрицательных элементов, расположенных в столбцах с четными номерами
8	Определить количество элементов, не принадлежащих промежутку (A,B) и расположенных в столбцах с нечетными номерами
9	Вычислить сумму элементов, не меньших заданного D и расположенных в строках с четными номерами
10	Вычислить произведение элементов, не больших заданного F и расположенных в строках с нечетными номерами
11	Вычислить сумму квадратов положительных элементов, расположенных в столбцах с номерами, кратными 3
12	Определить количество отрицательных элементов, расположенных в строках с номерами, кратными 3

Продолжение таблицы 6.21

1	2
13	Вычислить сумму элементов, принадлежащих промежутку [A,B) и расположенных в столбцах с четными номерами
14	Вычислить произведение элементов, больших заданного X и расположенных в столбцах с нечетными номерами
15	Вычислить сумму квадратов элементов, меньших Y и расположенных в строках с нечетными номерами
16	Определить, сколько положительных элементов расположено в строках с нечетными номерами
17	Вычислить сумму отрицательных элементов, расположенных в столбцах с номерами, кратными 3
18	Вычислить произведение элементов, не принадлежащих интервалу (X, Y) и расположенных в строках с номерами, кратными 3
19	Вычислить сумму квадратов элементов, не меньших заданного Z и расположенных в столбцах с четными номерами
20	Найти количество элементов, не больших S и расположенных в строках с нечетными номерами
21	Вычислить сумму положительных элементов, расположенных в строках с четными номерами
22	Вычислить произведение отрицательных элементов, расположенных в строках с нечетными номерами
23	Вычислить сумму элементов, принадлежащих промежутку [X, Y] и расположенных в столбцах с номерами, кратными 3
24	Определить, сколько элементов, больших заданного T, расположено в строках с номерами, кратными 3
25	Вычислить сумму квадратов элементов, меньших заданного D и расположенных в столбцах с четными номерами
26	Вычислить произведение положительных элементов, расположенных в столбцах с нечетными номерами
27	Вычислить сумму квадратов отрицательных элементов, расположенных в строках с четными номерами
28	Вычислить количество элементов, не принадлежащих промежутку [B,C) и расположенных в строках с нечетными номерами
29	Вычислить сумму элементов, по абсолютной величине больших K и расположенных в столбцах с номерами, кратными 3
30	Вычислить произведение элементов, по абсолютной величине меньших G и расположенных в строках с номерами, кратными 3
31	В каждой нечетной по номеру строке матрицы найти минимальный элемент и вычислить произведение этих элементов
32	В каждом столбце матрицы найти произведение положительных элементов и вычислить сумму этих произведений
33	Определить количество столбцов матрицы, в которых больше трех положительных элементов
34	Определить количество строк матрицы, в которых суммы всех элементов отрицательные
35	В каждой строке матрицы найти самый левый отрицательный элемент и вычислить произведение этих элементов
36	Определить количество строк матрицы, в которых нет положительных элементов
37	Заменить в матрице элементы последней строки на произведение элементов соответствующих столбцов
38	В каждом столбце матрицы найти минимальный элемент и вычислить сумму этих элементов
39	В каждой строке матрицы найти произведение отрицательных элементов и вычислить сумму этих произведений
30	Заменить в матрице элементы предпоследней строки на минимальные элементы соответствующих столбцов
41	Определить количество строк матрицы, в которых произведение положительных элементов больше заданного B

Продолжение таблицы 6.21

1	2
42	В каждой строке матрицы найти самый правый положительный элемент и вычислить сумму этих элементов
43	Определить количество столбцов матрицы, в которых нет положительных элементов
44	Заменить в матрице элементы последнего столбца на суммы элементов соответствующих строк
45	В каждом столбце матрицы найти максимальный элемент и вычислить произведение этих элементов
46	В каждой четной по номеру строке матрицы найти минимальный элемент и вычислить сумму этих элементов
47	В каждом столбце матрицы найти сумму отрицательных элементов и вычислить произведение этих сумм
48	Определить количество строк матрицы, в которых произведение элементов положительное
49	Заменить в матрице элементы второго столбца на суммы положительных элементов соответствующих строк
50	В каждом столбце матрицы найти первый отрицательный элемент и вычислить сумму этих элементов
51	Определить количество строк матрицы, в которых все элементы отрицательные
52	Заменить в матрице элементы предпоследнего столбца на суммы элементов соответствующих строк
53	В каждом нечетном по номеру столбце матрицы найти минимальный элемент и вычислить произведение этих элементов
54	В каждой строке матрицы найти сумму положительных элементов и вычислить произведение этих сумм
55	Определить количество столбцов матрицы, в которых все элементы положительные
56	Заменить в матрице элементы предпоследней строки на количество положительных элементов соответствующих столбцов
57	В каждом столбце матрицы найти самый нижний положительный элемент и вычислить произведение этих элементов
58	Определить количество строк матрицы, в которых нет положительных элементов
59	Определить количество столбцов матрицы, в которых все элементы нулевые
60	В каждом столбце матрицы найти минимальный элемент и вычислить сумму этих элементов
61	Совпадает ли заданный массив хотя бы с одной строкой матрицы
62	Совпадает ли заданный массив хотя бы с одним столбцом матрицы
63	Есть ли в матрице строка, состоящая только из положительных элементов
64	Есть ли в матрице столбец, состоящий только из отрицательных элементов
65	Есть ли в матрице строка, сумма элементов которой равна заданному B
66	Есть ли в матрице столбец, сумма элементов которого больше заданного B
67	Есть ли в матрице строка, в которой все элементы равны между собой
68	Есть ли в матрице столбец, в котором все элементы равны между собой
69	Есть ли в матрице строка, в которой все элементы принадлежат промежутку [X,Y]
70	Есть ли в матрице столбец, в котором все элементы не принадлежат промежутку [X,Y]
71	Есть ли в матрице строка, в которой нет нулевых элементов
72	Есть ли в матрице столбец, в котором только два положительных элемента
73	Определить номер первой строки матрицы, в которой больше трех положительных элементов
74	Определить номер первого столбца матрицы, в котором произведение элементов отрицательное
75	Есть ли в матрице строка, в которой каждый элемент больше соответствующего элемента заданного массива
76	Есть ли в матрице строка, в которой на первом месте стоит максимальный элемент строки
77	Есть ли в матрице столбец, в котором на первом месте стоит максимальный элемент строки
78	Есть ли в матрице столбец, в котором на первом месте стоит минимальный элемент столбца

Окончание таблицы 6.21

1	2
79	Есть ли в матрице строка, в которой на последнем месте стоит минимальный элемент строки
80	Есть ли в матрице столбец, в котором на последнем месте стоит максимальный элемент столбца
81	Определить номер первой строки матрицы, в которой сумма элементов отрицательная
82	Определить номер первого столбца матрицы, в котором меньше трех нулей
83	Есть ли в матрице строка, в которой все элементы четные числа
84	Есть ли в матрице столбец, в котором все элементы нечетные числа
85	Есть ли в матрице строка, в которой сумма элементов отрицательная
86	Есть ли в матрице столбец, в котором нет нечетных чисел
87	Есть ли в матрице строка, элементы которой образуют неубывающую последовательность
88	Есть ли в матрице столбец, элементы которого образуют невозрастающую последовательность
89	Есть ли в матрице строка, сумма всех положительных элементов которой меньше заданного B
90	Есть ли в матрице столбец, произведение всех положительных элементов которого больше заданного B?

Контрольные вопросы и задания:

1. Что такое двумерный массив?
2. Зачем нужны двумерные массивы (приведите примеры)?
3. Что такое индексы двумерного массива?
4. Сколько индексов у двумерного массива?
5. Как выполнить описание двумерного массива?
6. Как обратиться к элементу двумерного массива?
7. Опишите массив вещественных чисел, который может содержать не более 5 строк и 10 столбцов.
8. Какие способы ввода двумерного массива существуют?
9. Нарисуйте фрагмент блок-схемы алгоритма ввода двумерного массива и опишите его.
10. Как и куда можно вывести двумерный массив?
11. Какие способы вывода двумерного массива существуют?
12. Нарисуйте фрагмент блок-схемы алгоритма вывода двумерного массива и опишите его.
13. Как выполнить описание квадратной матрицы?
14. Как выполнить описание прямоугольной матрицы?
15. Чем квадратная матрица отличается от прямоугольной?
16. Нарисуйте фрагмент блок-схемы алгоритма перебора элементов двумерного массива по строкам.
17. Нарисуйте фрагмент блок-схемы алгоритма перебора элементов двумерного массива по столбцам.

ЗАКЛЮЧЕНИЕ

Отметим основные результаты, предложенные и описанные в учебно-методическом пособии:

– показаны три основных способа описания алгоритмов: словесный, графический, на алгоритмическом языке;

– рассмотрен алфавит языка программирования C#, его ключевые слова, три группы операций (унарные, бинарные и тернарные), в табличной форме представлены приоритеты операций;

– представлены и подробно описаны 5 групп констант (логические, целые, вещественные, символьные, строковые) и приведены примеры их использования;

– дано объяснение понятия «управляющая последовательность), показан их вид и название, приведены примеры использования;

– показана необходимость двух типов комментариев (однострочных и многострочных) и примеры их правильного написания;

– классифицированы типы данных языка программирования C# и приведены примеры использования всех (семи) встроенных типов данных, а также дано понятие структурированных типов данных;

– приведены механизмы явного и неявного преобразования данных, а также конкретные примеры;

– даны понятия таким ключевым элементам языка программирования, как переменные, операнды, выражения и операторы;

– уделено особое внимание 3-м структурированным типам данных: одномерные массивы, двумерные массивы, многоступенчатые массивы;

– предложены различные варианты структуры программ на языке программирования C#;

– представлена единая методика описания структур основных операторов языка программирования, подробно описаны правила их выполнения и приведены примеры их использования;

– особое внимание уделено консольному выводу данных, в том числе и форматированному с большим количеством примеров;

– кратко описана платформа .NET;

– приведена структура и содержание лабораторной работы;

– дано понятие трем уровням представления данных в автоматизированных информационных системах (логический уровень, уровень хранения и физический уровень);

– описаны правила оформления отчета по лабораторной работе;

– изложены по методике процедуры выполнения пяти лабораторных работ;

– приведены различные способы написания программ одной и той же задачи;

– предложено множество различных задач к каждой лабораторной работе;

– приведены различные элементы ГОСТов в 9-ти приложениях для написания отчетов по лабораторным работам.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Анализ хозяйственной деятельности сельскохозяйственных предприятий [Текст] : методические указания по выполнению курсовых работ для студентов специальности 1-25 01 08 Бухгалтерский учет, анализ и аудит / Г. В. Миренкова [и др.] ; Министерство сельского хозяйства и продовольствия Республики Беларусь, Главное управление образования, науки и кадров, УО «Белорусская государственная сельскохозяйственная академия», Кафедра статистики и экономического анализа. – Горки : БГСХА, 2012. – 97 с.

2. Бахтизин, В. В. Метрология, стандартизация и сертификация в информационных технологиях [Текст] : учебное пособие для студентов учреждений высшего образования по направлению образования «Информатика и вычислительная техника» и по специальностям «Автоматизированные системы обработки информации», «Информационные технологии и управление в технологических системах»; допущено Министерством образования Республики Беларусь : в 2 ч. / В. В. Бахтизин, Л. А. Глухова. – Министерство образования Республики Беларусь, Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники», Факультет компьютерных систем и сетей, Кафедра программного обеспечения информационных технологий. – Минск : БГУИР, 2016. – Ч. 2. – 202 с.

3. Глухова, Л. А. Основы алгоритмизации и программирования : лабораторный практикум [Текст] : учебно-методическое пособие : рекомендовано УМО по образованию в области информатики и радиоэлектроники для специальностей, закрепленных за УМО по образованию в области информатики и радиоэлектроники / Л. А. Глухова, Е. П. Фадеева, Е. Е. Фадеева ; Министерство образования Республики Беларусь, Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники». – Минск : БГУИР, 2013. – 58 с.

4. ГОСТ 19.002-80. Единая система программной документации. Схемы алгоритмов и программ. Правила выполнения. – Введ. 1981-01-07. – М. : Изд-во стандартов, 1981. – 27 с.

5. ГОСТ Р 517721– 2001. Аппаратура радиоэлектронная бытовая. Входные и выходные параметры и типы соединений. Технические требования [Текст]. – Введ. 2002-01-01. – М. : Изд-во стандартов, 2001. – IV, 27 с.

6. Информатика [Текст] : практикум для студентов заочной формы обучения / Министерство сельского хозяйства и продовольствия Республики Беларусь, УО «Белорусский государственный аграрный технический университет», Кафедра «Прикладная информатика» ; сост. В. Севернева. – Минск : БГАТУ, 2011. – 104 с.

7. Лубашева, Т. В. Основы алгоритмизации и программирования [Текст] : учебное пособие : допущено Министерством образования Республики Беларусь для учащихся учреждений образования, реализующих образовательные программы среднего специального образования по специальности «Программное обеспечение информационных технологий» / Т. В. Лубашева, Б. А. Железко. – Минск : РИПО, 2016. – 378 с.

8. Основы алгоритмизации и программирования (Язык С/С++) : лабораторный практикум [Текст] : учебно-методическое пособие для всех специальностей I ступени высшего образования : рекомендовано УМО по образованию в области информатики и радиоэлектроники / С. А. Беспалов [и др.]. – Министерство образования Республики Беларусь, Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники», Факультет информационных технологий и управления, Кафедра вычислительных методов и программирования. – Минск : БГУИР, 2017. – 71 с.

9. Основы алгоритмизации и программирования [Текст] : конспект лекций для учащихся специальности 2-14 01 03 «Сети телекоммуникаций» / Министерство связи и информатизации Республики Беларусь, Учреждение образования «Высший государственный

колледж связи», Кафедра программного обеспечения сетей телекоммуникаций ; сост. Н. Ф. Мелешко. – Минск : УО ВГКС, 2013. – 87 с.

10. Подготовка, оформление и защита курсовых, дипломных работ и магистерских диссертаций [Текст] : методические рекомендации для студентов очной и заочной форм получения образования факультета международных экономических отношений и менеджмента / УО Федерации профсоюзов Беларуси, Международный университет «МИТСО» ; сост. : И. В. Говорень, Е. И. Иванова, Л. П. Кисель ; под ред. Л. П. Кисель. – Минск : МИТСО, 2012. – 64 с.

11. Прищепов, М. А. Экзамен по информатике : основы алгоритмизации и программирования [Текст] : справочное пособие / М. А. Прищепов, В. П. Степанцов, Е. В. Севернева. – Минск : ТетраСистемс, 2001. – 192 с.

12. Сурогатова, Т. В. Основы алгоритмизации и программирования. Среда программирования Delphi [Текст] : методические указания по выполнению лабораторных работ учащихся специальности 2-40 01 01 «Программное обеспечение информационных технологий» / Т. В. Сурогатова, О. В. Кричевцов. – Министерство образования Республики Беларусь, Филиал учреждения образования «Белорусский государственный технологический университет», «Витебский государственный технологический колледж». – Минск : БГТУ, 2016. – 88 с.

13. Экономическая оценка предприятия [Текст] : пособие по написанию курсовых и дипломных работ для студентов специальности 1-25 01 07 «Экономика и управление на предприятии» специализации 1-25 01 07 11 «Экономика и управление на предприятии промышленности» / Белкоопсоюз, Учреждение образования «Белорусский торгово-экономический университет потребительской кооперации», Кафедра экономики АПК. – авт.-сост. Л. Н. Дробышевский [и др.]. – Гомель : учреждение образования «Белорусский торгово-экономический университет потребительской кооперации», 2013. – 48 с.

ПРИЛОЖЕНИЕ А

Образец титульного листа лабораторной работы

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УО «ПОЛЕССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Инженерный факультет

Кафедра информационных технологий и интеллектуальных систем

Отчет
Лабораторная работа № 3
по дисциплине: «Основы алгоритмизации и программирования»
на тему: «Табулирование функций»

Выполнила:
студентка группы 21ИТ-1 Т. Ф. Байдук

Проверил:
к.э.н, доцент Л. П. Володько

ПИНСК 2021

ПРИЛОЖЕНИЕ Б

Образец оформления оглавления

ОГЛАВЛЕНИЕ

и т. д.

ПРИЛОЖЕНИЕ В

Примеры библиографического описания изданий

Таблица В.1 – Правила оформления списка использованных источников

Характеристика источника	Пример оформления
1	2
Один, два или три автора	Котаў, А. І. Гісторыя Беларусі і сусветная цывілізацыя / А. І. Котаў. – 2-е выд. – Мінск : Энцыклапедыкс, 2013. – 168 с.
	Шотт, А. В. Курс лекций по частной хирургии / А. В. Шотт, В. А. Шотт. – Минск : Асар, 2012. – 525 с.
	Чикатуева, Л. А. Маркетинг : учеб. пособие / Л. А. Чикатуева, Н. В. Третьякова ; под ред. В. П. Федько. – Ростов н/Д : Феникс, 2014. – 413 с.
	Дайнеко, А. Е. Экономика Беларуси в системе всемирной торговой организации / А. Е. Дайнеко, Г. В. Забавский, М. В. Василевская ; под ред. А. Е. Дайнеко. – Минск : Ин-т аграр. экономики, 2011. – 323 с.
Четыре и более авторов	Культурология : учеб. пособие для вузов / С. В. Лапина [и др.] ; под общ. ред. С. В. Лапиной. – 2-е изд. – Минск : ТетраСистемс, 2014. – 495 с.
	Комментарий к Трудовому кодексу Республики Беларусь / И. С. Андреев [и др.] ; под общ. ред. Г. А. Василевича. – Минск : Амалфея, 2000. – 1071 с.
	Основы геологии Беларуси / А. С. Махнач [и др.] ; НАН Беларуси, Ин-т геол. Наук ; под общ. ред. А. С. Махнача. – Минск, 2010. – 391 с.
Коллективный автор	Сборник нормативно-технических материалов по энергосбережению / Ком. по энергоэффективности при Совете Министров Республики Беларусь ; сост. А. В. Филипович. – Минск : Лоранж-2, 2004. – 393 с.
	Национальная стратегия устойчивого социально-экономического развития Республики Беларусь на период до 2020 г. / Нац. комис. по устойчивому развитию Респ. Беларусь ; редкол.: Л. М. Александрович [и др.]. – Минск : Юнипак, 2004. – 202 с.
	Военный энциклопедический словарь / М-во обороны Российской Федерации, Ин-т воен. истории ; редкол.: А. П. Горкин [и др.]. – М. : Большая рос. энцикл. : РИПОЛ классик, 2002. – 1663 с.
Многотомное издание	Гісторыя Беларусі: у 6 т. / рэдкал.: М. Касцюк (гал. рэд.) [і інш.]. – Мінск : Экаперспектыва, 2000–2005. – 6 т.
	Гісторыя Беларусі: у 6 т. / рэдкал.: М. Касцюк (гал. рэд.) [і інш.]. – Мінск : Экаперспектыва, 2000–2005. – Т. 3 : Беларусь у часы Рэчы Паспалітай (XVII–XVIII ст.) / Ю. Бохан [і інш.]. – 2004. – 343 с.; Т. 4 : Беларусь у складзе Расійскай імперыі (канец XVIII– пачатак XX ст.) / М. Біч [і інш.]. – 2005. – 518 с.
	Багдановіч, М. Поўны збор твораў : у 3 т. / М. Багдановіч. – 2-е выд. – Мінск : Беларус. навука, 2001. – 3 т.

1	2
Отдельный том в многотомном издании	Гісторыя Беларусі : у 6 т. / рэдкал.: М. Касцюк (гал. рэд.) [і інш.]. – Мінск : Экаперспектыва, 2000 – 2005. – Т. 3 : Беларусь у часы Рэчы Паспалітай (XVII–XVIII ст.) / Ю. Бохан [і інш.]. – 2004. – 343 с.
	Гісторыя Беларусі : у 6 т. / рэдкал.: М. Касцюк (гал. рэд.) [і інш.]. – Мінск : Экаперспектыва, 2000–2005. – Т. 4 : Беларусь у складзе Расійскай імперыі (канец XVIII–пачатак XX ст.) / М. Біч [і інш.]. – 2005. – 518 с.
	Багдановіч, М. Поўны збор твораў : у 3 т. / М. Багдановіч. – 2-е выд. – Мінск : Беларус. навука, 2001. – Т. 1 : Вершы, паэмы, пераклады, наследаванні, чарнавыя накіды. – 751 с.
Законы и законодательные материалы	Конституция Республики Беларусь 1994 года (с изменениями и дополнениями, принятыми на республиканских референдумах 24 ноября 1996 г. и 17 октября 2004 г.). – Минск : Амалфея, 2005. – 48 с.
	Конституция Российской Федерации : принята всенар. голосованием 12 дек. 1993 г. : офиц. текст. – М. : Юрист, 2005. – 56 с.
	О нормативных правовых актах Республики Беларусь : Закон Республики Беларусь от 10 янв. 2000 г. № 361-З : с изм. и доп. : текст по состоянию на 1 дек. 2004 г. – Минск : Дикта, 2004. – 59 с.
	Инвестиционный кодекс Республики Беларусь : принят Палатой представителей 30 мая 2001г. : одобр. Советом Респ. 8 июня 2001 г. : текст Кодекса по состоянию на 10 февр. 2001 г. – Минск : Амалфея, 2005. – 83 с.
Сборник статей, трудов	Информационное обеспечение науки Беларуси : к 80-летию со дня основания ЦНБ им. Я. Коласа НАН Беларуси : сб. науч. ст. / НАН Беларуси, Центр. науч. б-ка ; редкол.: Н. Ю. Березкина (отв. ред.) [и др.]. – Минск, 2004. – 174 с.
	Современные аспекты изучения алкогольной и наркотической зависимости : сб. науч. ст. / НАН Беларуси, Ин-т биохимии ; науч. ред. В. В. Лелевич. – Гродно, 2004. – 223 с.
Сборники без общего заглавия	Певзнер, Н. Английское в английском искусстве / Н. Певзнер ; пер. О. Р. Демидовой. – СПб. : Азбука-классика, 2004. – 318 с.
Материалы конференций	Глобализация, новая экономика и окружающая среда : проблемы общества и бизнеса на пути к устойчивому развитию : материалы 7 Междунар. конф. Рос. о-ва экол. экономики, Санкт-Петербург, 23–25 июня 2005 г. / С.-Петерб. гос. ун-т ; под ред. И. П. Бойко [и др.]. – СПб., 2005. – 395 с.
	Правовая система Республики Беларусь : состояние, проблемы, перспективы развития : материалы V межвуз. конф. студентов, магистрантов и аспирантов, Гродно, 21 апр. 2005 г. / Гродн. гос. ун-т ; редкол.: О. Н. Толочко (отв. ред.) [и др.]. – Гродно, 2005. – 239 с.

Продолжение таблицы В.1

1	2
Инструкция	Инструкция о порядке совершения операций с банковскими пластиковыми карточками : утв. Правлением Нац. банка Республики Беларусь 30.04.04: текст по состоянию на 1 дек. 2004 г. – Минск : Дикта, 2004. – 23 с.
	Инструкция по исполнительному производству : утв. М-вом юстиции Респ. Беларусь 20.12.04. – Минск : Дикта, 2005. – 94 с.
Учебно-методические материалы	Горбаток, Н. А. Общая теория государства и права в вопросах и ответах : учеб. пособие / Н. А. Горбаток. – М-во внутр. дел Респ. Беларуь, Акад. МВД. – Минск, 2005. – 183 с.
	Использование креативных методов в коррекционно-развивающей работе психологов системы образования : учеб.-метод. пособие : в 3 ч. / Акад. последиплом. образования ; авт.-сост. Н. А. Сакович. – Минск, 2004. – Ч. 2 : Сказкотерапевтические технологии. – 84 с.
	Корнеева, И. Л. Гражданское право : учеб. пособие : в 2 ч. / И. Л. Корнеева. – М. : РИОР, 2004. – Ч. 2. – 182 с.
	Философия и методология науки : учеб.-метод. комплекс для магистратуры / А. И. Зеленков [и др.] ; под ред. А. И. Зеленкова. – Минск : Изд-во БГУ, 2004. – 108 с.
Информационные издания	Реклама на рубеже тысячелетий : ретросп. библиогр. указ. (1998–2003) / М-во образования и науки Рос. Федерации, Гос. публич. науч.-техн. б-ка России ; сост.: В. В. Климова, О. М. Мещеркина. – М., 2004. – 288 с.
	Щадов, И. М. Технолого-экономическая оценка экологизации угледобывающего комплекса Восточной Сибири и Забайкалья / И. М. Щадов. – М. : ЦНИЭИуголь, 1992. – 48 с. – (Обзорная информация / Центр. науч.-исслед. ин-т экономики и науч.-техн. информ. угол. пром-сти)
Каталог	Каталог жесткокрылых (Coleoptera, Insecta) Беларуси / О. Р. Александрович [и др.] ; Фонд фундам. исслед. Респ. Беларусь. – Минск, 1996. – 103 с.
	Памятные и инвестиционные монеты России из драгоценных металлов, 1921–2003 : каталог-справочник / ред.-сост. Л. М. Пряжникова. – М. : ИнтерКрим-пресс, 2004. – 462 с.
Авторское свидетельство	Инерциальный волноблок : а. с. 1696865 СССР, МКИ5 G 01 C 13/00 / Ю. В. Дубинский, Н. Ю. Мордашова, А. В. Ференц. – Казан. авиац. ин-т. – № 4497433 ; заявл. 24.10.88 ; опубл. 07.12.91 // Открытия. Изобрет. – 1991. – № 45. – С. 28.
Патент	Способ получения сульфокатионита : пат. 6210 Респ. Беларусь, МПК7 C 08 J 5/20, C 08 G 2/30 / Л. М. Ляхнович, С. В. Покровская, И. В. Волкова, С. М. Ткачев. – Заявитель Полоц. гос. ун-т. – № a 0000011 ; заявл. 04.01.00 ; опубл. 30.06.04 // Афіцыйны бюл. / Нац. цэнтр інтэлектуал. уласнасці. – 2004. – № 2. – С. 174.

1	2
Стандарт	Безопасность оборудования. Термины и определения : ГОСТ ЕН 1070 – 2003. – Введ. 01.09.04. – Минск : Межгос. совет по стандартизации, метрологии и сертификации : Белорус. гос. ин-т стандартизации и сертификации, 2004. – 21 с.
Нормативно-технические документы	Национальная система подтверждения соответствия Республики Беларусь. Порядок декларирования соответствия продукции. Основные положения – Нацыянальная сістэма пацвярджэння адпаведнасці Рэспублікі Беларусь. Парадак дэкларавання адпаведнасці прадукцыі. Асноўныя палажэнні: ТКП 5.1.03-2004. – Введ. 01.10.04. – Минск : Белорус. гос. ин-т стандартизации и сертификации, 2004. – 9 с.
	Государственная система стандартизации Республики Беларусь. Порядок проведения экспертизы стандартов : РД РБ 03180.53-2000. – Введ. 01.09.00. – Минск : Госстандарт : Белорус. гос. ин-т стандартизации и сертификации, 2000. – 6 с.
Препринт	Губич, Л. В. Подходы к автоматизации проектно-конструкторских работ в швейной промышленности / Л. В. Губич. – Минск, 2004. – 40 с. – (Препринт / Акад. наук Беларуси, Ин-т техн. кибернетики; № 3)
	Прогноз миграции радионуклидов в системе водосбор – речная сеть / В. В. Скурат [и др.]. – Минск, 2004. – 51 с. – (Препринт / НАН Беларуси, Объед. ин-т энергет. и ядер. исслед. – Сосны ; ОИЭЯИ-15).
Отчет о НИР	Разработка и внедрение диагностикума аденовирусной инфекции птиц : отчет о НИР (заключ.) / Всесоюз. науч.-исслед. ветеринар. ин-т птицеводства ; рук. темы А.Ф. Прохоров. – М., 2009. – 14 с. – № ГР 01870082247.
	Комплексное (хирургическое) лечение послеоперационных и рецидивных вентральных грыж больших и огромных размеров : отчет о НИР / Гродн. гос. мед. ин-т ; рук. В. М. Колтонюк. – Гродно, 2012. – 42 с. – № ГР 1993310.
Депонированные научные работы	Влияние деформации и больших световых потоков на люминесценцию монокристаллов сульфида цинка с микропорами / В.Г. Клюев [и др.] ; Воронеж. ун-т. – Воронеж, 2003. – 14 с. – Деп. в ВИНИТИ 10.06.93, № 1620-В93 // Журн. приклад. спектроскопии. – 2003. – Т. 59, № 3–4. – С. 368.
	Сагдиев, А. М. О тонкой структуре субарктического фронта в центральной части Тихого океана / А. М. Сагдиев. – Рос. акад. наук, Ин-т океанологии. – М., 2012. – 17 с. – Деп. в ВИНИТИ 08.06.92, № 1860-82 // РЖ: 09. Геофизика. – 2012. – № 11/12. – 11В68ДЕП. – С. 9.
	Широков, А. А. Исследование возможности контроля состава гальванических сред абсорбционно-спектроскопическим методом / А. А. Широков, Г. В. Титова. – Рос. акад. наук, Ульян. фил. ин-та радиотехники и электроники. – Ульяновск, 1993. – 12 с. – Деп. в ВИНИТИ 09.06.93, № 1561-В93 // Журн. приклад. спектроскопии. – 2013. – № 3–4. – С. 368.

1	2
Диссертация	Анисимов, П. В. Теоретические поблемы правового регулирования защиты прав человека : дис. …д-ра юрид. наук : 12.00.01 / П. В. Анисимов. – Н.Новгород, 2005. – 370 л.
	Лук'янюк, Ю. М. Сучасная беларуская філасофская тэрміналогія (семантычныя і структурныя аспекты) : дыс. ... канд. філал. навук : 10.02.01 / Ю. М. Лук'янюк. – Мінск, 2003. – 129 л.
Архивные материалы	1. Архив Гродненского областного суда за 2012 г. – Дело № 4/8117. 2. Архив суда Центрального района г.Могилева за 2014 г. – Уголовное дело № 2/1577.
	Центральный исторический архив Москвы (ЦИАМ). 1. Фонд 277. – Оп. 1. – Д. 1295 – 1734. Дела о выдаче ссуды под залог имений, находящихся в Могилевской губернии (имеются планы имений) 1884–1918 гг. 2. Фонд 277. – Оп. 1. – Д. 802 – 1294, 4974–4978, 4980–4990, 4994–5000, 5002–5013, 5015–5016. Дела о выдаче ссуды под залог имений, находящихся в Минской губернии (имеются планы имений) 1884–1918 гг. 3. Фонд 277. – Оп. 2, 5, 6, 7, 8.
Электронные ресурсы	Театр [Электронный ресурс]: энциклопедия : по материалам изд-ва «Большая российская энциклопедия» : в 3 т. – Электрон. дан. (486 Мб). – М. : Кордис & Медиа, 2013. – Электрон. опт. диски (CD-ROM) : зв., цв. – Т. 1 : Балет. – 1 диск ; Т. 2 : Опера. – 1 диск ; Т. 3 : Драма. – 1 диск.
	Регистр СНГ – 2013 : промышленность, полиблокия, торговля, ремонт, транспорт, строительство, сельское хозяйство [Электронный ресурс]. – Электрон. текстовые дан. и прогр. (14 Мб). – Минск : Комлев И. Н., 2013. – 1 электрон. опт. диск (CD-ROM).
Ресурсы удаленного доступа	Национальный Интернет-портал Республики Беларусь [Электронный ресурс] / Нац. центр правовой информ. Респ. Беларусь. – Минск, 2005. – Режим доступа: http://www.pravo.by. – Дата доступа: 25.01.2015.
	Proceeding of mini – symposium on biological nomenclature in the 21st centry [Electronic resource] / Ed. J.L. Reveal. – College Park M.D., 1996. – Mode of access: http://www.inform.ind.edu/PBIO/brum.html. – Date of access: 14.09.2005.
Автореферат диссертации	Иволгина, Н. В. Оценка интеллектуальной собственности: на примере интеллектуальной промышленной собственности : автореф. дис. …канд. экон. наук : 08.00.10 ; 08.00.05 / Н. В. Иволгина. – Рос. экон. акад. – М., 2005. – 26 с.
	Шакун, Н. С. Кірыла-Мяфодзіеўская традыцыя на Тураўшчыне (да праблемы лакальных тыпаў старажытнаславянскай мовы) : аўтарэф. дыс. ... канд. філал. навук : 10.02.03 / Н. С. Шакун. – Беларус. дзярж. ун-т. – Мінск, 2005. – 16 с.

ПРИЛОЖЕНИЕ Г

Образец оформления таблиц

Таблица 1.1 – Возрастная структура производственного оборудования в промышленности (в %)

Год	Все оборудование на конец года	Из него в возрасте, лет				Средний возраст, лет
		До 5	6–10	11–20	Свыше 20	
2013	100	35,5	28,7	25,1	10,7	9,5
2014	100	29,4	28,3	27,3	15,0	10,8
2015	100	10,1	29,8	36,9	23,2	14,3
2016	100	7,2	27,5	39,5	25,8	15,2
2017	100	5,2	24,1	42,2	29,0	16,1
2018	100	5,4	20,1	44,2	31,6	17,0
2019	100	4,1	15,2	45,8	34,8	17,9
2020	100	4,7	10,6	46,5	38,2	18,7
2021	100	5,7	7,6	45,1	41,6	19,4

При разрыве таблицы:

Таблица 1.1 – Возрастная структура производственного оборудования в промышленности (в %)

Год	Все оборудование на конец года	Из него в возрасте, лет				Средний возраст, лет
		До 5	6–10	11–20	Свыше 20	
1	2	3	4	5	6	7
2013	100	35,5	28,7	25,1	10,7	9,5
2014	100	29,4	28,3	27,3	15,0	10,8
2015	100	10,1	29,8	36,9	23,2	14,3
2016	100	7,2	27,5	39,5	25,8	15,2
2017	100	5,2	24,1	42,2	29,0	16,1

Разрыв страницы:

Продолжение таблицы 1.1

1	2	3	4	5	6	7
2018	100	5,4	20,1	44,2	31,6	17,0
2019	100	4,1	15,2	45,8	34,8	17,9
2020	100	4,7	10,6	46,5	38,2	18,7
2021	100	5,7	7,6	45,1	41,6	19,4

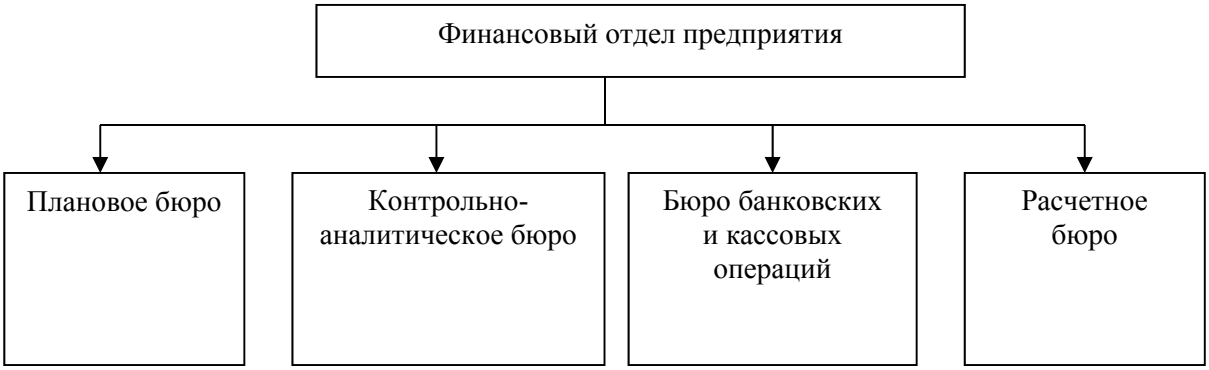
ПРИЛОЖЕНИЕ Д

Образец оформления рисунков

текст

Примерная структура финансового отдела предприятия представлена на рисунке 1.1.

пустая строка



пустая строка

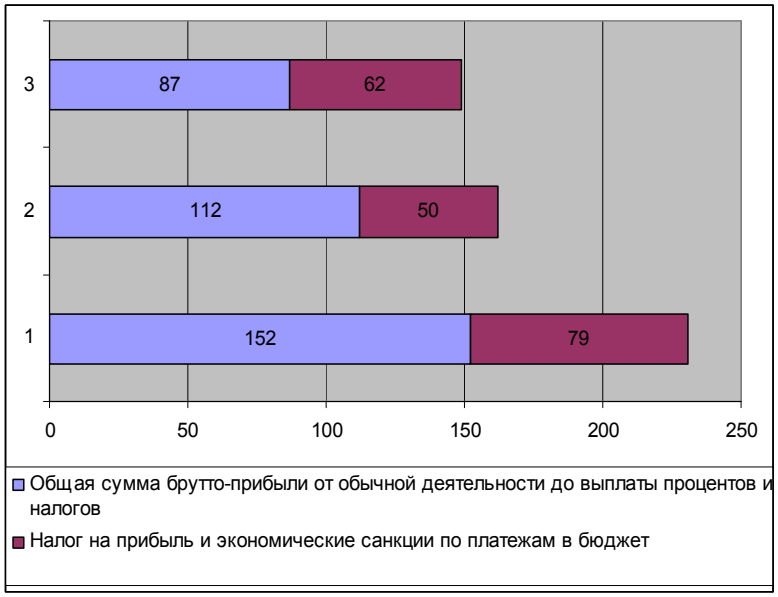
Рисунок 1.1 – Примерная структура финансового отдела предприятия

пустая строка

текст

Уровень налогового изъятия прибыли практически не изменился, но возросли экономические санкции по платежам в бюджет (рисунок 1.2).

пустая строка



пустая строка

Рисунок 1.2 – Соотношение платежей в бюджет и брутто-прибыли

ПРИЛОЖЕНИЕ Е

Образец оформления формул и расчета на их основе

текст

Фонд рабочего времени зависит от численности работающих, количества отработанных дней одним рабочим в среднем за год и средней продолжительности рабочего дня. Формула 2.8 для расчета фонда рабочего времени будет иметь следующий вид:

пустая строка

$$\mathbf{ФРВ = ЧР \times Д \times П}, \qquad (2.8)$$

пустая строка

где ФРВ – фонд рабочего времени;
ЧР – численность рабочих;
Д – количество рабочих дней в году;
П – продолжительность рабочего дня.

пустая строка

текст

Оформление расчета по формуле

Прирост прибыли в день рассчитывается по формуле (ППД):

$$ППД = Qкл * (C1 – C2), \qquad (1.4)$$

где Q – количество клиентов в день при машинной обработке, чел. (150 чел.);
C1 – затраты при ручном способе оформлении документов (4, 64 руб.);
C2 – затраты при машинном способе оформлении документов (3,54 руб.).
Он составит: ППД = 150 * (4,64 – 3,54) = 164 руб.

ПРИЛОЖЕНИЕ Ж

Образец оформления приложения

Тарификация сотрудников инженерного факультета и экономики и финансов

Таблица Ж.1 – Тарифные разряды и значения тарифных коэффициентов

Тарифный разряд	Тарифный коэффициент
1	2
1	1,00
2	1,07
3	1,14
4	1,21
5	1,29
6	1,38
7	1,47
8	1,57
9	1,68
10	1,79
11	1,91
12	2,03
13	2,17
14	2,31
15	2,47
16	2,63
17	2,81
18	3,00

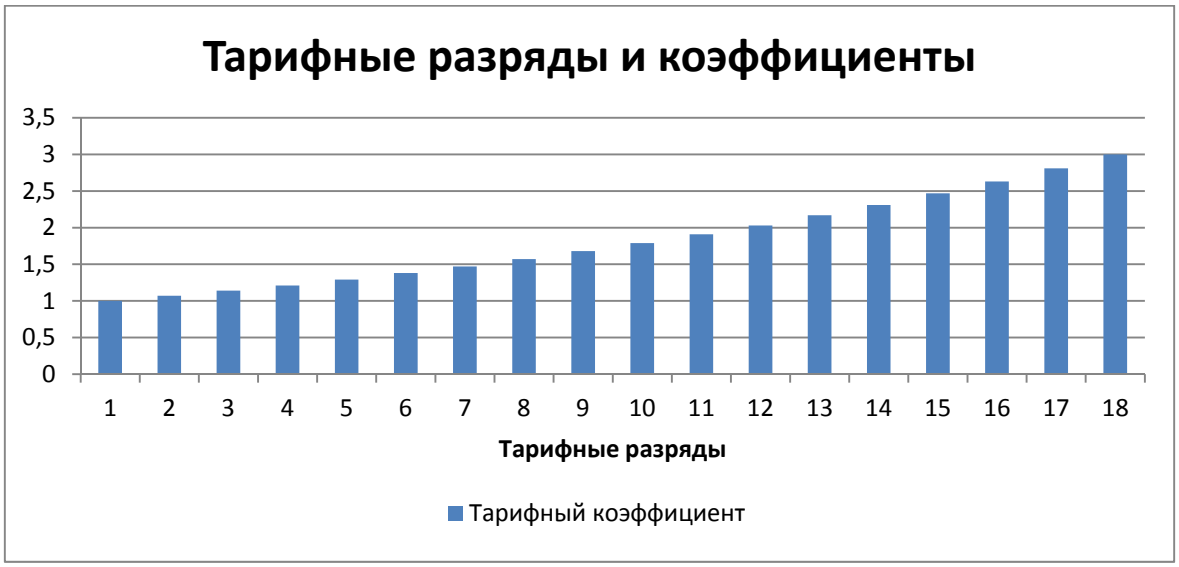


Рисунок Ж.1 – Диаграмма соответствия тарифных разрядов и тарифных коэффициентов

Таблица Ж.2 – Персональная тарификация сотрудников кафедры информационных технологий и интеллектуальных систем

Ф.И.О.	Должность	Тарифный разряд	Тарифный коэффициент	Тарифный оклад, руб.
1	2	3	4	5
1. Базака Людмила Николаевна	Ассистент	9	1,68	327,60
2. Вишняков Юрий Михайлович	Доцент	11	1,91	372,45
3. Володько Людвик Павлович	Доцент	11	1,91	372,45
4. Дегтярева Инна Ивановна	Ст. преподаватель	10	1,79	349,05
5. Иванцова Анастасия Олеговна	Специалист по организации и обеспечению образовательного процесса	4	1,21	235,95
6. Кисель Татьяна Васильевна	Ст. преподаватель	10	1,79	349,05
7. Клаченков Владислав Андреевич	Ассистент	9	1,68	327,60
8. Лешкевич Анна Петровна	Специалист по организации и обеспечению образовательного процесса	4	1,21	235,95
9. Липский Ярослав Николаевич	Ассистент	9	1,68	327,60
10. Минюк Ольга Николаевна	Доцент	11	1,91	372,45
11. Митянок Вячеслав Владимирович	Доцент	11	1,91	372,45
12. Павлов Павел Александрович	Доцент	11	1,91	372,45
13. Пигаль Анастасия Сергеевна	Ассистент	9	1,68	327,60
14. Пигаль Павел Борисович	Ст. преподаватель	10	1,79	349,05
15. Панкратенко Наталья Георгиевна	Специалист 1 категории по организации и обеспечению образовательного процесса	6	1,38	269,10
16. Романова Марина Александровна	Зав. кафедрой	13	2,17	423,15
17. Руско Дарья Игоревна	Ассистент	9	1,68	327,60
18. Сидская Ольга Владимировна	Ст. преподаватель	10	1,79	349,05
19. Товстыка Виктор Станиславович	Доцент	11	1,91	372,45
20. Штепа Владимир Николаевич	Профессор	12	2,03	395,85
21. Янковский Игорь Анатольевич	Доцент	11	1,91	372,45

Таблица Ж.3 – Персональная тарификация сотрудников кафедры банкинга и финансовых рынков

Ф.И.О.	Должность	Тарифный разряд	Тарифный коэффициент	Тарифный оклад, руб.
1	2	3	4	5
1. Давыдова Наталья Леонтьевна	Доцент	11	1,91	372,45
2. Золотарёва Ольга Александровна	Профессор	12	2,03	395,85
3. Игнатьева Елена Степановна	Ассистент	9	1,68	327,60
4. Козловская Евгения Евгеньевна	Преподаватель-стажёр	8	1,57	306,15
5. Лопух Юлия Ивановна	Ассистент	9	1,68	327,60
6. Лукашевич Валентина Алексеевна	Зав. кафедрой	13	2,17	423,15
7. Матяс Александр Анатольевич	Доцент	11	1,91	372,45
8. Петрукович Наталья Геннадьевна	Доцент	11	1,91	372,45
9. Сергеюк Валентина Степановна	Ассистент	9	1,68	327,60
10. Синкевич Алина Ивановна	Ст. преподаватель	10	1,79	349,05
11. Теляк Оксана Александровна	Доцент	11	1,91	372,45
12. Хворова Елена Ивановна	Специалист 1 категории по организации и обеспечению образовательного процесса	6	1,38	269,10
13. Хрусь Елена Александровна	Ст. преподаватель	10	1,79	349,05
14. Штепа Алена Григорьевна	Ассистент	9	1,68	327,60
15. Ярутич Наталья Степановна	Специалист 2 категории по организации и обеспечению образовательного процесса	5	1,29	251,55

Таблица Ж.4 – Персональная тарификация сотрудников кафедры финансового менеджмента

Ф.И.О.	Должность	Тарифный разряд	Тарифный коэффициент	Тарифный оклад, руб.
1	2	3	4	6
1. Бухтик Марина Игоревна	Доцент	11	1,91	372,45
2. Бондарь Алеся Геннадьевна	Ассистент	9	1,68	327,60
3. Вакулич Евгения Анатольевна	Специалист 1 категории по организации и обеспечению образовательного процесса	6	1,38	269,10
4. Галкина Марина Николаевна	Ассистент	9	1,68	327,60
5. Голикова Анна Сергеевна	Доцент	11	1,91	372,45
6. Данилкова Светлана Анатольевна	Доцент	11	1,91	372,45
7. Киевич Александр Владимирович	Профессор	12	2,03	395,85
8. Кислюк Татьяна Васильевна	Специалист 1 категории по организации и обеспечению образовательного процесса	6	1,38	269,10
9. Клещева Светлана Александровна	Ст. преподаватель	10	1,79	349,05

Окончание таблицы Ж.4

1	2	3	4	5
10. Конончук Ирина Анатольевна	Доцент	11	1,91	372,45
11. Ливенский Валентин Михайлович	Доцент	11	1,91	372,45
12. Лисовский Максим Иванович	Зав. кафедрой	13	2,17	423,15
13. Лобан Тамара Николаевна	Ст. преподаватель	10	1,79	349,05
14. Невдах Сергей Васильевич	Ст. преподаватель	10	1,79	349,05
15. Пригодич Ирина Александровна	Доцент	11	1,91	372,45
16. Ржевская Татьяна Александровна	Доцент	11	1,91	372,45
17. Самоховец Мария Павловна	Доцент	11	1,91	372,45
18. Чернорук Светлана Васильевна	Ст. преподаватель	10	1,79	349,05

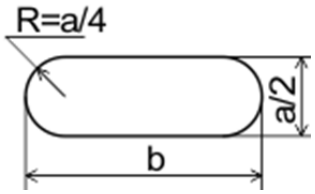
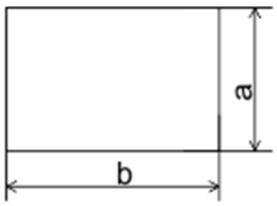
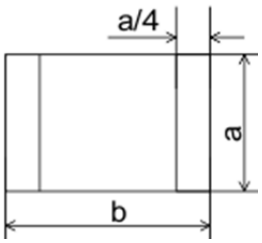
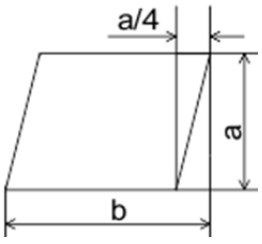
Рисунок Ж.2 – Удельный вес тарифных окладов сотрудников кафедры финансового менеджмента

ПРИЛОЖЕНИЕ К

Основные графические символы для составления блок-схем алгоритмов

Размер *a* должен выбираться из ряда 10, 15, 20 мм. Допускается увеличивать размер *a* на число, кратное 5 мм. Размер *b* равен 1,5*a*. Основным направлением потока в схемах алгоритмов принято направление сверху вниз, слева направо. Если линии потока идут в основном направлении и не имеют изломов, стрелками их можно не обозначать. В остальных случаях направление линии потока обозначать стрелкой обязательно. Записи внутри символа должны быть представлены так, чтобы их можно было читать слева направо и сверху вниз, независимо от направления потока. В схеме символу может быть присвоен идентификатор, который должен помещаться слева над символом. Допускается краткая информация о символе (описание, уточнение или другие перекрестные ссылки для более полного понимания функции данной части схемы). Описание символа должно помещаться справа над символом. В случае необходимости слияния линий потока место слияния должно быть обозначено точкой или символом 0.

Таблица К.1 – Основные блоки для составления алгоритмов

Название	Обозначение	Описание
1	2	3
1. Терминатор		Начало, конец, прерывание процесса обработки данных или выполнения программы
2. Процесс		Выполнение операции или группы операций, в результате которых изменяется значение, форма представления или расположение данных
3. Предопределенный процесс		Использование ранее созданных и отдельно описанных алгоритмов или программ
4. Ввод-вывод		Преобразование данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод)

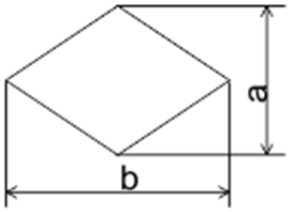
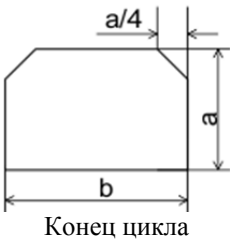
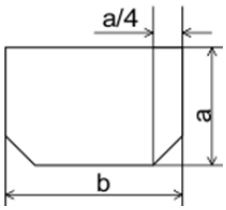
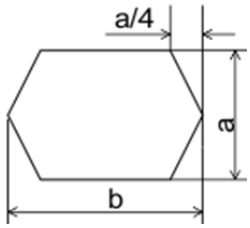
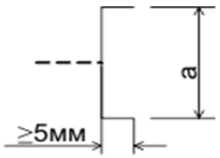
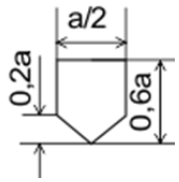
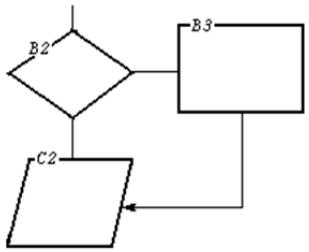
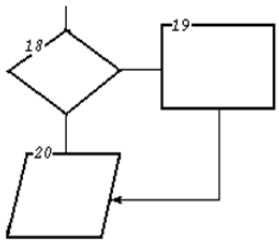
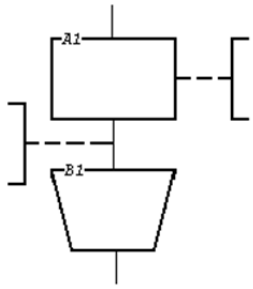
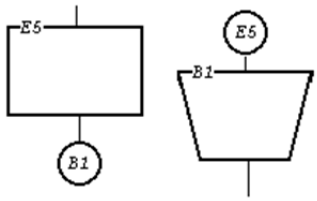
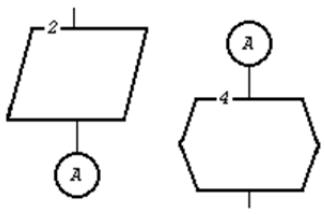
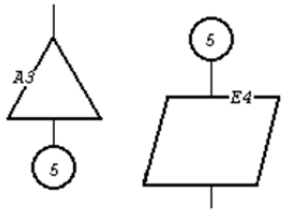
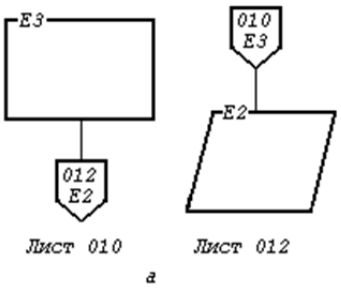
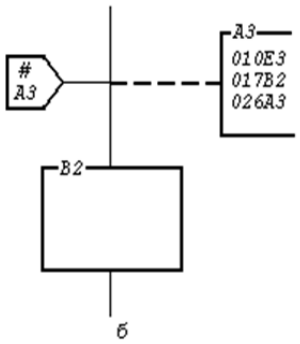
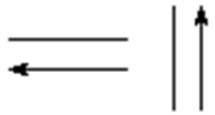
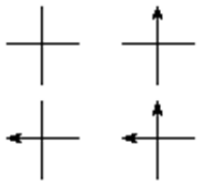
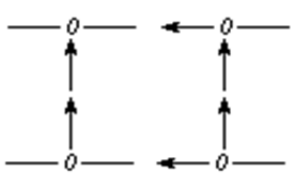
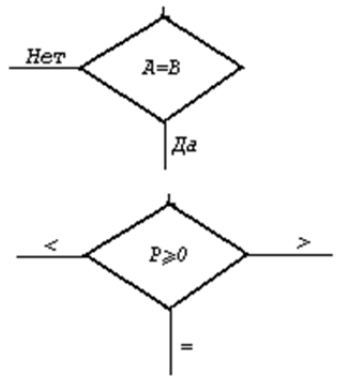
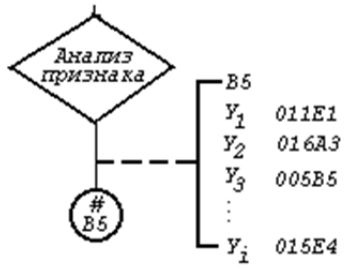
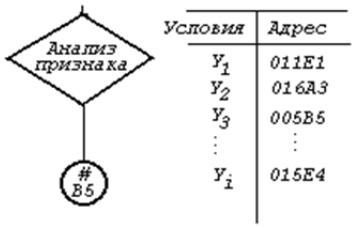
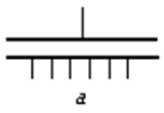
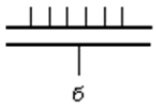
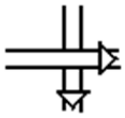
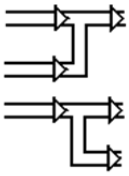
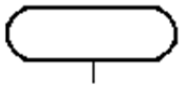
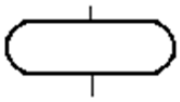
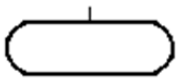
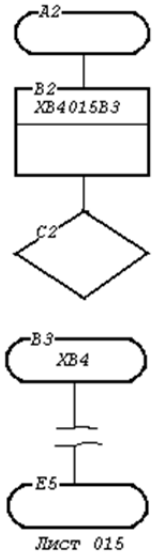
1	2	3
5. Решение		Выбор направления выполнения алгоритма или программы в зависимости от некоторых переменных условий. Блок решения имеет один вход и по крайней мере два выхода
6. Границы цикла	Начало цикла  Конец цикла 	Символ, состоящий из двух частей, отображает начало и конец цикла. Обе части символа имеют один идентификатор. Условия для инициализации, приращения, завершения и т. д. помещаются внутри символа в начале или в конце в зависимости от расположения операции, проверяющей условие
7. Модификатор		Выполнение операций, меняющих команды или группу команд, с целью воздействия на некоторую последующую функцию (установка переключателя, модификация регистра, инициализация программы)
8. Комментарий		Пояснение к элементу схемы (или линии связи)
9. Соединитель		При большой насыщенности схемы отдельные линии потока между удаленными символами допускается обрывать. При этом в конце (начале) обрыва должен быть помещен символ «Соединитель». Внутри блока соединителя указывается имя уникального идентификатора
10. Межстраничный соединитель		Связывание линией потока символов, которые находятся на разных листах. Первая строка внутри межстраничного соединителя определяет номер листа, вторая – идентификатор символа

Таблица К.2 – Основные правила применения символов

Фрагмент схемы	Содержание обозначения	Правила применения
1	2	3
	Возможные варианты обозначения символов в схемах: B2, B3, C3 – координаты зоны листа, в которой размещен символ	Координаты зоны символа или порядковый номер проставляют в верхней части символа в разрыве его контура
	18, 19, 20 – порядковые номера символов на схеме	
	Комментарий	Применяется, если пояснение не помещается внутри символа (для пояснения характера параметров, особенностей процесса, линий потока и др.). Комментарий записывают параллельно основной надписи. Комментарий помещают в свободном месте схемы на данном листе и соединяют с поясняемым символом
	Страничный соединитель: E5, B1, A, 5 – идентификаторы соединителя в виде: буквы и цифры (координаты зоны листа):	При большой насыщенности схемы символами отдельные линии потока между удаленными друг от друга символами допускается обрывать. При этом в конце (начале) обрыва должен быть помещен символ «Соединитель»
	буквы	
	цифры	

1	2	3
	Межстраничный соединитель: первая строка внутри межстраничного соединителя определяет номер листа, вторая – координату символа	а) связывание линией потока символы находятся на разных листах. *Примечание.* При изготовлении схем с помощью ЭВМ допускается указывать рядом с обрывом линии потока адресные ссылки без использования символов «Соединитель» и «Межстраничный соединитель»
	А3 – определяет зону на данном листе, где расположен символ «Комментарий» 010Е3 – определяет номер листа и зону расположения, связываемые с символом Е3	б) и в случае связи некоторого символа со многими другими символами, расположенными на разных листах, на входе этого символа помещают один символ «Межстраничный соединитель», внутри которого на первой строке помещают знак #, а на второй строке – координаты символа «Комментарий». Внутри символа «Комментарий» указывают номера страниц и координаты символов, связанных с поясняемым символом
	Линии потока	Применяют для указания направления линии потока: - можно без стрелки, если линия направлена слева направо и сверху вниз; - со стрелкой – в остальных случаях
	Излом линии под углом 90 градусов	Обозначает изменение направление потока
	Пересечение линий потока	Применяется в случае пересечения двух несвязанных потоков
	Слияние линий потока: место слияний потока обозначено точкой	Применяется в случае слияния линий потока, каждая из которых направлена к одному и тому же символу на схеме
	Место слияний потока обозначено цифрой 0	Место слияния линий потока допускается обозначать точкой или цифрой 0

1	2	3
	Возможные варианты отображения решения: A = B, $P \geqslant 0$ – условия решений; A, B, P – параметры	При числе исходов не более трех признак условия решения (Да, Нет, = , < , >) проставляют над каждой выходящей линией потока или справа от линии потока
	*Yi* – условие *i*-го исхода; 011E1, 016A3, 005B5, 015E4 – адреса исходов. Структура адреса имеет вид: XXX XX; | | координата символа; | номер листа схемы	При числе исходов более трех условие исхода проставляется в разрыве линии потока. Адрес исхода проставляется в продолжении условия исхода и отделяется от него пробелом
	B5 – знак, указывающий, что условия решения даются в виде таблицы или символа «Комментарий», расположенный на данном листе в зоне B5	В символе «Соединитель» указывают координату зоны, куда должна помещаться таблица или символ «Комментарий»
		В таблице (в символе «Комментарий») приводят адреса всех переходов
	Параллельные действия: начало	Применяется в случае одновременного выполнения операций, отображаемых несколькими символами
	конец	При этом в случае а изображается одна входная, а в случае б – одна выходная линия потока

1	2	3
	Взаимодействие материальных потоков	Применяют: при пересечении материальных потоков
		при объединении материальных потоков; при разветвлении материальных потоков
	Начало, прерывание и конец алгоритма или программы: пуск	Символы применяют в начале схемы алгоритма или программы, в случае прерывания и в конце. Внутри символов «Пуск» – «Останов» может указываться наименование действия или идентификатор программы
	Прерывание	
	Останов	
	Детализация некоторой программы, представленной в данной схеме одним символом: XB4 – идентификатор программы; 015 – номер листа, где проведено начало детализируемой программы; B3 – координата зоны листа	Применяется (в отличие от случая, когда применяется символ «Предопределенный процесс») для детализации в составе данной схемы программы. Детализируемая программа начинается и заканчивается символом «Пуск» – «Останов». Внутри символа, посредством которого детализируется программа, проводят горизонтальную линию. В данном примере детализируемая программа представлена посредством символа «Процесс». Слева над горизонтальной линией помещается идентификатор детализируемой программы, а справа – номер листа и координата зоны, где размещен символ «Пуск» – «Останов». Внутри символа «Пуск» – «Останов», обозначающее начало детализируемой программы, указывается идентификатор данной программы

Учебное издание

Володько Людвик Павлович

Основы алгоритмизации и программирования

Часть 1

Учебно-методическое пособие

Ответственный за выпуск Ю. В. Чечун

Редактор Т. И. Сакович
Корректор Ю. В. Цвикевич

Подписано в печать 12.11.2021 г. Формат 60×84/8.
Бумага офсетная. Гарнитура «Таймс». Ризография.
Усл. печ. л. 17,55. Уч.-изд. л. 7,80.
Тираж 82 экз. Заказ № 295.

Отпечатано в редакционно-издательском отделе
Полесского государственного университета
225710, г. Пинск, ул. Днепровской флотилии, 23.