**Лабораторная работа №1**

**«ВВЕДЕНИЕ В ЭТАПЫ РАЗРАБОТКИ ПРОГРАММ»**

**Задание.** Составить таблицу спецификаций, словесное описание алгоритма (по желанию), блок-схему, таблицу тестов, программу на языке C++ и оттестировать программы для 3-х задач из групп A, Б и В по указанию преподавателя в соответствии с индивидуальным вариантом. При этом при решении заданий необходимо соблюдать следующие требования:

* в задаче A использовать оператор if
* в задаче Б использовать оператор switch
* в задаче B НЕ использовать конструкции библиотеки STL за исключением типа данных string и работы с устройствами ввода-вывода.

**Основная цель:** научиться оформлять отчёт и осуществлять тестирование и компиляцию программы с использованием средств разработки.Поэтому, чтобы облегчить старт, для некоторых вариантов на сайте курса есть примеры кода (которые могут содержать ошибки).

Ожидается, что в ходе выполнения первой лабораторной студент:

1) Напишет сам или использует готовые коды для создания простых программ: объявление, инициализация и использование переменных различных типов; базовые команды для работы с устройствами ввода вывода; вызов библиотечных функций.

2) Изучит разные реализации разветвляющихся и циклических процессов с использованием парадигмы структурного программирования.

3) Научится оформлять отчёт.

4) Научится рисовать блок-схемы.

**Оформление отчёта:** правила оформления и пример приведен в соответствующих разделах методички.

**Критерии оценки**: оценка «зачтено» при условии выполнения всех заданий требуемыми способами.

Оглавление

[I Технология разработки программ на языке высокого уровня 3](#_Toc144729453)

[Этапы разработки программ на языке высокого уровня 3](#_Toc144729454)

[Виды вычислительных процессов 6](#_Toc144729455)

[Пример алгоритма разветвляющегося процесса 7](#_Toc144729456)

[Пример алгоритма циклического процесса 9](#_Toc144729457)

[II Программа на языке C++. 12](#_Toc144729458)

[Краткие сведения о языке С++ 12](#_Toc144729459)

[Транслятор и компоновщик 13](#_Toc144729460)

[Книги и электронные ресурсы 13](#_Toc144729461)

[Элементы языка С++ 14](#_Toc144729462)

[Вывод на консоль 16](#_Toc144729463)

[Точка входа 17](#_Toc144729464)

[Переменные 17](#_Toc144729465)

[Ввод с консоли 19](#_Toc144729466)

[Выражения и операции 20](#_Toc144729467)

[Операторы 22](#_Toc144729468)

[Условия 23](#_Toc144729469)

[Оператор if 23](#_Toc144729470)

[Оператор switch 25](#_Toc144729471)

[Оператор do while 27](#_Toc144729472)

[Оператор while 29](#_Toc144729473)

[Оператор for 30](#_Toc144729474)

[Массивы 32](#_Toc144729475)

[III Индивидуальные задания 35](#_Toc144729476)

[Группа A. Условное ветвление 35](#_Toc144729477)

[Группа Б. Выбор из нескольких возможностей 36](#_Toc144729478)

[Группа В. Циклический процесс, массивы и строки 38](#_Toc144729479)

[IV Оформление отчёта 39](#_Toc144729480)

[Правила оформления 39](#_Toc144729481)

[Создание блок-схемы в ворде 40](#_Toc144729482)

[Пример выполнения лабораторной работы 40](#_Toc144729483)

# I Технология разработки программ на языке высокого уровня

## Этапы разработки программ на языке высокого уровня

Разработка программы на языке высокого уровня никогда не должна начинаться собственно с программирования. При таком подходе неизбежны ошибки в программах, сложность отладки и внесения изменений в программу, что часто и происходит, так как заказчик, только увидев работающую программу, может оценить, что ему надо и чего не хватает в программе.

Поэтому любую, даже самую простую программу разрабатывают на основе *структурного подхода*, который предполагает использование наиболее простых структур как при проектировании, так и при программировании и отладке программ.

Разработка программы состоит из следующих этапов:

**1. Внешнее проектирование** (определение общей структуры и взаимодействия функций). На этом этапе рассматривается возможность разбиения задачи на подзадачи, каждая из которых решает некоторую законченную часть всей задачи и определяется иерархия этих подзадач (какая подзадача вызывает для выполнения другие подзадачи). Одна из подзадач всегда является основной, *главной.*

Кроме того, на этом этапе выясняется, какие данные нужно передать вызываемой подзадаче и какие данные получить от нее. Для каждой подзадачи описывается своя таблица внешних спецификаций. Если задача простая, то этот этап можно опустить.

**2. Составление внешних спецификаций** – приведение условия задачи к некоторому стандарту, что позволяет задачи из различных предметных областей описывать более или менее унифицировано. Таким стандартом для простых программ или отдельных функций являются *внешние спецификации. Спецификации* определяют входные и выходные данные задачи и оформляются в виде таблицы вида (Табл.1)

Таблица 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Имя | Назначение | Тип | Вх/Вых. | Диапазон |

*Входные величины* – это те, значения которых необходимо задать в начале решения задачи. *Выходные величины* – это те, которые являются результатом решения задачи.

*Имя* ставится в соответствие любой входной и выходной величине; имена сохраняются на всех этапах решения задачи.

*Назначение* – словесное описание величины, для чего она нужна.

*Тип* – определяет тип данных (целое число, действительное число, текст)

*Диапазон*  определяет область изменения величины (от 0 до 5, список строк и т.д.)

**3. Внутреннее проектирование** (разработка алгоритма решения задачи).

*Алгоритм –* это описание последовательности действий, необходимых для решения задачи.

Алгоритм обладает тремя свойствами:

**Массовость** – алгоритм должен описывать не одну конкретную задачу, а группу подобных задач (например, не решение одного конкретного квадратного уравнения , а любого квадратного уравнения ;

**Результативность** - алгоритм должен давать какой-то результат для любого варианта исходных данных. Например, если решается квадратное уравнение, то должны быть предусмотрены случаи:

* Существует два различных действител корня, если 
* Существует два равных действительных корня, если 
* Нет действительных корней, если 
* Уравнение вырождено, то есть A=0. При этом должно быть выдано соответствующее сообщение.

**Детерминированность** - алгоритм должен всегда давать один и тот же результат при одних исходных данных.

Будем описывать алгоритм двумя способами:

- словесным способом

- с помощью блок – схем.

*Словесное описание алгоритма –* это последовательность пронумерованных шагов, описывающих решение задачи. Шаги могут иметь многоуровневую нумерацию, чтобы показать, что одни шаги являются частью других шагов.

*Блок-схема решения задачи –* графическое изображение алгоритма в виде взаимосвязанных блоков.

Будем использовать следующие блоки (Рис. 1):

Линейный процесс

Развилка

Ввод данных

Заголовок цикла

Вывод данных на экран

Начало и конец

стрелка для соединения блоков

узел для соединения стрелок

Рисунок 1 – Виды блоков

**4. Программирование алгоритма**. На этом этапе (называемом также кодированием), происходит запись алгоритма на каком-либо языке программирования.

**5. Синтаксическая отладка программы**, то есть исправление ошибок, связанных с неверным использованием конструкций языка программирования. Эти ошибки выявляются на этапе компиляции программы.

**6. Тестирование программы**, то есть проверка ее работоспособности на различных вариантах исходных данных с просчитанными вручную результатами. Эти результаты должны совпасть с результатами, полученными при расчете на компьютере. При их несовпадении следует искать в программе логические ошибки, связанные с неверным алгоритмом. Тесты для проверки работоспособности программы должны подготавливаться в виде таблицы вида (Табл. 2):

Таблица 2

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Назначение теста | Входные данные | Выходные данные |
|  |  |  |  |

Количество тестов должно быть таким, чтобы каждый тест проверял какую-то специфическую ситуацию, и не было ситуаций, не проверенных ни одним из тестов.

## Виды вычислительных процессов

Существуют следующие виды вычислительных процессов:

*Линейный процесс -* последовательное размещение шагов.

*Разветвляющийся процесс -* в зависимости от условия нужно выполнять либо одно, либо другое действие. В алгоритме используются конструкции

ЕСЛИ условие ТО

Действия

ИНАЧЕ

Действия

ЕСЛИ ВСЕ

В качестве действий может стоять проверка другого условия . Чтобы избежать вложенных условия, используется конструкция ВЫБОР. Она позволяет иметь несколько ветвей для проверки равенства переменной одному из многих значений. Такая конструкция имеет вид:

ВЫБОР ПО переменная

Значение\_1: Действия

Значение\_2: Действия

Значение\_n: Действия

\*: Действия

ВЫБОР ВСЕ

\* означает, что переменная не равна ни одному значению.

*Циклический процесс –* это такой процесс, в котором некоторая последовательность действий выполняется несколько раз до тех пор, пока выполняются некоторые условия. В алгоритме используется конструкция

ПОКА условие ВЫПОЛНИТЬ

Действия

ПОКА ВСЕ

Все задачи являются комбинацией этих трех видов процессов.

В этой лабораторной будут приведены примеры решения задачи для разветвляющихся и циклических процессов.

## Пример алгоритма разветвляющегося процесса

**Условие задачи**. Студенты Иванов и Петров за время практики заработали определенную сумму. Кто из них заработал большую сумму? Определить средний заработок.

**Таблица спецификаций**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Имя | Назначение | Тип | Вх/Вых. | Диапазон |
| 1 | X | Сумма, заработанная Ивановым | Действительное число | вход | >0 |
| 2 | Y | Сумма, заработанная Петровым | Действительное число | вход | >0 |
| 3 | M | Средний заработок | Действительное число | выход | >0 |
| 4 | S | Сообщение о соотношении заработка | Текст | выход | {S.1, S.2, S.3} |

**Таблица сообщений**.

S.1: «Иванов заработал больше»,

S.2: «Петров заработал больше »,

S.3: «Они заработали поровну ».

**Словесное описание алгоритма**:

1. Ввод Х и У.

2. M=(X+Y)/2

3. ЕСЛИ Х>У ТО

3.1.S=Иванов заработал больше.

ИНАЧЕ

3.2. ЕСЛИ У>Х ТО

3.2.1. S=Петров заработал больше

ИНАЧЕ

3.2.2. S=Они заработали поровну

ЕСЛИ ВСЕ

ЕСЛИ ВСЕ

5.Вывод S и M

**Блок-схема алгоритма**

Начало

Конец

1. X, Y

3.

Х>У

2. M=(X+Y)/2

3.1.

S=«Иванов заработал больше»

да

3.2.

Х<У

3.2.1.

S=«Петров заработал больше»

да

3.2.2.

S=«Они заработали поровну »

4. S, M

нет

нет

**Таблица тестов**

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Назначение  теста | Входные данные | Выходные данные |
| 1 | Заработок Иванова больше, чем заработок Петрова | X=200  Y=100 | M=150  S=«Иванов заработал больше» |
| 2 | Заработок Петрова больше, чем заработок Иванова | X=100  Y=200 | M=150  S=«Петров заработал больше» |
| 3 | Заработки равны | X=200  Y=200 | M=200  S=«Они заработали поровну» |

**Проверка правильности алгоритма с помощью таблицы тестов**

Тест 1: блоки 1,2,3,3.1,5

Тест 2: блоки 1,2,3,3.2, 3.2.1, 4

Тест 3: блоки 1,2,3,3.2, 3.2.2, 4

## Пример алгоритма циклического процесса

**Условие задачи.** Группа студентов из N человек сдавала экзамен, и каждый получил оценку. Найти средний балл из оценок группы.

**Таблица спецификаций**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Имя | Назначение | Тип | Вх/Вых. | Диапазон |
| 1 | N | Число студентов, сдававших экзамен | Целое число | вход | >2 и <25 |
| 2 | C | Массив оценок из N элементов | Массив целых чисел | вход | Размер: [2, 25]  Значения: [2,5] |
| 3 | S | Средний балл группы | Действительное число | выход | >0 |
| 4 | M | Сообщение о неверном N | Текст | выход | «N выходит за диапазон возможных значений» |
| 5 | M1 | Сообщение о неверной оценке | Текст | выход | «Значение оценки неверно: должно быть от 2 до 5» |

**Словесное описание алгоритма**:

1. N=1

2. ПОКА N<=2 или N>25 ВЫПОЛНИТЬ

2.1. Ввод N

2.2. ЕСЛИ N<=2 или N>25 ТО

2.2.1. Вывод M.

ЕСЛИ ВСЕ

ПОКА ВСЕ

3.       I=1 S=0.

4.       ПОКА I<=N ВЫПОЛНИТЬ

4.1. C=1

4.2. ПОКА C<2 или C>5 ВЫПОЛНИТЬ

4.2.1. Ввод C

4.2.2. ЕСЛИ C<2 или C>5 ТО

4.2.2.1. Вывод M1

ЕСЛИ ВСЕ

ПОКА ВСЕ

4.3. S=S+C

4.4.  I=I+1

ПОКА ВСЕ

5.       S=S/N

6.       Вывод S

**Блок-схема алгоритма**

Начало

2.1. N

1. N=1

2. N<=2 ИЛИ N>=25

2.2.

N<=2 ИЛИ N>=25

2.2.1. M

3. **I**=1

**S**=0

Объявление массива **С** размера **N**

4.2.1. C[I]

4. I <= N

4.2.

С[I]<2 ИЛИ C[I]>5

4.2.2.1. M1

4.1. C[I]=1

4.2.2

С[I]<2 ИЛИ C[I]>5

4.3. S=S+C[I]

4.4. I=I+1

5. S=S/N

6. S

Конец

**Таблица тестов**

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Назначение теста | Входные данные | Выходные данные |
| 1 | Данные, соответствующие условию задачи | N=5  C=(3,4,5,2,3) | S=3.4 |
| 2 | Данные для N, не соответствуют условию задачи | N=30 | «N выходит за диапазон возможных значений» |
| 3 | Данные о введённой оценке не соответствуют условию задачи | N=6  C=(12,6,3,4,5,3) | «Значение оценки неверно: должно быть от 2 до 5.» |

# II Программа на языке C++.

## Краткие сведения о языке С++

Язык программирования С++ представляет высокоуровневый компилируемый язык программирования общего назначения со статической типизацией, который подходит для создания самых различных приложений. На сегодняшний день С++ является одним из самых популярных и распространенных языков.

Своими корнями он уходит в язык Си, который был разработан в 1969—1973 годах в компании Bell Labs программистом Деннисом Ритчи (Dennis Ritchie). В начале 1980-х годов датский программист **Бьерн Страуструп** (Bjarne Stroustrup), который в то время работал в компании Bell Labs, разработал С++ как расширение к языку Си. Фактически вначале C++ просто дополнял язык Си некоторыми возможностями объектно-ориентированного программирования. И поэтому сам Страуструп вначале называл его как "C with classes" ("Си с классами").

Впоследствии новый язык стал набирать популярность. В него были добавлены новые возможности, которые делали его не просто дополнением к Си, а совершенно новым языком программирования. В итоге "Си с классами" был переименован в С++. И с тех по оба языка стали развиваться независимо друг от друга.

Для разработки программ на С++ необходим **компилятор** - он транслирует исходный код на языке С++ в исполняемый файл, который затем можно запускать. Но в настоящий момент есть очень много различных компиляторов. Они могут отличаться по различным аспектам, в частности, по реализации стандартов. Базовый список компиляторов для С++ можно посмотреть в википедии. Рекомендуется для разработки выбирать те компиляторы, которые развиваются и реализуют все последние стандарты. Для создания программ можно использовать интегрированные среды разработки IDE, такие как Visual Studio, Netbeans, Eclipse, Qt и т.д.

В первых лабораторных работах мы будем разрабатывать **консольные приложения**, в котором реализуется только текстовый интерфейс пользователя. В таком типе приложений не предусматривается применение графических окон, благодаря чему в них не нужно заботиться ни о кнопках, ни о меню, ни о взаимодействии с курсором мыши и т.п. Вместо этого они запускаются в окне командной строки, далее команды вводятся с клавиатуры, результаты работы консольные приложения также выводят на экран в текстовом режиме. Пример вида консольного приложения приведен на рисунке 2.

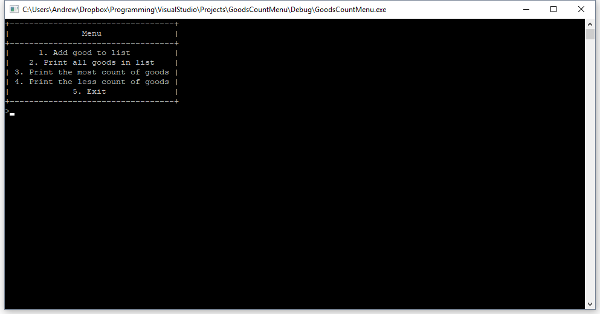


Рисунок 2 – Консольное приложение

## Транслятор и компоновщик

Файл с исходным текстом программы обрабатывается транслятором, который осуществляет перевод программы с языка программирования в понятную машине последовательность кодов. Процесс трансляции разделяется на несколько этапов.

На первом этапе исходный текст (он обычно хранится в виде текстового файла) подвергается лексической обработке. Программа разделяется на предложения, предложение делится на элементарные составляющие (лексемы). Каждая лексема распознаётся (имя, ключевое слово, литерал, символ операции или разделитель) и преобразуется в соответствующее двоичное представление. Этот этап работы транслятора называют лексическим анализом.

Затем наступает этап синтаксического анализа. На этом этапе из лексем собираются выражения, а из выражений - операторы. В ходе трансляции последовательности терминальных символов преобразуются в нетерминалы. Невозможность достижения очередного нетерминала является признаком синтаксической ошибки в тексте исходной программы.

После синтаксического анализа наступает этап поэтапной генерации кода. На этом этапе происходит замена операторов языка высокого уровня инструкциями ассемблера, а затем последовательностями машинных команд. Результат преобразования исходного текста программы записывается в виде двоичного файла (его называют **объектным модулем**) с расширением ".obj".

Объектный модуль можно выполнять лишь после специальной дополнительной обработки (компоновки), которая осуществляется специальной программой-компоновщиком. Преобразованная компоновщиком программа называется загрузочным или выполнимым модулем. Файлы, содержащие загрузочные модули, называют загрузочными или выполнимыми файлами.

## Книги и электронные ресурсы

Для изучения программирования на С++ необходимо опираться не только на методические рекомендации и лекции, получаемые Вами от преподавателя, но и заниматься самостоятельным обучением. Сейчас существует огромное количество материалов, включая видео и общедоступные электронные лекции. Начать можно со следующих ресурсов:

* https://metanit.com/cpp/
* https://docs.microsoft.com/ru-ru/cpp/cpp/?view=msvc-160
* Бьерн Страуструп: Язык программирования С++. Специальное издание. 1136 с.

**Массовый открытый онлайн-курс (МООК):**

* Два курса по введению в С++
  + <https://stepik.org/course/363/syllabus>
  + <https://stepik.org/course/7/syllabus>
* Сборник задач по темам на разных языках.
  + <https://stepik.org/course/56240/syllabus>

## Элементы языка С++

В тексте на любом естественном языке можно выделить четыре основных элемента: символы, слова, словосочетания и предложения. Подобные элементы содержит и **алгоритмический язык**, только слова называют **лексемами** (элементарными конструкциями), словосочетания — **выражениями**, а предложения — **операторами**. Лексемы образуются из символов, выражения — из лексем и символов, а операторы — из символов, выражений и лексем.

**Алфавит**

Алфавит (или множество литер) языка программирования C++ основывается на множестве символов таблицы кодов ASCII. Алфавит C++ включает:

* прописные и строчные латинские буквы;
* символ '\_' (подчерк - также считается буквой);
* арабские цифры от 0 до 9;
* специальные знаки: " { } , | [ ] + - % / \ ; ' : ? < > = ! & # ~ ^ . \*
* пробельные символы: пробел, символы табуляции, символы перехода на новую строку.

Алфавит C++ служит для построения слов, которые в C++ называются **лексемами**. Различают пять типов лексем:

* идентификаторы,
* ключевые слова,
* знаки (символы) операций,
* литералы,
* разделители.

Почти все типы лексем (кроме ключевых слов и идентификаторов) имеют собственные правила словообразования, включая собственные подмножества алфавита.

Лексемы разделяются разделителями. Этой же цели служит множество пробельных символов, к числу которых относятся пробел, символы горизонтальной и вертикальной табуляции, символ новой строки, перевода формата и комментарии.

**Правила образования идентификаторов**

Рассмотрим **правила построения идентификаторов** из букв алфавита (в C++ их три):

1. Первым символом идентификатора C++ может быть только буква.
2. Следующими символами идентификатора могут быть буквы, буквы-цифры и буквы-подчерки.
3. Длина идентификатора неограниченна (фактически же длина зависит от реализации системы программирования).

Вопреки правилам словообразования в C++ существуют ограничения относительно использования подчерка в качестве самой первой буквы в идентификаторах. Особенности реализации делают нежелательными для использования идентификаторы, которые начинаются с этого символа.

**Ключевые слова и имена**

Часть идентификаторов C++ входит в фиксированный словарь ключевых слов. Эти идентификаторы образуют подмножество **ключевых слов** (они так и называются ключевыми словами). Прочие идентификаторы после специального объявления становятся именами. Имена служат для обозначения переменных, типов данных, функций и меток.

Ниже приводится список ключевых слов:

asm auto break case catch char class const continue default do double else enum extern float for friend goto if inline int long new operator private protected public register return short signed sizeof static struct switch template this throw try typedef typeid union unsigned virtual void volatile while.

**Структура программы**

Программа представляет собой последовательность операций, манипулирующих данными. Код на С++ состоит из ряда операторов, каждый из которых оканчивается символом точка с запятой. С++ является языком с блочной структурой, то есть все операторы относятся к какому-то блоку кода. В каждом блоке, который отделяется от остальной части кода с помощью фигурных скобок: { и }, может содержаться любое количество операторов или вообще ни одного. Блоки кода также могут и вкладываться один в другой, то есть одни блоки могут содержать в себе другие блоки,

**Комментарии**

Комментарий – неисполняемая часть программного кода; используется для внесения пояснений и начинается с //

**Библиотеки**

Языки программирования предназначены для написания программ. Однако было бы странно писать всякий раз одни и те же программы или даже одни и те же подпрограммы (например, подпрограмму вывода информации на дисплей или на принтер - эта подпрограмма требуется практически в каждой программе).

К счастью, проблема многократного использования программного кода уже очень давно и успешно решена. Практически каждая система, реализующая тот или иной язык программирования (транслятор, компоновщик и прочее программное окружение) имеет набор готовых к использованию фрагментов программного кода. Этот код может находиться в разной степени готовности. Это могут быть фрагменты текстов программ, но, как правило, это объектный код, располагаемый в особых файлах. Такие файлы называются библиотечными файлами.

Для использования библиотечного кода программисту бывает достаточно указать в программе требуемый файл и обеспечить вызов соответствующих функций. Для использования библиотечного кода бывает достаточно стандартного набора языковых средств. Решение всех остальных проблем транслятор и компоновщик берут на себя. Разумеется, программисту должно быть известно о существовании подобных библиотек и о содержании библиотечных файлов.

## Вывод на консоль

По умолчанию язык C++ не содержит встроенных средств для ввода с консоли и вывода на консоль, эти средства предоставляются библиотекой **iostream**. В ней определены два типа: istream и ostream. istream представляет поток ввода, а ostream - поток вывода.

Вообще сам термин "поток" в данном случае представляет последовательность символов, которая записывается на устройство ввода-вывода или считывается с него. И в данном случае под устройством ввода-вывода рассматривается консоль.

Для записи или вывода символов на консоль применяется объект cout, который представляет тип ostream. А для чтения с консоли используется объект cin. Для использования этих объектов в начало исходного файла необходимо подключить библиотеку iostream:

#include <iostream>

Для вывода на консоль применяется операция <<. Эта операция получает два операнда. Левый операнд представляет объект типа ostream, в данном случае объект cout. А правый операнд - значение, которое надо вывести на консоль. Так как операция << возвращает левый операнд - cout, то с помощью цепочки операций мы можем передать на консоль несколько значений:

std::cout << "Name: " << "Tom" << std::endl;

Также цепочку операций << можно завершать значением **std::endl**, которое вызывает перевод на новую строку и сброс буфера. При выводе в поток данные вначале помещаются в буфер. И сброс буфера гарантирует, что все переданные для вывода на консоль данные немедленно будут выведены на консоль.

## Точка входа

Каждая программа на языке С++ должна иметь как минимум одну функцию - функцию main(). Именно с этой функции начинается выполнение приложения. Ее имя main фиксировано и для всех программ на Си и С++ всегда одинаково.

Функция также является блоком кода, поэтому ее тело обрамляется фигурными скобками, между которыми определяется набор инструкций.

#include <iostream>// подключаем заголовочный файл

int main()       // определяем функцию main

{                                   // начало функции

std::cout << "Hello World!";   // выводим строку на консоль

return 0;                       // выходим из функции

}                                   // конец функции

## Переменные

Впеременные можно помещать данные и затем при необходимости либо извлекать их оттуда, либо проверять их наличие. Переменные бывают разных типов. Чтобы использовать переменные, их нужно объявлять. Это означает, что им необходимо назначать имя и тип. После объявления их можно начинать использовать в качестве единиц хранения для данных того типа, для хранения которого они предназначены согласно объявлению.

Синтаксис определения переменной выглядит следующим образом:

***тип\_переменной имя\_переменной****;*

Также стоит учитывать, что C++ - регистрозависимый язык, а это значит, что регистр символов имеет значение (малые и большие буквы считаются различными символами.). Имена переменных состоят из латинских букв и цифр, начиная с буквы (см. правила образования идентификаторов).

При попытке использовать переменную, которая не была объявлена, код не будет компилироваться.

**Основные типы переменных**:

* int – целое число;
* float – действительное число в форме числа с плавающей точкой;
* bool – булевское значение, принимающее значения true (истина) и false (ложь);
* char – одиночный символ;
* string – последовательность символов

**Операции присваивания**

Операции присваивания позволяют присвоить некоторое значения. Эти операции выполняются над двумя операндами, причем **левый** **операнд** может представлять только модифицируемое именованное выражение, например, переменную.

Базовая операция присваивания = позволяет присвоить значение правого операнда левому операнду:

int x;

x = 2

В данном случае переменная x (левый операнд) будет иметь значение 2 (правый операнд).

Операции присваивания имеют правосторонний порядок, то есть выполняются справа налево. И, таким образом, можно выполнять множественное присваивание:

int a, b, c;

a = b = c = 34;//все переменные будут иметь значение 34

**Инициализация переменных**

После определения переменной можно присвоить некоторое значение:

*int age;*

*age = 20;*

или

*int age=20;*

Если переменную не инициализировать, то происходит ее инициализация по умолчанию. И переменная получает некоторое значение по умолчанию, которое зависит от места, где эта переменная определена.

Если переменная, которая представляет встроенный тип (например, тип int), определена внутри функции, то она получает неопределенное значение. Если переменная встроенного типа определена вне функции, то она получает то значение по умолчанию, которое соответствует ее типу. Для числовых типов это число 0. Например:

#include <iostream>

int x;

int main()

{

    int y;

    std::cout <<"X = " << x << "\n";

    std::cout <<"Y = " << y;

    return 0;

}

Переменная x определена вне функции, и поэтому она получит значение по умолчанию - число 0.

Гораздо сложнее дело обстоит с переменной y, которая определена внутри функции main - ее значение будет неопределенным, и многое будет зависеть от используемого компилятора. В частности, вывод программы, скомпилированной с помощью компилятора G++, может выглядеть следующим образом:

X = 0

Y = 4200475

А в Visual Studio отсутствие значения переменной y вызовет **ошибку**.

Но в любом случае перед использованием переменной лучше явным образом **назначать** ей **определенное** **значение**, а не полагаться на значение по умолчанию.

## Ввод с консоли

Для считывания с консоли данных применяется операция ввода >>, который принимает два операнда. Левый операнд представляет объект типа istream (в данном случае объект cin), с которого производится считывание, а правый операнд - объект, в который считываются данные.

#include <iostream>

int main()

{

    int age;

    double weight;

    std::cout << "Input age: ";

    std::cin >> age;

    std::cout << "Input weight: ";

    std::cin >> weight;

    std::cout << "Your age: " << age << "\t your weight: " << weight << std::endl;

    return 0;

}

Стоит отметить, что так операция ввода в первом случае будет добавлять данные в целочисленную переменную age, то он ожидает ввода числа. В случае с переменной weight оператор ввода ожидает дробное число, причем разделителем целой и дробной части должна быть точка. Поэтому мы не можем ввести любые значения, например, строки. В этом случае программа может выдать некорректный результат.

## Выражения и операции

Выражение – это синтаксическая единица языка, определяющая способ вычисления некоторого значения. Выражения состоят из

* + **операндов**,
  + **операций,**
  + **скобок**.

Каждый **операнд** является в свою очередь выражением или одним из его частных случаев

* + **константой**,
  + **идентификатором, переменной**,
  + **вызов функции**.

Значение выражения зависит от расположения знаков операций и круглых скобок в выражении, а также от приоритета выполнения операций.

**Вызов функций** состоит из имени функции, за которым следует необязательный список выражений в круглых скобках:

*идентификатор (список выражений)*

Выражения со знаками операций могут участвовать в выражениях как операнды. Выражения со знаками операций могут быть *унарными* (с одним операндом), *бинарными* (с двумя операндами) и *тернарными* (с тремя операндами).

**Унарное выражение** состоит из операнда и предшествующего ему знака унарной операции и имеет следующий формат (префиксная или постфиксная форма):

знак унарной операции операнд

или

операнд знак унарной операции

Примеры унарных операций с префиксной формой:

* **Операция логического отрицания:** !.
* **Операция получения адреса операнда:** &
* **Операция обращения по адресу или операция косвенного обращения:** \*
* **Операция унарный минус: -.** Операндом может быть любое выражение со значением арифметического типа. Операция преобразует положительное значение в отрицательное значение и наоборот.

Примеры унарных операций с префиксной и постфиксной формой:

* **Инкремент, или операция увеличения на единицу: ++.**
* **Операция уменьшения значения операнда на величину, кратную единице (декремент): --**
* В префиксной форме увеличение значения операнда производится до определения значения выражения. В результате значение выражения и значение операнда совпадают.

int a = 8;

int b = ++a;

std::cout << a << "\n"; // 9

std::cout << b << "\n"; // 9

* В постфиксной форме увеличение значения операнда производится после определения значения выражения. Поэтому значение выражения оказывается меньше значения операнда.

int a = 8;

int b = a++;

std::cout << a << "\n"; // 9

std::cout << b << "\n"; // 8

**Бинарное выражение** состоит из двух операндов, разделенных знаком бинарной операции:

операнд1 знак **бинарной операции** операнд2

Примеры бинарных операций:

* **Операция умножения** (\*) выполняет умножение операндов.
* **Операция деления** (/) выполняет деление первого операнда на второй. Если две целые величины не делятся нацело, то результат округляется в сторону нуля. При попытке деления на ноль выдается сообщение во время выполнения.
* **Операция остаток от деления** (%) дает остаток от деления первого операнда на второй. Знак результата совпадает со знаком делимого. Если второй операнд равен нулю, то выдается сообщение.
* **Операция вычитания** (-) вычитает второй операнд из первого.
* **Операция сложения** (+) для чисел выполняет сложение операндов
* Бинарная операция + для **строк** означает сцепление (конкатенация) двух строк и присвоение результата третьей строке.

В языке С++ присваивание также является выражением, и значением такого выражения является величина, которая присваивается. Например, x=a+b.

Кроме простого присваивания, имеется целая группа операций присваивания, которые объединяют простое присваивание с одной из бинарных операций. Такие операции называются составными операциями присваивания и имеют вид:

операнд-1 **бинарная операция** = операнд-2

int a = 5;

a += 10;        // 15

a -= 3;         // 12

Составное присваивание эквивалентно следующему простому присваиванию: операнд-1 = операнд-1 бинарная операция операнд-2.

**Тернарное выражение** состоит из трех операндов, разделенных знаками тернарной операции (?) и (:), и имеет формат:

операнд1 ? операнд2 : операнд3

(x < 10)? x = 25: x++

## Операторы

Все операторы языка С++ могут быть условно разделены на следующие категории:

* условные операторы, к которым относятся оператор условия **if** и оператор выбора **switch**;
* операторы цикла (**for**,**while**,**do while**);

Операторы в программе могут объединяться в блоки с помощью фигурных скобок. Все операторы языка С++, кроме блоков, заканчиваются точкой с запятой ";".

Выполнение составного оператора заключается в последовательном выполнении составляющих его операторов.

*Разветвлением* называется процесс управления тем, какая строка кода должна выполняться следующей. Для этого существуют условные операторы. Действие условного оператора основано на сравнении проверяемого значения с одним или более возможными значениями с применением булевской логики.

Далее описаны три доступные в С++ методики разветвления:

* тернарная операция;
* оператор if;
* оператор switch.

Самый простой способ выполнить сравнение — это воспользоваться **тернарной операцией**. Синтаксис этой операции выглядит следующим образом:

<условие> ? <выражение\_если\_истина> : <выражение если ложно>

Пример тернарной операции:

*string result = (mylnteger < 10) ? "Меньше 10" : "Больше или равно 10";*

Результатом приведенной тернарной операции будет какая-то одна из двух строк, присваиваемая переменной result. Выбор того, какая из них должна присваиваться, будет делаться сравнением значения mylnteger с числом 10, при этом в случае, если это значение меньше десяти, присваиваться будет первая строка, а если больше или равно 10 — то вторая.

Такая операция вполне подходит для простых присваиваний, подобных показанному в примере, но не совсем удобна для выполнения на основании сравнения более длинных фрагментов кода. Для них больше подходит оператор if.

## Условия

В качестве условия ставится сравнение, то есть два операнда (константы или идентификаторы), соединенные одним из знаков операции сравнения Знак операции Операция

< Меньше

<= Меньше или равно

>= Больше или равно

== Равно

!= Не равно

Результатом сравнения будет булевское значение (true или false)

Простые условия могут объединяться в сложные с помощью булевских операций:

Логическое НЕ varl != var2;

Логическое И varl = var2 && var3;

Логическое ИЛИ varl = var2 | | var3;

## Оператор if

Наиболее простой способ использования оператора if выглядит так, как показано ниже, и подразумевает вычисление выражения <условие> и выполнение следующей за ним строки кода в случае, если <условие> возвращает true:

if (<условие>)

<код, выполняемый, если <условие> равно true>;

После выполнения этого кода, или невыполнения из-за того, что в результате вычисления выражения <условие> было получено false, выполнение программы возобновляется со следующей строки кода.

Вместе с оператором if также может указываться и дополнительный код с помощью оператора else. Этот оператор выполняется в случае, если при вычислении выражения <условие> получается false:

if (<условие>)

<код, выполняемый, если <условие> равно true>;

else

<код, выполняемый, если <условие> равно false>;

Оба раздела кода могут занимать несколько строк и представлять собой заключенные в фигурные скобки блоки.

Пример:

if (i < j) i++; else { j = i-3; i++; }

Этот пример иллюстрирует также и тот факт, что на месте оператор-1, так же как и на месте оператор-2 могут находиться составные операторы.

Допускается использование вложенных операторов if. Оператор if может быть включен в конструкцию if или в конструкцию else другого оператора if. Чтобы сделать программу более читабельной, рекомендуется группировать операторы и конструкции во вложенных операторах if, используя фигурные скобки. Если же фигурные скобки опущены, то компилятор связывает каждое ключевое слово else с наиболее близким if, для которого нет else.

Пример:

{ int t=2, b=7, r=3;

if (t>b)

{ if (b < r) r=b; }

else r=t;}

В результате выполнения этой программы r станет равным 2.

Если же в программе опустить фигурные скобки, стоящие после оператора if, то программа будет иметь следующий вид:

{ int t=2,b=7,r=3;

if ( t>b)

if ( b < c ) t=b;

else r=t;}

В этом случае r получит значение равное 3, так как ключевое слово else относится ко второму оператору if, который не выполняется, поскольку не выполняется условие, проверяемое в первом операторе if.

Следующий фрагмент иллюстрирует вложенные операторы if:

char ZNAC;

int x,y,z;

if (ZNAC == '-') x = y - z;

else

if (ZNAC == '+') x = y + z;

else

if (ZNAC == '\*') x = y \* z;

else

if (ZNAC == '/') x = y / z;

else ...

Из рассмотрения этого примера можно сделать вывод, что конструкции, использующие вложенные операторы if, являются довольно громоздкими и не всегда достаточно надежными.

**Пример программы** (алгоритм приведен ранее)

#include <iostream>

#include <string>

int main()

{

float X, Y,M;

string S;

std::cin >> X ;

std::cin >> Y ;

M = (X + Y) / 2;

if (X > Y) S = "Иванов заработал больше";

else

if (X < Y) S = "Петров заработал больше";

else S = "Они заработали поровну";

std::cout<<"Результат: "<<S<<std::endl;

std::cout<<"Среднее: "<< M <<std::endl;

return 0;

}

## Оператор switch

Этот оператор похож на if тем, что тоже умеет условно выполнять код на основании проверяемого значения. Однако в отличие от него, switch позволяет проверять переменную сразу на соответствие множеству различных значений, а не с помощью множества отдельных условий. В таких проверках разрешено использовать только дискретные значения, а не конструкции вроде "больше чем X", поэтому и способ применения этого оператора немного отличается.

Структура оператор switch выглядит следующим образом:

switch (< проверяемая\_переменная>)

{ case <значение\_для\_сравпения\_ 1 >:

<код, подлежащий выполнению, если <проверяемая переменная> равна <значение\_для\_сравнения\_1>

break;

case <значение\_для\_сравнения\_2>:

<код, подлежащий выполнению, если <проверяемая\_переменная> равна<значение\_для\_сравнения\_2»

break;

…

case <значение\_для\_сравнения\_М>:

<код, подлежащий выполнению, если <проверяемая\_переменная> =

break;

default:

<код, подлежащий выполнению, если <проверяемая\_переменная> не равна ни одному из значений

break;

}

Значение в <проверяемая\_переменная> сравнивается с каждым из значений <значение\_для\_сравнения\_Х> (задаваемых с помощью операторов case) и если удается обнаружить совпадение, тогда выполняется тот код, который содержится в разделе, соответствующем обнаруженному совпадению. Если не удается обнаружить ни одного совпадения, выполняется тот код, который содержится в разделе default, при условии, что такой раздел существует.

В конце кода в каждом разделе обязательно добавляется дополнительная команда break, исключающая вероятность перехода после обработки одного блока case к обработке следующего оператора case, потому что это недопустимо.

#include <iostream>

int main()

{

    int x = 2;

    switch(x)

    {

        case 1:

            std::cout << "x = 1" << "\n";

            break;

        case 2:

            std::cout << "x = 2" << "\n";

            break;

        case 3:

            std::cout << "x = 3" << "\n";

            break;

        default:

            std::cout << "x is undefined" << "\n";

            break;

    }

    return 0;

}

1. Начало

8. Конец

3. Выбор по x

default

4. "x is undefined"

6. "x = 2"

2. Объявление и инициализация переменной х

5. "x = 1"

7. "x = 3"

1

2

3

Рисунок – Пример блок схемы для оператора выбора (switch statement)

## Оператор do while

Такой цикл функционируют следующим образом. Сначала выполняется код тела цикла, затем производится проверка, и если в ее результате возвращается true, данный код выполняется снова, и так повторяется до тех пор, пока после проверки не будет возвращено значение false, в этом случае цикл завершается.

Структура цикла do выглядит, как показано ниже, и подразумевает, что выражение <условие> должно возвращать одно из булевских значений:

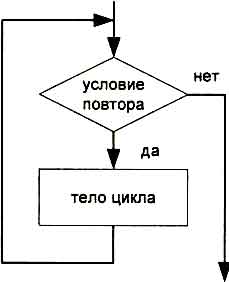
**do**

{

<код, подлежащий выполнению в цикле>

} **while** (<условие>)**;**

**Оформление на блок-схеме**



**Пример кода на С++** (алгоритм приведен ранее).

Здесь цикл do используется не только для обработки оценок, но и для проверки корректности ввода по значению.

#include <iostream>

#include <string>

using namespace std;

int main()

{

int N=1,C=1,i=1;

float S=0;

string M="Неверное N", M1="Неверная оценка";

do{

cout<<"Введите число студентов: ....?"<<endl;

cin>>N;

if (N<=2||N>=25) cout<<M<<endl;

} while (N<=0||N>=25) ;

do

{

do

{

cout<<"Введите оценку студента № "<<i<<": "<<endl;

cin>>C;

if (C<2||C>5) cout<<M1<<endl;

}while (C<2||C>5) ;

S+=C;

i++;

} while (i<=N) ;

S/=N;

cout<<"Средний балл = "<<S<<endl;

return 0;

}

## Оператор while

Этот цикл очень похож на цикл do…while, но имеет одно важное отличие: булевская проверка в нем выполняется в начале, а не в конце цикла. Если после выполнения этой проверки возвращается false, код в теле цикла вообще не выполняется. Вместо этого сразу выполняется код, следующий непосредственно за циклом. Структура цикла while:

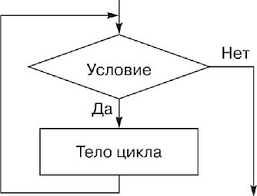
**while** (<условие>)

{

<код, подлежащий выполнению в цикле>

}

**Оформление на блок-схеме**



**Пример кода на С++** (алгоритм приведен ранее).

#include <iostream>

#include <string>

using namespace std;

int main()

{

int N=1,C=1,i=1;

float S=0;

string M="Неверное N", M1="Неверная оценка";

do{

cout<<"Введите число студентов: ....?"<<endl;

cin>>N;

if (N<=2||N>=25) cout<<M<<endl;

} while (N<=0||N>=25) ;

while (i<=N)

{

do

{

cout<<"Введите оценку студента № "<<i<<": "<<endl;

cin>>C;

if (C<2||C>5) cout<<M1<<endl;

}while (C<2||C>5) ;

S+=C;

i++;

};

S/=N;

cout<<"Средний балл = "<<S<<endl;

return 0;

}

## Оператор for

Цикл этого типа выполняется определенное количество раз и обладает собственным счетчиком. Для определения цикла for требуется описанная ниже информация:

* начальное значение для инициализации переменной, играющей роль счетчика;
* условие для продолжения цикла, в котором участвует переменная- счетчик;
* операция, которая должна выполняться над переменной-счетчиком в конце каждого цикла.

Например, при желании создать цикл for и сделать так, чтобы значение его счетчика увеличивалось на 1 с 1 до 10, в качестве начального значения потребуется указать 1, в качестве условия — то, что значение счетчика должно быть меньше или равно 10, а в качестве операции, которая должна выполняться в конце каждого такого цикла — увеличение счетчика на 1.

Вся эта информация должна размещаться в структуре цикла следующим образом:

**for** {<инициализация>; <условие>; <операция>)

{

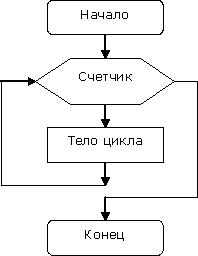
<код, подлежащий выполнению в цикле>

}

Формат цикла for делает код более удобным для восприятия, поскольку его синтаксис подразумевает задание всех деталей цикла в одном месте, а не разнесение его по нескольким операторам и их размещение в разных частях кода.

**Оформление на блок-схеме**

При использовании цикла типа **for** в отчётах используем новую отличную от **while** фигуру.



**Пример кода на С++**

#include <iostream>

#include <string>

using namespace std;

int main()

{

int N=1,C=1,i=1;

float S=0;

string M="Неверное N", M1="Неверная оценка";

do{

cout<<"Введите число студентов: ....?"<<endl;

cin>>N;

if (N<=2||N>=25) cout<<M<<endl;

} while (N<=0||N>=25) ;

for(i=1;i<=N;i++)

{

do

{

cout<<"Введите оценку студента № "<<i<<": "<<endl;

cin>>C;

if (C<2||C>5) cout<<M1<<endl;

}while (C<2||C>5) ;

S+=C;

}

S/=N;

cout<<"Средний балл = "<<S<<endl;

return 0;

}

## Массивы

У всех рассмотренных до этого типов была одна общая черта: каждый из них позволял хранить одно единственное значение. Однако бывают ситуации, в которых требуется сохранять много данных, и в таких ситуациях эти типы переменных являются не очень удобными. Иногда бывает необходимо сохранять несколько значений одного и того же типа одновременно без использования для каждого из них отдельной переменной.

Например, предположим, что требуется выполнить некоторую обработку имен трех друзей. Можно объявить простые строковые переменные:

string friendNamel = "Robert Barwell";

string friendName2 = "Mike Parry";

string friendName3 = "Jeremy Beacock";

Однако существует и альтернативный подход, который предусматривает применение массива. Массивы — это проиндексированные списки переменных, хранящиеся в единственной переменной типа массива. Например, для хранения трех показанных выше имен можно создать массив под названием friendNames. Получать доступ к отдельным членам этого массива можно будет указанием их индекса в квадратных скобках:

friendNames [<индекс>]

Этот индекс представляет собой просто целое число. Для первого элемента в массиве этим числом будет 0, для второго — 1 и т.д. Это означает, что по элементам массива можно проходить в цикле:

#include <iostream>

#include <string>

using namespace std;

int main()

{

int i;

string friendNames[3];

for (i = 0; i < 3; i++)

{

cout<<"Введите имя друга ...?"<<endl;

cin>>friendNames[i];

}

return 0;

}

Элементы массива имеют один единственный тип. Например, типом приведенного в качестве примера массива friendNames является string.

**Объявляются** массивы следующим образом:

**<тип данных> <имя одномерного массив>[<размерность>];**

Пример объявления **нескольких массивов** одного типа: int А[5], B[10];

Массивы должны обязательно **инициализироваться** перед тем, как к ним можно будет получить доступ или присваивать значения

Массивы могут быть инициализированы **при объявлении**:

int a[16] = { 5, -12, -12, 9, 10, 0, -9, -12, -1, 23, 65, 64, 11, 43, 39, -15 };

Инициализации массива **без определения** его **размера**:

int a[]={5,-12,-12,9,10,0,-9,-12,-1,23,65,64,11,43,39,-15};

В данном случае компилятор сам определит размер одномерного массива. Размер массива можно не указывать только при его инициализации, при обычном объявлении массива обязательно нужно указывать размер массива.

**Демонстрация приемов ввода и вывода одномерного массива.**

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

int X[100];

int N,i; // N – длина массива

//1. Ввод массива

do

{

cout <<"Введите количество элементов от 1 до 100: ";

cin>>N;

}

while (N<1 || N>100);

cout <<"В массиве "<< N <<" элементов\n";

for (i=0; i<N; i++)

{

cout<<"X["<<i+1<<"] =";

cin>>X[i];

}

// 2. Вывод массива

cout<<"Массив X[N]"<<endl;

for (i=0;i<N;i++)

cout<<X[i]<<" ";

return 0;

}

**Пример кода на С++** (алгоритм приведен ранее).

#include <iostream>

#include <string>

using namespace std;

int main()

{

int N=1;

float S=0;

string M="Неверное N", M1="Неверная оценка";

do{

cout<<"Введите число студентов: ....?"<<endl;

cin>>N;

if (N<=2||N>=25) cout<<M<<endl;

} while (N<=0||N>=25) ;

int C[N];

for(int i=0;i<N;i++)

{

do

{

cout<<"Введите оценку студента № "<<i+1<<": "<<endl;

cin>>C[i];

if (C[i]<2||C[i]>5) cout<<M1<<endl;

}while (C[i]<2||C[i]>5) ;

S+=C[i];

}

S/=N;

cout<<"Средний балл = "<<S<<endl;

return 0;

}

# III Индивидуальные задания

## Группа A. Условное ветвление

|  |
| --- |
|  |

1. Задан круг с центром в точке О(x0, y0) и радиусом R0 и точка А(x1, y1). Определить, находится ли точка внутри круга.
2. Задана окружность с центром в точке x0, y0 и радиусом R0 и прямая y=ax+b. Определить, пересекаются ли прямая и окружность. Если пересекаются, найти точки пересечения.
3. Определить, пересекаются ли параболы  у=аx2+bx+с  и у=dx2+ex. Если пересекаются, найти точки пересечения.
4. Пункты A, B и C находятся на одной прямой, пункт B между пунктами A и C. Расстояние между A и B равно L. Расстояние между B и C равно S. Из пункта A в пункт C выехал автомобилист со скоростью V0, а из пункта B в пункт C выехал мотоциклист со скоростью V1. Кто раньше приедет в пункт C?
5. Определите,  пересекаются ли парабола у=cx2+dx+f  и прямая y=ax+b. Если пересекаются, то определите, в каких точках.
6. Определите, имеет ли решение система.  Если имеет, найдите корни.
7. Заданы три точки A(a1,a2,a3), B(b1,b2,b3) и C(c1,c2,c3).  Определить, между какими точками расстояние будет наименьшим.
8. Даны уравнения двух прямых  y=a1x+b1 и y=a2x+b2. Определить, пересекаются ли эти прямые, совпадают или параллельны.
9. Задана точка M с координатами (x, y).  Определить является ли эта точка началом координат или лежит на одной из координатных осей.
10. Дана точка M с координатами (x, y). Определить является ли эта точка началом координат, если нет, то в каком координатном углу она расположена.
11. Заданы точки A(x1,y1) и B(x2,y2).  Определить лежат ли они на прямой  y=ax+b.
12. Известны координаты точки M(x,y). Определить принадлежит ли эта точка одной из прямых y=a1x+b1; y=a2x-b2.
13. Даны действительные числа A,B,C,D. Если A<=B<=C<D, то каждое число заменить наибольшим из них; если A>B>C>D, то числа оставить без изменения; в противном случае все числа заменить их квадратами.
14. Даны действительные числа Х, Y. Если оба числа отрицательны, то каждое число заменить его модулем; если отрицательно только одно из них, то оба значения увеличить на 0.5; если оба числа положительны и ни одно из них не принадлежит отрезку [ 0.5, 2.0 ], то оба значения уменьшить в 10 раз; в остальных случаях числа оставить без изменения.

## Группа Б. Выбор из нескольких возможностей

**1**. Дано целое число K. Вывести описание оценки, соответствующей числу K (1 — «плохо», 2 — «неудовлетворительно», 3 — «удовлетворительно», 4 — «хорошо», 5 — «отлично»). Если K не лежит в диапазоне 1–5, то вывести строку «ошибка».

**2**. Дан номер месяца — целое число в диапазоне 1–12 (1 — январь, 2 —февраль и т. д.). Вывести название соответствующего времени года («зима», «весна», «лето», «осень»).

**3**. Арифметические действия над числами пронумерованы следующим образом: 1 — сложение, 2 — вычитание, 3 — умножение, 4 — деление. Дан номер действия N (целое число в диапазоне 1–4) и вещественные числа A и B (В не равно 0). Выполнить над числами указанное действие и вывести результат.

**4**. Единицы длины пронумерованы следующим образом: 1 — дециметр, 2 — километр, 3 — метр, 4 — миллиметр, 5 — сантиметр. Дан номер единицы длины (целое число в диапазоне 1–5) и длина отрезка в этих единицах (вещественное число). Найти длину отрезка в метрах.

**5**. Единицы массы пронумерованы следующим образом: 1 — килограмм, 2 — миллиграмм, 3 — грамм, 4 — тонна, 5 — центнер. Дан номер единицы массы (целое число в диапазоне 1–5) и масса тела в этих единицах (вещественное число). Найти массу тела в килограммах.

**6**. Элементы окружности пронумерованы следующим образом: 1 — радиус R, 2 — диаметр D = 2·R, 3 — длина L = 2·р·R, 4 — площадь круга S = р·R2. Дан номер одного из этих элементов и его значение. Вывести значения остальных элементов данной окружности (в том же порядке). В качестве значения р использовать 3.14.

**7**. Элементы равнобедренного прямоугольного треугольника пронумерованы следующим образом: 1 — катет a, 2 — гипотенуза c = 2 *a* , 3 — высота h, опущенная на гипотенузу (h = c/2), 4 — площадь S = c·h/2. Дан номер одного из этих элементов и его значение. Вывести значения остальных элементов данного треугольника (в том же порядке).

**8**. Элементы равностороннего треугольника пронумерованы следующим образом: 1 — сторона a, 2 — радиус R1 вписанной окружности (R1 == 3 /6 a ), 3 — радиус R2 описанной окружности (R2 = 2\*R1), 4 — площадь S = 2 \*3 /4 a . Дан номер одного из этих элементов и его значение. Вывести значения остальных элементов данного треугольника (в том же порядке).

**9.** Мастям игральных карт присвоены порядковые номера: 1 — пики, 2 — трефы, 3 — бубны, 4 — червы. Достоинству карт, старших десятки, присвоены номера: 11 — валет, 12 — дама, 13 — король, 14 — туз. Даны два целых числа: N — достоинство (6 ≤ N ≤ 14) и M — масть карты (1 ≤ M ≤ 4). Вывести название соответствующей карты вида «шестерка бубен», «дама червей», «туз треф» и т. П.

**10.** Дано целое число в диапазоне 20–69, определяющее возраст (в годах). Вывести описание указанного возраста, обеспечив правильное согласование числа со словом «год», например: 20 — «двадцать лет», 32 — «тридцать два года», 41 — «сорок один год».

**11.** Дано целое число в диапазоне 100–999. Вывести описание данного числа, например: 256 — «двести пятьдесят шесть», 814 — «восемьсот четырнадцать».

**12.** В восточном календаре принят 60-летний цикл, состоящий из 12-летних подциклов, обозначаемых названиями цвета: зеленый, красный, желтый, белый и черный. В каждом подцикле годы носят названия животных: крысы, коровы, тигра, зайца, дракона, змеи, лошади, овцы, обезьяны, курицы, собаки и свиньи. По номеру года определить его название, если 1984 год — начало цикла: «год зеленой крысы».

**13.** Функция задана в виде графика (R = 2):

|  |
| --- |
| http://ips.ifmo.ru/courses/pascal/problems/l2/ru-ifmo-ips-pascal-lab02-task01-05-1.gif |

Написать программу, которая по введенному значению аргумента вычисляет значение функции.

## Группа В. Циклический процесс, массивы и строки

Дана непустая последовательность слов из строчных русских букв; между соседними словами – запятая, за последним словом – точка.

1. Множество всех букв, входящих в текст не менее 2 раз.
2. Все русские гласные буквы (а, е, и, о, у, ы, э, ю, я), входящие в этот текст.
3. Все гласные буквы, которые входят в каждое слово.
4. Все звонкие согласные буквы, которые входят хотя бы в одно слово.
5. Все глухие согласные буквы, которые НЕ входят хотя бы в одно слово.
6. Все согласные буквы, которые входят только в одно слово.
7. Все глухие согласные буквы, которые НЕ входят только в одно слово.
8. Все гласные буквы, которые входят более чем в одно слово.
9. Все звонкие согласные буквы, которые входят в каждое нечетное слово и не входят ни в одно четное слово.
10. Все гласные буквы, которые не входят более чем в одно слово.
11. Множество всех букв, входящих в текст не менее 2 раз.
12. Все русские гласные буквы (а, е, и, о, у, ы, э, ю, я), входящие в этот текст.
13. Все гласные буквы, которые входят в каждое слово.
14. Все звонкие согласные буквы, которые входят хотя бы в одно слово.

# IV Оформление отчёта

## Правила оформления

Общие требования к оформлению содержатся в регламентирующих документах ИРНИТУ:

* https://www.istu.edu/local/modules/doc/download/41649

Отчёт по лабораторной должен содержать следующие пункты (номера и названия пунктов не менять!)

|  |  |  |
| --- | --- | --- |
| Номер  пункта | Название  пункта | Пояснение |
| 1 | Условие задачи | В этом пункте необходимо включить формулировку общей и индивидуальной части задания, чтобы можно было осуществить проверку без необходимости смотреть недостающую информацию в методичке. |
| 2 | Математическая модель | Здесь при необходимости указываются математические рассуждения и вывод расчётных формулы, которые были использованы в программе.  Для формул использовать средства Word, а не картинки. |
| 3 | Таблица внешних спецификаций |  |
| 4 | Алгоритмизация | В этом пункте обязательна должна присутствовать блок-схема. При желании можно привести и словесное описание. При рисовании блок схемы в ворде использовать **полотно**. |
| 5 | Проектирование тестов | В этом пункте должна быть Таблица тестов |
| 6 | Проверка правильности алгоритма с помощью таблицы |  |
| 7 | Кодирование алгоритма или запись алгоритма на языке С++ | Весь текст программы, чтобы при проверке его можно было скопировать в редактор и сразу скомпилировать. |

## Создание блок-схемы в ворде

* Если используется Word, то рисовать блок-схемы с использованием:
  + меню «Вставка» -> кнопка «Фигуры»->раздел «Блок-схема»
* Все фигуры размещать внутри **полотна**:
  + меню «Вставка» -> кнопка «Фигуры»->кнопка «Новое полотно» (в самом низу списка фигур)
* Начать рисование можно, скопировав полотно блок-схемы примера из методички
* Можно использовать специализированные средства создания блок-схем, но как правило это сложнее сопровождать и вносить исправления по замечаниям на занятии

## Пример выполнения лабораторной работы

1. Условие задачи

Если три отрезка x, y, z образуют треугольник, то проверить, является ли он прямоугольным и вывести сообщение об этом, в противном случае (треугольник не прямоугольный) найти периметр треугольника и площадь.

Введем следующие обозначения:

Pr - периметр,

S - площадь.

2. Математическая модель

Используем следующие математические положения и формулы:

1. В треугольнике сумма длин двух любых его сторон больше третьей.

2. По теореме Пифагора в прямоугольном треугольнике квадрат гипотенузы равен сумме квадратов катетов.

3. Для вычисления площади треугольника применим формулу Герона

периметр Pr = x + y + z

площадь по формуле Герона S =(p (p-x) (p-y) (p-z))1/2, где p-полупериметр

3. Таблица внешних спецификаций

Таблица 1 – Внешние спецификации

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Имя | Назначение | Тип | Вх/Вых. | Диапазон |
| 1 | x | Отрезок 1 | Вещ. | вход | (3.4E-38; 3.4E+38) |
| 2 | y | Отрезок 2 | Вещ. | вход | (3.4E-38; 3.4E+38) |
| 3 | z | Отрезок 3 | Вещ. | вход | (3.4E-38; 3.4E+38) |
| 4 | Pr | Периметр | Вещ. | выход | (3.4E-38; 3.4E+38) |
| 5 | S | Площадь | Вещ. | выход | (3.4E-38; 3.4E+38) |
| 6 | C1 | Сообщение | Текст | выход | 2 вида |
| 7 | C2 | Сообщение | Текст | выход | 2 вида |
| 8 | C3 | Сообщение | Текст | выход | 2 вида |

Таблица сообщений

C1.1: “Треугольник существует”

C1.2: “Треугольник не существует”

C2.1: “Треугольник прямоугольный”

C2.2: “Треугольник не прямоугольный”

C3.1: ' периметр треугольника ='

C3.2: ' площадь треугольника ='

4.Алгоритмизация

Блок-схема алгоритма представлена на рис. 1.

1. Начало

11. Конец

2. x, y, z

3. Можно построить треугольник?

8. Вычисление Pr и S

да

6. Он прямоугольный?

да

9. C3.1, Pr, C3.2, S

нет

нет

4. C1.2

5. C1.1

10. C2.1

7. C2.2

Рисунок 1 – Блок-схема алгоритма

5. Проектирование тестов

Таблица 2 – Таблица тестов

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Назначение  теста | Входные данные | Выходные данные |
| 1 | Треугольник построить нельзя, так как есть данные не попадают в диапазон допустимых значений | x=0  y=0  z=1 | «Треугольник не существует» |
| 2 | Треугольник построить нельзя | x=2  y=2  z=5 | «Треугольник не существует» |
| 3 | Треугольник построить можно, треугольник прямоугольный | x=3  y=4  z=5 | “Треугольник существует”  “Треугольник прямоугольный” |
| 4 | Треугольник построить можно, но треугольник не прямоугольный | x=3  y=2  z=4 | “Треугольник существует”  “Треугольник не прямоугольный”  Полупериметр треугольника = 4.5  Площадь треугольника = 2.90474 |

6. Проверка правильности алгоритма с помощью таблицы

Тест 1,2: блоки 1,2,3,4,11

Тест 3: блоки 1,2,3,5,6,10,11

Тест 4: блоки 1,2,3,5,6,7,8,9,11

7. Кодирование алгоритма или запись алгоритма на языке С++

// пример лабораторной работы #1

#include <iostream>

#include <math.h>

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

float x,y,z;

float S,Pr,p;

cout << "Введите длины сторон трегольника" << endl;

cin >> x >>y >>z;

//проверяем существование треугольника

if ((x>0)&&(y>0)&&(z>0)&&(x+y>z)&&(y+z>x)&&(x+z>y))

{

cout << "Треугольник существует" << endl;

//проверяем по теореме Пифагора, является ли треугольник прямоугольным

if (fabs( x\*x - y\*y - z\*z )==0 ||

fabs(z\*z - y\*y - x\*x)==0 ||

fabs(y\*y - z\*z - x\*x)==0)

cout << "Треугольник прямоугольный" << endl;

else

{

cout << "Треугольник не прямоугольный" << endl;

Pr= x + y + z;

p=Pr/2;

cout << "Полупериметр треугольника = " <<p<< endl;

S= sqrt (p\* (p-x) \* (p- y) \* (p- z));

cout << "Площадь треугольника = " <<S<< endl;

}

} else cout << "Треугольник не существует" << endl;

return 0;

}