

Ширева С. Н.

**ОСНОВЫ
ПРОГРАММИРОВАНИЯ
НА ЯЗЫКЕ C/C++**

Практикум

РГПУ
2017

Аннотация

Практикум предназначен для студентов, изучающих язык С или С++ на практических занятиях или самостоятельно. Переменные, операторы присваивания, развилки и циклов, структурные типы данных (массивы и строки), реализация подпрограмм рассматриваются на примерах, сопровождаемых необходимыми теоретическими сведениями. Обсуждается использование функций в С и в С++. По каждой теме приведено по 12 вариантов заданий.

Содержание

Введение Знакомство со средой	3
Лабораторная 1 Создание проекта	3
Тема № 1 Линейный вычислительный процесс	31
Лабораторная № 2 Линейный вычислительный процесс	32
Контрольные вопросы	34
Индивидуальные задания	35
Тема 2 Разветвляющийся вычислительный процесс	38
Лабораторная работа № 3	38
Условный оператор	38
Теория.....	38
Примеры	42
Контрольные вопросы	46
Индивидуальные задания	47
Оператор выбора	49
Теория.....	49
Примеры	51
Контрольные вопросы	55
Индивидуальные задания	55
Тема № 3. Циклический вычислительный процесс	57
Лабораторная работа № 4	57
Циклы с параметром	57
Теория.....	57
Примеры	59
Контрольные вопросы	68
Индивидуальные задания	69
Цикл с условием	70
Теория.....	70
Примеры	71
Контрольные вопросы.	75
Индивидуальные задания	76
Вычисления с точностью*	78
Теория.....	78
Примеры	79
Контрольные вопросы	81
Индивидуальные задания	82
Тема 4 Вспомогательные алгоритмы	83
Лабораторная 5	85
Функции	85
Теория.....	85
Примеры	87
Контрольные вопросы	92
Индивидуальные задания	93
Процедуры (void-функции)	94

Теория.....	94
Примеры.....	96
Контрольные вопросы	100
Индивидуальные задания	100
Рекурсия*	101
Теория.....	101
Примеры	105
Контрольные вопросы	107
Индивидуальные задания	108
Тема 6. Алгоритмы работы со структурированными типами данных	110
Лабораторная работа 6.....	110
Стандартные алгоритмы работы с одномерными массивами	110
Теория.....	110
Примеры	115
Контрольные вопросы	121
Индивидуальные задания	122
Формирование массива.....	124
Теория.....	124
Примеры	124
Контрольные вопросы	125
Индивидуальные задания	125
Двумерный массив	126
Теория.....	126
Примеры	129
Контрольные вопросы	132
Индивидуальные задания	132
Лабораторная работа № 7 Строки в C++	133
Теория.....	133
Примеры	141
Контрольные вопросы	145
Индивидуальные задания	145
Литература	147

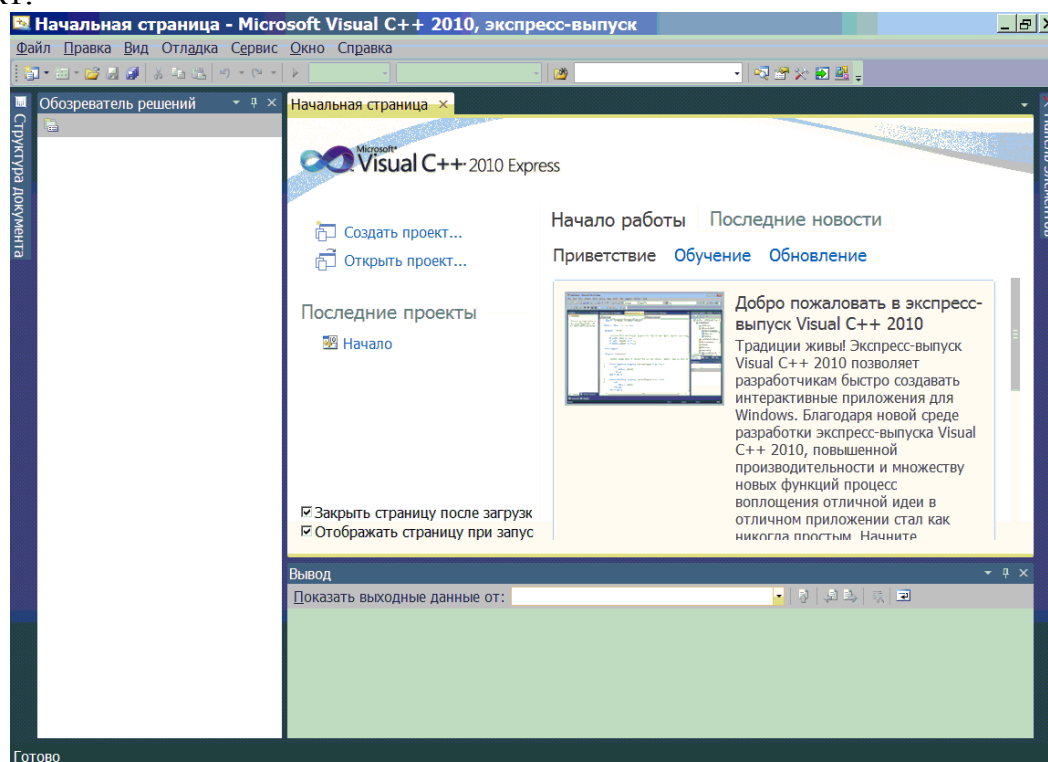
Введение Знакомство со средой

Независимо от того, какого вида программа разрабатывается, в Visual Studio необходимо создать Решение (Solution) и в нем Проект (Project). Создание пустого (без проектов) решения не имеет смысла, поэтому решение будет автоматически создано при создании нового проекта. Прежде чем описать последовательность действий, необходимых для создания и выполнения простой программы, остановимся на соотношении понятий «проект» и «решение». В одном решении могут одновременно входить проекты программ разных видов. Текст (код) программы может быть обработан средой Visual Studio, когда он помещен в проект, а проект включен в решение. Часто в одно решение помещают взаимосвязанные проекты, например, использующие одно и то же библиотеки.

В процессе изучения основ языка C++ мы будем создавать консольные приложения. Консольное приложение — это программа, отображающая текстовую информацию и позволяющая вводить символы с клавиатуры. Консольное приложение позволит не отвлекаться на изучение среды разработки, а полностью сосредоточить свое внимание на изучении синтаксиса языка.

Лабораторная 1 Создание проекта

Для создания проекта на начальной странице надо выбрать пункт Создать проект.



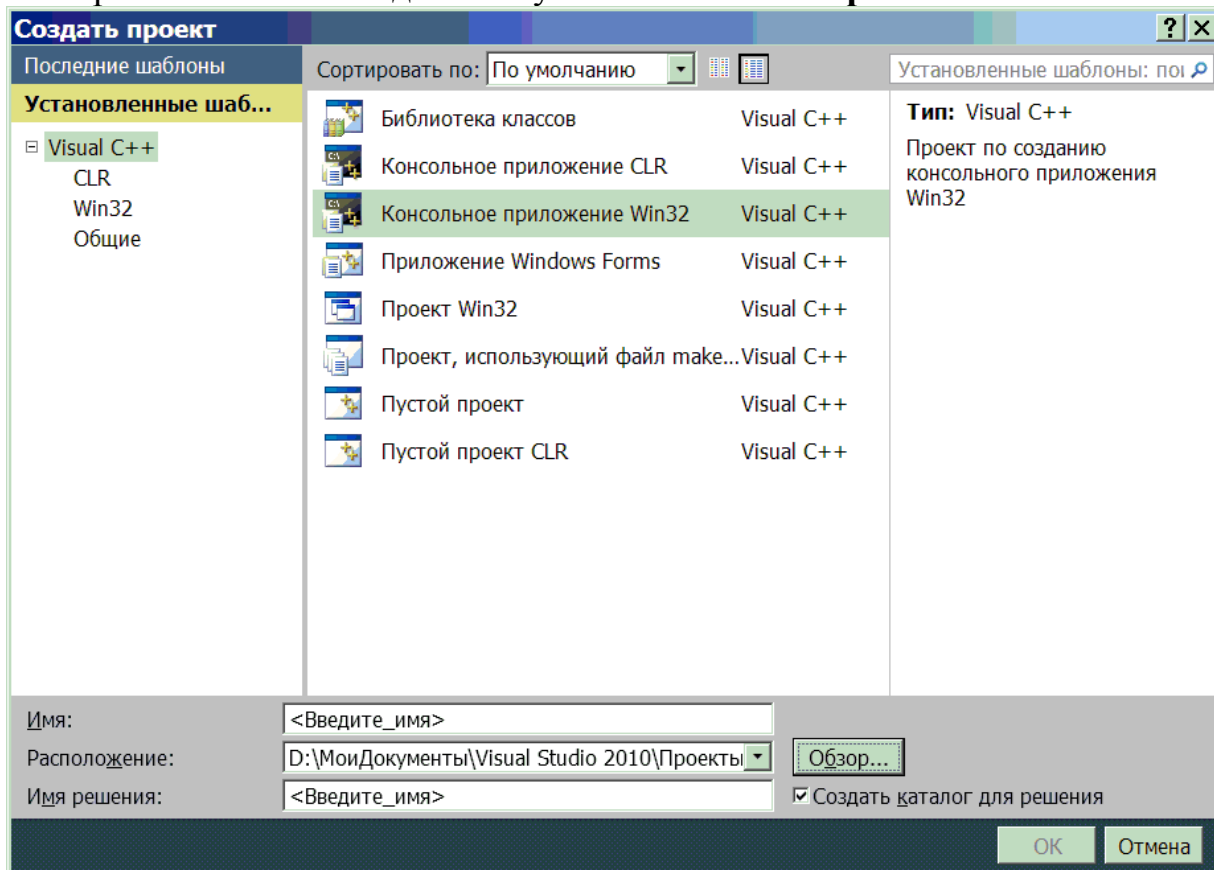
Создать консольное приложение можно двумя способами.

- Первый способ заключается в создании пустого проекта и написании кода с нуля.

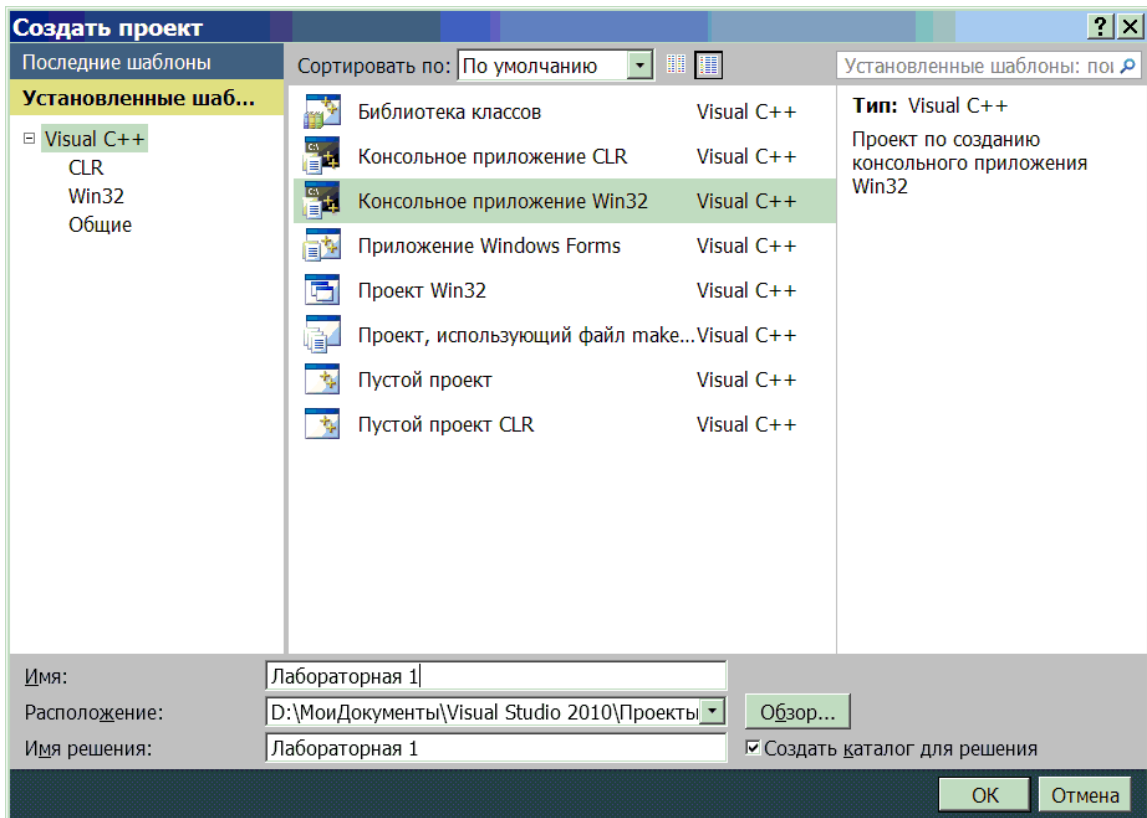
- Второй способ позволяет создать консольное приложение с помощью мастера. В этом случае мастер создаст все необходимые файлы и заполнит их шаблонами кода.

Создание пустого проекта

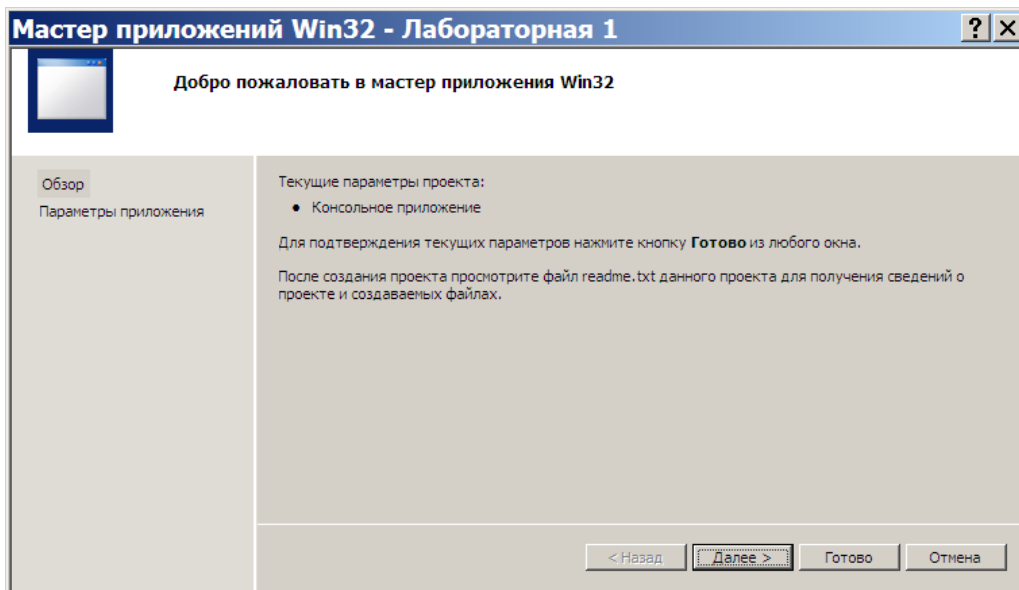
В открывшемся окне выделяем пункт **Консольное приложение Win32**.



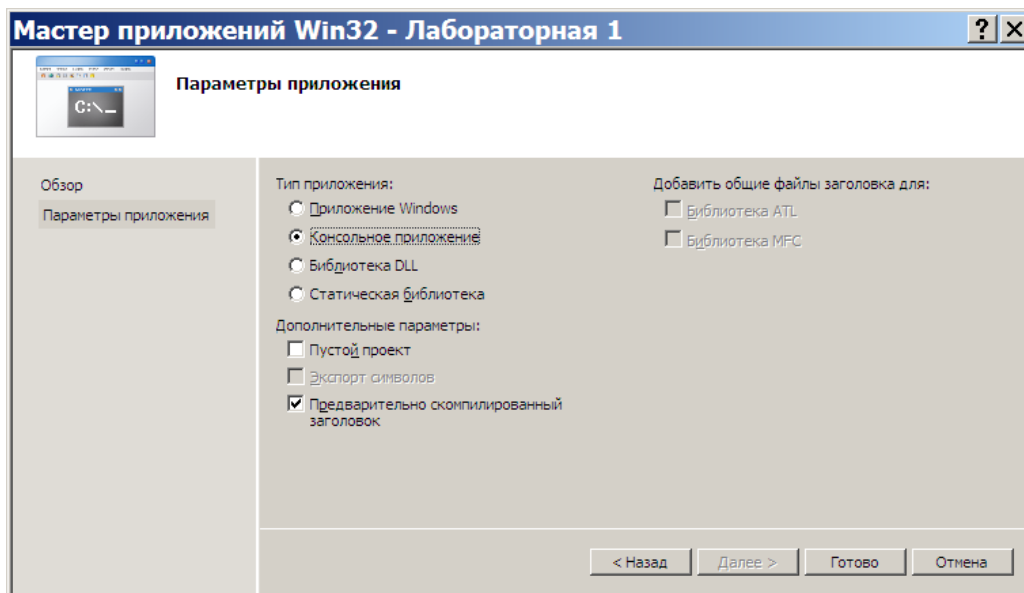
Вводим название проекта (например, "Лабораторная 1") в поле **Имя**. Введенное название автоматически копируется в поле **Имя решения**. В поле **Расположение** указываем путь к каталогу, в котором будет сохранен проект, и устанавливаем флажок **Создать каталог для решения**.



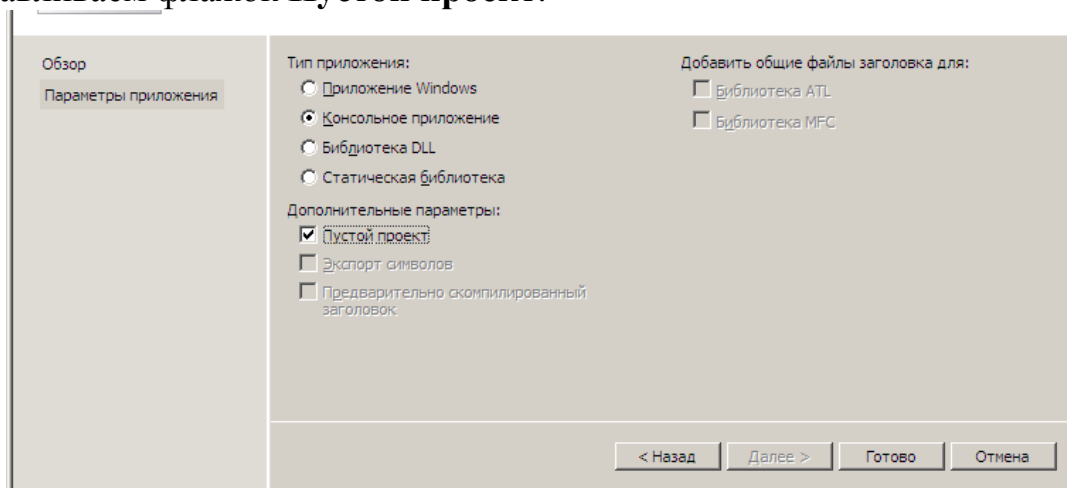
Нажимаем кнопку ОК. В результате откроется окно **Мастер приложения Win32**.



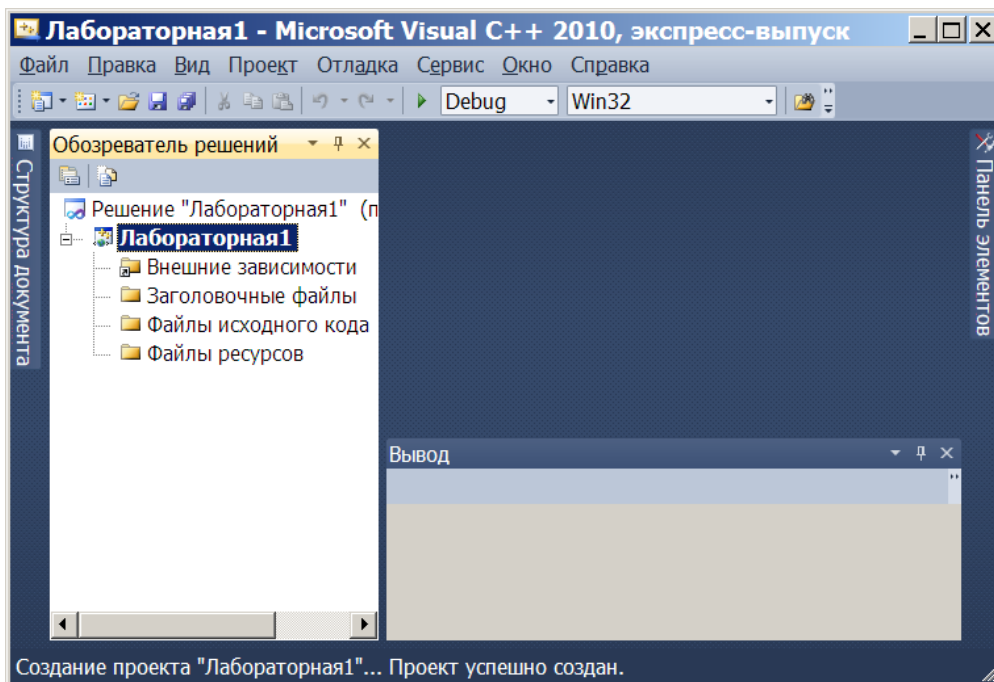
Нажимаем кнопку Далее.



В группе переключателей **Тип приложения** выбираем **Консольное приложение**. Снимаем флажок **Предварительно скомпилированный заголовок** и устанавливаем флажок **Пустой проект**.

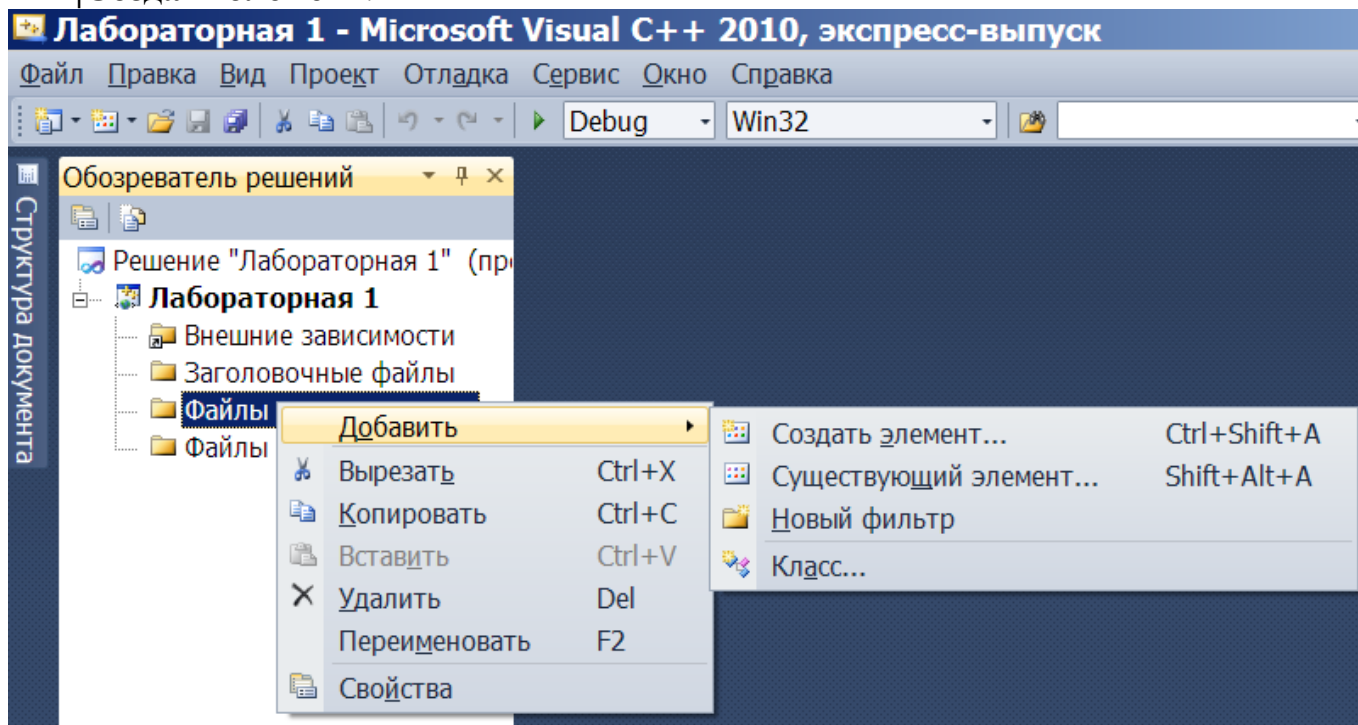


Нажимаем кнопку **Готово**. В результате проект будет создан и открыт в главном окне.



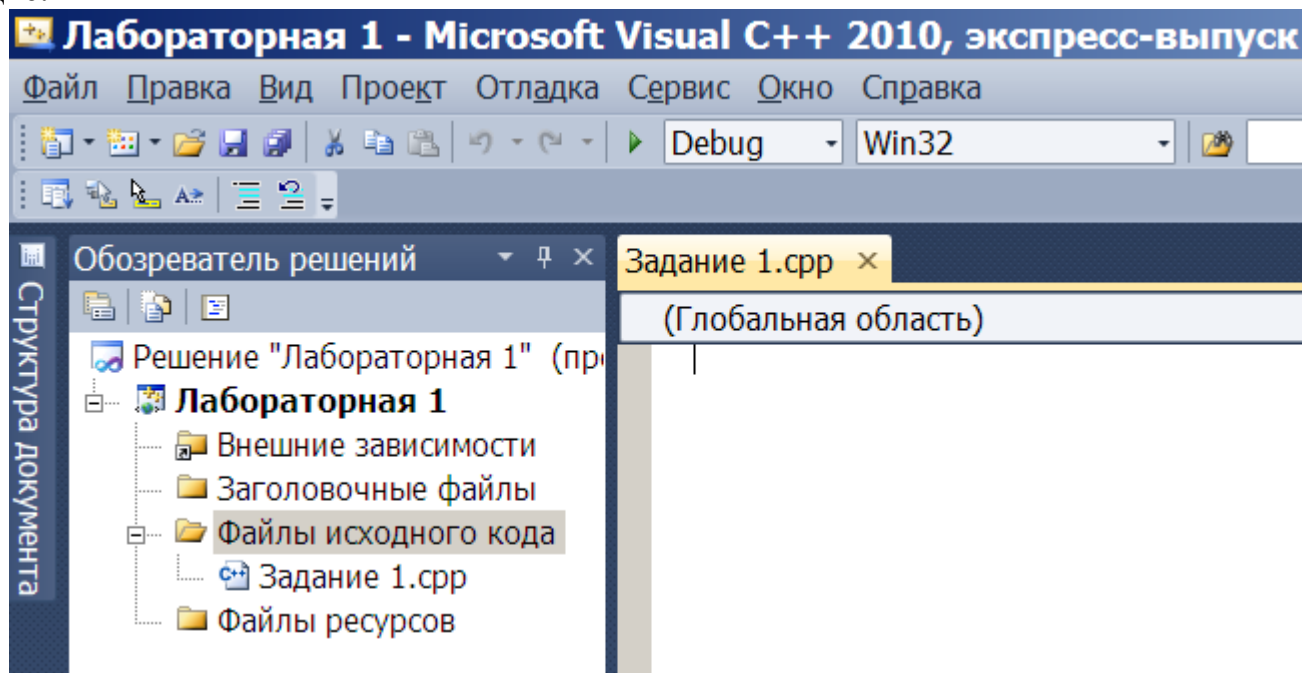
Создать пустой проект можно гораздо быстрее. Для этого в окне Создать проект вместо пункта **Консольное приложение Win32** следует выделить пункт **1. Пустой проект**. В этом случае проект будет создан сразу после нажатия кнопки **ОК**.

После создания пустого проекта необходимо добавить основной файл. Для этого в окне **Обозреватель решений** щелкаем правой кнопкой мыши на пункте **Файлы исходного кода** и из контекстного меню выбираем пункт **Добавить|Создать элемент**.

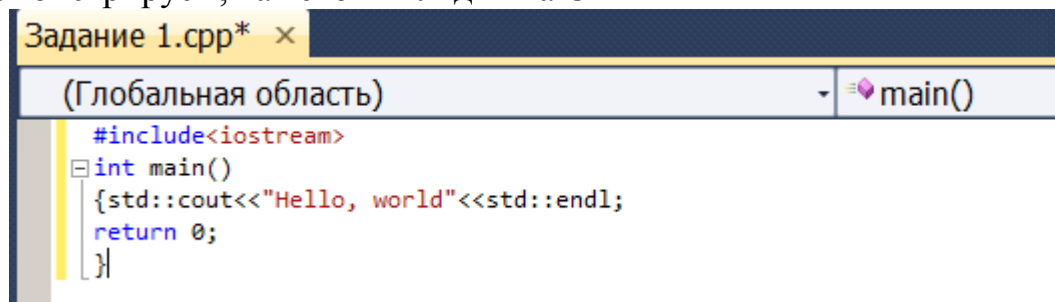


В результате откроется окно **Добавление нового элемента**. Выделяем пункт **Файл C++ (.cpp)**, вводим название файла (например, "Задание 1") и нажимаем кнопку **Добавить**. В результате файл будет добавлен в папку проекта и его название

отобразится в окне **Обозреватель решений**, а сам файл будет открыт на отдельной вкладке.



При изучении языков программирования принято начинать с программы, выводящей надпись "Hello, world!" в окно консоли. Не будем нарушать традицию и продемонстрируем, как это выглядит на C++

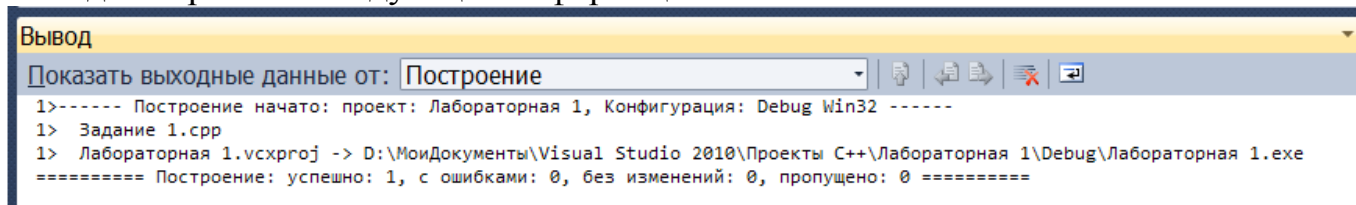


В первой строке программы с помощью директивы `include` подключается файл `iostream`, в котором объявлен объект `cout`, предназначенный для вывода данных в окно консоли. Далее создается функция `main()`, внутри которой расположены все остальные инструкции, ограниченные фигурными скобками. Именно функция с названием `main()` будет автоматически вызываться при запуске программы. Перед названием функции указывается тип возвращаемого значения. Ключевое слово `int` означает, что функция возвращает целое число. Возвращаемое значение указывается после ключевого слова `return` в самом конце функции `main()`. Число `0` в данном случае означает нормальное завершение программы. Если указано другое число, то это свидетельствует о некорректном завершении программы. Согласно стандарту, внутри функции `main()` ключевое слово `return` можно не указывать. В этом случае компилятор должен самостоятельно вставить инструкцию, возвращающую значение `0`.

Вывод строки "Hello, world!" осуществляется с помощью объекта `cout`. Чтобы исключить конфликт имен все стандартные идентификаторы в языке C++

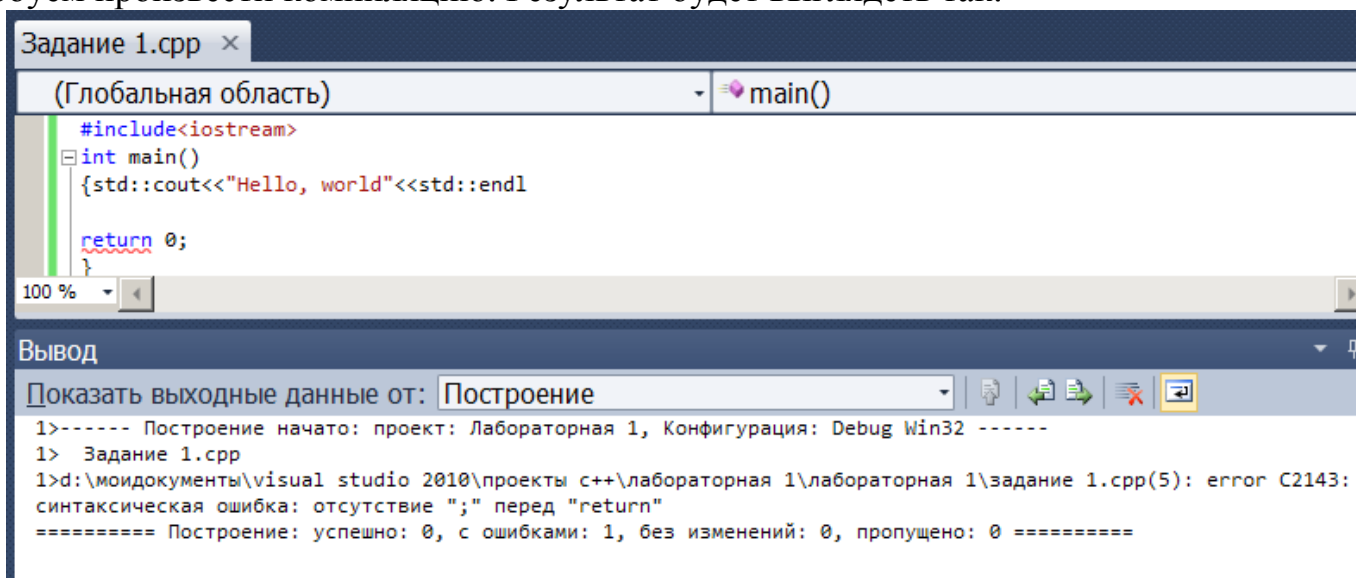
определены в пространстве имен **std**. Поэтому перед именем объекта `cout` необходимо указать, название пространства имен. Между названием пространства имен и названием объекта указываются два двоеточия (`std::cout`). И, наконец, константа `endl` (сокращение от "end line") вставляет символ перевода строки.

Теперь попробуем скомпилировать и запустить программу. Для этого добавляем содержимое листинга в созданный файл, а затем из меню **Отладка** выбираем пункт **Построить решение** или нажимаем клавишу **<F7>**. В результате в окне **Вывод** отобразится следующая информация:



```
Вывод
Показать выходные данные от: Построение
1>----- Построение начато: проект: Лабораторная 1, Конфигурация: Debug Win32 -----
1> Задание 1.cpp
1> Лабораторная 1.vcxproj -> D:\МоиДокументы\Visual Studio 2010\Проекты С++\Лабораторная 1\Debug\Лабораторная 1.exe
===== Построение: успешно: 1, с ошибками: 0, без изменений: 0, пропущено: 0 =====
```

Данный результат свидетельствует об успешном создании файла `helloworld.exe` в конфигурации **Debug** (отладка) в папке `D:\МоиДокументы\Visual Studio 2010 \Проекты С++\Лабораторная 1\Debug\`. Если при наборе были допущены ошибки, то их описание будет отображено в окне **Вывод**. В качестве примера уберем точку с запятой после константы `endl` и попробуем произвести компиляцию. Результат будет выглядеть так:



```
Задание 1.cpp x
(Глобальная область) main()
#include<iostream>
int main()
{std::cout<<"Hello, world"<<std::endl
return 0;
}
100 %
Вывод
Показать выходные данные от: Построение
1>----- Построение начато: проект: Лабораторная 1, Конфигурация: Debug Win32 -----
1> Задание 1.cpp
1>d:\моидокументы\visual studio 2010\проекты с++\лабораторная 1\лабораторная 1\задание 1.cpp(5): error C2143: синтаксическая ошибка: отсутствие ";" перед "return"
===== Построение: успешно: 0, с ошибками: 1, без изменений: 0, пропущено: 0 =====
```

Фраза "успешно: 0, с ошибками: 1" говорит о наличии ошибки в программе. Описание самой ошибки приводится строкой ранее. Указывается название файла, номер ошибки ("C2143"), а также подробное описание ошибки, в данном случае на русском языке. После исправления ошибки необходимо заново скомпилировать проект.

Чтобы посмотреть результат выполнения программы нажимаем комбинацию клавиш **<Ctrl>+<F5>** (Команда **Запуск без отладки**). В результате откроется окно консоли с приветствием, а также со строкой "Для продолжения нажмите любую клавишу". Нажатие любой клавиши приведет к закрытию окна консоли. Запустить программу можно также в режиме отладки. Для этого из меню **Отладка** выбираем пункт **Начать отладку** или нажимаем клавишу **<F5>**. В этом случае окно консоли откроется, затем сразу закроется, а в окне **Вывод** отобразится отладочная

информация. Как сделать, чтобы окно консоли сразу не закрывалось, мы рассмотрим далее.

Задание 1 Создать пстой проект как это указано выше. Набрать приведенный код. Обратите внимание, что при наборе кода редактор выводит списки команд для облегчения набора. Рассмотрите отличие запуска программы через комбинацию клавиш <Ctrl>+<F5> и <F5>.

Следует заметить, что при нажатии клавиши <F5> или комбинации клавиш <Ctrl>+<F5> производится проверка актуальности скомпилированного файла. Если файл устарел (в программе сделаны изменения), то будет выведено окно, в котором предлагается произвести повторную компиляцию. Чтобы выполнить повторную компиляцию нажимаем кнопку Да. Таким образом, можно не производить предварительную компиляцию, а сразу запускать программу.

Примечание

Если в меню **Отладка** нет пункта **Запуск без отладки**, то предварительно в меню **Сервис** необходимо выбрать пункт **Параметры | Расширенные параметры**

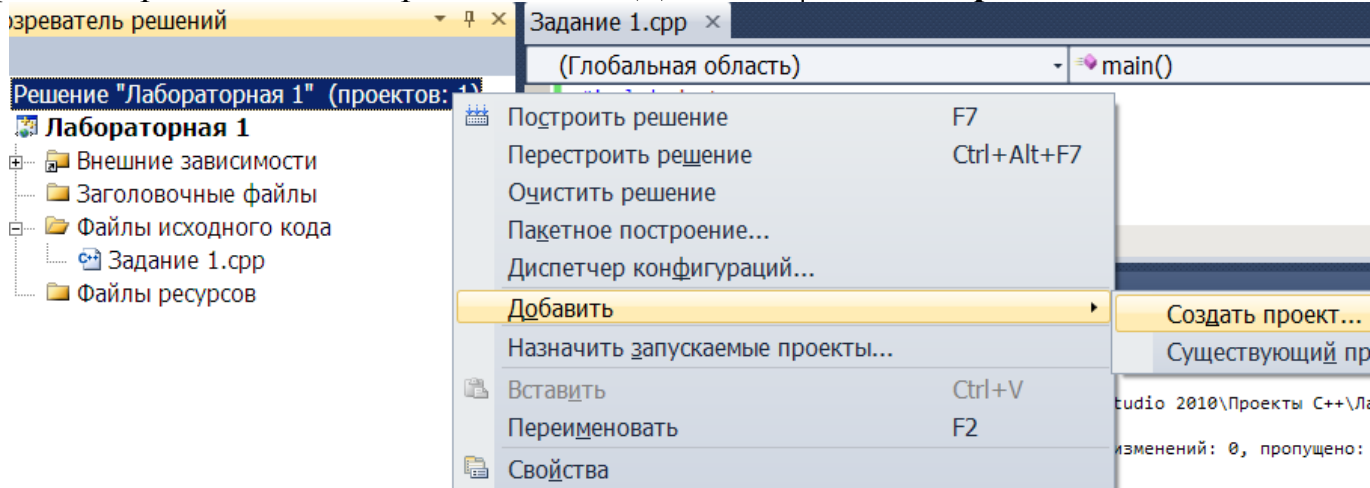
Создание консольного приложения

Чтобы сразу создать консольное приложение после запуска C++ на **Начальной странице** выбираем пункт **Создать проект**. В открывшемся окне выделяем пункт **Консольное приложение Win32**. Вводим название проекта (например, "welcome") в поле **Имя Введенное** название автоматически копируется в поле **Имя решения**. В поле **Расположение** указываем путь к каталогу, в котором будет сохранен проект, и устанавливаем флажок **Создать каталог для решения**. Нажимаем кнопку **ОК**. В результате откроется окно **Мастер приложений Win32**. Нажимаем кнопку **Далее**. В группе переключателей **Тип приложения** выбираем **Консольное приложение**. Устанавливаем флажок **Предварительно скомпилированный заголовок** **Флажок Пустой проект** должен быть снят. Нажимаем кнопку **Готово**.

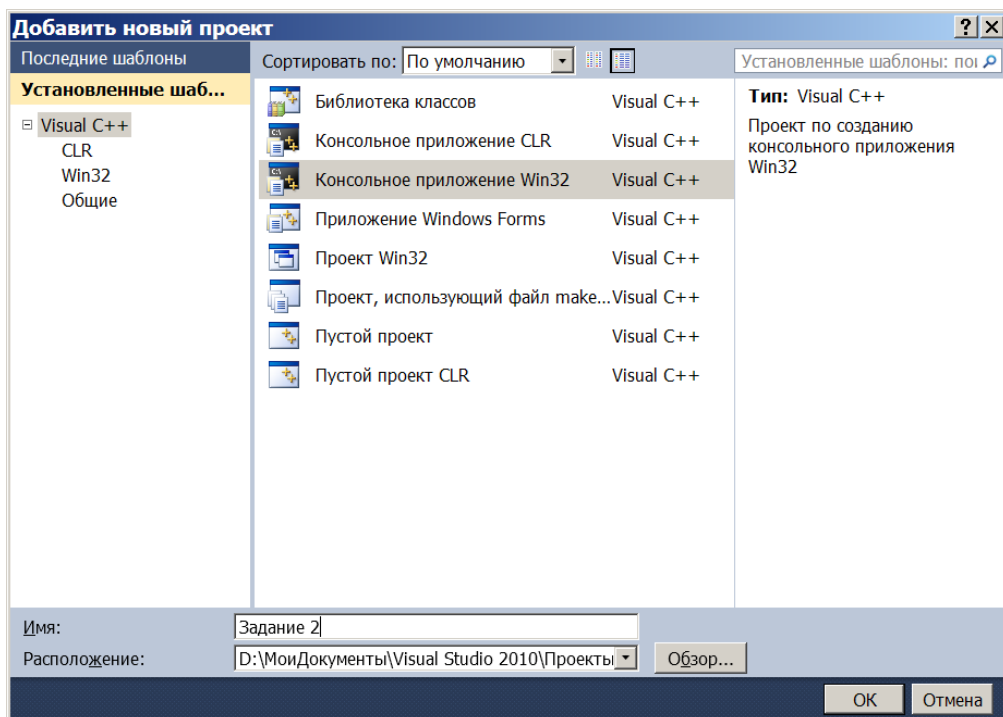
Добавление нового проекта

Создадим консольное приложение в нашем решении «Лабораторная 1».

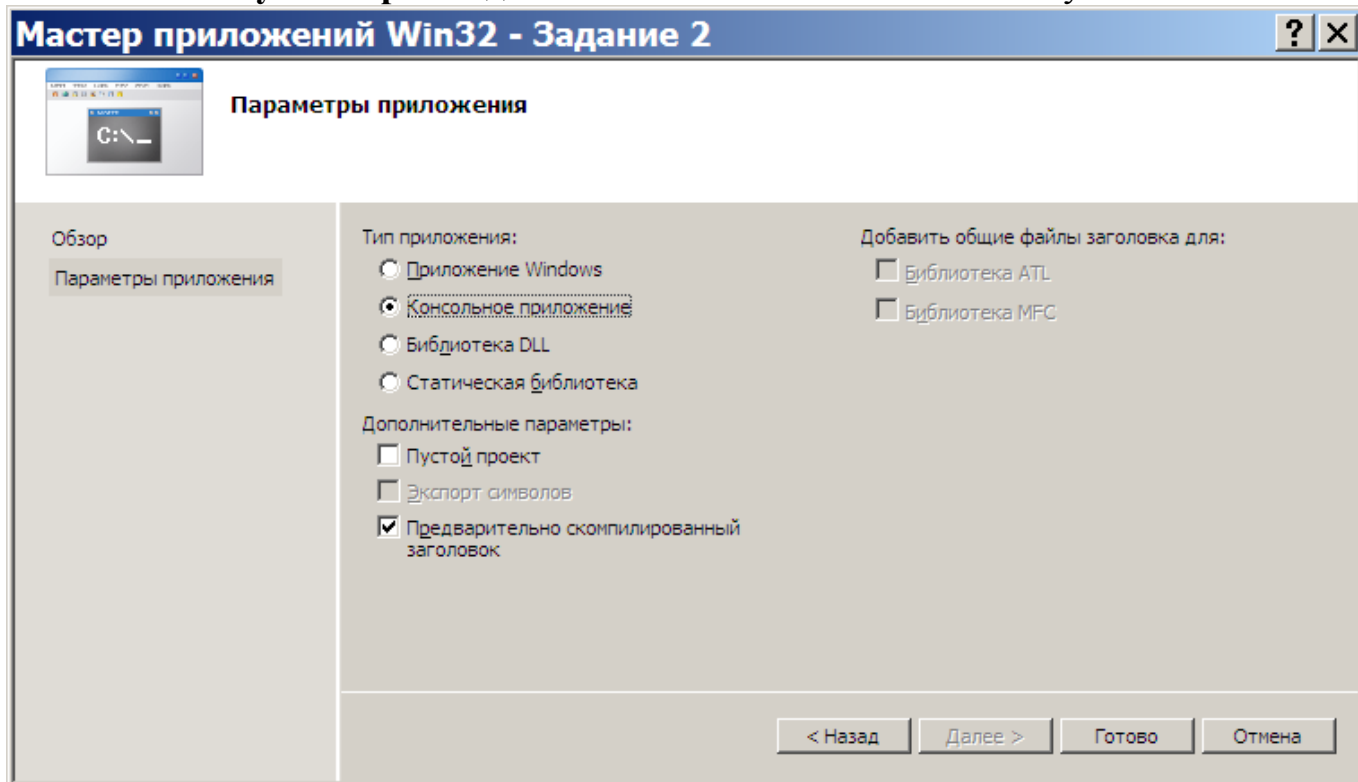
Для этого Щелкаем правой кнопкой мыши по решению Лабораторная 1 в Обозревателе решений и выбираем команды **Добавить | Создать проект**.



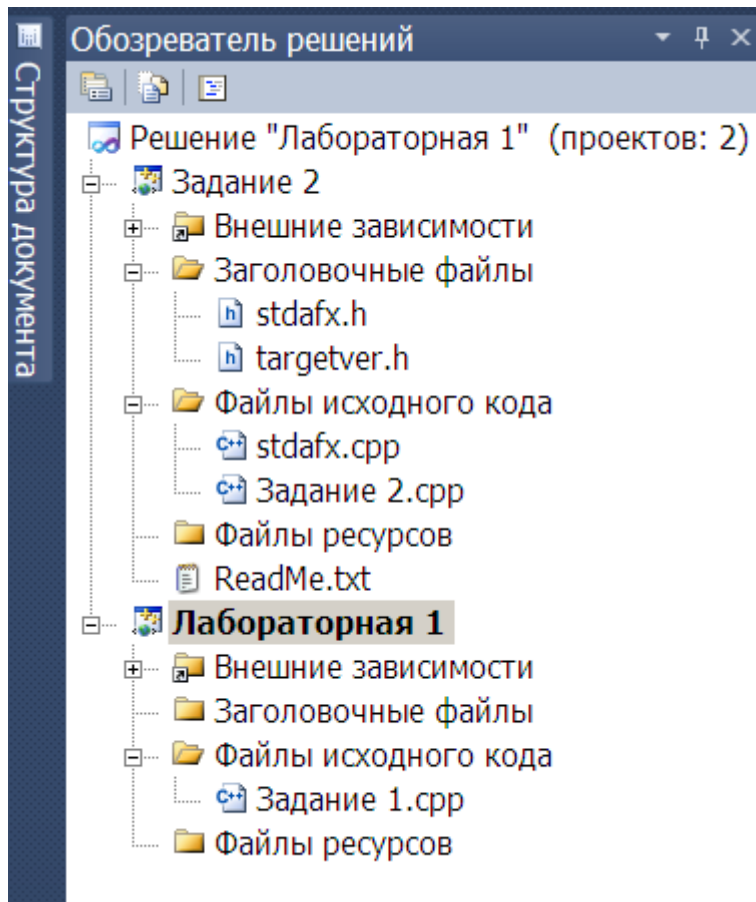
В открывшемся окне выделяем пункт **Консольное приложение Win32**. Вводим название проекта (например, "Задание 2") в поле **Имя** и щелкаем по кнопке **ОК**.



В результате откроется окно **Мастер приложений Win32**. Нажимаем кнопку **Далее**. В группе переключателей **Тип приложения** выбираем **Консольное приложение**. Устанавливаем флажок **Предварительно скомпилированный заголовок** Флажок **Пустой проект** должен быть снят. Нажимаем кнопку **Готово**.



В результате будет создан новый проект, файлы которого можно увидеть в **Обзревателе решений**. Рассмотрим, для чего предназначены файлы данного консольного приложения.



- ✓ Задание 2.cpp — основной файл проекта;
- ✓ stdafx.h — файл для подключения заголовочных файлов, используемых в проекте. Этот файл необходимо подключать во всех файлах проекта,
- ✓ stdafx.cpp — файлы stdafx.h и stdafx.cpp используются для построения файла предкомпилированного заголовка (Задание 2.pch) и файла предкомпилированных типов (stdafx.obj);
- ✓ targetver.h — в этом файле подключается файл SDKDDKVer.h, который обеспечивает определение последней доступной платформы Windows.

Файлы stdafx.cpp и targetver.h оставляем без изменений. В конец файла stdafx.h вставляем строку, позволяющую использовать в программе объект cout, предназначенный для вывода данных в окно консоли:

```
#include<iostream>
```

```
// stdafx.h: включаемый файл для стандартных системных включаемых файлов
// или включаемых файлов для конкретного проекта, которые часто используются, но
// не часто изменяются
#pragma once
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <iostream>
// TODO: Установите здесь ссылки на дополнительные заголовки, требующиеся для программы
```

Директива `pragma` со значением `once` указывает, что содержимое файла может быть включено только один раз. Директивы `include` подключают файлы, в которых содержатся объявления идентификаторов. Такие файлы называются *заголовочными файлами*. Они содержат только объявление идентификаторов без описания их реализации. Обратите внимание на то, что в директиве `include` заголовочные файлы подключаются разными способами. Поиск файлов, указанных внутри угловых скобок, производится в системных каталогах, а поиск файлов, указанных внутри кавычек, производится в каталоге проекта. Таким образом, названия файлов стандартной библиотеки необходимо указывать внутри угловых скобок, а пользовательских файлов внутри кавычек.

Три первых заголовочных файла содержат расширение `h`, а в последней директиве расширение файла не указано. Наличие расширения файла принято в стандартной библиотеке языка `C`. В стандартной библиотеке языка `C++` расширение файла принято не указывать. Так как язык `C++` наследует все библиотеки языка `C`, то файлы можно подключать как в стиле языка `C`, так и в стиле языка `C++`. Например, файл `string.h` из стандартной библиотеки языка `C` доступен в языке `C++` под названием `cstring`, а файл `math.h` под названием `cmath`. Отличие между этими способами подключения заключается в импортировании идентификаторов. В языке `C` при подключении файла (например, `math.h`) все идентификаторы импортируются в глобальное пространство имен, а в языке `C++` при подключении файла (например, `cmath`) идентификаторы определяются в пространстве имен под названием `std`. Поэтому перед идентификатором необходимо указать название пространства имен (например, `std::cout`). Использование пространств имен позволяет избежать конфликта имен в программе.

Чтобы посмотреть содержимое заголовочного файла (например, `stdio.h`) щелкните правой кнопкой мыши на строке `#include <stdio.h>` и из контекстного меню выберите пункт **Открыть документ <stdio.h>**. В результате заголовочный файл `stdio.h` будет открыт на отдельной вкладке. Следует заметить, что файл `stdio.h` содержит объявления идентификаторов, предназначенных для операций ввода/вывода в языке `C`. Так как для вывода мы используем объект `cout`, объявленный в файле `iostream`, то подключение файла `stdio.h` можно вообще удалить из файла `stdafx.h`. Производить операции ввода/вывода мы будем только в стиле языка `C++`.

Теперь рассмотрим содержимое файла `Задание 2.cpp`

```
Задание 2.cpp × Задание 1.cpp
(Глобальная область)
// Задание 2.cpp: определяет точку входа для консольного приложения.
//
#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])
{
    return 0;
}
```

Добавим текст программы в файл Задание 2.cpp

```
Начальная страница stdafx.h Задание 2.cpp ×
(Глобальная область)
// Задание 2.cpp: определяет точку входа для консольно
//
#include "stdafx.h"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
    cout<<"Welcome to C++!"<<endl;
    return 0;
}
```

Сравните этот листинг с предыдущим примером Задание 1.cpp. Первое, что должно броситься в глаза— это название функции (вместо названия `main()` используется название `_tmain()`) и наличие двух параметров. Параметры расположены внутри круглых скобок и разделены запятой. Параметр состоит из объявления типа данных и названия. Первый параметр (`argc`) имеет тип `int`, который соответствует целочисленному значению. Через этот параметр доступно количество аргументов, переданных в командной строке. Следует учитывать, что первым аргументом является название исполняемого файла. Поэтому значение параметра `argc` не может быть меньше единицы. Через второй параметр (`argv`) доступны все аргументы в виде строки (тип `_TCHAR*`). Квадратные скобки после названия второго параметра означают, что доступен массив строк.

Для вывода строки "Welcome to C++!", как и в предыдущем примере, используется объект `cout`, объявленный в файле `iostream`. Файл `iostream` мы подключили в файле `stdafx.h`, поэтому повторно подключать его не нужно. Достаточно подключить только файл `stdafx.h` в начале программы.

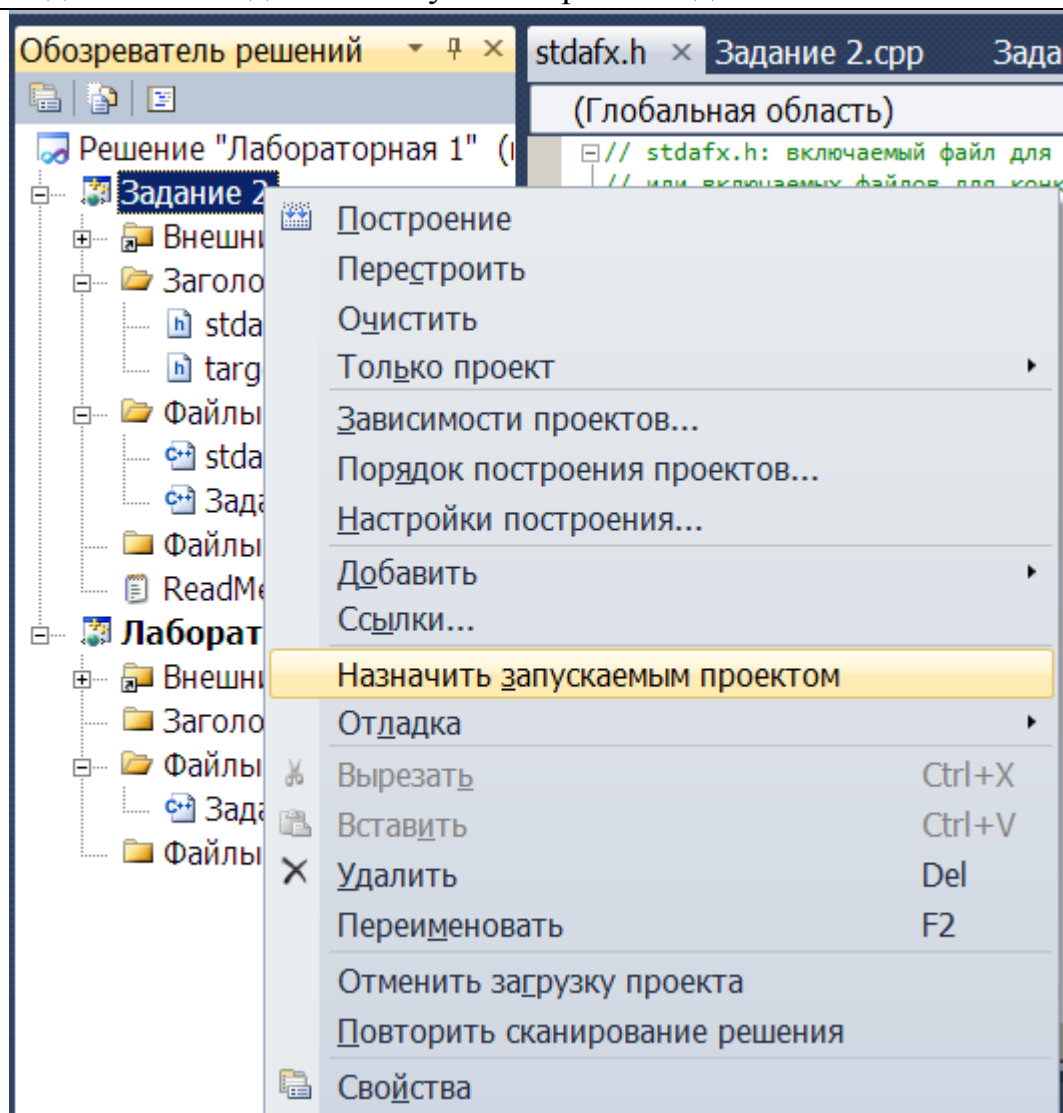
Обратите внимание на то, что перед объектом `cout` и константой `endl` не указано название пространства имен `std`, так как все идентификаторы, объявленные в пространстве имен `std` были импортированы в основную программу с помощью инструкции:

```
using namespace std;
```

Попробуем запустить проект Задание 2, выполнив уже описанные команды.

Но при запуске будет выполняться предыдущий проект. Для того, чтобы в рамках одного решения запускать разные проекты надо выполнить определенные действия. Перед запуском нового проекта надо в **Обозревателе решений** щелкнуть правой кнопкой по названию нового проекта и выбрать команду **Назначит запускаемым проектом** (см рисунок). После этого для запуска без отладки нажимаем комбинацию клавиш <Ctrl>+<F5>. Если нужно просто скомпилировать проект, то нажимаем клавишу <F7>. Запуск с отладкой осуществляется нажатием клавиши <F5>. Запомните эти комбинации клавиш наизусть.

Задание 2. Создайте и запустите проект Задание 2



Структура программы

Запускать программу мы научились, теперь можно приступать к изучению языка C++. Как видно из предыдущих примеров, программа состоит из инструкций, расположенных в текстовом файле. Структура обычной программы выглядит так:

```
<Подключение заголовочных файлов>  
<Объявление глобальных переменных>  
<Объявление функций, классов и др.>  
int main([ int argc, char *argv[] ])
```

```

{
<Инструкции>
return 0;
}
<Определение функций и классов>

```

В самом начале программы подключаются заголовочные файлы, в которых содержатся объявления идентификаторов без их реализации. Например, чтобы иметь возможность вывести данные в окно консоли необходимо подключить файл `iostream`, в котором содержится объявление объекта `cout`. Подключение осуществляется с помощью директивы `include`:

```
#include <iostream>
```

С помощью директивы `include` можно подключать как стандартные заголовочные файлы, так и пользовательские файлы.

После подключения файлов производится объявление *глобальных переменных*. Переменные предназначены для хранения значений определенного типа. Глобальные переменные видны во всей программе, включая функции. Если объявить переменную внутри функции, то область видимости переменной будет ограничена рамками функции и в других частях программы использовать переменную нельзя. Такие переменные называются *локальными*. Объявить целочисленную переменную можно так:

```
int x;
```

В этом примере мы объявили переменную с названием "x" и указали, что переменная может хранить данные, имеющие тип `int`. Ключевое слово `int` означает, что переменная предназначена для хранения целого числа. Попытка присвоить переменной значение, имеющее другой тип приведет к предупреждению или ошибке. Исключением являются ситуации, в которых компилятор может произвести преобразование типов данных без потери точности. Например, целое число без проблем преобразуется в вещественное число, однако попытка преобразовать вещественное число в целое приведет к выводу предупреждающего сообщения, даже если после точки стоит ноль. Пример сообщения:

```
warning C4244:: преобразование "double" в "int",
возможна потеря данных
```

Хотя компилятор предупреждает о потере данных, программа все равно будет скомпилирована. Поэтому обращайте внимание на сообщения, которые выводятся в окне Вывод на вкладке **Построение** при компиляции. Иначе программа будет работать, но результат выполнения будет некорректным.

Обратите внимание на то, что объявление переменной заканчивается точкой с запятой. Большинство инструкций в языке C++ должно заканчиваться точкой с запятой. Если точку с запятой не указать, то компилятор сообщит о синтаксической ошибке. Пример сообщения:

```
error C2144: синтаксическая ошибка: перед "int"
требуется ";"
```

Чтобы увидеть строку, о которой сообщает компилятор, сделайте двойной щелчок мышью на сообщении в окне **Вывод**. В результате строка с ошибкой в

программе станет активной и слева от нее отобразится маркер. В нашем случае активной будет строка, расположенная сразу после объявления глобальной переменной.

При объявлении можно сразу присвоить начальное значение переменной. Присваивание значения переменной при объявлении называется *инициализацией переменной*. Чтобы произвести инициализацию переменной после ее названия указывается оператор =, а затем значение. Пример:

```
int x=10;
```

Если глобальной переменной не присвоено значение при объявлении, то она будет иметь значение 0. Если локальной переменной не присвоено значение, то переменная будет содержать произвольное значение. Как говорят в таком случае переменная содержит "мусор".

Обратите внимание на то, что перед оператором = и после него вставлены пробелы. Количество пробелов может быть произвольным или пробелов может не быть вовсе. Кроме того, допускается вместо пробелов использовать символ перевода строки или символ табуляции. Например, эти инструкции вполне допустимы:

```
int    x=21;
int    y=                                85;
int    z
56;
```

Тем не менее, следует придерживаться единообразия в коде и обрамлять операторы одним пробелом. Если присваивание производится в объявлении функции (задается значение параметра по умолчанию), то пробелы принято не указывать вовсе. Следует учитывать, что это не строгие правила, а лишь рекомендации по оформлению кода.

Как видно из предыдущего примера, инструкция может быть расположена на нескольких строках. Концом инструкции является точка с запятой, а не конец строки. Например, на одной строке может быть несколько инструкций:

```
int x=21; int y=85; int z =56;
```

Из этого правила есть исключения. Например, после директив препроцессора точка с запятой не указывается. В этом случае концом инструкции является конец строки. Директиву препроцессора можно узнать по символу # перед названием директивы. Типичным примером директивы препроцессора является директива include, которая используется для подключения заголовочных файлов:

```
#include <iostream>
```

После объявления глобальных переменных могут располагаться объявления функций и классов. Такие объявления называются *прототипами*. Схема прототипа функции выглядит следующим образом:

```
<Тип возвращаемого значения> <Название функции> ([<Тип>
[<Параметр1>] [, ..., <Тип> [<Параметр N>] ] ]);
```

Например, прототип функции, которая складывает два целых числа и возвращает их сумму выглядит так:

```
int    summa (int    x,    int    y) ;
```

После объявления функции необходимо описать, ее реализацию, которая называется *определением функции*. Определение функции обычно располагается после определения функции `main()`. Обратите внимание на то, что объявлять прототип функции `main()` не нужно, так как определение этой функции обычно расположено перед определением других функций. Пример определения функции `summa()`:

```
int summa(int x, int y)    {  
    return x + y; }
```

Как: видно из примера, первая строка в определении функции `summa()` совпадает с объявлением функции. Следует заметить, что в объявлении функции можно не указывать названия параметров. Достаточно указать информацию о типе данных. Таким образом, объявление функции можно записать так:

```
int summa(int, int);
```

После объявления функции ставится точка с запятой. В определении функции внутри фигурных скобок должна быть описана реализация функции. Так как в объявлении указано, что функция возвращает целочисленное значение, следовательно, после описания реализации необходимо вернуть значение. Возвращаемое значение указывается после ключевого слова `return`. Если функция не возвращает никакого значения, то перед названием функции вместо типа данных указывается ключевое слово `void`. Пример объявления функции, которая не возвращает значения:

```
void print(int);
```

Пример определения функции `print()`:

```
void print(int x)    {  
    cout << x << endl; }
```

Вся реализация функции должна быть расположена внутри фигурных скобок. Открывающая скобка может находиться на одной строке с определением функции, как в предыдущем примере, или вначале следующей строки. Пример:

```
void print(int x)  
{  
    cout << x << endl; }
```

Многие программисты считают такой стиль наиболее приемлемым, так как открывающая и закрывающая скобки расположены друг под другом. У этого стиля есть определенный недостаток: образуется лишняя пустая строка. Так как размеры экрана ограничены, при наличии пустой строки на экран помещается меньше кода и приходится чаще пользоваться полосой прокрутки. Если размещать инструкции с равным отступом, то блок кода выделяется визуально и следить за положением фигурных скобок просто излишне. Тем более, что редактор кода позволяет подсветить парные скобки. Чтобы найти пару фигурных скобок следует поместить указатель ввода перед скобкой. В результате скобки будут подсвечены. Какой стиль использовать, зависит от личного предпочтения программиста или от правил оформления кода, принятых в определенной фирме. Главное, чтобы стиль оформления внутри одной программы был одинаковым.

Перед инструкциями внутри фигурных скобок следует размещать одинаковый отступ. В качестве отступа можно использовать пробелы или символ табуляции. При использовании пробелов размер отступа равняется трем или четырем пробелам для блока первого уровня. Для вложенных блоков количество пробелов умножают на уровень вложенности. Если для блока первого уровня вложенности использовалось три пробела, то для блока второго уровня вложенности должно использоваться шесть пробелов, для третьего уровня — девять пробелов и т. д. В одной программе не следует использовать и пробелы и табуляцию в качестве отступа. Необходимо выбрать что-то одно и пользоваться этим во всей программе.

Закрывающая фигурная скобка обычно размещается в конце блока на отдельной строке. После скобки точка с запятой не указывается. Однако наличие точки с запятой ошибкой не является, так как инструкция может не содержать выражений вообще. Например, такая инструкция вполне допустима, хотя она ничего не делает:

Самой главной функцией в программе является функция `main()`. Именно функция с названием `main()` будет автоматически вызываться при запуске программы. Функция имеет два прототипа:

```
int main();  
int main(int argc, char *argv[]);
```

Первый прототип функции имеет синоним:

```
int main(void);
```

Значение `void` означает, что функция не принимает параметры. Подобный синтаксис используется в языке программирования C. В языке C++ указание ключевого слова `void` является излишним. Достаточно указать пустые скобки.

Второй прототип применяется для получения значений, указанных при запуске программы из командной строки. Количество значений доступно через параметр `argc`, а сами значения через параметр `argv`.

Перед названием функции указывается тип возвращаемого значения. Ключевое слово `int` означает, что функция возвращает целое число. Возвращаемое значение указывается после ключевого слова `return` в самом конце функции `main()`. Число 0 означает нормальное завершение программы. Если указано другое число, то это свидетельствует о некорректном завершении программы. Согласно стандарту, внутри функции `main()` ключевое слово `return` можно не указывать. В этом случае компилятор должен самостоятельно вставить инструкцию, возвращающую значение 0. Возвращаемое значение передается операционной системе и может использоваться для определения корректности завершения программы.

В листинге 1.4 приведен пример программы, структуру которой мы рассмотрели в этом разделе. Не расстраивайтесь, если что-то показалось непонятным. Все это мы будем изучать более подробно в дальнейшем. На этом этапе достаточно лишь представлять структуру программы и самое главное уметь ее скомпилировать и запустить на выполнение

Пример программы

```
//Подключение заголовочных файлов  
#include <iostream>
```

```

using namespace std;
//Объявление глобальных переменных
int x =21;
int y =85;
//Объявление функций, классов и др.
int summa(int x,int y) ;
void print(int x);
// Главная функция
int main () {
int z; // Объявление локальной переменной
z= summa(x,y); //Вызов функции summa()
print (z);      //Вызов функции print ()
return 0;
}
//Определение функций и классов
int summa(int x, int y)      {
return x+y; }
void print(int x)          {
cout<<x<<endl;
}

```

Комментарии в программе

Комментарии предназначены для вставки пояснений в текст программы и компилятор полностью их игнорирует. Внутри комментария может располагаться любой текст, включая инструкции, которые выполнять не следует. Помните, комментарии нужны программисту, а не компилятору. Вставка комментариев в код позволит через некоторое время быстро вспомнить предназначение фрагмента кода.

В языке C++ присутствуют два типа комментариев: однострочный и многострочный. *Однострочный комментарий* начинается с символов // и заканчивается в конце строки. Вставлять однострочный комментарий можно как в начале строки, так и после инструкции. Если символ комментария разместить перед инструкцией, то она не будет выполнена. Если символы // расположены внутри кавычек, то они не являются признаком начала комментария. Примеры однострочных комментариев:

```

//Это комментарий
cout<<"Hello, world!"; //Это комментарий
//cout<<"Hello, world!"; //Инструкция выполнена не
будет
char s[] = "// Это НЕ комментарий!!!";

```

Многострочный комментарий начинается с символов /* и заканчивается символами */. Комментарий может быть расположен как на одной строке, так и на нескольких.


Кроме того, много строчный комментарий можно размещать внутри выражения, хотя это и нежелательно. Следует иметь в виду, что много строчные комментарии не могут быть вложенными, поэтому при комментировании больших блоков следует проверять, что в них не встречается закрывающая комментарий комбинация символов */. Тем не менее,


однострочный комментарий может быть расположен внутри много строчного комментария. Примеры много строчных комментариев:

```
*/
Многострочный комментарий
*/
cout<<"Hello, world!"; /* Это комментарий */
/*cout<<"Hello, world!";// Инструкция выполнена не будет
*/

int x;
x = 10 /* Комментарий */ + 50 /* внутри выражения */;
char s[] = "/* Это НЕ комментарий!!! */";
```

Редактор в VC++ позволяет быстро закомментировать фрагмент кода. Для этого необходимо выделить одну инструкцию (или сразу несколько инструкций), а затем из меню **Правка** выбрать пункт **Дополнительно|Преобразовать выделенный фрагмент в комментарий**. Можно также нажать кнопку

Закомментировать выделенные строки  на панели инструментов **Текстовый редактор**. Если панель инструментов не отображена, в меню Вид выбираем пункт **Панели инструментов | Текстовый редактор**. Кроме того, можно воспользоваться клавишами быстрого доступа. Для этого следует нажать комбинацию клавиш **<Ctrl>+<K>**, а затем комбинацию клавиш **<Ctrl>+<C>**. Если строка выделена полностью, то перед ней будет вставлен однострочный комментарий. Если выделен лишь фрагмент строки, то он будет обернут в много строчный комментарий.

Чтобы убрать комментарий следует выделить фрагмент, а затем из меню **Правка** выбрать пункт **Дополнительно | Отменить преобразование в комментарий**. Можно также нажать кнопку **Отменить преобразование выделенных строк**  в комментарий на панели инструментов **Текстовый редактор**. Кроме того, можно воспользоваться клавишами быстрого доступа. Для этого следует нажать комбинацию клавиш **<Ctrl>+<K>**, а затем комбинацию клавиш **<Ctrl>+<U>**.

Вывод данных в языке C++

Для вывода данных в языке C++ предназначены объекты `cout`, `cerr` и `clog`, объявленные в файле `iostream`. Объект `cout` используется для вывода обычных сообщений в окно консоли. Объекты `cerr` и `clog` применяются для вывода сообщений об ошибках. Также как и объект `cout` объекты `cerr` и `clog` первоначально связаны с окном консоли, однако возможно перенаправить поток на другое устройство или в файл. Для понимания всех возможностей этих объектов требуется знание принципов объектно-ориентированного программирования (ООП). Так как ООП мы еще не изучали, в этом разделе будут рассматриваться только основные возможности объектов.

Прежде чем использовать объекты необходимо подключить файл `iostream` с помощью директивы `include`:

```
#include <iostream>
```

Обратите внимание на то, что название файла указывается внутри угловых скобок без расширения `h`, так как файл `iostream` входит в состав стандартной библиотеки `C++`. После подключения файла все объекты будут доступны через пространство имен `std`, поэтому при обращении к объекту необходимо указать название пространства имен и два символа двоеточия перед названием объекта. Например, вывести строку "Hello,

world!" можно так:

```
std::cout<<"Hello, world!";
```

Если каждый раз указывать название пространства имен лень, то можно импортировать все идентификаторы в глобальное пространство имен с помощью инструкции:

```
using namespace std;
```

В этом случае название пространства имен указывать не нужно:

```
Cout<<"Hello, world!";
```

После названия объекта указывается оператор `<<`, который как бы указывает направление вывода. После оператора `<<` передается объект, который будет выведен в окне консоли. В качестве объекта можно указать число, строку, а также объект, в котором оператор `<<` перегружен. Пример:

```
cout<<10; // Целое число
cout<<81.5; //Вещественное число
cout<<"Hello, world!"; //Строка
char str[]="Hello, world!";
cout <<str; //Строка
```

Результат выполнения этого примера будет таким:

```
1081.5Hello, world!Hello, world!
```

Как видно из результата, все данные выводятся на одной строке. Чтобы данные выводились на отдельных строках можно воспользоваться константой `endl` (сокращение от "end line"). Пример:

```
cout<<"String1";
cout<<endl;
cout<<"String2";
```

Результат в окне консоли:

```
String1
String2
```

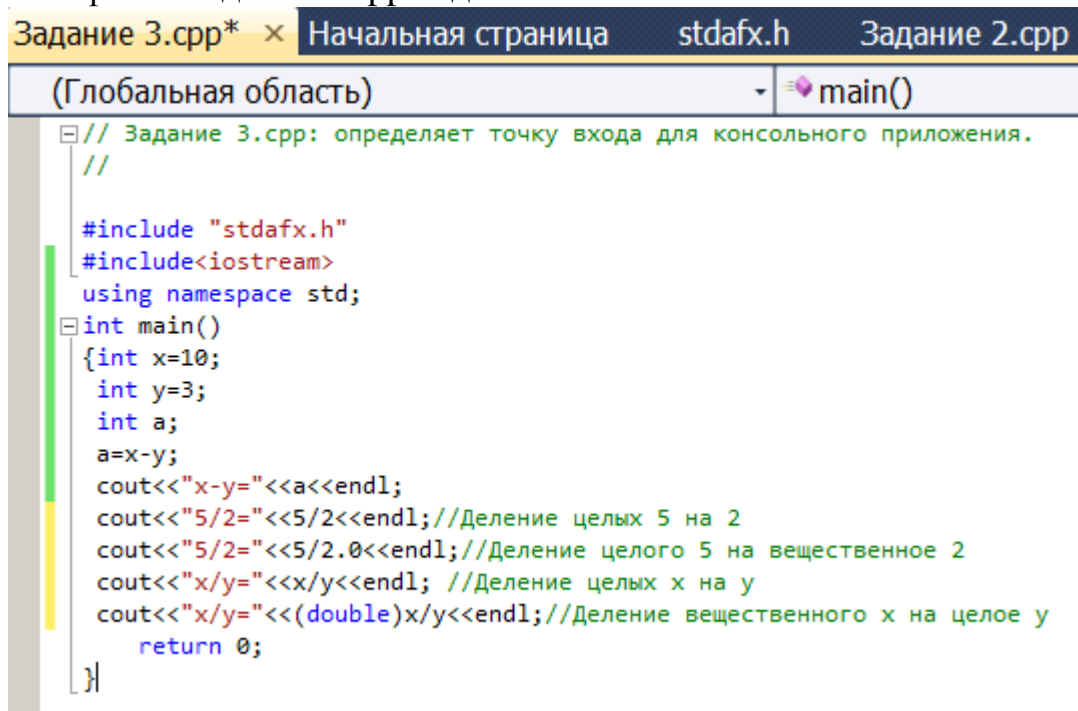
На самом деле `endl` не совсем константа. Можно сказать, что это функция (называемая манипулятором), внутри которой производится вывод символа перевода строки и возвращается ссылка на поток вывода. Благодаря возврату ссылки можно строить цепочки из операторов `<<` Например, предыдущий пример можно записать так:

```
cout<<"String1"<<endl<<"String2";
```

Для перевода строки можно также воспользоваться комбинацией символов `"\n"`, которые обозначают символ перевода строки. Пример:

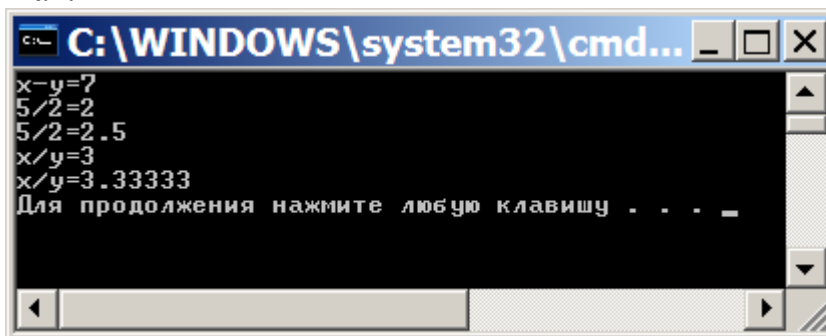
```
cout<<"String1"<<"\n"<<"String2";
```


Задание 3 Создать новый проект «Задание 3» в решении «Лабораторная 1». Набрать в файле Задание 3.cpp код:



```
Задание 3.cpp* x Начальная страница stdafx.h Задание 2.cpp
(Глобальная область) main()
// Задание 3.cpp: определяет точку входа для консольного приложения.
//
#include "stdafx.h"
#include<iostream>
using namespace std;
int main()
{int x=10;
 int y=3;
 int a;
 a=x-y;
 cout<<"x-y="<<a<<endl;
 cout<<"5/2="<<5/2<<endl; //Деление целых 5 на 2
 cout<<"5/2="<<5/2.0<<endl; //Деление целого 5 на вещественное 2
 cout<<"x/y="<<x/y<<endl; //Деление целых x на y
 cout<<"x/y="<<(double)x/y<<endl; //Деление вещественного x на целое y
 return 0;
}
```

Установите Задание 3 загружаемым проектом и запустите без отладки. Результат:



```
C:\WINDOWS\system32\cmd...
x-y=7
5/2=2
5/2=2.5
x/y=3
x/y=3.33333
Для продолжения нажмите любую клавишу . . .
```

Обратите внимание на разные результаты деления. Это объясняется тем, что в С результат арифметических операций имеет тот же тип, что и данные, участвующие в этой операции. Операция вывода cout не позволяет форматировать выводимые данные. Для форматированного вывода используются другие операции из другой библиотеки.

Форматированный вывод данных в языке С

Язык С++ поддерживает также операции вывода из языка С, объявленные в файле stdio.h. В языке С++ вместо этого файла можно подключать файл cstdio:

```
# include <cstdio>
```

Для вывода одиночного символа применяется функция putchar(). Пример вывода символа:

```
putchar('w'); // w
putchar(119); // w
```

Вывести строку позволяет функция `puts()`. Функция выводит строку `Str` и вставляет символ перевода строки. Пример:

```
puts("String1");  
puts("String2");
```

Результат выполнения:

```
String1  
String2
```

Для форматированного вывода используется функция `printf()`. Прототип функции:

```
int printf(const char *Format, ...) ;
```

В параметре `Format` указывается строка специального формата. Внутри этой строки можно указать обычные символы и спецификаторы формата, начинающиеся с символа `%`. Вместо спецификаторов формата подставляются значения, указанные в качестве параметров. Количество спецификаторов должно совпадать с количеством переданных параметров. В качестве значения функция возвращает количество выведенных символов. Пример вывода строки и числа:

```
printf ("String\n") ;  
printf("Count %d\n", 10);  
printf("%s %d\n", "Count", 10);
```

Результат выполнения:

```
String  
Count 10  
Count 10
```

В первом примере строка формата не содержит спецификаторов и выводится как есть. Во втором примере внутри строки формата используется спецификатор `%d`, предназначенный для вывода целого числа. Вместо этого спецификатора подставляется число 10, переданное во втором параметре. В третьем примере строка содержит сразу два спецификатора `%s` и `%d`. Спецификатор `%s`, предназначен для вывода строки, а спецификатор `%d`— для вывода целого числа. Вместо спецификатора `%s` будет подставлена строка "Count", а вместо спецификатора `%d`— число 10. Обратите внимание на то, что тип данных переданных значений должен совпадать с типом спецификатора. Если в качестве значения для спецификатора `%s` указать число, то это приведет в ошибку времени исполнения. Никакой проверки соответствия типа на этапе компиляции не производится.

Спецификаторы имеют следующий синтаксис:

```
%[<Модификаторы>] [<Ширина>] [.<Точность>] <Тип>
```

В параметре `<Тип>` могут быть указаны следующие символы:

→ `c` — символ:

```
printf("%c", 'w'); //w  
printf{"%c", 119); //w
```

→ `s` - строка:

```
printf("%s", "String"); //String
```

→ `d` или `i` - десятичное целое число со знаком:

```
printf("%d %i", 10, 30); //10 30  
printf("%d %i", -10, -30); //-10 -30
```

→ `u` - десятичное целое число без знака:

```
printf("%u",10); //10
```

→ o - восьмеричное число без знака:

```
printf("%o %o",10,077); / 12 77
```

```
printf("%#o %#o",10,077); // 012 077
```

→ f - вещественное число в десятичном представлении:

```
printf("%f %f",18.65781452,12.5); //18.657815 12.500000
```

```
printf("%f",-18.65781452); //-18.657815
```

```
printf("%#.0f %.0f",100.0,100.0); //100. 100
```

→ e - вещественное число в экспоненциальной форме (буква "e" в нижнем регистре):

```
printf ("%e",18657.81452); //1.865781e+004
```

```
printf("%e",0.000081452); //8.145200e-005
```

→ E - вещественное число в экспоненциальной форме (буква "e" в верхнем регистре):

```
printf("%E",18657.81452); //1.865781E+004
```

→ g - эквивалентно f или e (выбирается более короткая запись числа):

```
printf("%g %g %g",0.086578,0.000086578,1.865E-005);
```

```
//0.086578 8.6578e-005 1.865e-005
```

→ G - эквивалентно f или E (выбирается более короткая запись числа):

```
printf("%G %G %G",0.086578,0.000086578,1.865E-005);
```

```
//0.086578 8.6578E-005 1.865E-005
```

→ % - символ процента (%):

```
printf("10%%"); //10%
```

Параметр <Ширина> задает минимальную ширину поля. Если строка меньше ширины поля, то она дополняется пробелами. Если строка не помещается в указанную ширину, то значение игнорируется и строка выводится полностью:

```
printf("'%3s'", "string"); //'string'
```

```
printf("'%10s'", "string"); //'      string'
```

Задать минимальную ширину можно не только для строк, но и для других типов:

```
printf("'%10d'",25); //'          25'
```

```
printf("'%10f'",12.5); //' 12.500000'
```

Параметр <Точность> задает количество знаков после точки для вещественных чисел. Перед этим параметром обязательно должна стоять точка. Пример:

```
printf("'%10.5f'",3.14159265359); //'    3.14159'
```

```
printf("'%.3f'",3.14159265359); //'3.142'
```

Если параметр <точность> используется применительно к целому числу, то он задает минимальное количество цифр. Если число содержит меньшее количество цифр, то вначале числа добавляются нули. Пример:

```
printf("'%7d'",100); //'          100'
```

```
printf("'%.7d'",100); //'0000100'
```

```
printf("'%.7d'",123456739); //'123456739'
```

Если параметр <Точность>;- используется применительно к строке, то он задает максимальное количество символов. Символы, которые не помещаются будут отброшены.

Пример:

```
printf("%5.7s","Hello, world!"); //'Hello, \
printf("%15.20s","Hello, world ! "); //' Hello,
world!'
```

Вместо минимальной ширины и точности можно указать символ *. В этом случае значения передаются через параметры функции printf () в порядке указания символов в строке формата. Примеры:

```
printf("%*.*f",10,5,3.14159265359); //' 3.14159'
printf("%.*f",3,3.14159265359); //'3.142'
printf("%*s",10,"string"); //' string'
```

В первом примере вместо первого символа * подставляется число 10, указанное во втором параметре, а вместо второго символа * подставляется число 5, указанное в третьем параметре. Во втором примере, вместо символа * подставляется число 3, которое задает количество цифр после точки. В третьем примере символ * заменяется числом 10, которое задает минимальную ширину поля.

В параметре <Модификаторы> могут быть указаны следующие символы:

→ # - для вещественных чисел указывает всегда выводить дробную точку, даже если задано значение 0 в параметре <Точность>:

```
printf("%#.0f %.0f",100.0,100.0); //100. 100
```

→ 0 — задает наличие ведущих нулей для числового значения:

```
printf ("%07d",100); //' 100'
printf ("%07d",100); //'0000100'
```

→ - - задает выравнивание по левой границе области. По умолчанию используется выравнивание по правой границе. Пример:

```
printf ("%5d' %-5d'",3,3); //' 3' '3'
printf ("%05d' %-05d'",3,3); //'00003' '3'
```

→ пробел - вставляет пробел перед положительным числом. Перед отрицательным числом будет стоять минус. Пример:

```
printf ("% d' % d'",-3,3); //' -3' '3'
```

→ + - задает обязательный вывод знака, как для отрицательных, так и для положительных чисел. Пример:

```
printf ("%+d' %+d'",-3,3); //' -3' '+3'
```

→ h - предназначен для вывода значения переменной, имеющей тип short int. Пример:

```
short int x=32767;
printf ("%hd",x); //32767
```

→ l (буква "эль") — предназначен для вывода значения переменной, имеющей тип long int. Пример:

```
long int x=2147483647;.
printf ("%ld",x); //2147483647
```

→ L - предназначен для вывода значения переменной, имеющей тип long double. Пример:

```
long double x=8e+245;
printf ("%Le",x); //8.000000e+245
```

Следует учитывать, что функции `putchar ()`, `puts ()` и `printf ()` применяются в языке C. Хотя их можно использовать и в языке C++, тем не менее для вывода данных стоит отдать предпочтение объекту `cout`.

Ввод данных в языке C++

Для ввода данных в языке C++ предназначен объект `cin`, объявленный в файле `iostream`. Объект `cin` позволяет ввести данные любого встроенного типа, например, число, символ или строку. В этом разделе мы рассмотрим лишь основные возможности этого объекта, так как для понимания всех возможностей требуется знание принципов объектно-ориентированного программирования (ООП).

Прежде чем использовать объект `cin` необходимо подключить файл `iostream` с помощью директивы `include`:

```
#include <iostream>
```

В качестве примера использования объекта `cin` произведем суммирование двух целых чисел, введенных пользователем.

Суммирование двух введенных чисел

```
#include <iostream>
using namespace std;
int main() {
    int x=0, y=0;
    cout<<"x=";
    cin>>x;
    cout<<"y=";
    cin>>y;
    cout<<"Summa = "<<x+y<<endl;
    return 0; }
```

В первой строке производится подключение файла `iostream`. Далее инструкция подключения пространства имен `std`. Затем внутри функции `main()` объявляются две локальные переменные: `x` и `y`. Ключевое слово `int` в начале строки означает, что объявляются целочисленные переменные. При объявлении переменным сразу присваивается начальное значение (0) с помощью оператора `=`. Если значение не присвоить, то переменная будет иметь произвольное значение, так называемый "мусор". Как видно из примера, на одной строке можно объявить сразу несколько переменных, разделив их запятыми.

В следующей строке с помощью объекта `cout` выводится подсказка для пользователя ("`x =` "). Благодаря этой подсказке пользователь будет знать, что от него требуется. После ввода числа и нажатия клавиши `<Enter>` значение будет присвоено переменной `x`. Ввод значения производится с помощью объекта `cin`. После названия объекта указывается оператор `>>`, который как бы указывает направление ввода. При разработке собственных объектов можно перегрузить этот оператор и тем самым управлять вводом. После оператора `>>` передается название переменной, в которой будут сохранены введенные данные.

Далее производится вывод подсказки и получение второго числа, которое сохраняется в переменной `y`. В следующей строке производится сложение двух

чисел и вывод результата. Процесс ввода двух чисел и получения суммы выглядит так:

```
x=10
y=20
Summa=30
```

При вводе данных производится попытка преобразования к типу переменной. В нашем примере строка "10" будет преобразована в число 10, а строка "20" в число 20. Однако, вместо числа пользователь может ввести буквы. В этом случае преобразование закончится неудачей, переменной не будет присвоено значение, а введенное значение останется в буфере и будет автоматически считано следующей операцией ввода.

Предотвращение закрытия окна консоли

При запуске программы с помощью комбинации клавиш **<Ctrl>+<F5>** в конце автоматически вставляется строка *"Для продолжения нажмите любую клавишу ..."*. Нажатие любой клавиши приводит к закрытию окна консоли. Однако конечный пользователь таким образом запускать программу не будет. Если запуск производится из командной строки, то пользователь сможет увидеть результат, но если запуск производится с помощью двойного щелчка на значке файла, то окно консоли откроется, а затем сразу закроется. Чтобы окно не закрывалось необходимо вставить инструкцию, ожидающую нажатие клавиши. Сделать это можно несколькими способами.

Первый способ заключается в использовании функции `_getch()`. Возвращаемое функцией значение можно проигнорировать. Прежде чем использовать функцию необходимо подключить файл `conio.h`.

Использование функции `getch()`

```
#include <iostream>
#include <conio.h> // Для _getch()
using namespace std;
int main() {
// Инструкции
cout <<"Press <Enter> ... ";
_getch();
return 0; }
```

Второй способ заключается в использовании функции `system()`. Эта функция позволяет передать команду операционной системе. Для вывода строки "Для продолжения нажмите любую клавишу" и ожидания нажатия клавиши предназначена команда `pause`. Прежде чем использовать функцию необходимо подключить файл `cstdlib` (или `stdlib.h`). Прототип функции:

Использование функции `system()`

```
#include <iostream>
#include <cstdlib> // Для system()
using namespace std;
int main() {
// Инструкции
system("pause"); }
```

```
return 0; }
```

Настройка отображения русских букв в консоли

До сих пор во всех операциях вывода мы использовали латинские буквы. Для этого были основания. Например, попытаемся вывести текст на русском языке:

```
cout << "Добро пожаловать в C++!";
```

В результате в окне консоли отобразится нечитаемый текст:

```
—юсЁю яюцрюютрЕН! т С++!
```

Причина искажения русских букв заключается в том, что по умолчанию в окне консоли используется кодировка `cp866`, а в программе мы выводим текст в кодировке Windows -1251. Коды русских букв в этих кодировках отличаются, поэтому происходит искажение. При использовании командной строки пользователь может сменить кодировку вручную, выполнив команду: `chcp 1251`

Однако этого не достаточно. Кроме смены кодировки необходимо изменить название шрифта, так как по умолчанию используются точечные шрифты, которые не поддерживают кодировку Windows-1251. Для нормального отображения русских букв следует в свойствах окна выбрать шрифт `Lucida Console`.

Все эти действия вы можете произвести на своем компьютере, однако пользователи не знают в какой кодировке выводятся данные из вашей программы. Следовательно необходимо предусмотреть вывод русских букв в текущей кодировке консоли. Изменить кодировку консоли можно с помощью функции `system()`, однако сменить шрифт довольно проблематично. Проще изменить кодировку текста с Windows-1251 на `cp866`. При использовании Visual C++ преобразование кодировки производится автоматически после настройки *локали* (локальных настроек компьютера). Настроить локаль позволяет функция `set locale()`. Для использования этой функции необходимо подключить файл `locale` (или `locale.h`). Прототип функции:

```
#include <locale>
char *setlocale(int Category, const char *Locale);
```

В первом параметре указывается категория в виде числа от 0 до 5. Вместо чисел можно использовать макроопределения `LC_ALL`, `LC_COLLATE`, `LC_CTYPE`, `LC_MONETARY`, `LC_NUMERIC` и `LC_TIME`. При компиляции макроопределения *будут* заменены соответствующими числами. Во втором параметре задается название локаль и в виде строки (например, `"Russian_Russia. 1251"` или `"1251"`) более короткий вариант: `setlocal(0, "")`. Вместо названия можно указать пустую строку. В этом случае будет использоваться локаль, настроенная в системе. Пример настройки локали и вывода русских букв

Настройка локали

```
#include <iostream>
#include <locale>
#include <conio.h> // Для getch()
using namespace std;
int main ()      {
setlocale(0, "");
```

```
cout<<"Добро пожаловать в C++!"<<endl;  
cout<<"Нажмите <Enter> для закрытия окна...";  
return 0; }
```

При использовании Visual C++ эта программа корректно выводит русские буквы вне зависимости от того, используется в консоли кодировка Windows-1251 или cp866.

Задание 4. Создайте в решении «Лабораторная 1» проект «Задание 4». Напишите код в котором:

вводятся значения в переменные целого типа x и y ;
вычисляется сумма, разность, произведение и отношение этих переменных;
вывести результаты вычислений в виде

Сумма $x+y =$

Разность $x - y =$

Произведение $x*y=$

Отношение $x/y=$

отладить и запустить проект.

Контрольные вопросы

1. Что такое Решение и как оно связано с проектом.
2. Что такое пустой проект.
3. Чем отличается создание пустого проекта от консольного приложения.
4. Какого назначения файла `stdafx.h`.
5. Где отображается результат построения проекта.
6. Что делать, если в меню нет пункта «Запуск без отладки».
7. Если решение содержит несколько проектов, Что надо сделать, чтобы запустить другой проект.
8. Для чего надо подключать директиву `iostream`.
9. Чем отличается комментарий `//` от комментария `/*, */`.
10. Какой оператор выводит данные на экран.
11. Для чего предназначен `endl`.
12. Что надо сделать, чтобы задержать экран консоли при выводе результатов.
13. Какой оператор предназначен для ввода данных.
14. С какой целью в начале кода пишется строка `using namespace std`; и что надо делать, если эта строка отсутствует.
15. Что надо делать, чтобы на консольном экране отображались русские буквы.

Тема № 1 Линейный вычислительный процесс

Этапы разработки программы

Разработка любой программы, может быть разбита на ряд этапов.

1. Определение входных и выходных данных, требований к программе.

На первом этапе определяются входные и выходные данные программы.

2. Разработка алгоритма.

На этом шаге производится определение последовательности действий, ведущих к решению задачи и запись их в виде блок-схемы.

3. Кодирование (программирование).

Третий этап - это перевод алгоритма на язык программирования и создание *исходного текста программы*. Программа на любом языке состоит из *операторов* - так называются отдельные действия, разрешенные в языке.

4. Компиляция и отладка.

Исходный текст не будет непосредственно исполняться компьютером - для работы программы ее требуется *откомпилировать*, т. е., перевести в машинный код. Программа, которую удалось откомпилировать, не обязательно работает правильно. Она может содержать ошибки, для выявления которых предназначен этап *отладки* - поиска ошибок в программе.

Возможны программные ошибки трех видов:

- *синтаксические* (ошибки в правилах языка);
- *алгоритмические* (ошибки в логике программы);
- *ошибки времени исполнения*, возникающие в процессе работы запущенной программы.

Компилятор способен найти только синтаксические ошибки, для выявления же алгоритмических ошибок служит этап *тестирования* программы. Ошибки времени исполнения возникают как результат некорректных действий пользователя, недопустимых операций над данными (например, попытки извлечь квадратный корень из отрицательного числа, поделить на ноль) или ошибок программного и аппаратного обеспечения ЭВМ

5. Тестирование.

Тестированием называют проверку правильности работы программы на наборах "пробных" (*тестовых*) данных с заранее известным результатом. Конечно же, тестирование всей программы сразу возможно лишь для несложных учебных задач. Реальные программы, как правило, тестируются "по частям" - отдельными функциями и модулями.

Блок-схема

Блок-схема – это последовательность блоков, предписывающих выполнение определенных операций, и связей между этими блоками. Внутри блоков указывается информация об операциях, подлежащих выполнению.

Блоки соединяются линиями потоков. При соединении блоков следует использовать только вертикальные и горизонтальные линии потоков. Горизонтальные потоки, имеющие направление справа налево, и вертикальные

потоки, имеющие направление снизу вверх, должны быть обязательно помечены стрелками.

Любой алгоритм начинается с блока «Начало», а заканчивается блоком «Конец», которые изображаются в виде овалов



Каждому блоку можно поставить в соответствие оператор языка. Исключение составляет только блоки «Начало» и «Конец».

На блок-схеме оператор присваивания соответствует блоку процесс:



На блок-схеме оператор ввода изображается в виде параллелограмма:



На блок-схеме оператор вывода также изображается в виде параллелограмма:



Лабораторная № 2 Линейный вычислительный процесс

Линейным принято называть вычислительный процесс, в котором операции выполняются последовательно, в порядке их записи. Каждая операция является самостоятельной, независимой от каких-либо условий. На схеме блоки, отображающие эти операции, располагаются в линейной последовательности. Основу программы ЛВП составляют операторы присваивания, ввода и вывода данных.

Линейные вычислительные процессы имеют место, например, при вычислении арифметических выражений, когда имеются конкретные числовые данные и над ними выполняются соответствующие условию задачи действия.

В качестве операндов могут выступать стандартные арифметические функции. Для использования этих функций требуется дополнительно подключать библиотеки. В таблице приведены стандартные математические функции и библиотеки.

Имя	Описание	Тип аргумента	Библиотека
M_PI	Число π		cmath
abs(x)	Абсолютное значение аргумента типа int	int	cstdlib
fabs(x)	Абсолютное значение аргумента типа double	double	cmath
pow(a,b)	Возведение a в степень b	double	cmath
cos(X)	Косинус x (x в радианах)	double	cmath
sin(X)	Синус x (x в радианах)	double	cmath
acos(x)	Арккосинус от x. Аргумент функции должен лежать в пределах от -1 до 1	double	cmath

asin(x)	Арксинус от x. Аргумент функции должен лежать в пределах от —1 до 1	double	cmath
tan(x)	Тангенс от x (x в радианах)	double	cmath
atan(x)	Арктангенс от x	double	cmath
exp(x)	e ^x - экспонента	double	cmath
log(x)	Натуральный логарифм от x	double	cmath
log10(x)	Десятичный логарифм x	double	cmath
sqrt(x)	Квадратный корень X	double	cmath
srand(time(NULL))	Инициализирует генератор случайных чисел некоторой случайной величиной		time.h
rand()	генерирует последовательность псевдослучайных чисел. Всякий раз при ее вызове возвращается число в интервале от 0 до RAND_MAX.	int	cstdlib

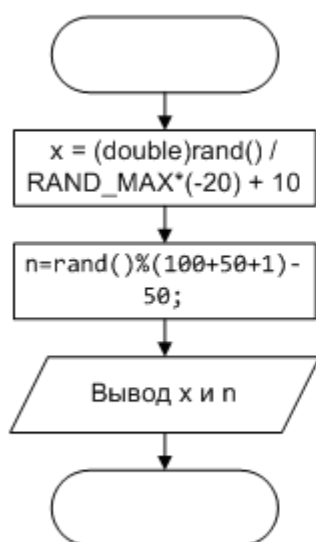
ПРИМЕРЫ

Пример 1.

Сгенерировать вещественное число в интервале от -10 до 10 и целое число в интервале от -50 до 100

Исходные данные: интервал для вещественных чисел от -10 до 10, интервал для целых чисел от - 50 до 100.

Результат: x – вещественное число, n – целое число.



```

#include <iostream>
#include <time.h>
#include <stdlib.h>
#include <conio.h>
using namespace std;
int main() {
    int n; double x;
    srand((unsigned)time(NULL));
    x = (double)rand() / RAND_MAX * (-20) + 10 ;

    n=rand()%(100+50+1)-50;
    cout<<"x="<<x<<" n="<<n<<endl;
    _getch();
    return 0;
}
  
```

Пример 2.

Найти z :

$$z = |a - 10| \cdot \log_2(4 - b) + 2(b - 10) + \sqrt[5]{a^4}, \text{ где } a = b^{-0.25} \cdot \arccos 0.6 - (d\sqrt{d})^{\frac{b}{3}} \cdot \operatorname{tg} b,$$
$$b = \frac{d(1 - \cos 2\alpha + \sin 2\alpha)}{1 + \cos 2\alpha + \sin 2\alpha} + d, \quad d = \frac{\sin \alpha}{1 + \frac{\cos k + 1}{\operatorname{tg}^2 15 \cdot k}}.$$

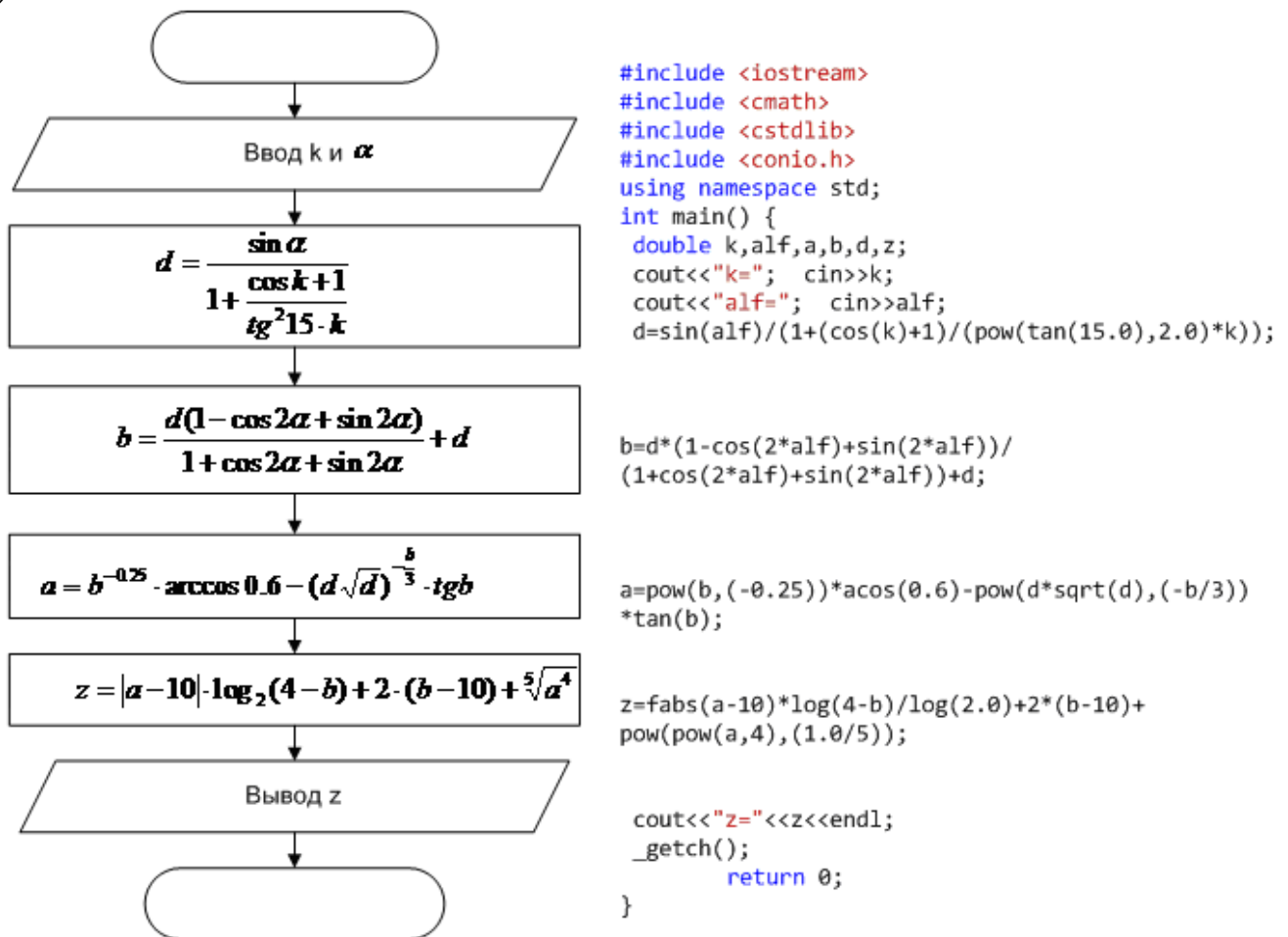
Для произвольных k и α .

Исходные данные: k – вещественный тип, α будет обозначаться alf – вещественный тип.

Результат: z – вещественный тип.

При вычислении надо учитывать, что при вычислении выражения, все переменные, участвующие в этом выражении должны быть известны. Это диктует последовательность вычислений.

Тестовый пример: при $k=1$ и $\alpha=1$ $z=-0,48473$. (Проверка результатов выполнялась в Excel)



Контрольные вопросы

1. Для чего надо подключать библиотеку `conio.h`?
2. Какой тип имеет результат выражения $10/2$?
3. Какие ограничения существуют при возведении числа в произвольную степень?

4. Выражение $\frac{1}{2a}$ должно быть записано как $1/(2*a)$. Почему знаменатель заключен в круглые скобки?
5. Почему для ввода данных требуется два оператора: `cout` и `cin`?
6. Можно ли в Примере 2 изменить порядок вычислений. Например, а вычислить раньше b?
7. Почему в программе некоторые числа записываются с точкой ?

Индивидуальные задания

Создать программу вычисления выражений и выполнить проверку средствами Excel.

1. Найти t :

$$t = c - ae^{-2} + \operatorname{arctg}k, \text{ где}$$

$$a = 3^{-c} \frac{\sin \pi \cdot k}{\pi \cdot k}$$

$$k + \sqrt[3]{\cos^2 x + |\sin c^2|}$$

$$c = \sqrt{\lg \frac{1}{|\cos x|} + y^2}$$

для произвольных x и y .

2. Найти U :

$$U = x + y + e^{-wvf}$$

где

$$w = f + v + \ln|x \cdot y \cdot z|$$

$$v = \sqrt[3]{\sin^2(\cos f) + x}$$

$$f = -e^{x\sqrt{|y|}} + z$$

для произвольных x , y и z .

3. Найти m :

$$m = \sqrt[3]{|n \cdot x \cdot a|}$$

где

$$n = b \cdot \sin x + t$$

$$t = \ln b - e^a$$

$$b = \operatorname{tg} \sqrt{\frac{|x|}{|a|}} + x$$

для произвольных a и x .

4. Найти i

$$i = \ln \sqrt[k]{|m + n|}$$

где

$$k = m^{m \cdot \sin b}$$

$$b = \sqrt[3]{\operatorname{tg}^2(m \cdot d)}$$

$$n = e^{d-m}$$

для произвольных d и m .

5. Найти w

$$w = \sqrt{|a^m \cdot b^n|}$$

где

$$a = n \cdot \sin^4 m$$

$$n = \sqrt[m]{b \cdot k}$$

$$b = e^k$$

для произвольных m и k .

6. Найти p

$$p = g + \frac{f}{s} - e^{\sqrt{|s|}}$$

где

$$s = f + \sin g^t$$

$$g = t^3 + \sqrt[5]{|f \cdot t - 5|}$$

$$t = f + \cos^2 k$$

для произвольных f и k .

7. Найти a

$$a = e^b e^c e^d$$

где

$$b = \cos^2 c + \sin^2 d$$

$$c = d \sqrt{xy} + e^d$$

$$d = \frac{(e^x)^{\frac{1}{y}} - \lg|xy|}{x+y}$$

для произвольных x и y .

8. Найти q

где

$$q = \operatorname{tg} p^t$$

$$p = \sqrt[5]{acb}$$

$$t = \sin^a b^2$$

$$c = -e^{\sqrt{a+b+5}}$$

для произвольных a и b

9. Найти x

$$x = \frac{a+b-e^{2 \cdot c}}{c + \frac{d}{f+a}}$$

где

$$d = c^{\sin a + \cos b}$$

$$f = \frac{-b + \sqrt{b^2 - 4a \cdot c}}{a \cdot c}$$

$$c = \sqrt[3]{tga^b}$$

для произвольных a и b

10. Найти z

$$z = \operatorname{arctgy} + \ln|x-1|$$

где

$$x = \frac{b \cdot \sqrt{a^2 - 1}}{a^2 + b^2} - 2.47 \cdot a$$

$$y = \sin^2(\sqrt[3]{x}) - e^{0.4 \cdot (1+a)}$$

$$a = 5 \cdot \cos c^3$$

для произвольных c и b

11. Найти v

$$v = \sqrt{2 \cdot tgu - \frac{1}{u}} + 2 \cdot \sqrt{m^2 + n^2}$$

где

$$u = \frac{2(1 - \cos 2a + \sin 2a)}{1 + \cos 2b + \sin 2b}$$

$$m = \ln|\operatorname{arctga} + \operatorname{arctgb}| + a^b$$

$$n = \lg \sqrt{2a} + 3 \lg \frac{a \cdot b}{a+b}$$

для произвольных a и b .

12. Найти f

$$f = 4^{3 \ln|p+t|} + \sqrt{2ptga}$$

где

$$p = \frac{\cos^3 x}{\sin^2(1 + tg^2 a)}$$

$$t = e^{\sqrt{x+a}} + a^{2 \cdot b}$$

$$x = \frac{a \cdot \sin a}{b \cdot \cos b}$$

для произвольных a и b .

Тема 2 Разветвляющийся вычислительный процесс

Разветвляющимся называется алгоритм, в котором действие выполняется по одной из возможных вервей решения задачи, в зависимости от выполнения условий. В отличие от линейных алгоритмов, в которых команды выполняются последовательно одна за другой, в разветвляющиеся алгоритмы входит условие, в зависимости от выполнения или невыполнения которого выполняется та или иная последовательность действий.

Разветвляющиеся алгоритмы можно реализовать с помощью двух операторов: условного оператора и оператора выбора.

Лабораторная работа № 3

Условный оператор

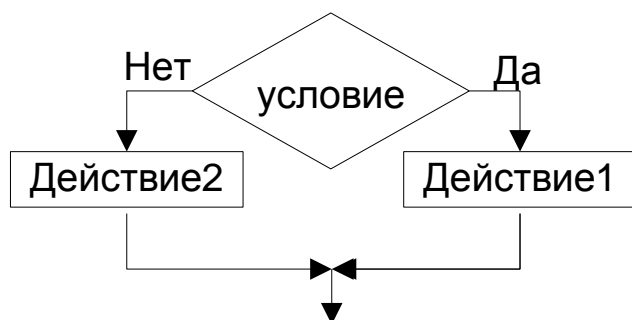
Теория

Назначение условного оператора – обеспечить ветвление алгоритма в зависимости от выполнения некоторого условия.

Блок проверки условия (блок принятия решения): изображается ромбом с двумя выходами «Да» и «Нет».

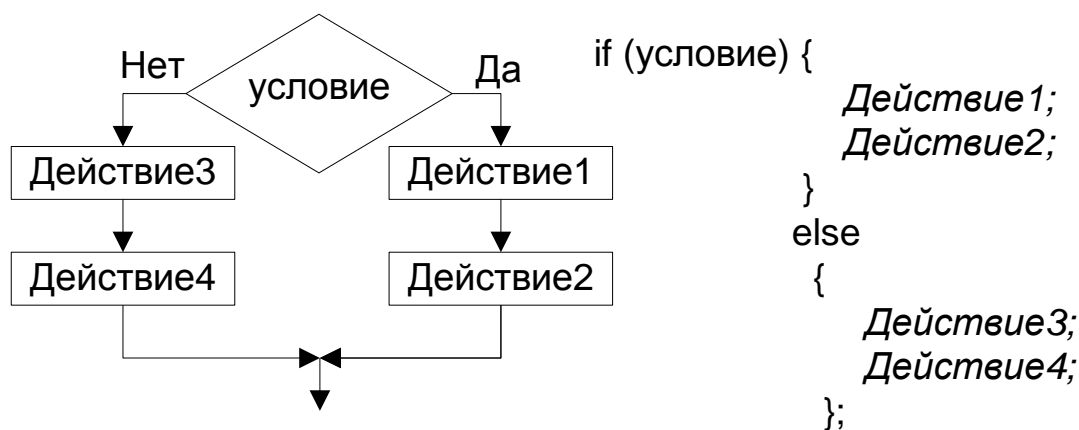


В языке C++ условный оператор имеет следующий вид: Если условие, записанное после служебного слова `.if`, верно, то выполняется действие 1, иначе – действие 2.

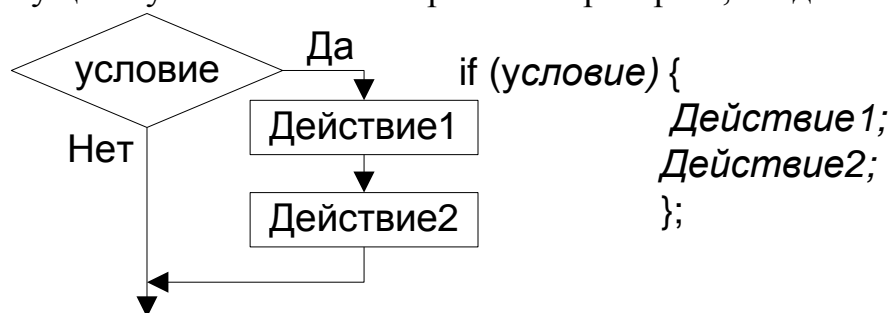


```
if (условие)
    Действие1;
else
    Действие2;
```

Если на ветках «Да» или «Нет» надо выполнять несколько действий, эти операторы надо заключать в операторные скобки { и }.



Существует не полный вариант оператора if, когда ветка «Нет» пустая.



Условие представляет собой логическое выражение. **Логическое выражение** — это любое выражение, которое может быть либо истинным, либо ложным. Логическое выражение состоит из одного или нескольких отношений. Операция отношения выполняет сравнение двух операндов. Результат операции отношения значение булевого типа true или false.

Основные операции отношения:

- == равно;
- <> не равно;
- >больше;
- < меньше; >=
- >=больше или равно; <=
- <=меньше или равно.

Если условие имеет более сложный вид, т. е. содержит несколько отношений, эти условия должны содержать логические операции, которые и объединяют отношения. В C++ имеются следующие логические операции:

a	b	!a	a && b	a b
true	true	false	true	true
true	false	false	false	true
false	true	true	false	true
false	false	true	false	false

Пример использования в инструкции if-else:

```
if ((score>0) && (score<10))  
cout<<"счет в пределах от 0 до 10.\n"; else  
cout<<"счет выходит за рамки от 0 до 10.\n";
```

Если значение переменной score больше 0 и меньше 10, то выполняется первая cout-инструкция, в противном случае выполняется вторая cout-инструкция.

Пример использования в инструкции if-else.

```
if ((x==1) || (x==y))  
cout<<"x равно 1 или y\n"; else  
cout<<"x не равно ни 1, ни y\n";
```

Если значение переменной x равно 1 или значению y, то выполняется первая cout-инструкция; в противном случае выполняется вторая cout-инструкция.

Если в логическом выражении круглые скобки опущены, компьютер группирует элементы в соответствии с **приоритетом операций**. Приоритеты некоторых операций в языке C++ приведены в таблице. Говорят, что оператор имеет **более высокий приоритет**, если по умолчанию он выполняется до другого оператора. Бинарные операторы с одинаковым приоритетом выполняются слева направо; унарные операторы с одинаковым приоритетом — справа налево.

Унарные операторы	+, -, ++, --, !	Строки таблицы расположены в порядке понижения приоритета
Бинарные арифметические операторы	*, /, %	
Бинарные арифметические операторы	+, -	
Логические операторы: <, >, <=, >=	<, >, <=, >=	
Логические операторы: ==, !=	==, !=	
Логический оператор	& &	
Логический оператор		

Иногда в языке C++ в качестве логических значений используются целые числа. В частности, C++ преобразует целое значение 1 в true, а 0 — в false. На самом деле ситуация несколько сложнее, нежели просто использование 1 и 0 в качестве значений true и false соответственно. На самом деле компилятор трактует всякое отличное от нуля значение как true, а 0 — как false. Такое преобразование не создает каких-либо проблем, если не допускать ошибок при составлении логических выражений. До тех пор пока вы не допускаете ошибок в логических выражениях, вам не требуется знать, как логические и целые значения преобразуются друг в друга. Однако эта информация может понадобиться при отладке программ. Из-за этой ситуации компилятор не всегда сообщает об ошибке. Рассмотрим примеры таких ошибок.

Множественные неравенства

Не используйте в программе строк с множественными неравенствами, подобными следующим:

```
if (x<z<y) //Так делать нельзя!  
cout<<"z находится в интервале между x и y.";
```

При использовании приведенного фрагмента программа, скорее всего, откомпилируется и запустится, но результат ее работы, несомненно, будет

неправильным. Проблема не ограничивается только одним знаком < и возникает при использовании любого оператора сравнения. Правильная проверка на принадлежность переменной диапазону значений использует оператор &&:

```
if ((x<z) && (z<y)) // Правильная проверка
cout<<"z находится в интервале от x до y.";
```

Использование = вместо ==

К сожалению, в C++ можно написать множество инструкций, которые, с одной стороны, будут скомпилированы без ошибок, а с другой — будут выполнять не те действия, которые от них ожидают. Иногда можно просто не заметить неверную, запись, и это может привести к серьезным проблемам. На поиск ошибки программист может потратить немало времени, пока не поймет, что именно в программе сделано не так. Одной из распространенных ошибок является использование символа = вместо ==. Рассмотрим, например, инструкцию if-else, которая начинается следующим образом:

```
if (x = 12)
    Да_инструкция; else
    Нет_инструкция
```

Вероятно, программист хотел проверить, равно ли значение x числу 12, и на самом деле он должен был написать не =, а ==. Вам кажется, что компилятор должен сообщить об ошибке— ведь выражение $x = 12$ не является условием, которое может или выполняться, или нет. Это инструкция присвоения, поэтому компилятор должен сообщить об ошибке! К сожалению, на самом деле это не так. В C++ $x = 12$ является выражением, которое имеет значение, и в этом смысле оно не отличается от таких выражений, как $x + 12$ или $2 + 3$. Инструкция присвоения принимает значение выражения, которое находится слева от знака равенства, так что значение инструкции $x = 12$ равно 12. При обсуждении совместимости типов отмечалось, что ненулевые величины типа int могут быть преобразованы в логическое значение true, а потому в приведенном выше примере всегда будет выполняться первая инструкция (*Да_инструкция*).

Такую ошибку очень трудно обнаружить, потому что она *не похожа на ошибку!* Но если поместить в левую часть соотношения число 12, как показано в следующем примере, то в случае использования одного знака равенства вместо двух компилятор тут же обнаружит ошибку.

```
if(12==x)
    Да_инструкция; else
    Нет_инструкция;
```

Помните о том, что использование одного знака равенства = вместо двух == является широко распространенной ошибкой, причем компилятор в таких случаях не выдает сообщений об ошибке, а обнаружить ее самому очень трудно. В C++ многие выполняемые инструкции могут служить выражениями других видов, в частности логическими выражениями в инструкции if-else. Так, если инструкцию присвоения поместить там, где должно быть логическое выражение, то она будет интерпретирована именно так, как ожидается, т.е. как логическое выражение. Конечно, формально результат "проверки" такого условия будет абсолютно

корректным, но совершенно не тем, который ожидает допустивший опечатку программист.

При использовании оператора if следует помнить:

- Если на какой-нибудь ветке надо выполнить несколько действий, эти действия надо заключать в операторные скобки { };
- Если на ветке «Да» используются операторные скобки, перед служебным словом else нельзя ставить точку с запятой;
- Если условный оператор содержит другой условный оператор, то ветка else относится к ближайшему условию. Поэтому, если при вложении операторов if какой-то оператор имеет не полную форму, рекомендуется использовать операторные скобки.
- Если выражение, которое записывается в условии длинное, имеет смысл записать условие как отдельный оператор присваивания. В этом случае следует дополнительно объявить переменную булевского типа и записать туда это выражение.

Примеры

Пример 1.

Вычислить A :

$$a = \frac{b+c}{2 \cdot c}, \text{ где } c = 4 \cdot b - 20$$

Для произвольного b .

Исходные данные: b – вещественного типа.

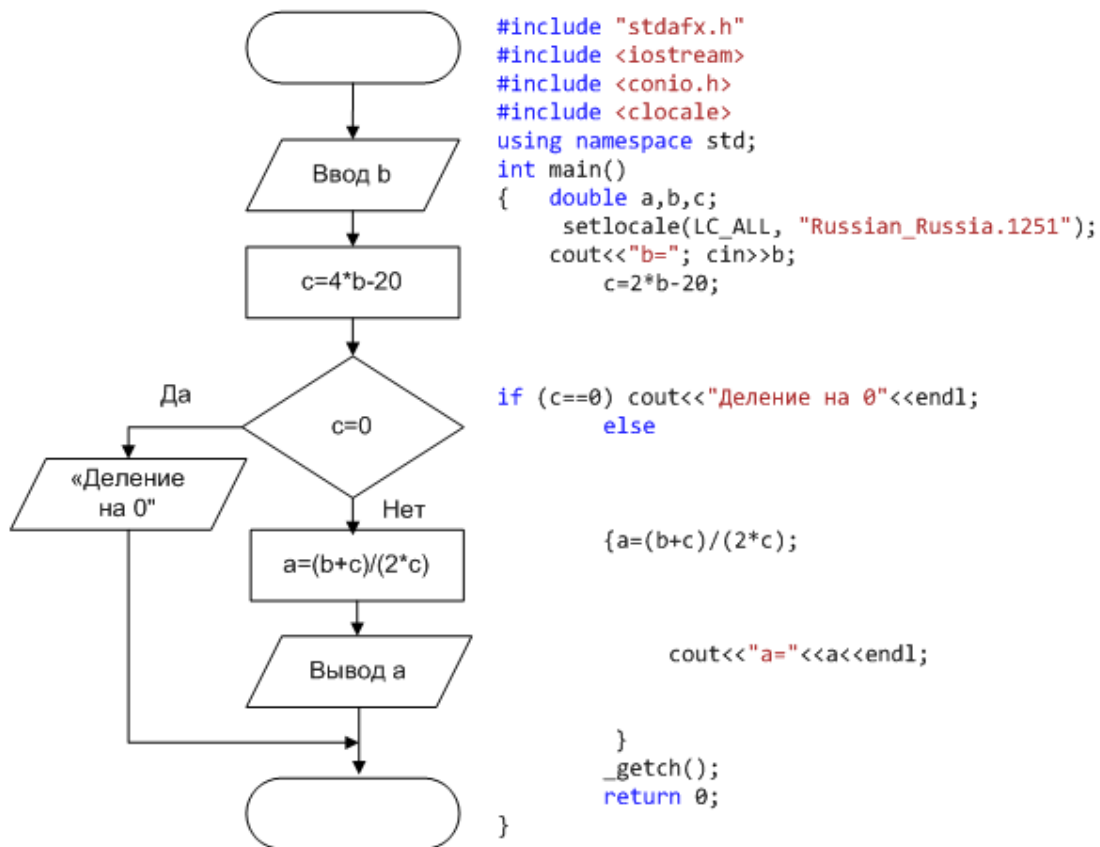
Результат: b – вещественного типа.

Сначала надо вычислить b , которая является частью выражения для вычисления a . При вычислении a может возникнуть ошибка – деление на ноль. Следовательно, перед делением надо выполнить проверку. Если знаменатель равен нулю - прекратить вычисление и выйти из программы.

Тестовый пример:

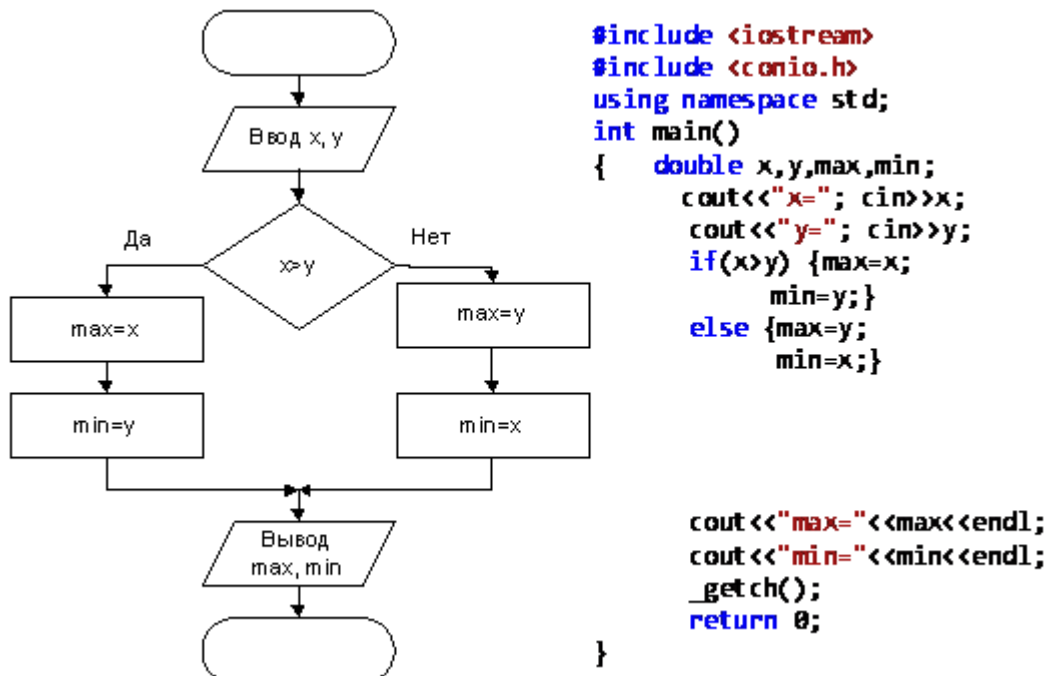
a. при $b=1$ $a=0.468$.

b. при $b=5$, выводится сообщение «Деление на ноль»



Пример 2.

Даны два произвольных числа. Определить максимальное и минимальное из этих чисел.



Исходные данные: x и y – вещественного типа.

Результат: max и min – вещественного типа.

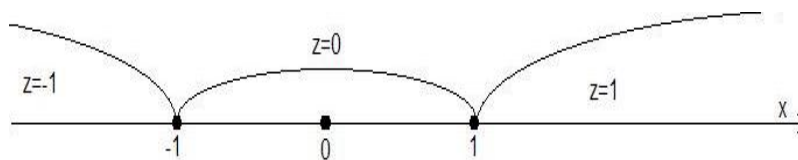
Тестовый пример:

при x=5, y=10, max=10, min=5.

Пример 3.

Вычислить z:

$$z = \begin{cases} -1, & \text{если } x < -1 \\ 0, & \text{если } -1 \geq x \geq 1 \\ 1, & \text{если } x > 1 \end{cases}$$



Для произвольного x.

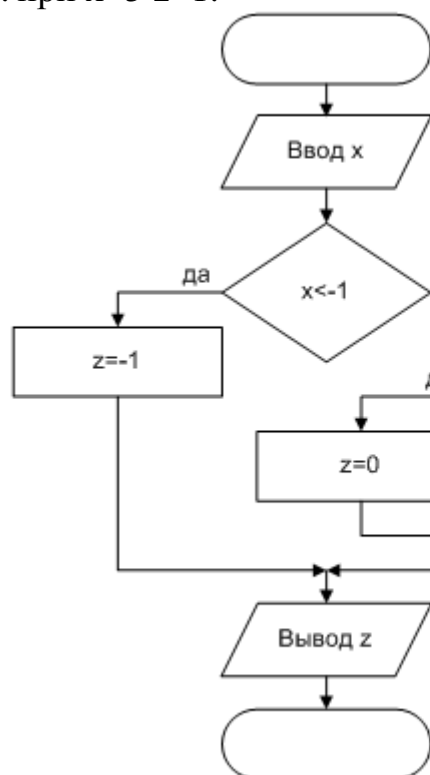
Исходные данные: x – вещественного типа.

Результат: z – вещественного типа.

Необходимо определить в какую часть числовой оси попадает x

Тестовый пример:

- a. при x=-10 z=-1;
- b. при x=1 z=0;
- c. при x=5 z=1.



```
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{ double x;
  int z;
  cout<<"x="; cin>>x;
  if(x<-1) z=-1;
  else
    if(x<=1) z=0;
    else z=1;
  cout<<"z="<<z<<endl;
  _getch();
  return 0;
}
```

Пример 4.

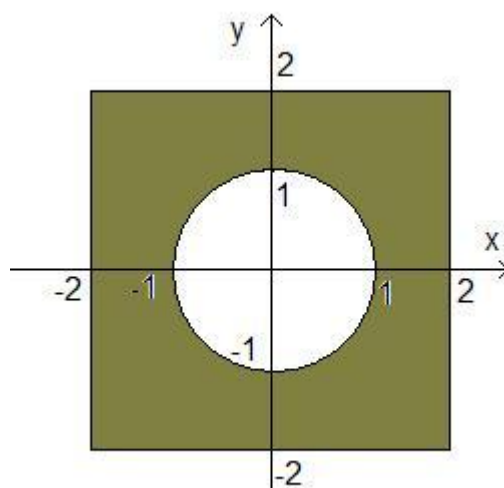
Определить попадает ли точка с координатами (a,b) в заштрихованную область представленную на рисунке.

Исходные данные: a и b координаты точки–вещественного типа, радиус окружности – 1, половина стороны квадрата – 2.

Результатом будет сообщение попадает ли точка в данную область.

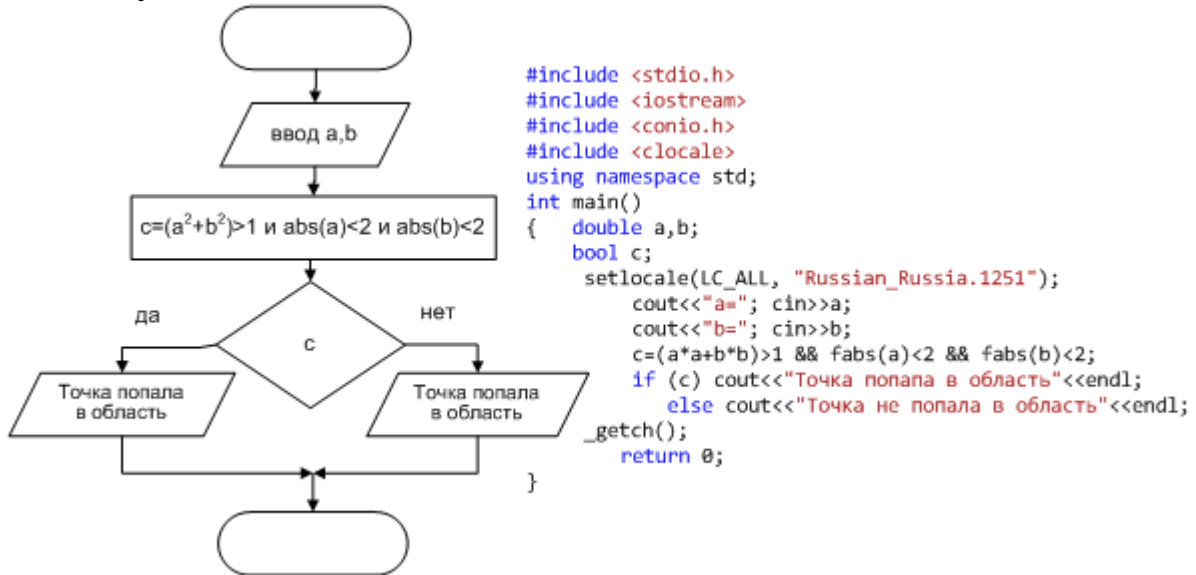
Тестовый пример:

- a. при x=0.5, y=0.5, точка не попала в область;
- b. при x=1.2, y=-1.5, точка попала в область;



с. при $x=3, y=-4$, точка не попала в область.

Формула окружности $x^2+y^2=r^2$, где r – радиус окружности. Для точек вне окружности выполняется неравенство: $x^2+y^2>r^2$, если центр окружности имеет координаты $(0,0)$ В квадрат попадают точки, для которых выполняется условие $|x|<h/2$ и $|y|<h/2$, где h – сторона квадрата и центр квадрата имеет координаты $(0,0)$. Все эти условия должны выполняться одновременно, чтобы точка попала в заштрихованную область.



Пример 5.

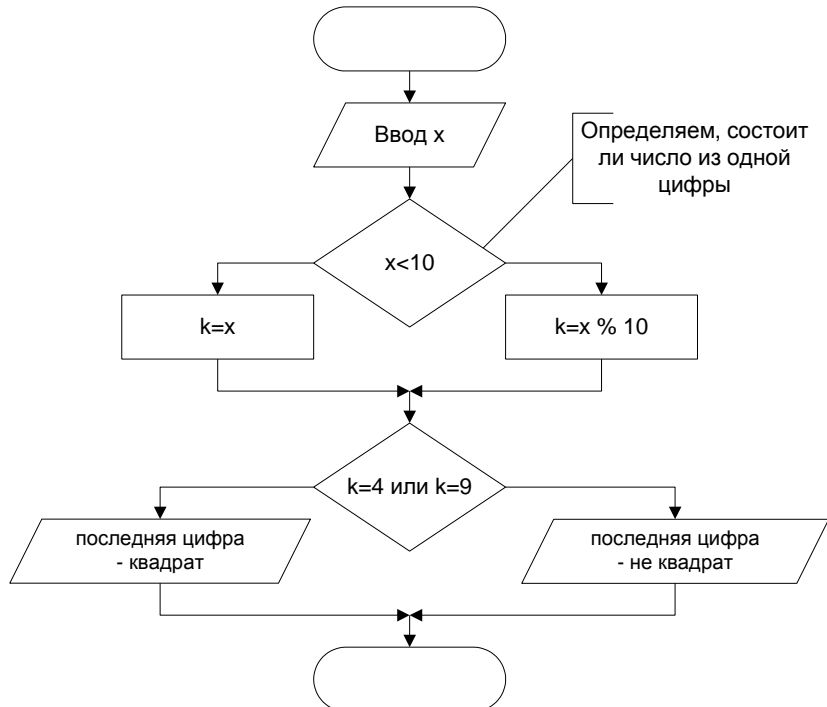
Дано целое число. Определить, является ли последняя цифра данного числа квадратом другого числа.

Исходные данные: Заданное число x – целого типа.

Результатом будет сообщение, является ли последняя цифра квадратом другого числа.

Среди чисел меньших 10 имеется только два числа, которые являются квадратами другого

числа – это 4 и 9. Если заданное число больше 9, то надо выделить последнюю цифру. Обозначим эту цифру k . k можно получить как остаток от деления данного числа на 10. Если заданное число меньше 10, то k и есть данное число.



Тестовый пример:

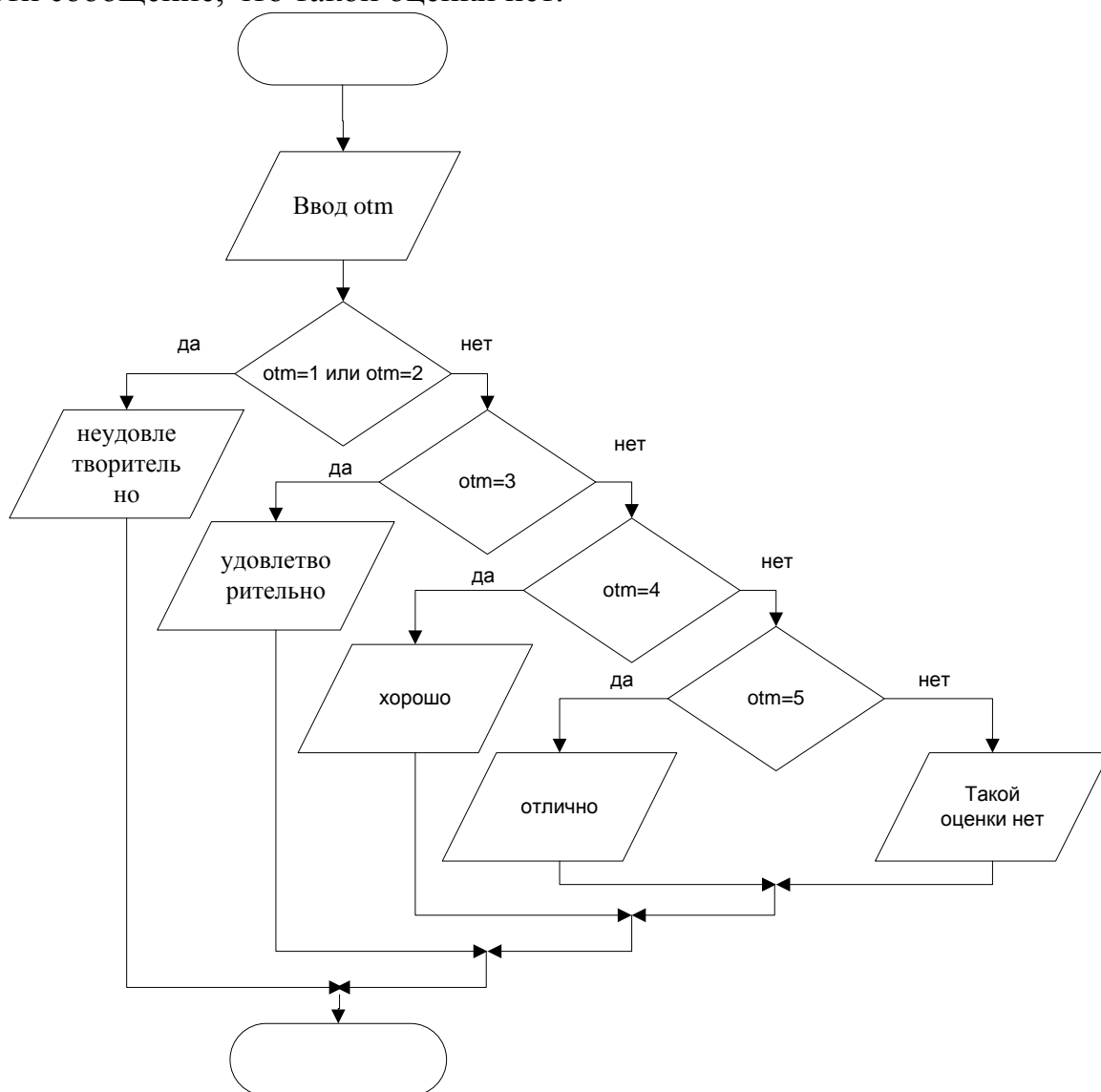
- a. при $x=8$, последняя цифра не квадрат;
- b. при $x=9$, последняя цифра квадрат;
- c. при $x=41$, последняя цифра не квадрат;
- d. при $x=129$, последняя цифра квадрат.

Пример 6.

Ввести оценку в виде числа и вывести в виде текста.

Исходные данные: Оценка otm – целое число.

Результат: оценка в виде сообщения: *неудовлетворительно*, *удовлетворительно*, и т.д. Если введенная оценка не попадает в интервал от 1 до 5, вывести сообщение, что такой оценки нет.



Задание 1 Ввести и отдалить программу для примера 5.

Контрольные вопросы

1. В какой ситуации в условном операторе используются $\{ \}$.
2. Почему при записи $(x < z < y)$ не будет сообщения об ошибке.

3. Расставьте в порядке понижения приоритета следующие операции: +, &&, >=, ||, *, <, %.
4. Почему в примере 3 нет проверки $x > 1$.
5. Изобразите блок-схему примера 5 без использования сложного условия.
6. По какой ветке будет выполняться алгоритм примера 2, если x и y равны.

Индивидуальные задания

Требуется написать программу, которая вводит с клавиатуры координаты точки на плоскости (x, y – вещественные числа) и определяет принадлежность точки заштрихованной области, включая ее границы.

При составлении таблицы контрольных примеров отдельно пропишите проверку попадания точки в области, обозначенные на рисунке заглавными латинскими буквами.

№ вар.	Описание задания
1	
2	
3	

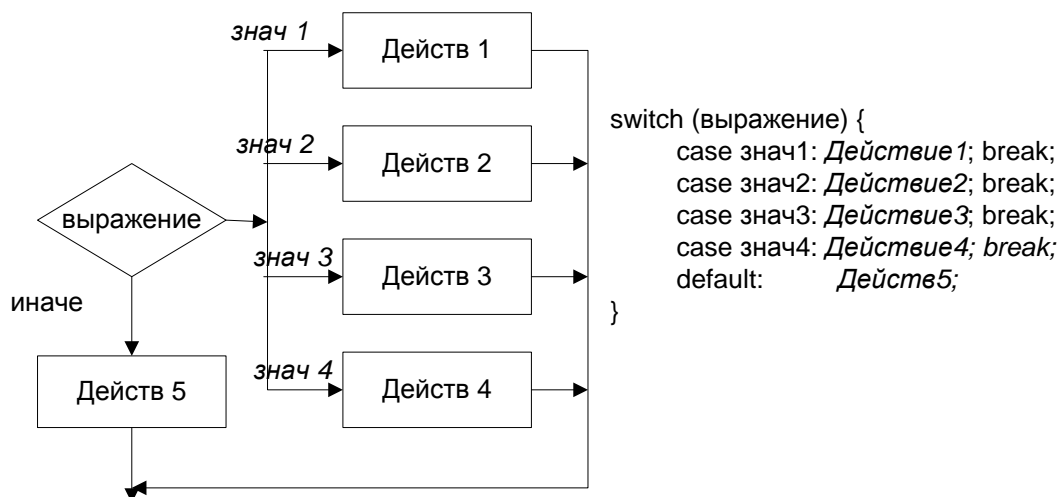
4	<p>Graph showing the function $y = \sin x$ on the interval $[0, \pi]$. The area under the curve is shaded and divided into regions labeled A, B, C, D, E, F, G, and H. A horizontal dashed line is drawn at $y = 1$, and a vertical dashed line is drawn at $x = \pi$.</p>
5	<p>Graph showing the parabola $y = 3 - x^2$ and the line $y = x + 1$. The area between the parabola and the line is shaded and divided into regions labeled A, B, C, D, E, F, G, H, J, and K.</p>
6	<p>Graph showing the unit circle $x^2 + y^2 = 1$, the line $y = x - 1$, and the horizontal line $y = 1$. The area between the circle and the line is shaded and divided into regions labeled A, B, C, D, E, F, G, H, J, and K.</p>
7	<p>Graph showing the parabola $y = x^2$ and the vertical line $x = 1$. The area between the parabola and the y-axis is shaded and divided into regions labeled A, B, C, D, E, and F.</p>

8	
9	
10	

Оператор выбора

Теория

Оператор выбора предназначен для организации выбора одной из любого количества ветвей алгоритма в зависимости от значения некоторого выражения. Оператор выбора обычно используется, когда требуется выполнять много проверок и использование вложенных операторов if слишком громоздко. Для оператора выбора ГОСТ предлагает следующую блок-схему:



Выбор выполняемой ветви инструкции switch осуществляется по значению управляющего выражения, приведенного в круглых скобках после зарезервированного слова switch. Вообще говоря, управляющее выражение инструкции switch может иметь тип bool, целочисленным значением или символом, т.е. выражение в операторе выбора должен иметь порядковый тип.

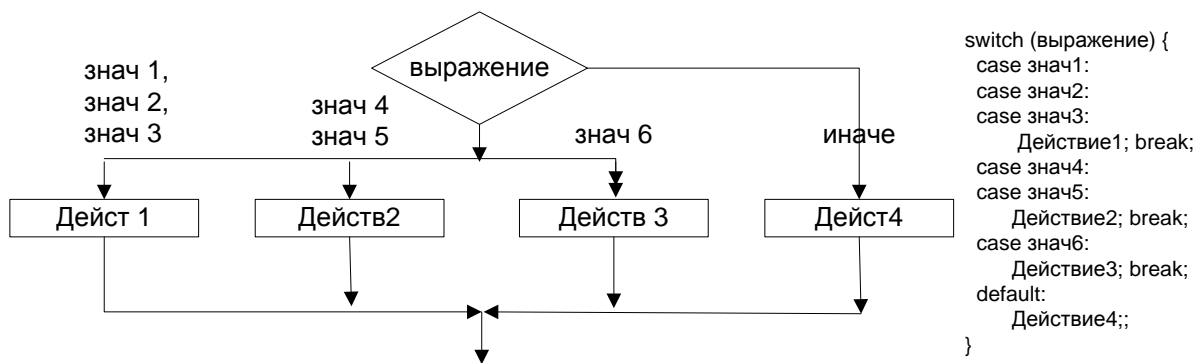
При выполнении инструкции switch сначала вычисляется значение управляющего выражения, затем компьютер просматривает значения констант различных идентификаторов case в поиске совпадения со значением управляющего выражения. Если таковое обнаруживается, выполняется код, идущий за соответствующим case.

Обратите внимание на наличие двоеточия после константы; кроме того, нельзя использовать два оператора case с одинаковой константой, поскольку это вызывает неоднозначность.

Учтите, что компьютер выполняет все инструкции, следующие после удовлетворяющей значению управляющего выражения метки case вплоть до первой встреченной инструкции break. Если инструкция break пропущена, то после выполнения кода для одного case компьютер продолжит выполнение инструкций для последующих case.

Если значение управляющего выражения не совпадает ни с одной из констант меток case, будут выполнены инструкции, следующие за меткой default. Присутствие раздела default необязательно. Если этот раздел отсутствует и ни одна из меток не соответствует значению управляющего выражения, то в результате выполнения инструкции switch ничего не произойдет. Однако безопаснее всегда включать раздел default в инструкцию switch. Если вы уверены в том, что предусмотренные вами метки case учитывают все возможные варианты значений управляющего выражения, в раздел default можно поместить сообщение об ошибке.

Один и тот же код может использоваться несколькими разными метками case.



Пропуск Break

Пропуск `break` в инструкции `switch` не приводит к появлению сообщения компилятора об ошибке. Синтаксически такая инструкция остается совершенно корректной. Однако ее поведение может резко отличаться от ожидаемого разработчиком. Если случайно будет пропущена отмеченная соответствующим комментарием инструкция `break`, то при выполнении инструкции `switch` обработка совпавшей с управляющим выражением по значению метки `case` будет продолжаться до тех пор, пока либо не встретится инструкция `break`, либо не будет достигнут конец всей инструкции `switch`.

Например:

```

int tip;
cout<<"Введите тип транспортного средства:"; cin>>tip;
switch (tip) {
  case 1:
    cout<<"Легковой автомобиль."; toll=0.50;
    break; //Если пропустить эту инструкцию break,
           //пассажиры легковых автомобилей будут платить $1.50.
  case 2:
    cout<<"Автобус"; tol=1.50; break;
  case 3:
    cout<<"Грузовик."; toll=2.00; break;
  default:
    cout>>"Транспортное средство неизвестного типа!";
}

```

Если случайно будет пропущена отмеченная соответствующим комментарием инструкция `break`, то при значении переменной `tip`, равном 1, действия метки `case 1:` будут корректно выполнены, но затем компьютер выполнит действия следующей метки `case`. Это приведет к весьма загадочному выводу: транспортное средство является одновременно и легковым автомобилем, и автобусом; кроме того, окончательное значение переменной `toll` будет равно 1.50, а не 0.50, как должно быть.

Примеры

Пример 7.

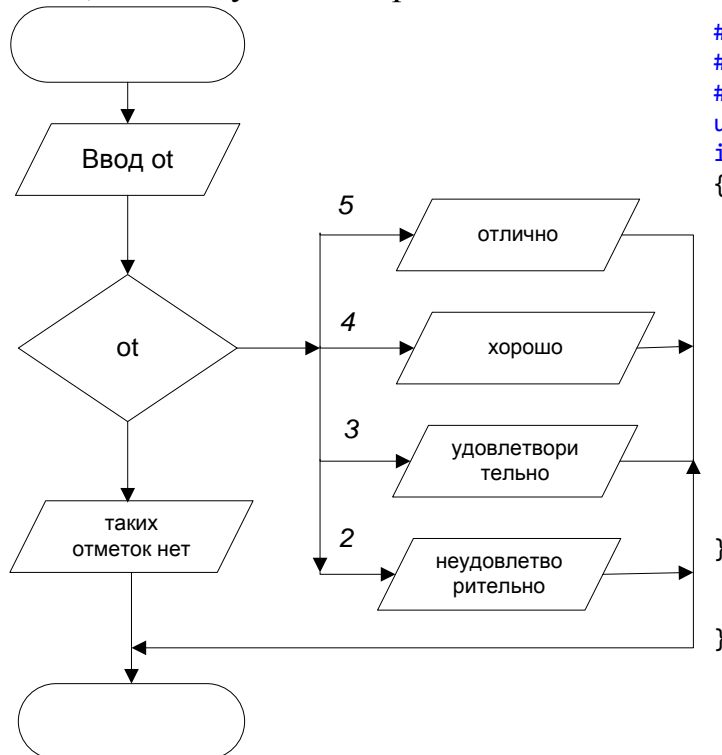
Ввести числовые оценки, вывести текстовые оценки.

Исходные данные: `ot` – целый тип числовая оценка.

Результатом является вывод оценки в виде текста.

Тестовый пример:

- при $ot=4$, вывод «хорошо»;
- при $ot=4$, вывод «таких отметок нет»;
- при $ot=3$, вывод «удовлетворительно».

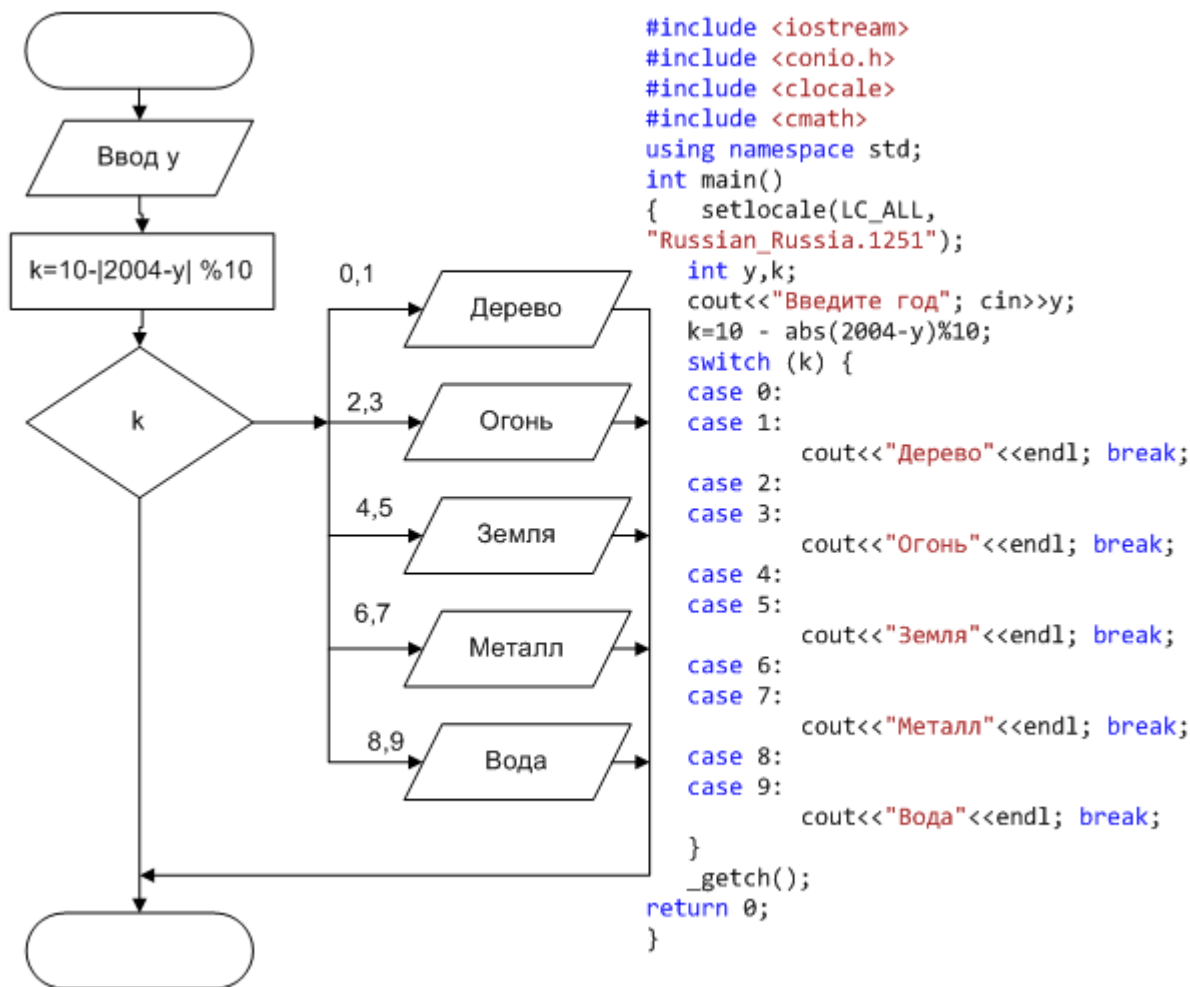


```
#include <iostream>
#include <conio.h>
#include <locale>
using namespace std;
int main()
{ setlocale(LC_ALL, "Russian_Russia.1251");
  int ot;
  cout<<"Введите отметку "; cin>>ot;
  switch (ot) {
    case 5: cout<<"отлично"<<endl; break;
    case 4: cout<<"хорошо"<<endl; break;
    case 3: cout<<"удовлетворительно"<<endl;
             break;
    case 2:
      cout<<"неудовлетворительно"<<endl;
      break ;
    default: cout<<"таких отметок нет"<<endl;
  }
  _getch();
  return 0;
}
```

Пример 8.

В восточном календаре за шестидесятилетний календарный цикл каждый год является не только годом какого-нибудь животного, но и относится в какой-нибудь стихии. Каждая стихия охватывает два года подряд. Стихии следуют в следующем порядке: Дерево, Огонь, Земля, Металл, Вода.

Составить программу, в которой по году определить, к какой стихии он относится. 2004 год – это первый год стихии дерева. Разработать оптимальный вариант программы



Исходные данные: у – год - целый тип;

Результатом является вывод соответствующей стихии.

Для определения стихии надо найти разность по модулю между введенным годом и 2004 годом. Затем найти остаток от деления на 10 от этой разности. Если остаток 0 или 1, то это стихия Дерево, если 2 или 3 – Огонь, если 4 или 5 – Земля, если 6 или 7 – Металл, если 8 или 9 – Вода.

Тестовый пример:

при $y=1966$, вывод «Огонь»;

Пример 9.

Ввести день, месяц и текущего года. Проверить правильность введенной даты.

Исходные данные: m – номер месяца целый тип;

d – номер дня – целый тип.

Результатом является сообщения «правильно» при правильно введенной дате и «не правильно».

В данной задаче следует учитывать, то 1, 3, 5, 7, 8, 10 и 12 месяц имеют 31 день. 4, 6, 9 и 11 месяц имеют 30 дней. Для 2-го месяца следует учитывать, является ли текущий год високосным.

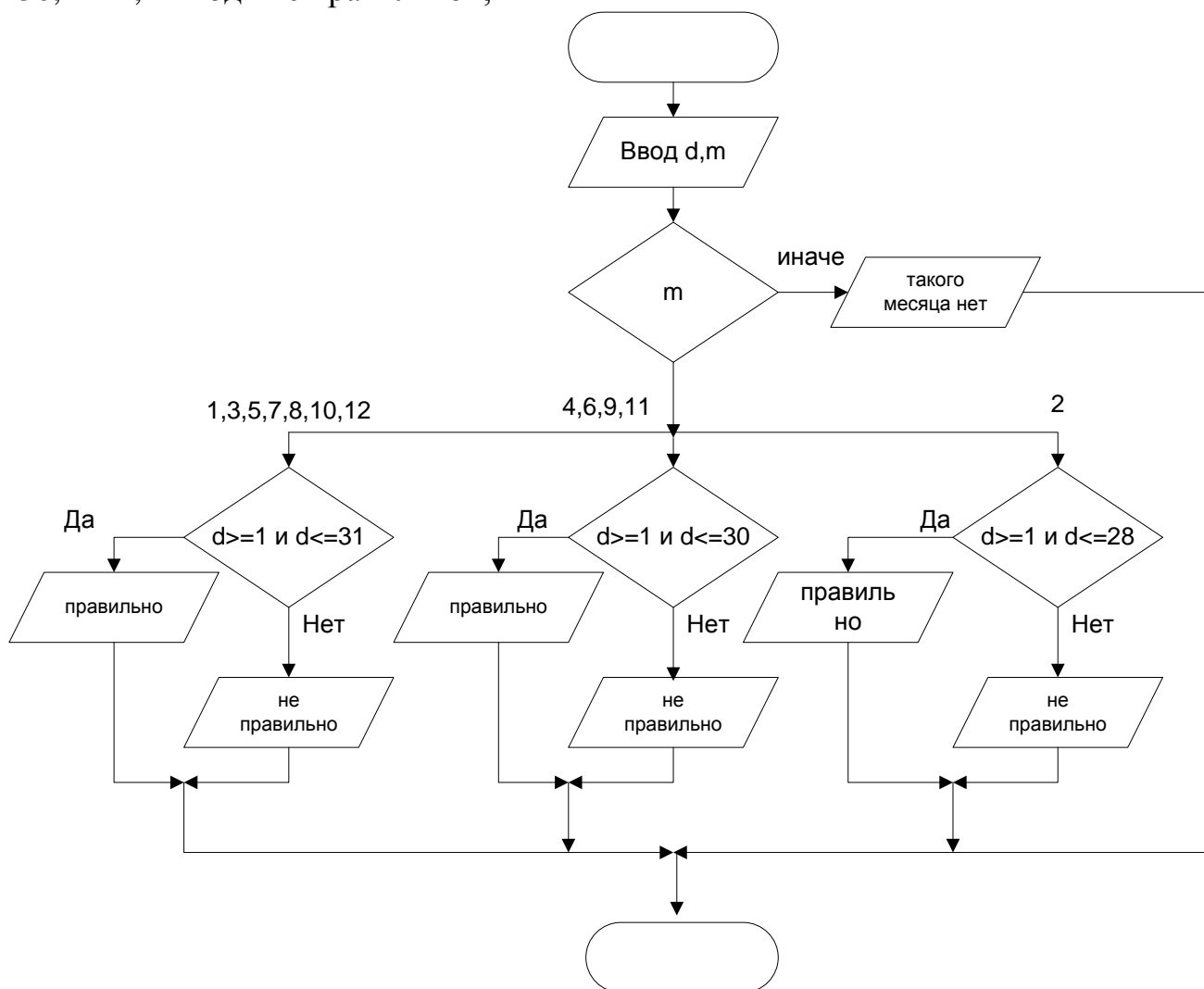
Тестовый пример:

при $d=40$, $m=5$, вывод «не правильно»;

при $d=31$, $m=5$, вывод «правильно»;

при $d=10$, $m=5$, вывод «такого месяца нет»;

при $d=30$, $m=2$, вывод «не правильно»;



Пример 10.

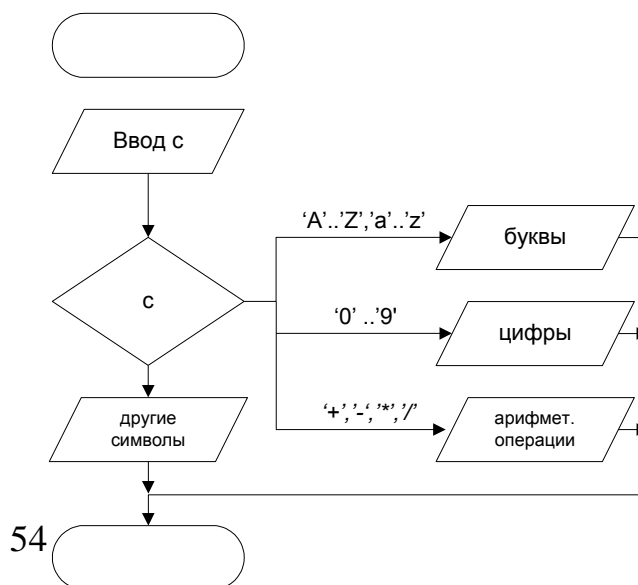
Ввести символ и указать, к какой из групп он относится – буква, цифра, знак арифметических операций или другие символы.

Исходные данные: с введенный символ символьного типа;;

Результатом является вывод соответствующей группы.

Тестовый пример:

- a. при $s='+'$, вывод «арифмет. операции»;
- b. при $s='7'$, вывод «цифры»;
- c. при $s='$',$ вывод «другие символы».



Контрольные вопросы

1. Какой тип должно иметь выражение в операторе switch.
2. Если переменная z в условии задачи может принимать 4 значения, в зависимости от того попадает ли x в интервалы меньше 0, от 0 до корня квадратного от 2, от корня квадратного от 2 до 10, больше 10, может ли эта задача быть решена с помощью оператора switch.
3. Что будет, если пропустить оператор break во всех ветках, в первой ветке, в последней ветке.
4. Определите, чему будет равно S после выполнения следующего оператора при $k=0$:

$S=10$;

```
switch k {  
  case 1: S=0; break;  
  case 2: S=1; break;  
  case 3: S=3; break;  
}
```

5. Определите, чему будет равно S после выполнения следующего оператора $S=1$;

```
switch k {  
  case 1: S=10; break;  
  case 0: S=S+1; break;  
  case 2: S=S+3; break;  
  default S=0; break;  
}
```

Если $k=1$;

6. Почему в примере 8 отсутствует ветка default.

Задание 2 Ввести и отладить программу для примера 10.
--

Индивидуальные задания

1. Некоторое предприятие ежедневно расходует X Квт/час электроэнергии – зимой, Y Квт/час – летом и Z квт/час - весной и осенью. Составить программу, вычисляющую расход электроэнергии R для заданного месяца текущего года.
2. Написать программу, которая бы по введенному месяцу выдает все приходящиеся на этот месяц праздничные дни.
3. Написать программу, которая по введенному номеру месяца выводила название сезона, к которому этот месяц относится, и проверяла правильность введенного месяца.

4. Написать программу, которая бы по введенному номеру единицы измерения (1 – килограмм, 2 – миллиграмм, 3 - грамм, 4 – тонна, 5 – центнер) и массе m выдавала бы соответствующее значение массы в килограммах.
5. Ввести день, месяц и год, проверить правильность введенной даты и вывести дату следующего дня.
6. Дано натуральное число N . Если оно делится на 4, вывести на экран ответ $N=4k$ (где k – соответствующее частное), если остаток от деления на 4 равен 1 – $N=4k+1$, если остаток от деления на 4 равен 2 – $N=4k+2$, если остаток от деления на 4 равен 3 – $N=4k+3$. Например: $12=4*3$, $22=4*5+2$.
7. Ввести два положительных числа меньше 10. Сложить эти два числа и написать результат сложения этих чисел - числителем.
8. Вычислить номер дня в не високосном году по заданным числу и месяцу.
9. Для введенного числа k от 1 до 99 вывести «копеек» учитывая значение k . Например: 20 копеек, но 32 копейки.
10. Вывести по введенному году, какому животному восточного календаря соответствует этот год. Годы внутри цикла носят названия: крыса, бык, тигр, заяц, дракон, змея, лошадь, овца, обезьяна, петух, собака, свинья. Известно, что 2000 год – это год дракона.
11. Составить программу, которая бы по вводимой дате выводила бы название дня недели, соответствующей этой дате.
12. Написать программу, которая бы по введенному номеру времени года (1 – зима, 2 – весна, 3 – лето, 4 - осень) выдавала соответствующее этому времени года месяцы, количество дней в каждом из этих месяцев.

Тема № 3. Циклический вычислительный процесс

Циклическим называется алгоритм, в котором некоторая часть операция (тело цикла последовательность команд) выполняется многократно. Цикл не может быть бесконечным, т.к. алгоритм должен приводить к результату через конечное число шагов. В цикл обязательно должен входить блок проверки условия. В зависимости от того, где располагается это условие, циклы делятся на циклы типа «пока» (условие располагается перед телом цикла) и циклы типа «до» (условие располагается после тела цикла).

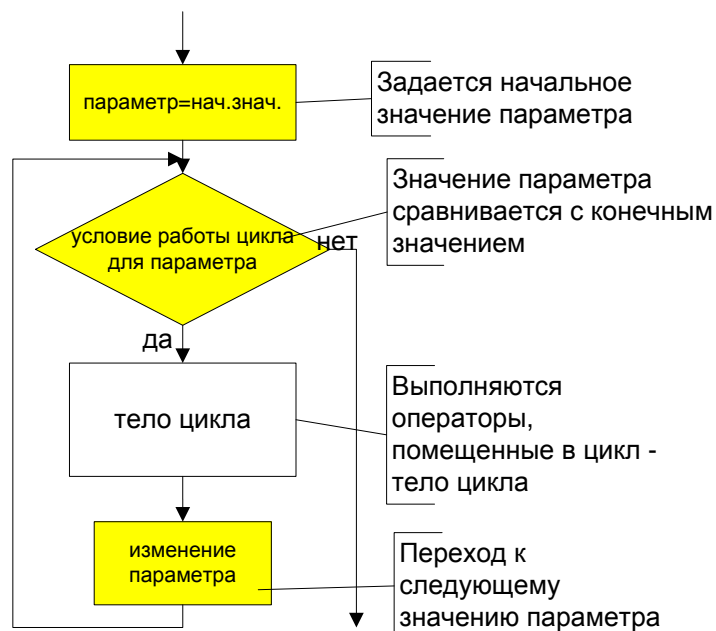
Другой виз классификации циклов – это циклы с параметром и циклы с выходом по условию.

Лабораторная работа № 4

Циклы с параметром

Теория

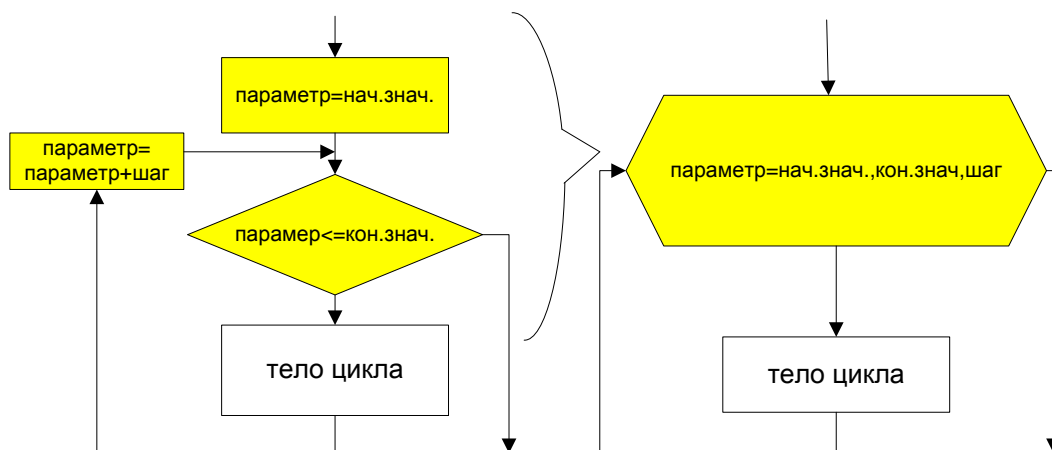
Цикл с параметром или цикл со счетчиком используется для организации циклов, в которых можно выделить параметр с известным начальным значением, конечным значением и определить закономерность, по которому параметр изменяется при каждой итерации цикла. **Итерацией** цикла называется каждое повторение исполнения тела цикла Алгоритм работы оператора цикла с параметром можно представить следующей блок-схемой:



Для реализации этого алгоритма оператор цикла с параметром имеет следующий вид:

```
for (параметр=нач.знач.; условие_работы_цикла; изменение_параметра)  
оператор_тела_цикла;
```

Если тело цикла состоит более чем из одного оператора, эти операторы заключаются в фигурные скобки. Как видно из блок-схемы, в цикле for проверка выполнения условия прекращения работы цикла осуществляется перед первой итерацией, и, таким образом, возможен цикл for, тело которого не выполняется ни разу. В большинстве случаев параметр цикла с каждой итерацией изменяется на одну и ту же величину – шаг. В этом случае блок схема может быть сокращена, и, согласно ГОСТ принято еще одно обозначения цикла с параметром:

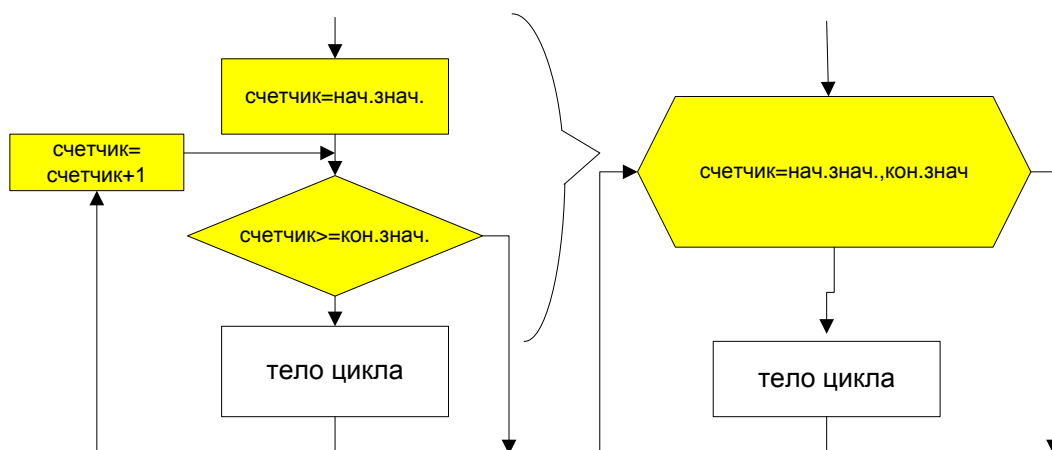


Оператор соответствующий такой блок-схеме имеет следующий вид:
for (парам.=нач.знач.; парам.<=кон.знач.; парам.=парам.+шаг)

Например:

```
for (a=0; a<=3; a=a+0.1)
cout<<"sin("<<a<<")="<<sin(a);
```

Если надо считать, сколько раз проработал цикл, то шаг равен 1, и цикл можно назвать не цикл с параметром, а цикл со счетчиком. В этом случае в сокращенной блок-схеме можно не указывать шаг.



Оператор соответствующий такой блок-схеме имеет следующий вид:
for (счетчик=нач.знач.; счетчик<=кон.знач.; счетчик++)

Тип параметра можно описывать непосредственно в цикле, но при этом следует помнить, что за пределами цикла этот параметр не определен.

Например:

```
for (int i=1; i<=10; i++)
```

Примеры

Пример 1.

Вычислить 10 значений d : $d=a^2-b^2+ab-8$,

$$\text{где } b = \begin{cases} c, & \text{если } a < -3 \\ c/3, & \text{если } -3 \leq a \leq 0 \\ a+c+4, & \text{если } a > 0 \end{cases}$$

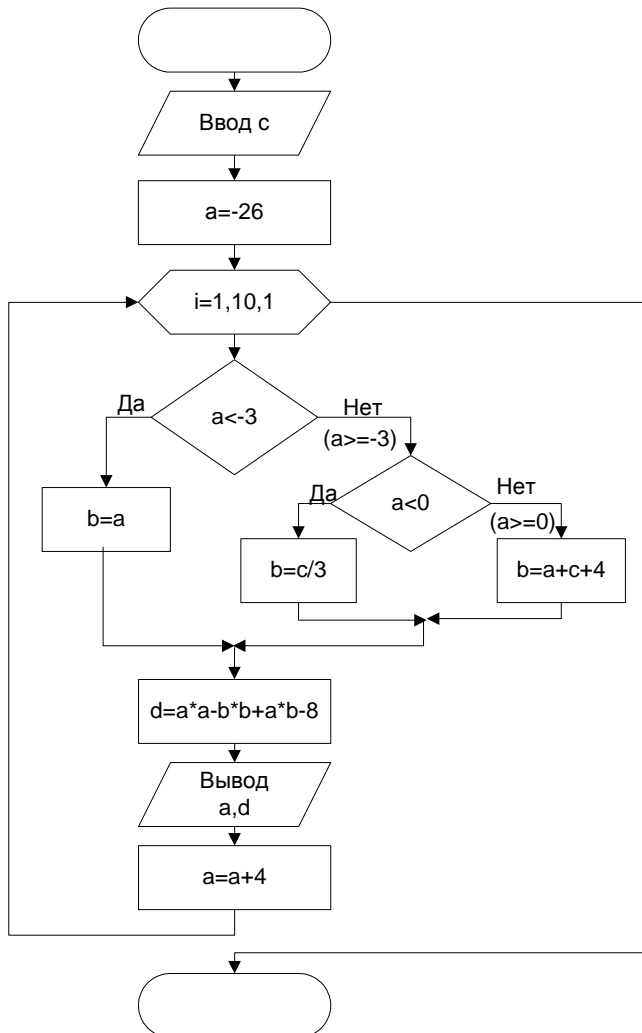
a изменяется с шагом 4, начальное значение a равно -26. Значение c – произвольное число.

Исходные данные: Значение c – вещественное число.

Результат: 10 значений d .

Тестовый пример: при $c=3$

i	a	d
1	-26	668
2	-22	476
3	-18	316
4	-14	188
5	-10	92
6	-6	28
7	-2	-7
8	2	-67
9	6	-63
10	10	-27



```

#include <iostream>
#include <conio.h>
using namespace std;
int main()
{double a,b,c,d;
  cout<<"c="; cin>>c;

  a=-26;

  for(int i=1 ;i<=10;i++)

    {if (a<-3) b=a;
     else

        if (a<0) b=c/3;
        else b=a+c+4;

      d=a*a-b*b+a*b-8;

      cout<<"a= "<<a<<"\td= "<<
        d<<endl;

      a=a+4;
    }
  _getch(); return 0;
}

```

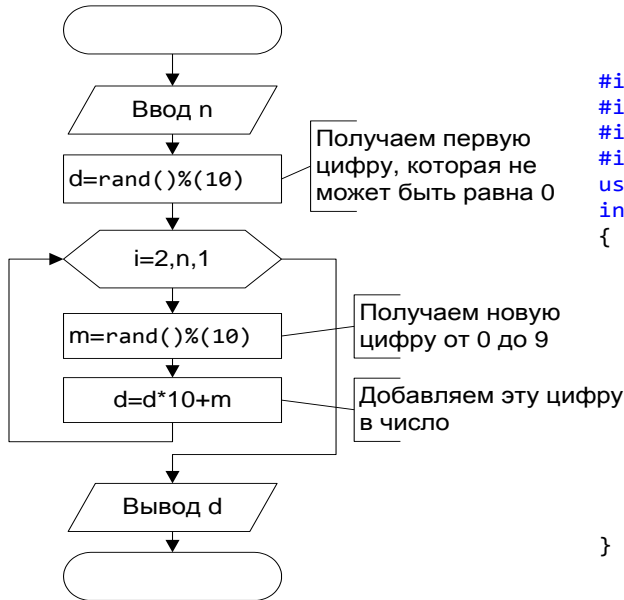
Пример 2.

Дано натуральное $n < 10$. Получить n -значное натуральное число.

Исходные данные: количество цифр n – целый тип.

Результат: d – целый тип.

Тестовый пример: проверяется только количество цифр.



```
#include <iostream>
#include <conio.h>
#include <time.h>
#include <stdlib.h>
using namespace std;
int main()
{   int n,m,d;
    srand((unsigned)time(NULL));
    cout<<"n= "; cin>>n;
    d=rand()%(10);
    for(int i=2;i<=n;i++)
    {
        m=rand()%(10);
        d=d*10+m;
    }
    cout<<"d= "<<d<<endl;
    _getch();
    return 0;
}
```

Пример 3.

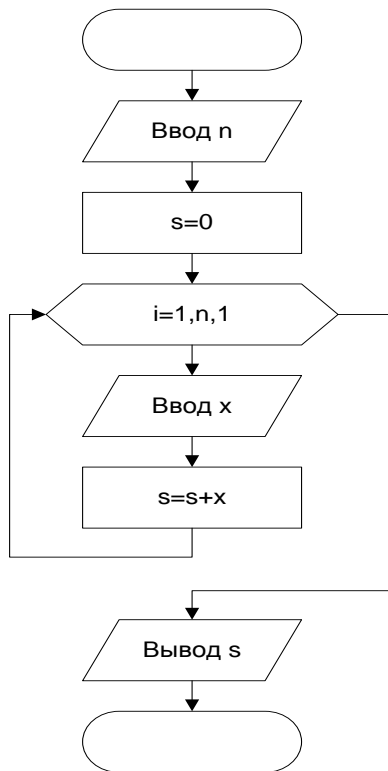
Вычислить сумму n вещественных чисел.

Исходные данные: n - количество введенных чисел - целый тип; x – переменная, куда помещаются вводимые числа - вещественный тип.

Результат: s - сумма введенных чисел – вещественный тип.

При вычислении суммы выполняется операция накопления данных в одной переменной, в данном случае в переменной s : $s=s+x$. Эта операция должна выполняться n раз. Для того чтобы в результате первой операции накопления было получено значение первого x , переменную S предварительно надо очистить, т.е. $s=0$.

Тестовый пример: при $n=7$ и последовательности: 1.2, 3, 5.6, 6.1, -4, -1, 4.2, $s=17.1$.



```

#include <iostream>
#include <conio.h>
using namespace std;
int main()
{int n,x,s=0;
  cout<<"n="; cin>>n;

```

```

  for (int i=1; i<=n; i++)
  {
    cout<<"x= "; cin>>x;

    s+=x;
  }

  cout<<"s= "<<s;
  _getch();
  return 0;
}

```

Пример 4.

Во многих языках программирования отсутствует стандартная операция возведения в степень. Необходимо написать программу возведения в целую степень вещественного числа.

Исходные данные: n - степень, в которую возводится число – целый тип. x - число, которое возводится в степень n – вещественный тип.

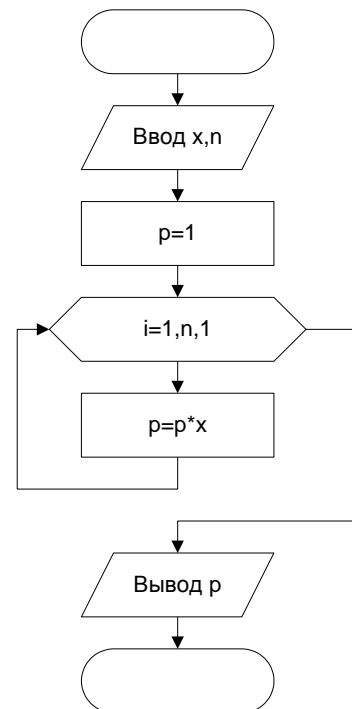
Результат: p – вещественный тип.

Возвести в n-ю степень число – значит перемножить это число n раз

$$p = \underbrace{x \cdot x \cdot x \cdot \dots \cdot x}_n$$

Операция перемножения x похожа на операцию сложения, только знак «+» заменяется на знак «*». Как и при сложении, в результате первого действия умножения надо получить само число x, т.е., и в этом отличие от суммирования, первоначально p должно быть равно 1.

Тестовый пример: При x=4 и n=5, p=1024.



Пример 5.

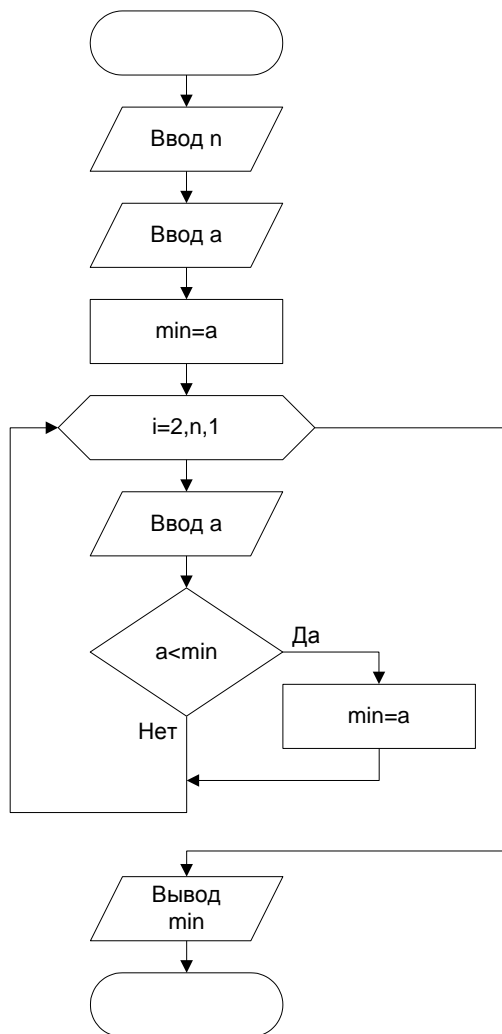
Ввести последовательность из n целых чисел. Найти минимальный член этой последовательности.

Исходные данные: n - количество введенных чисел - целый тип; а переменная, куда помещаются вводимые числа – целый тип.

Результат: min – целый тип.

Поиск минимального члена последовательности чисел похож на процедуру определения лучшего результата на соревновании лыжников с раздельным стартом. Когда первый лыжник прибегает на финиш, его результат считается лучшим, т.е. в min записывается этот результат. Затем, когда прибегает очередной лыжник, его результат сравнивается с лучшим (min), и если этот результат меньше, то в значение min записывается этот результат. Следует обратить внимание, что первоначальное значение min не может равняться нулю, так как в результате будет получен этот ноль.

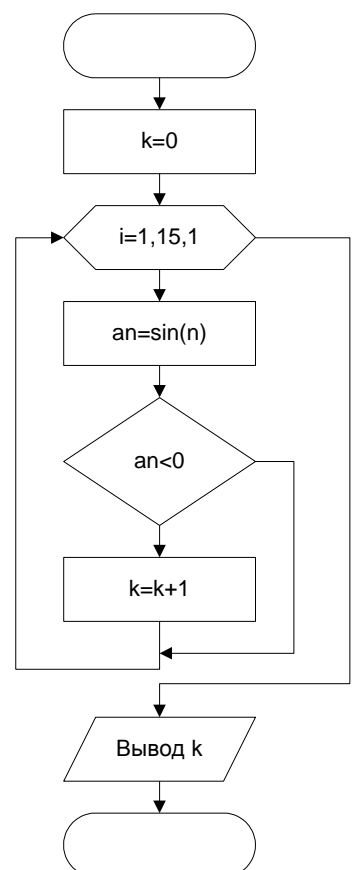
Поиск максимального элемента отличается от поиска минимального только проверкой: если новое значение больше текущего максимального, в максимальное значение записывается это новое значение.



Тестовый пример: при n=7, последовательность: 8, 0, -1, 2, 10, 3, 5, min=-1.

Пример 6.

Дана числовая последовательность $a_n = \sin n$, где n=1, ... 15. Определить, сколько отрицательных элементов в этой последовательности.



Исходные данные: количество элементов последовательности – 15.

Результат: количество отрицательных элементов k – целый тип.

Для нахождения количества отрицательных членов необходимо вычислить член последовательности и сравнить его с 0, если член меньше 0 увеличить k на единицу. В начале задачи k необходимо обнулить.

Тестовый пример: С помощью Excel получаем $k=6$.

Рассмотренные алгоритмы относятся к так называемым стандартным алгоритмам. Отметим, что у всех этих алгоритмов много общего. Рассмотрим следующую таблицу:

этап	Сумма s	Произведение p	Количество k	Минимум min
До цикла	$s=0$	$p=1$	$k=0$	$min=$ первый элемент
В цикле	1	Получение элемента последовательности	Получение элемента последовательности	Получение элемента последовательности
	2	Проверка, надо ли данный элемент суммировать (необязательно)	Проверка, надо ли данный элемент умножать (необязательно)	Проверка, элемент меньше Min?
	3	$s=s+элемент$	$p=p*элемент$	$k=k+1$

Пример 7.

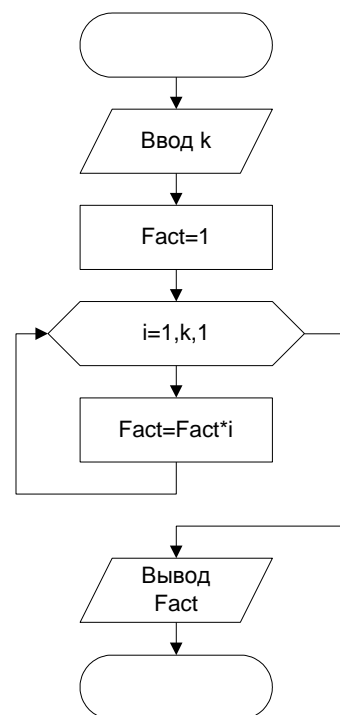
Вычислить факториал k . Факториалом называется произведение $k!=1\cdot2\cdot3\cdot\dots\cdot k$.

Исходные данные: k – целый тип.

Результат: fact – тип long int.

Факториал числа очень быстро растет. Поэтому результат должен иметь длинный целый тип.

Тестовый пример: При $k=5$, fact=120.



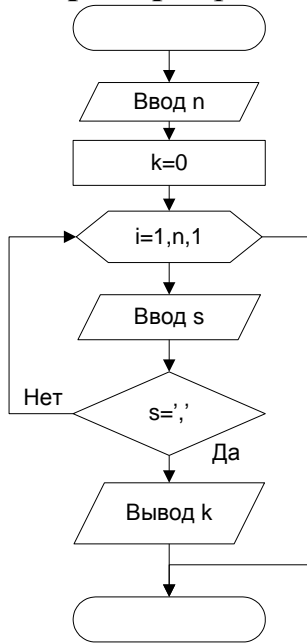
Пример 8.

Даны натуральное число n , символы s_1, \dots, s_n . Известно, что среди s_1, \dots, s_n есть по крайней мере одна запятая. Найти такое натуральное i , что s_i – первая по порядку запятая;

Исходные данные: Количество элементов n целое число, s – вводимый символ.

Результат: k – номер первой запятой.

Тестовый пример: при n=10 и последовательности '4hgj,e,6+', k=5.



```
#include <iostream>
#include <conio.h>
#include <locale>
using namespace std;
int main()
{int n,k=0;
char s;
setlocale(0, "");
cout<<"n="; cin>>n;
cout<<"Введите последовательность символов в строку ";
for(int i=1;i<=n;i++)
{ cin>>s;
if (s==',')
{cout<<"номер первой запятой - "<<i<<endl;
break;
}
}
_getch();
return 0;
}
```

Во многих задачах требуется выполнять вычисления над элементами числовой последовательности, в которой каждый член a_n является функцией натурального аргумента n. В примере 6 приведен вариант такой последовательности. Часто по элементам числовой последовательности требуется определить формулу для вычисления n-го элемента.

Пример 9.

Дана числовая последовательность $\{a_1=2, a_2=5, a_3=8, \dots\}$. Члены последовательности с четными номерами заменили на обратные им числа (5 на -5). Найти сумму членов последовательности с десятого по тридцать первый включительно.

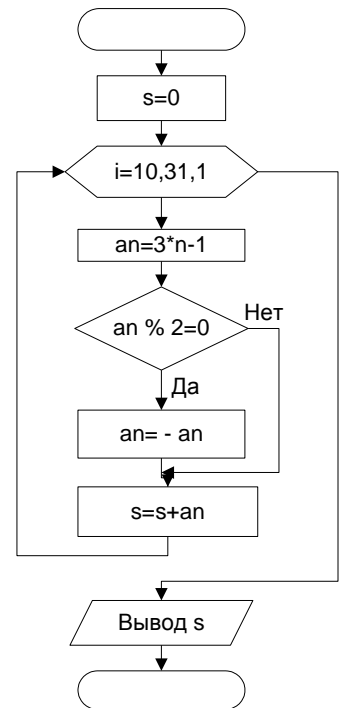
Исходные данные: начальный член последовательности 10, конечный член последовательности – 31.

Результат: Сумма членов последовательности s – целый тип.

Необходимо определить формулу для вычисления члена последовательности в зависимости от номера члена. С увеличением номера на 1 член последовательности изменяется на 3, т.е. член зависит от $3 \cdot n$. При n=1 элемент равен 2. Чтобы получить 2 необходимо из $3 \cdot n$ вычесть 1. Таким образом $a_n=3 \cdot n-1$.

Чтобы определить четность номера достаточно найти остаток от деления номера на 2. Если остаток равен 0, то номер четный.

Тестовый пример: с помощью Excel получаем s=-33



Существуют числовые последовательности, в которых каждый новый член вычисляется через предыдущие. Формула, в которой каждый член последовательности вычисляется через предыдущие, называется **рекуррентной**. Рекуррентная формула, если она существует, позволяет вычислить любой член последовательности посредством вычисления всех предыдущих членов.

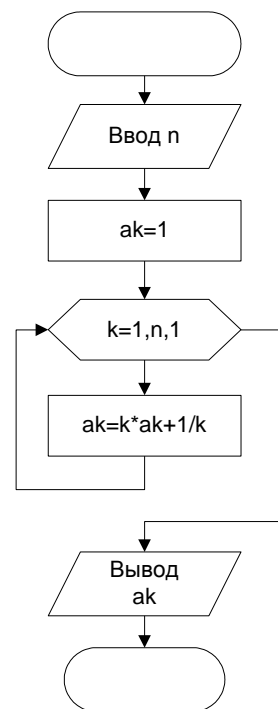
Пример 10.

Пусть $a_0=1$; $a_k=ka_{k-1}+1/k$, $k=1,2,\dots$. Дано натуральное число n . Получить a_n .

Исходные данные: номер элемента n - целый тип.

Результат: n -й элемент последовательности a – вещественный тип.

Тестовый пример: с помощью Excel получаем при $n=5$, $a_n=278.12$.



Рассмотрим пример более сложной рекуррентной формулы

Пример 11.

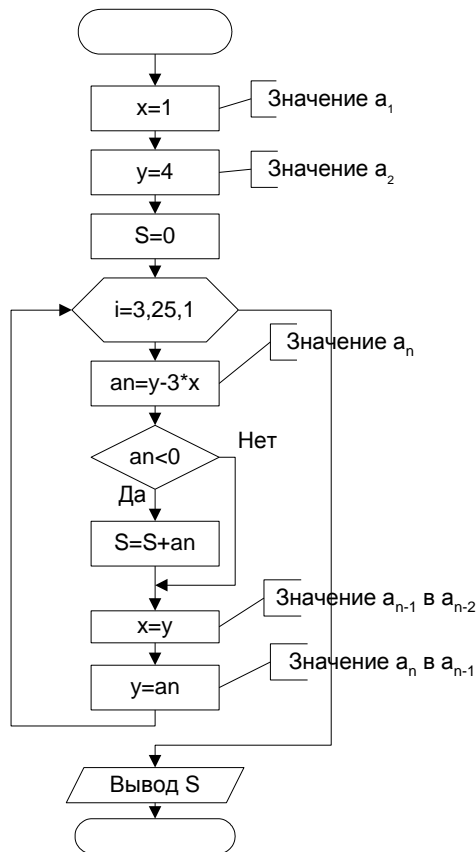
Дана числовая последовательность: $\{a_1=0, a_2=4, \dots, a_n=a_{n-1}-3a_{n-2}\}$. Найти сумму отрицательных элементов при $n=25$.

Исходные данные: количество элементов n - целый тип, значение x - a_{n-2} целый тип, y - a_{n-1} целый тип.

Результат: Сумма отрицательных элементов S – целый тип.

Для вычисления нового значения a_n необходимо знать 2 предыдущих значения. Для получения нового значения a_n надо знать a_{n-1} и a_{n-2} , которые необходимо сохранять в дополнительных переменных.

Тестовый пример: с помощью Excel получаем $S= -1644128$



Пример 12.

Даны натуральное число n , символы s_1, \dots, s_n . Выяснить, имеются ли в последовательности s_1, \dots, s_n такие члены последовательности, что s_i и s_{i+1} совпадает с буквой t .

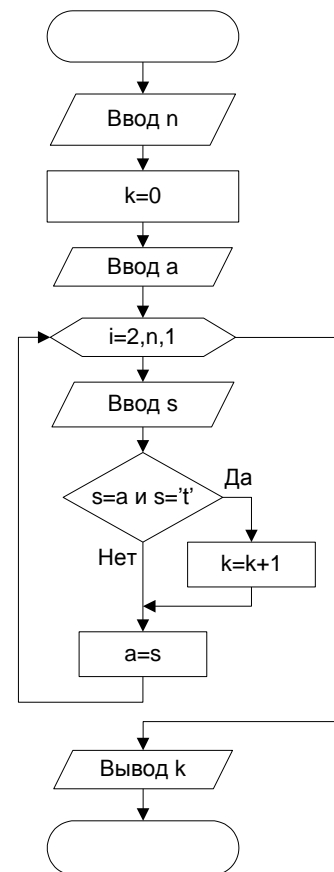
Исходные данные: Количество элементов n целое число, s – вводимый символ.

Результат: k – количество пар, равных t .

Для сравнения пар введенных символов необходимо сохранять предыдущий введенный символ в переменной a .

Тестовый пример:

при $n=15$ в последовательности 'ertbyttbnvttvcd' $k=2$.



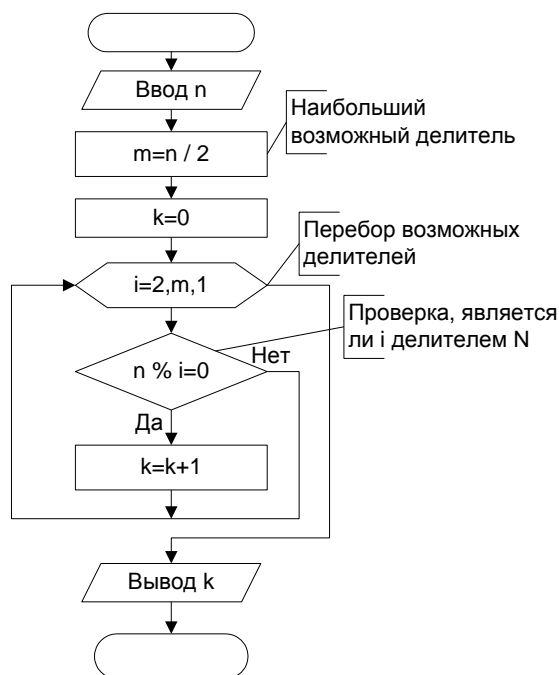
Пример 13.

Дано натуральное число n . Определить, сколько у этого числа делителей (1 и само число не учитывать).

Исходные данные: n – целый тип.

Результат: количество делителей k - целый тип.

Тестовый пример: при $n=24$, $k=6$.



Пример 14.

Даны натуральное число n , символы s_1, \dots, s_n . Подсчитать наибольшее количество идущих подряд запятых.

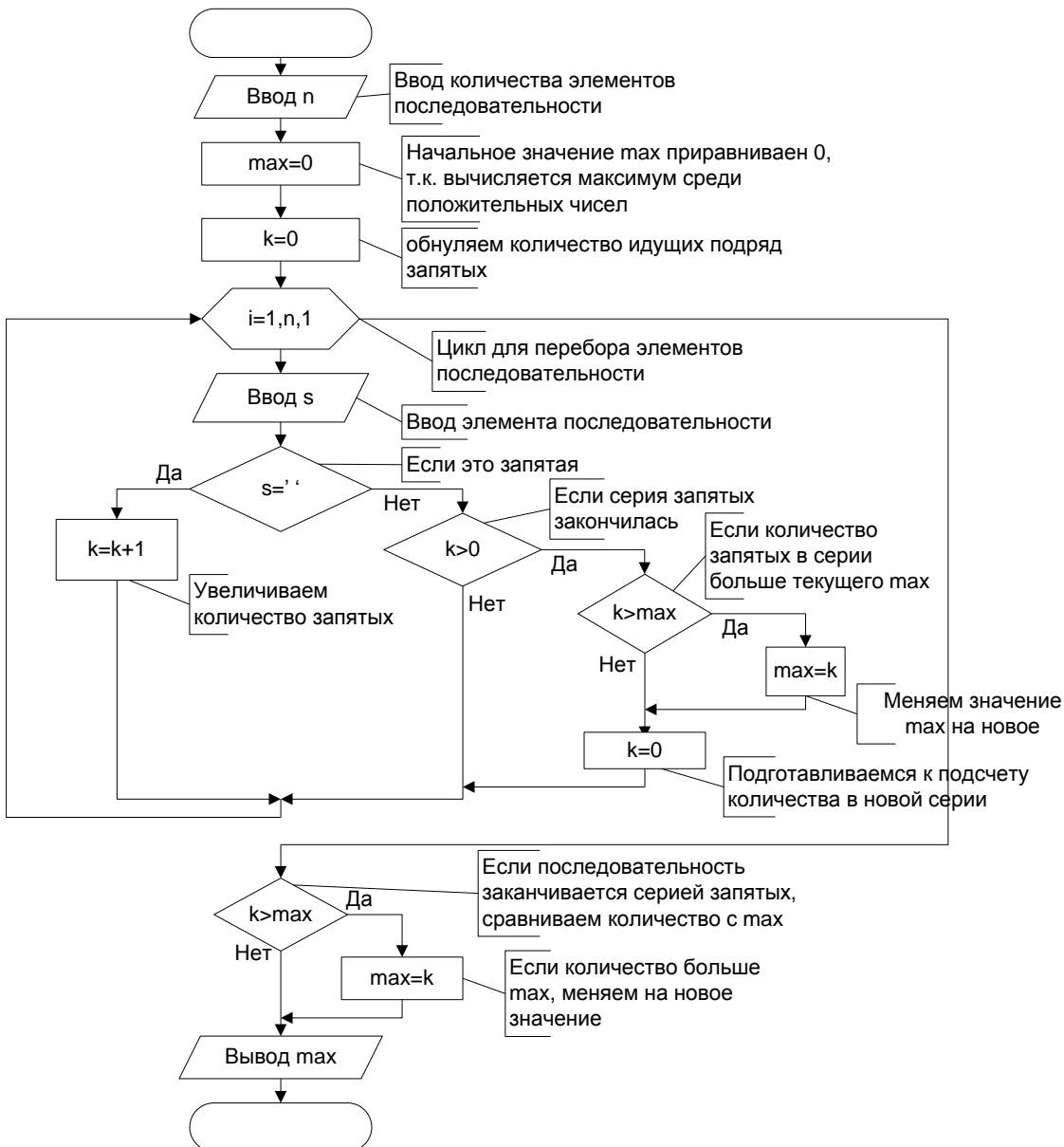
Исходные данные: натуральное число n , вводимые символы s .

Результат: Максимальное количество пробелов \max целый тип.

Если введенный символ равен ‘,’ надо увеличивать количество запятых идущих подряд на 1 (k). Если символ не запятая, надо проверить k , если символ следует сразу же после запятой ($k > 0$) надо сравнить k с максимальным количеством запятых на данный момент. Если $k > \max$, надо изменить значения \max . После сравнения надо обнулить k перед поиском количества в новой серии запятых.

Тестовый пример:

при $n=20$, последовательность ‘23,,wer,,,,tyu,,,cv’, $\max=5$.



Задание 1 Написать программу и отладить Пример 13

Контрольные вопросы

1. Как можно выйти досрочно из цикла.
2. Если начальной значение счетчика окажется меньше конечного значения, будет ли выполняться тело цикла хотя бы один раз.
3. Можно ли при поиске максимального значения в произвольной последовательности чисел первоначальное значения максимума задавать равное нулю и почему.
4. Как будет выглядеть блок-схема примера 3, если надо найти не минимум, а максимум.
5. Почему в примере 12 $m=n/2$ – наибольший возможный делитель?
6. Объясните проверку условия в примере 12.
7. Объяснить в примере 11 две последние операции в цикле.

Индивидуальные задания

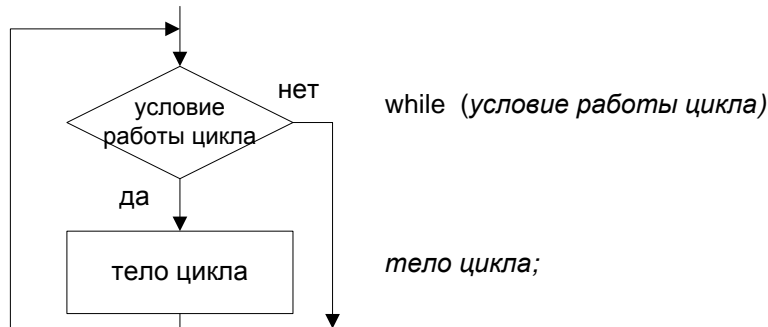
1. Спортсмен в первый день пробежал 10 км. Каждый следующий день он увеличивал дневную норму на 10% от результата предыдущего дня. Найти какой путь пробежит спортсмен на 7 день.
2. Даны натуральное число n , символы s_1, \dots, s_n . Подсчитать:
 - сколько раз среди данных символов встречается символ $+$ и сколько раз символ $*$;
 - общее вхождение символов $+$, $-$, $*$ в последовательность s_1, \dots, s_n .
3. Даны натуральное число n , символы s_1, \dots, s_n . Известно, что среди s_1, \dots, s_n есть по крайней мере одна запятая. Найти такое натуральное i , что s_i – последняя по порядку запятая.
4. Дана числовая последовательность $\left\{ \frac{1}{8}, \frac{1}{12}, \frac{1}{16}, \dots \right\}$. Найти сумму членов с 10 по 25-й включительно.
5. Дано натуральное число n . Найти наибольшее среди чисел $ke^{\sin^2(k+1)}$ ($k=1, \dots, n$), а также сумму всех этих чисел.
6. Вычислить $\sum_{i=1}^{30} (a_i - b_i)^2$, где
$$a_i = \begin{cases} i, & \text{если } i \text{ - нечетное} \\ i/2, & \text{в противном случае} \end{cases} \quad b_i = \begin{cases} i^2, & \text{если } i \text{ - нечетное} \\ i^3, & \text{в противном случае} \end{cases},$$
7. Дано натуральное число n , действительные числа y_1, y_2, \dots, y_n . Найти:
$$\text{Max}(|z_1|, |z_2|, \dots, |z_n|), \text{ где}$$
$$z_i = \begin{cases} y_i, & \text{при } |y_i| \leq 2, \\ 0.5, & \text{в противном случае} \end{cases}$$
8. Даны натуральное n . Действительные числа a_1, \dots, a_n . Определить в этой последовательности число соседств двух чисел разного знака.
9. Пусть $x_1=0.3$; $x_2=-0.3$; $x_i=i+\sin(x_{i-2})$, $i=3,4, \dots, 100$. Найти в этой последовательности ближайшее к какому-нибудь целому.
10. Даны натуральное число n , символы s_1, \dots, s_n . Выяснить, имеются ли в последовательности s_1, \dots, s_n такие члены последовательности s_i, s_{i+1} , что s_i – это запятая, а s_{i+1} – тире.
11. Даны натуральное число n , символы s_1, \dots, s_n . Выяснить, верно ли, что в последовательности s_1, \dots, s_n имеются пять идущих подряд букв e .
12. Даны натуральное число n , символы s_1, \dots, s_n . Группы символов, разделенные пробелами (одним или несколькими) и не содержащими пробелов внутри себя, будем называть словами. Подсчитать количество букв в последнем слове данной последовательности.

Цикл с условием

Теория

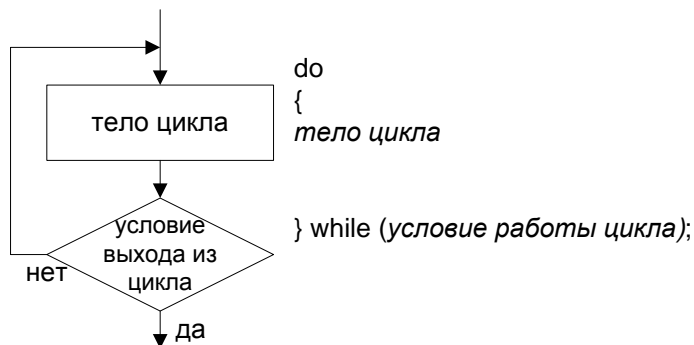
Когда не известно сколько раз проработает цикл, используются циклы с условием. Окончание цикла происходит, когда выполняется/не выполняется условие. Существует два таких оператора цикла. Это цикл с предусловием или цикл «пока», и цикл с постусловием или цикл «до».

Цикл с предусловием сначала проверяет, нужно ли выполнять операторы в цикл (*тело цикла*), а затем выполняет эти операторы. Блок-схема такого цикла и оператор ее реализующую:



Точно так же как и в операторе со счетчиком, если тело цикла состоит из нескольких операторов, используются операторные скобки { }. Если тело цикла состоит из одного оператора, операторные скобки можно не использовать.

В цикле с постусловием сначала выполняется тело цикла, а потом идет проверка условия выхода из цикла. Блок-схема и оператор имеет вид.



При разработке программы выбор конкретного типа цикла лучше отложить. Вначале следует разработать блок-схему цикла, а уже затем преобразовать блок-схему в инструкции языка C++. На этом этапе будет проще принять решение об использовании определенного типа цикла языка C++.

Цикл for обычно используется при числовых расчетах с применением переменной, изменяющейся при каждом проходе цикла на одну и ту же величину (вообще говоря, цикл for следует рассматривать в качестве основной альтернативы всякий раз, когда мы имеем дело с числовыми расчетами).

В большинстве прочих случаев следует использовать циклы while и do-while; выбор нужного — задача довольно легкая. Если тело цикла должно быть выполнено по крайней мере один раз, следует использовать цикл do-while. Если же возможна

ситуация, когда тело цикла не будет выполняться ни разу, остановите свой выбор на цикле `while`. Он довольно часто используется при считывании входных данных, когда есть вероятность того, что входные данные могут отсутствовать вообще.

Оператор досрочного выхода из цикла `break`.

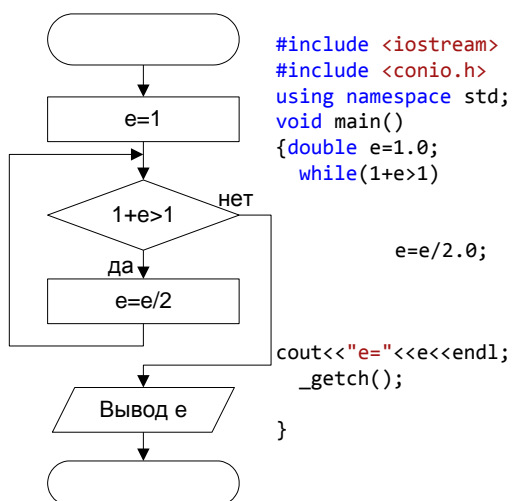
Примеры

Пример 15.

Вещественные числа в программировании всегда являются приближенными т.е. несут в себе погрешность, которая называется погрешностью машинного округления. Разность между вещественной единицей и ближайшим к ней числом, представимым в памяти компьютера называется машинным эпсилон. Найти машинный ϵ .

Исходные данные: начальное значение $e=1$. e – вещественное число.

Результат: значение, которое существенно для компьютера.

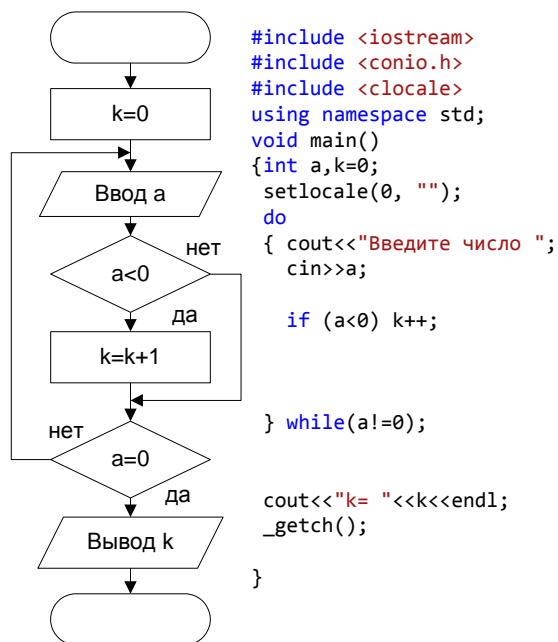


Пример 16.

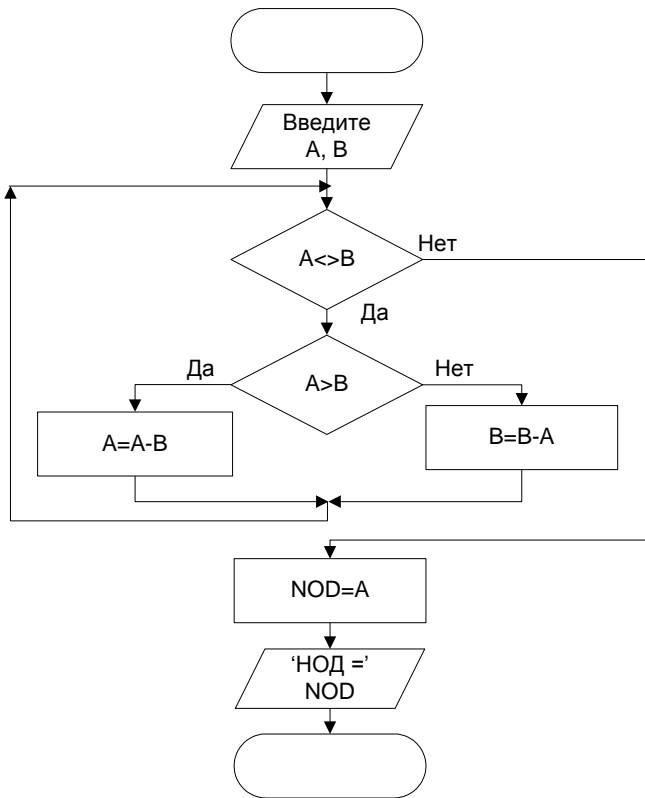
Дан ряд целых чисел. Сколько отрицательных чисел стоит до первого нуля.

Исходные данные: члены ряда записываются в переменную a – целый тип.

Результат: количество отрицательных цифр k – целое.



Тестовый пример: для ряда 5 -9 6 -8 -3 5 9 0, $k=3$.



ана числовая последовательность:
 $\{a_1=8.2, a_2=7.9, a_3=7.6, \dots\}$
 Найти сумму всех положительных членов.

Исходные данные: элементы последовательности a – вещественный тип.
 Из приведенных данных видно, что очередной член числовой последовательности получен из предыдущего члена вычитанием 0.3.
Результат: Сумма положительных элементов последовательности S – вещественный тип.

Тестовый пример: $S=116.2$.

Пример 19.

Дано натуральное число. Определить, сколько цифр в этом числе.

Исходные данные: n – целый тип.
Результат: количество цифр k – целый тип.

Пример 17.

Алгоритм Евклида. Даны целые числа A и B . Найти наибольший общий делитель этих чисел.

Исходные данные: A, B – целый тип

Результат: NOD наибольший общий делитель

Тестовый пример:
 при $A=36, B=48, \text{НОД}=12$.

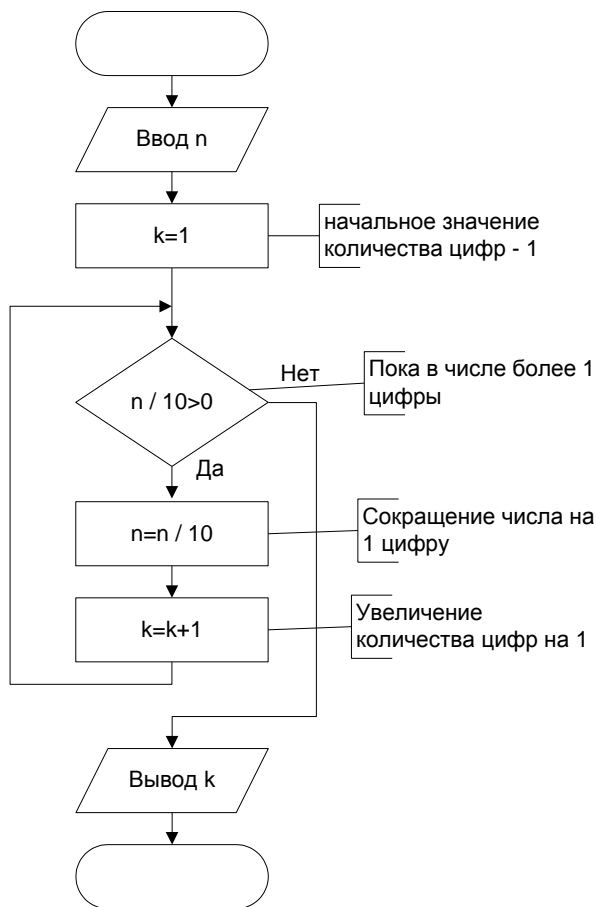
Пример 18.
 Д



Для получения количества цифр надо выполнять целочисленное деление на 10, до достижения 0.

Тестовый пример:

при n=12345, k=5.



Пример 20.

Создать программу вычисляющую $\sqrt[k]{a}$.

Для этого надо вычислить элементы числовой последовательности:

$$x_0 = a; x_i = \frac{k-1}{k} x_{i-1} + \frac{a}{k \cdot x_{i-1}^{k-1}}, i = 1, 2, \dots$$

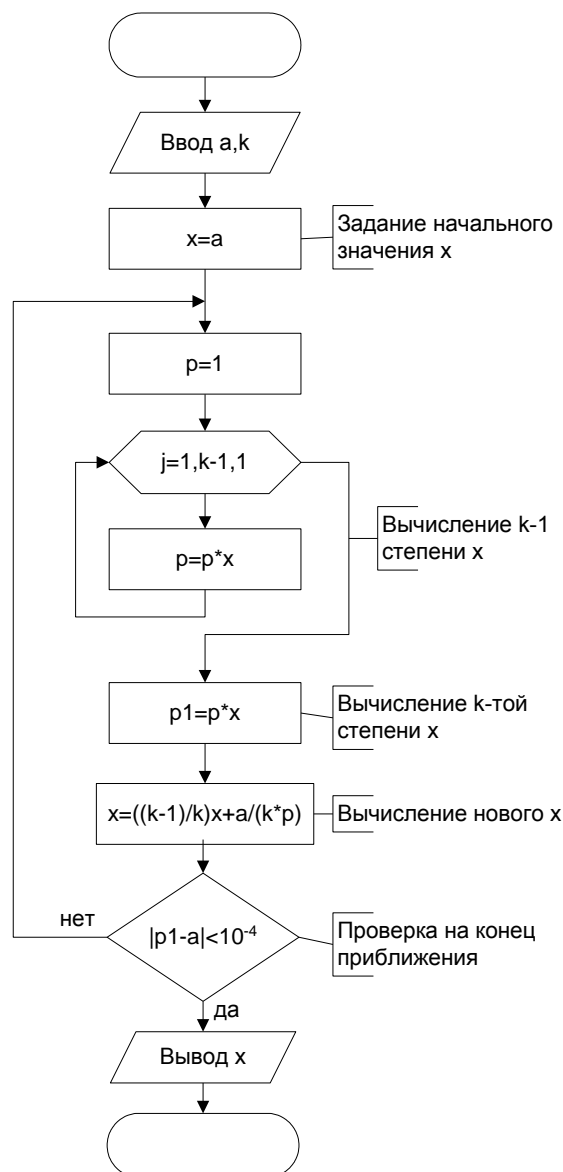
Найти первое значение x_n , для которого $|x_n^k - a| < 10^{-4}$.

Исходные данные: a – вещественный тип, k – целочисленный тип

Результат: x – вещественный тип

Тестовый пример: при a=8, k=3, x=2.

Пример 21.



Дано натуральное число N . Верно ли утверждение, что цифры в этом числе образуют возрастающую последовательность.

Исходные данные: N целый тип.

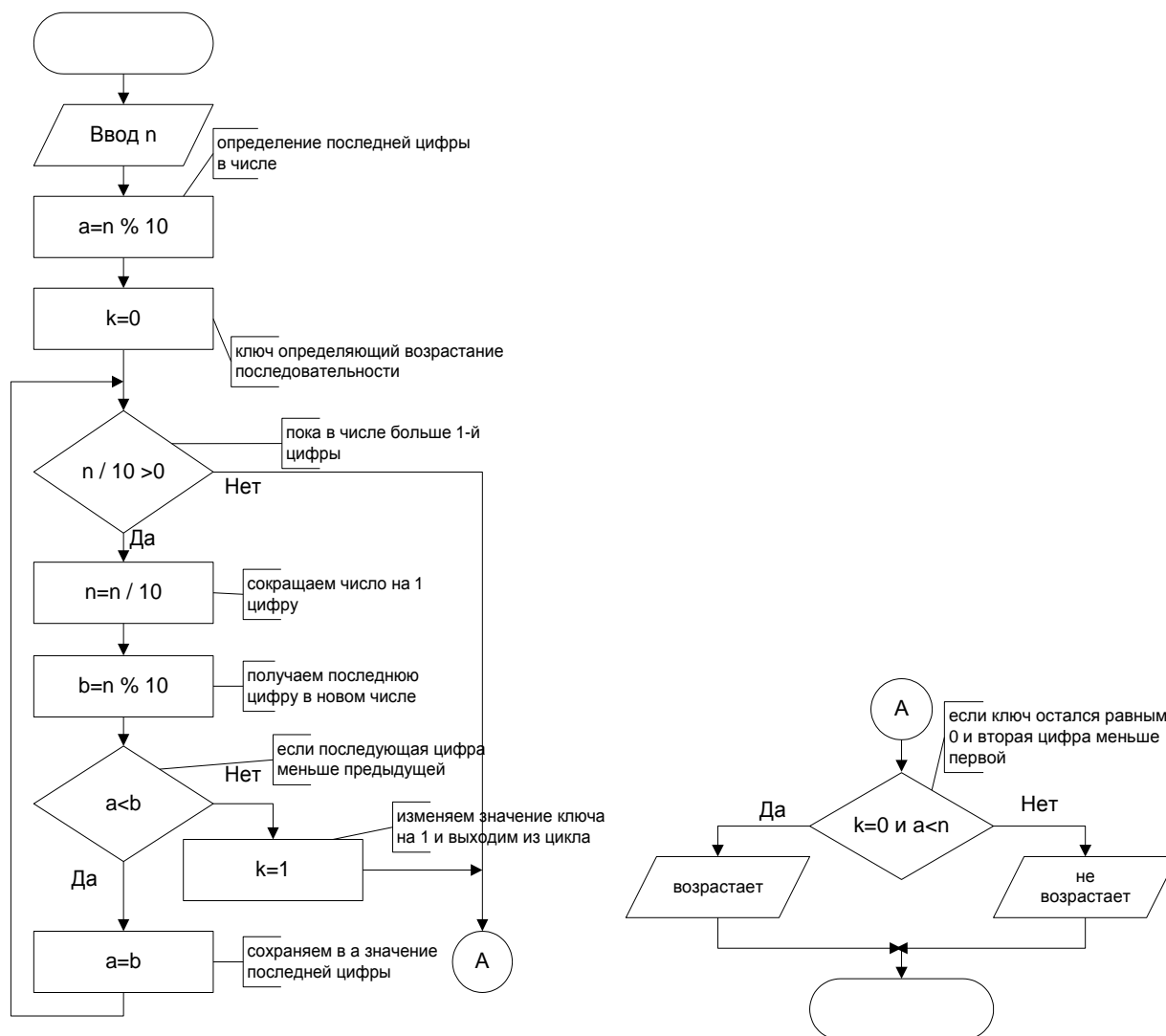
Результат: ключ k – равен нулю, если последовательность цифр возрастает, единице, если не возрастает.

Для сокращения числа на 1 цифру делим число на 10, для получения цифр в числе вычисляем остаток от деления на 10. Чтобы сравнивать две цифры используем переменные a и b . a – правая цифра в паре, b – левая цифра в паре.

Тестовый пример:

при $N=35679$ вывод 'возрастает';

при $N=35479$ вывод 'не возрастает'.



Пример 22.

Даны натуральное число n , символы. Известно, что символ s_1 отличен от восклицательного знака и что среди s_2, s_3, \dots есть по крайней мере один восклицательный знак. Пусть s_1, s_2, \dots, s_n – символы данной последовательности, предшествующие первому восклицательному знаку (n заранее неизвестно). Выяснить, верно ли, что среди символов имеются все буквы, входящие в слово *дом*.

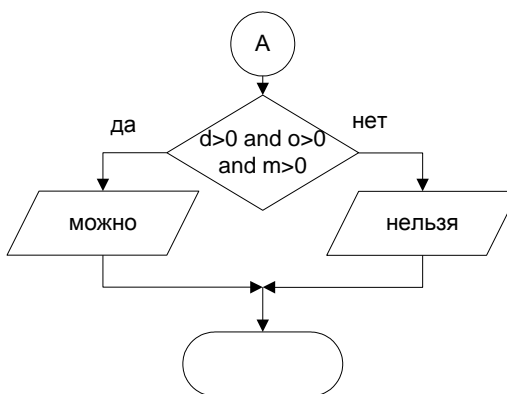
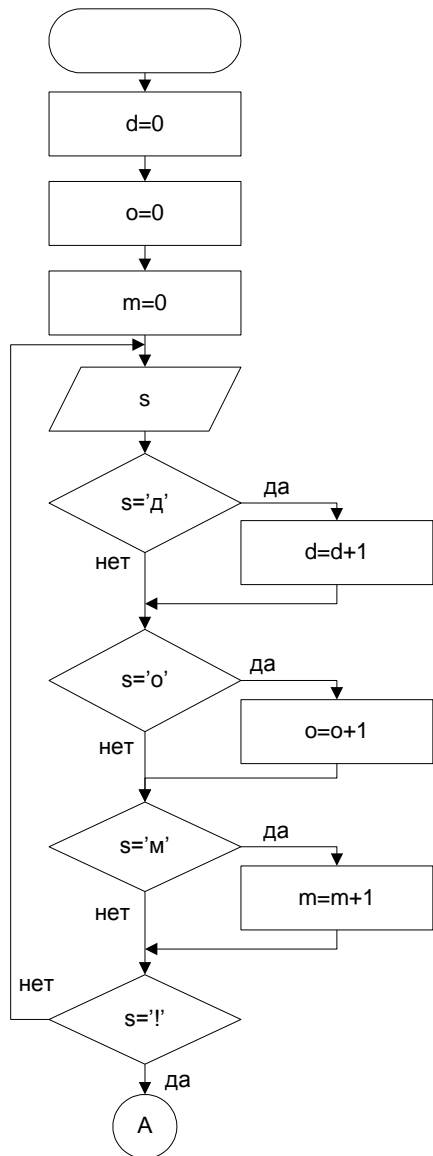
Исходные данные: элементы последовательности s – символьный тип.

Результат: количество букв «о» o – целый тип, количество букв «д» d – целый тип, количество букв «м» m – целый тип. Если каждое из этих значений больше 0 – из символов последовательности можно составить слово «дом».

Тестовый пример:

при вводе последовательности ‘имркпоовдя!’ – вывод ‘можно’

при вводе последовательности ‘имркпгрпоовя!’ – вывод ‘нельзя’



Задание 2 Написать и отладить программу для примеров 21 и 22.

Контрольные вопросы.

1. Когда надо использовать цикл «до», а когда цикл «пока».
2. К какому виду цикла относится цикл со счетчиком.
3. Какой из 3 циклов является универсальным.
4. Укажите, в каких ситуациях вместо цикла со счетчиком приходится использовать цикл while.
5. Всегда ли цикл while можно заменить циклом do.
6. Какой оператор надо использовать, если надо досрочно выйти из цикла.

7. Что означает условие выхода из цикла в примере 15.
8. Что такое наибольший общий делитель и каком алгоритм его нахождения.
9. Как можно сократить пример 20.

Индивидуальные задания

1. Спортсмен в первый день пробежал 10 км. Каждый следующий день он увеличивал дневную норму на 10% от результата предыдущего дня. Найти через сколько дней спортсмен пробежит суммарный путь более 100 км?

2. Сколько чисел нужно взять в последовательности $\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots$, чтобы получить число, больше чем N?

3. Шары расположены в форме треугольника так, что в первом ряду находится один шар, во втором – два, в третьем – три и так далее. Сколько рядов удастся построить, если имеется N шаров?

4. Вывести значения K, для которых $Z=U+M-20K+T$ больше 0,

$$\text{где } K = \begin{cases} 2U, & \text{если } M < 2 \\ U, & \text{если } 2 \leq M \leq 4 \\ 1-M, & \text{если } M > 4 \end{cases}$$

и $U=3M+1$,

M изменяется от 0 до 6 с шагом 1.5, T – произвольное число.

5. Дано натуральное n. Выяснить, входит ли цифра 3 в запись числа n2.

6. Дано натуральное n. Чему равна сумма его цифр?

7. Даны действительные числа x, y ($x > 1$, $y > 1$). Получить целое число k, удовлетворяющее условию $y^{k-1} \leq x < y^k$

8. Даны натуральное число n, символы. Известно, что символ s_1 отличен от восклицательного знака и что среди s_2, s_3, \dots есть по крайней мере один восклицательный знак. Пусть – символы данной последовательности, предшествующие первому восклицательному знаку (n заранее неизвестно). Определить количество пробелов среди s_1, \dots, s_n .

9. Даны натуральное число n, символы. Известно, что символ s_1 отличен от восклицательного знака и что среди s_2, s_3, \dots есть по крайней мере один восклицательный знак. Пусть – символы данной последовательности, предшествующие первому восклицательному знаку (n заранее неизвестно). Выяснить, имеется ли среди s_1, \dots, s_n пара соседствующих одинаковых символов.

10. Даны натуральное число n, символы. Известно, что символ s_1 отличен от восклицательного знака и что среди s_2, s_3, \dots есть по крайней мере один восклицательный знак. Пусть – символы данной последовательности, предшествующие первому восклицательному знаку (n заранее неизвестно). Выяснить, верно ли, что в последовательности имеются пять идущих подряд букв e.

11. Дано натуральное число N. Если число содержит 5 цифр, то получить новое число M, которое образуется путем исключения средней цифры исходного числа. Если количество цифр не 5, то $M=N$.

12. Среди всех n -значных чисел указать те, сумма цифр которых равна заданному числу k .

Вычисления с точностью*

Теория

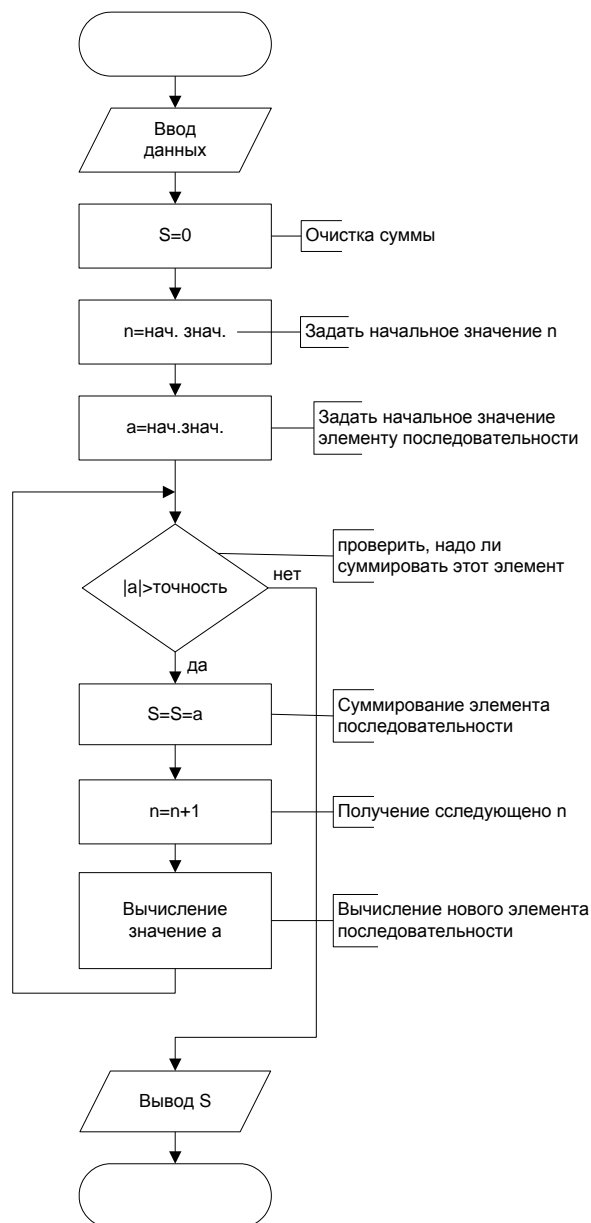
Число x называется пределом числовой последовательности $\{a_1, a_2, \dots, a_n\}$, если для любого сколь угодно малого ε можно указать такое достаточно большое положительное число N , что для всех $n > N$ выполняется неравенство $|a_n - x| < \varepsilon$.

Многие из математических величин или значений функций могут быть выражены как сумму таких бесконечных последовательностей. Например функции $\sin(x)$ и $\cos(x)$.

$$\sin(x) = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots + (-1)^n \cdot \frac{x^{2n+1}}{(2n+1)!} + \dots$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots + (-1)^n \cdot \frac{x^{2n}}{(2n)!} + \dots$$

Чем больше членов ряда участвуют в вычислении суммы, тем более точным получается результат. Разность между суммой ряда и суммой бесконечного ряда называется погрешностью сложения. Часто оценивают n -ый член, если он достаточно мал, т.е. меньше некоторого числа ε , которое часть называют точностью, считается, что найденная сумма достаточно хорошо приближается к действительному значению суммы и следующие слагаемые можно не учитывать.



Приведем блок-схему алгоритма вычисления суммы с заданной точностью ε .

Примеры

Пример 23.

Вычислить бесконечную сумму с заданной точностью ε ($\varepsilon > 0$).

$$\sum_{i=0}^{\infty} \frac{1}{4^i + 5^{i+2}}$$

Исходные данные:

точность ε
вещественный тип,
член последовательности
 a – вещественный тип.

Результат: сумма S – вещественный тип.

Тестовый пример:

при $\varepsilon = 10^{-4}$, $S = 0.0482$.

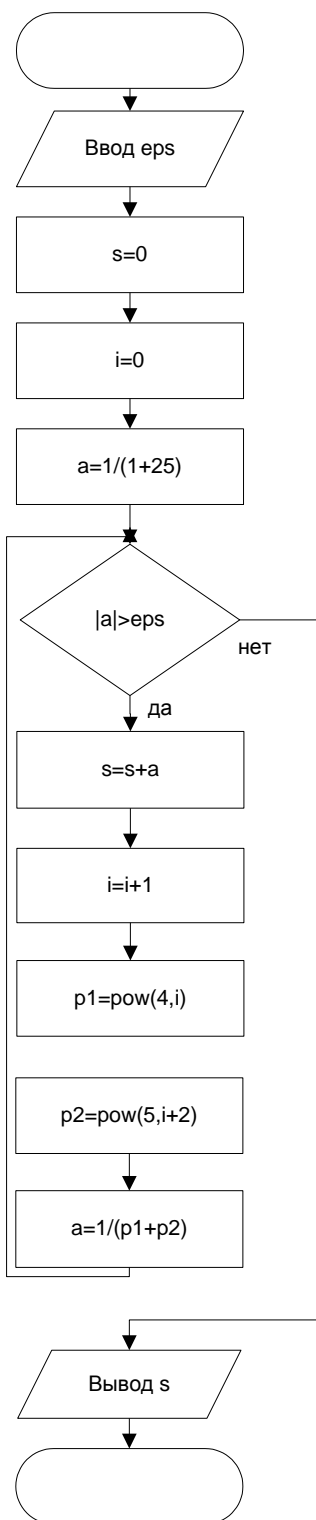
При вычислении суммы бесконечного ряда, члены ряда с увеличением номера стремятся к нулю.

Это происходит потому, что значение знаменателя быстро растет и в конце концов достигает очень большого значения, что приводит к ошибке переполнения. Чем выше точность, тем легче получить такую ошибку. В некоторых ситуациях этого можно избежать, если использовать рекуррентную формулу, т.е. выразить новый член ряда через предыдущий.

Рассмотрим примеры построения рекуррентной формулы.

Пример 24.

Даны действительные числа x , ε ($x \neq 0$, $\varepsilon > 0$). Вычислить с точностью ε :



```
#include <iostream>
#include <conio.h>
#include <math.h>
using namespace std;
void main()
{ int i;
  double a,eps,p1,p2,s=0;
  cout<<"eps="; cin>>eps;
  i=0;
  a=1.0/(1+25);

  while (fabs(a)>eps)
  {
    s=s+a;

    i++;

    p1=pow(4.0,i);

    p2=pow(5.0,i+2);

    a=1/(p1+p2);
  }
  cout<<"s= "<<s<<"i= "<<i<<endl;
  _getch();
}
```

$$\sum_{k=0}^{\infty} \frac{x^{2k}}{2^k k!}$$

Попробуем выразить формулу для a_{k+1} через a_k .

$$a_k = \frac{x^{2k}}{2^k k!}$$

$$a_{k+1} = \frac{x^{2(k+1)}}{2^{k+1} (k+1)!} = \frac{x^{2k} \cdot x^2}{2^k \cdot 2 \cdot k! \cdot (k+1)} = \frac{x^{2k}}{2^k k!} \cdot \frac{x^2}{2 \cdot (k+1)} = a_k \cdot \frac{x^2}{2(k+1)}$$

Таким образом:

$$a_{k+1} = \frac{x^2}{2(k+1)} a_k$$

Исходные данные:

x – вещественный тип,
 eps – вещественный тип,
член последовательности a – вещественный тип.

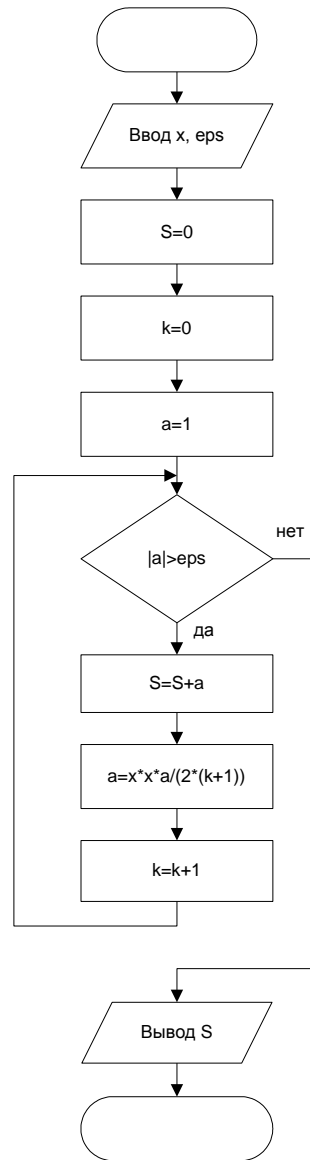
Результат:

сумма S – вещественный тип.

Тестовый пример:

при $eps=10^{-4}$, $x=1$, $S=1.6488$.

Следует обратить внимание, что так как рекуррентная формула составлена для a_{k+1} , увеличение k на единицу выполняется после вычисления a .



Рассмотрит еще пример построения рекуррентной формулы:

$$a_k = \frac{(-1)^k x^{4k+3}}{(2k+1)!(4k+3)}$$

$$a_{k+1} = \frac{(-1)^{k+1} x^{4(k+1)+3}}{(2(k+1)+1)!(4(k+1)+3)} = \frac{(-1)^k \cdot (-1) \cdot x^{4k+4+3}}{(2k+3)!(4k+7)} = \frac{(-1)^k \cdot (-1) \cdot x^{4k+3} \cdot x^4}{(2k+1)!(2k+2) \cdot (2k+3) \cdot (4k+7)} =$$

$$= \frac{(-1)^k \cdot x^{4k+3}}{(2k+1)!} \cdot \frac{(4k+3)}{(4k+3)} \cdot \frac{(-1) \cdot x^4}{(2k+2) \cdot (2k+3) \cdot (4k+7)} = a_k \cdot \frac{(-1) \cdot x^4 \cdot (4k+3)}{(2k+2)(2k+3)(4k+7)}$$

В некоторых числовых последовательностях требуется получать элементы до тех пор, пока разность между элементами не достигнет заданной точности: $|a_n - a_{n-1}| < \epsilon$. В этом случае надо сохранять в памяти два элемента последовательности.

Пример 25.

Дано действительное число ϵ ($\epsilon > 0$). Последовательность a_1, a_2, \dots образована по следующему закону:

$$a_n = \left(1 - \frac{1}{2}\right) \cdot \left(1 - \frac{1}{3}\right) \cdot \dots \cdot \left(1 - \frac{1}{n+1}\right)$$

Найти первый член a_n ($n \geq 2$), для которого выполняется условие $|a_n - a_{n-1}| < \epsilon$.

Исходные данные:

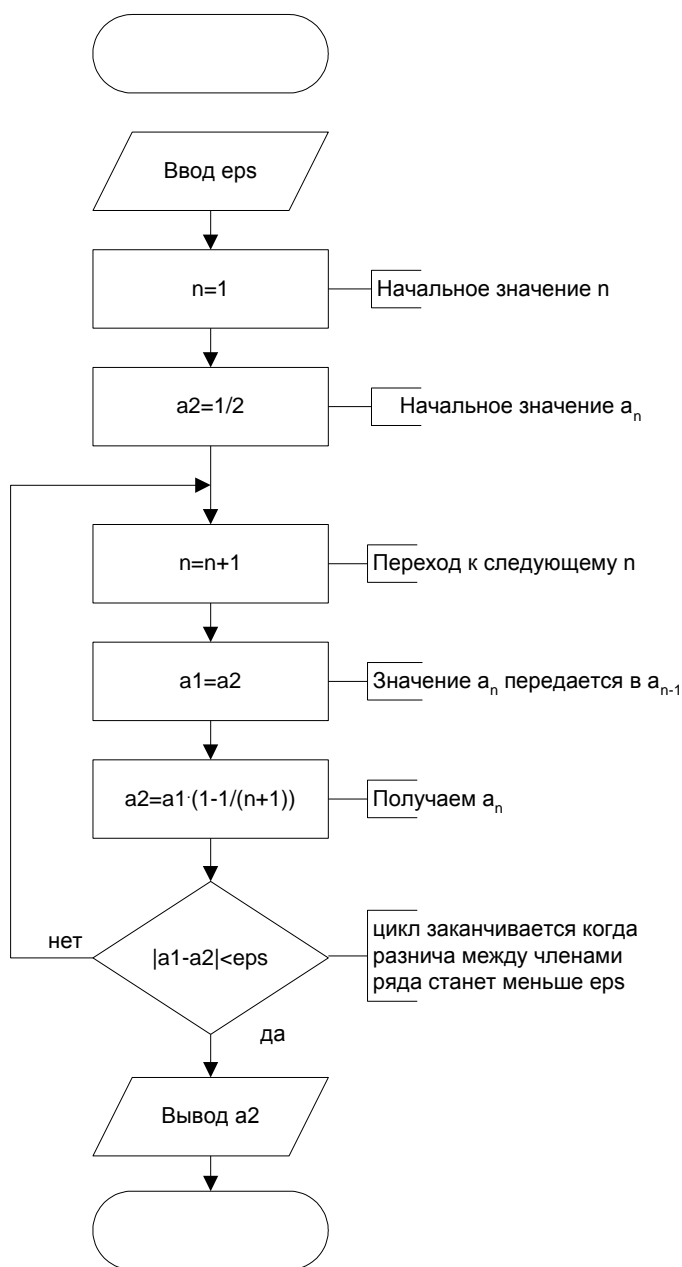
ϵ ps – вещественный тип,
элемент a_{n-1} $a1$ – вещественный тип,
элемент a_n $a2$ вещественный тип

Результат:

элемент $a2$ - вещественный тип.

Тестовый пример:

при ϵ ps= 10^{-4} , $a2=0.14$.



Задание 3 Написать и отладить программу для примера 23.

Контрольные вопросы

1. Почему при вычислении суммы бесконечной последовательности можно ограничить количество членов.
2. Почему в проверке на достижение точности член ряда указан по модулю.
3. В какой ситуации при вычислении с заданной точностью, несмотря на увеличение члена ряда с ростом n , можно завершить вычисления в цикле.
4. Почему рекомендуется получать рекуррентное соотношение для вычисления суммы ряда, который содержит факториал и возведение в целую степень.
5. Можно ли для примера 1 построить рекуррентное соотношение.

6. Если в примере 2 условие выхода из цикла будет не значение элемента меньше ε , а значение разности меньше ε . Количество слагаемых будет больше или меньше.

Индивидуальные задания

В приведенных вариантах заданий вычислить значения функций с помощью бесконечного ряда и с помощью соответствующих математических функций. Сравнить результаты.

Вариант	Задание
1	$\ln \frac{x+1}{x-1} = 2 \sum_{n=0}^{\infty} \frac{1}{(2n+1) \cdot x^{2n+1}} = 2 \left(\frac{1}{x} + \frac{1}{3x^3} + \frac{1}{5x^5} + \dots \right), \text{ при } x < \infty$
2	$e^{-x} = \sum_{n=0}^{\infty} \frac{(-1)^n x^n}{n!} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \dots, \text{ при } -1 \leq x \leq 1$
3	$\ln(x+1) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{n+1}}{n+1} = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} - \dots, \text{ при } -1 \leq x \leq 1$
4	$\ln \frac{1+x}{1-x} = 2 \sum_{n=0}^{\infty} \frac{x^{2n+1}}{2n+1} = 2 \left(x + \frac{x^3}{3} + \frac{x^5}{5} + \dots \right), \text{ при } x < 1$
5	$\ln(1-x) = - \sum_{n=1}^{\infty} \frac{x^n}{n} = - \left(x + \frac{x^2}{2} + \frac{x^4}{4} + \dots \right), \text{ при } -1 \leq x < 1$
6	$\text{arccctg}(x) = \frac{\pi}{2} + \sum_{n=0}^{\infty} \frac{(-1)^{n+1} x^{2n+1}}{2n+1} = \frac{\pi}{2} - x + \frac{x^3}{3} - \frac{x^5}{5} - \dots, \text{ при } x \leq 1$
7	$\text{arctg}(x) = \frac{\pi}{2} + \sum_{n=0}^{\infty} \frac{(-1)^{n+1}}{(2n+1)x^{2n+1}} = \frac{\pi}{2} - \frac{1}{x} + \frac{1}{3x^3} - \frac{1}{5x^5} \dots, \text{ при } x > 1$
8	$\text{arctg}(x) = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^{2n+1}}{2n+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots, \text{ при } x \leq 1$
9	$e^{-x^2} = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^{2n}}{n!} = 1 - x^2 + \frac{x^4}{2!} - \frac{x^6}{3!} + \frac{x^8}{4!} - \dots, \text{ при } x < \infty$
10	$\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!} = 1 - \frac{x^2}{(2)!} + \frac{x^4}{(4)!} - \frac{x^6}{(6)!} + \dots, \text{ при } x < \infty$
11	$\frac{\sin(x)}{x} = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n+1)!} = 1 - \frac{x^2}{(3)!} + \frac{x^4}{(5)!} - \frac{x^6}{(7)!} + \dots, \text{ при } x < \infty$
12	$\ln(x) = 2 \cdot \sum_{n=0}^{\infty} \frac{(x-1)^{2n+1}}{(2n+1) \cdot (x+1)^{2n+1}} = 2 \left(\frac{x-1}{x+1} + \frac{(x-1)^3}{3(x+1)^3} + \frac{(x-1)^5}{5(x+1)^5} + \dots \right), \text{ при } x > 0$

Тема 4 Вспомогательные алгоритмы

Процесс решения сложной задачи часть сводится к решению нескольких более простых подзадач. Соответственно процесс разработки сложного алгоритма разбивается на этапы составления отдельных алгоритмов, которые называются *вспомогательными*. Каждый вспомогательный алгоритм описывает решение какой-либо подзадачи.

При разработке сложных алгоритмов используется метод последовательной детализации, который состоит в следующем. Сначала алгоритм формулируется в крупных блоках, которые еще нельзя перевести на язык операторов и записываются как вызовы вспомогательных алгоритмов. Затем происходит детализации, и все вспомогательные алгоритмы подробно расписываются для перевода на язык команд. Вспомогательные алгоритмы на языке программирования реализуются через подпрограммы.

Программы, которые неразделимы на отдельные структурные элементы, называются *монолитными*. Большие монолитные программы сложны для разработки, отладки и сопровождения. Минимальным автономным элементом монолитной программы является оператор.

Естественно стремление разбить программу на более крупные, чем оператор, компоненты. Роль таких компонентов выполняют подпрограммы. Программы, содержащие подпрограммы, называются *модульными*. Подпрограммы, в свою очередь, могут вызывать подпрограммы более низкого уровня и т.д. Таким образом, каждая модульная программа имеет иерархическую структуру.

Использование аппарата подпрограмм позволяет сократить объем и улучшить структуру программы с точки зрения наглядности и читаемости, уменьшить вероятность ошибок и облегчить процесс отладки программы. Разложение программы на взаимосвязанные, но замкнутые и логически завершенные компоненты дает возможность выполнять разработку отдельных подпрограмм разным программистам и более или менее независимо друг от друга. Кроме того, подпрограмма может быть рассмотрена как отдельная единица (со своими входными и выходными данными), что позволяет использовать ее в общем иерархическом подходе при конструировании алгоритма и программы по принципам нисходящего проектирования.

В языке C++ подпрограммы реализуются в виде функций и void функций. Существуют встроенные (стандартные) функции, которые являются частью языка и могут вызываться по имени без предварительного описания.

Программист имеет возможность определять свои собственные функции. Это можно делать либо в том же файле, в котором хранится основная часть программы, либо в отдельном файле, с тем чтобы эту функцию можно было использовать и в других программах. И в том и в другом случае определение будет одним и тем же, но пока предположим, что функция определяется в том же файле, в котором содержится функция main программы.

Описание функции состоит из двух частей, которые называются *прототипом функции* (function prototype) и *определением функции* (function

definition). **Прототип функции** описывает, как эта функция должна вызываться. В C++ перед вызовом функции в коде должно быть помещено либо полное определение функции, либо ее прототип.

Прототип функции предоставляет всю информацию, необходимую для того, чтобы вызвать эту функцию. В коде прототип должен располагаться до вызова функции, если только она не была определена до этого. Обычно прототипы функций размещают перед функцией `main` программы.

В определении функции описано, как именно эта функция должна вычислять возвращаемое значение. Если представить себе функцию как маленькую программу внутри вашей большой программы, то определение функции — это ее код. Синтаксис определения функции очень похож на синтаксис основной части программы. Определение функции состоит из *заголовка функции*, вслед за которым следует *тело функции*. Заголовок функции записывается так же, как и ее прототип, но в конце заголовка, в отличие от прототипа, *не* ставится точка с запятой. Это приводит к тому, что заголовок повторяется, но в этом нет ничего страшного.

Хотя прототип функции и сообщает всю информацию, необходимую при ее вызове, он не сообщает, какое именно значение эта функция возвращает. Возвращаемое функцией значение определяется инструкциями *тела функции*. **Тело функции** следует после ее заголовка и завершает определение функции, состоящее из объявлений и выполняемых инструкций, заключенных в фигурные скобки. Таким образом, тело функции выглядит точно так же, как и та часть программы, которая следует после слова `main`. При вызове функции вместо ее формальных параметров подставляются реальные аргументы, после чего выполняются инструкции тела этой функции. Возвращаемое функцией значение определяется в момент вычисления инструкции `return`. Инструкция `return` состоит из ключевого слова `return`, за которым следует выражение.

Чтобы лучше разобраться в использовании функций, запомните три положения..

- Определение функции похоже на небольшую программу, а ее вызов подобен запуску этой "маленькой" программы.

- В функции вместо ввода, поступающего в программу с помощью потока `cin`, используются формальные параметры. Аргументы функции (**фактические параметры**), которые подставляются вместо формальных параметров, служат для нее входной информацией.

- Функция ничего не выводит на экран, однако возвращает некоторую информацию вызвавшей ее программе. Для этого она использует инструкцию `return`..

Размещение определения функции

Мы обсудили, где, как правило, размещаются прототипы и определения функций. В обычной ситуации указанное место является для них самым лучшим. Однако компилятор будет воспринимать и такие программы, в которых эти элементы кода размещаются в некоторых других местах. Сформулируем эти правила более точно: до вызова каждой функции компилятор должен встретить либо ее прототип, либо определение. Например, если в программе все определения

функций разместить перед основной частью программы, то прототипы функций уже не нужны. Знание этого общего правила поможет понять программы на C++, приведенные в других книгах, но все же лучше придерживаться образца, по которому написаны программы, представленные в этих работах. Используемого нами стиля придерживается большая часть программистов, которые работают на C++.

Процедурная абстракция

Применительно к определению функции принцип процедурной абстракции означает, что функцию нужно писать так, чтобы ее можно было использовать подобно *черному ящику*. Это значит, что программисту, который захочет использовать эту функцию, не нужно будет заглядывать в ее тело, чтобы понять, как она работает. Прототип функции и прилегающий к ней комментарий должны предоставлять всю необходимую для использования этой функции информацию. Чтобы определение функции удовлетворяло этому требованию, нужно строго придерживаться приведенных ниже правил.

Как писать определение функции в виде черного ящика

- Комментарии к прототипу должны сообщать программисту все, что необходимо знать об аргументах функции, и описывать значение, возвращаемое функцией при ее вызове с этими аргументами.
- Все переменные, используемые в теле функции, должны быть объявлены в теле функции (формальные параметры объявлять не нужно, потому что они перечислены в ее прототипе).

Принцип процедурной абстракции гласит, что функция должна быть оформлена в виде самодостаточного модуля, который разрабатывается отдельно от остальной программы. При работе над большими программными проектами функции могут разрабатываться разными программистами, которые должны подбирать для формальных параметров имена, наиболее подходящие по смыслу. Аргументам (фактическим параметрам), которые подставляются вместо формальных параметров (и зачастую не тем программистом, который писал определение функции), тоже следует давать имена со смыслом. При этом может оказаться, что некоторые аргументы имеют одинаковые имена с формальными параметрами. Это был бы идеальный вариант. Однако независимо от того, какие имена выбраны для переменных, которые будут использованы в качестве аргументов, эти имена не перепутаются с именами формальных параметров. Ведь функции используют только значения аргументов, не обращая внимания на их имена.

Лабораторная 5

Функции

Теория

Функция – это подпрограмма, вычисляющая и возвращающая некоторое значение.

Функция пользователя используется только тогда, когда требуется вычислить единственное значение. Функция возвращает результат в точку своего вызова. Поэтому функция является частью какого-нибудь выражения.

Синтаксис функции, которая возвращает значение:

Прототип функции:

Возвратаемый_тип Имя_функции {Список_параметров}; Комментарии_к_прототипу

Определение функции:

// Следующая строка представляет собой заголовок функции

Возвращаемый_тип Имя_функции(Список_параметров)

{ // Начало тела функции

Объявление_1

Объявление_2

Объявление_N

Исполняемая_инструкция_1 // Одна из этих

Исполняемая_инструкция_2 // инструкций

. . . // должна быть

Исполняемая_инструкция_N // инструкцией return

} // Конец тела функции

Поскольку функция не возвращает значения, пока не выполнит инструкцию `return`, в ее теле должна находиться по крайней мере одна такая инструкция (в теле функции может находиться и несколько инструкций `return`).

Для определения функции допустим любой образец размещения с помощью пробелов и символов новой строки. Однако лучше придерживаться тех же правил использования отступов от начала строки и размещения элементов кода, которые приняты для основной части программы. В частности открывающая (`{`) и закрывающая (`}`) скобки, которыми обозначаются границы тела функции, размещаются на отдельных строках. Тем самым тело функции отделяется от остального кода.

Данные, описанные в подпрограмме, действительны только в пределах данной пользовательской подпрограммы. Они называются **локальные параметры**. Данные, описанные в основной программе, называются глобальными и их можно так же использовать в подпрограмме. Если имя глобальной переменной совпадает с именем локальной, внутри подпрограммы эта переменная интерпретируется как локальная.

Параметры функции позволяют вычислять функцию с различными начальными данными. Параметры указываются в заголовке функции и называются формальными параметрами. По сути, они являются переменными, локальными для определения данной функции; их можно рассматривать как локальные переменные, которые объявляются при определении функции.

Описание формальных параметров имеет вид: *тип имя параметра*. Описание параметров разделяются запятой.

При вызове функции список формальных параметров заменяется фактическими параметрами. Между формальными и фактическими параметрами должны выполняться следующие правила соответствия:

- по количеству (количество формальных и фактических параметров одинаково);

- по последовательности (первому формальному параметру соответствует первый фактический параметр, второй – второму и т.д.);
- по типам (типы соответствующих формальных и фактических параметров должны совпадать).

Фактические параметры-аргументы могут быть выражениями соответствующего типа.

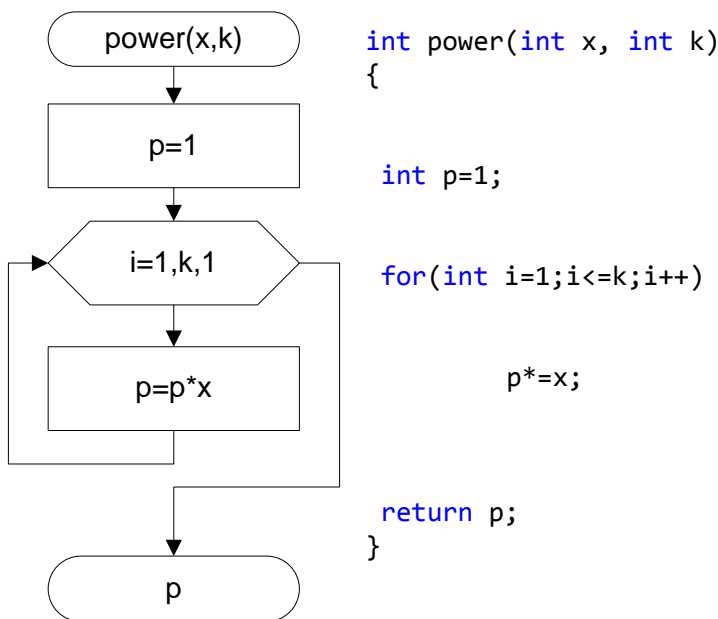
При разработке блок-схемы вспомогательный алгоритм разрабатывается как отдельная блок-схема, но в блоке «Начало» указывается заголовок подпрограммы.

Примеры

Пример 1.

Вычислить с заданной точностью ε .
$$\sum_{i=0}^{\infty} \frac{1}{4^i + 5^{i+2}}$$

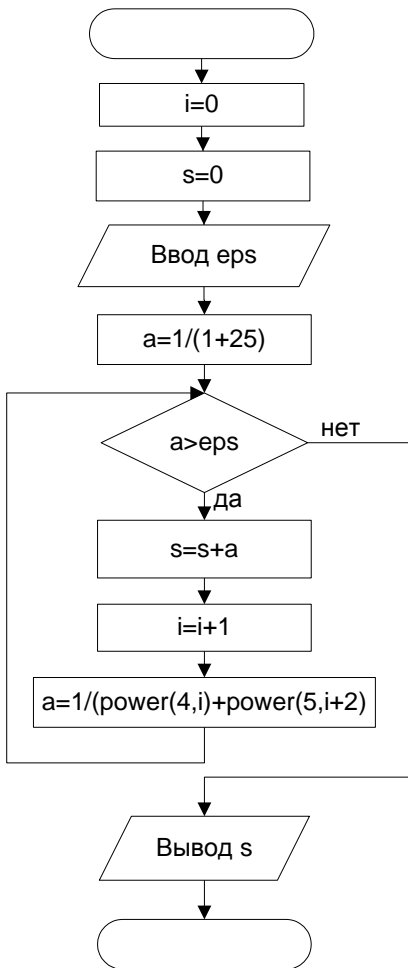
Этот пример рассматривался в предыдущей лабораторной работе. Возведение в степень в нем выполнялось с помощью функции pow, но результат функции – это вещественное число, хотя в степень возводятся целые числа и результат тоже будет целым. В этом случае можно создать функцию возводящую целые числа в целую степень.



Исходные данные: точность eps вещественный тип, член последовательности a – вещественный тип.

Результат: сумма S – вещественный тип.

Тестовый пример: при eps=10⁻⁴, S=0.0097.



```

#include <iostream>
#include <conio.h>
using namespace std;
int power(int x, int k);
void main()
{ int i=0;
  double eps,a,s=0;
  cout<<"eps="; cin>>eps;
  a=1.0/(1+25);

  while (a>eps)
    {s=s+a;
     i++;
     a=1.0/
(power(4,i)+power(5,i+2));
    }
  cout<<"s= "<<s<<endl;
  _getch();
}

int power(int x, int k)
{
  int p=1;
  for(int i=1;i<=k;i++)
    p*=x;
  return p;
}

```

Пример 2.

Даны натуральные числа n, m , целые числа $a_1, \dots, a_n, b_1, \dots, b_m, c_1, \dots, c_{10}$. Получить $w = x^2 + y^2 + z^2$, где $x = \min(a_1, \dots, a_n)$, $y = \min(b_1, \dots, b_m)$, $z = \min(c_1, \dots, c_{10})$.

a_n), $y = \min(b_1, \dots, b_m)$, $z = \min(c_1, \dots, c_{10})$.

Исходные данные: n, m – целого типа, элемент последовательности a – целого типа, элемент последовательности b целого типа, элемент последовательности c – целого типа,

Результат: w целого типа

Приведен пример блок-схемы вычисления функции нахождения минимального значения.

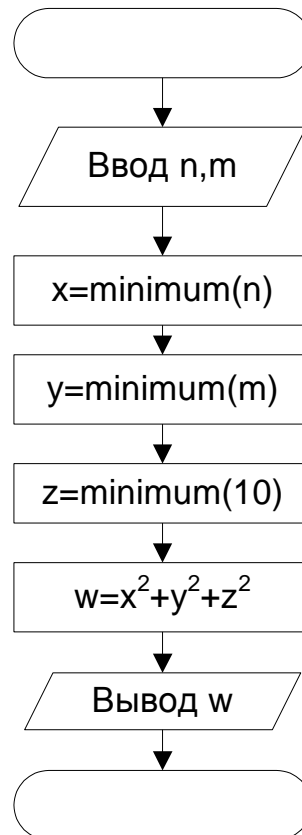
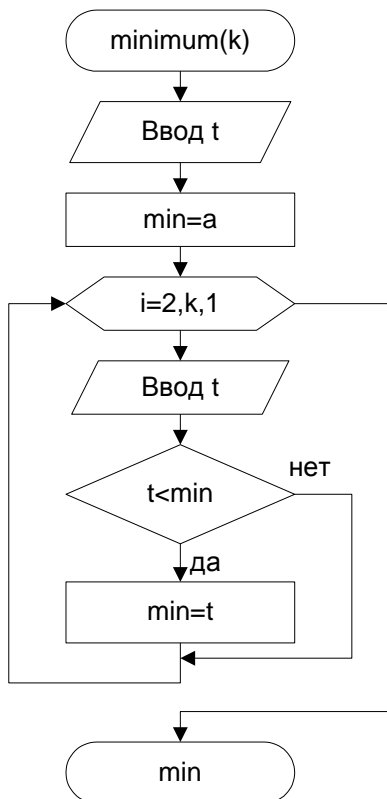
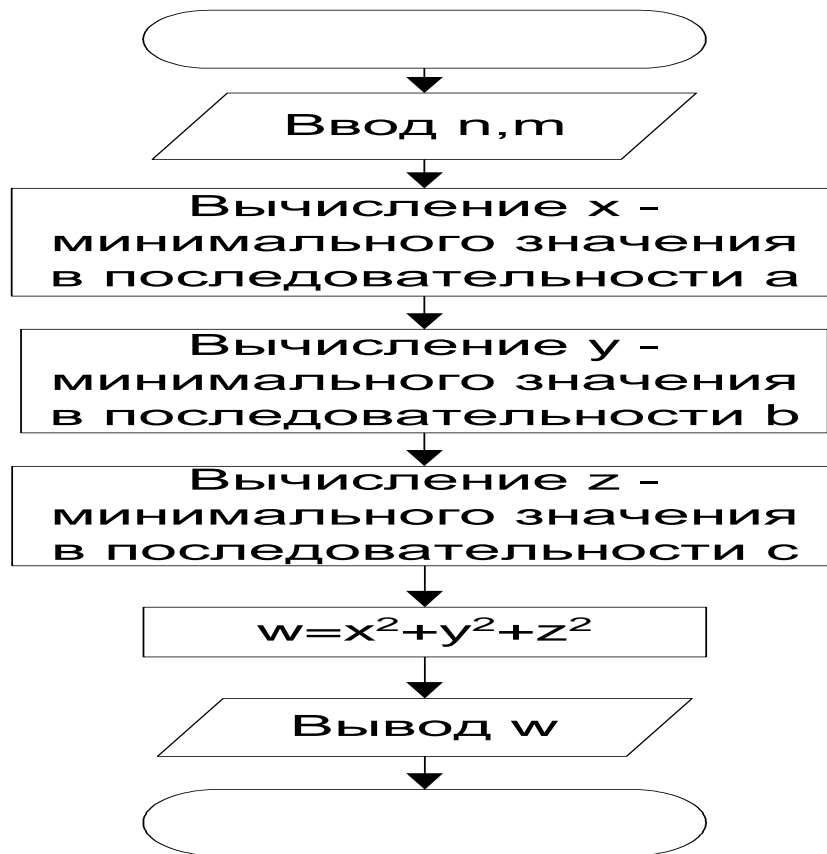
Тестовый пример:

при $n=7, m=8$, последовательность a :

5, 2, 7, 0, 6, 1, 4; последовательность b :

-1, -3, -5, -3, -6, -2, -2, -7; последовательность c

-2, 4, 6, -8, 3, 5, -3, -5, -2, 1 $w=65$.

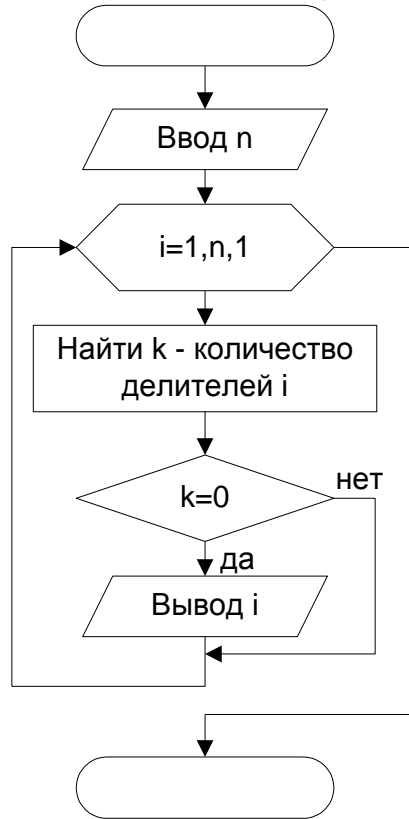


Пример 3.

Составить программу нахождения всех простых чисел не превосходящих n .

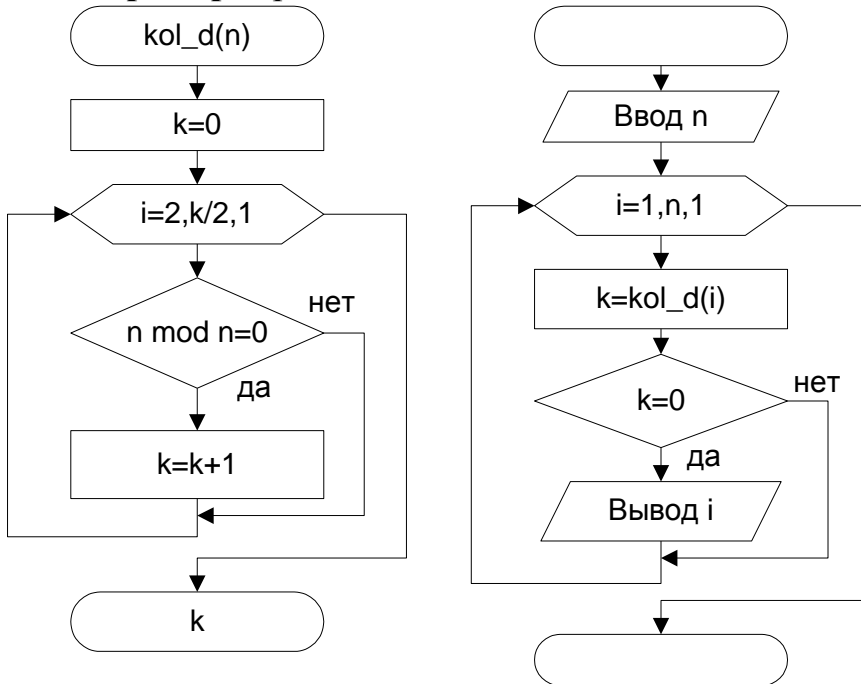
Исходные данные: n – целый тип.

Результат: вывод i числа, количество делителей у которого равно 0.



При построении укрупненной блок схемы не выявлены повторяющиеся задачи, но подсчет количества делителей у числа можно рассматривать как самостоятельную задачу. Для этой задачи можно написать отдельную функцию.

Тестовый пример: при $n=15$, вывод 1, 2, 3, 5, 7, 11, 13.



Пример 4.

Написать программу, которая проверяет, являются ли во введенном четырехзначном числе вида $abcd$, все цифры разные.

Исходные данные: четырехзначное число m .

Результаты: цифры числа m - a, b, c, d целый тип,

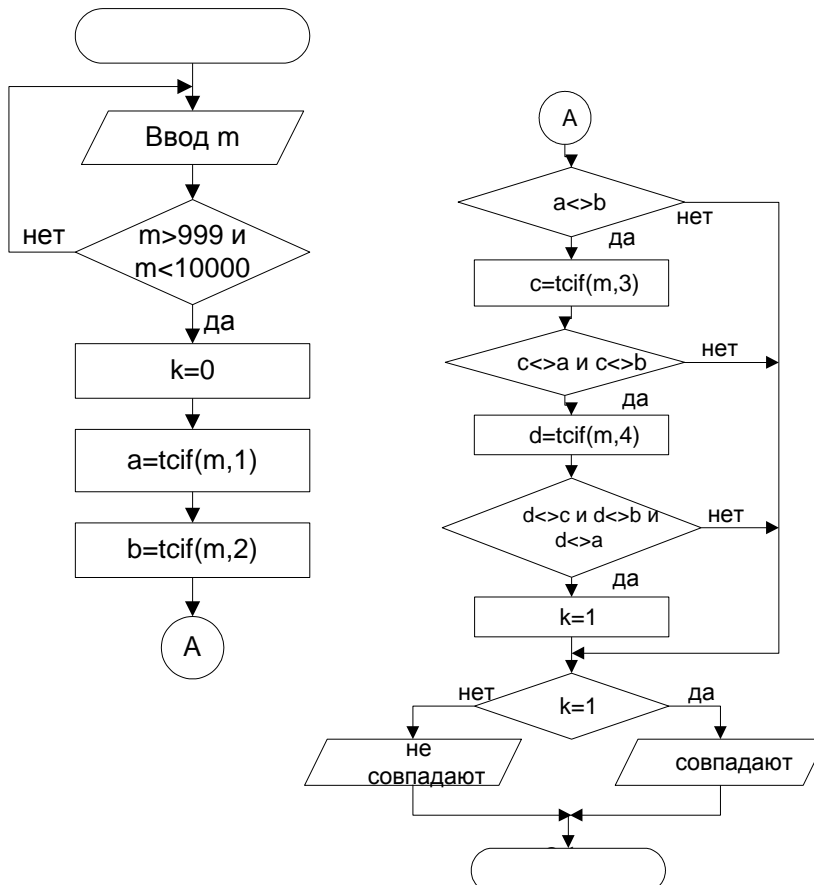
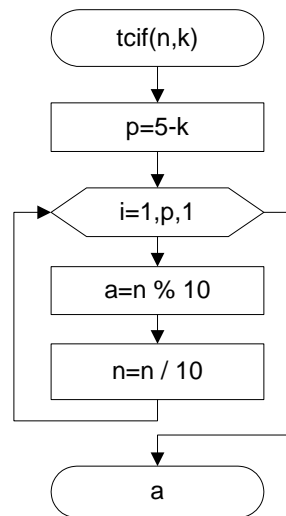
$k=0$, если есть совпадение цифр, $k=1$ если совпадения цифр нет.

Следует создать функцию, которая выделяет заданную цифру из данного числа. Эта функция работает только с 4-значным числом, поэтому в начале программу следует проверить количество цифр в числе.

Тестовый пример:

при 2467 – все цифры разные;

при 1233 –цифры совпадают.

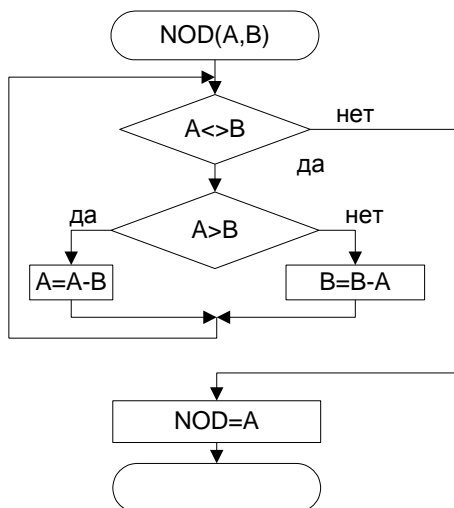


Пример 5.

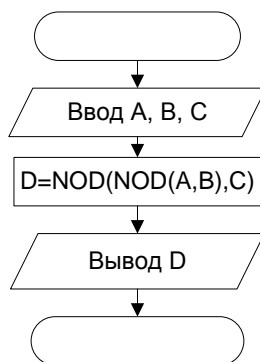
Найти наибольший общий делитель 3 чисел.

Исходные данные: A, B, C – целого типа.

Результат: наибольший общий делитель чисел A, B, C – D целого типа.



Нахождение наибольшего общего делителя (НОД) двух чисел выполняется в функции NOD. Сначала находится НОД для чисел A, B, а затем для полученного значения и C.



Задание 1 Написать и отладить программу для примера 4.

Контрольные вопросы

1. Может ли функция не иметь параметров.
2. Могут ли в качестве фактических параметров использоваться выражения.
3. Какие правила соответствия должны выполняться для формальных и фактических параметров.
4. Что будет, если имя формального параметра совпадает с глобальным параметром.
5. В Примере № 3 в основной программе используется цикл с параметром i и в подпрограмме используется цикл с параметром i. Не запутается ли программа с такими совпадениями.

6. Можно ли в теле функции использовать несколько операторов return.
 7. Можно ли из одной функции вызывать другую функцию.

Индивидуальные задания

1. Даны действительные числа s, t . Получить:

$$h(s,t) + \max(h^2(s-t, st), h^4(s-t, s+t)) + h(1,1), \text{ где}$$

$$h(a,b) = \frac{a}{1+b^2} + \frac{b}{1+a^2} - (a-b)^3$$

2. Даны действительные числа a, b , натуральное число n ($b > a$). Получить $(f_1 + \dots + f_n)h$,
 где

$$h = \frac{b-a}{n}, f_i = \frac{a + \left(i - \frac{1}{2}\right)h}{1 + \left(a + \left(i - \frac{1}{2}\right)h\right)^2}, i = 1, 2, \dots, n.$$

3. Даны натуральные числа k, l, m целые числа $x_1, \dots, x_k, y_1, \dots, y_l, z_1, \dots, z_m$.
 Получить

$$l = \begin{cases} (\max(y_1, \dots, y_l) + \max(z_1, \dots, z_m))/2 & \text{при } \max(x_1, \dots, x_k) \geq 0, \\ 1 + (\max(x_1, \dots, x_k))^3 & \text{в противном случае.} \end{cases}$$

4. Дано натуральное n . Вычислить сумму

$$\sum_{k=1}^n \frac{k!}{S_k} \text{ где } S_k = \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k+1}$$

5. Даны натуральные u, v и n . Найти

$$\sum_{k=1}^n \frac{a_k b_k}{(k+1)!},$$

где $a_1 = u, b_1 = v, a_k = 2b_{k-1} + a_{k-1}; b_k = 2a_{k-1}^2 + b_{k-1}, k = 2, 3, \dots$. Вычисление a_k и b_k и факториала выполнять с помощью функции.

6. Даны действительные числа x и ε ($x \neq 0, \varepsilon > 0$). Вычислить с точностью ε и указать количество учтенных слагаемых

$$\sum_{k=0}^{\infty} \frac{(-1)^{k+1} x^{2k-1}}{(2k-1)!(2k+1)!}$$

Возведение в степень и факториал вычислять с помощью функций

7. Дано действительное число y . Получить

$$\frac{1.7t(0.25) + 2t(1+y)}{6 - t(y^2 - 1)}, \text{ где } t(x) = \frac{\sum_{k=0}^{10} \frac{x^{2k+1}}{(2k+1)!}}{\sum_{k=0}^{10} \frac{x^{2k}}{(2k)!}}$$

8. Составить программу для нахождения чисел из интервала $[M, N]$, имеющих наибольшее количество делителей.
9. Составить программу, определяющую, в каком из данных двух чисел больше цифр.
10. Два натуральных числа называются «дружественными», если каждое из них равно сумме всех делителей (кроме его самого) другого числа (например 220 и 284). Найти все пары «дружественных» чисел, которые не больше данного числа N .
11. Натуральное число, в записи которого n цифр, называется числом Амстронга, если сумма его цифр, возведенная в степень n , равна самому числу. Найти все числа Амстронга от 1 до k .
12. Найти все натуральные числа, не превосходящие n , которые делятся на каждую из своих цифр.

Процедуры (void-функции)

Теория

В C++ подзадачи реализуются в виде функций. Функции, которые рассматривались выше, всегда возвращают одно значение. Однако бывают и другие виды подзадач, например, когда требуется вернуть из функции несколько значений либо не возвращать вовсе ничего. Функции, которые не возвращают ни одного значения, называются процедурами (в C - void-функциями). Например, одной из типичных подзадач программы является вывод результатов вычислений. В этой подзадаче осуществляется вывод на экран, но она не вычисляет никаких значений, которые бы использовались в остальной части программы. Такую разновидность подзадач можно реализовать в виде void-функций.

Процедура (void-функция) – это независимая именованная часть программы, которую после однократного описания можно многократно вызывать по имени из последующих частей программы для выполнения определенных действий.

В C++ эти void-функции определяются аналогично функциям, возвращающим значение. Между определением void-функции и определениями функций, есть только два различия. Одно из них заключается в том, что вместо указания типа возвращаемого значения в определении void-функции используется ключевое слово `void`. Тем самым компилятору сообщается, что эта функция не возвращает никаких значений. Само имя `void` говорит: "эта функция не возвращает никаких значений". Второе различие состоит в том, что в инструкции `return` не содержится выражения,

по которому должно вычисляться возвращаемое значение, потому что никакого возвращаемого значения не существует.

Синтаксис определения void-функции

Прототип void-функции:

```
void Имя_функции(Список_параметров); //Комментарий_к_прототипу
```

Определение void-функции:

```
void Имя_функции(Список_параметров) //Заголовок функции
{ //Начало тела функции
Объявление_1 // Объявления можно чередовать
Объявление_2 // с исполняемыми инструкциями
Объявление_N
Выполняемая_инструкция_1 // Можно включить одну или
Выполняемая_инструкция_2 // несколько (необязательных)
Выполняемая_инструкция_3 // инструкций return
Выполняемая_инструкция_N
} //Конец тела функции
```

Вызов процедуры (void-функции) является выполняемой инструкцией. Она состоит из имени функции и списка фактических параметров отделенных друг от друга запятыми и заключенными в круглыми скобками. Иногда нужны функции без аргументов, что вполне корректно. В этом случае в прототипе функции список формальных параметров просто отсутствует и при ее вызове не передается ни одного аргумента.

Процедуры имеют более широкий спектр применения, чем функции:

- с помощью процедур можно одновременно вычислять несколько значений;
- в процедурах можно изменять входные параметры;
- результатом процедур не обязательно является вычисление значений, это может быть выполнение каких-нибудь действий, например ввод или вывод данных.

В процедурах (void-функциях), как и в функциях, возвращающих значение, могут присутствовать инструкции return. Если функция возвращает значение, оно определяется инструкцией return. В void-функциях эта инструкция просто завершает вызов функции, т.е.означает досрочный выход из функции.

При вызове функции ее аргументы подставляются вместо формальных параметров из определения этой функции; иными словами, аргументы "подключаются" к формальным параметрам. Есть несколько механизмов реализации такого процесса подстановки. Механизм, использовавшийся в предыдущем разделе, называется механизмом *передачи параметров по значению* (call-by-value). Второй важный механизм подстановки аргументов известен под названием *передачи параметров по ссылке* (call-by-reference).

При выполнении некоторых подзадач оказывается, что механизма передачи параметров по значению недостаточно. Например, довольно распространенной задачей является вычисление в функции одной или нескольких величин.

Аргументы функции, которые использовались до сих пор, могут быть переменными, но вместо формального параметра вызова по значению подставляется только *значение* аргумента. Для функции вычисляющей несколько значений нужно, чтобы вместо вычисляемого формального параметра подставлялась сама *перемен-*

ная, а не только ее значение. Механизм передачи параметров по ссылке работает следующим образом. Для формального параметра, передаваемого по ссылке, соответствующий аргумент вызова функции должен быть переменной, и эта переменная подставляется в тело функции вместо формального параметра. Все происходит так, как если бы переменная-аргумент (а не ее значение!) буквально копировалась в тело определения функции вместо формального параметра. После этого при выполнении тела функции значение переменной-аргумента может изменяться.

Чтобы компилятор мог отличить параметр, передаваемый по ссылке, от параметра, передаваемого по значению, необходимо явно указать способ передачи по ссылке. Для этого используется знак амперсанда (&), который располагается после имени типа формального параметра, передаваемого по ссылке, как в прототипе функции, так и в заголовке ее определения.

В блок-схеме программы блок-схема процедуры, как и блок-схема функции, изображается отдельно. В блок-схеме основной программы оператор вызова подпрограммы изображается в виде блока:



Примеры

Пример 6.

Дана дробь в виде $\frac{A}{B}$ (A, B – натуральные числа). Представить эту дробь в виде несократимой дроби.

Исходные данные: A – числитель, целый тип и B – знаменатель, целый тип.

Результат: A – числитель после сокращения, B – знаменатель после сокращения.

Для сокращения дроби надо найти наибольший общий делитель (НОД) для числителя и знаменателя, а затем разделить числитель и знаменатель на это число.

Сокращение дроби запишем в отдельной void-функции SOKR. Эта void-функция в дальнейшем может быть использована в других задачах.

Данную задачу можно решить только с использованием void-функции, т.к. в результате этой подпрограммы изменяются входные параметры.

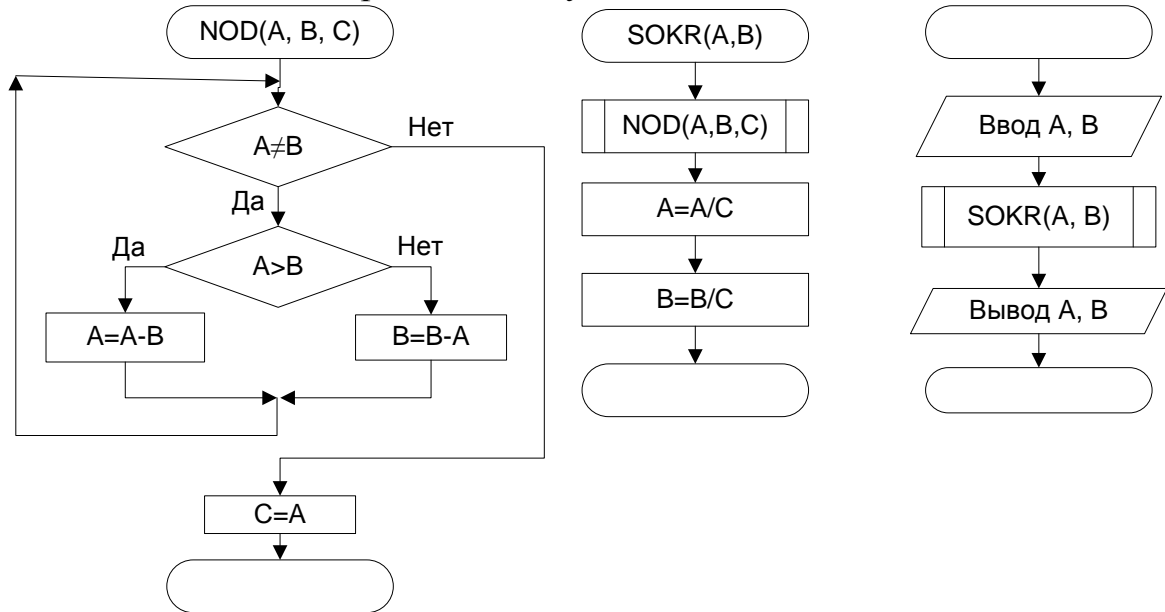
Обратите внимание, что функция NOD, которая была рассмотрена в предыдущей лабораторной работе, в данном примере представлена как void-функция.

Любая функция может быть представлена как void-функция. В этом случае, полученный результат является еще одним параметром void-функции, причем параметром, передаваемым по ссылке.

В данном примере используется две void-функции: void-функция NOD, для нахождения наибольшего общего делителя и void-функция SOKR для сокращения числителя и знаменателя. Void-функция SOKR вызывает процедуру NOD, поэтому в описании прототипов void-функция NOD должна предшествовать void-функции SOKR

Тестовый пример:

При $A=7$, $B=56$, после сокращения получаем $A=1$, $B=8$.



```

#include <iostream>
#include <conio.h>
using namespace std;
//Нахождение наибольшего общего делителя - с
void nod(int a, int b, int& c);
// сокращение дроби a/b
void socr(int& a, int& b);
void main()
{int a,b;
  cout<<"a="; cin>>a;
  cout<<"b="; cin>>b;
  socr(a,b);
  cout<<"a= "<<a<<" b= "<<b<<endl;
  _getch();
}
void nod(int a, int b, int& c)
{while (a!=b)
  {if (a>b)
    a=a-b;
   else b=b-a;
  }
  c=a;
}
void socr(int& a, int& b)
{int c;
  nod(a,b,c);
  a=a/c;
  b=b/c;
}
  
```

Пример 7.

Дана последовательность из n натуральных чисел. Для каждого числа указать его наименьший и наибольший делитель. Для простого числа – 1 и само число, для остальных это должны быть числа отличные от 1 и самого числа.

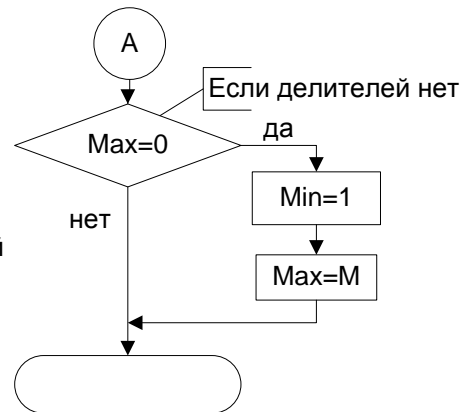
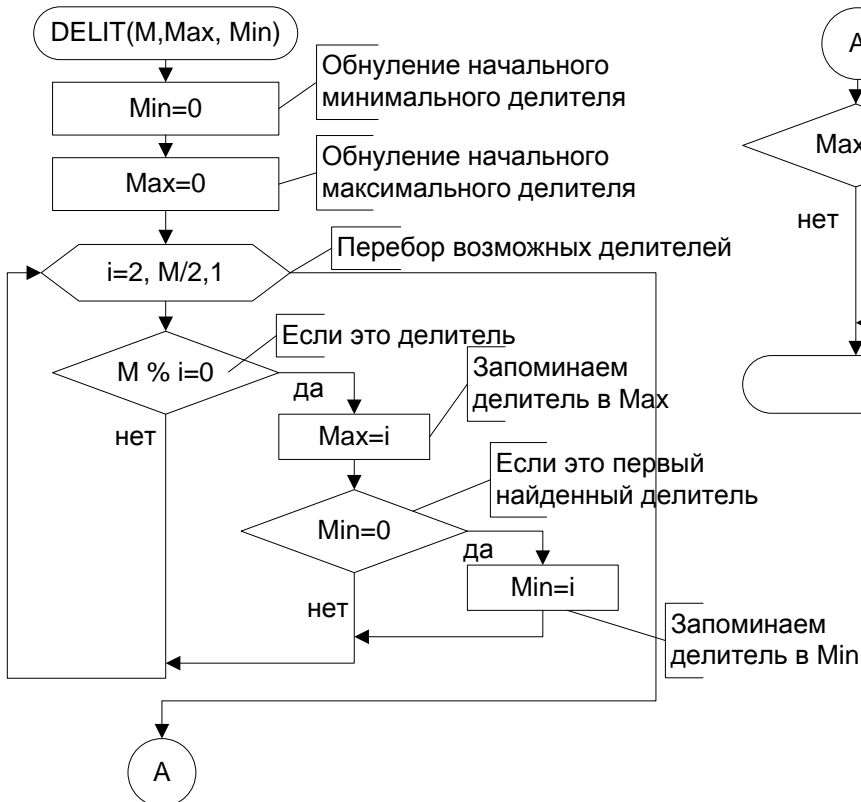
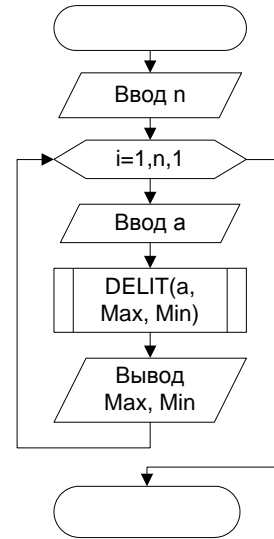
Исходные данные: n - количество элементов последовательности. a – элемент последовательности.

Результат: Max – наибольший делитель, Min – наименьший делитель каждого элемента последовательности.

Нахождение наибольшего и наименьшего делителя выполняется в процедуре DELIT. Void-функция в этом случае используется потому, что в результате получается не одно значение, а два.

Тестовый пример:

- при $n=5$ и
- $a=6, Max=3, Min=2$
- $a=7, Max=7, Min=1$
- $a=24, Max=12, Min=2$
- $a=15, Max=5, Min=3$
- $a=63, Max=7, Min=3$



```

#include <iostream>
#include <conio.h>
#include <locale>
using namespace std;
  
```

```

void delit(int m,int& min, int& max);
void main()
{int n,a,min,max;
setlocale(0, "");
cout<<"n=";cin>>n;
for(int i=1;i<=n;i++)
{cout<<"Введите число ";
cin>>a;
delit(a,min,max);
cout<<"минимальный делитель "<<min;
cout<<" максимальный делитель "<<max<<endl;
}
_getch();
}
void delit(int m,int& min, int& max)
{min=0;
max=0;
for(int i=2; i<=m/2;i++)
{
    if (m % i==0) {max=i;
                    if (min==0) min=i;
                }
}
if (max==0) {min=1;
              max=m;
            }
}

```

Пример 8.

Найти все натуральные числа, не превосходящие заданное N, которые делятся на каждую из своих цифр.

Исходные данные: N – целый тип.

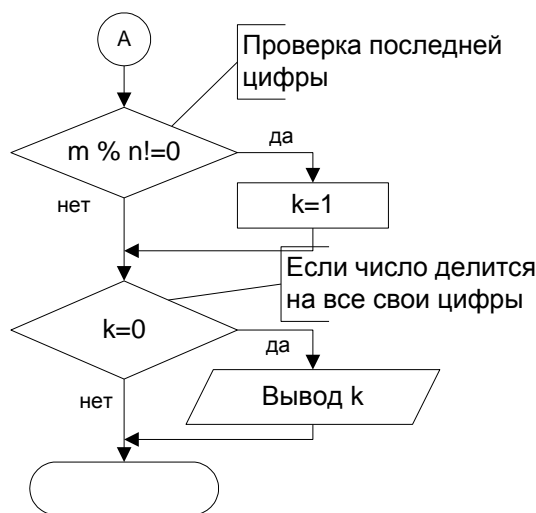
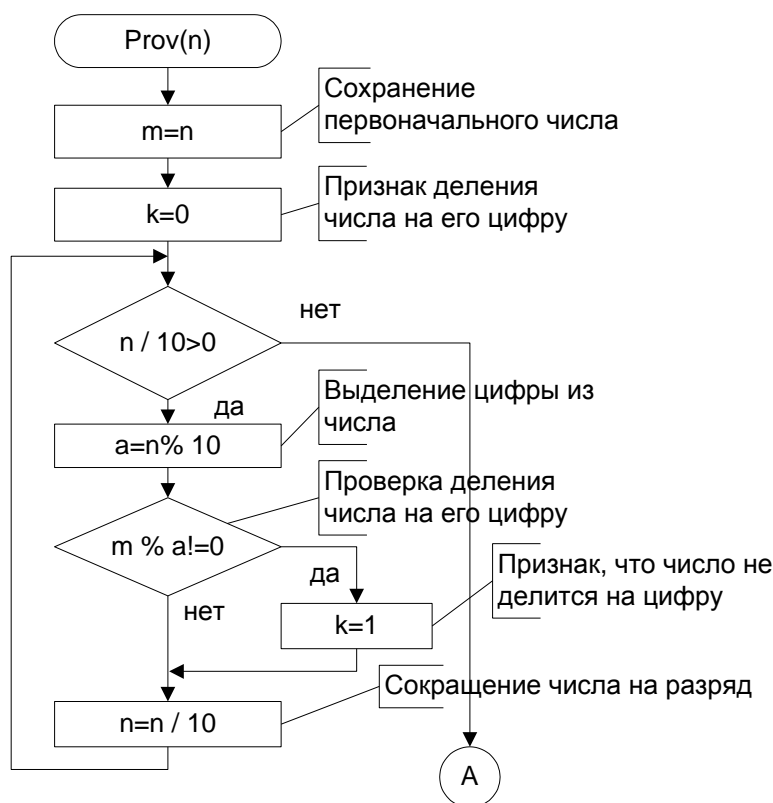
Результат: Вывод чисел, которые делятся на каждую из своих цифр.

В процедуре Proc проверяется делится ли число на все свои цифры, если делится - данное число выводится на экран. Результатом процедуры является печать чисел, удовлетворяющих заданному условию.

Тестовый пример:

При N=20 выводятся числа: 1,2,3,4,5,6,7,8,9,10,11,12,15,20

Приведен пример блок-схемы процедуры Proc, которая проверяет деление числа на свои цифры.



Задание 2 Написать и отладить программу для примера 8.

Контрольные вопросы

1. Можно ли в Примере 6 при вычислении NOD параметры A и B описать как параметры, передаваемые по ссылке.
2. Можно ли в Примере 6 поменять местами описание void-функции NOD и SOKR.
3. Перечислите преимущества void-функций перед функциями.
4. Чем параметры передаваемые по ссылке отличаются от параметров передаваемых по значению.
5. Можно ли в Примере 6 использовать не void-функцию, а функцию.
6. В каких случаях void-функцию можно заменить на функцию.
7. Можно ли в качестве фактических параметров по ссылке использовать выражения.
8. В каких ситуациях удобнее использовать функцию вместо void-функции.

Индивидуальные задания

1. Даны две дроби $\frac{A}{B}$ и $\frac{C}{D}$ (A, B, C, D – натуральные числа). Составить программу вычитания этих дробей. Результат должен быть несократимой дробью.
2. Даны две дроби $\frac{A}{B}$ и $\frac{C}{D}$ (A, B, C, D – натуральные числа). Составить программу для деления этих дробей. Результат должен быть несократимой дробью.

3. Даны натуральные n , m и последовательности вещественных чисел x_1, x_2, \dots, x_n , y_1, y_2, \dots, y_m . Найти разности максимумов и минимумов каждой последовательности.
4. Два простых числа называются «близнецами», если они отличаются друг от друга на 2 (например, 41 и 43). Напечатать все пары «близнецов» из отрезка $[n, 2n]$, где n – заданное натуральное число.
5. Дано натуральное n -значное число. Оставить в этом числе только первую цифру, а остальные заменить нулями.
6. Дано натуральное число N . Вывести все совершенные числа, которые $\leq N$. Совершенным называется число равное, сумме своих делителей.
7. Дана последовательность из n натуральных чисел и натуральное число A . Найти в данной последовательности число, которое имеет самый большой наибольший общий делитель с числом A .
8. Дано натуральное число N . Найти и вывести все числа в интервале от 1 до $N-1$, у которых сумма всех цифр совпадает с суммой цифр данного числа.
9. Найти все натуральные n -значные числа, цифры в которых образуют строго возрастающую последовательность (например, 1234).
10. Из заданного числа вычли сумму его цифр. Из результата вновь вычли сумму его цифр и т.д. сколько таких действий надо произвести, чтобы получить нуль?
11. Для последовательности $a_1=1$, $a_{n+1} = a_n + \frac{1}{1-a_n}$ составить программу печати k -того члена в виде обыкновенной несократимой дроби.
12. Дано четное число $n > 2$. Проверить для него гипотезу Гольдбаха: каждое четное n представляется в виде суммы двух простых чисел.

Рекурсия*

Теория

Рекурсия – это такой способ организации вычислительного процесса, при котором процедура или функция в ходе выполнения составляющих ее операторов обращается сама к себе.

В рекурсивной процедуре или функции обязательно должно содержаться условие окончания рекурсии.

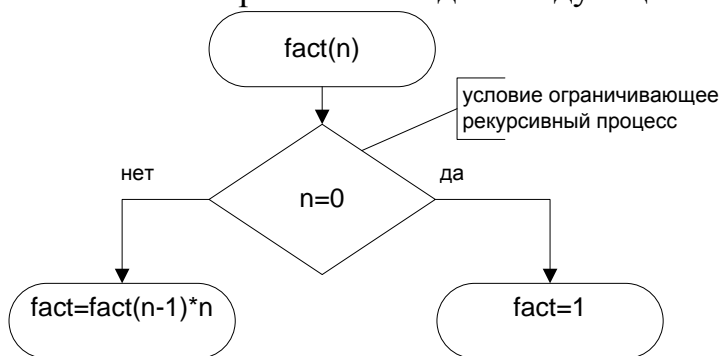
При выполнении рекурсии, когда подпрограмма доходит до рекурсивного вызова, процесс приостанавливается и новый процесс запускается с начала, но уже на новом уровне. Прерванный же процесс запоминается. Новый процесс тоже может

быть приостановлен и т.д. Образуется последовательность прерванных процессов. Последовательность рекурсивных обращений должна **обязательно** выходить на определенное значение.

При каждом очередном входе в подпрограмму ее локальные параметры и адреса возврата размещаются в специальной области памяти – стеке. После выхода на определенной значение выполняется обратный ход- цепочка вычислений по обратному маршруту, сохраненному с стеке.

Пример 9.

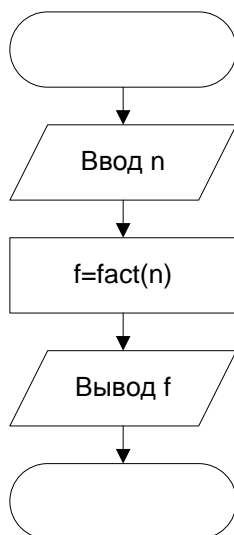
Рассмотрим это на примере функции вычисления факториала. Хорошо известно, что $0!=1$, $1!=1$. А как вычислить величину $n!$ для большого n ? Если бы мы могли вычислить величину $(n-1)!$, то тогда мы легко вычислим $n!$, поскольку $n!=n*(n-1)!$. Но как вычислить $(n-1)!$? Если бы мы вычислили $(n-2)!$, то мы сможем вычислить и $(n-1)!=(n-1)*(n-2)!$. А как вычислить $(n-2)!$? Если бы... В конце концов, мы дойдем до величины $0!$, которая равна 1. Таким образом, для вычисления факториала мы можем использовать значение факториала для меньшего числа. Блок-схема этого алгоритма выглядит следующим образом:



Программа соответствующего алгоритма имеет вид:

```
int factorial (int n)
{
    if (n == 0)
    {
        return 1;
    }
    else
    {
        return n * factorial(n - 1);
    }
}
```

Программа вызова рекурсивной функции вычисления факториала будет иметь вид:



```

program recurc1;
var n: integer;
    f: real;
function fact(n: integer): real;
begin
  if n=0 then fact:=0
    else fact:=fact(n-1)*n;
end;
begin
  write('n='); readln(n);
  f:=fact(n);
  writeln('n!=', n:5:0);
end.
  
```

Рекурсивные функции являются мощным механизмом в программировании. К сожалению, они не всегда эффективны. Также часто использование рекурсии приводит к ошибкам, наиболее распространенная из таких ошибок – бесконечная рекурсия, когда цепочка вызовов функций никогда не завершается и продолжается, пока не кончится свободная память в компьютере. Две наиболее распространенные причины для бесконечной рекурсии:

1. Неправильное оформление выхода из рекурсии. Например, если мы в программе вычисления факториала забудем поставить проверку `if (n == 0)`, то `factorial(0)` вызовет `factorial(-1)`, тот вызовет `factorial(-2)` и т.д.

2. Рекурсивный вызов с неправильными параметрами. Например, если функция `factorial(n)` будет вызывать `factorial(n)`, то также получится бесконечная цепочка.

Поэтому при разработке рекурсивной функции необходимо прежде всего оформлять условия завершения рекурсии и думать, почему рекурсия когда-либо завершит работу.

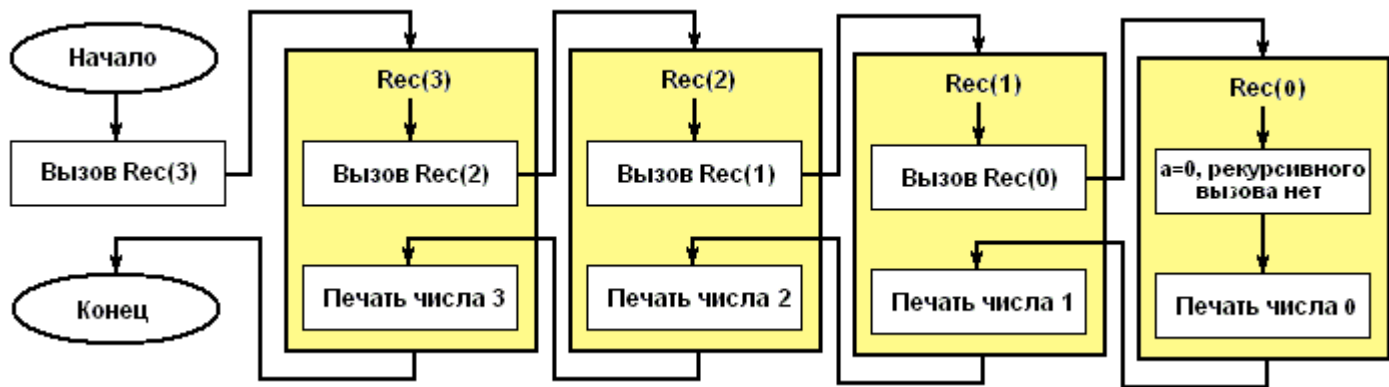
Общее требование к задачам с рекурсией: когда вы сдаёте задачу, использующую рекурсию, вы устно обосновываете, почему рекурсия всегда когда-нибудь закончится.

Еще один пример рекурсивной void-функции:

```

void Rec(int a)
{ if (a>0) Rec(a-1);
  cout<<a<<endl;
}
  
```

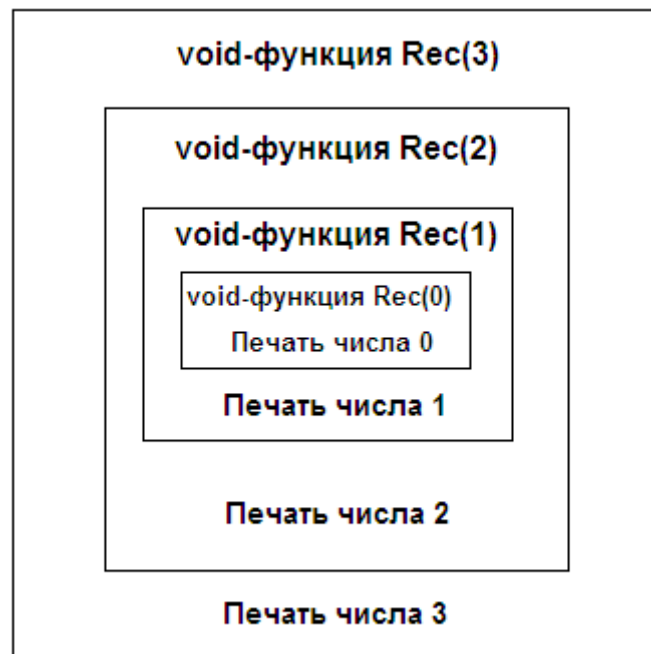
Рассмотрим, что произойдет, если в основной программе поставить вызов, например, вида `Rec(3)`. Ниже представлена блок-схема, показывающая последовательность выполнения операторов.



Void-функция Res вызывается с параметром $a = 3$. В ней содержится вызов void-функции Res с параметром $a = 2$. Предыдущий вызов еще не завершился, поэтому можете представить себе, что создается еще одна void-функция и до окончания ее работы первая свою работу не заканчивает. Процесс вызова заканчивается, когда параметр $a = 0$. В этот момент одновременно выполняются 4 экземпляра void-функции. Количество одновременно выполняемых void-функций называют *глубиной рекурсии*.

Четвертая вызванная void-функция (Res(0)) напечатает число 0 и закончит свою работу. После этого управление возвращается к void-функции, которая ее вызвала (Res(1)) и печатается число 1. И так далее пока не завершатся все void-функции. Результатом исходного вызова будет печать четырех чисел: 0, 1, 2, 3.

Еще один визуальный образ происходящего представлен на рисунке.



Выполнение void-функции Res с параметром 3 состоит из выполнения void-функции Res с параметром 2 и печати числа 3. В свою очередь выполнение void-функции Res с параметром 2 состоит из выполнения void-функции Res с параметром 1 и печати числа 2. И т. д.

Использование рекурсивных подпрограмм – красивый прием с точки зрения программирования. Программа выглядит более компактно. Но у рекурсии есть недостатки: рекурсия выполняется более медленно и глубина рекурсии ограничена.

Примеры

Пример 1.

Даны действительные числа x и ε ($x \neq 0$, $\varepsilon > 0$). Вычислить сумму с точностью ε .

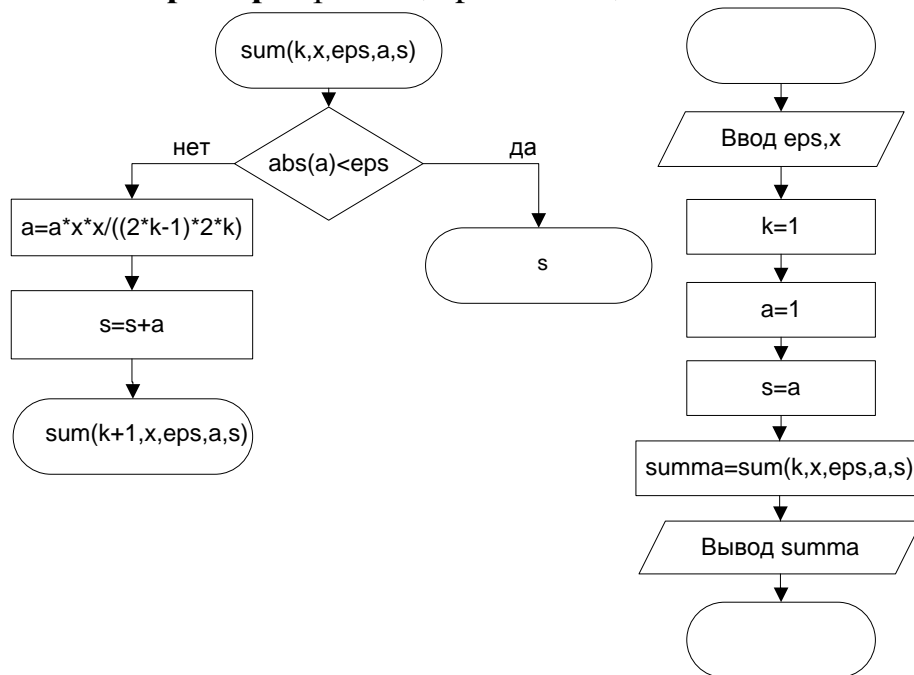
$$\sum_{k=0}^{\infty} \frac{x^{2k}}{(2k)!}$$

Исходные данные: x – вещественный тип, точность eps – вещественный тип.

Результат: сумма summa – вещественный тип.

Вычисление суммы выполняем в функции sum . Рекуррентная формула, которая используется при вычислении суммы: $a_k = a_{k-1} \cdot (x^2) / ((2k-1)(2k))$. Вызов функции заканчивается при $\text{abs}(a) < \text{eps}$. Начальное значение $a_0 = 1$. Это значение суммируется в основной программе.

Тестовый пример: при $x=4$, $\text{eps}=0.0001$, $\text{summa}=27.3082$



```

namespace Сумма
{
    class Program
    {
        static double summ(int k, double x, double eps, ref double a, ref double s)
        {
            if (Math.Abs(a) < eps) return s;
            else
            {
                a = a * x * x / ((2 * k - 1) * 2 * k);
                s = s + a;
                return summ(k + 1, x, eps, ref a, ref s);
            }
        }
    }
}
  
```

```

static void Main(string[] args)
{
    double summa;
    double eps = 0.0001f;
    double x = 4f;
    int k = 1;
    double a = 1f;
    double s = a;
    summa = summ(k,x,eps, ref a, ref s);
    Console.WriteLine("summa={0}", summa);
    Console.ReadKey();
}
}
}

```

Пример 2.

Подсчитать количество цифр в заданном натуральном числе.

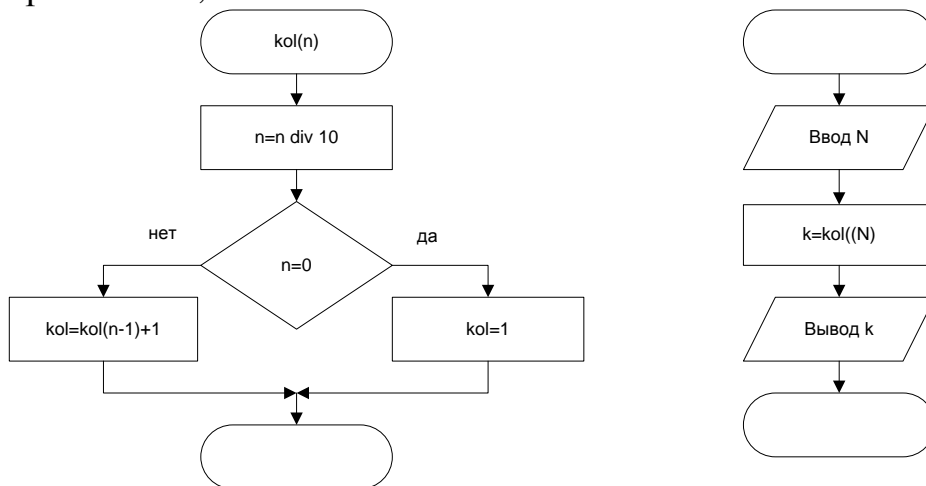
Исходные данные: заданное число N – целый тип.

Результат: количество цифр k – целый тип.

В рекурсивной функции заданное число делится на 10, для уменьшения количества цифр. Деление прекращается, когда результатом деления будет ноль. Это граничное условие для рекурсии.

Тестовый пример:

при N=5674, k=4.



Приведем пример с рекурсивной процедурой.

Пример 3.

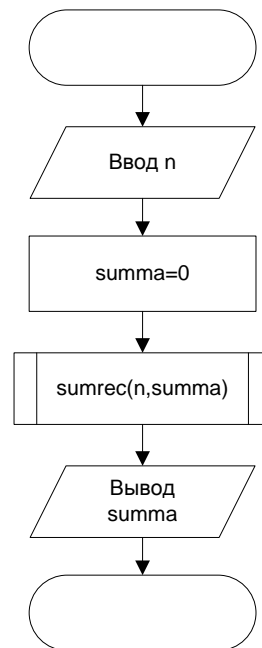
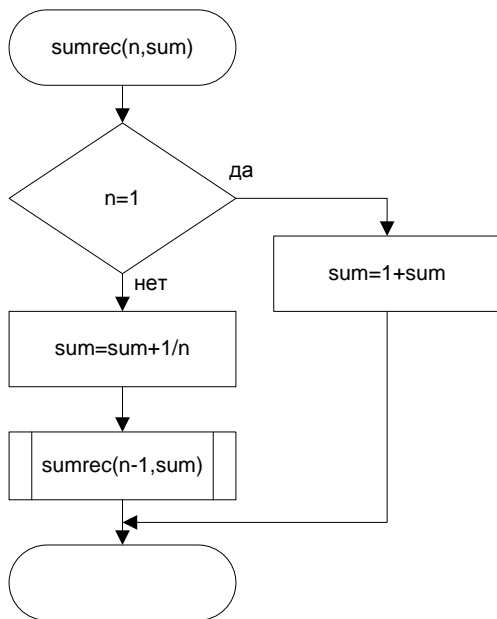
Вычислить сумму n членов ряда: $\sum_{i=1}^n \frac{1}{i}$

Исходные данные: количество слагаемых n – целый тип.

Результат: сумма ряда summa – целый тип.

Тестовый пример:

при n=5 summa=2.2833



```

namespace Рекурсия
{
    class Program
    {
        static void sumrec(int n, ref double sum)
        {
            if (n==1) sum=1+sum;
            else {sum=sum+1.0/n;
                sumrec(n-1, ref sum);
            }
        }

        static void Main(string[] args)
        {
            int n;
            double summa = 0;
            Console.WriteLine("n");
            n = Convert.ToInt32(Console.ReadLine());
            sumrec(n, ref summa);
            Console.WriteLine("summa={0}", summa);
            Console.ReadKey();
        }
    }
}
  
```

Задание Написать и отладить программу для примера 2.

Контрольные вопросы

1. Какая подпрограмма называется рекурсивной.
2. Почему в рекурсивной подпрограмме отсутствуют операторы цикла.
3. Может ли в рекурсивной подпрограмме отсутствовать развилка.
4. Почему в Примере 10 отсутствует операция $k=k+1$.
5. Как будет работать рекурсивная подпрограмма в Примере 10 при $\text{eps}=0.1$ и $x=1$.
6. Как будет работать рекурсивная подпрограмма в Примере 11 при $k=333$.
7. В Примере 12 укажите параметры по ссылке и параметры по значению в процедуре sumrec.

Индивидуальные задания

1. Найти сумму цифр заданного натурального числа.
2. Написать программу вычисления целой степени любого вещественного числа.
3. Составить программу для нахождения числа, которое образуется из данного натурального числа при записи его цифр в обратном порядке
4. Даны неотрицательные целые числа n, m ; вычислить $A(n, m)$, где

$$A(n, m) = \begin{cases} m + 1, & \text{если } n = 0, \\ A(n - 1, 1), & \text{если } n \neq 0, m = 0, \\ A(n - 1, A(m - 1)), & \text{если } n > 0, m > 0 \end{cases}$$

5. Создать программу вычисляющую $\sqrt[k]{a}$.
Для этого надо вычислить элементы числовой последовательности:

$$x_0 = a: x_i = \frac{k-1}{k} x_{i-1} + \frac{a}{k \cdot x_{i-1}^{k-1}}, \quad i = 1, 2, \dots$$

Найти первое значение x_n , для которого $|x_n^k - a| < 10^{-4}$.

6. Создать логическую функцию, которая возвращает True, если ее аргумент - простое число.
7. Даны действительные числа x и ε ($x \neq 0, \varepsilon > 0$). Вычислить с точностью ε и указать количество учтенных слагаемых

$$\sum_{k=0}^{\infty} \frac{(-1)^k}{((k+1)!)^2} \left(\frac{x}{2}\right)^{2(k+1)}$$

Вычисление выражения под знаком суммы выполнить через рекурсию.

8. Даны действительные числа x и ε ($x \neq 0, \varepsilon > 0$). Вычислить с точностью ε и указать количество учтенных слагаемых

$$\sum_{k=0}^{\infty} \frac{(-1)^{k+1} x^{2k-1}}{(2k-1)!(2k+1)!}$$

Вычисление выражения под знаком суммы выполнить через рекурсию.

9. Дано действительное число ε ($\varepsilon > 0$). Последовательность a_1, a_2, \dots образована по следующему закону:

$$a_n = \left(1 - \frac{1}{2}\right) \cdot \left(1 - \frac{1}{3}\right) \cdot \dots \cdot \left(1 - \frac{1}{n+1}\right)$$

С помощью рекурсии найти первый член a_n ($n \geq 2$), для которого выполнено условие: $|a_n - a_{n-1}| < \varepsilon$.

10. Дано действительное число ε ($\varepsilon > 0$). Последовательность a_1, a_2, \dots образована по следующему закону:

$$a_n = \left(1 - \frac{1}{2!}\right) \cdot \left(1 + \frac{1}{3!}\right) \cdot \dots \cdot \left(1 - \frac{(-1)^n}{(n+1)!}\right)$$

С помощью рекурсии найти первый член a_n ($n \geq 2$), для которого выполнено условие: $|a_n - a_{n-1}| < \varepsilon$.

11. Даны действительные числа x и ε ($\varepsilon > 0$). Последовательность a_1, a_2, \dots образована по следующему закону: $a_1 = x$; далее, для $n=2, 3, \dots$ выполнено:

$$a_n = \frac{2}{a_{n-1}} + \frac{x}{4 + a_{n-1}^2}$$

С помощью рекурсии найти первый член a_n ($n \geq 2$), для которого выполнено условие: $|a_n - a_{n-1}| < \varepsilon$ (ограничиться рассмотрением первых 10^4 членов).

12. Даны действительные числа x и ε ($\varepsilon > 0$). Последовательность a_1, a_2, \dots образована по следующему закону: $a_1 = x$; далее, для $n=2, 3, \dots$ выполнено:

$$a_n = \frac{1}{\sqrt{|4a_{n-1}^2 - 2x + 1|}}$$

С помощью рекурсии найти первый член a_n ($n \geq 2$), для которого выполнено условие: $|a_n - a_{n-1}| < \varepsilon$ (ограничиться рассмотрением первых 10^4 членов).

Тема 6. Алгоритмы работы со структурированными типами данных

Наряду с простыми типами, рассмотренными в предыдущих темах, существуют структурированные типы данных, которые могут представлять совокупность значений. Переменные этих типов имеют структуры, которые определяются Виртом как совокупность связанных данных и множество правил, определяющих их организацию и способ доступа к элементам данных. Выбором той или иной структуры данных определяется и алгоритм обработки данных, от которого зависит эффективность их обработки.

В языке C++ определены следующие стандартные структурированные типы данных:

- массив;
- строка;
- структура;

В данной лабораторной работе будут рассмотрены алгоритмы работы с массивами.

Лабораторная работа 6

Стандартные алгоритмы работы с одномерными массивами

Теория

Массив представляет собой совокупность пронумерованных однотипных значений, имеющих общее имя. Элементы массива обозначаются переменными с индексами. Индексы записывают в квадратных скобках после имени массива. Например:

$t[0]$, $t[5]$, $t[i]$, $h[1][9]$, $h[i][j]$ и т.п.

Массив, хранящий линейную таблицу, называется одномерным. Тип элементов массива называется его *базовым* типом.

Объявление массива:

Имя_типа *Имя_массива*[*Объявленный_размер*] ;

Объявление массива в таком виде задает элементы в количестве *Объявленный_размер* пронумерованные от *Имя_массива*[0] до *Имя_массива*[*Объявленный_размер* - 1]. Значение каждого из них имеет тип *Имя_типа*.

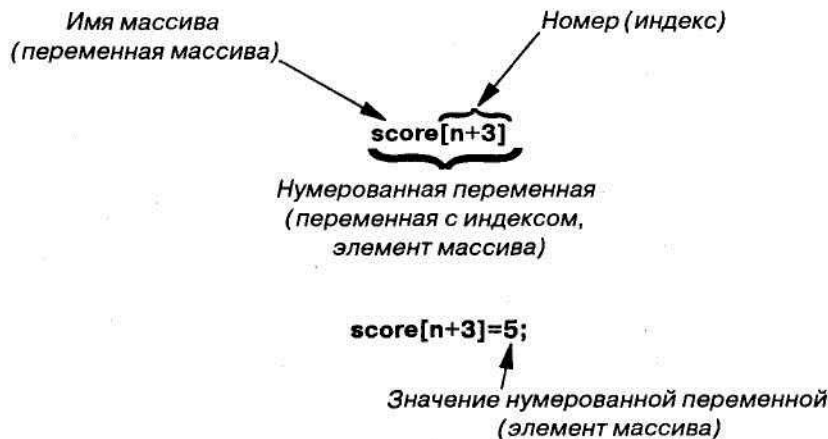
Индексы массивов всегда начинаются с нуля и заканчиваются целым числом, которое на единицу меньше размера массива.

Например:

```
int score [6];
```

Описание массива определяет, во-первых, размещение массива в памяти, во-вторых, правила его дальнейшего употребления в программе. Последовательные элементы массива располагаются в последовательных ячейках памяти ($score[0]$, $score[1]$ и т. д.), причем значения индекса не должны выходить из диапазона 0...5. В

качестве индекса может употребляться любое выражение соответствующего типа. Например, `score[n+3]`.



Одной из наиболее часто встречающихся ошибок, допускаемых при написании программ с применением массивов, является попытка обращения к несуществующим элементам используемого массива. Рассмотрим следующее объявление массива:

```
int a[6];
```

Любое выражение, выступающее в роли индекса массива `a`, должно давать одно из целых чисел от 0 до 5. Например, если в программе содержатся элементы `a[i]`, то переменная `i` должна принимать значение, равное одному из шести целых чисел 0, 1, 2, 3, 4 или 5. Если же значение этой переменной будет иным, то это приведет к ошибке. Когда выражение, играющее роль индекса, принимает значение, отличное от допускаемого объявлением массива, говорят, что индекс **выходит за пределы** массива. К сожалению, в большинстве компиляторов сообщение о выходе за пределы массива не выдается; результатом использования неверного индекса массива будет некорректное поведение программы, возможно, приводящее к ее аварийному останову (который может произойти без всякого предупреждающего сообщения).

Инициализация массивов

Массив тоже можно инициализировать как обычную переменную. При этом значения элементов заключаются в фигурные скобки и отделяются друг от друга запятыми. Например, следующая инструкция инициализирует все три элемента массива `children`:

```
int children[3]={2, 12, 1};
```

Приведенное выше объявление эквивалентно следующему коду:

```
int children[3];
```

```
children[0]=2; children[1]=12; children[2]=1;
```

Если в фигурных скобках приведено меньше значений, чем имеется элементов массива, то оставшиеся элементы инициализируются нулевым значением базового типа массива.

Если массив инициализируется при объявлении, то его размер можно опустить — он автоматически будет равен количеству инициализирующих значений. Например, объявление

```
int b[]={5, 12, 11};
```

эквивалентно объявлению

```
int b[3]={5, 12, 11};
```

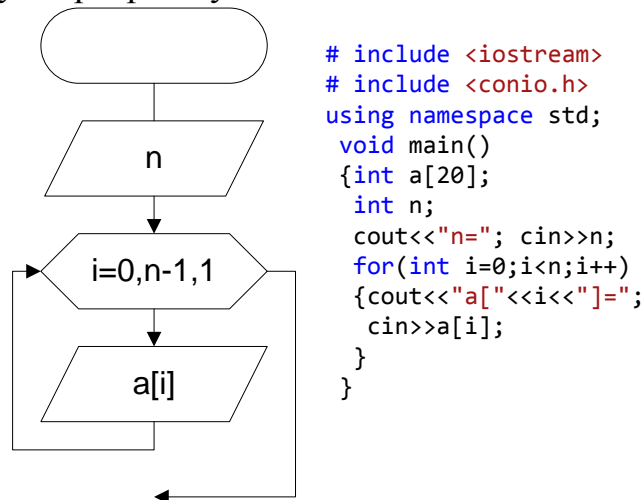
При описании массива надо точно указать границы его индексов. Изменение размеров массива происходит через изменение в тексте программы и повторную компиляцию. Один из способов сделать программу более гибкой — использовать в ней именованные константы в качестве размеров каждого массива.

```
const int N=10;  
int a[N];
```

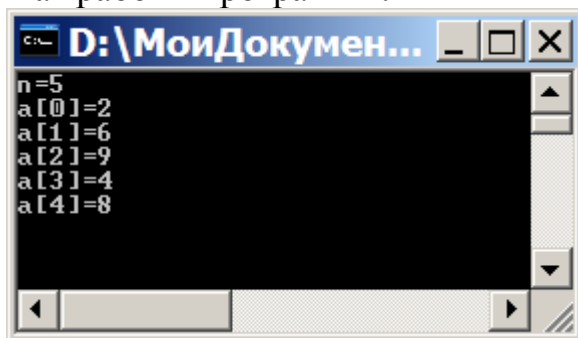
Теперь для изменения размеров массива *a* и всех операторов программы, связанных с этими размерами, достаточно отредактировать только одну строку в программе — описание константы *N*.

Другой вариант использовать в программе массива с изменяющимся количеством элементов – это зарезервировать место для элементов массива с «запасом». А во время работы программы вводить количество используемых элементов массива.

Обработка массивов в программах производится покомпонентно т.е. в цикле. Приведем блок-схему и программу ввода значений в массив:



В программе приведен пример ввода элементов массива с комментариями. Результат работы программы:



Пример программы ввода массива в строчку:

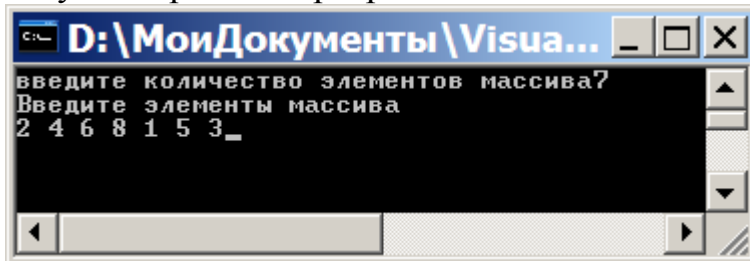
```
# include <iostream>  
# include <conio.h>  
#include <locale>  
using namespace std;  
void main()
```

```

{
    setlocale(0, "");
    int a[20];
    int n;
    cout<<"Введите количество элементов массива"; cin>>n;
    cout<<"Введите элементы массива"<<endl;
    for(int i=0;i<n;i++)
        cin>>a[i];
}

```

Результат работы программы:



Использование массива в качестве аргумента функции

В списке формальных параметров функции может быть массив, так что при ее вызове аргументом, который подставляется вместо этого формального параметра, является весь массив. Однако в этом случае параметр не передается в функцию ни по значению, ни по ссылке. Это новый вид формального параметра, который называется **параметр-массив**. Рассмотрим пример функции для ввода элементов массива. Это void-функция, которая выполняет последовательность действий: ввод элементов массива в цикле. В данной функции, задан один параметр-массив *a*, который при вызове этой функции будет заменен массивом-аргументом. В эту функцию передается по значению еще один обычный параметр — *n*; предполагается, что этот параметр является целым числом, равным размеру массива. Функция заполняет свой параметр-массив (т.е. элементы массива) вводимыми с клавиатуры значениями.

Прототип функции:

```

void vvod(int a[], int n);
// Предусловие: объявлен массив a с размером n.
// Пользователь вводит n целых чисел.
// Постусловие: массив a заполнен введенными с клавиатуры
// целыми числами в количестве n.

```

Определение функции:

```

void vvod(int a[], int size)
{cout<<"Введите "<<n<<" чисел:\n";
  for (int i=0; i<n; i++)
    cin>>a[i];
}

```

Формальный параметр `int a []` является параметром-массивом. Об этом в C++ свидетельствуют квадратные скобки без каких-либо индексных выражений. Параметр-массив — это *не* совсем то же самое, что параметр, передаваемый по ссылке, но во многих ситуациях эти два вида параметров ведут себя очень схоже. Разберем подробно приведенный пример, чтобы понять, как же работает аргумент-

массив в данном случае. (**Аргумент-массив** — это, конечно же, массив, который подставляется вместо параметра-массива, такого как `a []`.)

В вызове функции `vvod` должны быть указаны два аргумента: первый является массивом целых чисел, а второй — объявленным размером этого массива. Например, допустим следующий вызов функции:

```
int score[5], num = 5;
vvod(score, num);
```

Такой вызов функции `vvod` заполнит массив `score` пятью целыми числами, введенными с клавиатуры. Обратите внимание, что формальный параметр `a []` (который используется в прототипе функции и в заголовке ее объявления) содержит квадратные скобки без индексного выражения. (В эти квадратные скобки можно вставить какое-нибудь число, но компилятор его проигнорирует.) Аргумент же, указываемый при вызове функции (в нашем примере это `score`), не содержит никаких квадратных скобок и индексных выражений.

При вызове функции, в качестве аргумента которой используется массив, любое действие, выполняемое с параметром-массивом, выполняется с аргументом-массивом, значения которого могут изменяться функцией. Если формальный параметр изменяется в теле функции (например, с помощью `cin`-инструкции), то аргумент-массив будет изменен.

При использовании массива в качестве аргумента функции передается только адрес первого элемента массива. Базовый тип аргумента-массива должен совпадать с типом формального параметра, поэтому функция уже имеет информацию о базовом типе аргумента. *Однако через аргумент-массив в функцию не передаются сведения о размере массива.* При выполнении кода функции компьютер знает, с какой ячейки памяти начинается массив и сколько памяти занимает каждый элемент массива, *но не знает, сколько элементов содержится в массиве* (если не указать ему этого дополнительно). Вот почему важно всегда задавать дополнительный аргумент типа `int`, в котором содержится информация о размере массива. (В этом и состоит различие между параметром-массивом и параметром, передаваемым по ссылке. Параметр-массив можно представить себе в виде неполноценной разновидности передаваемого по ссылке параметра, когда функция располагает всеми сведениями о массиве, *за исключением его размера.*)

Применение модификатора `const`

При вызове с аргументом-массивом функция может изменять значения, которые хранятся в этом массиве. Обычно это не вызывает проблем. Однако при написании сложного определения функции можно непреднамеренно изменить одну или несколько величин, которые хранятся в массиве, даже если этот массив не должен меняться. Можно указать компилятору, что вы не собираетесь изменять аргумент-массив, и тогда он будет проверять, не выполняется ли в коде изменение значений массива, и сообщать об этом. Для этого перед параметром-массивом нужно вставить модификатор `const`. Параметр-массив с модификатором `const` называется **константным параметром-массивом** (`constant array parameter`).

Например, приведенная ниже функция выводит хранящиеся в массиве значения, но не изменяет их.

Прототип функции:

```
void vivod(const int a[], int n)
// Предусловие: объявлен массив a с размером n
// Всем элементам массива a присвоены значения.
// Постусловие: значения массива a выведены на экран.
```

Определение функции:

```
void vivod(const int a[], int n)
{
    cout<<"В массиве содержатся следующие значения:\n";
    for (int i=0; i<n; i++)
        cout<<a[i]<<" ";
    cout << endl;
}
```

Модификатор const можно применять к параметру любого типа, но обычно он применяется к параметрам-массивам.

В дальнейшем ввод и вывод элементов массива будет выполняться в void-функциях.

Примеры

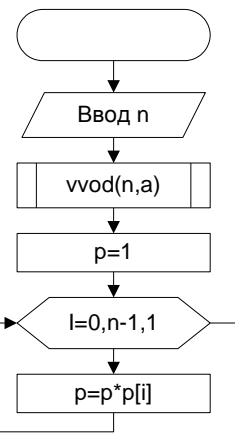
Пример 1.

Вычислить произведение элементов массива.

Исходные данные: n элементов вещественного массива a, n – целый тип.

Результат: Произведение элементов массива p – тип.

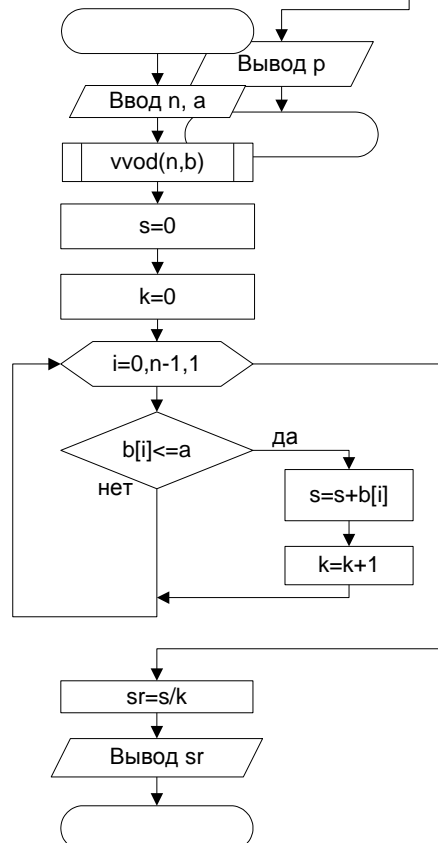
Тестовый пример: при n=5, элементы массива a=: 1 3 5 3 4, p=180.



целый

Пример 2.

Найти среднее арифметическое значение элементов массива, не превышающих числа a



Исходные данные: N элементов вещественного массива b. Вещественное число a.

Результат: Среднее арифметическое значение sr.

Промежуточные значения:

i - Индекс элементов массива

s - Сумма элементов массива, не превышающих число a.

k - Количество элементов массива, не превышающих число a.

Тестовый пример:

при n=7, a=2,

элементы массива b=1 -3 5 4 -4 6 2, sr=-1

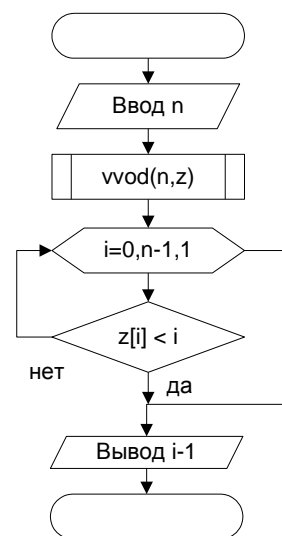
Пример 3.

Определить, сколько элементов целочисленного массива стоит до первого элемента, значение которого меньше своего индекса.

Исходные данные: Целочисленный массив z из n элементов.

Результат: i-1 - Количество элементов, стоящих до первого элемента, меньшего своего индекса.

Тестовый пример: при n=7, элементы массива z=1 6 15 4 0 6 -1, i-1=4.



Пример 4.

Найти минимальный элемент и его индекс среди элементов массива с номерами от k до l.

Исходные данные: Целочисленный массив d из n элементов

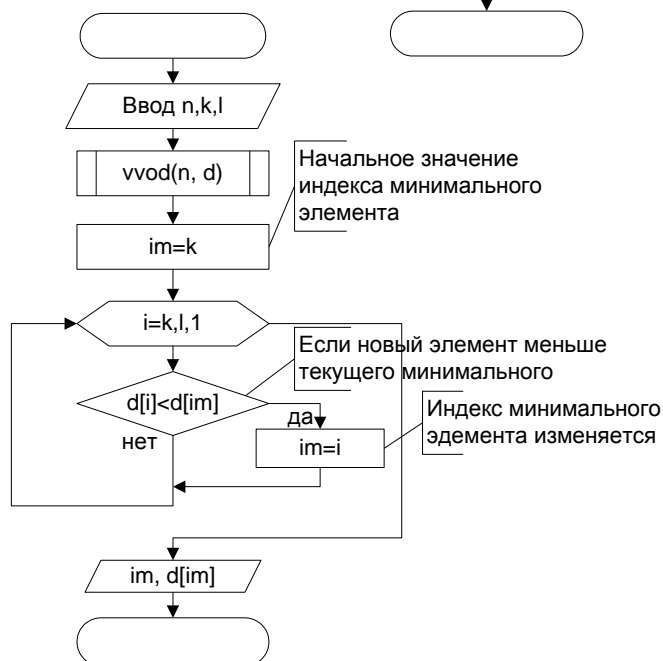
k – начало поиска.

l – конец поиска.

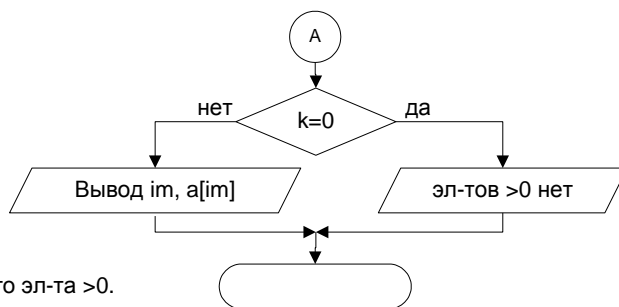
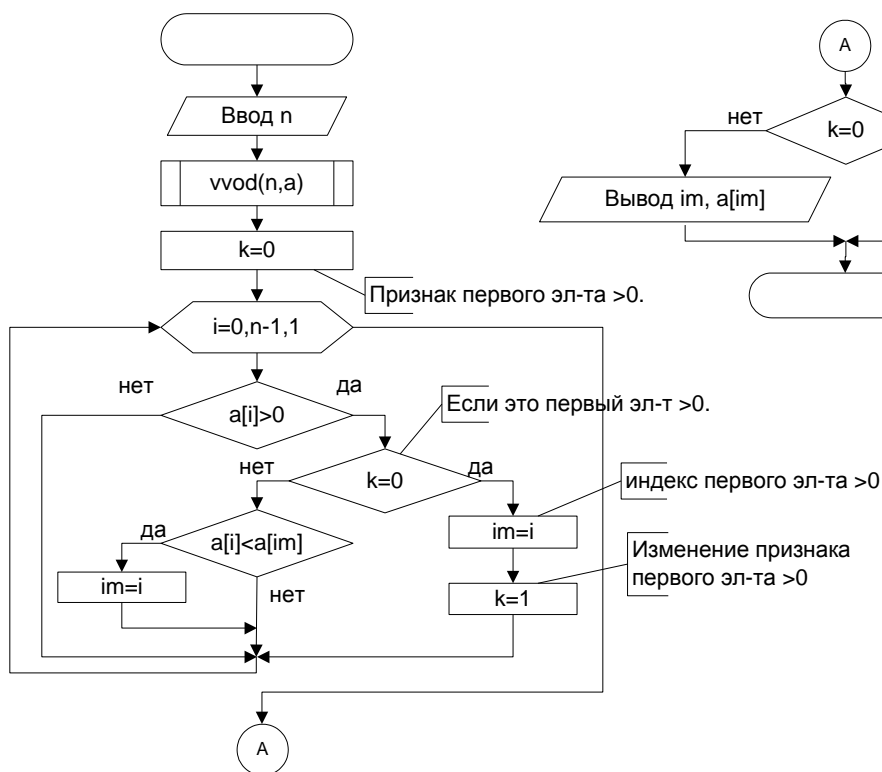
Результаты: im - Номер минимального элемента в заданном наборе элементов. Для минимального элемента дополнительная переменная не нужна, потому что значение минимального элемента – это d[im].

Тестовый пример: при n=7, k=1, l=4 элементы массива d=1 -3 5 4 -4 -6 2, im=4, d[im]=-4.

Пример 5.



Определить номер и значение минимального элемента среди положительных элементов массива вещественных чисел



Исходные данные: Вещественный массив a из n элементов.

Результат: im – Номер минимального элемента

Промежуточные значения

я: k – определяет наличие положительного элемента в массиве. k равно нулю пока не встретится положительный элемент.

Тестовый пример: при $n=10$ и элементах: $-3, -5, 7, -2, 5, 2, -7, 5, 3, 6$
 $im=5, a[5]=2$

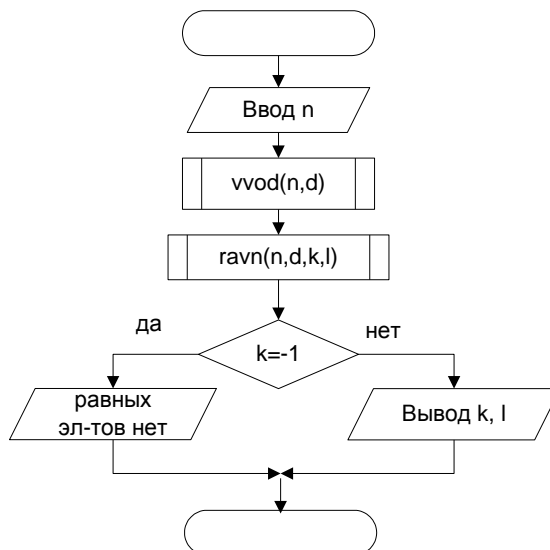
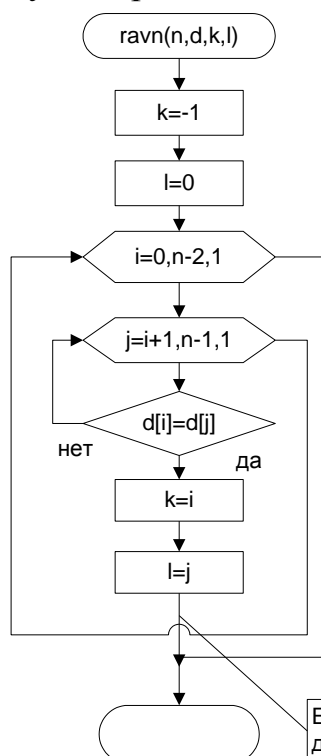
Пример 6.

В целочисленном массиве найти номера двух первых равных элементов.

Исходные данные: Целочисленный массив d из n элементов.

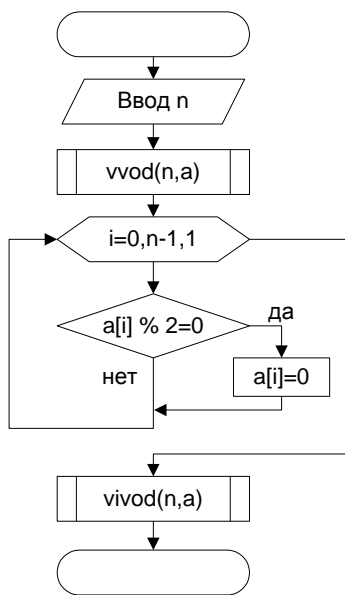
Результат: k, l – Номера первых равных элементов.

В данном примере поиск равных элементов лучше вести в процедуре, в этом случае при нахождении равных элементов можно досрочно выйти из процедуры, а значит сразу из двух циклов.



значит сразу из двух циклов.

Тестовый пример: при $n=7$, элементы $4, 6, 3, 6, 3, 7, 1, k=1, l=3$.



Пример 7.

Заменить элементы, имеющие четное значение нулем.

Исходные данные: Целочисленный массив a из n элементов.

Результат: Целочисленный массив a из n элементов.

Тестовый пример: при n=7

введенный массив: 4, 6, 7, 9, 3, 2, 1;

преобразованный массив: 0, 0, 7, 9, 3, 0, 1.

значения элементов двух массивов, имеющих четные индексы.

Исходные данные: вещественные массивы c и d из n элементов.

Результат: вещественные массивы c и d из n элементов.

Тестовый пример: при n=5

введенные массивы:

c: 1, 2, 3, 4, 5

d: -1, -2, -3, -4, -5.

преобразованные массивы:

c: 1, -2, 3, -4, 5

d: -1, 2, -3, 4, -5.

Пример 9.

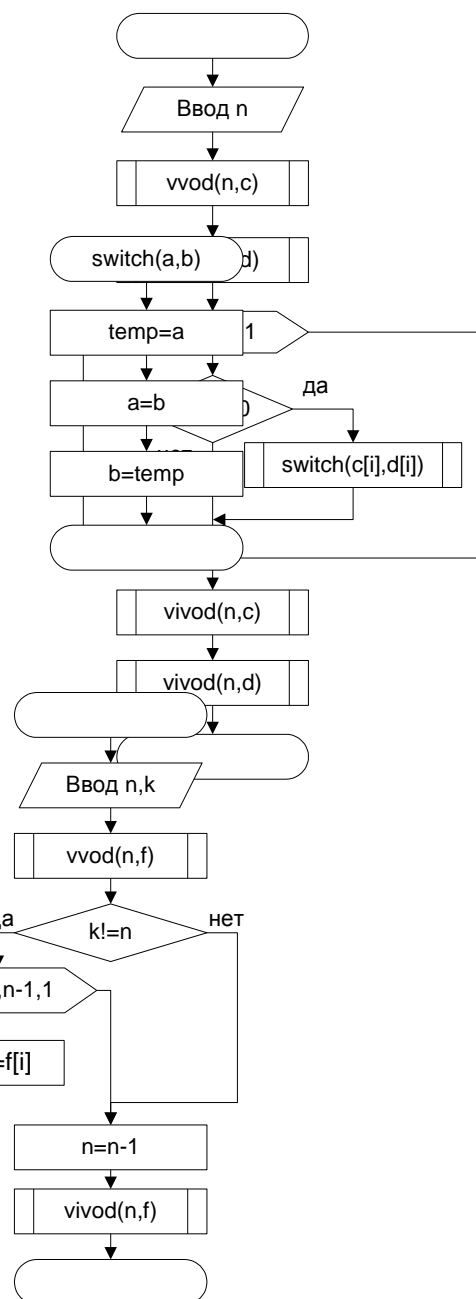
Удалить элемент с номером k.

Дано: вещественный массив f из n элементов,

k – номер удаляемого элемента.

Пример 8.

Переставить местами



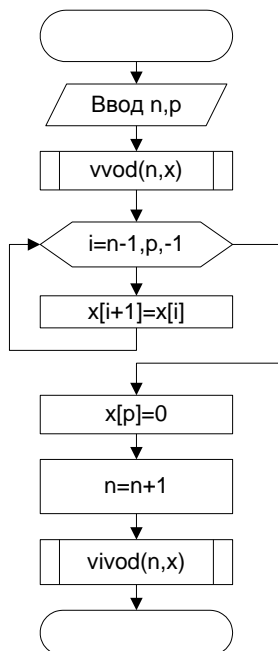
Результат: Вещественный массив f из $n - 1$ элементов.

При удалении элементы массива сдвигаются влево.

Тестовый пример: при $n=7$ и $k=4$

ввод: 5, 6, 4, 8, 1, 9, 2

вывод: 5, 6, 4, 1, 9, 2; $n=6$.



Пример 10.

Вставить в массив после заданного элемента нуль.

Дано: Целочисленный массив x из n элементов,
 p – номер элемента.

Результат: Целочисленный массив x из $n+1$ элементов.

При вставке элементы массива сдвигаются вправо.

Чтобы не потерять элементы, сдвиг выполняется с конца

Тестовый пример: $n=5$, $p=3$,

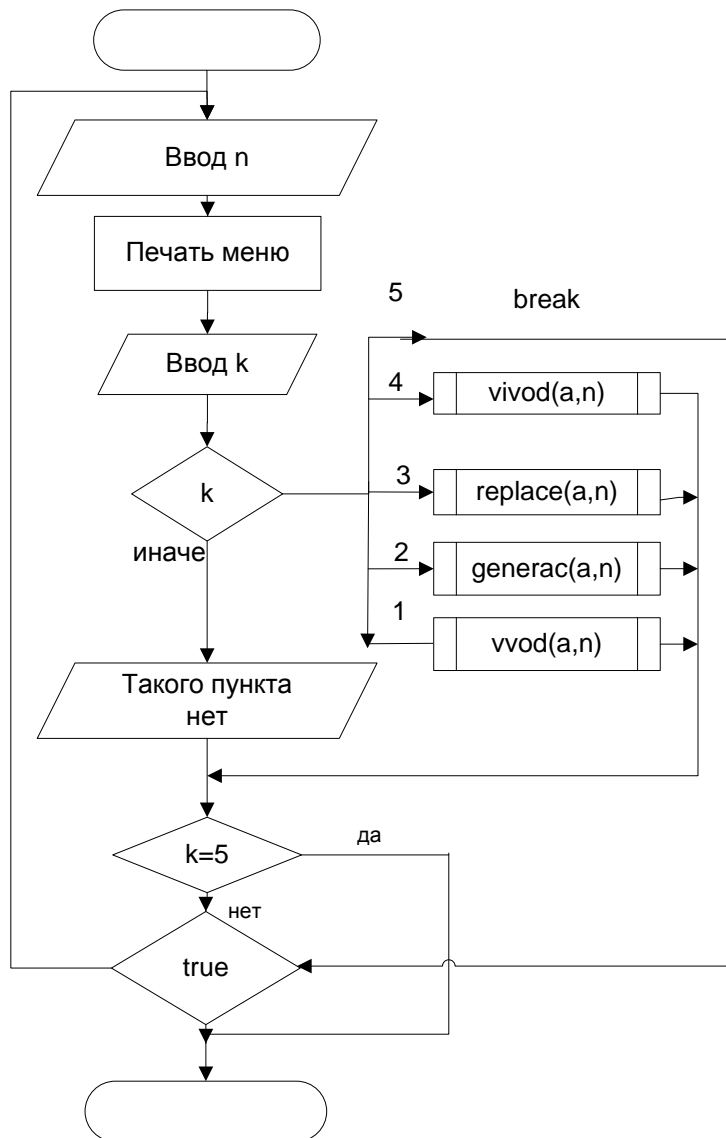
ввод: 7, 3, 4, 8, 1; вывод: 7, 3, 4, 0, 8, 1.

Пример 11.

Создать программу, обеспечивающую работу следующих пунктов меню.

1. Ввод массива целых чисел из 20 элементов.
2. Генерация 20 элементов массива от -100 до 100.
3. Замена в массиве отрицательных элементов нулем.
4. Вывод элементов массива.
5. Конец работы.

Исходные данные: Массив a из 20 целых чисел.



Текст программы:

```

#include <iostream>
#include <conio.h>
#include <locale>
#include <time.h>
#include <cstdlib>
using namespace std;
void vvod(int a[], int n);
void generac(int a[], int n);
void replace(int a[], int n);
void vivod(int a[], int n);
void main()
{
    srand((unsigned)time(NULL));
    setlocale(0, "");
    int a[20];
    int n,k;
    cout<<"Введите количество элементов массива"; cin>>n;
    do
    {
        cout<<"1. Ввод массива a"<<endl;
        cout<<"2. Генетация элементов массива a"<<endl;
        cout<<"3. Замена отрицательных элементов нулями"<<endl;
        cout<<"4. Вывод элементов массива"<<endl;
        cout<<"5. Конец работы"<<endl;
        cout<<"Укажите номер пункта меню"; cin>>k;
        switch (k)
    }

```

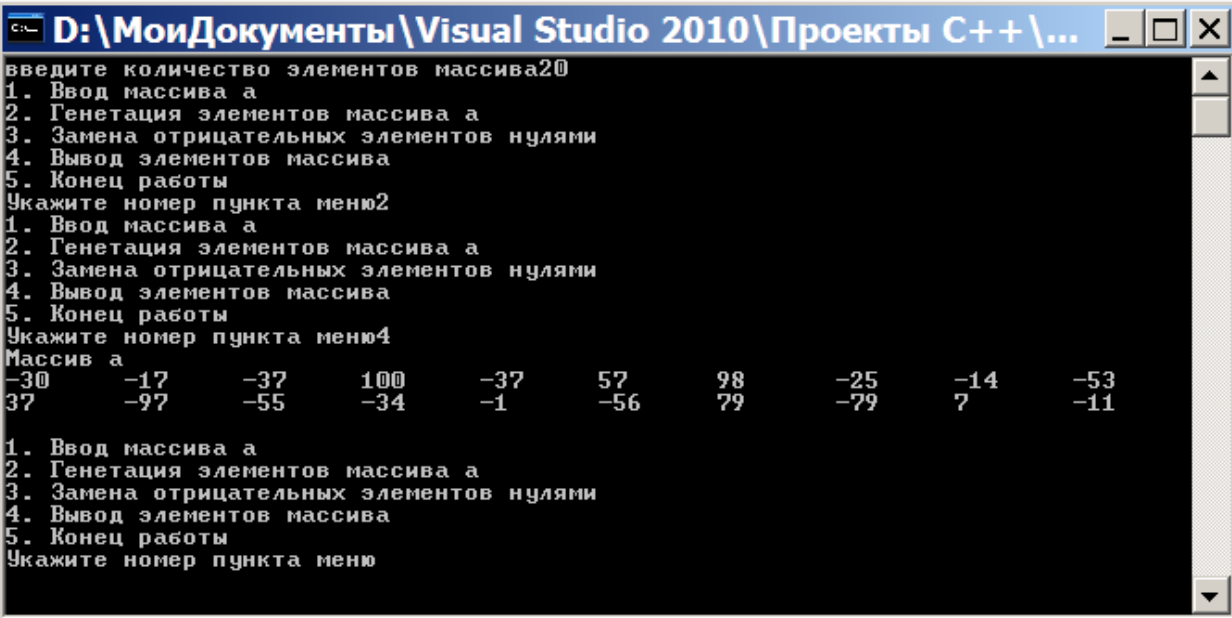
```

        {case 1: vvod(a,n);break;
        case 2: generac(a,n); break;
        case 3: replace(a,n); break;
        case 4: vivod(a,n);break;
        case 5: cout<<"Конец работы"<<endl; break;
        default: cout<<"Такого пункта нет"<<endl;
        }
    }
    if (k==5) break;
} while(true);
_getch();
}

void vvod(int a[], int n)
{
cout<<"Введите элементы массива"<<endl;
    for(int i=0;i<n;i++)
        cin>>a[i];
}
void generac(int a[], int n)
{ for(int i=0;i<n;i++)
    a[i]=rand()%(201)-100;
}
void replace(int a[], int n)
{for(int i=0;i<n;i++)
    if(a[i]<0) a[i]=0;
}
void vivod(int a[], int n)
{cout<<"Массив a"<<endl;
    for(int i=0;i<n;i++)
        cout<<a[i]<<"\t";
    cout<<endl;
}
}

```

Результат:



```

D:\МоиДокументы\Visual Studio 2010\Проекты C++ \...
введите количество элементов массива20
1. Ввод массива а
2. Генетация элементов массива а
3. Замена отрицательных элементов нулями
4. Вывод элементов массива
5. Конец работы
Укажите номер пункта меню2
1. Ввод массива а
2. Генетация элементов массива а
3. Замена отрицательных элементов нулями
4. Вывод элементов массива
5. Конец работы
Укажите номер пункта меню4
Массив а
-30    -17    -37    100    -37    57    98    -25    -14    -53
37     -97    -55    -34    -1     -56   79    -79    7     -11
1. Ввод массива а
2. Генетация элементов массива а
3. Замена отрицательных элементов нулями
4. Вывод элементов массива
5. Конец работы
Укажите номер пункта меню

```

Задание 1. Написать и отладить программы для примера 6.

Контрольные вопросы

1. Какой тип данных не допускается для индекса.
2. Могут ли в описании массива в индексах содержаться переменные.
3. Что надо делать, если заранее количество элементов неизвестно.

4. Почему в Примере 6 сравнивать элементы лучше в void-функции.
5. Почему в Примере 10 цикл выполняется от n-1 до 0.
6. Может ли массив являться параметром цикла и что для этого надо сделать.
7. В каких ситуациях при передаче массива в функцию перед ним нужно писать const.

Индивидуальные задания

- 1.** Создать меню и выполнение всех его пунктов.
 1. Ввод массива целых чисел.
 2. Вывод массива в строку.
 3. Вычисление среднего арифметического значения элементов массива и замена положительных элементов массива целой частью от среднего арифметического значения.
 4. Конец работы.
- 2.** Создать меню и выполнение всех его пунктов.
 5. Ввод массива целых чисел.
 6. Вывод массива в строку.
 7. Вычисление минимального элемента. Вычисление максимального элемента. Замена всех элементов равных максимальному элементу значением минимального элемента.
 8. Конец работы.
- 3.** Создать меню и выполнение всех его пунктов.
 1. Ввод массива целых чисел.
 2. Вывод массива в строку.
 3. Найти минимальный элемент. Найти максимальный элемент. Заменить все элементы, стоящие между минимальным и максимальным элементом - нулями.
 4. Конец работы.
- 4.** Создать меню и выполнение всех его пунктов.
 1. Ввод массива целых чисел.
 2. Вывод массива в строку.
 3. Найти количество положительных элементов в массиве. Заменить все четные элементы массива на их индексы.
 4. Конец работы.
- 5.** Создать меню и выполнение всех его пунктов.
 1. Ввод массива целых чисел.
 2. Вывод массива в строку.
 3. Найти минимальный элемент. Найти количество элементов имеющих минимальное значение. Удалить все минимальные элементы из массива
 4. Конец работы.
- 6.** Создать меню и выполнение всех его пунктов.
 1. Ввод массива целых чисел.
 2. Вывод массива в строку.

3. Найти первый отрицательный элемент в массиве. Найти сумму отрицательных элементов. Заменить все элементы, стоящие перед первым отрицательным элементом на сумму отрицательных элементов.
4. Конец работы.
- 7.** Создать меню и выполнение всех его пунктов.
 1. Ввод массива целых чисел.
 2. Вывод массива в строку.
 3. Найти минимальный элемент среди элементов, имеющих четный индекс. Найти минимальный элемент среди элементов, имеющих нечетный индекс. Заменить все элементы, стоящие между этими минимальными элементами на их индексы.
 4. Конец работы.
- 8.** Создать меню и выполнение всех его пунктов.
 1. Ввод массива целых чисел.
 2. Вывод массива в строку.
 3. Найти минимальный элемент среди положительных элементов массива. Заменить все отрицательные элементы массива на этот минимальный элемент.
 4. Конец работы.
- 9.** Создать меню и выполнение всех его пунктов.
 1. Ввод массива целых чисел.
 2. Вывод массива в строку.
 3. Найти второй по порядку минимальный элемент. Заменить все элементы, кратные трем, на этот элемент.
 4. Конец работы.
- 10.** Создать меню и выполнение всех его пунктов.
 1. Ввод массива целых чисел.
 2. Вывод массива в строку.
 3. Найти минимальный элемент. Вставить после минимального элемента его индекс.
 4. Конец работы.
- 11.** Создать меню и выполнение всех его пунктов.
 1. Ввод массива целых чисел.
 2. Вывод массива в строку.
 3. Найти максимальный элемент. Найти количество элементов имеющих максимальное. Удалить все максимальные элементы из массива
 4. Конец работы.
- 12.** Создать меню и выполнение всех его пунктов.
 1. Ввод массива целых чисел.
 2. Вывод массива в строку.
 3. Найти количество элементов, стоящих после последнего отрицательного элемента в массиве. Вставить после этого элемента найденное количество.
 4. Конец работы.

Формирование массива

Теория

Существует целый ряд задач, в которых требуется на основе исходного, или исходных массивов получить новый массив. В общем случае при формировании нового массива следует использовать новый индекс для этого массива. В новом массиве следует зарезервировать количество элементов с учетом количества элементов в исходном массиве.

Примеры

Пример 12.

Сформировать новый массив из индексов элементов массива x , равных элементу с номером p .

Исходные данные: Целочисленный массив x из n элементов,
 p – номер элемента.

Результат: Целочисленный массив y из j элементов.

При описании массива y следует зарезервировать столько же элементов, сколько в массиве x , т.к. все элементы массива x могут удовлетворять данному условию.

Тестовый пример: при $n=10$, $p=1$

массив x : 4, 6, 8, 4, 6, 2, 8, 4, 7, 4.

массив y : 1, 4, 8, 10

Пример 13.

Получить новый массив из элементов данного массива, расположенных между экстремальными элементами включительно.

Дано: Целочисленный массив t из n элементов,

Результат: Целочисленный массив v из h элементов

Промежуточные значения:

a – номер минимального элемента

b – номер максимального элемента

p – номер левого экстремального элемента,

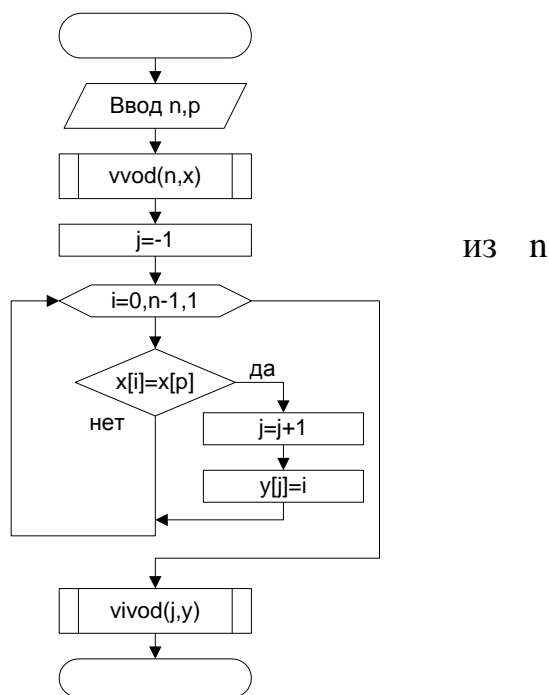
r – номер правого экстремального элемента

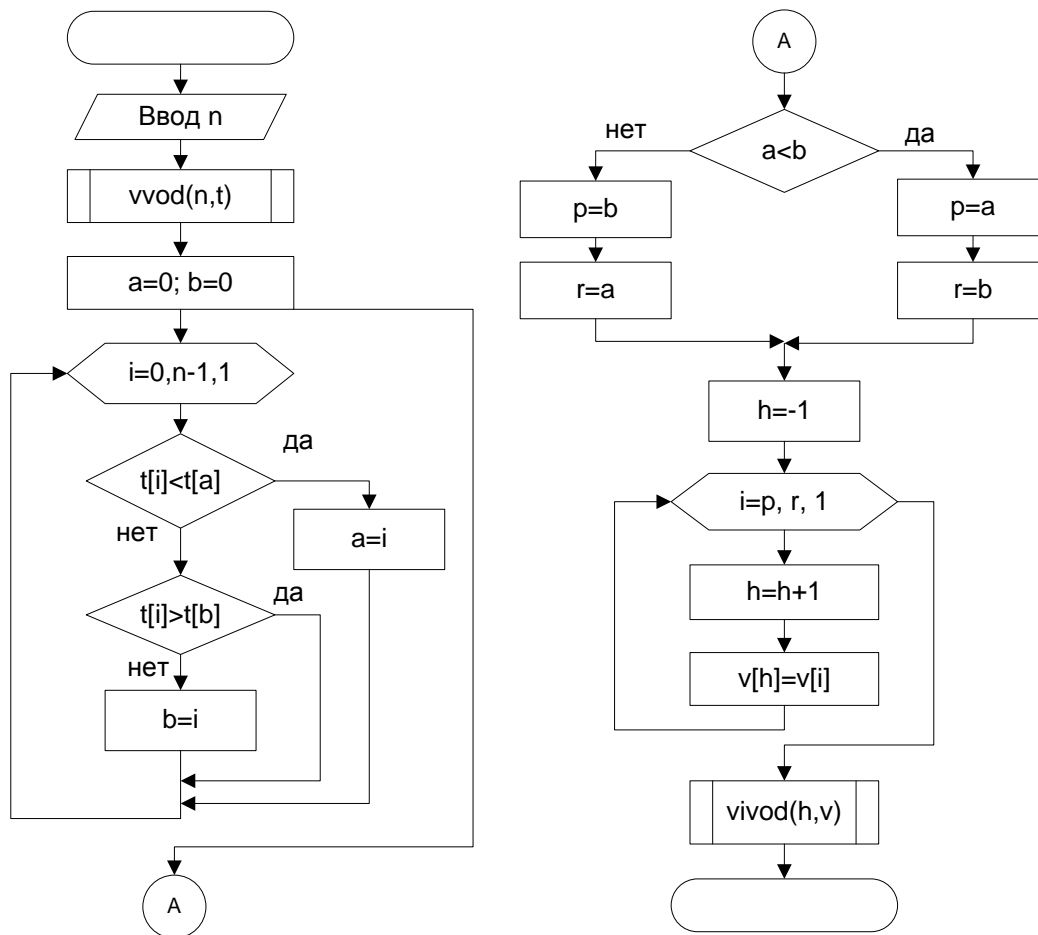
h – текущий индекс элементов массива v ,

Тестовый пример: при $n=10$,

массив t : 3, 6, 9, 3, 7, 4, 2, 8, 5, 6

массив v : 9, 3, 7, 4, 2.





Задание 2. Написать и отладить программы для примера 12.

Контрольные вопросы

1. Если новый массив получен из элементов исходного, сколько элементов надо зарезервировать для нового массива.
2. В каких ситуациях при получении нового массива не надо использовать новый индекс.
3. Для чего в Примере 13 используются переменные p и r.
4. Если новый массив получен из двух исходных массивов, сколько элементов надо зарезервировать для нового массива.

Индивидуальные задания

1. Дан массив целых чисел. Получить новый массив из данного без нулевых элементов.
2. Дан массив целых чисел, каждое из которых отлично от нуля. Если в массиве отрицательные и положительные члены чередуются (+, -, +, -, ... или -, +, -, +, ...), то получить новый массив совпадающий с данным. Иначе получить новый массив из отрицательных элементов данного массива, сохранив порядок их следования.

3. Дан массив действительных чисел. Получить новый массив, выбросив из исходного все члены равные максимальному и минимальному элементам данного массива.
4. Дан массив целых чисел. Если в данном массиве ни одно четное число не расположено после нечетного, то получить новый массив содержащий все отрицательные члены данного массива, иначе все положительные. Порядок следования чисел в обоих случаях заменяется на обратный.
5. Дан массив целых чисел, в котором могут встречаться повторяющиеся элементы. Получить новый массив, содержащий повторяющиеся элементы данного массива по одному разу.
6. Дан массив действительных чисел. Получить новый массив из элементов данного массива, значение которых больше среднего значения. Элементы в новом массиве должны располагаться в обратном порядке.
7. Дан массив действительных чисел. Получить новый массив из данного без элементов, расположенных между максимальным и минимальным элементом этого массива.
8. Дан массив целых чисел получить новый массив из положительных четных элементов первоначального массива.
9. Дан массив действительных чисел получить новый массив из элементов данного массива не меньших первого элемента данного массива. Элементы второго массива расположить в обратном порядке.
10. Дан массив действительных чисел. Получить новый массив их элементов данного массива, расположенных между максимальным и минимальным элементами.
11. Дан массив целых чисел. Получить новый массив из первоначального, отбросив все нулевые элементы в этом массиве.
12. Дан массив целых чисел. Получить новый массив их элементов данного последующему правилу: если в первоначальном массиве встречается элемент равный нулю, переписать в новый массив элемент, предшествующий нулю и следующий за нулем.

Двумерный массив

Теория

Данные из прямоугольных таблиц хранятся в двумерных массивах или матрицах. Каждый элемент двумерного массива характеризуется двумя индексами: первый индекс это номер строки, в которой находится данный элемент, второй индекс – номер столбца.

Массивы языка C могут иметь только один индекс, но при этом элементами массивов могут быть другие массивы, что позволяет создавать многомерные массивы. Пример двумерного массива в C:

```
char page[30] [100];
```


Приведенный массив, по существу, является одномерным 30-элементным массивом, базовый тип которого представлен одномерным 100-символьным массивом.

Объявление многомерного массива Синтаксис:

Тип Имя_массива[Разм_Измер_1] [Разм_Измер_2]...[Разм_Измер_N];

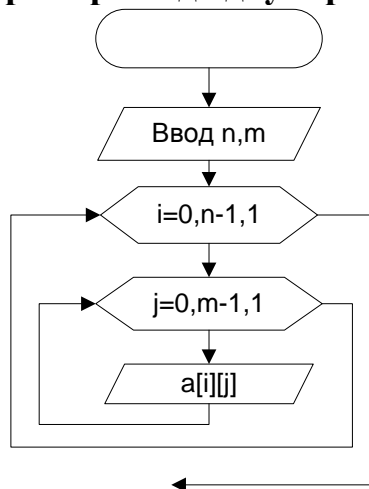
Например:

`int matrix[2][3];`

`double three_d_picture[10][20][30];`

Для работы с элементами двумерного массива требуется два вложенных цикла: внешний цикл перебирает строки, а внутренний цикл – столбцы в этих строках.

Пример ввода двумерного массива А:

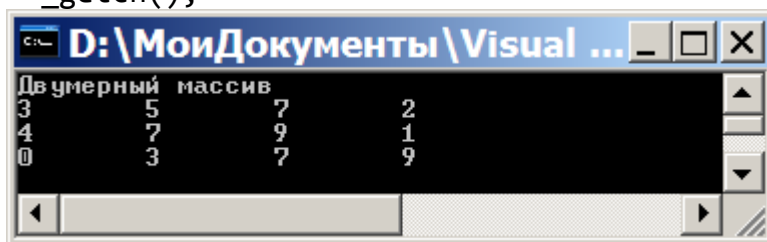


```
using namespace std;
void main()
{
    setlocale(0, "");
    int a[20][20];
    int n,m;
    cout<<"введите количество строк в массиве";
    cin>>n;
    cout<<"введите количество столбцов в массиве";
    cin>>m;
    cout<<"Введите элементы массива"<<endl;
    for(int i=0;i<n;i++)
        for(int j=0;j<m;j++)
            cin>>a[i][j];
}
```

Так же как и в одномерном массиве, блок-схема вывода и ввода массива совпадают.

Пример вывода двумерного массива как таблицы:

```
cout<<"Двумерный массив"<<endl;
for(int i=0;i<n;i++,cout<<endl)
    for(int j=0;j<m;j++)
        cout<<a[i][j]<<"\t";
_getch();
```



Многомерные массивы как параметры функций

Существует по крайней мере одна ситуация, когда двухмерный массив проявляет себя именно как массив массивов. Это бывает в функциях, параметры которых являются двухмерными массивами.

Когда в заголовке или прототипе функции фигурирует параметр, являющийся многомерным массивом, то размер первого его измерения не

указывается; размеры остальных измерений должны быть обязательно указаны.

Поскольку размер первого измерения не приводится, обычно возникает необходимость в дополнительном параметре типа `int`, который передает информацию об этом размере. Ниже приведен пример прототипа функции с двухмерным массивом `p` в качестве параметра.

```
void display_page(const char p[][100], int size_dimension_1)
```

Это правило обретает четкий смысл, если вспомнить, что параметр представляет собой массив массивов. Поскольку параметр `const char p[][100]` — это двухмерный массив, который представляет собой массив массивов, то лишь первое его измерение является индексом массива и трактуется так же, как и индекс одномерного массива. Вторая же размерность является частью описания базового типа, который представляет собой массив из 100 символов.

Пример void-функции ввода двумерного массива A, количество строк `n`, количество столбцов `m`.

```
void vvod2(int n,int m, int a[][10])
{
    for(int i=0;i<n;i++)
        for(int j=0;j<m;j++)
            cin>>a[i][j];
}
```

Двумерный массив можно вводить данные как таблицу.

Пример процедуры вывода двумерного массива A по строкам:

```
void vivod2(int n, int m, int a[][10])
{
    for(int i=0;i<n;i++,cout<<endl)
        for(int j=0;j<m;j++)
            cout<<a[i][j]<<"\t";
}
```

Обратите внимание на оператор `for(int i=0;i<n;i++,cout<<endl)` в нем при переходе на новую строку в матрице выполнен переход на новую строку на экране.

Программа ввода и вывода двумерного массива:

```
#include <iostream>
#include <conio.h>
#include <locale>
using namespace std;
void vvod2(int n,int m, int a[][10]);
void vivod2(int n, int m, int a[][10]);
void main()
{
    setlocale(0, "");
    int x[20][10];
```

```

int n,m;
cout<<"введите количество строк в массиве"; cin>>n;
  cout<<"введите количество столбцов в массиве"; cin>>m;
cout<<"Введите элементы массива"<<endl;
vvod2(n,m, x);
cout<<"Двумерный массив"<<endl;
vivod2(n,m, x,);
_getch();
}

```

Обратите внимание, что у формального параметра, массив a, - второе измерение имеет значение 10 и у фактического параметра, массив x – второе измерение имеет значение 10. Если, например, фактический параметр x будет описан как `int x[20][20]`, возникнет синтаксическая ошибка, связанная с несовпадением типа фактического и формального параметров.

Результат работы этой программы:

```

D:\МоиДокументы\Visual Studio ...
введите количество строк в массиве3
введите количество столбцов в массиве4
Введите элементы массива
3 5 8 2
6 10 5 7
3 1 8 2
Двумерный массив
3      5      8      2
6     10     5      7
3      1      8      2

```

В двумерном массиве, который является и матрицей существуют элементы главной и побочной диагонали. Элементы главной диагонали удовлетворяют условию $i=j$. Элементы побочной диагонали удовлетворяют условию: $i+j=m$.

Примеры

Пример 14.

Дан целочисленный двумерный массив. Определить сколько в этом массиве элементов больше среднего арифметического.

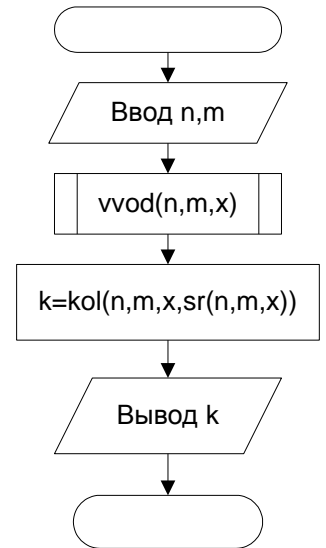
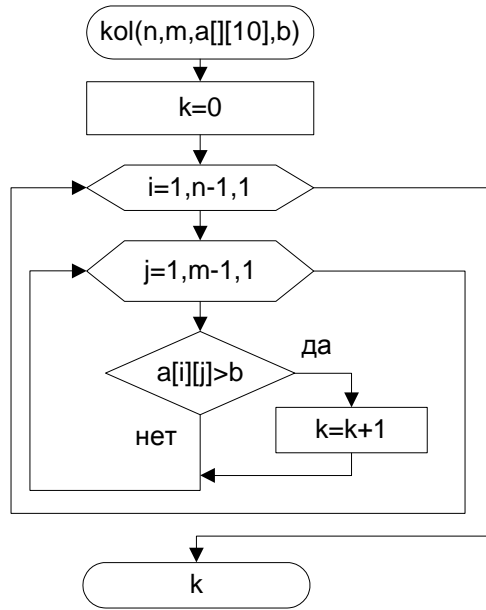
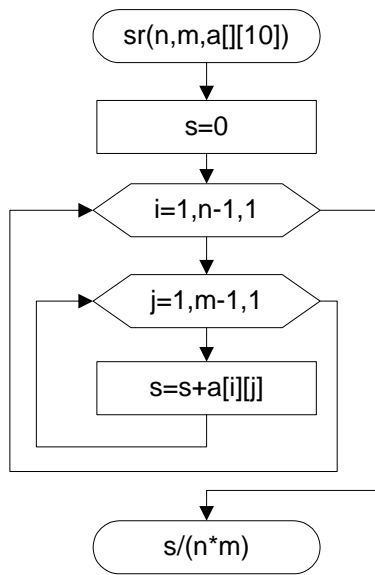
Исходные данные:

Целочисленный массив x, количество строк n, количество столбцов m.

Результат:

k- количество элементов больших среднего значения.

Среднее значение и количество элементов больших среднего вычисляется в функциях.



Пример 15.

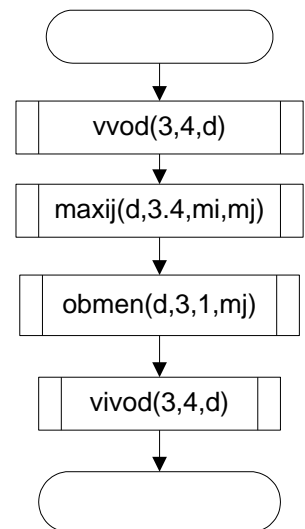
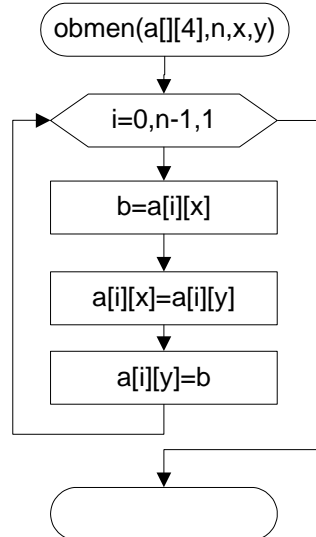
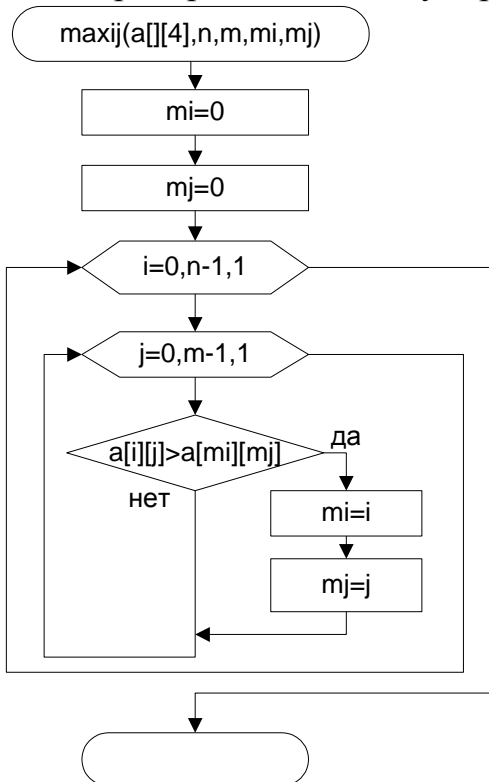
Дан двумерный массив 3 на 4 вещественных чисел. Переставить столбец, содержащий максимальный элемент с первым столбцом.

Исходные данные:

Двумерный массив d из 3 строк и 4 столбцов.

Результат:

Преобразованный двумерный массив d.



```

#include <iostream>
#include <conio.h>
#include <locale>
using namespace std;
void vvod(int n,int m, int a[][4]);
void vivod(int n, int m, int a[][4]);
void maxij(int a[][4],int n,int m, int& mi, int& mj);
void obmen(int a[][4],int n,int x, int y);
void main()
{
    setlocale(0, "");
    int d[3][4];
    int mi,mj;
    cout<<"Введите элементы массива"<<endl;
    vvod(3,4,d);
    maxij(d,3,4,mi,mj);
    obmen(d,3,0,mj);
    cout<<"Двумерный массив"<<endl;
    vivod(3,4,d);
    _getch();
}
void vvod(int n,int m,int a[][4])
{
    for(int i=0;i<n;i++)
        for(int j=0;j<m;j++)
            cin>>a[i][j];
}
void vivod(int n, int m, int a[][4])
{
    for(int i=0;i<n;i++,cout<<endl)
        for(int j=0;j<m;j++)
            cout<<a[i][j]<<"\t";
}
void maxij(int a[][4],int n,int m, int& mi, int& mj)
{mi=0;
mj=0;
for(int i=0;i<n;i++)
    for(int j=0;j<m;j++)
        if (a[i][j]<a[mi][mj])
            {mi=i;
            mj=j;
            }
}
void obmen(int a[][4],int n,int x, int y)
{int b;
    for(int i=0;i<n;i++)
        {b=a[i][x];
        a[i][x]=a[i][y];
        a[i][y]=b;
        }
}
}

```

Пример 16.

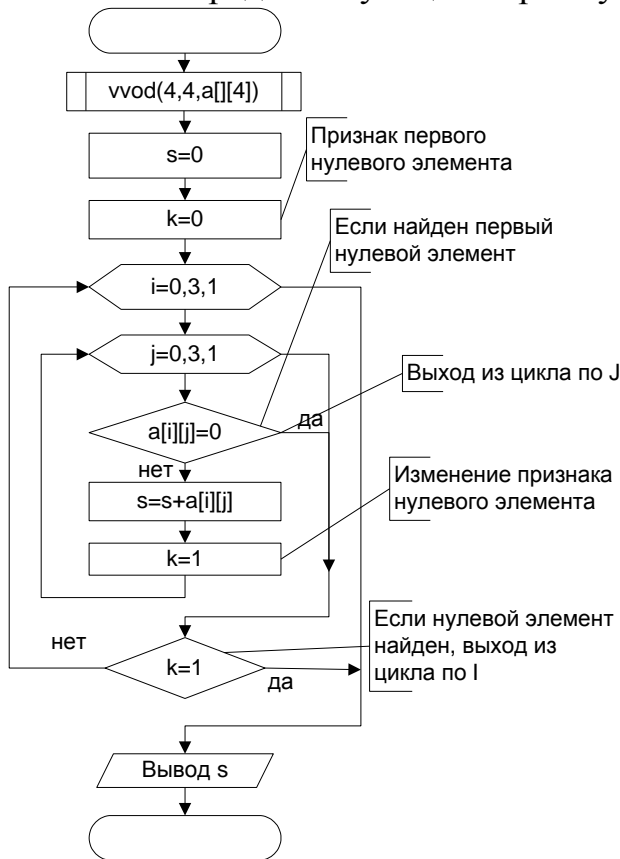
Дан двумерный массив целых чисел. Просуммировать элементы стоящие до первого нуля. Просмотр выполнять по строкам.

Исходные данные:

Двумерный массив а размером 4 на 4.

Результат:

s – сумма элементов предшествующих первому нулевому элементу.



Задание 3. Написать и отладить программы для примера 16.

Контрольные вопросы

1. Сколько элементов в массиве размером 4 на 5
2. Какое условие выполняется для элементов ниже главной диагонали.
3. Можно ли в Примере 15 вместо void-функции Max использовать функцию.
4. Укажите параметры по ссылке и параметры по значению в void-функции обмен из Примера 15.
5. Можно ли в Примере 16 обойтись без признака нулевого элемента k.

Индивидуальные задания

1. Дан двумерный массив целых чисел из 4 столбцов и 3 строк. Переставить местами строку, содержащую минимальный элемент и последнюю строку..
2. Дан двумерный массив вещественных чисел из 3 столбцов и 4 строк. Найти, сколько в каждом столбце отрицательных элементов.
3. Дан двумерный массив вещественных чисел из 4 столбцов и 3 строк. Найти сумму максимальных элементов каждой строки.
4. Дан двумерный массив целых чисел из 4 столбцов и 4 строк. Найти сколько нулей находится выше главной диагонали.

5. Дан двумерный массив целых чисел из 4 столбцов и 3 строк. Найти максимальный элемент в массиве и заменить его нулем.
6. Дан двумерный массив вещественных чисел из 3 столбцов и 4 строк. Все элементы в массиве стоящие после минимального элемента заменить первым элементом. Замену выполнять по строкам.
7. Дан двумерный массив вещественных чисел из 4 столбцов и 3 строк. Первый элемент в каждом столбце заменить на среднее арифметическое от всех элементов массива.
8. Дан двумерный массив целых чисел из 4 столбцов и 4 строк. Все элементы главной диагонали заменить на сумму элементов в данном массиве, имеющих четное значение.
9. Дан двумерный массив вещественных чисел из 4 столбцов и 3 строк. Поменять местами первый столбец и столбец и столбец, где находится максимальный элемент массива. Вывести массив по строкам до и после перестановки.
10. Дан двумерный массив целых чисел из 4 столбцов и 4 строк. Найти минимальный элемент в первой половине массива (просмотр вести по строкам) и во второй половине массива. Поменять местами эти минимальные элементы массива.
11. Дан двумерный массив целых чисел из 4 столбцов и 3 строк. Найти сумму элементов в этом массиве стоящих после первого нуля..
12. Дан двумерный массив целых чисел из 4 столбцов и 3 строк. Найти максимальный элемент среди отрицательных элементов этого массива. Вывести массив по строкам.

Лабораторная работа № 7

Строки в C++

Теория

Строковые значения (string values) нам уже встречались. Таким значением является, например, строка "Привет! ", которая входит в следующую cout-инструкцию: `cout << "Привет!";`

Строковые значения в C++ можно хранить в переменных и обрабатывать их подобно тому, как обрабатываются данные других типов.

В C++ существуют 2 вида строк:

- **C-строки**, унаследованные от своего предшественника — языка C
- Класс **string**.

Рассмотрим каждый из этих видов строк более подробно.

C-строки

Строковая переменная (cstring variable) — это то же самое, что и массив символов. Естественно представлять ее себе именно в таком виде. Строка суть множество символов, и для его хранения нам нужно не что иное, как массив символов. Таким образом, следующее объявление массива предоставляет

возможность хранить строковую переменную, состоящую не более чем из 9 символов: `char s [10];`

Ошибки здесь нет; как было отмечено, строковая переменная `s` может хранить не более 9 символов. Хранить 10 символов она не может, потому что используется несколько иначе, чем обычный массив символов.

На самом деле строковая переменная является частично заполненным массивом символов. Подобно любому другому частично заполненному массиву, она использует элементы массива, начиная с нулевого и заканчивая тем, в который записывается последний символ. Однако для того, чтобы запомнить количество используемых элементов частично заполненного массива, в строковой переменной применяется свой специфический метод. Здесь для записи количества используемых элементов нам не нужна дополнительная переменная типа `int`. *Вместо этого в массиве, в котором хранится строковая переменная, сразу после последнего символа строки ставится специальный символ '\0'*. Таким образом, если значением переменной `s` является строка "Привет! ", то элементы массива заполнены следующим образом:

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]
П	р	и	в	е	т	!	\0	?	?

Символ '\0' служит сигнальной меткой, обозначающей конец строки. Читая ее символы — `s[0]`, `s[1]`, `s[2]` и т.д., мы знаем, что, как только встретимся с символом '\0', это будет означать достижение конца строки. Поскольку символ '\0' всегда занимает один элемент массива, максимальная длина строки, которая может храниться в массиве, на единицу меньше его размера.

Символом '\0' помечается окончание строки, хранящейся в массиве символов. При таком использовании массив носит название строковой переменной. Хотя '\0' записывается в программе с помощью двух символов, на самом деле это один символ, который можно разместить в одной переменной типа `char` или в одном элементе массива символов.

Объявление строковых переменных

Строковая переменная — это в точности то же, что и массив символов, хотя и несколько иначе используемый. Строковая переменная объявляется как обычный массив символов.

Синтаксис:

```
char Имя_массива[Максимальный_размер_строки + 1];
```

Пример:

```
char my_cstring[11];
```

Добавление единицы (+1) позволяет зарезервировать место для нуля-символа '\0', который завершает каждую строку, хранящуюся в массиве. Например, в объявленной выше строковой переменной `my_string` можно хранить строку длиной не более 10 символов.

Символ '\0' называется нуль-символом. Как и символ новой строки '\n', он является специальным символом. На этом аналогия не кончается. Символ '\0', как и '\n', в программе записывается в виде двух символов, хотя на самом деле это всего одно символьное значение. Подобно любому другому символу, '\0' можно хранить в

одной переменной типа `char` или в одном элементе массива символов. Отличие `'\0'` от других символов позволяет использовать его как сигнальную метку, обозначающую окончание строки обычных символов.

Единственное, что отличает строковую переменную от обычного массива, — это то, что в конце строковой переменной обязательно должен быть символ `'\0'`. Впрочем, это отличие скорее относится к использованию массивов, нежели к тому, чем эти массивы являются. *Строковая переменная является массивом символов, хотя и используется несколько иначе.*

Особенности ввода-вывода строк.

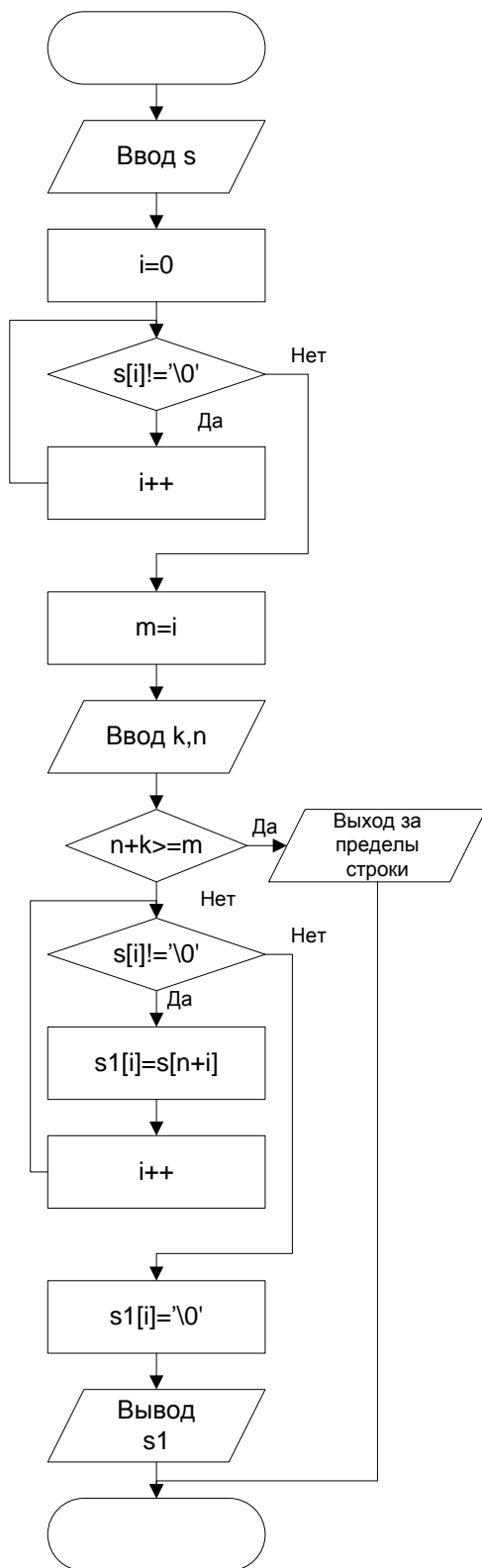
Выводить строки в поток можно с помощью оператора вывода `<<`. Например:

```
string s1;  
cout<<"Результат"<<s1<<endl;
```

Строковую переменную можно записывать с помощью оператора ввода `>>`, но при этом нужно иметь в виду одну особенность. Как при работе с другими типами данных, при таком чтении строки все пробельные символы (пробелы или символы новой строки) пропускаются. Более того, чтение ввода завершается первым же пробелом или символом новой строки. Если нужно, чтобы программа читала всю строку, для каждого слова можно использовать свой оператор ввода `>>`. Для чтения всей вводимой строки и присвоения ее строковой переменной можно воспользоваться библиотечной функцией `getline`, которая является функцией-членом каждого входного потока (такого, как `cin` или файловый входной поток). Эта функция будет рассмотрена далее.

Приведем пример программы работы с C-строками.

Задание: Выделить из заданного слова `s` подстроку `s1` из `k` символов начиная с позиции `n`.



```

#include<iostream>
#include <conio.h>
# include <windows.h>
using namespace std;
void main()
{SetConsoleCP(1251);
 SetConsoleOutputCP(1251);
 char s[50],s1[50];
 int n,k,m;
 cout<<"Строка="; cin>>s;
 cout<<s<<endl;
 /* Определение длины строки s*/
 int i=0;
 while (s[i]!='\0')
   i++;
 m=i;
 cout<<"n="; cin>>n;
 cout<<"k="; cin>>k;
 //Проверка на выход за пределы строки
 if (n+k>=m) {cout<<"Выход за пределы строки";
   return;}
 /*Получение новой подстроки из k символов с
 позиции n*/
 i=0;
 while(s[i]!='\0' && i<k)
 { s1[i]=s[n+i];
   i++;}
 /* Добавление в новую строку символа
 окончания строки*/
 s1[i]='\0';
 cout<<s1<<endl;
 _getch();
 }

```

Ввод кириллицы

Обратите внимание на подключение библиотеки `windows.h`. Ранее использовалась библиотека `locale` и метод `setlocale(0, "")` для отображения русских букв, набранных в редакторе кода. Но в данной задаче может возникнуть ситуация ввода русских букв в консольном экране. В этой ситуации метод `setlocale` не поможет.

Для ввода и вывода русских букв в консольном приложении надо:

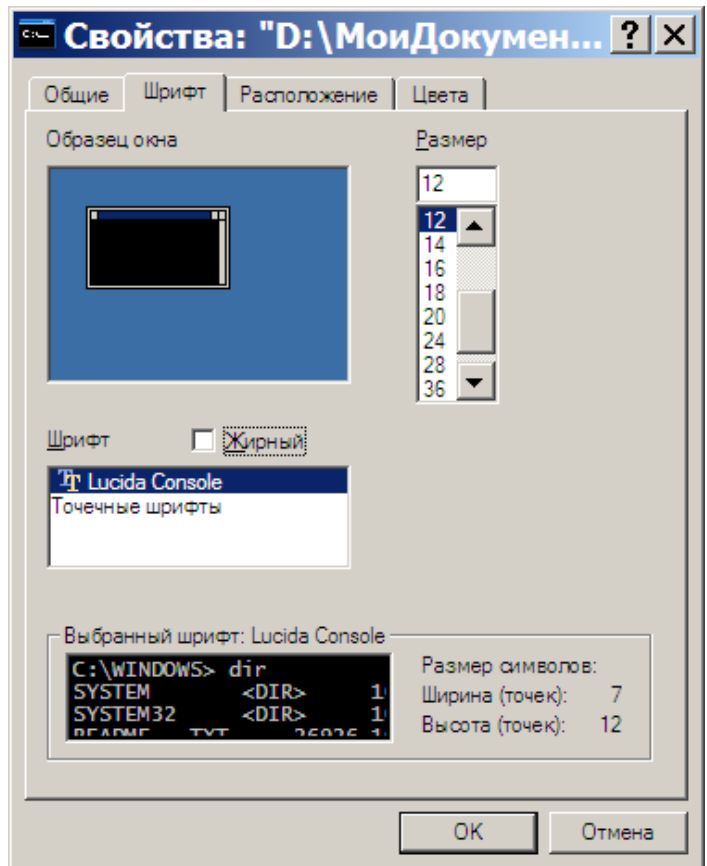
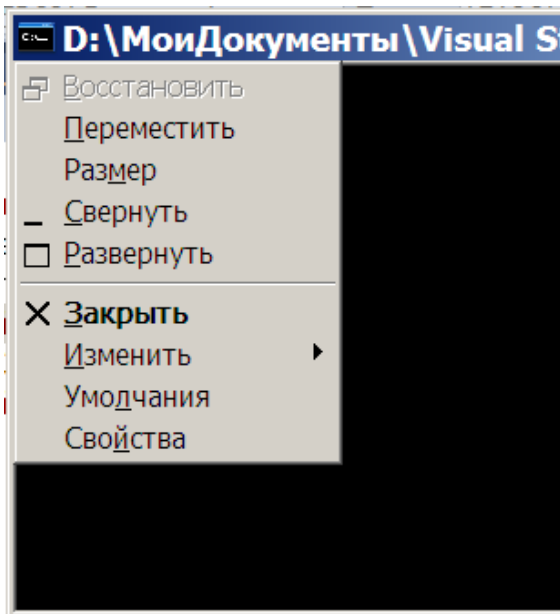
1. Подключить библиотеку `windows.h`:

```
# include <windows.h>
```

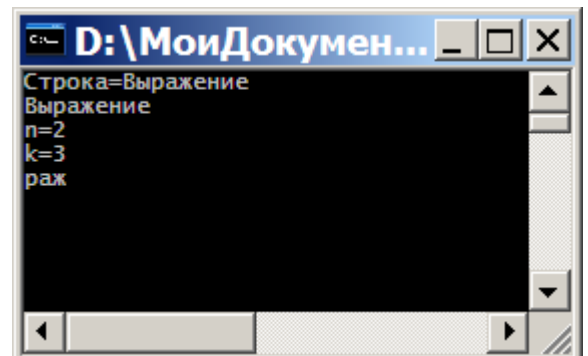
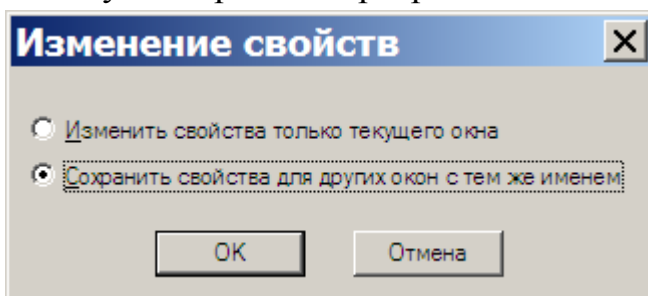
2. Вызвать методы:

```
SetConsoleCP(1251);  
SetConsoleOutputCP(1251);
```

3. В свойствах консоли (в окошке которое появляется при запуске программы) выбрать язык Lucide Console:



Результат работы программы:



Для С-строк не определены операции присваивания и конкатенации, для С-строк нет операций сравнения, в отличие от других языков программирования. Для этих операций созданы специальные функции в библиотеке *cstring*.

- `strlen(строка)` – возвращает целое число, равное длине строки без учета нуля-символа;
- `strcpy(строка1, строка2)` – копирует значение строка2 в переменную строка1;
- `strcat(строка1, строка2)` – присоединяет строковое значение строка2 к окончанию значения строковой переменной строка1;
- `strcmp(строка1, строка2)` – если строка1 и строка2 совпадают, возвращает 0, если строка1 меньше, чем строка2, возвращает отрицательное число, если строка1 больше строка2, возвращает положительное число.

Для С-строк существуют функции преобразования строк в числа, которые находятся в библиотеке *cstdlib*:

- `atoi(строка_цифр)` – возвращает значение типа `int`;
- `atol(строка_цифр)` – возвращает значение типа `long`;
- `atof(строка_цифр_с_разделителем_точка)` – возвращает значение типа `double`.

Если строковый аргумент не допускает такого преобразования, функция возвращает нулевое значение.

Работа с С-строками достаточно сложная, кроме того, для этих строк работа с существующими функциями не всегда корректна, а обнаружить эту опасность достаточно сложно. Поэтому мы будем работать с другим видом строк – с классом `string`, который похож на строки в других языках программирования.

Класс `string`

Для работы с классом `string` требуется подключать библиотеку `string`. Для класса `string` определена операция присваивания (=) копирование одной строки в другую и операция конкатенации (+) – сцепления строк:

```
s3=s1+s2
```

```
s="Пример"+" "+"текста"
```

Описание строковых переменных:

```
string s;
```

```
string s1("Программирование");
```

```
string s2="Пример ввода"
```

При объявлении переменной `s` она инициализируется пустой строкой, при объявлении переменной `s1` она инициализируется значением заданного аргумента: "Программирование", а `s2` - инициализируется значением "Пример ввода".

Ввод –вывод строк

С их помощью операторов `<<` и `>>` ввод и вывод объектов типа `string` выполняется в том же виде, что и ввод-вывод С-строк. Заметим, что оператор `>>`

точно так же игнорирует пробелы, как и обычный оператор `>>` при использовании со стандартными типами, например с типом `int`. Если в строке встречаются пробелы, то перегруженный оператор `>>` считывает не всю строку, а только ту ее часть, которая идет до пробела (игнорируя сам пробел). Таким образом, если ввод осуществляется с помощью оператора `>>`, то программа считывает не строки, а слова. Под **словом** в данном случае подразумевается любая строка, в которую не входят пробельные символы. Чтобы прочитать всю C-строку используется функция-член класса `istream` под названием `getline`. Соответствующая функция, предназначенная для использования с объектами класса `string`, тоже называется `getline`, но она не является ни членом класса `istream`, ни членом класса `string`. Это отдельная функция; ее первый аргумент — объект типа `istream`, второй — объект типа `string`, а третий (если он есть) — символ, появление которого во входном потоке приводит к прекращению вывода. По умолчанию таким признаком завершения является символ `'\n'`. Пример использования функции `getline` приведен далее.

```
#include <iostream>
#include <string>
using namespace std;
// ...
string strl; // объявление пустой строки
getline (cin, strl); // Вносит в strl все, что вводится в строке,
                    // вплоть до символа '\n'. Символ '\n'
                    // удаляется из ввода и отбрасывается.
```

У функции `getline`, использующейся с объектами класса `string`, есть два прототипа:

```
strings getline(istream& ins, string& strVar, char delimiter);
и
strings getline(istream& ins, string& srVar);
```

При работе первой версии этой функции выполняется чтение символов из объектов класса `istream`, заданных в качестве первого аргумента функции, и вставка этих символов в строковую переменную. Это происходит до тех пор, пока в потоке не встретится символ-ограничитель (`delimiter`). Он удаляется из ввода и отбрасывается. Во второй версии функции роль ограничителя играет символ `'\n'`; в остальном она работает точно так же, как и первая.

Функция-член `ignore`

В конструкциях вида `cin>> intvar` (где `intvar`— переменная типа `int`) все, введенное после числа, продолжает оставаться в потоке и быть доступным следующей операции ввода (включая символ `'\n'`). (Данное утверждение справедливо для переменных любого типа, которые извлекаются из входного потока.) Это может привести к нежелательному поведению функции `getline`. Одно из возможных решений этой проблемы является использование следующей функции-члена класса `cin`:

```
istream& ignore(int count, char delimiter);
```

Функция либо считает count символов, либо будет считывать их до тех пор, пока ей не встретится символ-ограничитель delimiter (в зависимости от того, какое событие наступит раньше), а затем просто проигнорирует считанное.

Потеря ввода при смешивании cin>> variable и getline

Используя функцию getline, старайтесь свести к минимуму ввод с помощью инструкций вида cin >> variable, так как могут возникнуть проблемы. При использовании конструкции cin >> variable опускаются все пробелы, стоящие перед вводом, а символ '\n' остается в потоке. Функция же getline считывает все подряд, включая символ '\n', и останавливается. Если вызвать функцию getline после cin >> x, то она тут же встречает несчитанный символ '\n' и полагает, что имеет дело с пустой строкой. Если вы видите, что ваша программа пропускает некоторые вводимые данные, проверьте, не смешаны ли в ней два вида ввода. В такой ситуации можно использовать функцию-член класса istream с именем ignore. Например, вызов cin.ignore(10000, '\n'); приведет к считыванию 10000 символов, которые будут успешно проигнорированы (конечно же, если раньше встретится символ '\n', то функция прекратит работу, как только считает его).

Для объектов класса string используются как и в массивах квадратные скобки. Благодаря этому выражение с индексом s[i] ведет себя точно так же, как и для C-строк: оно вычлняет из строкового значения, хранящегося в объекте типа string с именем s, один символ. Таким образом, выражение s[i] можно использовать для извлечения символов точно так же, как и символы из обычного символьного массива.

Важно помнить о том, что в перегрузку квадратных скобок для объектов класса string не входит проверка выхода значений индексов за дозволенные пределы. Это означает, что правильность использования значений индексов не проверяется. Проверку такого типа выполняет функция-член под названием at, которая в основном работает точно так же, как и квадратные скобки, но все же имеет два отличия. Для нее используются обозначения, принятые для функций, поэтому вместо s[i] нужно использовать s.at(i). Второе отличие состоит в том, что функция-член at проверяет, является ли значение аргумента i правильным индексом. В двух приведенных ниже примерах фрагментов кода предпринимается попытка выйти за пределы строки. В первом из них сообщение об ошибке не выведется, несмотря на то что в нем происходит обращение к переменной с несуществующим индексом:

```
string str("Mary");  
cout << str[6] << endl;
```

Однако во втором фрагменте произойдет аварийный останов программы, из чего можно сделать вывод, что в программе имеется ошибка.

```
string str("Mary");  
cout << str.at(6) << endl;
```

Отдельно взятый символ строки можно изменить, присвоив переменной с индексом name[i] новое значение. То же можно сделать и с помощью функции-члена at(pos). Например, чтобы заменить третий символ строки s символом 'X', можно использовать следующий фрагмент кода:

s.at(2) = 'X'; или s[2] = 'X';

Как и в обычном массиве, индексация позиций символов в объектах типа string начинается с 0, поэтому индекс третьего символа строки равен 2.

Часто используемые функции-члены класса string

Функции	Пояснение
s[i]	Обращение для чтения и записи к символу с индексом i
s.substr(pos, len)	Возвращает подстроку вызывающего объекта, начинающуюся с позиции pos и имеющую длину len (доступ только для чтения)
s.c_str()	Предоставляет доступ только для чтения к C-строке, представленной объектом str
s.at(i)	Обращается для чтения и записи к символу строки s, который имеет индекс i
s1=s2;	Выделяет для строки s1 объем памяти, равный длине строки s2, и инициализирует строку s1 значением строки s2
s+=s2;	Символы строки s2 добавляются в конец строки s1, для которой выделяется необходимый объем памяти
s.empty()	Если строка s является пустой, возвращает true, если s не пустая— false
s.clear()	Удаляет все символы в s
s.insert(pos, s2)	Помещает строку s2 в строку s, начиная с позиции pos
stoi(s)	Возвращает значение типа int (s – строка цифр)
stod(s)	Возвращает значение типа double (s – строка цифр и разделитель точка)
to_string	Преобразовывает значение в значение string .
s1==s2 s1!=s2	Проверяет, равны строки или нет; возвращает соответствующее логическое значение
s1<s2, s1>s2, s1<=s2 s1>=s2	Лексикографическое сравнение строк
s.find(s1)	Возвращает индекс начала подстроки s1, входящей в строку s
s.find(s1,pos)	Возвращает индекс начала подстроки s1, входящей в строку s; поиск начинается с позиции pos
s.length() s.size()	Возвращает текущее количество символов в строке s

Примеры

Пример 1.

Дан текст. Разделить текст на слова и записать их в массив.

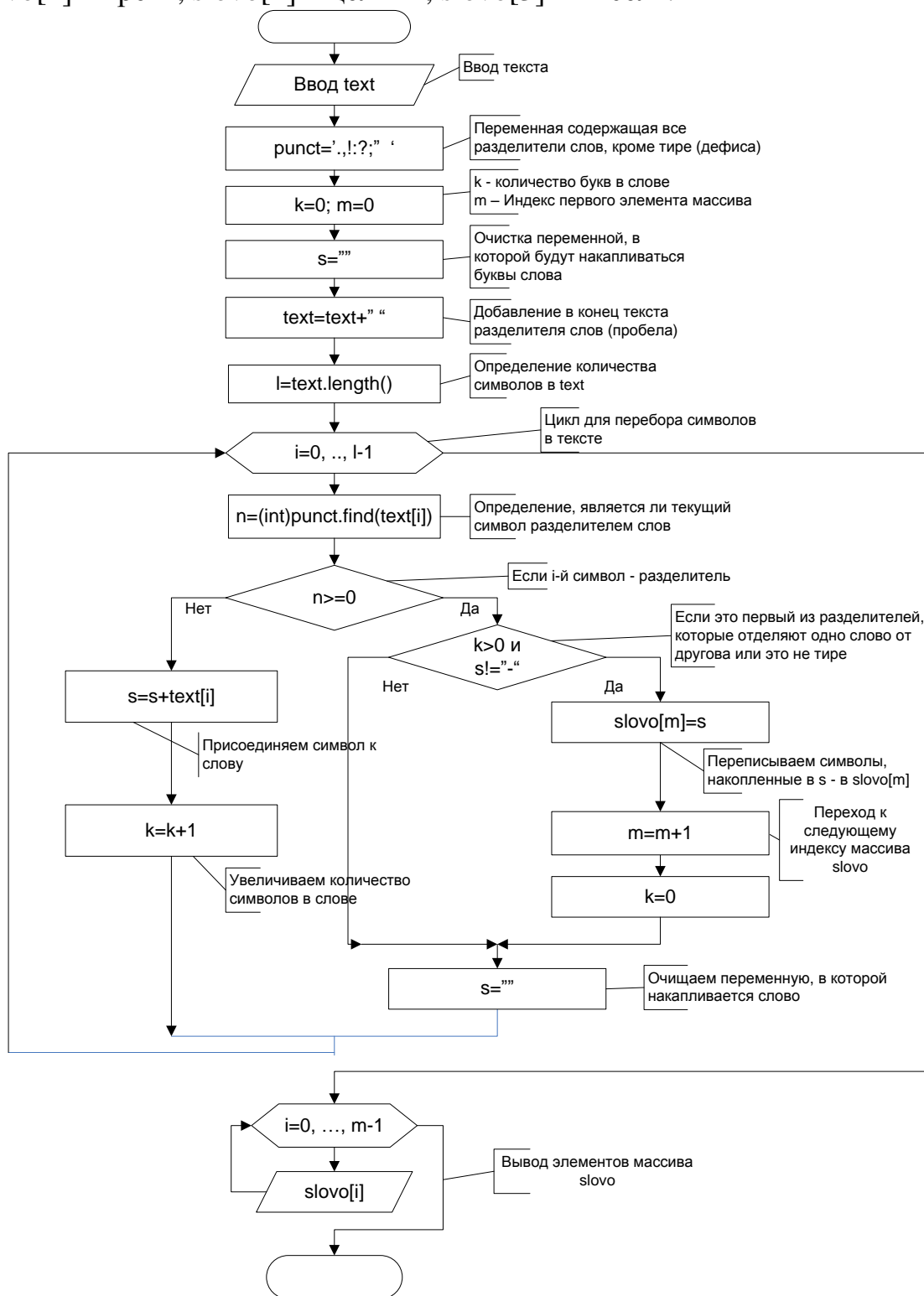
Исходные данные: text – строкового типа.

Результат: массив слов slovo.

Просматриваются все символы текста и, если встречаются символы разделители слов - достигнут конец слова, это слово записывается массив и готовится для записи новый элемент массива.

Тестовый пример:

при `text="Сумма трех целых чисел"` получаем массив: `slovo[0]="Сумма"`, `slovo[1]="трех"`, `slovo[2]="целых"`, `slovo[3]="чисел"`.



Код программы

```

#include <iostream>
#include <string>
#include <conio.h>
#include <windows.h>
  
```



```

using namespace std;
void main()
{SetConsoleCP(1251);
 SetConsoleOutputCP(1251);
 string text, s;
 string punct="., !?:";
 int k=0,n,l,m=0;
 string slovo[20];
 cout<<"Введите текст ";
 getline(cin, text);
 cout<<text<<endl;
 s="";
 text=text+" ";
 l=text.length();
 for(int i=0; i<l;i++)
 {n=(int)punct.find(text[i]);
  if (n>=0)
      {if (k>0 && s!="-") {slovo[m]=s;
                           m++;
                           k=0;
                           }
        s="";
      }
  else {s+=text[i];
        k++;
      }
  }
 for(int i=0;i<m;i++)
  cout<<slovo[i]<<endl;
 _getch();
}

```

Пример 2.

Дан текст, содержащий русские буквы. Заменить в тексте все строчные буквы заглавными.

В C++ нет функций, которые преобразовывали бы строчные буквы в заглавные, тем более, если это русские буквы. Для решение этой задачи создадим функцию преобразования строчных русских букв в заглавные:

```
string upperRu(string str)
```

Код программы, использующий эту функцию, будет иметь вид:

```

# include <iostream>
# include <string>
#include <conio.h>
# include <windows.h>
using namespace std;
string upperRu(string str);
void main()
{
  SetConsoleCP(1251);
  SetConsoleOutputCP(1251);
  string text1, text2;
  cout<<"Введите текст";
  getline(cin,text1);

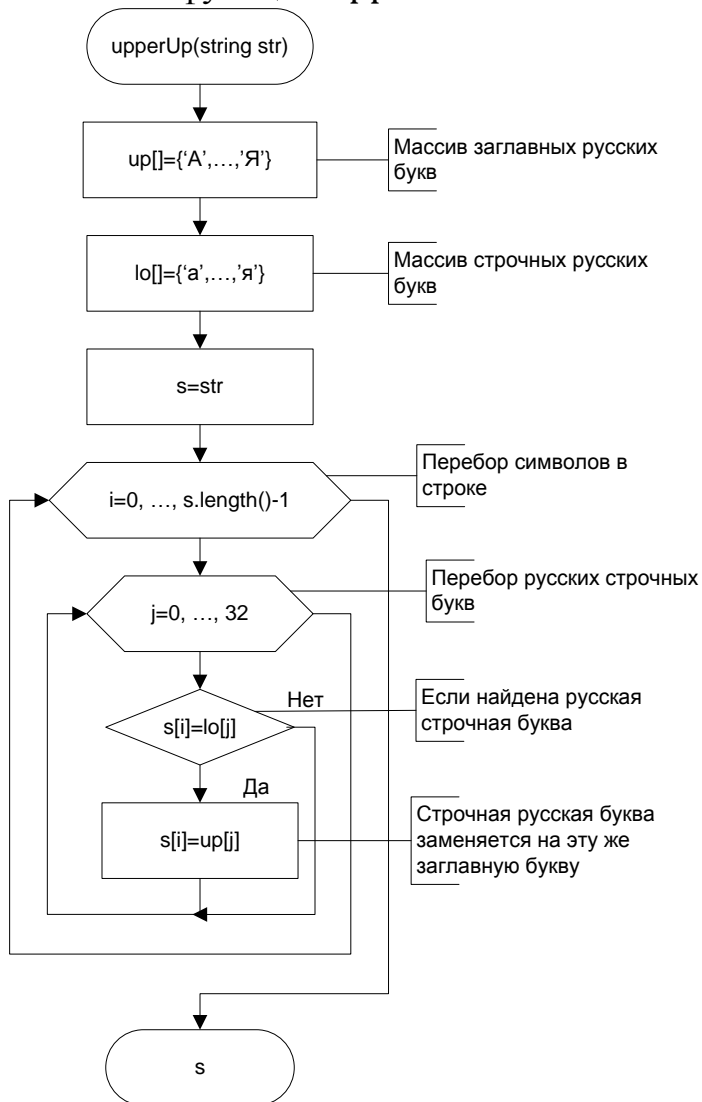
```

```

text2=upperRu(text1);
cout<<text2<<endl;
getch();
}

```

Блок схема функции upperRu:



Пример 3.

Дан текст. Получить из этого текста новую строку, в которой отсутствуют пробелы.

Исходные данные: Строка s – типа string.

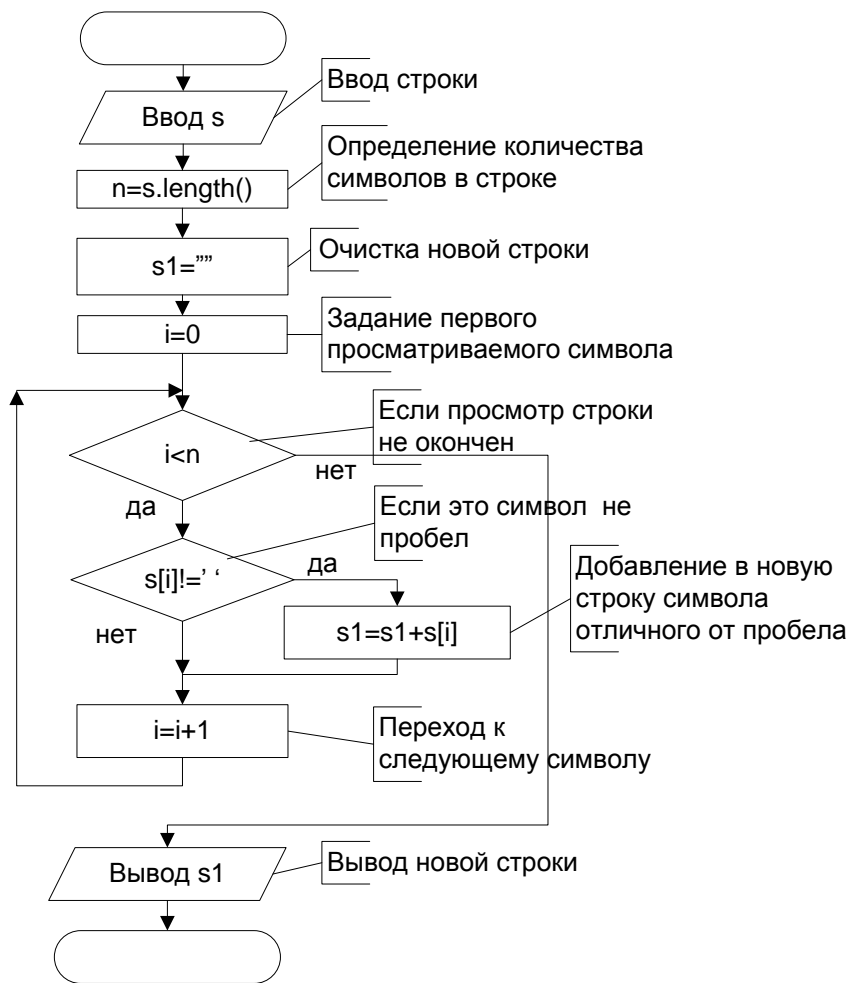
Результат: Строка s1 без пробелов.

Следует обратить внимание, что после удаления символа индекс символа i не изменяется из-за сдвига символов после удаления.

Тестовый пример:

при s="Разделение строки на отдельные слова"

конечный результат: s1="Разделениестрокинаотдельныеслова"



Задание 2. Написать и отладить программу для примера 3

Контрольные вопросы

1. Чем C-строки отличаются от обычного массива символов.
2. В чем заключается недостаток использования оператора ввода `>>` для строк.
3. Почему не рекомендуется смешивать для ввода оператор `>>` и `getline`. Как избежать ошибок при таком смешивании.
4. Какие преимущества имеет класс `string` перед C-строками.
5. Какую библиотеку надо подключать? чтобы использовать функции для объединения C-строк.
6. Можно ли в примере 3 `s[i]` сравнить с `" "`.
7. В какой библиотеке описан класс `string`.
8. Какие функции надо использовать, чтобы преобразовать строку в числовой тип.

Индивидуальные задания

- 1 Дана строка. Определить количество слов, равных самому короткому слову в строке.
- 2 Дана строка. Определить количество слов, равных самому длинному слову в строке.
- 3 Дана строка. Поменять местами самое длинное и самое короткое слово.

- 4** Дана строка. Сколько слов имеют длину, равную трём символам?
- 5** Дана строка. Удалить слова со второго по четвёртое.
- 6** Дана строка. Перевернуть каждое слово в строке. Порядок слов не менять.
- 7** Дана строка. Определить количество слов в данной строке и заменить все разделители слов (пробелы) на знак “ + ”.
- 8** Дана строка. Поменять местами первое и последнее слова.
- 9** Дана строка. Определить количество слов, длина которых равняется чётному числу.
- 10** Дана строка. Определить количество слов, длина которых равняется нечётному числу.
- 11** Дана строка. Вывести слова в обратном порядке, начиная с последнего.
- 12** Дан текст. Сколько в тексте слов начинаются и заканчиваются на одну и ту же букву.

Литература

1. Иванова, Г. С. Технология программирования : учебник для вузов [ГрифУМО] / Г. С. Иванова. - 3-е изд., стер. - Москва : КноРус, 2013. - 333 с.
2. Головин, И. Г. Языки и методы программирования : учебник для вузов [Гриф УМО] / И. Г. Головин, И. А. Волкова. - Москва : Академия, 2012. 303 с.
3. Павловская Т. А. С/С++. Программирование на языке высокого уровня [Текст]: учеб. для вузов / Т.А. Павловская. – СПб.: Питер, 2011.– 432 с.
4. Парфилова, Н. И. Программирование. Основы алгоритмизации и программирования : учебник для вузов [Гриф МГТУ им. Н. Э. Баумана] / Н.И. Парфилова, А. Н. Пылькин, Б. Г. Трусов ; под ред. Б. Г. Трусова.. Москва : Академия, 2014. - 239 с. [и предыдущие издания]