Основы программирования. Алгоритмизация и программирование на C++

Основы алгоритмизации

***Алгоритм* – это конечная последовательность действий, позволяющая по заданным исходным данным получить результат решения задачи.**

Алгоритм разбивается на шаги. Для каждого шага есть конкретный исполнитель.

Исполнитель алгоритма может быть человеком или автоматом.

Вид алгоритма зависит от исходных данных.

Результат работы алгоритма с одними и теми же исходными данными не зависит от исполнителя.

Основы алгоритмизации

Алгоритм имеет две характеристики.

1. ***Конечность, или результативность***. Алгоритм приводит к получению результата за конечное число шагов..

2. ***Однозначность, или определенность***. При одинаковых входных данных алгоритм выдает одинаковый результат.

Алгоритм также обладает следующими свойствами.

1. ***Массовость, или универсальность***. Алгоритм выдает результат при любых однотипных входных данных.

2. ***Модульность, или дискретность***. Алгоритм можно представить в виде последовательности более элементарных алгоритмов.

Проектирование сверху вниз

Основной метод создания алгоритмов — ***проектирование, или программирование, сверху вниз, или пошаговая детализация.*** Он заключается в разбиении исходной задачи на последовательность нескольких меньших подзадач. Эти подзадачи, в свою очередь, тоже распадаются на подзадачи и т. д. до тех пор, пока не останутся только элементарные алгоритмы.

При программировании сверху вниз алгоритмы и данные делятся на относительно независимые части, называемые ***модулями***. Некоторые из модулей являются стандартными и поставляются в составе языков программирования, например, вычисление элементарных математических функций квадратный корень, логарифм, синус и т. д.

Проектирование сверху вниз

Главные модули все равно приходится проектировать программистам. Таким образом, алгоритм является *деревом модулей*:

одни модули вызывают другие модули, начиная с самого верхнего первого модуля, называемого *корневым модулем, или головной программой*.

При проектировании сколько-нибудь больших алгоритмов невозможно держать в памяти одновременно детали всех модулей алгоритма. Если модуль составлен правильно, то с ним можно обращаться как с черным ящиком.

Принцип черного ящика означает, что не имеет значения, *как* модуль выполняет свою функцию, какие алгоритмы скрыты у него внутри. Это не важно для остальных модулей алгоритма. Для модулей, которые обращаются к этому модулю, имеет значение только следующее:

1) какова функция модуля, т. е. *что* он делает;

2) описание входных и выходных данных модуля.

Правильное проектирование алгоритмов позволяет абстрагироваться от внутренней структуры модулей и рассматривать при сборке полного алгоритма только функции модулей.

Структурное программирование

Структурное программирование позволяет проектировать алгоритмы только из трех элементарных алгоритмов. Каждый модуль является иерархией этих элементарных алгоритмов. Алгоритм, состоящий только из этих трех элементарных алгоритмов, присутствующих на всех его уровнях, называется *структурным*.

Основная теорема структурного программирования утверждает, что любой алгоритм можно преобразовать к структурному виду.

Тремя элементарными структурными алгоритмами являются следующие.

1. *Следование, или цепочка, или составная инструкция*.
2. *Выбор, или ветвление, или условная инструкция*.
3. *Цикл, или возврат, или циклическая инструкция*.

Объектно-ориентированное программирование

Компьютер = аппаратура + программы, а программа = алгоритм + данные.

Объектно-ориентированное программирование (ООП) организует данные и алгоритмы, обрабатываемые программой. При этом программист создает формы данных и алгоритмы, соответствующие основным характеристикам решаемой проблемы. Модели данных и алгоритмы, их обрабатывающие, называются *классами*, а *объекты* — это конкретные их представители, используемые в программе. Из общих объектов создаются другие, более специализированные. Механизм создания таких подобъектов называется *наследованием*. В итоге данные программы представляют из себя *объектную модель* — дерево объектов, начиная с самого верхнего наиболее абстрактного и общего объекта.

Визуальное программирование

Визуальное программирование существенно облегчает программирование для графического интерфейса типа Windows, который состоит из множества графических объектов: кнопок, окон, меню и т. д. Система *визуального программирования* предоставляет программисту:

1) готовую объектную визуальную модель, содержащую множество графических диалоговых объектов (кнопки, окошки, меню и т. д.) и программных модулей, которые их реализуют;
2) среду визуального программирования, в которой графические диалоговые объекты, которые будут определять интерфейс программы, просто размещаются на экране мышью.

Язык программирования, программа

Чтобы создать компьютерную программу, нужно записать алгоритм по специальным правилам на *языке программирования*, который понимает и человек, и компьютер. Такая запись называется ***исходным текстом программы**, или программой*. Программа пишется в простом текстовом редакторе. Затем программа переводится в машинные коды, выполняемые процессором компьютера, специальной программой-переводчиком.

Компилятор — программа-переводчик с языка программирования в машинные коды, а процесс перевода — это *компилирование* программы. Комплекс программ, включающий компилятор и другие средства написания программ, называется *системой программирования*.

Возможны два способа компиляции.

Первый способ называется ***трансляцией*** и заключается в компилировании сразу всей программы в машинные коды. Затем строится *выполняемый файл*, содержащий эту программу. И только потом программа выполняется путем запуска выполняемого файла. Благодаря трансляции получается более быстрая по времени работы программа, но выполняемый файл имеет большой объем. Кроме того, выполняемый файл запускается только на компьютере того типа, где программа была транслирована.

Второй способ. При ***интерпретации*** компьютер читает программу по одной строке и сразу выполняет эту строку. При интерпретации программу не надо переводить всю сразу в машинные коды, и поэтому она имеет маленький объем, равный объему исходного текста программы. Такая программа запускается на любом компьютере, на котором находится *интерпретатор или виртуальная машина*.

Сборщик, приложение

Большинство современных компиляторов работают в режиме трансляции. При трансляции модулей исходных текстов, оформленных специальным образом и называемых *подпрограммами*, получается набор оттранслированных подпрограмм. Подпрограмма, в которую входит корневой модуль, называется ***головной программой***.

Выполняемый файл, или приложение создается из этих подпрограмм с помощью еще одной специальной программы — ***сборщика или компоновщика***, или линковщика, или редактора связей. Сборщик связывает на уровне машинных кодов подпрограммы в цельную программу. Таким образом, получается дерево подпрограмм, начиная с головной программы. Эти подпрограммы при выполнении вызывают друг друга. Головная программа вызывает свои подпрограммы, те, в свою очередь, подпрограммы следующего уровня и т. д., пока вся программа не выполнится.

Ошибки программирования

Ошибки в программах бывают двух видов.

1. **Синтаксические ошибки** — несоответствие формальным требованиям языка программирования. На них указывает транслятор при трансляции и линковщик при сборке программы.
2. **Семантические ошибки** — смысловые ошибки; при них программа «работает», но работает неправильно. Поиск этих ошибок происходит с помощью логического анализа работы программы и ее тестирования.

Средства изображения алгоритмов

Основными изобразительными средствами алгоритмов являются следующие способы их записи:

- словесный;
- формульно-словесный;
- блок-схемный;
- псевдокод;
- структурные диаграммы;
- языки программирования.

Словесный – содержание этапов вычислений задается на естественном языке в произвольной форме с требуемой детализацией.

Рассмотрим пример словесной записи алгоритма. Пусть задан массив чисел. Требуется проверить, все ли числа принадлежат заданному интервалу. Интервал задается границами A и B.

п.1 Берем первое число. На п.2.

п.2 Сравниваем: выбранное число принадлежит интервалу; если да, то на п.3, если нет – на п.6.

п.3 Все элементы массива просмотрены? Если да, то на п.5, если нет – то на п.4.

п.4 Выбираем следующий элемент. На п.2.

п.5 Печать сообщения: все элементы принадлежат интервалу. На п.7.

п.6 Печать сообщения: не все элементы принадлежат интервалу. На п.7.

п.7 Конец.

При этом способе отсутствует наглядность вычислительного процесса, т.к. нет достаточной формализации.

Формульно-словесный – задание инструкций с использованием математических символов и выражений в сочетании со словесными пояснениями.

Например, требуется написать алгоритм вычисления площади треугольника по трем сторонам.

п.1 – вычислить полупериметр треугольника

$p=(a+b+c)/2$. К **п.2**.

п.2 – вычислить

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$

К **п.3**.

п.3 – вывести S, как искомый результат и прекратить вычисления.

При использовании этого способа может быть достигнута любая степень детализации, более наглядно, но не строго формально.

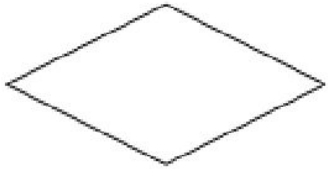
Блок-схемный – это графическое изображение логической структуры алгоритма, в котором каждый этап процесса переработки данных представляется в виде геометрических фигур (блоков), имеющих определенную конфигурацию в зависимости от характера выполняемых операций.

Блок-схемы могут быть традиционные и структурированные.

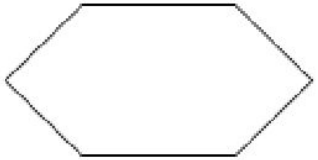
Основные символы блок-схем:

- ввод-вывод;

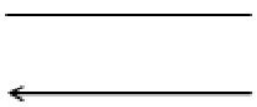
- процесс (выполнение операций или группы операций);



- решение (выбор направления);



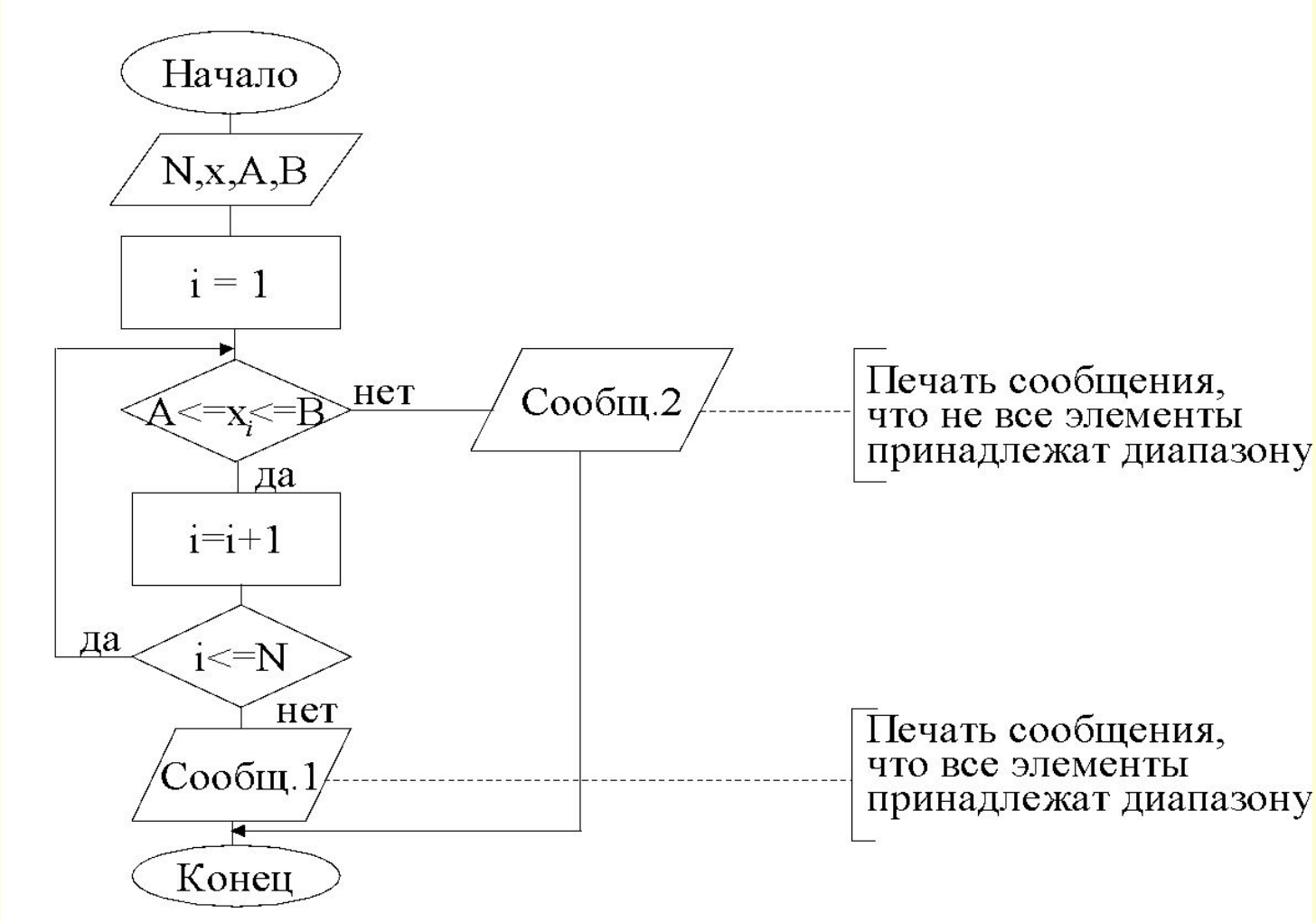
- модификация (организация цикла);



- линии потока данных

- пуск-останов (начало, конец программы).

Рассмотрим пример блок-схемы той же задачи, для которой приведен словесный алгоритм.



Псевдокод - позволяет формально изображать логику программы, не заботясь при этом о синтаксических особенностях конкретного языка программирования. Обычно представляет собой смесь операторов языка программирования и естественного языка. Является средством представления логики программы, которое можно применять вместо блок-схемы. Запись алгоритма в виде псевдокода:

Выбираем первый элемент ($i=1$)

IF $A > x_i$ или $x_i > B$ THEN

печать сообщения и переход на конец

ELSE

переход к следующему элементу($i = i + 1$)

IF массив не кончился ($i <= n$) THEN

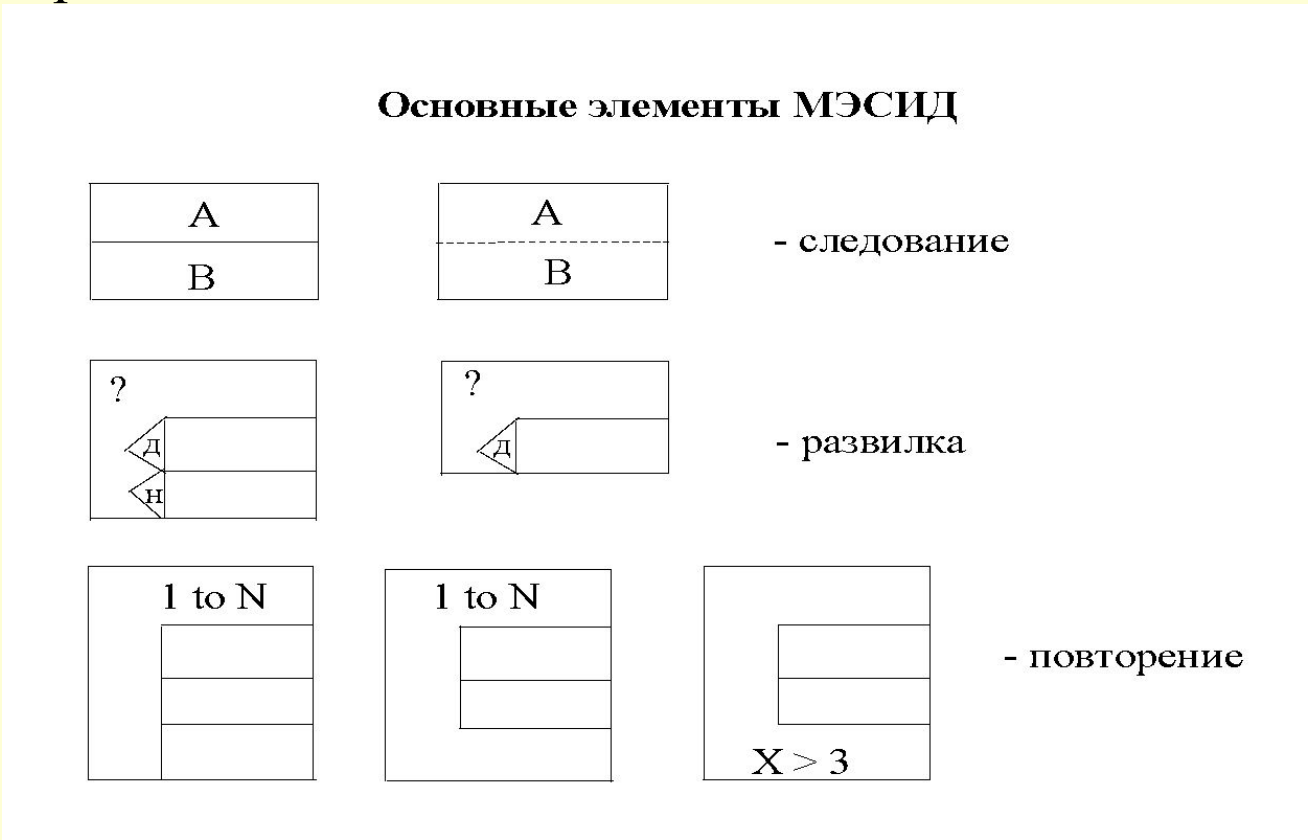
переход на проверку интервала

ELSE

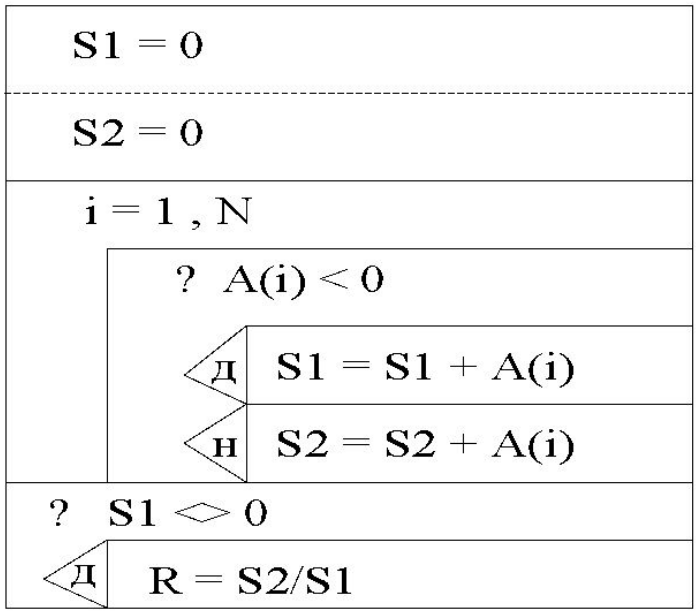
печать сообщения, что все элементы входят в интервал

Конец

Структурные диаграммы - могут использоваться в качестве структурных блок-схем, для показа межмодульных связей, для отображения структур данных, программ и систем обработки данных. Существуют различные структурные диаграммы: диаграммы Насси-Шнейдермана, диаграммы Варнье, Джексона, МЭСИД и др.



Рассмотрим пример использования диаграмм МЭСИД.
Задан одномерный массив из положительных и отрицательных чисел. Требуется определить частное от деления суммы положительных элементов на сумму отрицательных элементов этого массива. Справа от диаграммы приводятся соответствующие операторы языка Паскаль.




```
. . . S1:=0;
      S2:=0;
FOR I :=1 TO N DO
      IF A[I] < 0 THEN
                  S1:=S1 + A[I]
                     ELSE
                  S2:=S2 +A[I] ;
IF S1 <> 0 THEN
                  R:= S2/S1; . . .
```

Паскаль

Базовые канонические структуры алгоритмов

Доказано, что любую программу можно написать, используя комбинации трех управляющих структур:

- ***следования или последовательности операторов;***
- ***развилки или условного оператора;***
- ***повторения или оператора цикла.***

Программа, составленная из канонических структур, будет называться регулярной программой, т.е. иметь 1 вход и 1 выход, каждый оператор в программе может быть достигнут при входе через ее начало (нет недостижимых операторов и бесконечных циклов). Управление в такой программе передается сверху-вниз.

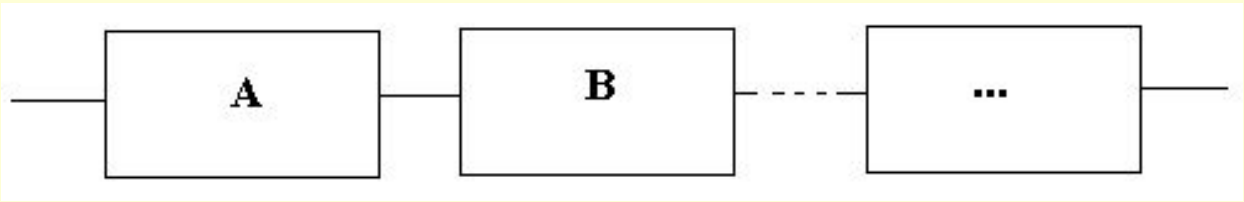
Базовые канонические структуры алгоритмов

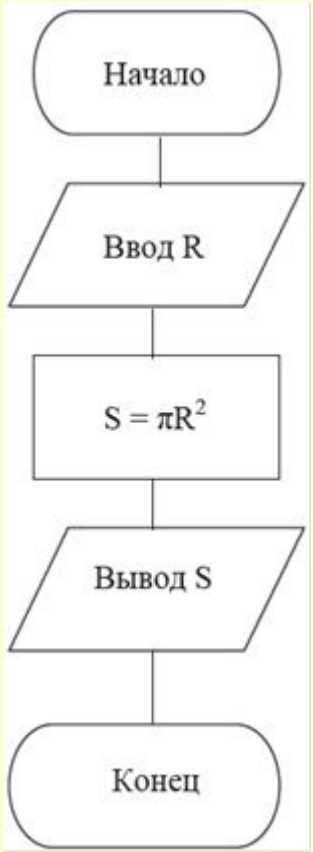
1. **Следование, линейная инструкция.**

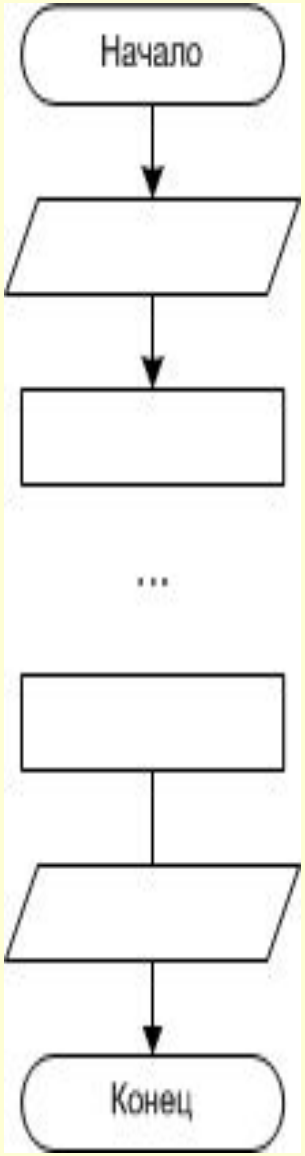
Линейная инструкция

Действия *A* и *B* могут быть:

- отдельным оператором;
- вызовом с возвратом некоторой процедуры;
- другой управляющей структурой.







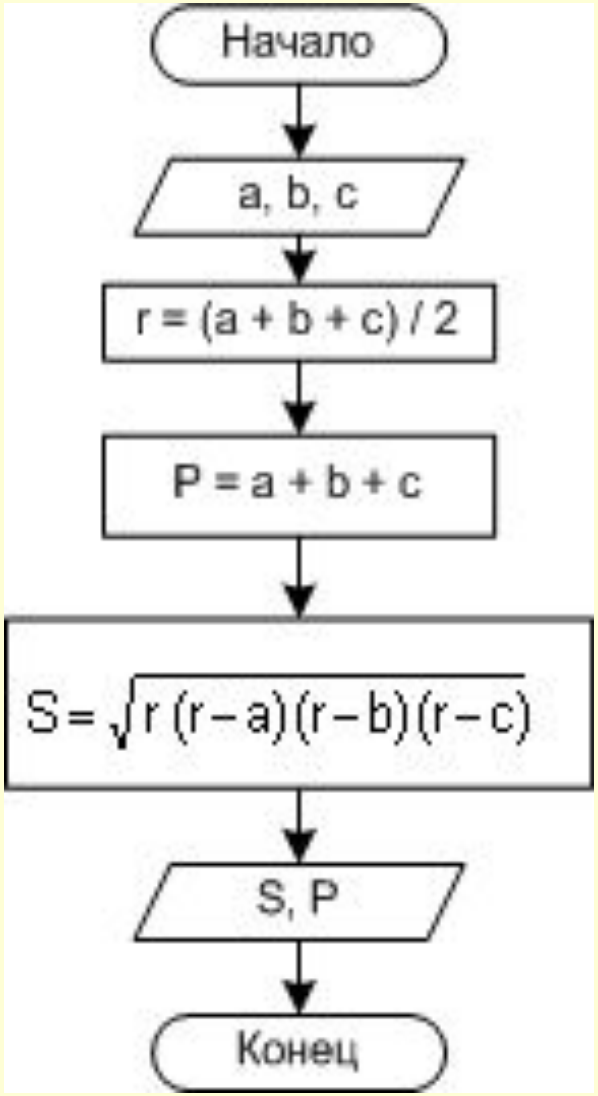
ПРИМЕР. Зная длины трех сторон треугольника, вычислить площадь и периметр треугольника.

Пусть **a, b, c** - длины сторон треугольника. Необходимо найти **S** - площадь треугольника, **P** - периметр. Для нахождения площади можно воспользоваться формулой Герона:

$$S = \sqrt{r(r-a)(r-b)(r-c)}$$

где **r** - полупериметр.

Входные данные: **a, b, c**.
Выходные данные: **S, P**.



ПРИМЕР.

Известны плотность и геометрические размеры цилиндрического слитка, полученного в металлургической лаборатории. Найти объем, массу и площадь основания слитка.

Входные данные:

R - радиус основания цилиндра,

h - высота цилиндра,

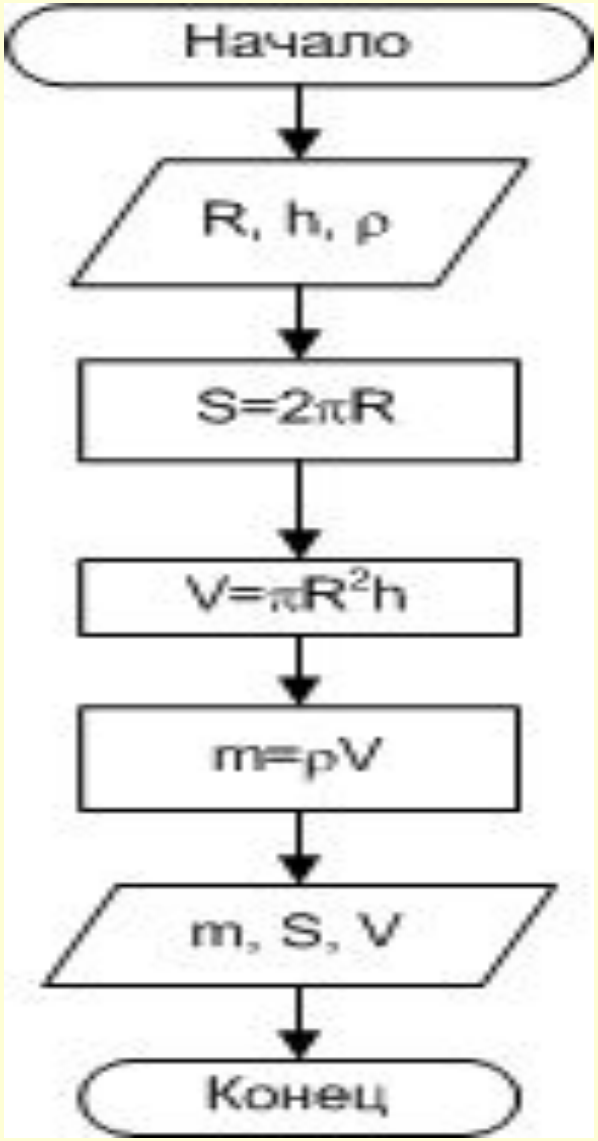
ρ- плотность материала слитка.

Выходные данные:

m - масса слитка,

V - объем,

S - площадь основания.



ПРИМЕР

Заданы длины двух катетов в прямоугольном треугольнике. Найти длину гипотенузы, площадь треугольника и величину его углов.

Входные данные:

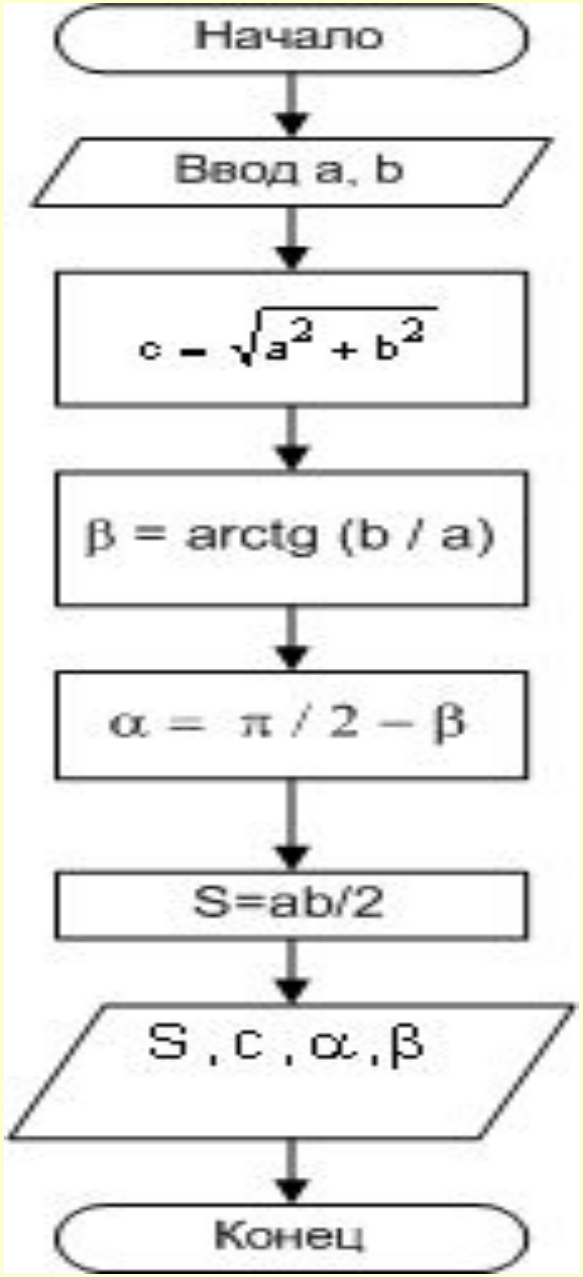
a, b - длины катетов.

Выходные данные:

c - длина гипотенузы,

S - площадь треугольника,

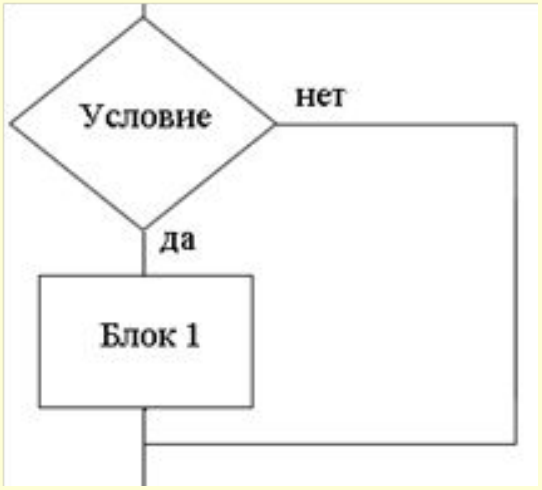
α, β - углы.

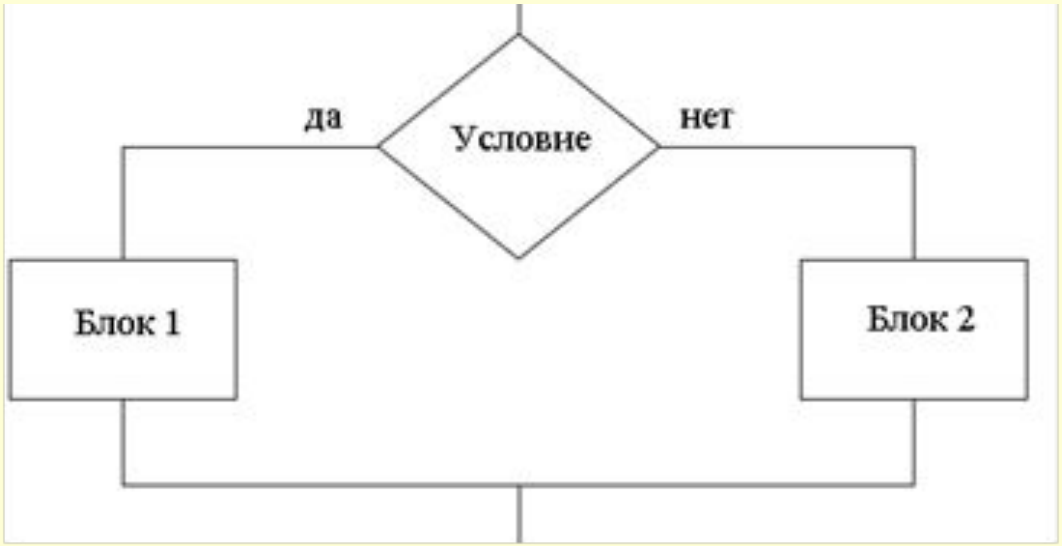


Базовые канонические структуры алгоритмов

2) Алгоритмы **разветвленной** структуры (*развилка)* применяются, когда в зависимости от некоторого условия необходимо выполнить либо одно, либо другое действие

Проверка P представляется *предикатом*, т.е. функцией, задающей логическое выражение или условие, значением которого может быть *истина* или *ложь*. Эта структура может быть неполной, когда отсутствует действие, выполняемое при ложном значении логического выражения.





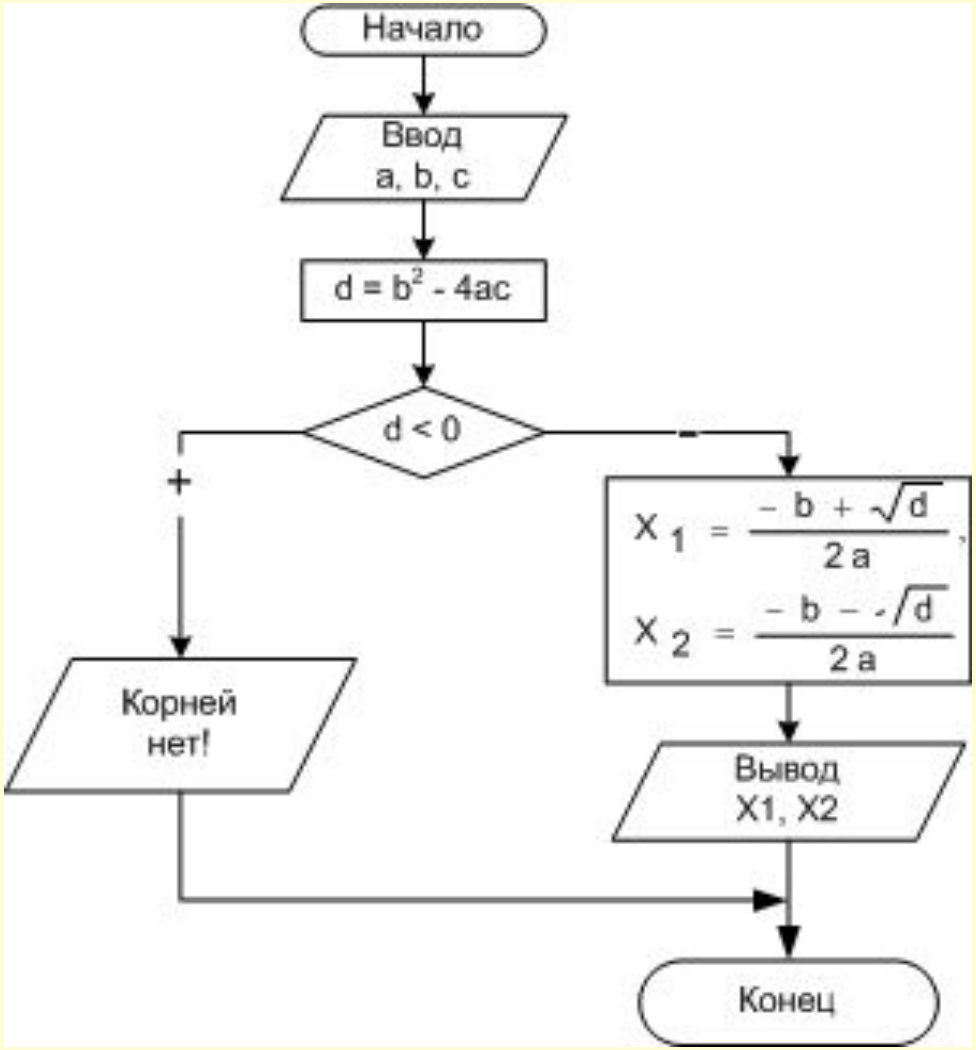
ПРИМЕР

Известны коэффициенты **a, b** и **c** квадратного уравнения.

Вычислить корни квадратного уравнения.

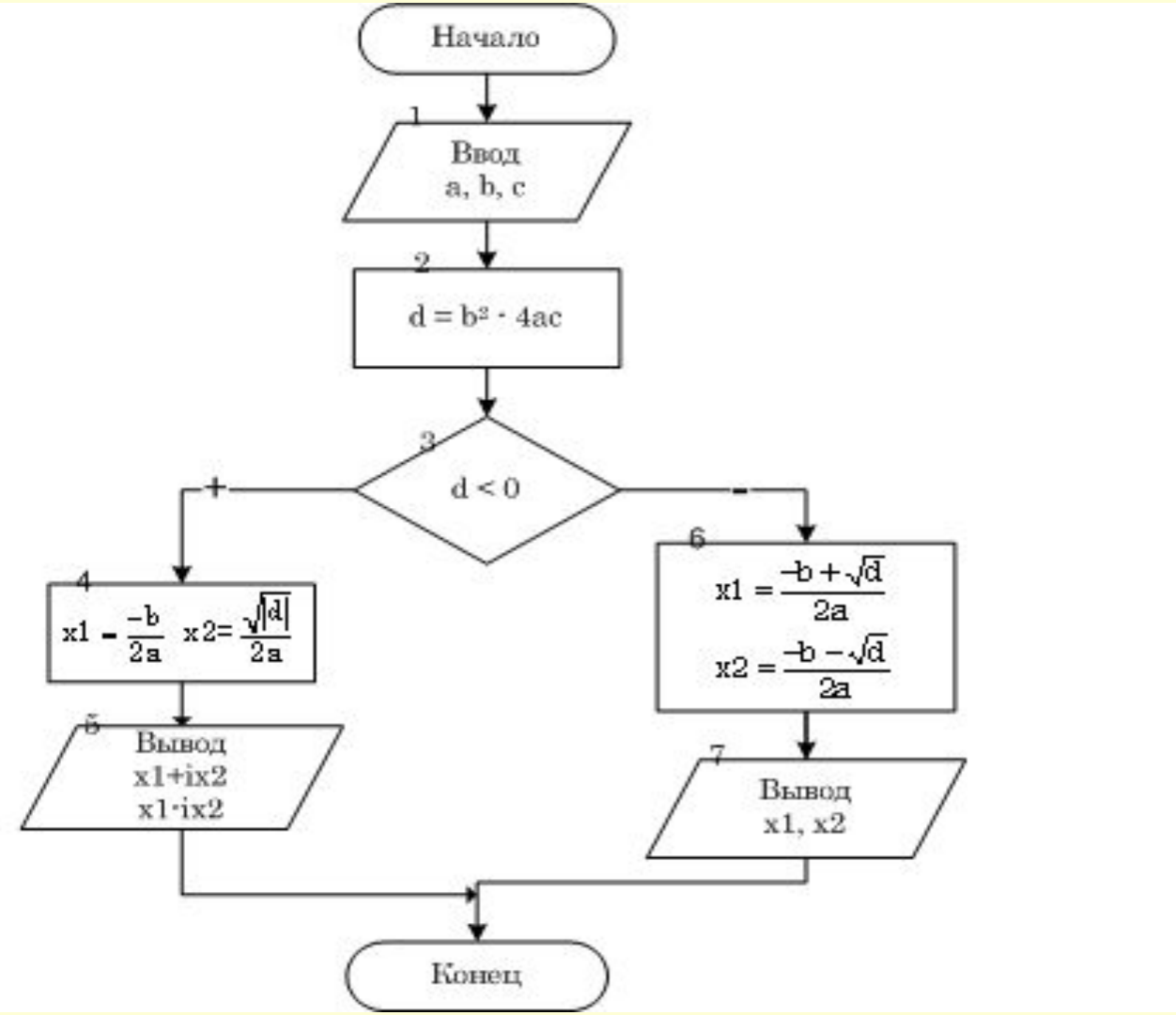
Входные данные: **a, b, c**.

Выходные данные: $\mathbf{x_1}$, $\mathbf{x_2}$.



ПРИМЕР

Составить программу нахождения действительных и комплексных корней квадратного уравнения.

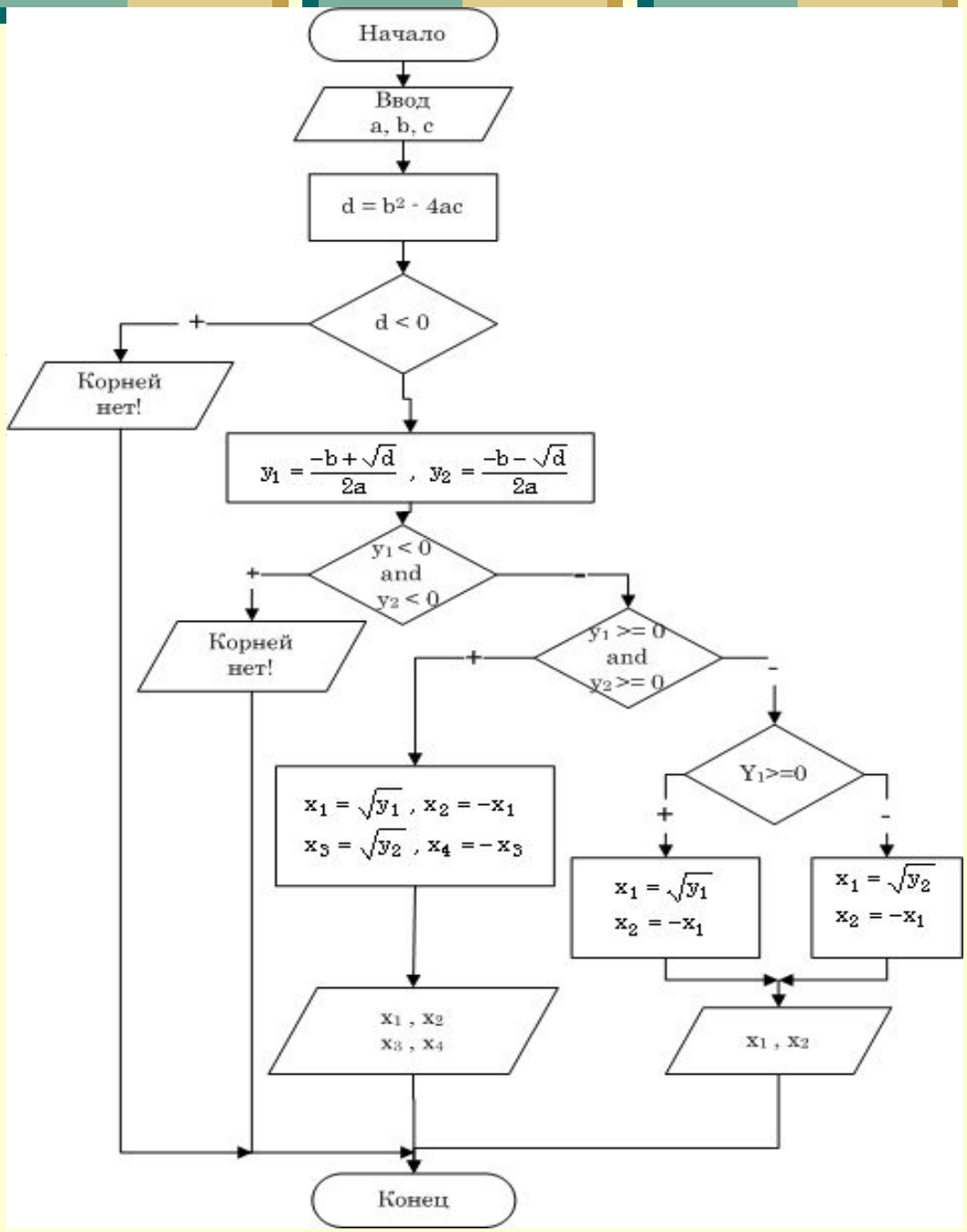


ПРИМЕР

Заданы коэффициенты **a, b** и **c** биквадратного уравнения $\mathbf{ax^4 + bx^2 + c = 0}$. Решить уравнение. Для решения биквадратного уравнения необходимо заменой $\mathbf{= x^2}$ привести его к квадратному решить это уравнение.

Входные данные: **a, b, c**.

Выходные данные: $\mathbf{x_1, x_2, x_3, x_4}$.

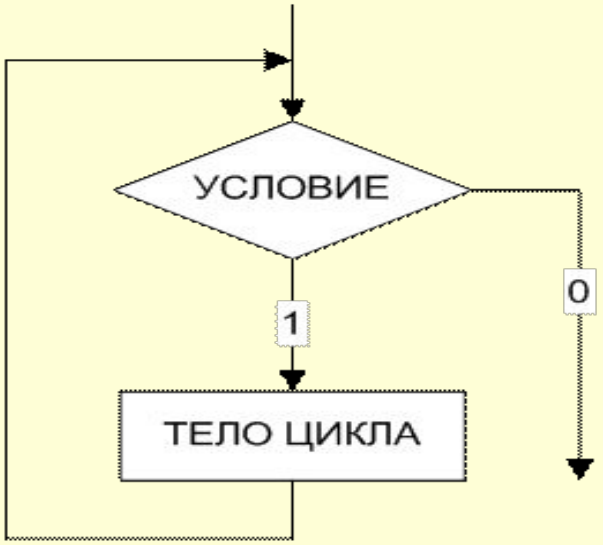


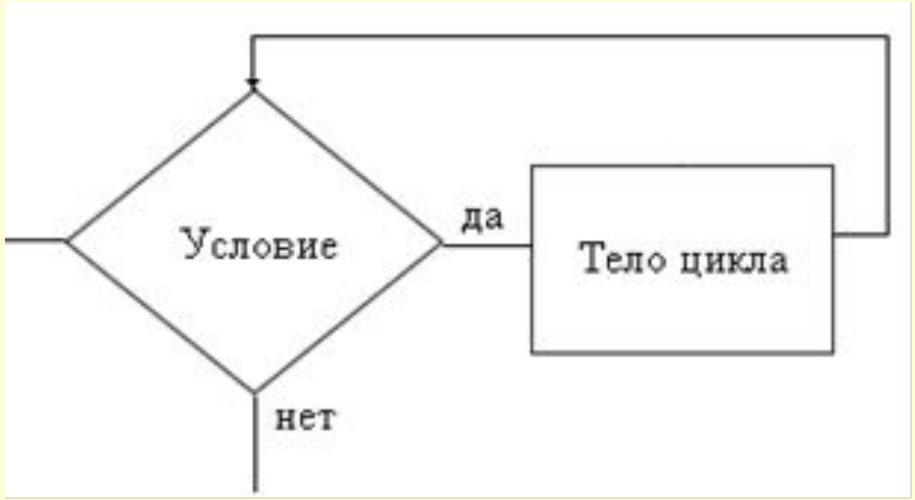
Базовые канонические структуры алгоритмов

3) Цикл (повторение)

Циклом в программировании называют повторение одних и тех же действий (шагов). Последовательность действий, которые повторяются в цикле, называют ***телом цикла.***

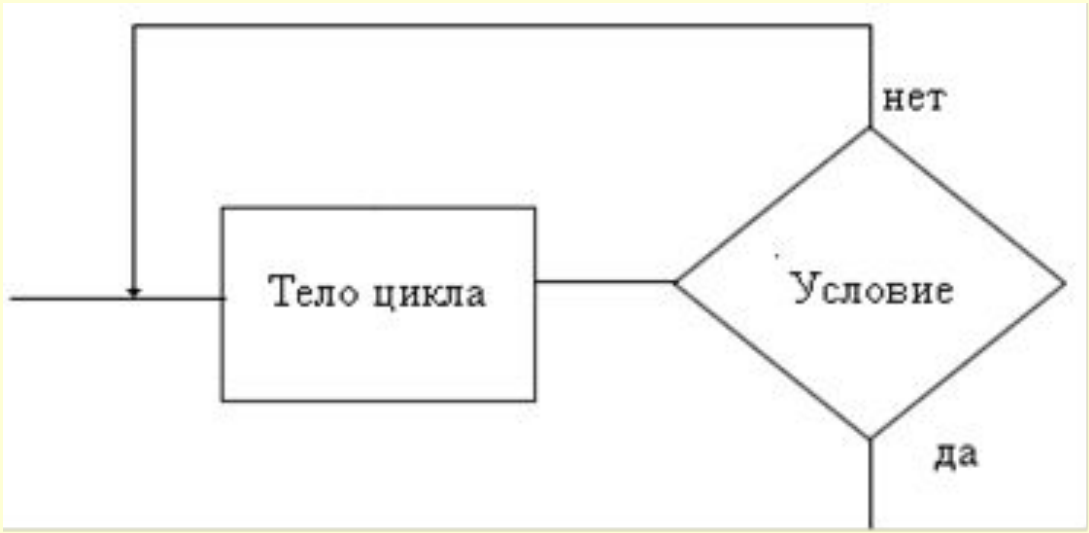
Цикл с предусловием



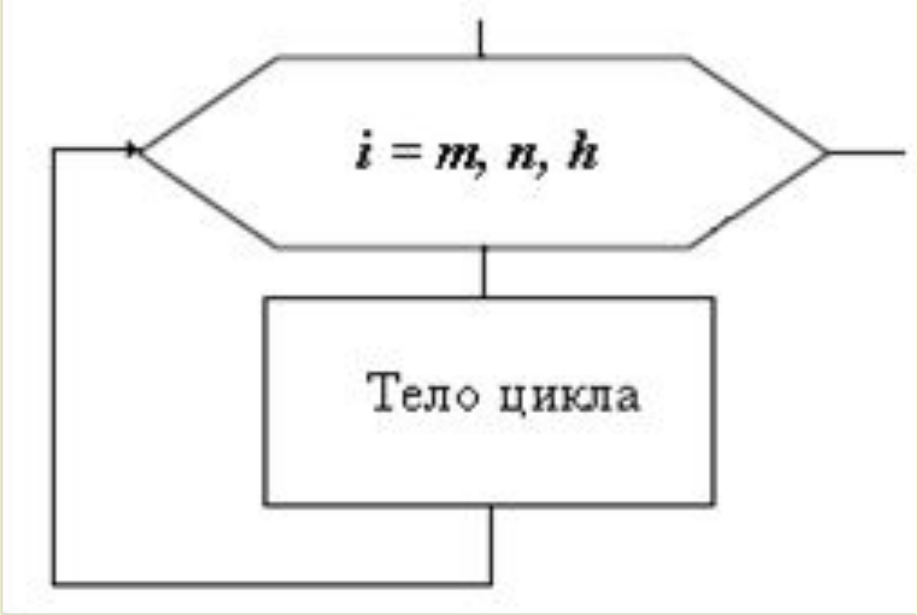


Цикл с постусловием

Тело цикла всегда выполняется хотя бы один раз.
Тело цикла перестает выполняться, как только предикат становится истинным.



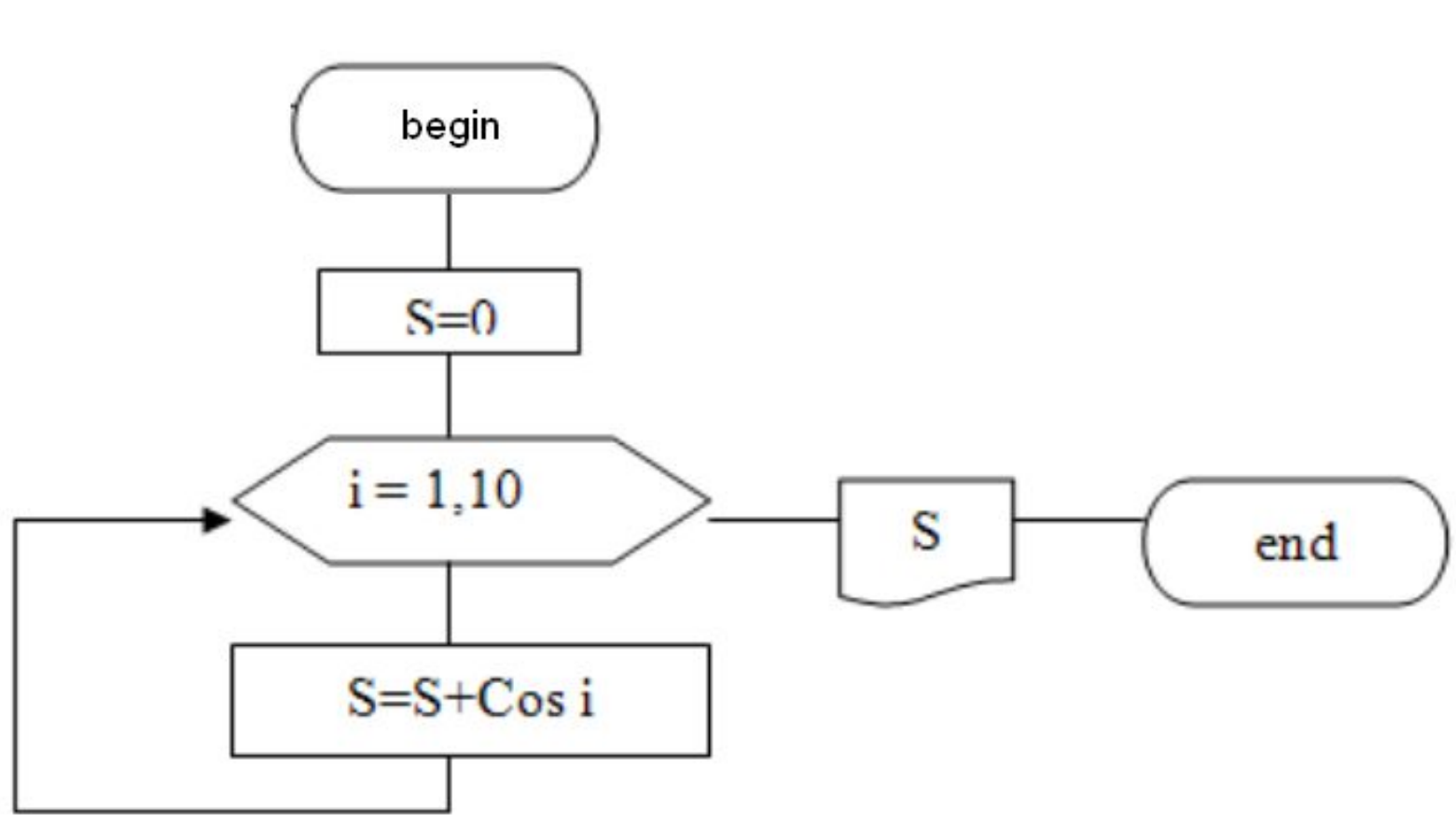
Цикл с параметром



Вычисление суммы

Найти сумму первых 10 членов последовательности

$$S = \sum_{i=1}^{10} Cosi$$



Найти произведение n чисел вида 1 · 3 · 5 · 7 · …

1. Формулы. Нужно найти произведение n первых нечетных чисел. Чему равно n-е нечетное число? В данном случае можно рассуждать так. Нечетное число меньше четного на единицу. Очевидно, что n-е четное число равно $2n$. Можно предположить, что в нашем случае n-е число будет равно $2n – 1$. Проверим нашу гипотезу. Методом математической индукции можно доказать, что n-е нечетное число равно $2n – 1$. Но в простых случаях, подобных этому, можно поступить проще. Проверим гипотезу для первых *трех* чисел. Если первые три числа ей удовлетворяют, то можно считать, что проблема решена.

При $n = 1$ получаем, что $2 \cdot 1 – 1 = 1$. При $n = 2$ имеем: $2 \cdot 2 – 1 = 3$. Наконец, когда $n = 3$, то $2 \cdot 3 – 1 = 5$. Следовательно, формула для выражения n-го нечетного числа верна. Итак, нужно найти произведение $1 \cdot 3 \cdot 5 \cdot 7 \cdot \ldots \cdot (2n – 1)$.

2. Входные данные. Входным данным является количество сомножителей n.

3. Результат. Результатом работы алгоритма является вывод вычисленного произведения.

Найти произведение n чисел вида 1 · 3 · 5 · 7 · …

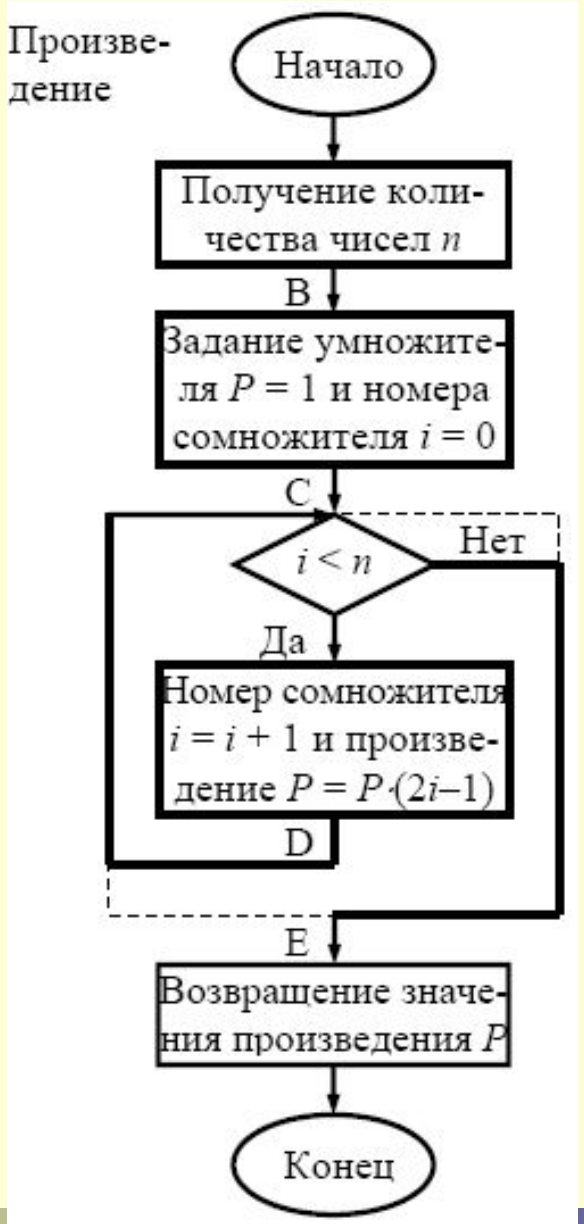
Проектирование алгоритма.

Алгоритм совпадает с предыдущим алгоритмом, исключая функцию. Сумму нужно заменить на произведение. В алгоритме вычисления функции используем цикл для накопления

произведения в переменной P. Текущий сомножитель произведения имеет вид $2i - 1$. Тогда произведение можно накопить, если:

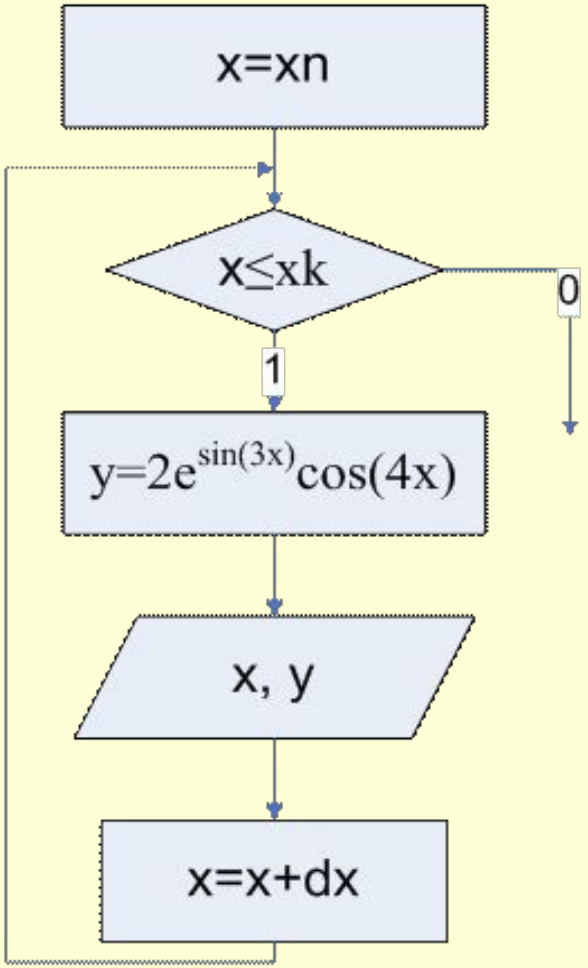
1) присвоить ему начальное значение $P = 1$;

2) умножать P в теле цикла на число $2i - 1$, изменяя i от 1 до n.



ПРИМЕР

Составить таблицу значений функции $y=2\,e^{\sin(3x)}\cos(4x)$ на отрезке [xn; xk] с шагом dx (табулирование функции). Найти сумму положительных y и произведение отрицательных y.



Платформа .NET. Обзор технологий .NET

- ПЛАТФОРМА – в контексте информационных технологий – среда, обеспечивающая выполнение программного кода. Платформа определяется характеристиками процессоров, особенностями операционных систем. Microsoft .NET – платформа.
- Framework – это инфраструктура среды выполнения программ, нечто, определяющее особенности разработки и выполнения программного кода на данной платформе. Предполагает средства организации взаимодействия с операционной системой и прикладными программами, методы доступа к базам данных, средства поддержки распределенных (сетевых) приложений, языки программирования, множества базовых классов, унифицированные интерфейсы пользователя, парадигмы программирования. .NET Framework – инфраструктура платформы Microsoft .NET.

Платформа .NET. Обзор технологий .NET

.NET Framework – инфраструктура платформы Microsoft .NET включает следующие основные компоненты: Common Language Runtime (CLR) и .NET Framework Class Library (.NET FCL).
CLS (Common Language Specification) – общая спецификация языков программирования. Это набор конструкций и ограничений, которые являются руководством для создателей библиотек и компиляторов в среде .NET Framework. Библиотеки, построенные в соответствии с CLS, могут быть использованы из любого языка программирования, поддерживающего CLS. Языки, соответствующие CLS (к их числу относятся языки Visual C#, Visual Basic, Visual C++), могут интегрироваться друг с другом. CLS – это основа межъязыкового взаимодействия в рамках платформы Microsoft .NET.

CLR (Common Language Runtime) – Среда Времени Выполнения или Виртуальная Машина. Обеспечивает выполнение сборки. Основной компонент .NET Framework. Под Виртуальной Машиной понимают абстракцию инкапсулированной (обособленной) управляемой операционной системы высокого уровня, которая обеспечивает выполнение (управляемого) программного кода.

Управляемый код – программный код, который при своем выполнении способен использовать службы, предоставляемые CLR.

Об особенностях управляемого кода можно судить по перечню задач, решение которых возлагается на CLR:

- Управление кодом (загрузка и выполнение).
- Управление памятью при размещении объектов.
- Изоляция памяти приложений.
- Проверка безопасности кода.
- Преобразование промежуточного языка в машинный код.
- Доступ к метаданным (расширенная информация о типах).
- Обработка исключений, включая межъязыковые исключения.
- Взаимодействие между управляемым и неуправляемым кодами (в том числе и COM-объектами).
- Поддержка сервисов для разработки (профилирование, отладка и т.д.).

Ядро среды выполнения реализовано в виде библиотеки mscoree.dll. При компоновке сборки в нее встраивается специальная информация, которая при запуске приложения (EXE) или при загрузке библиотеки (обращение к DLL из неуправляемого модуля – вызов функции LoadLibrary для загрузки управляемой сборки) приводит к загрузке и инициализации CLR. После загрузки CLR в адресное пространство процесса, ядро среды выполнения производит следующие действия:
находит расположение сборки; загружает сборку в память; производит анализ содержимого сборки (выявляет классы, структуры, интерфейсы); производит анализ метаданных; обеспечивает компиляцию кода на промежуточном языке (IL) в платформозависимые инструкции (ассемблерный код); выполняет проверки, связанные с обеспечением безопасности; используя основной поток приложения, передает управление преобразованному в команды процессора фрагменту кода сборки.

FCL (.NET Framework Class Library

соответствующая CLS-спецификации объектно-ориентированная библиотека классов, интерфейсов и системы типов, которые включаются в состав платформы Microsoft .NET.

Эта библиотека обеспечивает доступ к функциональным возможностям системы и предназначена служить основой при разработке .NET-приложений, компонент, элементов управления.

.NET библиотека классов является вторым компонентом CLR.

.NET FCL могут использовать все .NET-приложения, независимо от назначения архитектуры используемого при разработке языка программирования. Реализуется на языках программирования, соответствующих CLS.

MSIL (Microsoft Intermediate Language) – промежуточный язык платформы Microsoft .NET. Исходные тексты программ для .NETприложений пишутся на языках программирования, соответствующих спецификации CLS. Для таких языков может быть построен преобразователь в MSIL. Благодаря соответствию CLS, в результате трансляции программного кода, написанного на разных языках, получается совместимый IL-код.
Фактически MSIL является ассемблером виртуального процессора.
МЕТАДАННЫЕ – при преобразовании программного кода в MSIL также формируется блок МЕТАДАННЫХ, который содержит информацию о данных, используемых в программе. Фактически это наборы таблиц, которые включают в себя информацию о типах данных, определяемых в модуле (о типах данных, на которые ссылается данный модуль). Например, приложение могло включать информацию об интерфейсах, которая описывалась на Interface Definition Language (IDL). Теперь метаданные являются частью управляемого модуля.

Компиляция

В прошлом почти все компиляторы генерировали код для конкретных процессорных архитектур. При разработке платформы .NET от этой зависимости постарались избавиться. Для этого ввели двухшаговую компиляцию.

На первом этапе все .NET компиляторы генерируют промежуточный код, код на языке Intermediate Language (IL — промежуточный язык) или IL-код. Т.е. компиляция со всех языков программирования .NET, включая C#, происходит в этот промежуточный язык. IL-код не является специфическим ни для какой операционной системы и ни для какого языка программирования. Он может быть выполнен в любой среде, для которой реализована CLR-система.

На втором этапе IL-код переводится в код, специфичный для конкретной операционной системы и архитектуры процессора. Эта работа возлагается на JITкомпилятор (Just In Time compiler - компилирование точно к нужному моменту). Только после этого операционная система может выполнить приложение.

Типы приложений Visual Studio

- ***Console Application*** – позволяют выполнять ввод/вывод с использованием «консоли», то есть в окно командного процессора. Данный тип приложений существует со времен операционных систем с текстовым пользовательским интерфейсом, например MSDOS. Их применение может быть связано с отсутствием необходимости в графическом интерфейсе. Еще одним вариантом применения консольного ввода/вывода является встраивание его в программы с графическим интерфейсом. Дело в том, что современные программы содержат очень большое число команд, значительная часть которых никогда не используется обычными пользователями. В то же время, эти команды должны быть доступны в случае необходимости. Ярким примером использования данного подхода являются компьютерные игры.

Типы приложений Visual Studio

- ***Windows Forms*** – используют элементы графического оконного интерфейса, включая формы, кнопки, флажки и т.д. Приложения такого типа более удобны для пользователя, так как позволяют ему отдавать команды щелчком мыши, а не ручным вводом команд, что позволяет значительно повысить скорость работы по сравнению с консольными приложениями. Типичным примером приложения, построенного с применением графического интерфейса, является MS Word.
- ***Библиотека классов (Class Library)*** – представляет собой библиотеки, содержащие классы и методы. Библиотеки не являются полноценным самостоятельными приложениями, но могут использоваться в других программах. Как правило, в библиотеки помещают алгоритмы и структуры данных, которые могут быть полезны более чем одному приложению. Специфика платформы .NET такова, что она «подходит» для разработки «офисных» приложений, Web-приложений, сетевых приложений и приложений для мобильных устройств.

Объектно-ориентированный подход

Программные системы предназначены для моделирования реальных систем, поэтому очень важно, в каких терминах описываются эти реальные системы. Описание в виде последовательности действий (процедурный подход к программированию) оказалось слишком сложным. Объектно-ориентированный подход (ООП) предлагает описывать системы с позиций учета взаимодействия объектов, входящих в их состав.

Операции, которые существенны при построении всей системы, могут быть скрыты от конкретных объектов.

Объект - это что-то, с чем можно оперировать.

У объекта есть *состояние, поведение и возможность отличить его от других объектов.*

Объектно-ориентированный подход

Инкапсуляция

Инкапсуляция – это сокрытие информации о внутреннем устройстве объекта. В основе построения объектно-ориентированных систем положен принцип:

объекты должны знать об устройстве друг друга только то, что необходимо для их взаимодействия и не более того.

Объектно-ориентированный подход
Состояние объекта

Каждый объект характеризуется своим состоянием.

Состояние объекта характеризуется текущим значением его *атрибутов*.

Атрибутами могут быть не только простейшие значения - числа, строки знаков, логические значения и т.п., но и сложные, например – объекты, при этом не все атрибуты объекта могут изменяться.

Объектно-ориентированный подход
Идентификация объекта

Метод идентификации объекта должен отвечать на вопрос, как отличить один объект от другого. Иными словами, если имеются два объекта, как можно определить, что эти объекты разные.
Если программа обращается к двум объектам, она может сравнить все известные значения атрибутов объектов и определить, равны ли они. Для того чтобы сравнивать объекты на тождество, необходимо, чтобы у любого объекта существовала некоторая уникальная характеристика.

Объектно-ориентированный подход
Идентификация объекта

Во многих случаях один из атрибутов объекта по определению является уникальным. При этом ответственность за правильность идентификации объекта лежит на программисте, разрабатывающем систему. Чтобы избежать появления возможных ошибок при формировании уникального значения атрибута, идентифицирующего объект, целесообразно автоматизировать этот процесс за счет использования специальных типов данных (например, тип данных «Счетчик» в MS Access, Autoincrement поля).

Объектно-ориентированный подход
Идентификация объекта

При создании любого нового объекта ему присваивается уникальный идентификатор, отличающийся от идентификаторов всех остальных имеющихся в системе объектов. Форма этого идентификатора может быть различной. Это может быть число с достаточно большим диапазоном значений, чтобы хватило на все объекты, или адрес в универсальной системе адресации, например URL для идентификации страниц в Интернете, или специальным образом генерируемое имя.

Зная идентификаторы двух объектов, всегда можно сделать вывод об их тождественности или не тождественности.

Особое значение правильность идентификации объектов приобретает в распределенных системах, в которых программы, работающие на разных компьютерах, независимо друг от друга в произвольное время создают и уничтожают объекты.

Объектно-ориентированный подход
Интерфейс объекта

Важнейшей характеристикой объекта является описание того, как он может взаимодействовать с окружающим миром. Это описание называется *интерфейсом* объекта.

Объекты взаимодействуют между собой с помощью сообщений. Принимая сообщение, объект выполняет соответствующее действие. Эти действия обычно называют *методами* (другое название - *операции*).

Интерфейс - это внешнее описание объекта.

Объектно-ориентированный подход
Интерфейс объекта

Наряду с методами и атрибутами, входящими в интерфейс и доступными другим объектам, у объекта могут быть методы или атрибуты, предназначенные для «внутреннего употребления», к которым может обращаться только сам объект.

Объект известен другим объектам только по своему интерфейсу. Внутренняя структура его скрыта. Важным следствием является возможность изменения внутренней структуры объекта независимо от других взаимодействующих с ним объектов. В подобном разделении внутренней структуры и интерфейса объекта и заключается суть принципа инкапсуляции.

Объектно-ориентированный подход
Интерфейс объекта

Применительно к программированию это означает, что, используя принцип инкапсуляции, намного легче и безопаснее производить модификацию системы. Для объекта, описывающего трехмерное изображение на экране монитора, существует метод «вращать». Если мы придумали более эффективную реализацию этого метода и ее реализовали, то тестирование всей системы фактически сводится к тестированию только этого нового метода. Поскольку интерфейс объекта не изменился, работоспособность всей графической системы, частью которой является этот объект, не нарушилась.

Интерфейс объекта

Наряду с методами и атрибутами, входящими в интерфейс и доступными другим объектам, у объекта могут быть методы или атрибуты, к которым может обращаться только сам объект.

Объект известен другим объектам только по своему интерфейсу. Внутренняя структура его скрыта. Важным следствием является возможность изменения внутренней структуры объекта независимо от других взаимодействующих с ним объектов. В подобном разделении внутренней структуры и интерфейса объекта и заключается суть принципа инкапсуляции.

Объектно-ориентированный подход
Создание и уничтожение объектов

В любой системе объекты создаются, функционируют и, в конце концов, уничтожаются.

Создание объектов всегда выполняется явно. При этом задавать, когда и какие объекты создаются, можно либо на стадии разработки, либо на стадии выполнения.

При написании программы в ней объявляются переменные, которые обозначают объекты. Компилятор обеспечивает создание этих объектов при запуске программы.

Динамическое создание объектов на стадии выполнения означает, что программа в ходе своей работы может обращаться к неким особым *фабрикам объектов* для создания новых объектов. Фабрика объектов может быть реализована в разных средах по-разному. Это может быть особый объект, отдельная функция или отдельная подсистема языка.

Объектно-ориентированный подход
Создание и уничтожение объектов

К удалению объектов существует два подхода:
объекты должны уничтожаться явно, с помощью специальных вызовов,
объекты уничтожаются тогда, когда они больше не нужны, например при завершении программы.
Возможно существование объектов и после завершения программы, если они создаются не в оперативной памяти, а на постоянном носителе информации (на диске). Средой жизни объектов в таком случае может являться, например, объектно - ориентированная база данных. Подобные объекты называются *постоянными*.

Классы

В системе обычно функционирует множество объектов. Некоторые из них «похожи» или од в кл но уп се об от оди и реализ тот интерфейс одним и тем же способом. Два объекта одного класса могут отличаться только текущим состоянием. Индивидуальные объекты называются экземплярами класса, а класс - это шаблон, по которому строятся объекты.

Считаем термины «объект» и «экземпляр» эквивалентными.

Классы.
Различие понятий «интерфейс» и «тип»

Интерфейс - это внешняя часть класса. Интерфейс определяет, как объекты данного класса могут взаимодействовать с другими объектами этого или других классов. Если у двух объектов совпадают интерфейсы, это еще не означает, что они принадлежат к одному и тому же классу. Кроме совпадения интерфейсов необходимо совпадение реализации этих интерфейсов, совпадение поведения объектов.

Классы.
Различие понятий «тип» и «класс»

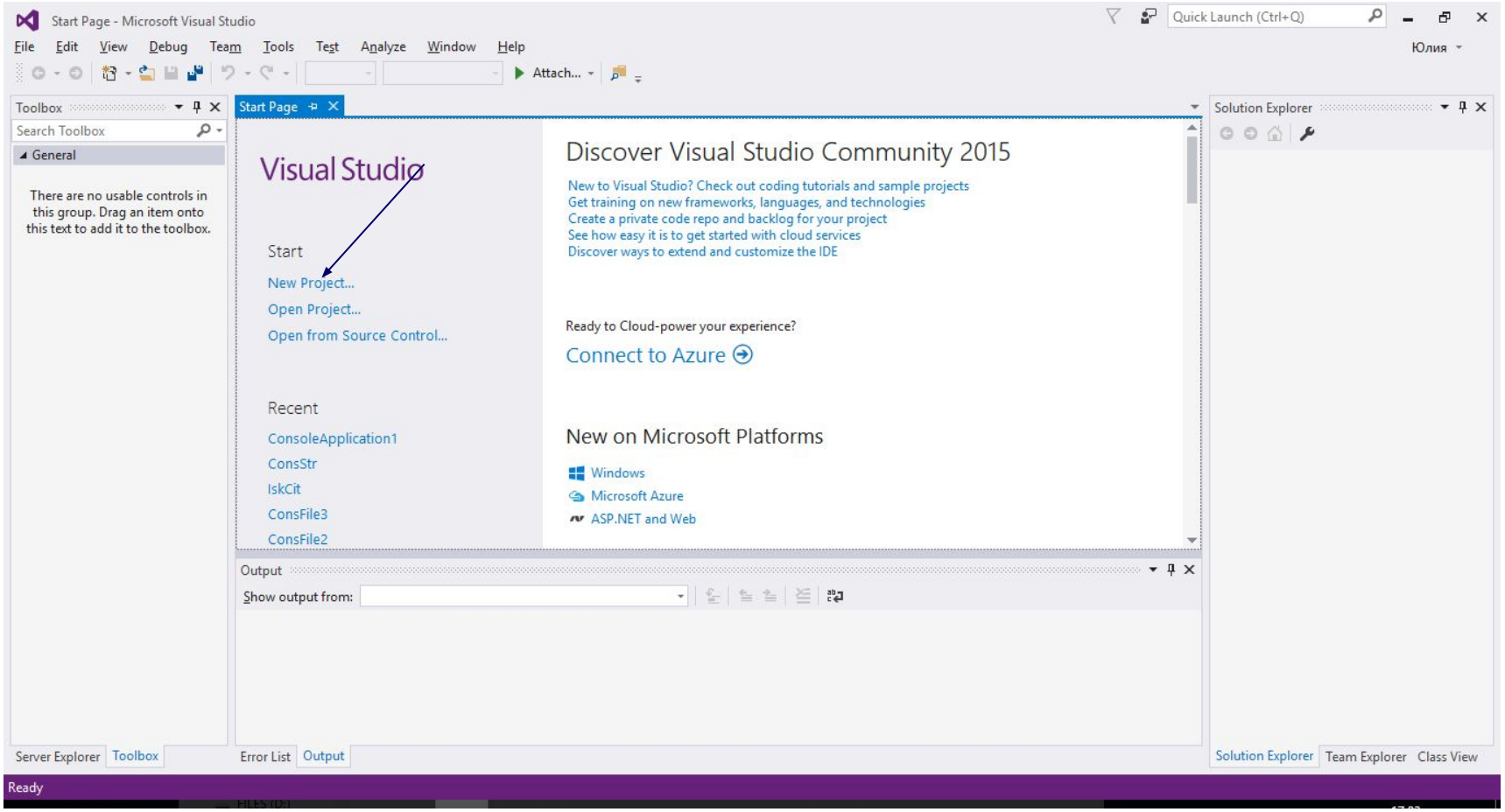
Понятия типа и класса часто употребляются в одном и том же смысле. Термин «тип» прежде всего используется в контексте языков программирования. Тип - это область определения некой величины, то есть множество ее возможных значений и набор применимых к ней операций. Тип может задаваться классом, но определяется не только им. Например, во многих объектно-ориентированных языках программирования существуют простейшие типы данных, не являющиеся классами: целые числа, символы и т.п.

Пространство имен

Пространство имен – это способ организации системы типов в единую группу. В рамках .NET существует единая (общеязыковая) библиотека базовых классов. Концепция пространства имен обеспечивает эффективную организацию и навигацию по этой библиотеке. Вне зависимости от языка программирования, доступ к определенным классам обеспечивается за счет их группировки в рамках общих пространств имен.

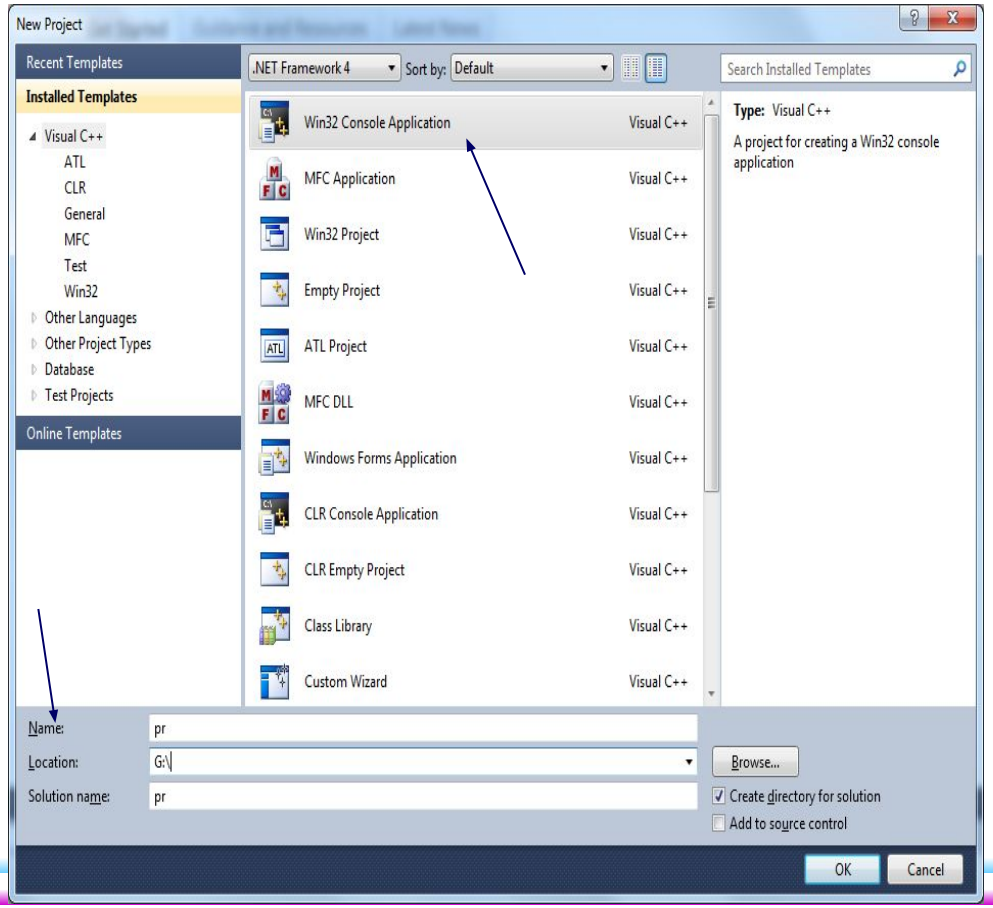
Создание консольного приложения

Запускаем Visual Studio. Выбираем *Новый проект (New project)*



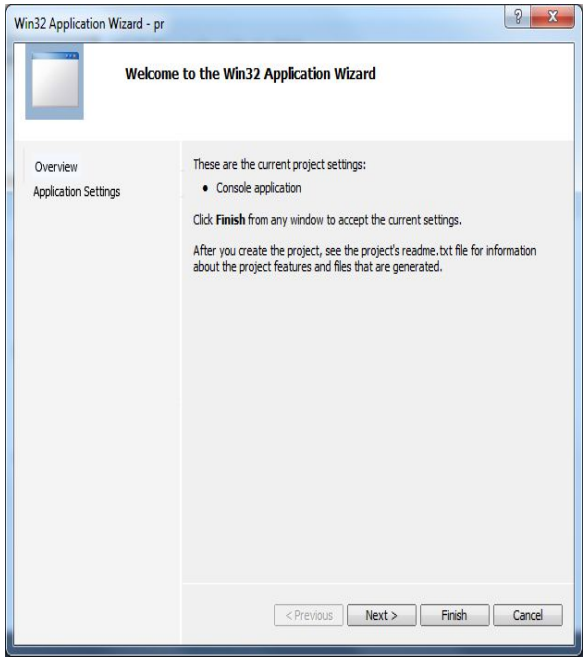
Создание консольного приложения

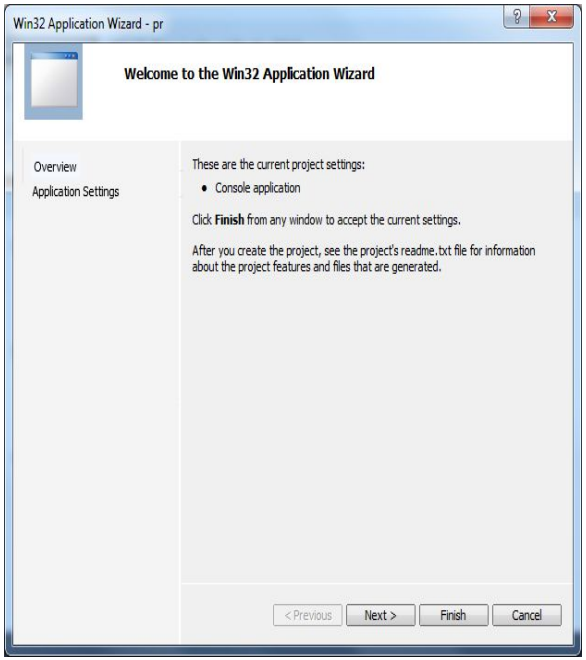
Выбираем Win 32 Console Application. Задаем имя проекта и выбираем папку, в которой будет располагаться проект.



Создание консольного приложения

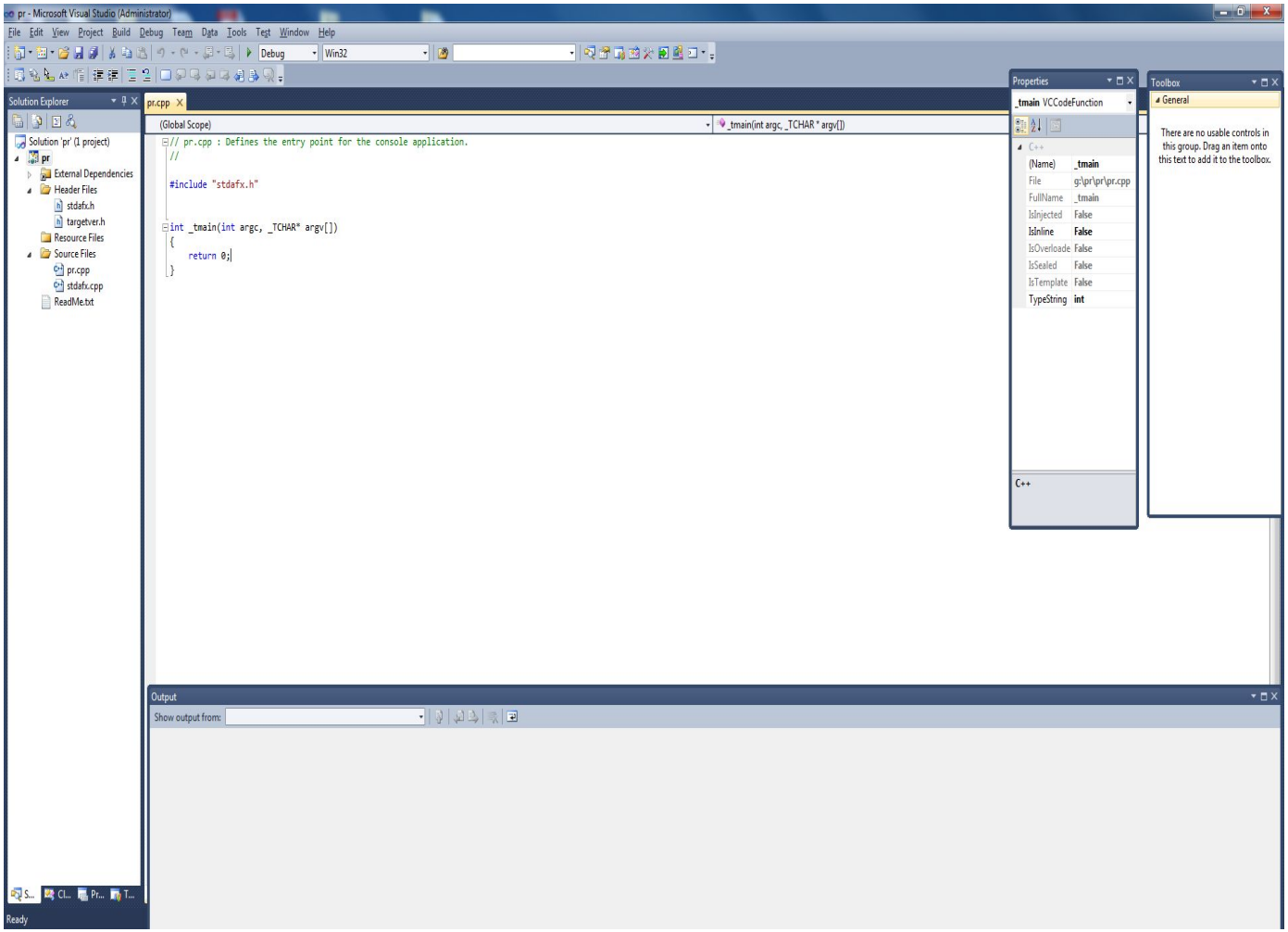
Щелкаем по кнопке: *Далее (Next)*, в новом окне – по *Выход (Finish)*





Создание консольного приложения

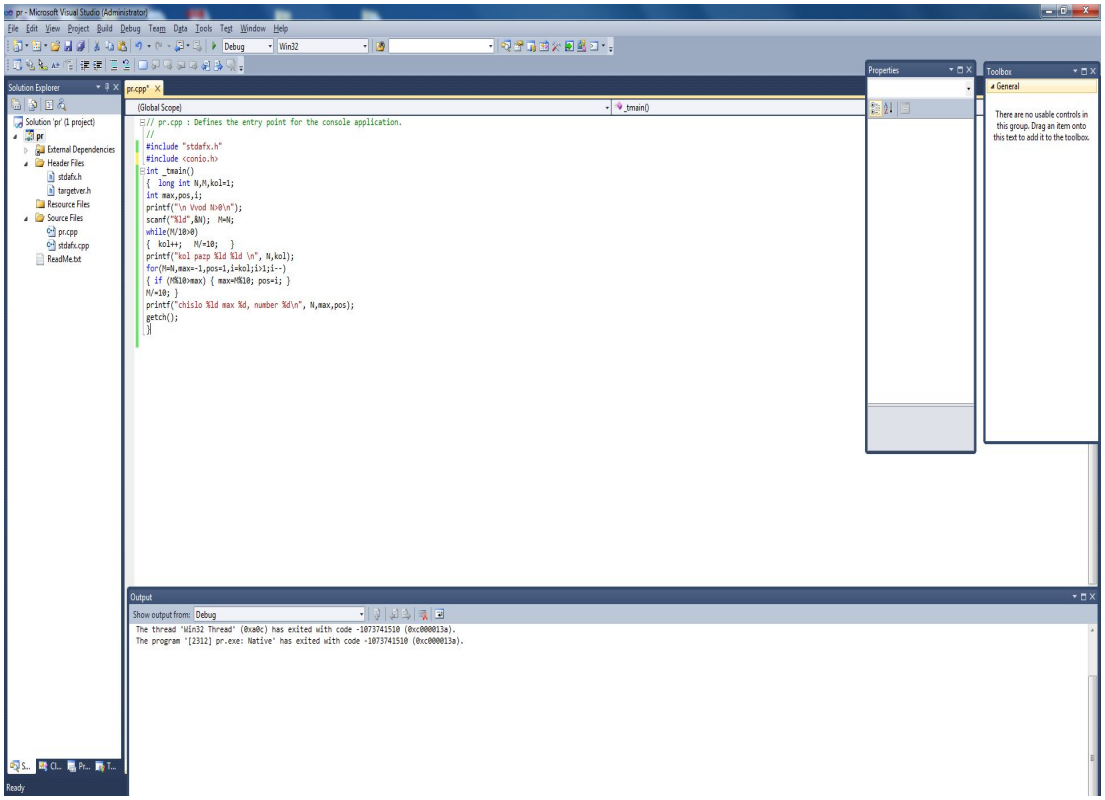
Открывается окно Редактора кода.



Создание консольного приложения

Пример. Вводится целое число, программа подсчитывает количество разрядов, определяет максимальную цифру и ее положение в числе.

Вводим текст программы:



Создание консольного приложения

Задача. Составить схему алгоритма и программу для вычисления значений функций Y и F для заданных значений переменной x и постоянной a= 1. Вывести на экран значения F, Y для соответствующих значений x=4 и x=1,7.

$$Y = ax^5\left(arctg(a+x) - \sqrt{|(x-a)|} + \ln(x+1)^2\right);$$

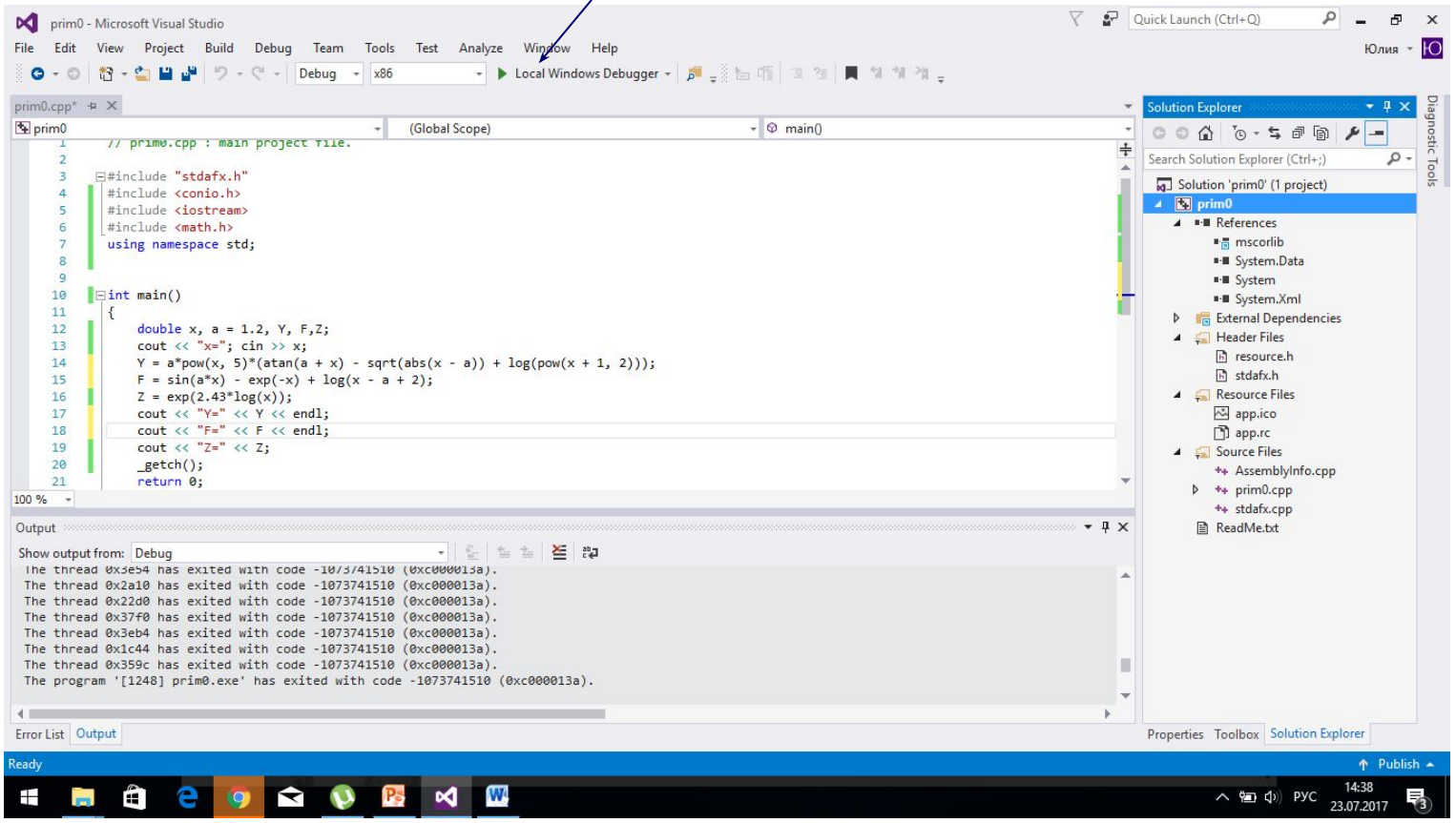
$$F = Sin(ax) - e^{-x} + \ln(x-a+2);$$

Создание консольного приложения

```
#include "stdafx.h"
#include <conio.h>
#include <iostream>
#include <math.h>
 using namespace std;
int main()
{
double x, a = 1.2, Y, F;
cout << "x="; cin >> x;
Y = a*pow(x, 5)*(atan(a + x) - sqrt(abs(x - a)) + log(pow(x + 1,
2)));
F = sin(a*x) - exp(-x) + log(x - a + 2);
cout << "Y=" << Y << endl;
cout << "F=" << F;
//_getch();
    return 0;
}
```

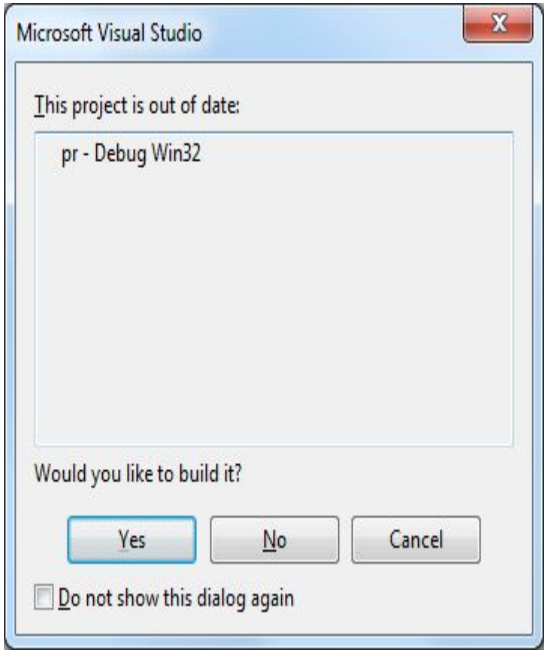
Создание консольного приложения

Запускаем приложение



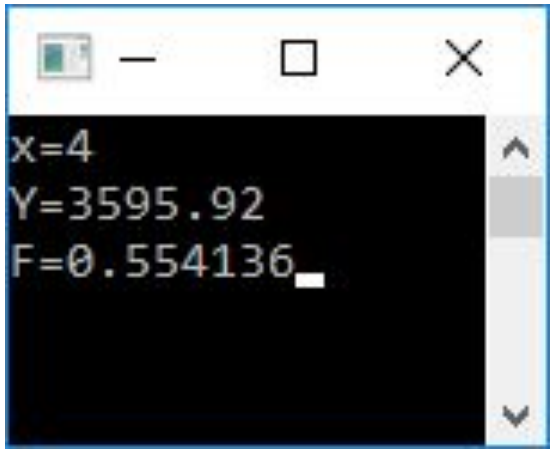
Создание консольного приложения

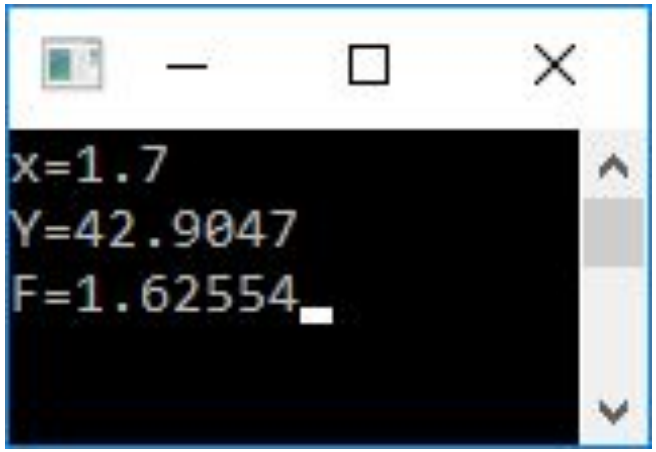
В диалоговом окне щелкаем по кнопке <Yes>



Создание консольного приложения

Если нет ошибок, то программа начинает работать.
Вводим исходные данные и получаем результат.





Язык C++. **Алфавит языка**

<алфавит> **::** = <буквы> | **<**цифры**>** | <ограничители>
<буквы> **::** = **A | B | …| Z | a | b | …| z | _**
<цифры> **::** = **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**
<ограничители>**::=**<знаки операций>|<скобки> |
<зарезервированные слова>| <разделители>

Пробел является разделителем.

Комментарий - это текст, начинающийся знаками *//* и до конца строки или текст, заключенный между знаками /* */. Может содержать буквы (в том числе и русские), цифры, специальные знаки. Используется для обозначения участков программ или пояснения команд. Может располагаться в любом месте программы.

; - завершает оператор.

Структура программы

Любая программа на языке C++ представляет собой одну или несколько функций. В любой программе обязательно должна быть одна функций main(). С этой функции начинается выполнение программы. Правилом хорошего тона в программировании является разбиение задачи на подзадачи, и в главной функции чаще всего должны быть операторы вызова других функций.

```
Объявление глобальных переменных
Тип_ результ main(Список_переменных)
{
 Операторы
}
Тип_результата f1(Список_переменных)
{
 Операторы
 }
```

Здесь Тип_результата
- тип возвращаемого функцией значения.

Основные этапы обработки программы на языке С++

- Сначала программа обрабатывается препроцессором, который выполняет директивы препроцессора, в нашем случае это директивы включения заголовочных файлов (файлов с расширением **h**) - текстовых файлов, в которых содержится описание используемых библиотек. В результате формируется полный текст программы, который поступает на вход компилятора.
- Компилятор разбирает текст программ на составляющие элементы, проверяет синтаксические ошибки и в случае их отсутствия формирует объектный код (файл с расширением **obj**).
- Компоновщик подключает к объектному коду программы объектные модули библиотек и других файлов (если программа состоит из нескольких файлов) и генерирует исполняемый код программы (файл с расширением **exe**), который уже можно запускать на выполнение.

Элементарные конструкции

Различные объекты программы должны иметь имена. В качестве имен используются идентификаторы.

Идентификатор (имя объекта программы) - последовательность букв и цифр, начинающаяся с буквы. Желательно выбирать значащие идентификаторы, например, MAX, MIN, SUMMA и т.д.

Зарезервированные слова нельзя использовать в качестве идентификаторов.

Константы – это данные, значения которых не изменяются в процессе работы программы. Константы могут быть поименованными (с типом и без типа) и явно заданными.

Арифметические константы предназначены для представления целых и вещественных числовых данных.

Для *вещественных констант* используется форма записи с **фиксированной** точкой и **плавающей** точкой (экспоненциальная форма). В форме с фиксированной точкой целая и дробная части разделяются **точкой**, при экспоненциальной форме число представляется в виде мантиссы и порядка, между которыми стоит буква E.

<мантисса>E{ ±}<порядок>

Данные в языке С++

Для решения задачи в любой программе выполняется обработка каких-либо данных. Данные могут быть самых различных типов: целые и вещественные числа, символы, строки, массивы. Данные в языке С++ описываются в начале функции.

Типы данных в языке С++

В С++ определены пять основных типов данных:
char – символьные, **int** – целые, **float** – с плавающей точкой, **double** – двойной точности, **void** – без значения (бестиповый).
На базе этих типов формируются другие типы.
Данные типа **char** всегда занимают один байт.

Данные в языке C++

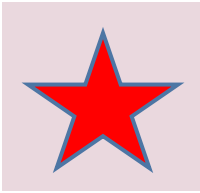
Типы данных	Размер в байтах	Минимально допустимый диапазон значений
Целые значения		
char	1	-128÷127
unsigned char	1	0 ÷ 255
signed char	1	-128 ÷ 127
int	2 или 4	-32768 ÷ 32767
unsigned int	2 или 4	0 ÷ 65535
signed int	2 или 4	-32768 ÷ 32767
short int	2	-32768 ÷ 32767
unsigned short int	2	0 ÷ 65535
signed short int	2	-32768 ÷ 32767
long int	4	-2147483648 ÷ 2147483647
long long int	8	$-(2^{63}-1) \div (2^{63}-1)$
signed long int	4	-2147483648 ÷ 2147483647
unsigned long int	4	0 ÷ 4294967295
unsigned long long int	8	$0 \div 2^{64}-1$

Данные в языке С++

Типы данных	Размер в байтах	Минимально допустимый диапазон значений
float	4	3,4E-38 ÷3,4E+38
double	8	1E-37 ÷ 1E+37
long double	10	1E-37 ÷ 1E+37
bool	1	true, false

Данные в языке С++

Как видно из таблицы, базовые типы могут быть расширены с помощью спецификаторов (модификаторов) **signed, unsigned, long, short.** Следует учитывать, что вещественные числа хранятся в экспоненциальной форме **mE±p**, где **m** – мантисса (целое или дробное число с десятичной точкой), **p** – порядок (целое число). Для того, чтобы перевести число в экспоненциальной форме к обычному представлению с фиксированной точкой, необходимо **мантиссу** умножить на **десять** в степени **порядок**.



Примеры

$-6.42E+2 = -6.42.10^{2} = -642$

$-3.2E-6 = -3.2.10^{-6} = -0.0000032$

Переменные языка С++

Переменная – поименованный участок памяти, в котором хранится значение.

Имя (**идентификатор**) в языке С++ – совокупность букв, цифр и символа подчеркивания, начинающаяся с буквы или символа подчеркивания. В С++ **строчные и прописные буквы считаются разными** (т.е. **abc** и **Abc** – разные переменные). Имена в С++ бывают внешние и внутренние. **Внешние имена** обрабатываются во время внешнего процесса компоновки, это имена функций и глобальных переменных, которые совместно используются в различных исходных файлах. Все остальные имена – **внутренние**. Длина имени не ограничена, но ограничено количество значащих символов.

Переменные языка C++

Все переменные в языке C++ должны быть описаны.

Оператор описания переменных имеет вид:

тип список_переменных;

тип – один из типов,

список_переменных – один или несколько идентификаторов, разделенных запятыми.

Например,

int a,bc,f;

float g,u,h12;

В C++ могут обрабатываться структурированные типы данных: массивы, строки, записи, файлы, множества.

Массивы

Массив – совокупность данных одного и того же типа. Число элементов массива фиксируется при описании типа. Для доступа к элементу необходимо указать имя массива и его номер в квадратных скобках. Описание одномерного массива имеет вид:

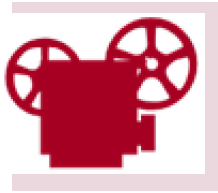
тип имя_переменной [n];

где **n** – количество элементов в массиве; элементы в массиве нумеруются с нуля, таким образом, элементы в массиве нумеруются от **0** до **n-1**.

Пример:

double a[25];

Описан массив a из 25 вещественных чисел (типа double), элементы нумеруются от 0 до 24 (a[0]…a[24]).



Многомерные массивы

В C++ определены и многомерные массивы. Двумерный массив (матрицу) можно объявить так:

тип имя_переменной [n][m];
где **n** – количество строк в матрице(строки нумеруются от **0** до **n-1), m** – количество столбцов (столбцы нумеруются от **0** до **m-1**).

Пример:
int h[10][15];
Описана матрица h, состоящая из 10 строк и 15 столбцов (строки нумеруются от 0 до 9, столбцы от 0 до 14).

Для обращения к элементу матрицы необходимо указать ее имя, и в квадратных скобках номер строки, а затем в квадратных скобках – номер столбца.
Например, **h[2][4]** – элемент матрицы h, находящийся в третьей строке и пятом столбце.
В C++ можно описать многомерные массивы, которые можно объявить с помощью оператора следующей структуры:

тип имя_переменной [n1][n2]…[nk];

Строки

Строка – последовательность символов. Если в выражении встречается одиночный символ, он должен быть заключен в **одинарные кавычки**. При использовании в выражениях строка заключается в **двойные кавычки**. Признаком конца строки является нулевой символ ‘\0’. Строки можно описать, как массив символов (массив элементов типа char). Объявляя такой массив, следует предусмотреть место для хранения признака конца строки (‘\0’).

Например, описание строки из 25 символов должно выглядеть так:

char s[26];

Здесь элемент с номером 26 предназначен для хранения символа конца строки.

Примеры символьных констант: ‘F’, ‘*’, ‘Щ’.

Примеры строк знаков: “Здесь был Петя”, “\t f=\xF5\n”

По месту объявления переменные в языке Си можно разделить на три класса:

1. Локальные – переменные, которые объявляются внутри функции и доступны только в ней.

```
int main()
{
float s; s=4.5;
 }
int f1()
{
 int s;
 s=6;
}
 int f2()
{
long int s;
s=25;
}
```

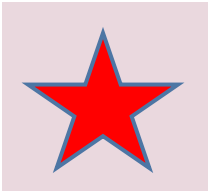
2. Глобальные – переменные, которые описаны до всех функций, они доступны из любой функции.

```
#include <stdio.h>
float s;
int main()
{ s=4.5; }
int f1()
{ s=6; }
int f2()
{ s=25; }
```

Определена глобальная переменная s (типа float), в функции main ей присваивается значение 4.5, в функции f1 – присваивается значение 6, а в функции f2 – присваивается значение 25.

3. Формальные параметры функций описываются в списке параметров функции.

Константы в языке С++



Константы не изменяют своего значения в процессе выполнения программы.
Если при объявлении переменной используется модификатор **const**, то она не может изменять свое значение.
const double pi=3.141592653589793;
В константе явно или не явно присутствует тип. По умолчанию константа будет принадлежать к типу наименьшего возможного размера. Однако, используя суффикс (символ после значения константы) можно явно указать тип. Если после вещественного числа в ***экспоненциальной форме*** присутствует символ **F**, то константа принадлежит к типу **float**, а если символ **L** – то к типу **long double**. Для целых чисел суффикс **U** обозначает **unsigned**, **L – long**.

Целые числа можно записывать в восьмеричной или шестнадцатеричной системе. Шестнадцатеричное число начинается с **0x**(например, 0x80 – $(80)_{16}$), восьмеричное с **0** (например, 025 – $(25)_8$).
Примеры: 0xA, 0X00FF, 01, 07155/
Константа может быть определена до главной функции main. В этом случае можно использовать директиву **#define**. Например, для определения константы p можно перед функцией main вставить строку
#define PI= 3.141592653589793
Если в тексте программы будет встречаться имя PI, оно автоматически будет заменяться значением 3.141592653589793.

Операции в языке С++

Операция присваивания

В операторе присваивания слева всегда стоит имя переменной, а справа – значение, например:

a=b;

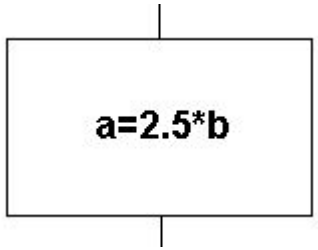
где **a** – имя переменной или элемента массива, **b** – выражение, переменная, константа или функция.

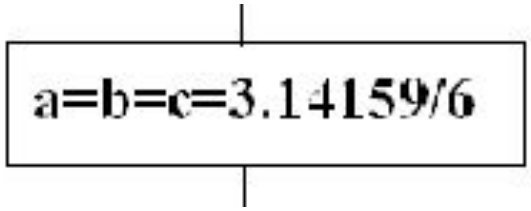
В результате выполнения оператора **a=b** переменной **a** присваивается значение **b**. Если в операции присваивания встречаются переменные разных типов, происходит преобразование типов. В операции присваивания значение в правой части преобразуется к типу переменной левой части.

Множественное присваивание

Множественное присваивание – присваивание нескольким переменным одного и того же значения.

a=b=c=3.14159/6;

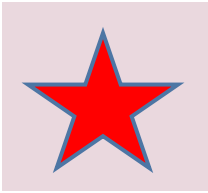




Арифметические операции

Операция	Действие	Тип операнда	Тип результата
+	Сложение	Целый, вещественный	Целый, вещественный
-	Вычитание, унарный минус	Целый, вещественный	Целый, вещественный
*	Умножение	Целый, вещественный	Целый, вещественный
/	Деление	Вещественный	Вещественный

Арифметические операции



Операция	Действие	Тип операнда	Тип результата
/	Целочисленное деление	Целый	Целый
%	Остаток от деления	Целый	Целый
- -	Декремент, уменьшение на 1	Целый	Целый
++	Инкремент, увеличение на 1	Целый	Целый

Арифметические операции

Операция	Действие	Тип операнда	Тип результата
&	"и"	Целый	Целый
|	"или"	Целый	Целый
^	Исключающее "или"	Целый	Целый
~	Логическое отрицание	Целый	Целый
<<	Сдвиг влево	Целый	Целый
>>	Сдвиг вправо	Целый	Целый

Операции увеличения (инкремента) и уменьшения (декремента)

Оператор p=p+1;
можно записать в префиксной форме ++p;
так и в постфиксной p++;
Эти формы отличаются при использовании их в выражении.

Если знак декремента (инкремента) предшествует операнду, то сначала выполняется увеличение (уменьшение) значения операнда, а затем операнд участвует в выражении.

Пример: x=12; y=++x;
В результате в y будет храниться число 13. Если знак декремента (инкремента) следует после операнда, то сначала операнд участвует в выражении, а затем выполняется увеличение (уменьшение) значения операнда.

Пример: x=12; y=x++;
В результате в y будет храниться число 12.

Составное присваивание

К операторам составного присваивания относятся +=, -=, *=, /=.

Оператор x+=p; предназначен для увеличения x на величину p.

Оператор x-=p; предназначен для уменьшения x на величину p.

Оператор x*=p; предназначен для умножения x на p.

Оператор x/=p; предназначен для деления x на p.

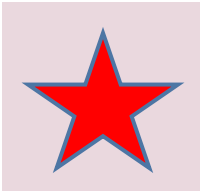
Операции целочисленной арифметики

К операциям целочисленной арифметики относятся:
целочисленное деление /
остаток от деления %.
При целочисленном делении операция / возвращает целую часть частного (дробная часть отбрасывается), а операция % – остаток от деления. Ниже приведены примеры этих операций

11 % 4 = 3
11 / 4 = 2
7 % 3 = 1
7 / 3 = 2

Операции битовой арифметики

Во всех операциях битовой арифметики действия происходят над двоичным представлением целых чисел. К операциям битовой арифметики относятся следующие операции С++.



Арифметическое И (&). Оба операнда переводятся в двоичную систему, затем над ними происходит логическое поразрядное умножение операндов по следующим правилам.

1 & 1 = 1 1 & 0 = 0 0 & 1 =0 0 & 0 = 0

```
int main ()
{ int A, B;
  A=13;
  B=23;
  printf("\n%d\n", A & B) }
```

Этот участок программы работает следующим образом. Число A=13 и B=23 переводятся в двоичное представление 0000000000001101 и 0000000000010111. Затем над ними поразрядно выполняется логическое умножение.

```
  0000000000001101
& 0000000000010111
------------------
  0000000000000101
```

Результат переводится в десятичную систему счисления, в нашем случае будет число 5. Таким образом, 13 & 23 = 5.

Арифметическое ИЛИ (|). Здесь также оба операнда переводятся в двоичную систему, после чего над ними происходит логическое поразрядное сложение операндов по следующим правилам.

1 | 1 = 1 1 | 0 = 1 0 | 1 =1 0 | 0 = 0

Например:

```
int main ()
{ int A, B;
   A=13;
   B=23;
   printf("\n%d\n", A | B) }
```

Над двоичным представлением значений A и B выполняется логическое сложение.

```
|   0000000000001101
    0000000000010111
    0000000000011111
```

После перевода результата в десятичную систему имеем 13 | 23 =31.

Арифметическое исключающее ИЛИ (^). Здесь также оба операнда переводятся в двоичную систему, после чего над ними происходит логическая поразрядная операция ^ по следующим правилам.

1 ^ 1 = 0 1 ^ 0 = 1 0 ^ 1 =1 0 ^ 0 = 0

Арифметическое отрицание (~). Эта операция выполняется над одним операндом. Применение операции not вызывает побитную инверсию двоичного представления числа. Например, рассмотрим операцию not 13.

	0000000000001101
~a	1111111111110010

После перевода результата в десятичную систему получаем ~13=-14.

Сдвиг влево (M << L). Двоичное представление числа M сдвигается влево на L позиций.

Пример **17 << 3**. Представляем число 17 в двоичной системе 10001, сдвигаем число на 3 позиции влево 10001000, в десятичной системе это число 136. 17 << 3 =136. Заметим, что сдвиг на один разряд влево соответствует умножению на 2, на два разряда – умножению на 4, на три – умножению на 8. Таким образом, операция M << L эквивалентна $M \cdot 2^L$.

Сдвиг вправо (M >> L). В этом случае двоичное представление числа M сдвигается вправо на L позиций, что эквивалентно целочисленному делению числа M на 2^L. Например, 25 >> 1=12, 25 >> 3= 3.

Логические операции и операции отношения

Логические операции выполняются над логическими значениями ИСТИНА (true) и ЛОЖЬ (false). В языке C++ ложью является 0, а истина – любое значение, отличное от нуля. В C++ появился тип bool. Результатами операций отношения (<, <=, >, >=, ==, ~=) или логической операции является ИСТИНА (true, 1) или ЛОЖЬ (false, 0). В языке определены следующие логические операции ИЛИ (||), И (&&), НЕТ (!)

Обозначение	Действие
==	равно
>	больше
>=	больше или равно
<	меньше
<=	меньше или равно
!=	неравно

Логические операции и операции отношения

A	B	!A	A&&B	A||B
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

Операция ? *Тернарное условие*

Для организации разветвлений в простейшем случае можно использовать оператор ? следующей структуры:

Условие? Выражение1: Выражение 2;

Операция работает так.

Если **Условие** истинно (не равно 0), то результатом будет **Выражение1**, в противном случае **Выражение2**.

Например, оператор

y=x<0 ? –x: x;

записывает в переменную y модуль числа x.

Операция явного приведения типа

Для приведения выражения к другому типу данных в C++ существует операция явного приведения типа:
(тип) выражение
Здесь **тип** – любой поддерживаемый в C++ тип данных.
Например,

```
x=5;
y=x/2;
z=(float) x/2;
```

В результате этого участка программы переменная y принимает значение 2 (результат целочисленного деления), а переменная z – значение 2.5

Стандартные математические функции в языке C++

Обозначение	Действие
abs(x)	Модуль целого числа
abs(x)	Модуль вещественного числа
sin(x)	Функция синус
cos(x)	Функция косинус
tan(x)	Функция тангенс

Стандартные математические функции в языке C++

Обозначение	Действие
atan(x)	Арктангенс $\left[-\frac{\pi}{2};\frac{\pi}{2}\right]$
atan2(y,x)	Арктангенс $\frac{y}{x}$ в диапазоне $[-\pi,\pi]$
sinh(x)	Гиперболический синус x
cosh(x)	Гиперболический косинус x
tanh(x)	Гиперболический тангенс x

Стандартные математические функции в языке C++

Обозначение	Действие
exp(x)	e^x
log(x)	Функция натурального логарифма ln(x), x>0
log10(x)	Функция десятичного логарифма log10(x), x>0
pow(x,y)	x^y. Ошибка области определения, если x=0, y≤0 или x<0 и y – не целое
sqrt(x)	$\sqrt{x}$, x≥0

Возведение в степень C++

Определенную проблему представляет возведение *x* в степень y. Есть функция ***pow(x,y)***, которая позволяет возвести *x* в степень *y*.

Можно воспользоваться формулой, которая для *x>0* программируется с помощью стандартных функций:

exp(y*log(x))

Если основание степени отрицательное число, то можно воспользоваться формулами:

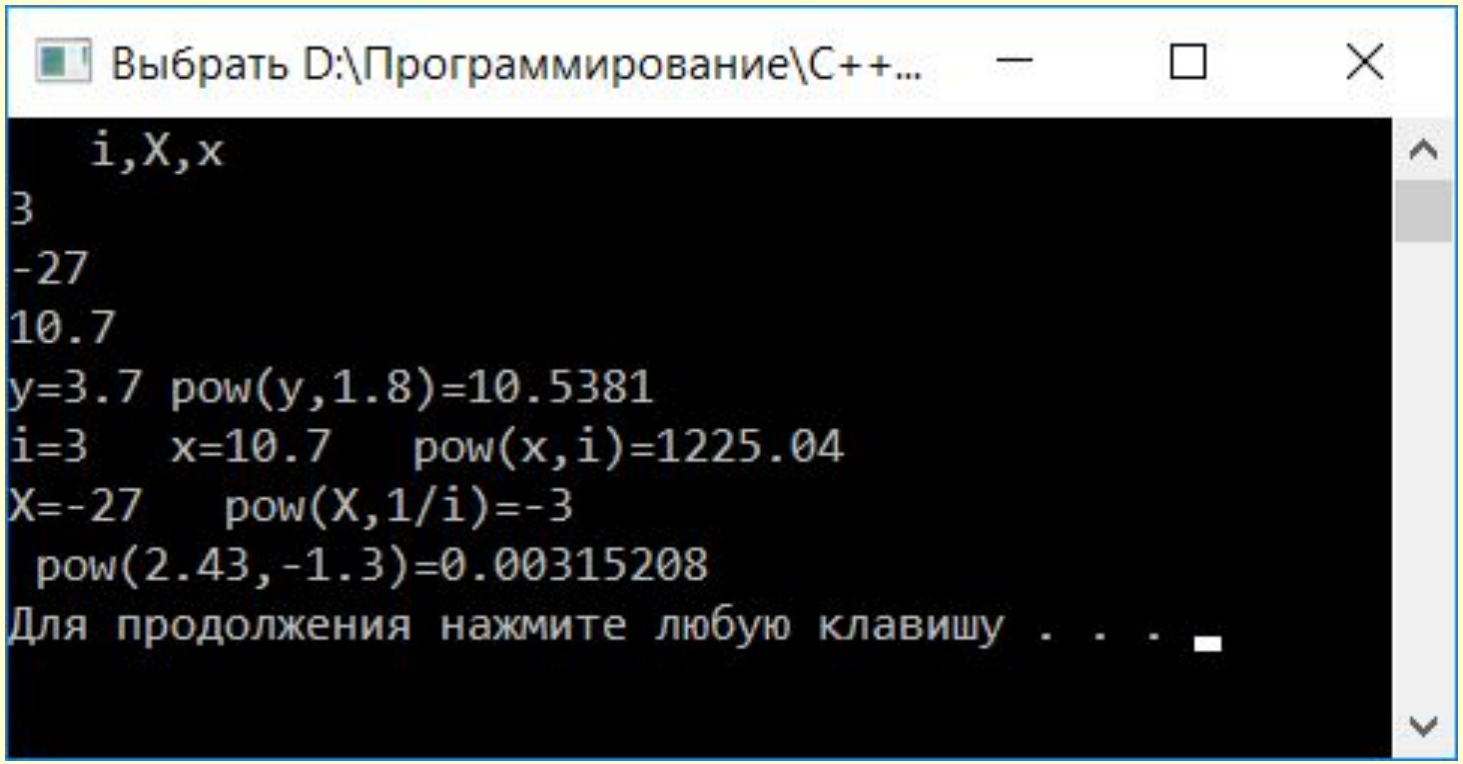
-exp(y*log(abs(x)))

-pow(abs(x),y)

Написать программу, которая демонстрирует различные варианты операции возведения в степень.

```
#include <iostream>
using namespace std;
 int main()
{ int i = 0; int X=0; double x=0; double y=3.7;
cout<< " i,X,x  "<<endl;
cin >> i; cin >> X; cin >> x;
cout << "y=" << y << " pow(y,1.8)=" << exp(1.8*log(y)) << endl;
cout << "i=" << i;  cout << "   x=" << x << "   pow(x,i)=" <<
pow(x, i) << endl;
cout << "X=" << X << "   pow(X,1/i)=" << -exp(1.0/i*log(abs(X)))
<< endl;
cout << " pow(2.43,-1.3)=" <<1/exp(2.43*log(abs(x))) << endl;
system("Pause");
return 0;     }
```

Результаты работы программы



Ввод данных с помощью функции cin.
Вывод данных с помощью функции cout

#include <iostream>

using namespace std;

Оператор ввода

***cin>>*имя объекта;**

Оператор вывода

***cout<<* выражение;**

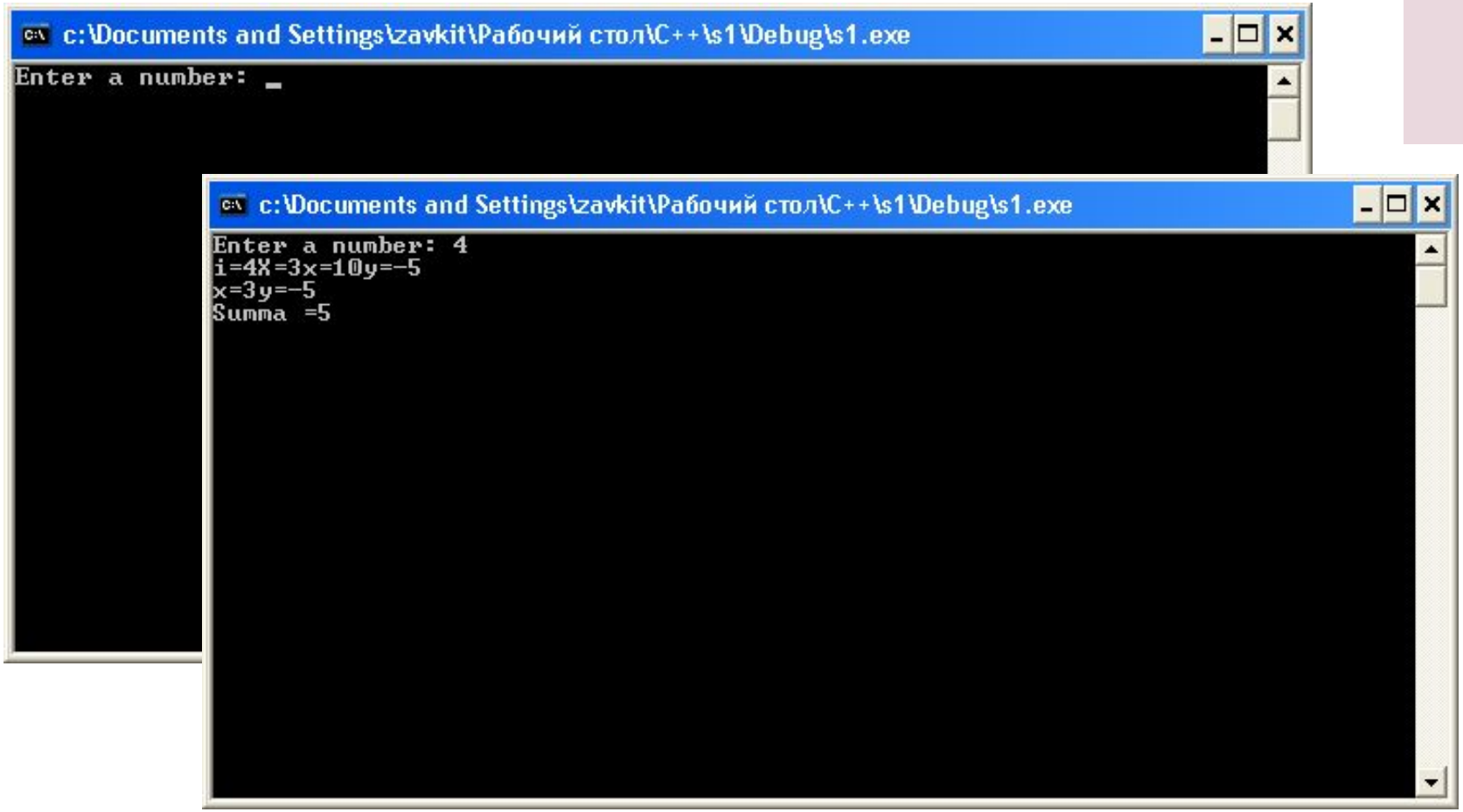
Примеры

```
cin >>n;
cin >>xn>>xk>>dx;

cout<<"X="<<X;
cout<<"Summa ="<<x+y;
```

Ввод данных с помощью функции cin.
Вывод данных с помощью функции cout

```
#include <iostream>
using namespace std;

int main()
{int i = 0;
int X=3;
int x=10;
int y=-5;
cout<< "Enter a number: ";
cin>>i;
   cout<<"i="<<i;
   cout<<"X="<<X;
cout<<"x="<<x<<"y="<<y<<
"\n";

cout<<"x="<<X<<"y="<<y<<
endl;
cout<<"Summa ="<<x+y;

}
```

Результаты работы программы



Условный оператор IF

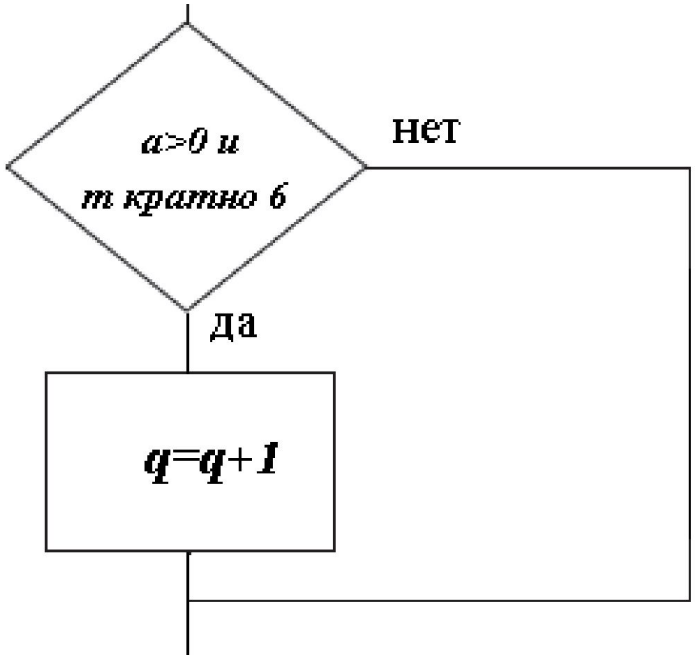
IF < выражение> <оператор 1>**; [ELSE** <оператор 2>]
Оператор выполняется таким образом: если результат вычисления выражения не равен 0 (**TRUE)**, то выполняется <оператор 1>, затем <следующий оператор >; если – равен 0 (**FALSE)**, то выполняется <оператор 2>, затем <следующий оператор>. Операторы 1 и 2 могут быть простыми или составными. Если часть оператора, начинающаяся **ELSE**, отсутствует, то при логическом выражении равным **FALSE**, будет выполняться <следующий оператор>. При вложенности условных операторов **ELSE** всегда относится к ближайшему предшествующему **IF**. Следует избегать большой глубины вложенности условных операторов, так как при этом теряется наглядность и возможно появление ошибок.

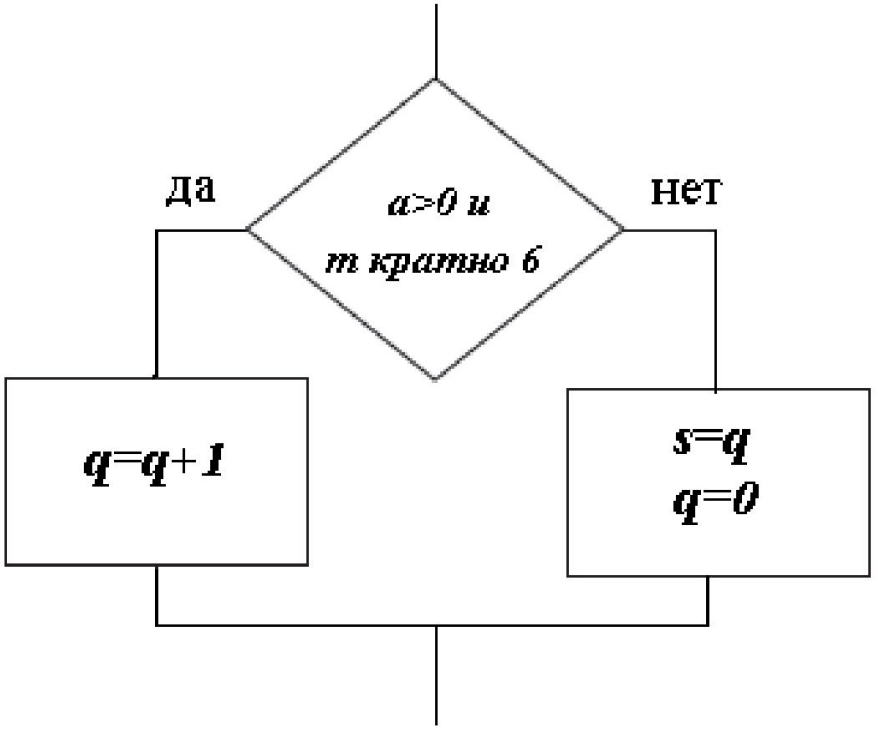
Условный оператор IF

Примеры.

If ((a>0) &&(m%6==0)) q+=1;

if ((a>0) &&(m%6==0)) q+=1; else {s=q; q=0;}





Оператор варианта switch

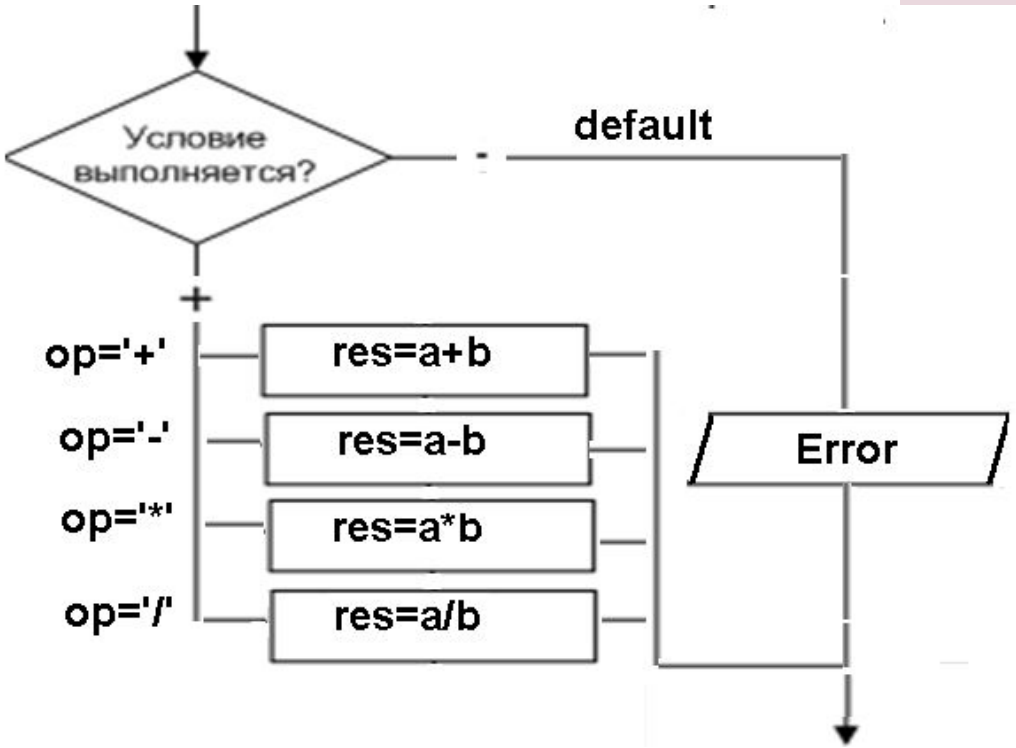
Оператор ***switch*** предназначен для варианта ветви вычислительного процесса в зависимости от значения параметра. Оператор switch имеет следующую структуру:

switch (параметр)
{
case значение_1: Операторы_1; break;
case значение_2: Операторы_2; break;
case значение_3: Операторы_3; break;
…
default: Операторы; break;
}

Значение **параметра** должен быть целым.

Пример:

```
op=‘*‘;
switch (op) {
case ‘+’: res=a+b; break;
case ‘-’: res=a-b; break;
case ‘*’: res=a*b; break;
case ‘/’: res=a/b; break;
default  cout<<“\nError”;
}
```



Циклом в программировании называют повторение одних и тех же действий (шагов). Последовательность действий, которые повторяются в цикле, называют *телом цикла*.

1. Цикл с предусловием. Оператор while

while (условие) оператор;

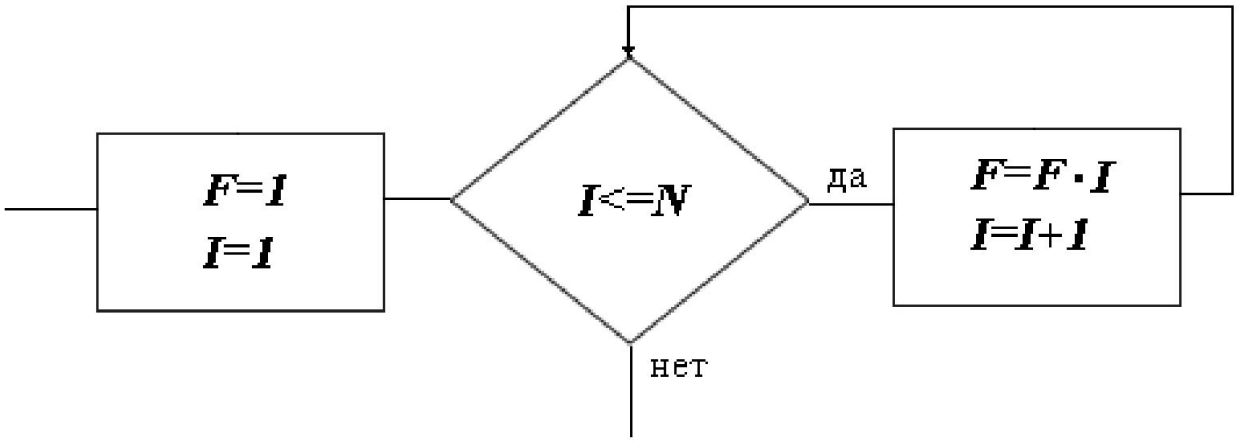
```
while (условие)
{
оператор 1;
оператор 2;
…
оператор n;
}
```

Циклические алгоритмы, их реализация в С++

Вычислить значение факториала F=N!

Фрагмент программы:

```
…
F=1;  I=1;
     WHILE (I<=N)
{    F*=I;
          I++;
}
…
```



Циклические алгоритмы, их реализация в С++

2. Цикл с постусловием. Оператор do-while

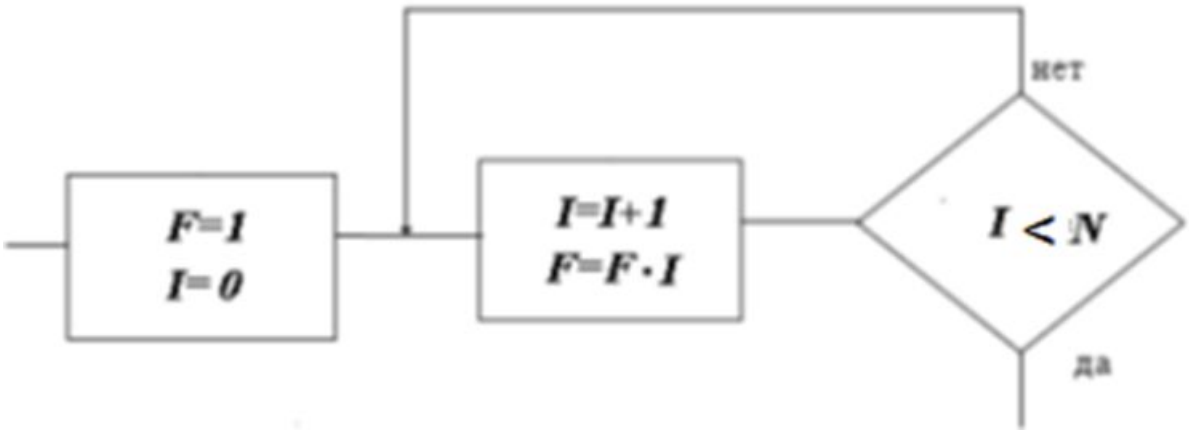
```
do
{
        оператор;

} while (условие);
```

Циклические алгоритмы, их реализация в С++

Вычислить значение факториала F=N!

Фрагмент программы:

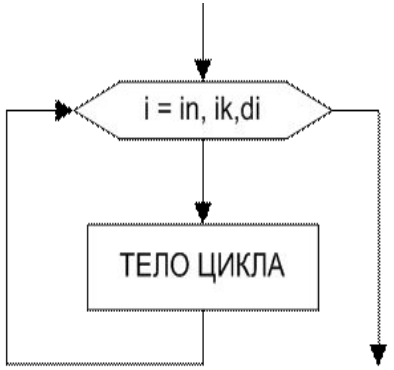
```
…
F = 1;  I = 0;
    do
    {I++;
     F *= I;
    } while (I < N);
…
```



Цикл с параметром

Оператор for

```
for(начальные_присваивания ; условие; приращение)
оператор;
for(начальные_присваивания ; условие; приращение)
{
оператор 1;
оператор 2;
…
оператор n;
}
for (i=in; i<=ik; i=i+di)
оператор;
for (i=in; i<=ik; i+=di)
{
операторы;
}
```



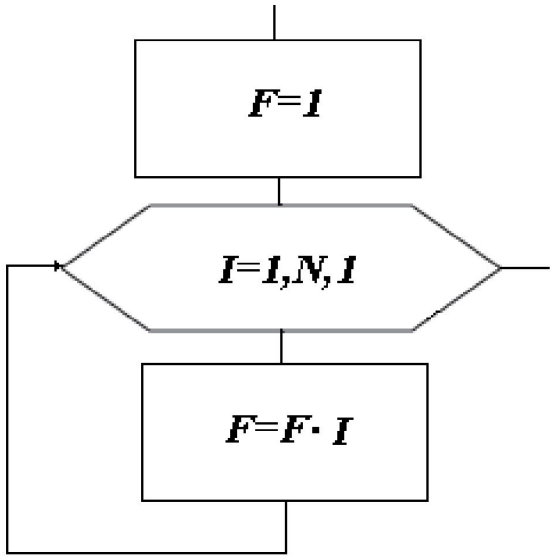
Цикл с параметром

Вычислить значение факториала
F=N!
Фрагмент программы:
Решение 1:

```
…
for (F=1, I=1; I<=N; I++)
F*=I;
….
```

Решение 2:

```
…
F = 1;
for (int I = 1; I < N; I++, F *= I);
…
```



Операторы break, continue

Оператор **break** осуществляет немедленный выход из циклов **while**, **do-while** и **for**. Его можно использовать внутри циклов.

Оператор **continue** начинает новую итерацию цикла, даже если предыдущая не была завершена. Его можно использовать только внутри цикла.

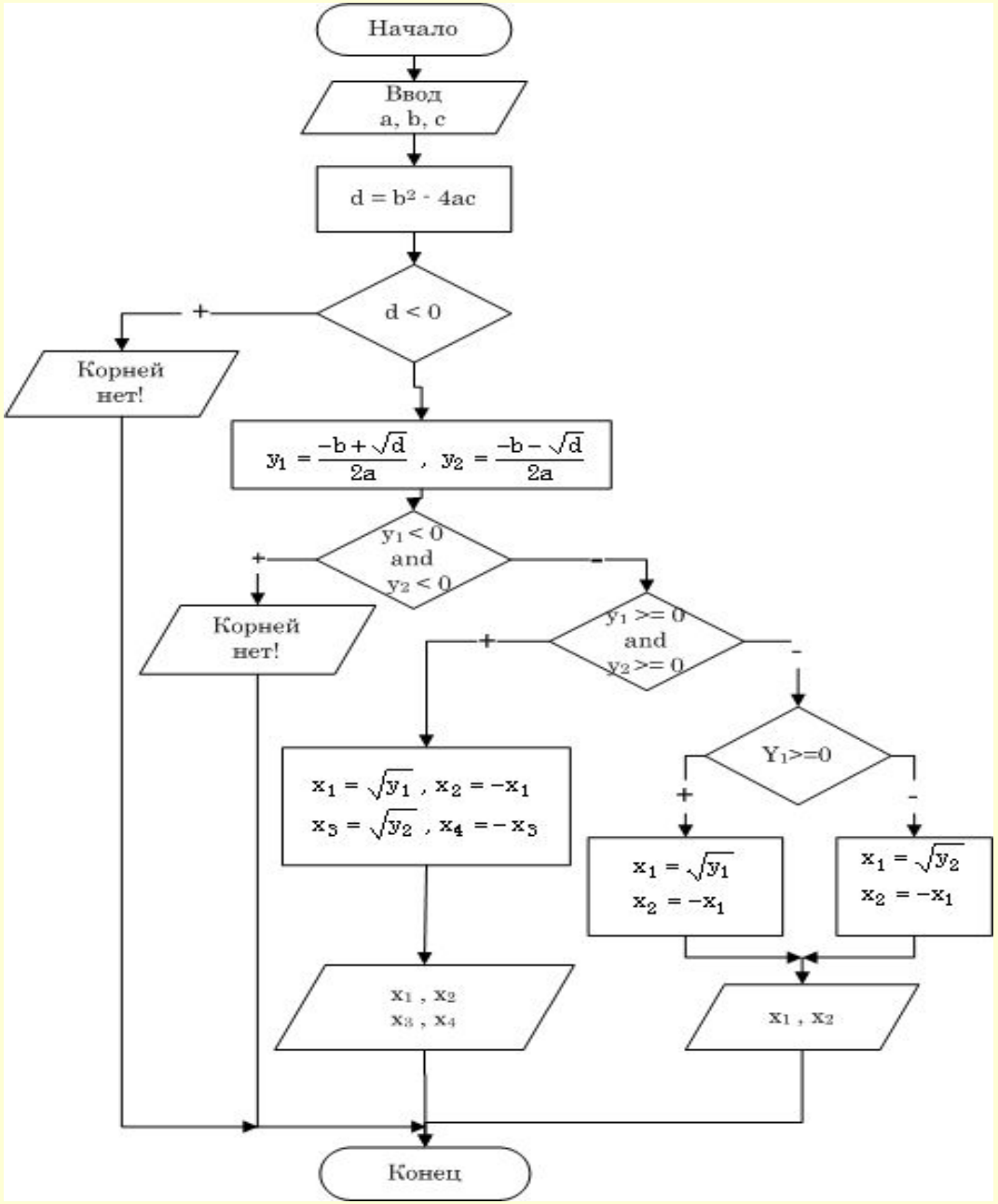
Пример

Заданы коэффициенты a, b и c биквадратного уравнения $ax^4 + bx^2 + c = 0$. Решить уравнение.

Дано:
a, b, c – коэффициенты биквадратного уравнения.
Найти:
x1, x2, x3, x4 – корни уравнения.

Для решения биквадратного уравнения необходимо заменой $y = x^2$ привести его к квадратному и решить это уравнение.

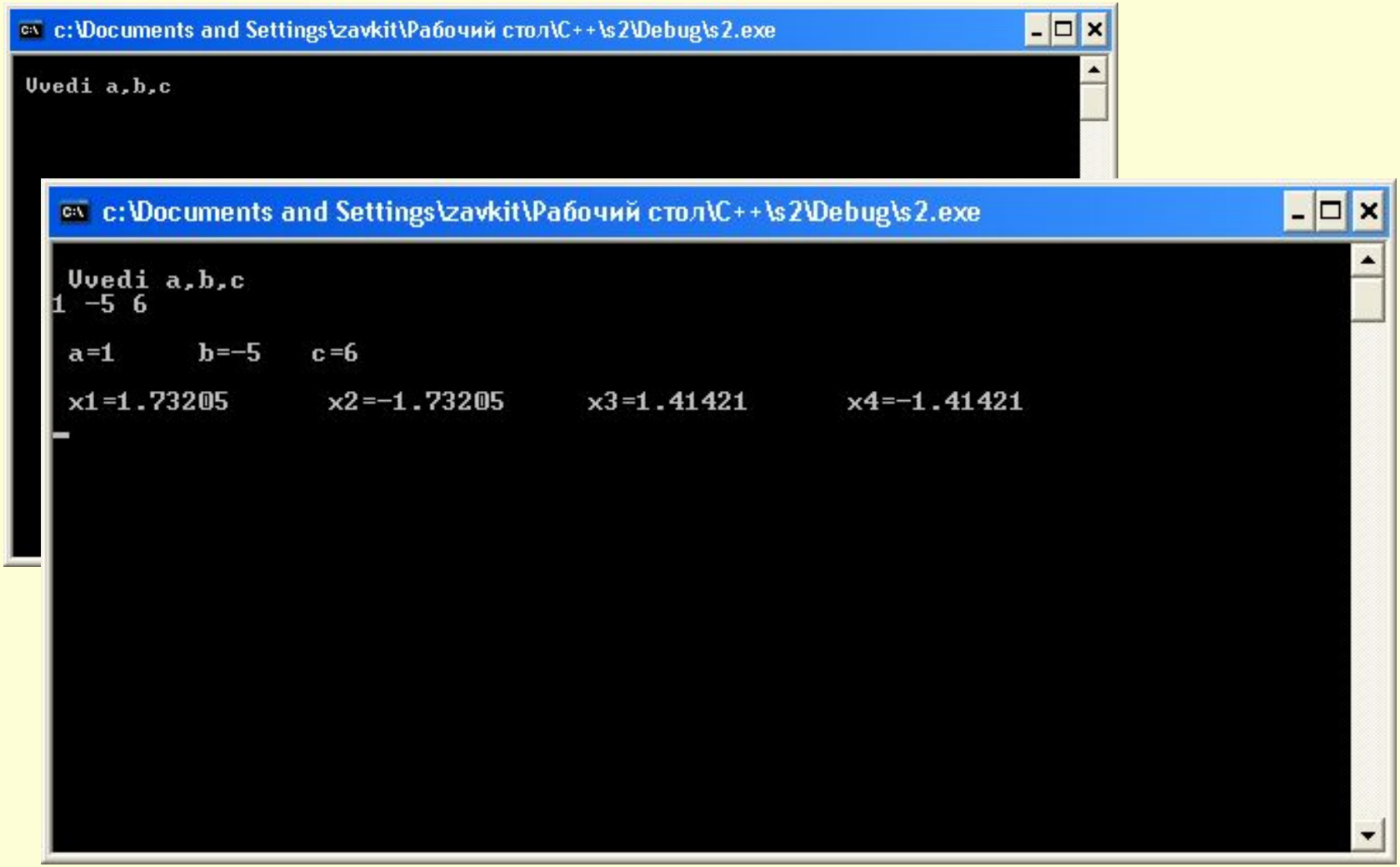
Блок - схема


Начало
Ввод
a, b, c
d = b² - 4ac
d < 0
+
Корней
нет!
$y_1 = \frac{-b+\sqrt{d}}{2a}$, $y_2 = \frac{-b-\sqrt{d}}{2a}$
$y_1 < 0$
and
$y_2 < 0$
+
Корней
нет!
-
$y_1 >= 0$
and
$y_2 >= 0$
+
$x_1 = \sqrt{y_1}$, $x_2 = -x_1$
$x_3 = \sqrt{y_2}$, $x_4 = -x_3$
x_1 , x_2
x_3 , x_4
-
$Y_1>=0$
+
$x_1 = \sqrt{y_1}$
$x_2 = -x_1$
-
$x_1 = \sqrt{y_2}$
$x_2 = -x_1$
x_1 , x_2
Конец

Текст программы

```
#include <iostream>
#include  <cmath>
 using namespace std;
int  main()
{
float a,b,c,d,y1,y2,x1,x2,x3,x4;
printf("\n Vvedi a,b,c\n");
scanf_s("%f%f%f",&a,&b,&c);
printf("\n a=%g\tb =%g\tc=%g\n",
a,b,c);
d=b*b-4*a*c;
If (d<0) printf("No \n");
else { y1=(-b+sqrt(d))/2/a;
y2=(-b-sqrt(d))/2/a;
if ((y1<0) && (y2<0)) printf("No\n");
else if((y1>=0) &&(y2>=0))
{ x1=sqrt(y1); x2=-x1; x3=sqrt(y2);
x4=-sqrt(y2);
printf
("\n x1=%g \t x2=%g \t x3=%g \t4=%g\n"
, x1,x2,x3,x4);
} else
if (y1>=0)
{ x1=sqrt(y1); x2=-x1;
printf("\n x1=%g \t x2=%g \n", x1,x2); }
else
{ x1=sqrt(y2); x2=-x1;
printf("\n x1=%g \t x2=%g \n", x1,x2);
} }
}
```

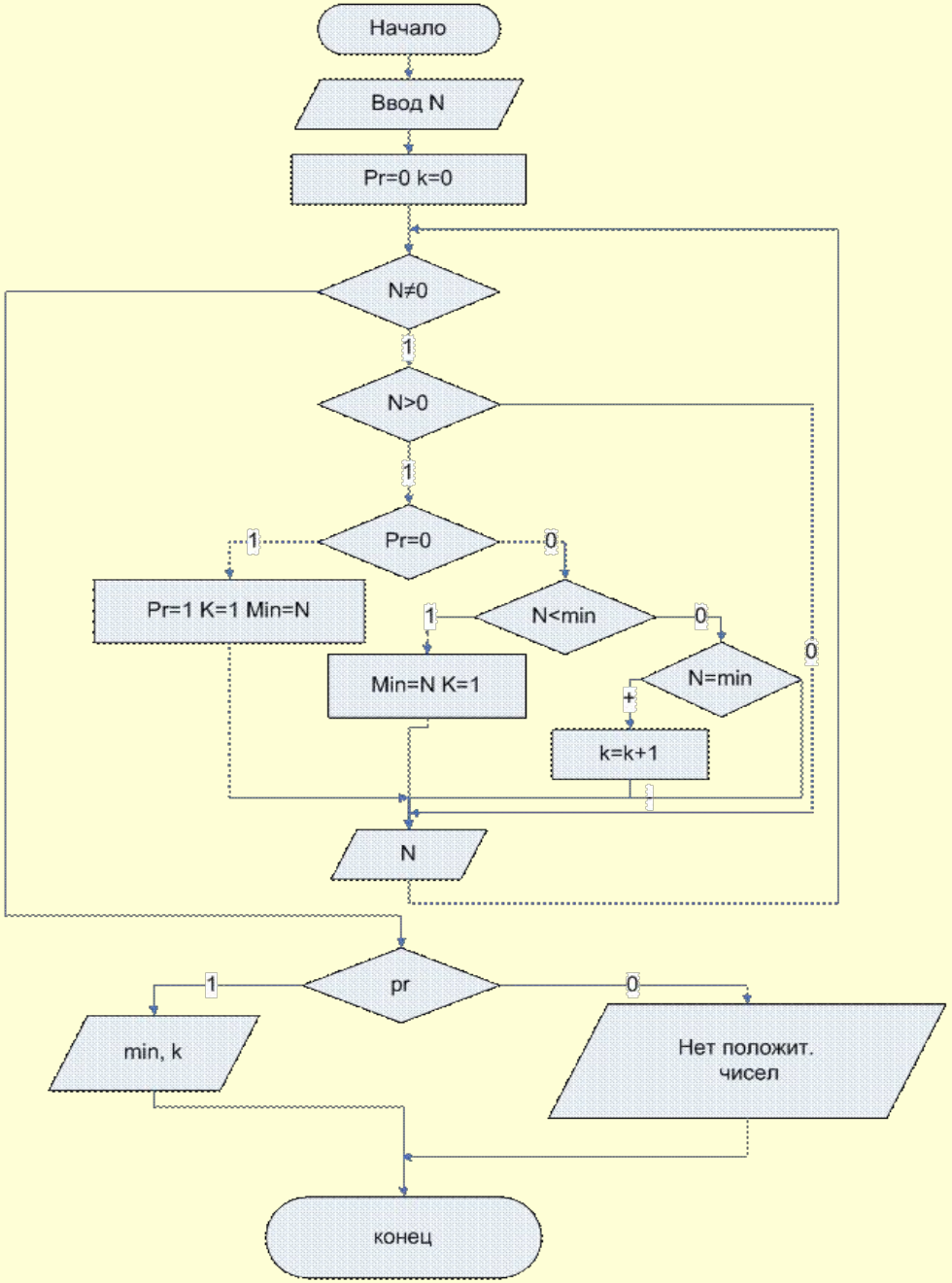
Результаты


c:\Documents and Settings\zavkit\Рабочий стол\C++\s2\Debug\s2.exe
Uvedi a,b,c
c:\Documents and Settings\zavkit\Рабочий стол\C++\s2\Debug\s2.exe
Uvedi a,b,c
1 -5 6
a=1 b=-5 c=6
x1=1.73205 x2=-1.73205 x3=1.41421 x4=-1.41421

Пример

Вводится последовательность целых чисел, 0 – конец последовательности. Найти минимальное среди положительных, если таких значений несколько, определить, сколько их.

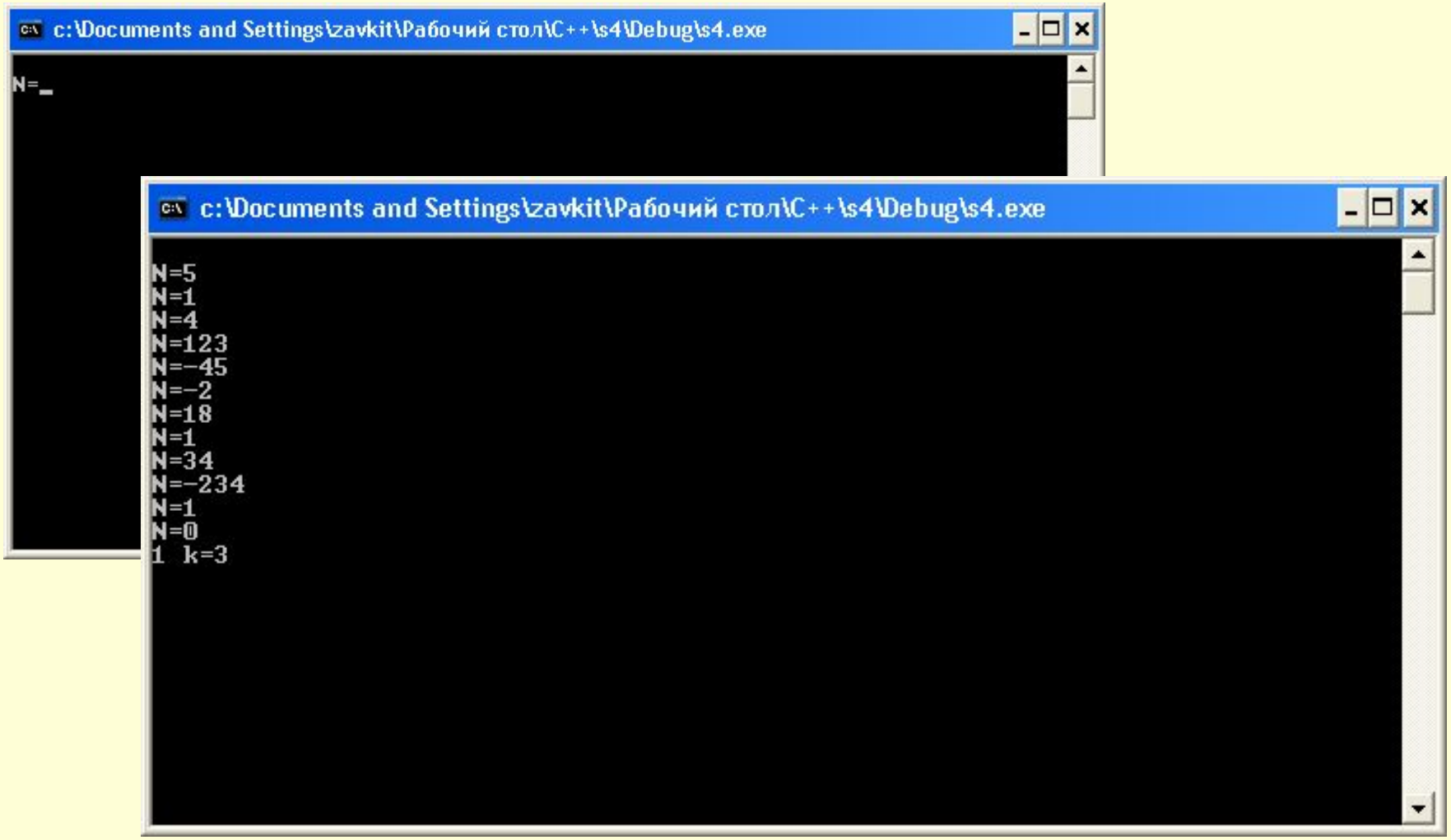
Блок - схема


Начало
Ввод N
Pr=0 k=0
N≠0
1
N>0
1
Pr=0
1
0
Pr=1 K=1 Min=N
1
N<min
0
Min=N K=1
N=min
0
+
k=k+1
N
pr
1
0
min, k
Нет положит. чисел
конец

Текст программы

```
#include "stdafx.h"
#include <iostream>
#include <windows.h>
#include <conio.h>
using namespace std;

int main(int argc, char* argv[])
{ int N,pr,k,min;
cout<<"\nN=";
cin>>N;
for(pr=k=0;N!=0;cout<<"N=",cin>>N)
if(N>0)
if (pr==0)
{ pr=1; min=N; k=1; }
else
if (N<min) { min=N; k=1; }
else
if (N==min) k++;
if (pr)
{cout<<min;
cout<<"k=";
cout<<k<<endl;
}
else
cout<<"В последовательности
нет положительных чисел";
 _getch();
}
```

Результаты


c:\Documents and Settings\zavkit\Рабочий стол\C++\s4\Debug\s4.exe
N=_
c:\Documents and Settings\zavkit\Рабочий стол\C++\s4\Debug\s4.exe
N=5
N=1
N=4
N=123
N=-45
N=-2
N=18
N=1
N=34
N=-234
N=1
N=0
1 k=3

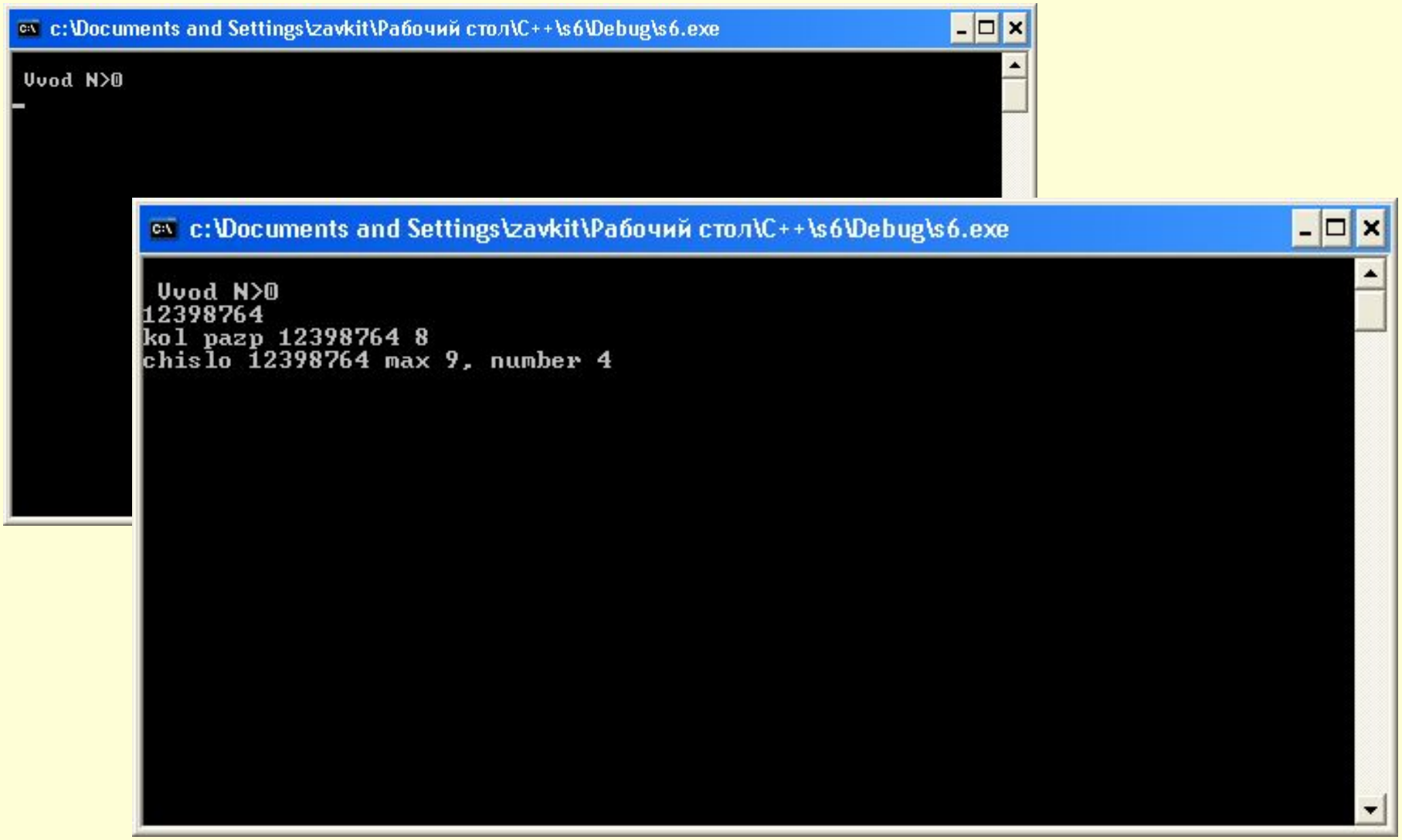
Пример

Дано натуральное число N. Определить самую большую цифру и ее позицию в числе (N=573863, наибольшей является цифра 8, ее позиция – четвертая слева).

Текст программы

```
#include "....."
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
long int N,M,kol=1;
int max,pos,i;
printf("\n Vvod N>0\n");
scanf_s("%ld",&N);
M=N;
while(M/10>0)
{
kol++;
M/=10;
}
printf("kol pazp %ld %ld \n",
N,kol);
for(M=N,max=-1, pos=1, i=kol; i>=1; i--)
{
if (M%10>max)
{
max=M%10;
pos=i;
}
M/=10;
}
printf("chislo %ld max %d, number %d\n",
N,max,pos);
}
```

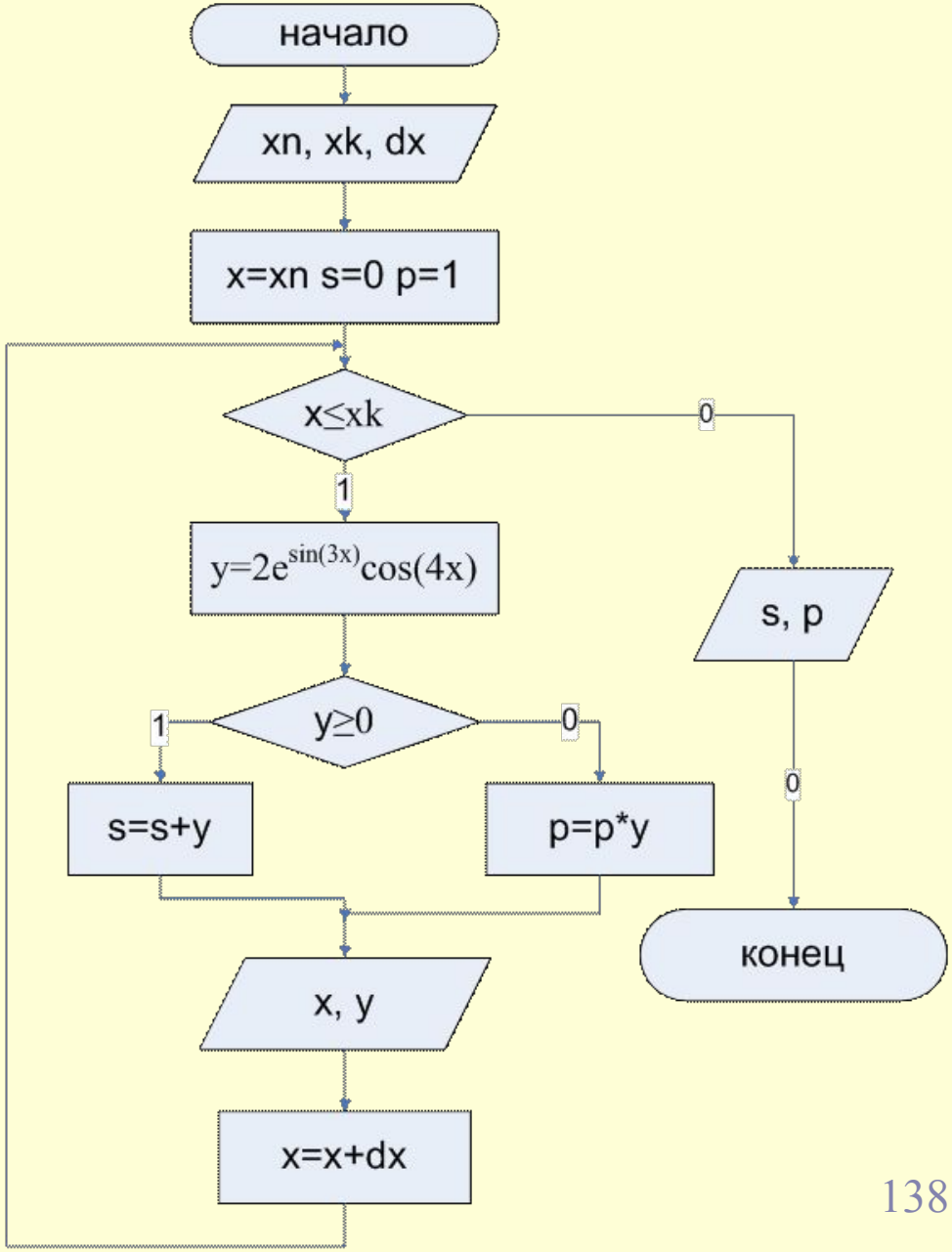
Результаты


c:\Documents and Settings\zavkit\Рабочий стол\C++\s6\Debug\s6.exe
Vvod N>0
c:\Documents and Settings\zavkit\Рабочий стол\C++\s6\Debug\s6.exe
Vvod N>0
12398764
kol pazp 12398764 8
chislo 12398764 max 9, number 4

Пример

Составить таблицу значений функции $y=2e^{\sin(3x)}\cos(4x)$ на отрезке [xn; xk] с шагом dx. Найти сумму положительных y и произведение отрицательных y.

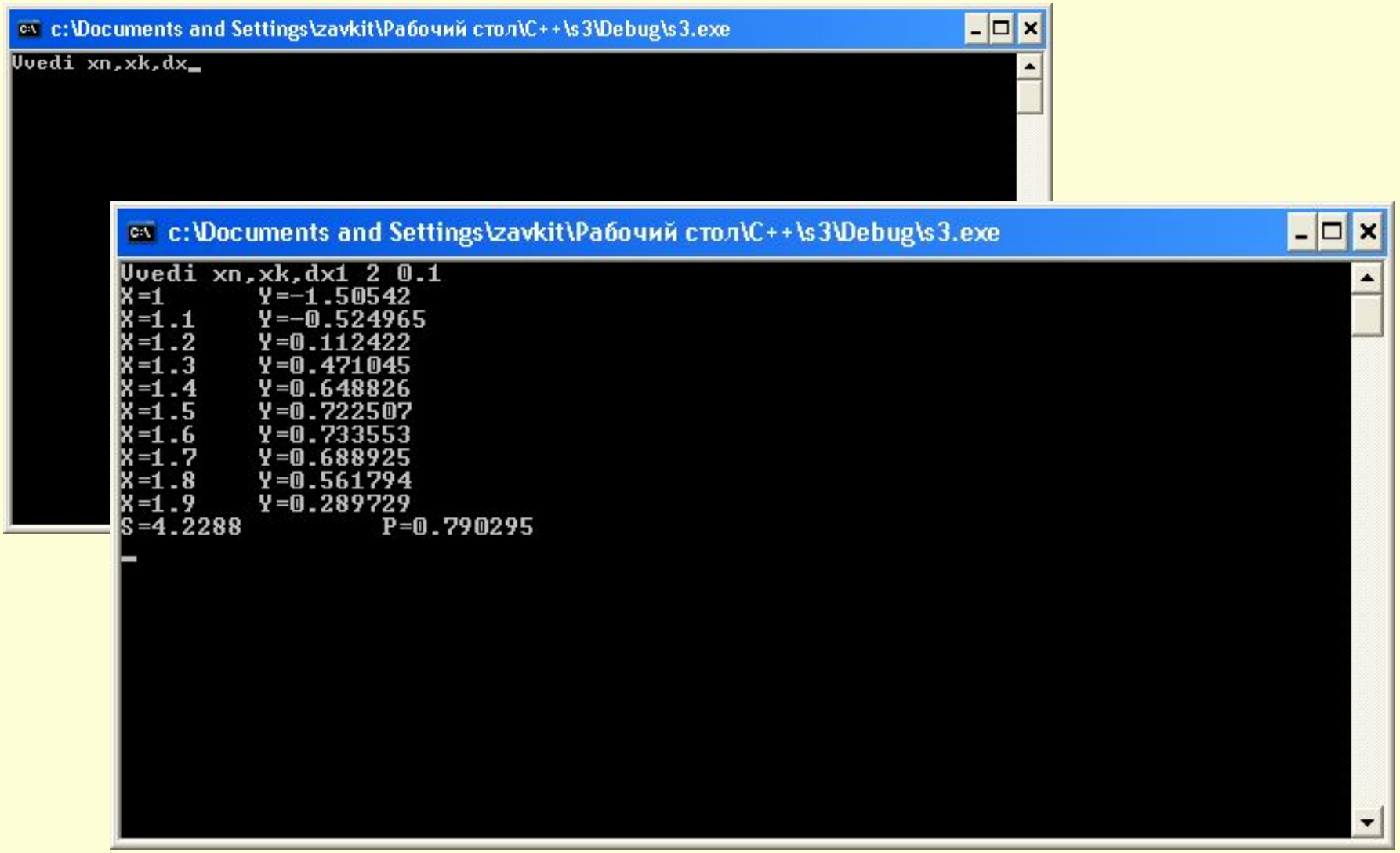
Блок - схема


начало
xn, xk, dx
x=xn s=0 p=1
x≤xk
0
1
y=2e^sin(3x)cos(4x)
s, p
y≥0
1
0
0
s=s+y
p=p*y
конец
x, y
x=x+dx

Текст программы

```
#include <iostream>
#include  <cmath>

using namespace std;

int main()
{
float xn, xk, dx, x, y, s, p;
cout<<"xn,xk,dx= ";
cin>>xn>>xk>>dx;
for(s=0,p=1,x=xn;x<=xk;x+=dx)
{
y=2*exp(sin(3*x))*cos(4*x);
cout<< " x= "<<x<< "   y= "<<y<<endl;
if (y>=0) s+=y;
else p*=y;
}
cout<< " s= "<<s<< "   p= "<<p;
}
```

Результаты


c:\Documents and Settings\zavkit\Рабочий стол\C++\s3\Debug\s3.exe
Vvedi xn,xk,dx
c:\Documents and Settings\zavkit\Рабочий стол\C++\s3\Debug\s3.exe
Vvedi xn,xk,dx1 2 0.1
X=1 Y=-1.50542
X=1.1 Y=-0.524965
X=1.2 Y=0.112422
X=1.3 Y=0.471045
X=1.4 Y=0.648826
X=1.5 Y=0.722507
X=1.6 Y=0.733553
X=1.7 Y=0.688925
X=1.8 Y=0.561794
X=1.9 Y=0.289729
S=4.2288 P=0.790295

Пример

Условие задачи. Построить таблицу значений альтернативно заданной функции f(d)

$$f=\begin{cases}\dfrac{1}{\sqrt{d}}\operatorname{arctg}\dfrac{b+c}{\sqrt{d}}, & \text{если } d>0;\\ -\dfrac{1}{b+c}, & \text{если } d=0;\\ \dfrac{1}{2\sqrt{-d}}\ln\sqrt{-d}, & \text{если } d<0.\end{cases}$$

d изменяется от начального значения dn до конечного dk с шагом dh. Значения b и c заданы.

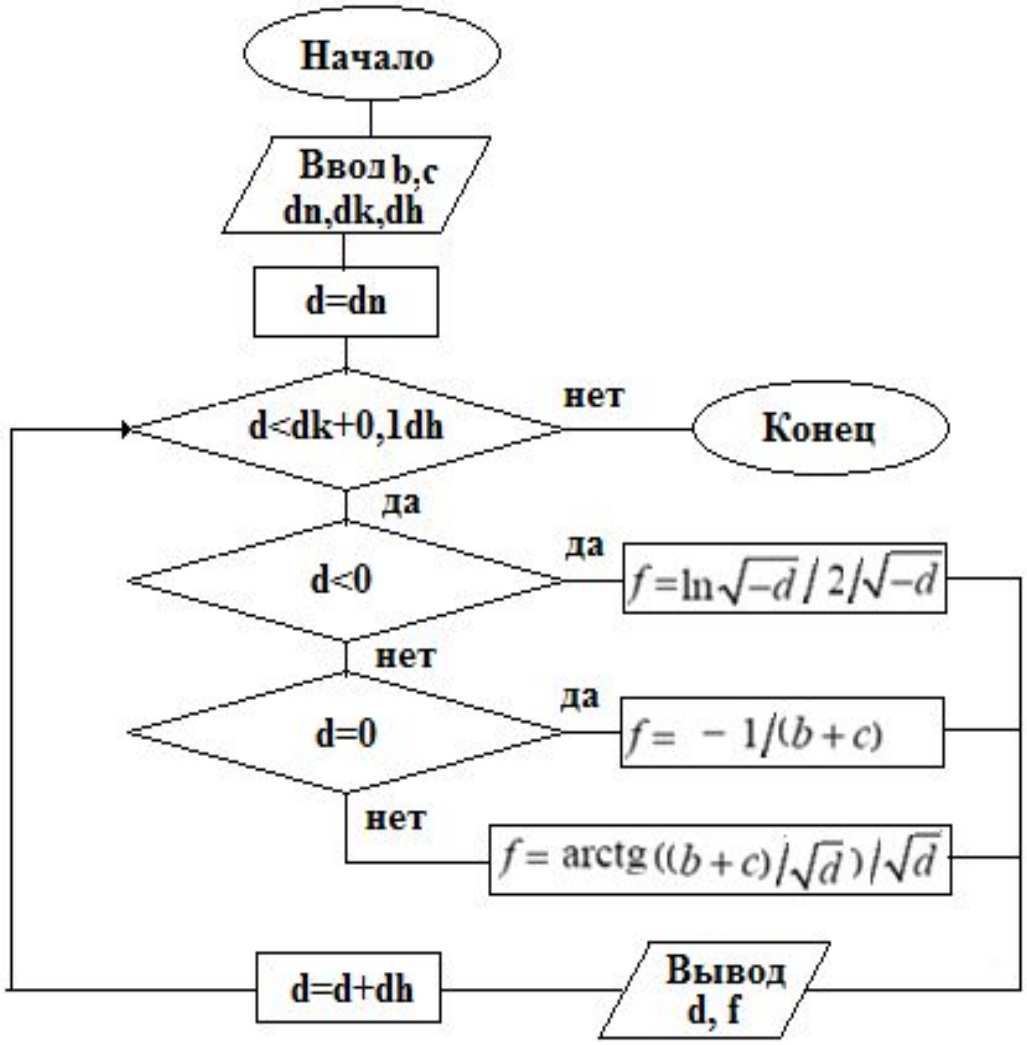
Решение задачи

Формализация задачи

Дано: **b, c, dn, dk,dh**.

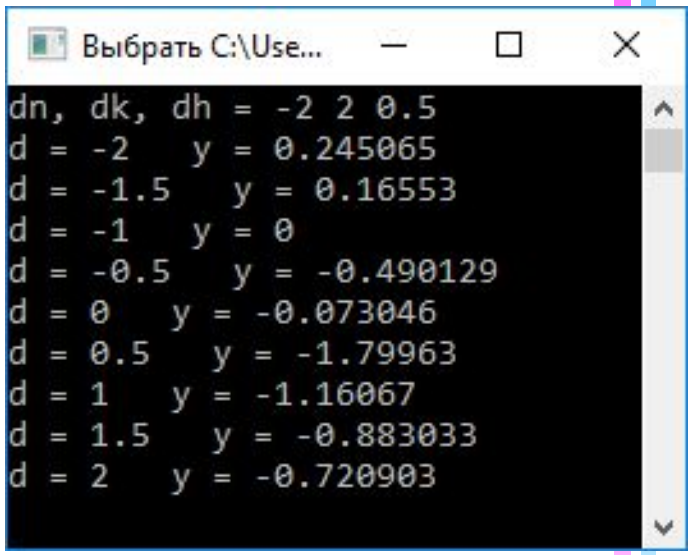
Найти: **f**.

Блок-схема


Начало
Ввод b,c
dn,dk,dh
d=dn
d<dk+0,1dh
нет
Конец
да
d<0
да
$f=\ln\sqrt{-d}\,/\,2/\sqrt{-d}$
нет
d=0
да
$f=-1/(b+c)$
нет
$f=\operatorname{arctg}((b+c)/\sqrt{d})/\sqrt{d}$
d=d+dh
Вывод
d, f

Программа

```
#include <iostream>
#include  <cmath>
using namespace std;
int main()
{    double dn, dk, dh, d, s, y,
     b=1.2, c=-3.5;
     cout << "dn, dk, dh = ";
     cin >> dn >> dk >> dh;
     for ( d = dn; d <= dk+0.1* dh; d += dh)
     {       if (d > 0) y = 1 / sqrt(d)*atan((b + c) / sqrt(d));
          else if (d == 0) y = -1 / (b*b + c*c);
          else y = 1 / sqrt(-d)* log(sqrt(-d));
          cout <<"d = " << d << "   y = " << y << endl;
          }
      return 0;
}
```



Условие задачи

Для x, изменяющегося в интервале от x0 до xk с шагом h,

вычислить значения бесконечной суммы $\mathbf{S(x)} = \sum_{n=0}^{\infty} \frac{(-1)^n (2x)^n}{(2n)!}$
с точностью ε=0.00001 и функции

$$\mathbf{y(x)} = \cos(\sqrt{2x})$$

Формализация задачи

Дано: **x0, xk, h**.

Найти: **S, y**.

Вывод рекуррентной формулы для расчета текущего члена ряда.

Формула общего члена ряда $u_i = \frac{-1^i \cdot (2 \cdot x)^i}{(2 \cdot i)!}$

$i = 0 \quad u_0 = \frac{-1^0 \cdot (2 \cdot x)^0}{(2 \cdot 0)!} = 1$

$i = 1 \quad u_1 = \frac{-1^1 \cdot (2 \cdot x)^1}{(2 \cdot 1)!} = -x$

……

$i = n \quad u_n = \frac{-1^n \cdot (2 \cdot x)^n}{(2 \cdot n)!}$

$i = n + 1 \quad u_{n+1} = \frac{-1^{n+1} \cdot (2 \cdot x)^{n+1}}{(2 \cdot (n+1))!} = \frac{-1^{n+1} \cdot (2 \cdot x)^{n+1}}{(2 \cdot n + 2)!}$

Найдем отношение:

$$\frac{u_{n+1}}{u_n} = \frac{-1^{n+1} \cdot (2 \cdot x)^{n+1} \cdot (2 \cdot n)!}{(2 \cdot n + 2)! \cdot -1^n \cdot (2 \cdot x)^n} = -\frac{2 \cdot x}{(2 \cdot n + 1) \cdot (2 \cdot n + 2)}$$

$$= -\frac{x}{(n+1) \cdot (2 \cdot n + 1)}$$

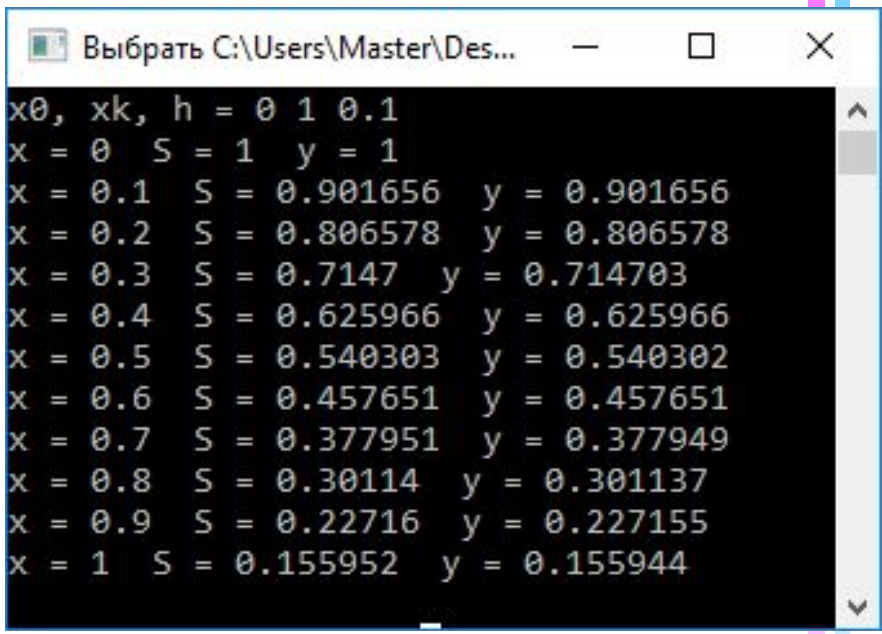
Выразим u_{n+1} и получим рекуррентную формулу

$$u_{n+1} = -\frac{x}{(n+1) \cdot (2 \cdot n + 1)} \cdot u_n$$

Эта формула позволяет вычислить любой текущий член ряда кроме u_0.

Фрагмент текста программы

```
double x0, x, xk, h, u,s,y,e=1E-5;
int n;
cout << "x0, xk, h = ";
cin >> x0>> xk >> h;
for (x = x0; x <= xk+0.1* h; x += h)
{
s = 0; u = 1; n = 0;
while (abs(u) > e)
{
s += u;
u = -x / ((n + 1)*(2 * n + 1))*u;
n++;
}
y = cos(sqrt(2 * x));
cout << "x = " << x << "  S = " << s << "  y = " << y << endl;
}
```



Использование функций

Подпрограмма – именованная, логически законченная группа операторов языка, которую можно вызвать для выполнения любое количество раз из различных мест программы. В языке **C/C++** подпрограммы реализованы в виде функций. Функция принимает параметры и возвращает единственное скалярное значение.
Функция должна быть описана перед своим использованием. Описание функции состоит из заголовка и тела функции.

Заголовок_функции
{
тело_функции
}

Использование функций

Заголовок функции имеет вид
<тип> имя_функции ([список параметров])
<тип> – тип возвращаемого функцией значения;
список параметров – список передаваемых в функцию величин, которые отделяются запятыми, каждому параметру должен предшествовать его тип;

В случае, если вызываемые функции идут до функции main, структура программы будет такой.

директивы компилятора

...

Тип_результата f1(Список_переменных)

{ Операторы }

Тип_результата f2(Список_переменных)

{ Операторы }

...

Тип_результата fn(Список_переменных)

{ Операторы}

int main(Список_переменных)

{

Операторы основной функции, среди которых могут операторы вызова функций f1, f2, ..., fn

}

Если вызываемые функции идут после функции main, структура программы будет такой (заголовки функций должны быть описаны до функции main()). Опережающие заголовки функций называют **прототипами (**сигнатурами) функций.

директивы компилятора

...

Тип_результата f1(Список_переменных);

Тип_результата f2(Список_переменных);

...

Тип_результата fn(Список_переменных);

int main(Список_переменных)

{Операторы основной функции, среди которых могут операторы вызова функций f1, f2, ..., fn }

Тип_результата f1(Список_переменных)

{ Операторы }

Тип_результата f2(Список_переменных)

{ Операторы }

...

Тип_результата fn(Список_переменных)

{ Операторы }

Для того, чтобы функция вернула какое-либо значение, в ней должен быть оператор
return значение;
Для вызова функции необходимо указать имя функции и в круглых скобках список передаваемых в функцию значений.

Передача параметров

Параметры, указанные в заголовке функции, называются **формальными**. Параметры, передаваемые в функцию, называются **фактическими**.
При обращении к функции фактические параметры передают свое значение формальным и больше не изменяются. **Типы, количество и порядок следования формальных и фактических параметров должны совпадать.** С помощью оператора ***return*** из функции возвращается единственное значение.

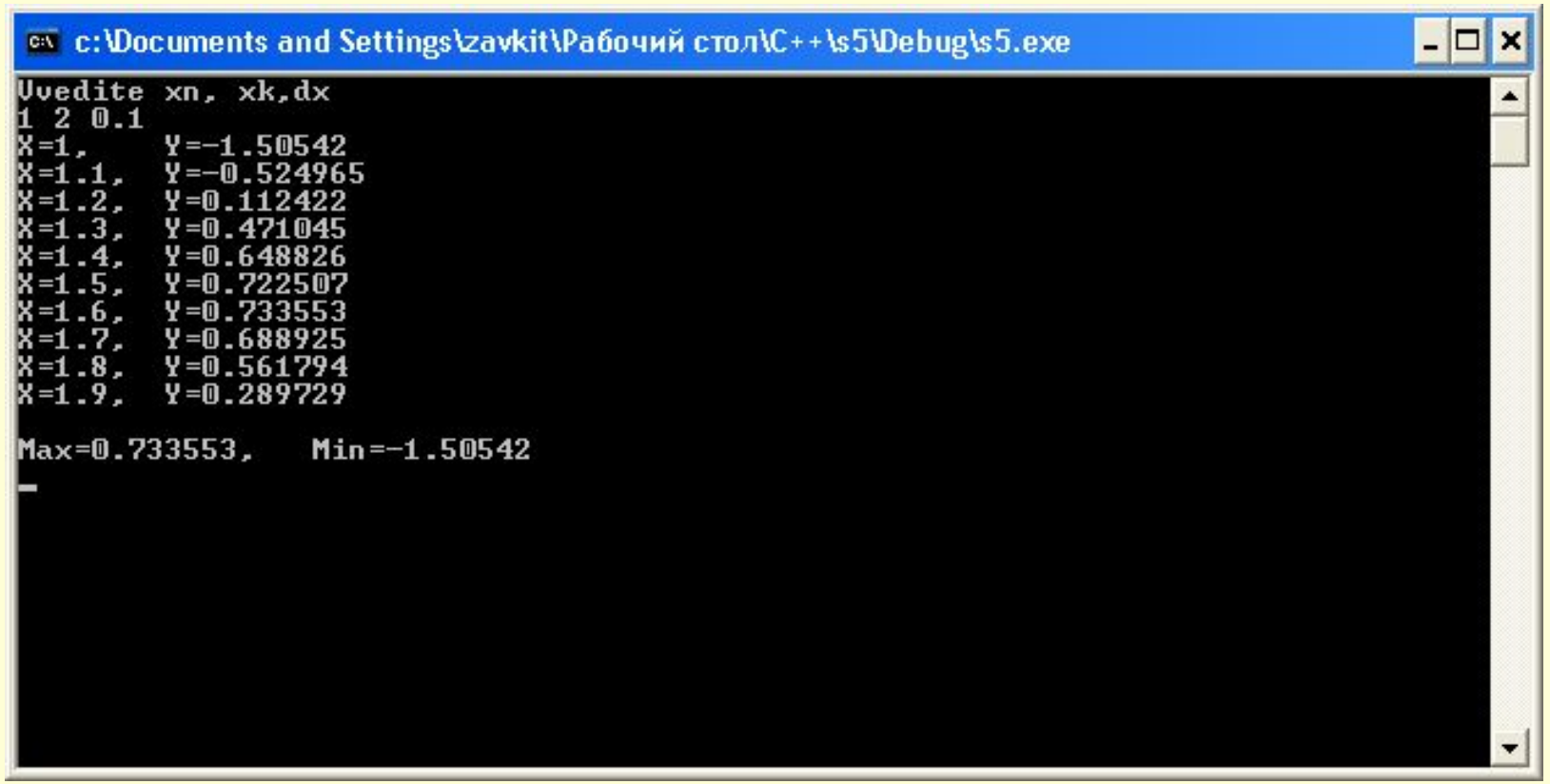
Пример

Составить таблицу значений функции $y=2e^{\sin(3x)}\cos(4x)$ на отрезке [xn; xk] с шагом dx. Найти максимальное и минимальное значение y. *Расчет **y** оформить в виде функции*.

Текст программы

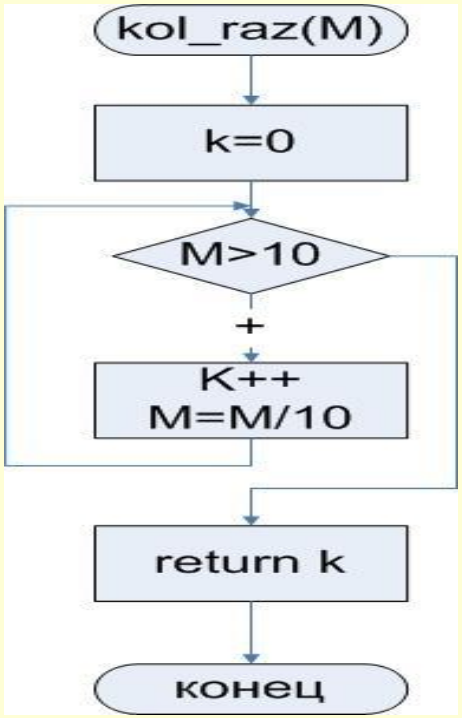
```
#include <iostream>
#include  <cmath>

using namespace std;
float f(float x, float a, float b, float
c);
int  main()
{
float xn, xk, dx, x, y, max,min;
int k=0;
cout<<"Vvedite xn, xk,dx\n";
cin>>xn>>xk>>dx;
for(x=xn;x<=xk;x+=dx)
{
        y=f(x,2,3,4);
cout<<"X="<<x<<",\tY="<<y<<endl;
        k++;        if (k==1)
        {       max=y;
                min=y;        }
        else
        {   if (y>max) max=y;
            if (y<min) min=y;        }}
cout<<endl<<"Max="<<max<<",
\tMin="<<min<<endl;
}
float f(float x, float a, float b, float c)
{
        float y;
        y=a*exp(sin(b*x))*cos(c*x);
        return y;
}
```

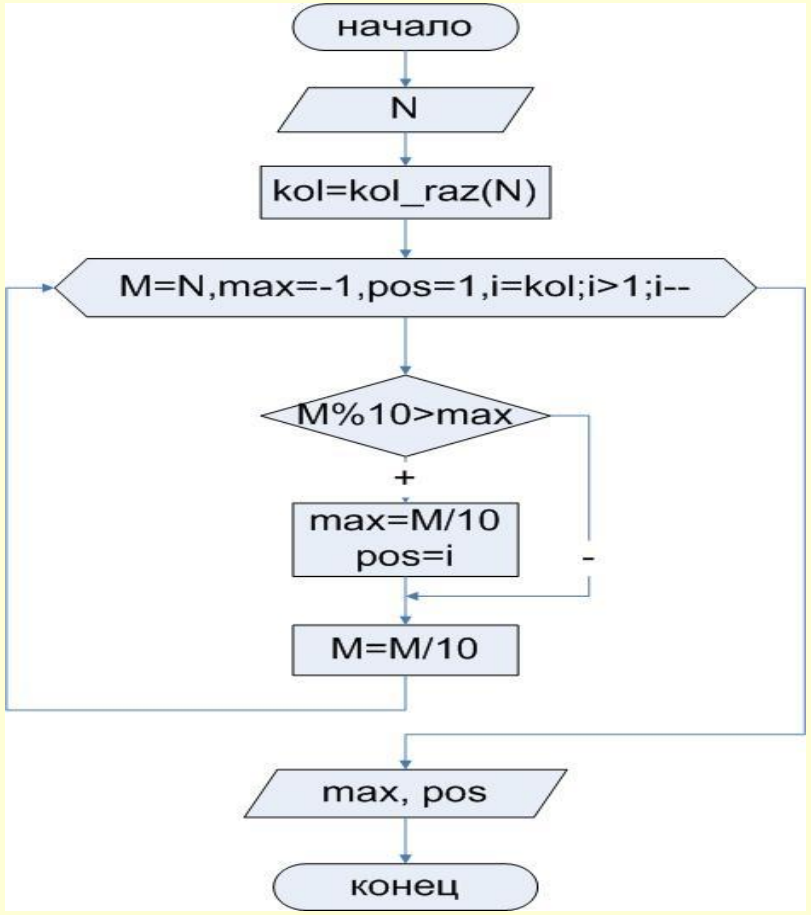
Результаты


c:\Documents and Settings\zavkit\Рабочий стол\C++\s5\Debug\s5.exe
Uvedite xn, xk,dx
1 2 0.1
X=1, Y=-1.50542
X=1.1, Y=-0.524965
X=1.2, Y=0.112422
X=1.3, Y=0.471045
X=1.4, Y=0.648826
X=1.5, Y=0.722507
X=1.6, Y=0.733553
X=1.7, Y=0.688925
X=1.8, Y=0.561794
X=1.9, Y=0.289729
Max=0.733553, Min=-1.50542

Пример

Дано натуральное число N. Определить самую большую цифру и ее позицию в числе (N=573863, наибольшей является цифра 8, ее позиция – четвертая слева)

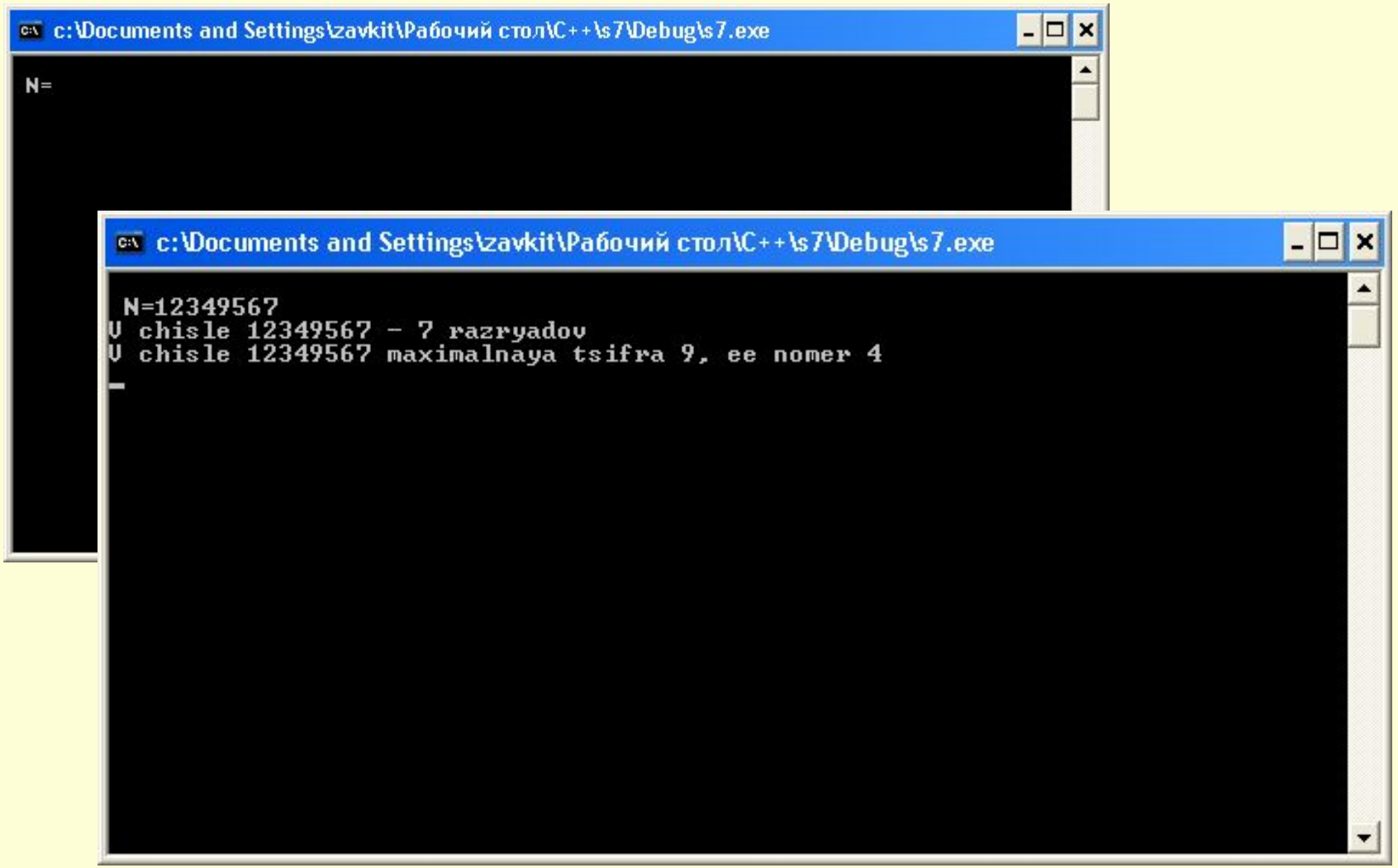




Текст программы

```
#include <cmath>
#include <iostream>
using namespace std;

int kol_raz(int M)
{ int k=0; while(M/10>0)
{ k++; M/=10;}
return k;
}
int main()
{ long int N,M,kol=1;
int max,pos,i;
printf("\n N="); // Ввод числа N
scanf("%ld",&N);
// Вычисление количества разрядов в числе (kol).
kol=kol_raz(N);
printf("V chisle %ld - %ld razryadov \n“,
N,kol);
// Вычисление максим. цифры в числе, и ее номера.
for(M=N, max=-1, pos=1, i=kol;i>1;i--)
{ if (M%10>max)
{
max=M%10;
pos=i;
}
M/=10; }
// Вывод на экран максимальной цифры
// в числе, и ее номера.
printf("V chisle %ld maximalnaya
tsifra %d, ee nomer %d\n",
N,max,pos);
}
```

Результаты


c:\Documents and Settings\zavkit\Рабочий стол\C++\s7\Debug\s7.exe
N=
c:\Documents and Settings\zavkit\Рабочий стол\C++\s7\Debug\s7.exe
N=12349567
V chisle 12349567 - 7 razryadov
V chisle 12349567 maximalnaya tsifra 9, ee nomer 4

Пример

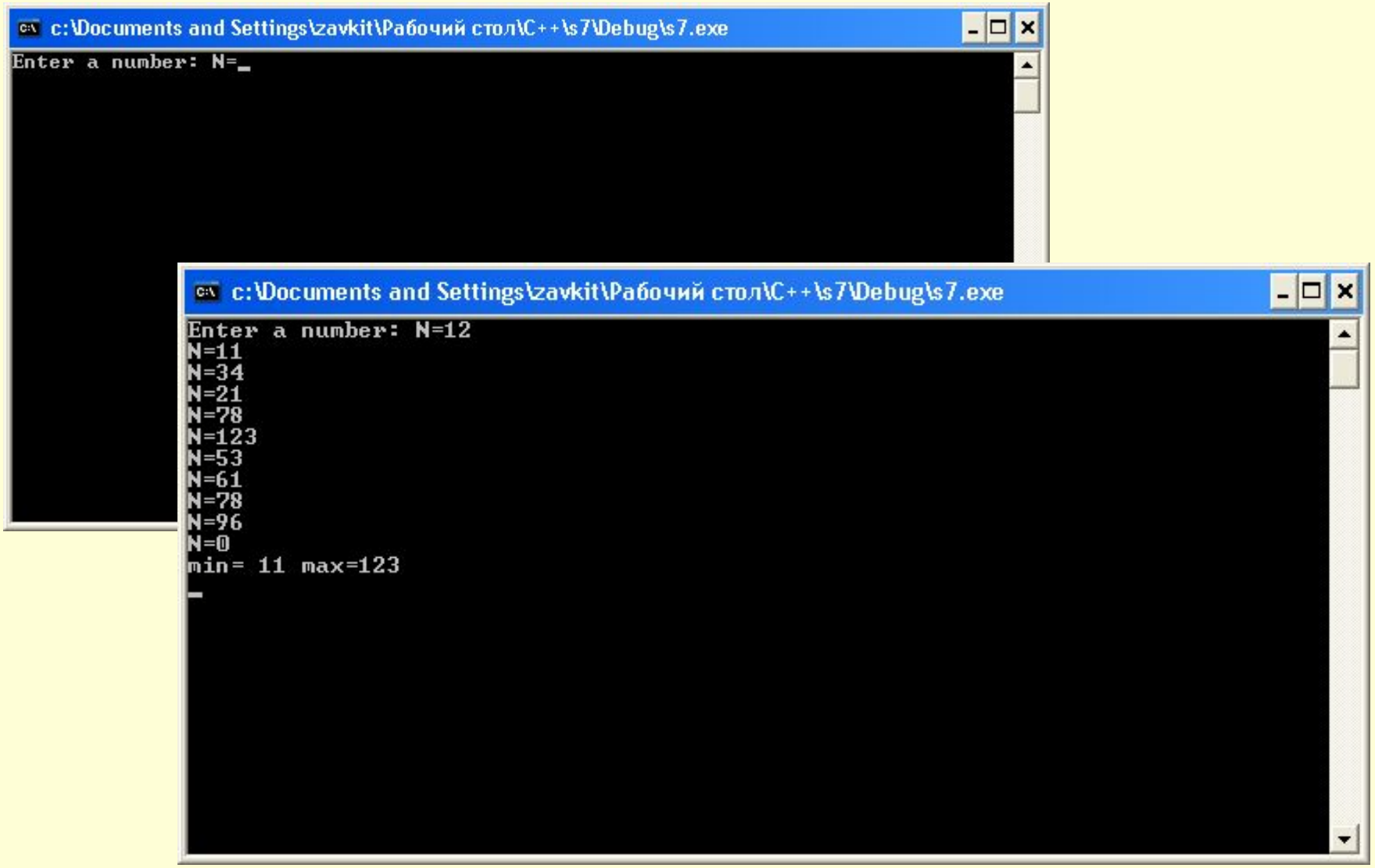
Вводится последовательность целых чисел, 0 – конец последовательности. Найти минимальное среди простых чисел и максимальное, среди чисел, не являющихся простыми.

Целое число называется **простым**, если оно делится нацело только на самого себя и единицу. Алгоритм проверки, что число N является простым состоит в следующем: если разделим N без остатка хотя бы на одно число в диапазоне от 2 до N пополам, то число не является простым. Если не найдем ни одного делителя числа, число N – простое. Проверку является ли число N простым оформим в виде отдельной функции с именем **prostoe**. Входным параметром функции будет целое число N, функция будет возвращать значение 1, если число простое и 0 – в противном случае.

Текст программы

```
#include <iostream>
using namespace std;

int prostoe(int N)
{ int i,pr;
if (N<1) pr=0;
    else
for(pr=1,i=2;i<=N/2;i++)
if (N%i==0) {pr=0;break;}
return pr; }

int  main()
{ int kp=0,knp=0,min,max,N;
cout<< "Enter a number: ";
for (cout<<"N=", cin>>N; N!=0;
cout<<"N=", cin>>N)
    if (prostoe(N))
{       kp++;
      if (kp==1) min=N;
      else if (N<min) min=N;
   }
   else
   {       knp++;
      if (knp==1) max=N;
      else if (N>max) max=N;
   }
if (kp>0) cout<<"min= "<<min<<"\t";
else cout<<"Net prostih";
if (knp>0)
cout<<"max="<<max<<endl;
else cout<<"Net ne prostih";
}
```

Результаты


c:\Documents and Settings\zavkit\Рабочий стол\C++\s7\Debug\s7.exe
Enter a number: N=
c:\Documents and Settings\zavkit\Рабочий стол\C++\s7\Debug\s7.exe
Enter a number: N=12
N=11
N=34
N=21
N=78
N=123
N=53
N=61
N=78
N=96
N=0
min= 11 max=123

Пример

Вводится последовательность из N целых чисел, найти среднее арифметическое совершенных чисел и среднее геометрическое простых чисел.

Число называется **совершенным**, если сумма всех делителей, меньших его самого равна самому числу.

При решении этой задачи понадобятся две функции:

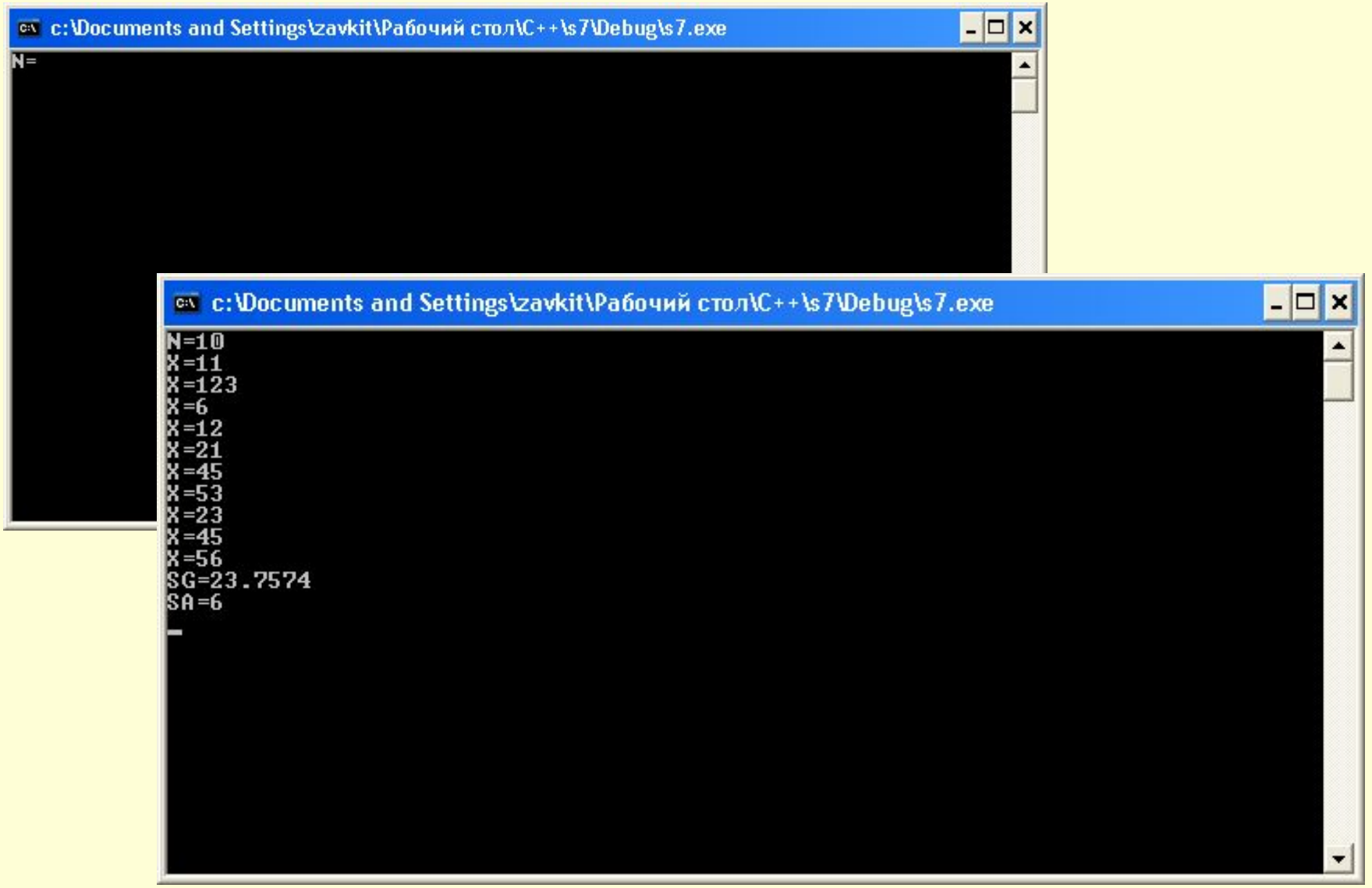
- функция **prostoe**,
- функция **soversh**, которая определяет является ли число совершенным; входным параметром функции будет целое число N, функция будет возвращать значение 1, если число совершенным и 0 – в противном случае.

Текст программы

```
#include <cmath>
#include <iostream>

using namespace std;
int prostoe(int N)
{ int i,pr;
if (N<1) pr=0;      else
for(pr=1,i=2;i<=N/2;i++)
if (N%i==0) {pr=0;break;}
return pr; }
int soversh(int N)
{ int i,S;
if (N<1) return 0;
     else for(S=0,i=1;i<=N/2;i++)
if (N%i==0) S+=i;
if (S==N) return 1; else return 0; }
```

```
int main()
{ int i,N,X,S,kp,ks;  int P;
cout<<"N="; cin>>N;
for(kp=ks=S=0,P=1,i=1;i<=N;i++)
{     cout<<"X=";     cin>> X;
     if (prostoe(X))
     { kp++; P*=X; }
     if (soversh(X))
     { ks++;S+=X;  } }
if (kp>0)
cout<<"SG="<<powf((float)P,(float)1/
kp)<<endl;
else
cout<<"Net prostih";
if (ks>0)
 cout<<"SA="<<(float)S/ks<<endl;
else cout<<"Net soversh";
_getch(); }
```

Результаты


c:\Documents and Settings\zavkit\Рабочий стол\C++\s7\Debug\s7.exe
N=
c:\Documents and Settings\zavkit\Рабочий стол\C++\s7\Debug\s7.exe
N=10
X=11
X=123
X=6
X=12
X=21
X=45
X=53
X=23
X=45
X=56
SG=23.7574
SA=6

Рекурсивные функции

Рекурсия — это такой способ организации вспомогательного алгоритма (подпрограммы), при котором эта подпрограмма (процедура или функция) в ходе выполнения ее операторов обращается *сама к себе*. **Рекурсивным** называется любой объект, который частично определяется через себя. Например, приведенное ниже определение двоичного кода является рекурсивным:

<двоичный код> ::= <двоичная цифра> | <двоичный код>
<двоичная цифра> ::= 0 | 1

Здесь для описания понятия были использованы, так называемые, металингвистический формулы Бэкуса-Наура (язык БНФ); знак "::=" обозначает "по определению есть", знак "|" — "или".

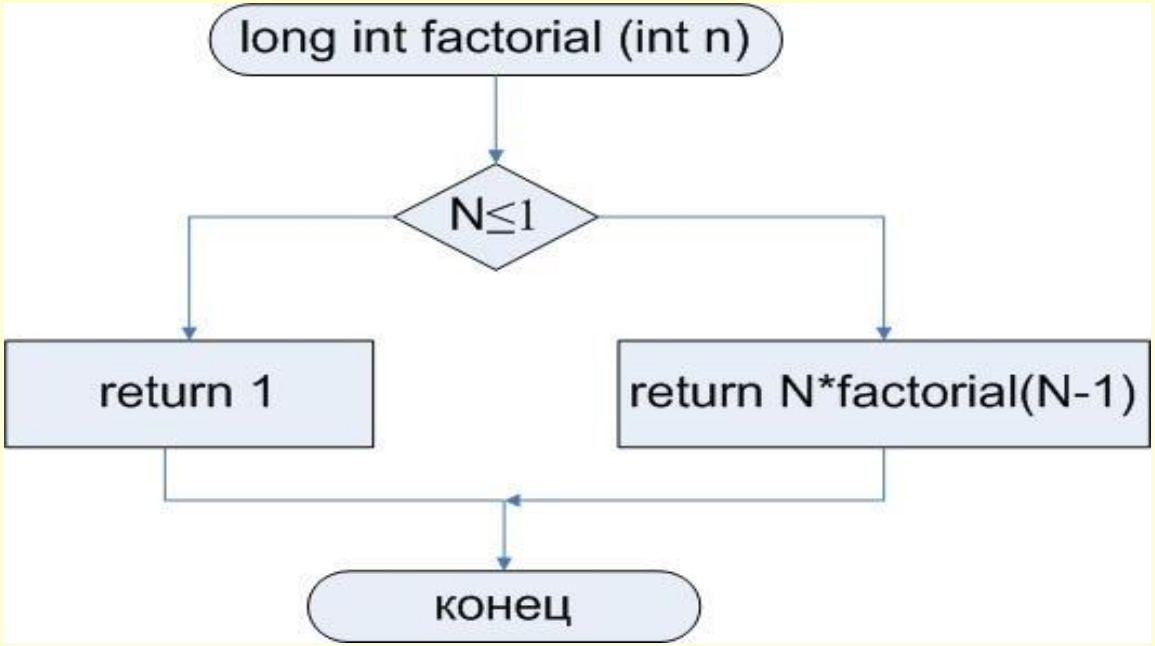
Вообще, в рекурсивном определении должно присутствовать ограничение, *граничное условие*, при выходе на которое дальнейшая инициация рекурсивных обращений прекращается.

Рекурсия бывает прямая и косвенная.

Рекурсивные функции

Рекурсивные версии большинства подпрограмм могут выполняться немного медленнее, чем их итеративные эквиваленты, поскольку к необходимым действиям добавляются вызовы функций. Но в большинстве случаев это не имеет значения. Много рекурсивных вызовов в функции может привести к переполнению стека. Поскольку местом для хранения параметров и локальных переменных функции является стек и каждый новый вызов создает новую копию переменных, пространство стека может исчерпаться. Если это произойдет, то возникнет ошибка - переполнение стека. Основным преимуществом применения рекурсивных функций является использование их для более простого создания версии некоторых алгоритмов по сравнению с итеративными эквивалентами. Например, сортирующий алгоритм Quicksort достаточно трудно реализовать итеративным способом. Некоторые проблемы, особенно связанные с искусственным интеллектом, также используют рекурсивные алгоритмы.

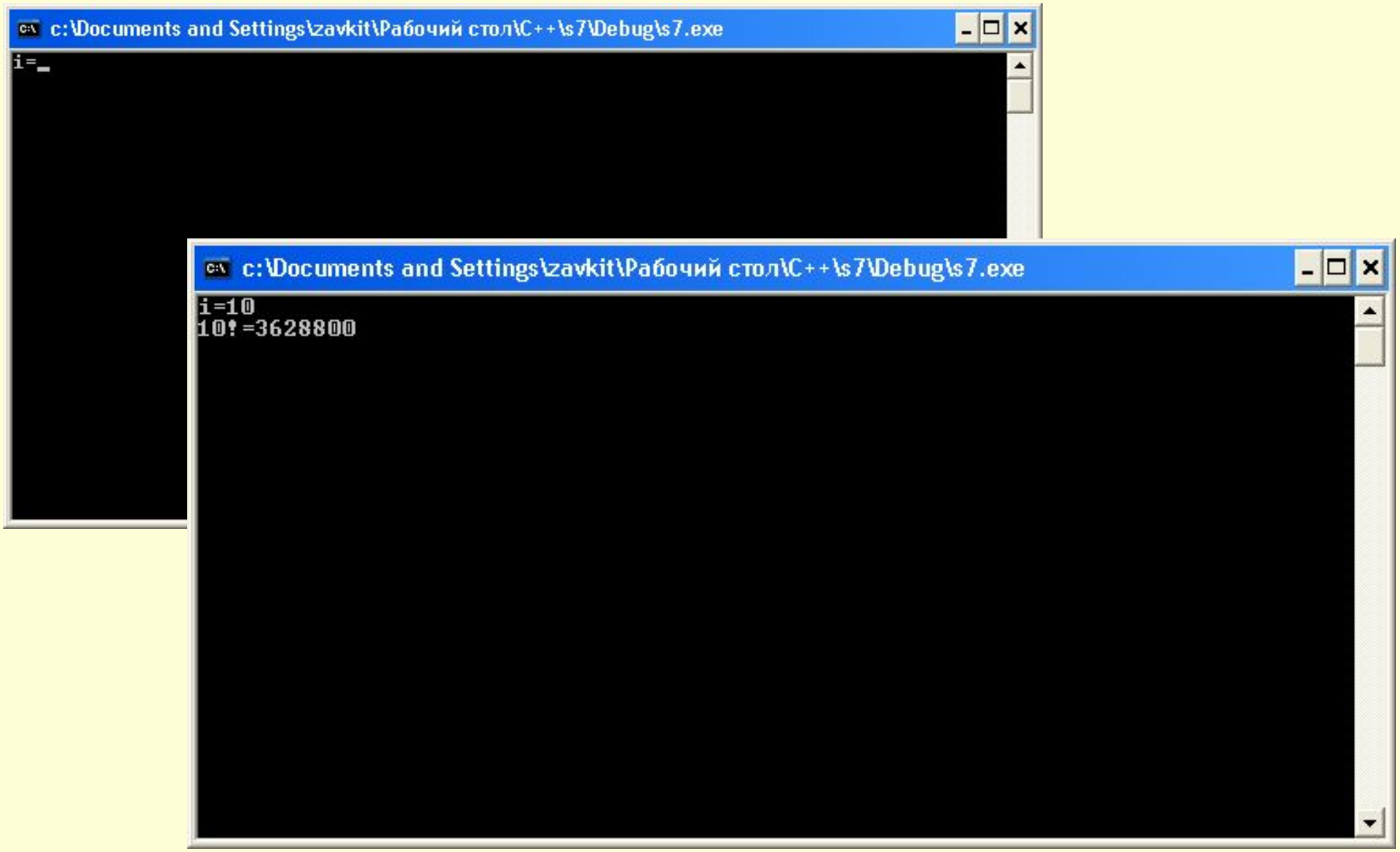
long int factoial(int n)
предназначена для вычисления факториала числа **n**.



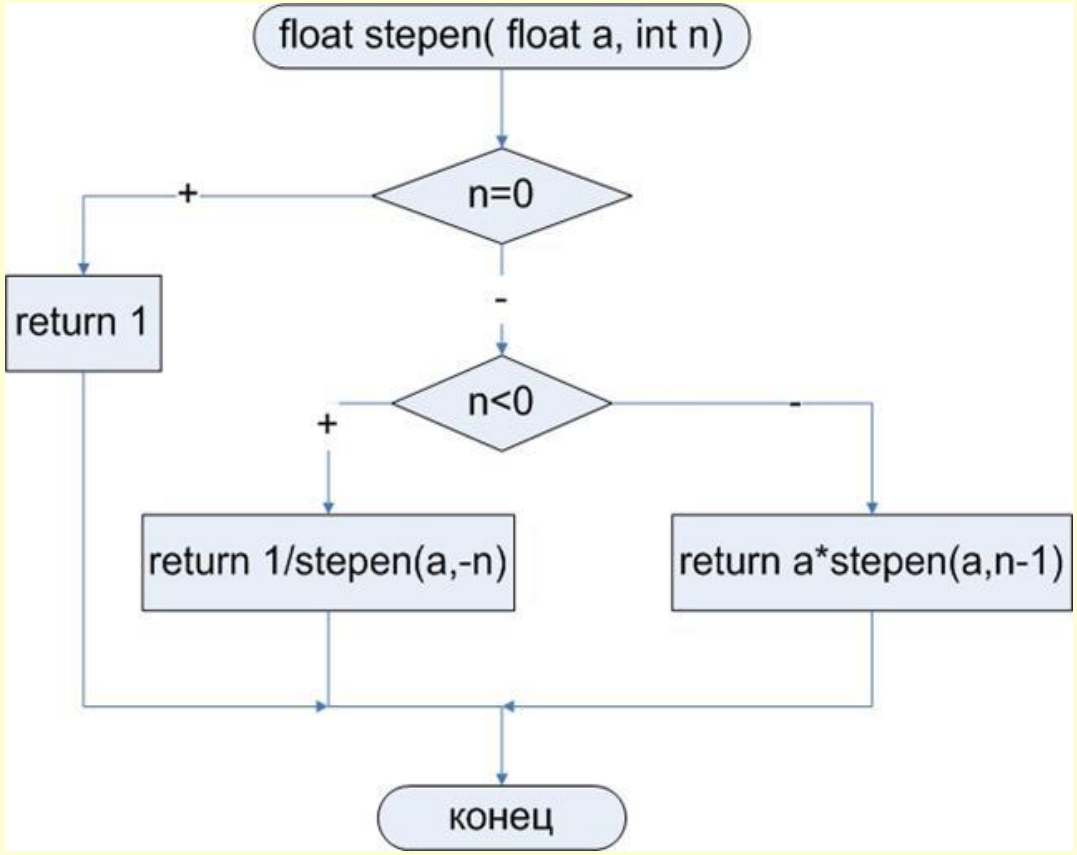
Текст программы

```
#include "stdafx.h"
#include <iostream>

using namespace std;

long int factorial(int n)
{
if (n<=1)
    return(n);
else
    return(n*factorial(n-1));
}

int main()
{
int i;
long int f;
cout<<"i=";
cin>>i;
f=factorial(i);
cout<<i<<"!="<<f<<endl;
}
```

Результаты


c:\Documents and Settings\zavkit\Рабочий стол\C++\s7\Debug\s7.exe
i=_
c:\Documents and Settings\zavkit\Рабочий стол\C++\s7\Debug\s7.exe
i=10
10!=3628800

float stepen(float a, int n) предназначена для возведения числа **a** в степень **n**.

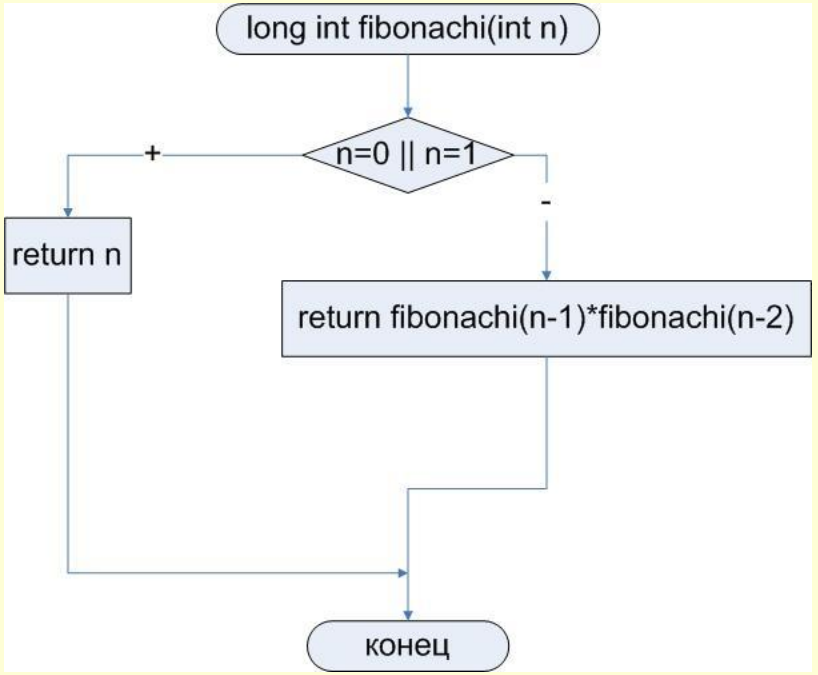


Текст программы

```
#include "........"
#include <cmath >
#include <iostream>

using namespace std;

float stepen(float a, int n)
{
if (n==0)
   return(1);
else
    if (n<0)
      return(1/stepen(a,-n));
    else
       return(a*stepen(a,n-1));
}
int main()
{
int i;
float s,b;
long int f;
cout<<"b=";
cin>>b;
cout<<"i=";
cin>>i;
s=stepen(b,i);
cout<<"s="<<s<<endl;
}
```

long int fibonachi(int n) предназначена для вычисления n-го числа Фибоначчи.

Если нулевой элемент последовательности равен 0, первый – 1, а каждый последующий равен сумме двух предыдущих, то это последовательность чисел Фибоначчи (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...).

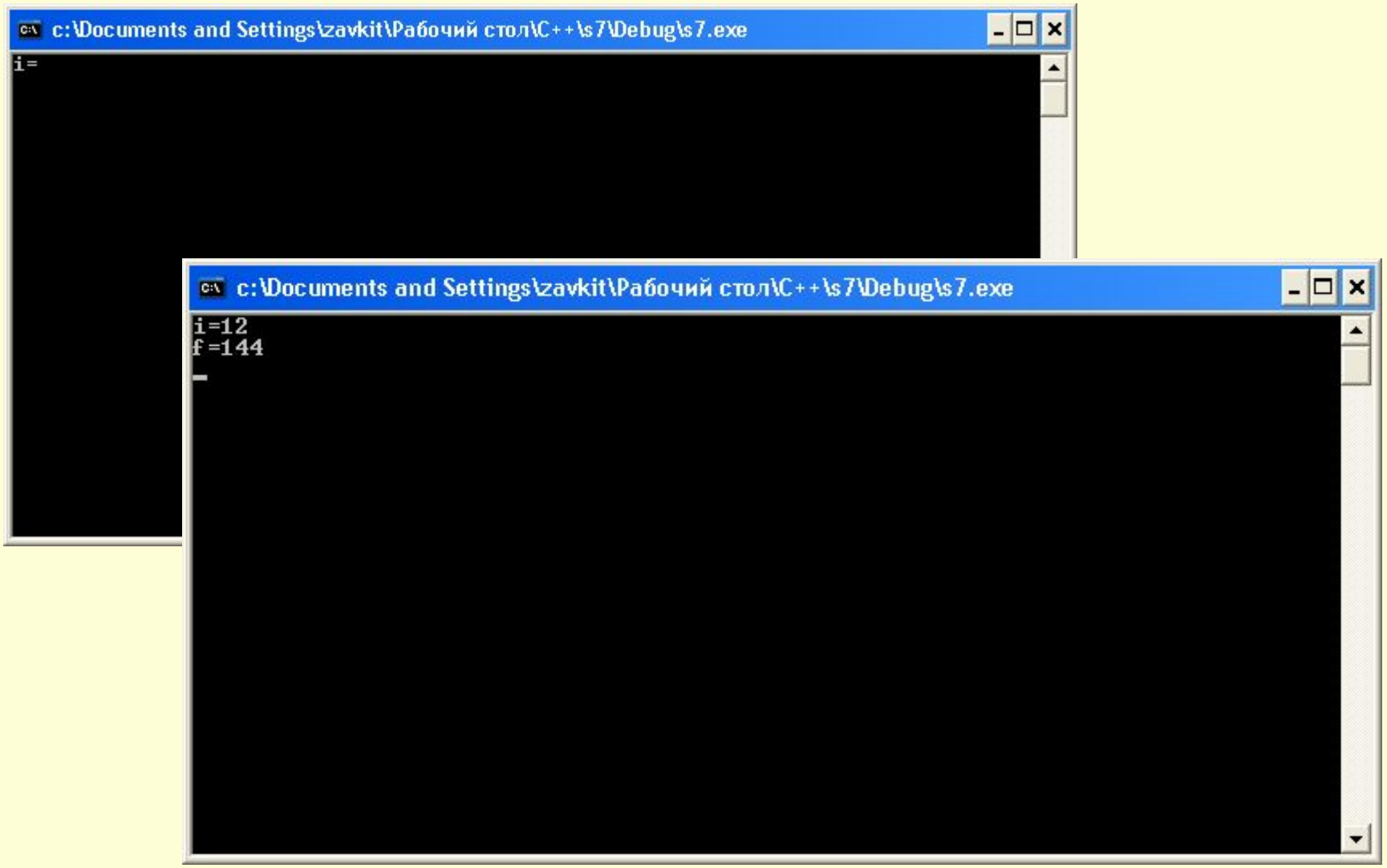


Текст программы

```
#include "stdafx.h"
#include <math.h>
#include <iostream>
#include  <conio.h>

using namespace std;

long int fibonachi(unsigned int
n)
{
if ((n==0)||(n==1)) return(n);
else
return(fibonachi(n-1)+
fibonachi(n-2));
}
```

```
int main()
{
int i;
long int f;
cout<<"i=";
cin>>i;
f=fibonachi(i);
cout<<"f="<<f<<endl;
_getch();
}
```

Результаты


c:\Documents and Settings\zavkit\Рабочий стол\C++\s7\Debug\s7.exe
i=
c:\Documents and Settings\zavkit\Рабочий стол\C++\s7\Debug\s7.exe
i=12
f=144

Область видимости переменных в функциях C/C++, расширение области видимости переменных

Рассмотрим особенности использования локальных и глобальных переменных в программах на C++:

1. Область видимости и использования локальной переменной ограничена функцией, где она определена.
2. Глобальные переменные объявляются вне любых функций и их областью видимостью является весь файл.
3. Одно и тоже имя может использоваться при определении глобальной и локальной переменной. В этом случае в функции, где определена локальная переменная действует локальное описание, вне этой функции «работает» глобальное описание.

Из функции, где действует локальное описание переменной можно обратиться к глобальной переменной с таким же именем, используя оператор расширения области видимости **::переменная**.

Рассмотрим пример:

```
float pr=100.678; int prostoe (int n) { int pr=1,i;
if (n<0) pr=0; else      for (i=2;i<=n/2;i++)
    if (n%i==0){pr=0;break;}
// Вывод локальной переменной
cout<<"local pr="<<pr<<endl;
// Вывод глобальной переменной
cout<<"global pr="<<::pr<<endl;
return pr; }
int main()
{ int g;
cout<<"g=“; cin>>g;
if (prostoe(g)) cout<<"g – prostoe";
else cout<<"g – ne prostoe";
}
```

Результаты работы программы

g=7
local pr=1
global pr=100.678
g - prostoe

Перегрузка и шаблоны функций

Язык C++ позволяет связать с одним и тем же именем функции различные определения, т. е. возможно существование нескольких функций с одним и тем же именем. У этих функций может быть разное количество параметров или разные типы параметров. Создание двух или более функций с одним и тем же именем называется **перегрузкой имени функции**. Перегруженные функции следует создавать, когда одно и то же действие следует выполнить над разными типами входных данных, а иногда одна и та же функция над разными типами входных данных выполняется с помощью разных алгоритмов.

Перегрузка и шаблоны функций

Пример перегрузки функций.

Рассмотрим различные варианты операции возведения в степень:

1) $a^{k/m}$, a - вещественное значение в том числе и 0, k и m – целые числа, k/m – вещественное значение;
2) a^n, a- вещественное значение, n– целое число (положительное, отрицательное, 0);
3) a^n, a – целое число (положительное, отрицательное, 0), n – целое число (положительное, отрицательное, 0)

В ситуациях, когда операция возведения в степень неопределена 0^0 или при возведении отрицательного x в дробную степень $1/m$, когда m четно, будем считать, результат равен 0.

Описание функции для операции 1:

```
float  ctep (float  a, int  k, int  m)
{
    cout << "Функция 1\t";
    if (a == 0)       return (0);
    else
        if (k == 0) return(1);
    else
        if (a>0)
        return(exp((float)k / m*log(a)));
    else
        if (m % 2 != 0)
        return (-(exp((float)k / m*log(-a))));
}
```

Описание функции для операции 2:

```
float  ctep(float  a, int  n)
{     cout <<"Функция 2\t";
      if (a == 0)
      {
            return (0);
      }
      if (n == 0)
      {
             return (1);
           };
           if (n < 0) { return(1 / pow(a, -n)); }
           else
           {    return(pow(a, n));   }
}
```

Описание функции для операции 3:

```
int  ctep(int  a, int  n)
{ cout << "Функция 3\t";
    if (a == 0)
    {
        return (0);
    }
    else    if (n == 0)
    {
        return (1);
    }
    else
        if (n < 0) return(1 / pow(a, -n));
        else
            return(a*pow(a, n - 1));
}
```

```
#include…
#include <iostream>
#include <cmath>
using  namespace  std;
// сигнатуры  перегружаемых  функций
float ctep (float a, int k, int m);
float ctep (float a, int n);
int ctep(int a, int n);
      int main()
      {     setlocale(LC_ALL, "Russian");
            float a;  int k, n, m;
            cout << "a="; cin >> a;      cout << "k="; cin >> k;
            cout << "s=" << ctep(a, k) << endl;
            cout << "s=" << ctep((int)a, k) << endl;
            cout << "a="; cin >> a;      cout << "k="; cin >> k;
            cout << "m="; cin >> m;
            cout << "s=" << ctep(a, k, m) << endl;
            system("Pause");            return 0;
      }
// описания перегружаемых функций
……..
```

D:\Программирование\C++\При...

```
a=2.54
k=4
Функция 2        s=41.6231
Функция 3        s=16
a=2.54
k=3
m=5
Функция 1        s=1.74944
Для продолжения нажмите любую клавишу . . .
```

D:\Программирование\C++\Пр...

```
a=5.2
k=3
Функция 2        s=140.608
Функция 3        s=125
a=-8
k=1
m=3
Функция 1        s=-2
Для продолжения нажмите любую клавишу . . .
```

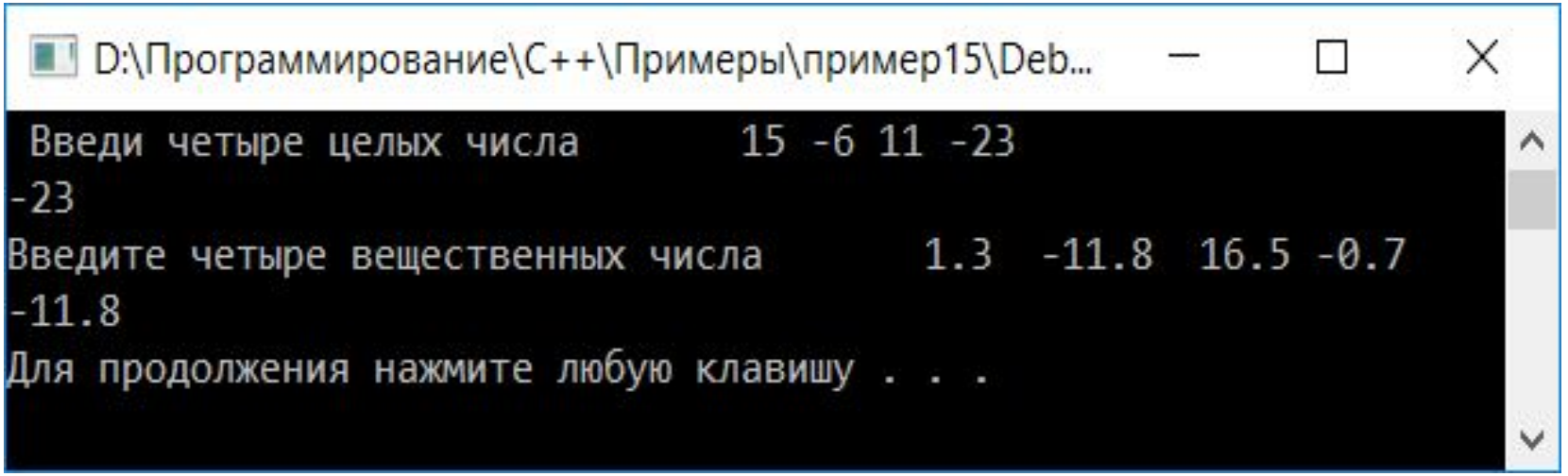
Шаблоны

Перегрузку можно применять при использовании различных алгоритмов решения задачи, при различных типах исходных данных и просто при различных типах исходных данных.
При использовании только *различных типов исходных данных* целесообразно применять **шаблоны**.
Шаблон – это особый вид функций, который начинается со служебного слова ***template***, за которым в угловых скобках (<>) следует список используемых в функции типов данных. Каждый тип предваряется служебным словом ***class***. Можно сказать, что в случае шаблона в качестве параметров выступают не только переменные, но их типы.

Рассмотрим пример шаблона поиска наименьшего из четырех чисел.

```
#include "stdafx.h"
#include <iostream.h>
//Определяем абстрактный тип данных с
// помощью служебного слова Type
template <class Type>
// Определяем функцию с
// использованием
// типа данных Type
Type minimum(Type a, Type b, Type c,
Type d)
{
Type min=a;
if (b<min) min=b;
if (c<min) min=c;
if (d<min) min=d;
return min;
}
int main()
{
int ia,ib,ic,id,mini;
float ra,rb,rc,rd,minr;
cout<<"Vvod 4 thelih chisla\t";
cin>>ia>>ib>>ic>>id ;
//Вызов функции minimum, в которую
// передаем 4 целых значения
mini=minimum(ia,ib,ic,id);
cout<<"\n"<<mini<<"\n";
cout<<"Vvod 4 vecshestvenih chisla\t";
cin>>ra>>rb>>rc>>rd;
//Вызов функции minimum, в которую
// передаем 4 вещественных
// значения
minr=minimum(ra,rb,rc,rd);
cout<<"\n"<<minr<<"\n";

return 0;
}
```

Результаты работы программы:



Использование значений формальных параметров по умолчанию

В C++ существует возможность задать значение некоторых формальных параметров по умолчанию, если такой параметр будет отсутствовать в вызове функции, то будет работать значение по умолчанию. Формальные параметры со значениями по умолчанию должны быть самыми последними в списке.

Использование значений формальных параметров по умолчанию

```
float stepen(float a, int n=3)
{   if (n==0) return(1);
    else
        if (n<0) return(1/stepen(a,-n));
        else
            return(a*stepen(a,n-1)); }
int main()
{    int a;
    long int f;
    cout<<"a=“;    cin>>a;
    f=stepen(a, 5);
    cout<<"f="<<f<<endl;
    f=stepen(a);
    cout<<"f="<<f<<endl;
    return 0; }
```

Обработка массивов

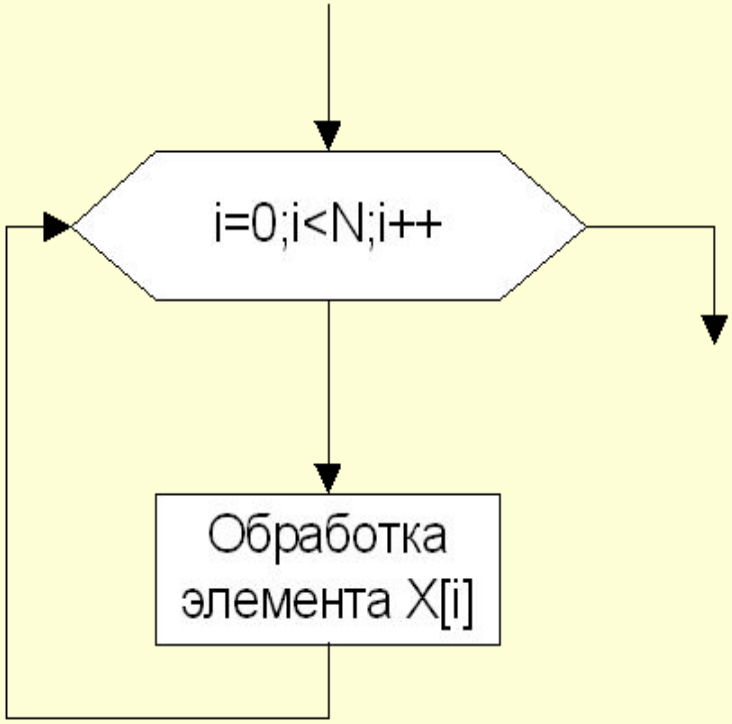
12.1	0.13	-1,5	0	21.9	-3.7	5.0	121.7
0-й элемент массива	1-й элемент массива	2-й элемент массива	3-й элемент массива	4-й элемент массива	5-й элемент массива	6-й элемент массива	7-й элемент массива

Одномерный массив описывают так:
тип имя_переменной [n];
где n – количество элементов в массиве, причем нумерация начинается с нуля: от 0 до n–1.
Например:
int x[10];
float g[25];
Обратиться к элементу одномерного массива можно, указав имя массива и номер элемента в квадратных скобках.

Двумерный массив (*матрицу*) можно объявить так:
тип имя_переменной [n][m];
где n – количество строк (от 0 до n-1), m – количество столбцов (от 0 до m-1).
Например, **double m[3][4];**
Обращаются к элементу матрицы, указывая последовательно в квадратных скобках соответствующие индексы:
Например, a[1][2] – элемент матрицы a, находящийся в первой строке и втором столбце.
Массиву, как и простой переменной, можно присвоить начальные значения в момент его описания.
Например,
float a[5]={1.2,(float)3/4, 5./6,6.1,7.8};

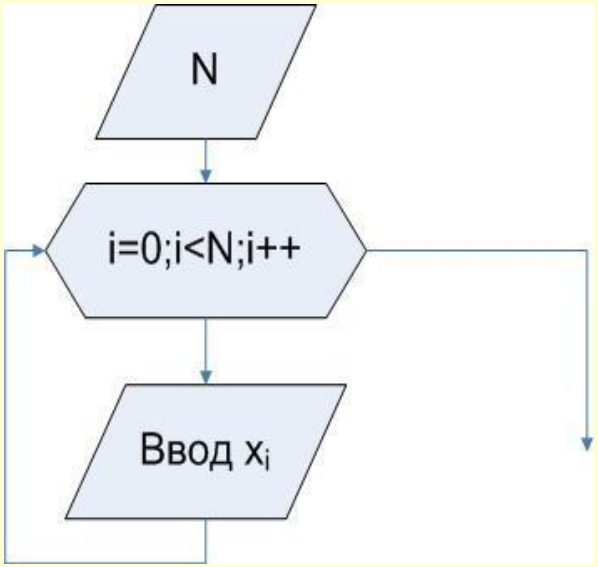
Обработка массива

```
for (i=0; i<N; i++)
обработка X[i];
```



Ввод элементов массива

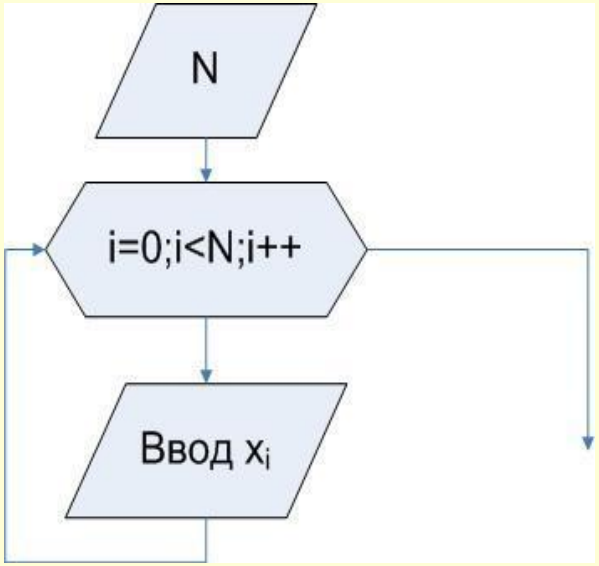
```
float x[10];
int i,n;
cout<< "n= "; cin>>n;
cout<< "Vvedi x ";
for(i=0;i<n;i++)
cin>>x[i];
```



Ввод элементов массива

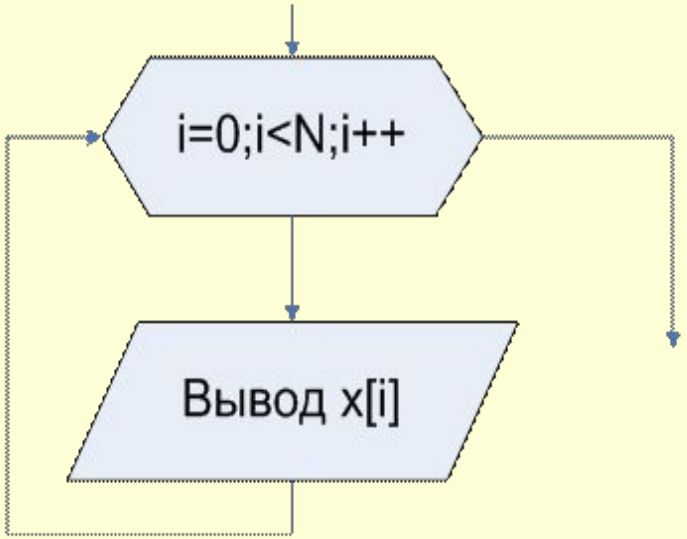
```
float x[10];
int i,n;
printf("\n N=");
scanf_s("%d",&n);
printf("\n Введите массив X \n");
for(i=0;i<n;i++)
scanf_s("%f",&x[i]);
```

```
float  x[10];
inti,n;
cout<<"n= "; cin>>n;
cout<<"Vvedi x ";
for (i=0; i<n; i++)
cin>>x[i];
```



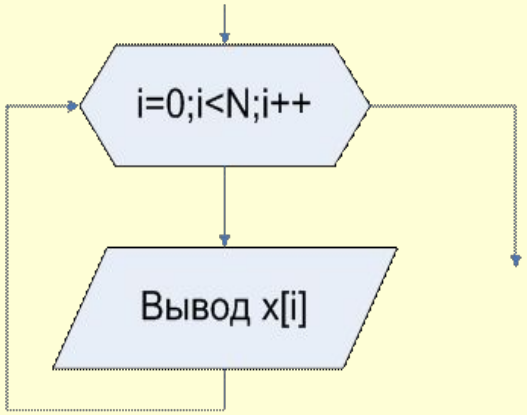
Вывод элементов массива

```
cout<<endl;

for (i=0; i<n; i++)
cout<< x[i] << "    ";

cout<<endl;
```



Вывод элементов массива

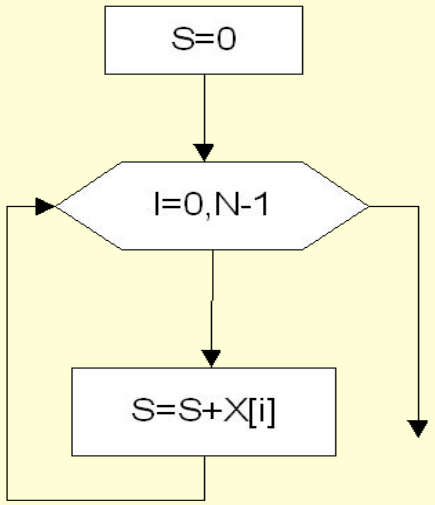
При организации вывода элементов массива можно использовать специальные символы \t \n.



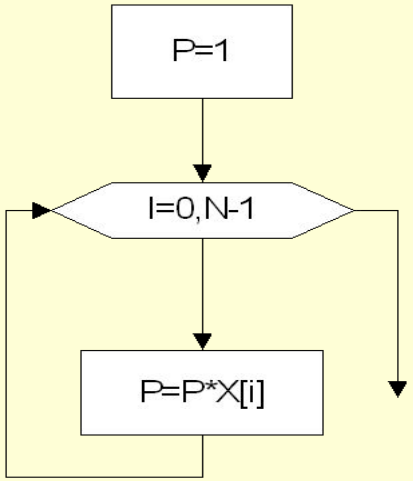

```
printf("\n Массив X\n");
for(i=0; i<n; i++)
printf("%g\t“ ,x[i]);
printf("\n");
```

Алгоритм вычисления суммы и произведения элементов массива

```
for (s=0,i=0; i<n; i++)
s+=X[i];
```




```
for(P=1, i=0; i<n; i++)
P*=X[i];
```



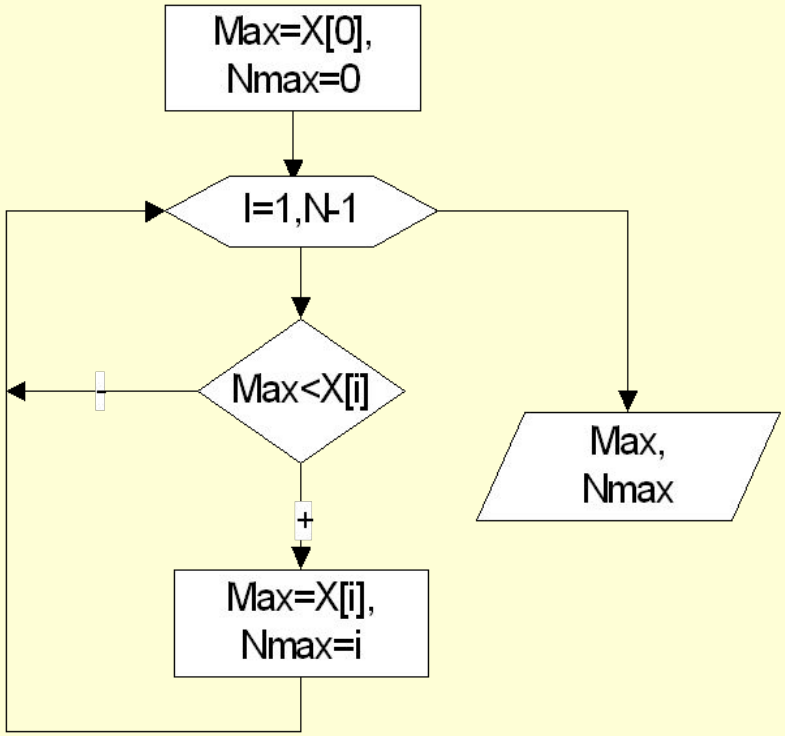
Поиск максимального элемента и его номера

```
for (nmax=0, i=1; i<n; i++)
if (X[i]>X[nmax])
{
nmax=i;
}
```

---

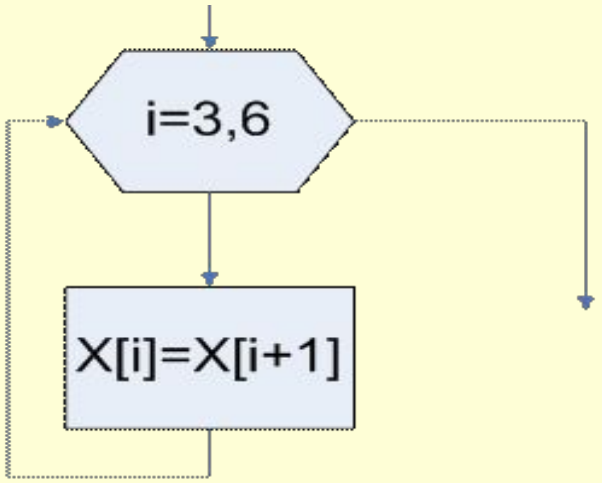
nmax – индекс максимального элемента,

X[nmax] – значение максимального элемента



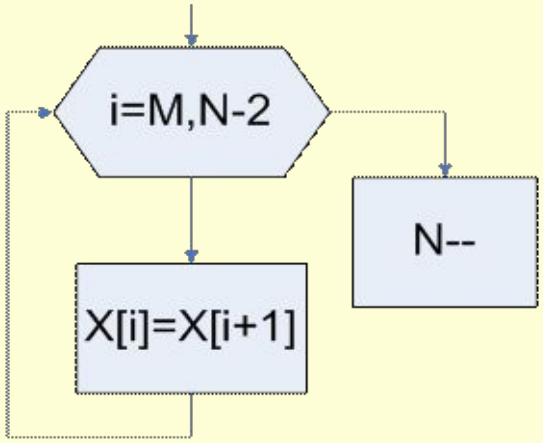
Алгоритм удаления элемента из массива

Пусть необходимо удалить из массива, состоящего из семи элементов, четвертый по номеру элемент.



В общем случае:
Удаляем элемент с номером **M** из массива **X**, в котором **N** элементов.

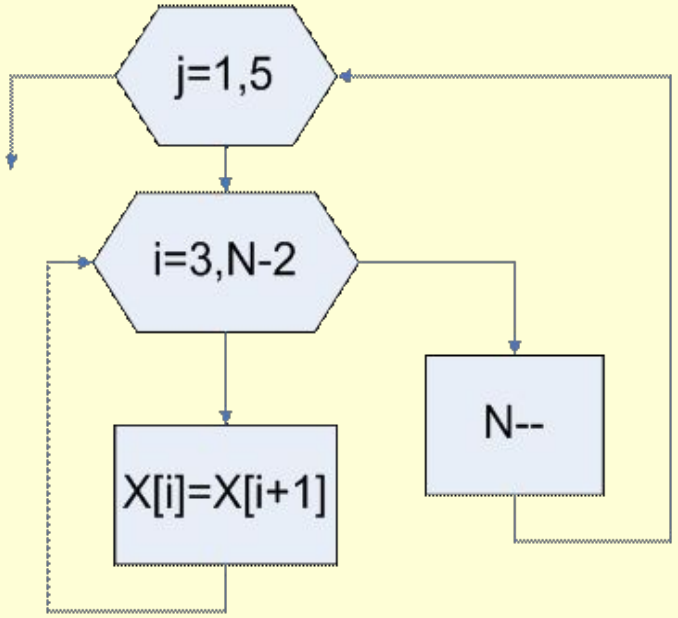
```
for (i=M; i<=N-2; i++)
X[i]=X[i+1]; N--;
```



Алгоритм удаления элемента из массива

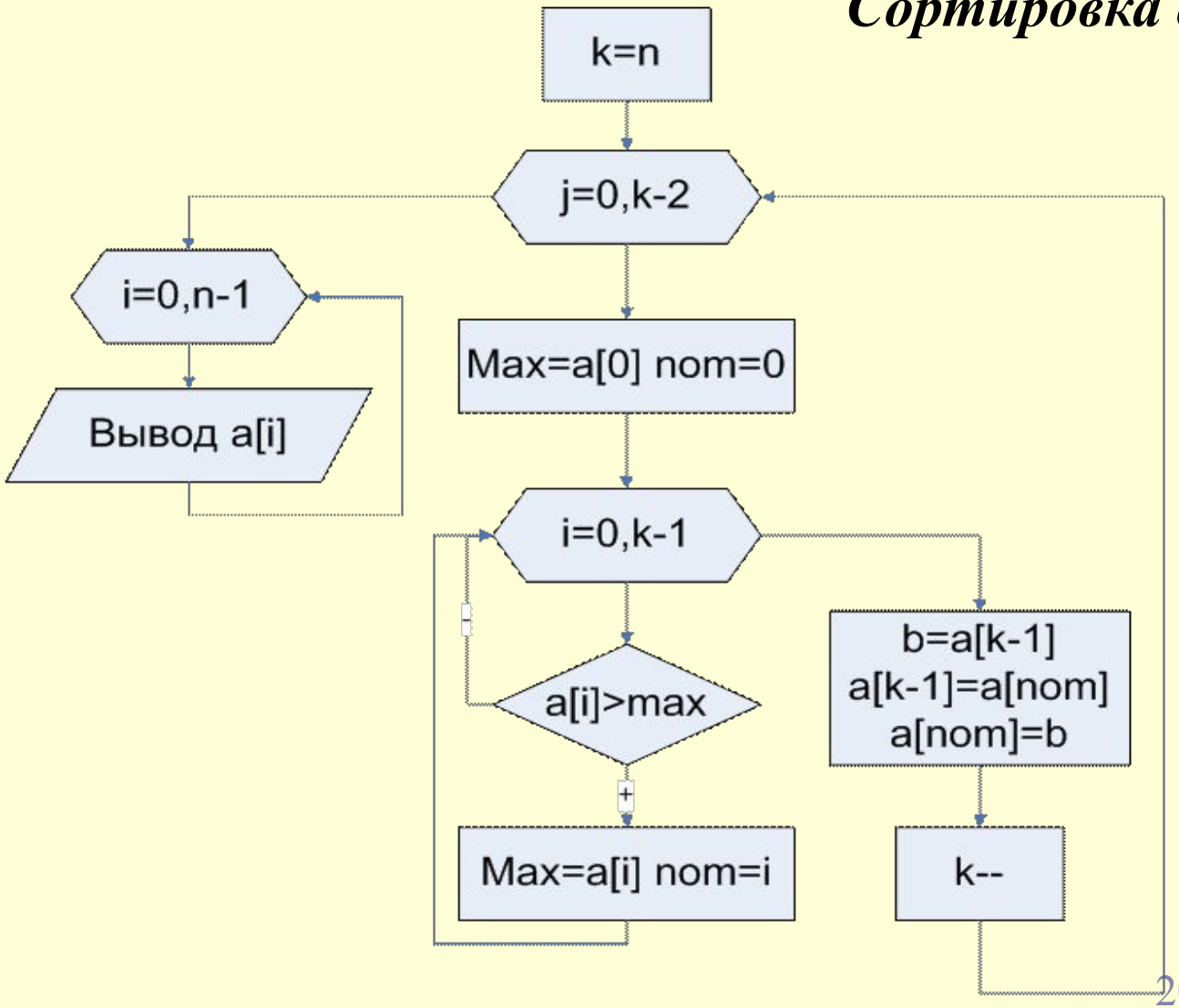
Удалить элементы с 4-го по
8-й в массиве из N элементов

```
for (j=1; j<=5; j++, N--)
for (i=3; i<= N-2; i++)
    X[i]=X[i+1];
```



Упорядочение элементов массива

Сортировка выбором

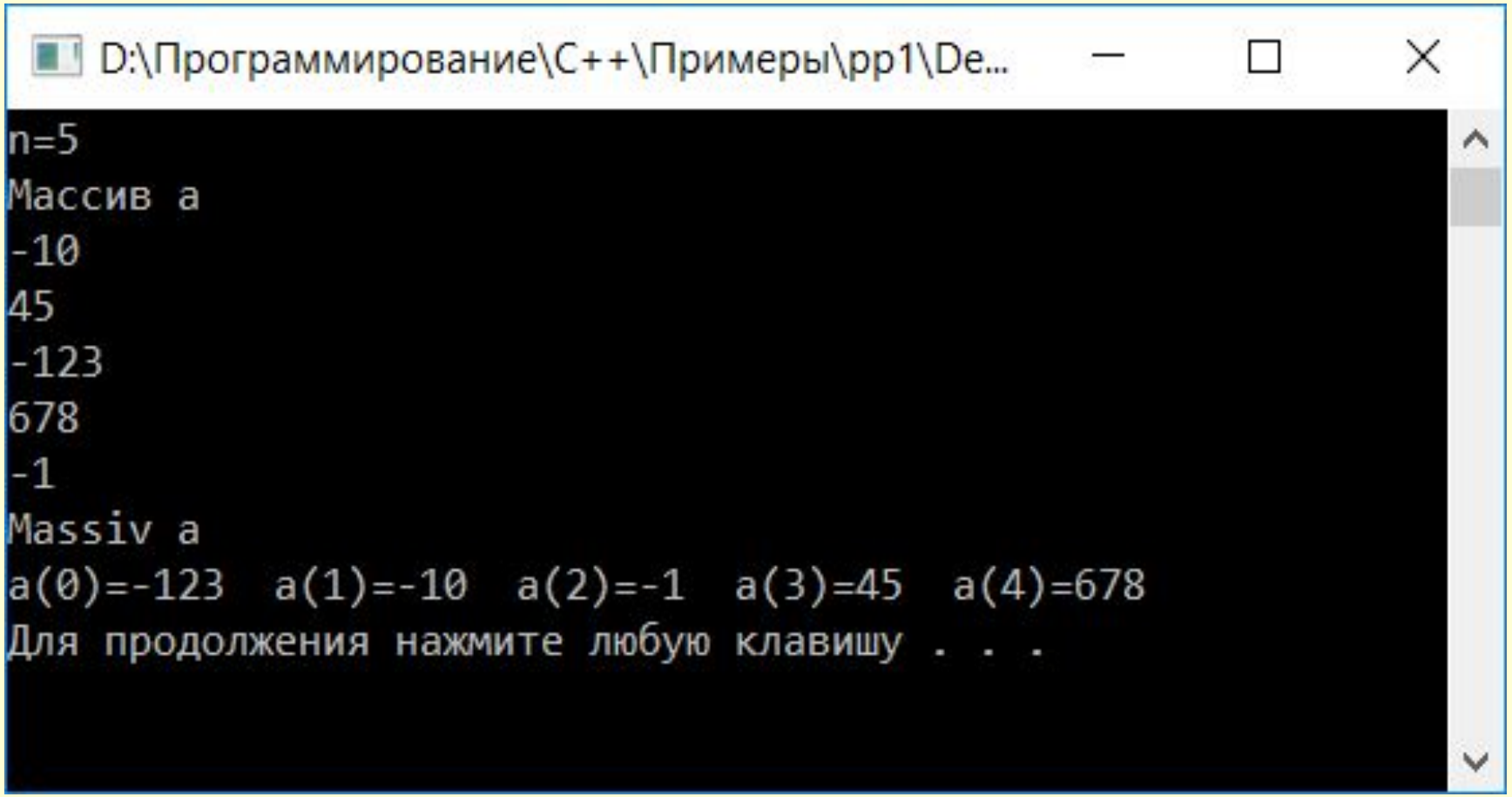


Сортировка выбором

```
#include …
#include <iostream>
using namespace std;
      int main()
{ setlocale(LC_ALL,
"Russian");
      float b, max, a[20];
      int i, j, n, k, nom;
      cout << "n="; cin >> n;
      cout << "Массив a\n";
      for (i = 0; i<n; i++)
      cin >> a[i];  k = n;
for (j = 0; j <= n - 1; j++)
{ max = a[0]; nom = 0;
```

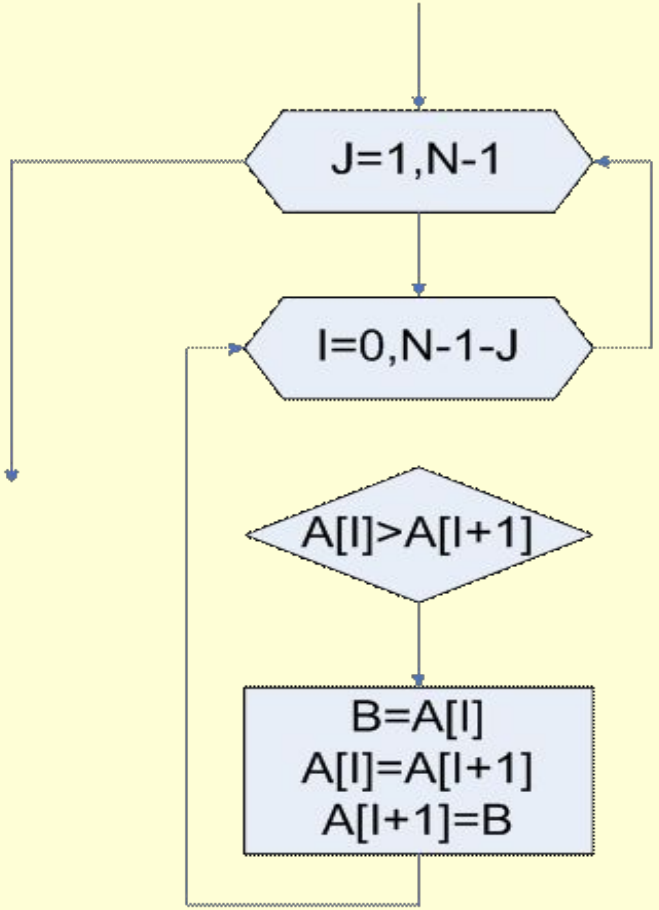
```
for (i = 1; i<k; i++)
if (a[i]>max) { max = a[i];
nom = i; } b = a[k - 1];

a[k - 1] = a[nom];
      a[nom] = b;
      k--;}
      cout << "Massiv a\n";
      for (i = 0; i<n; i++)
      cout << "a(" << i << ")=" <<
a[i] << "  ";
            cout << endl;
            system("Pause");
            return 0;   }
```

Результаты работы программы:



Упорядочение элементов массива

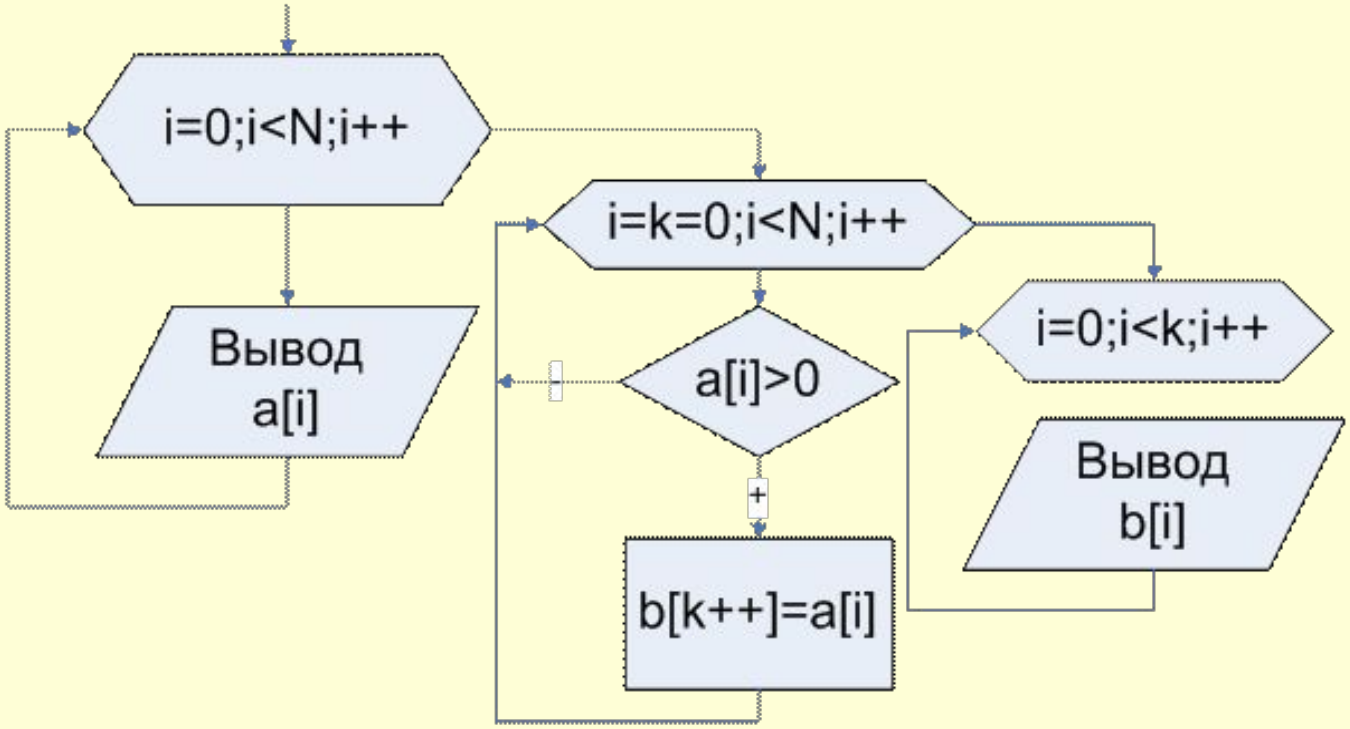
Сортировка методом пузырька



Упорядочение элементов массива

```
int main()
{
   float b,max,a[20];
   int i,j,n,k,nom;
   cout<<"n=";
   cin>>n;
   cout<<"Massiv a\n";
   for(i=0; i<n; i++)
       cin>>a[i];
   for(j=1; j<=n-1; j++)
       for(i=0; i<=n-1-j; i++)
          if (a[i]>a[i+1])
          {
             b=a[i];
             a[i]=a[i+1];
             a[i+1]=b;
          }
   cout<<"Massiv a\n";
   for(i=0; i<n; i++)
cout<<"a("<<i<<")="<<a[i]<<"\t";
   cout<<endl;
   return 0;  }
```

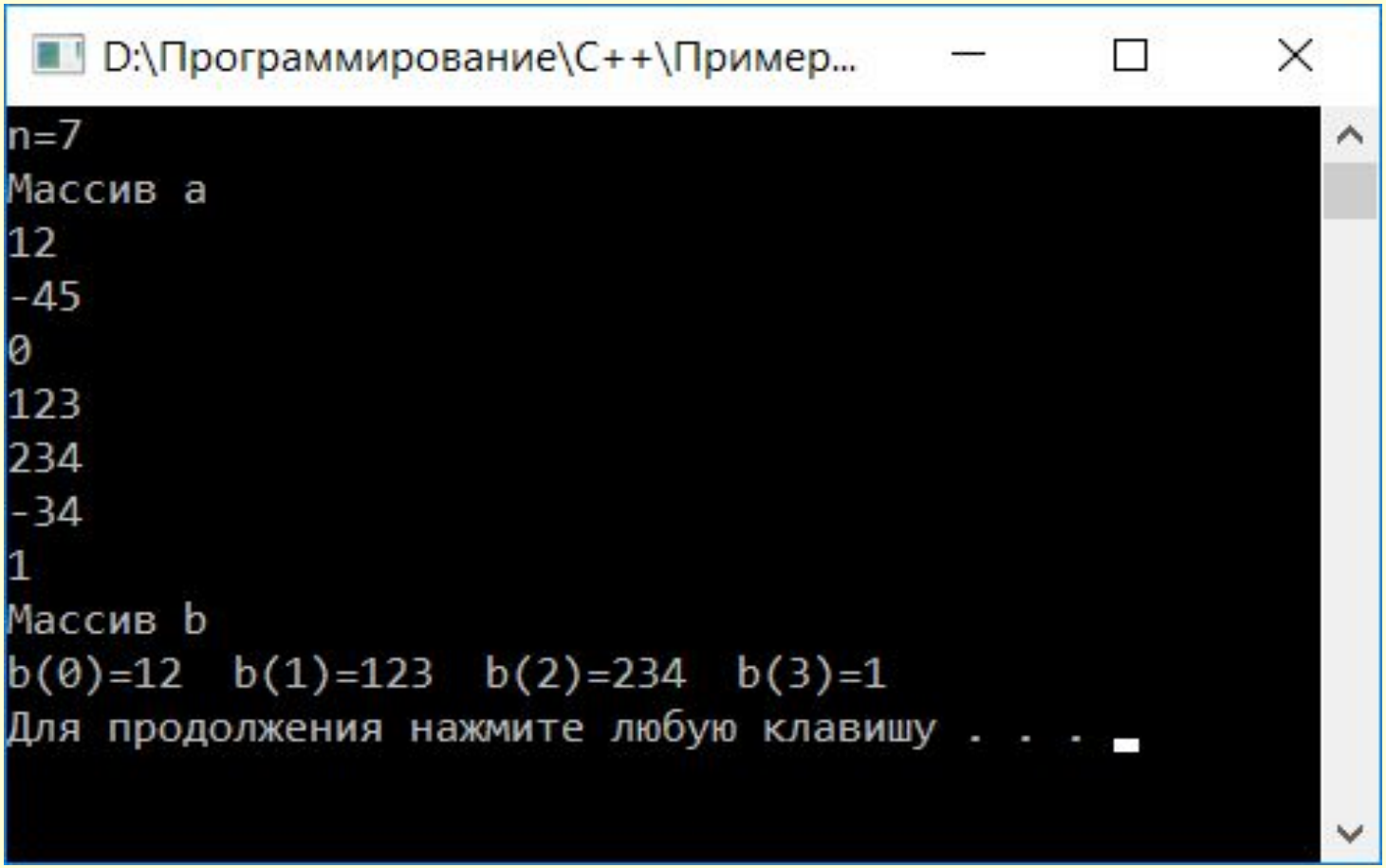
Запись положительных элементов массива A в массив B



Запись положительных элементов массива A в массив B

```
int main()
{   float a[20],b[20];
    int i,n,k;
    cout<<"n=";
    cin>>n;
    cout<<"Massiv a\n";
    for(i=0;i<n;i++)
        cin>>a[i];
    for(k=i=0;i<n;i++)
        if (a[i]>0)  b[k++]=a[i];
    cout<<"Massiv b\n";
    for(i=0;i<k;i++)     cout<<"b("<<i<<")="<<b[i]<<"\t";
    cout<<endl;
    return 0;
}
```

Запись положительных элементов массива A в массив B

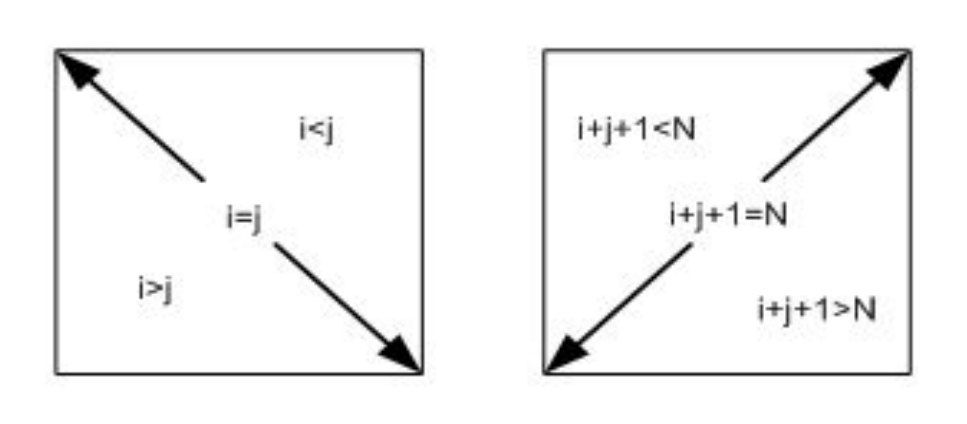


Свойства матриц

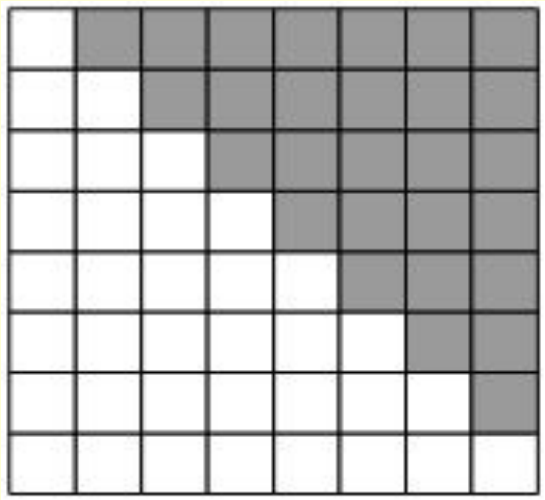
В программах матрицы представляются в виде двумерных массивов.

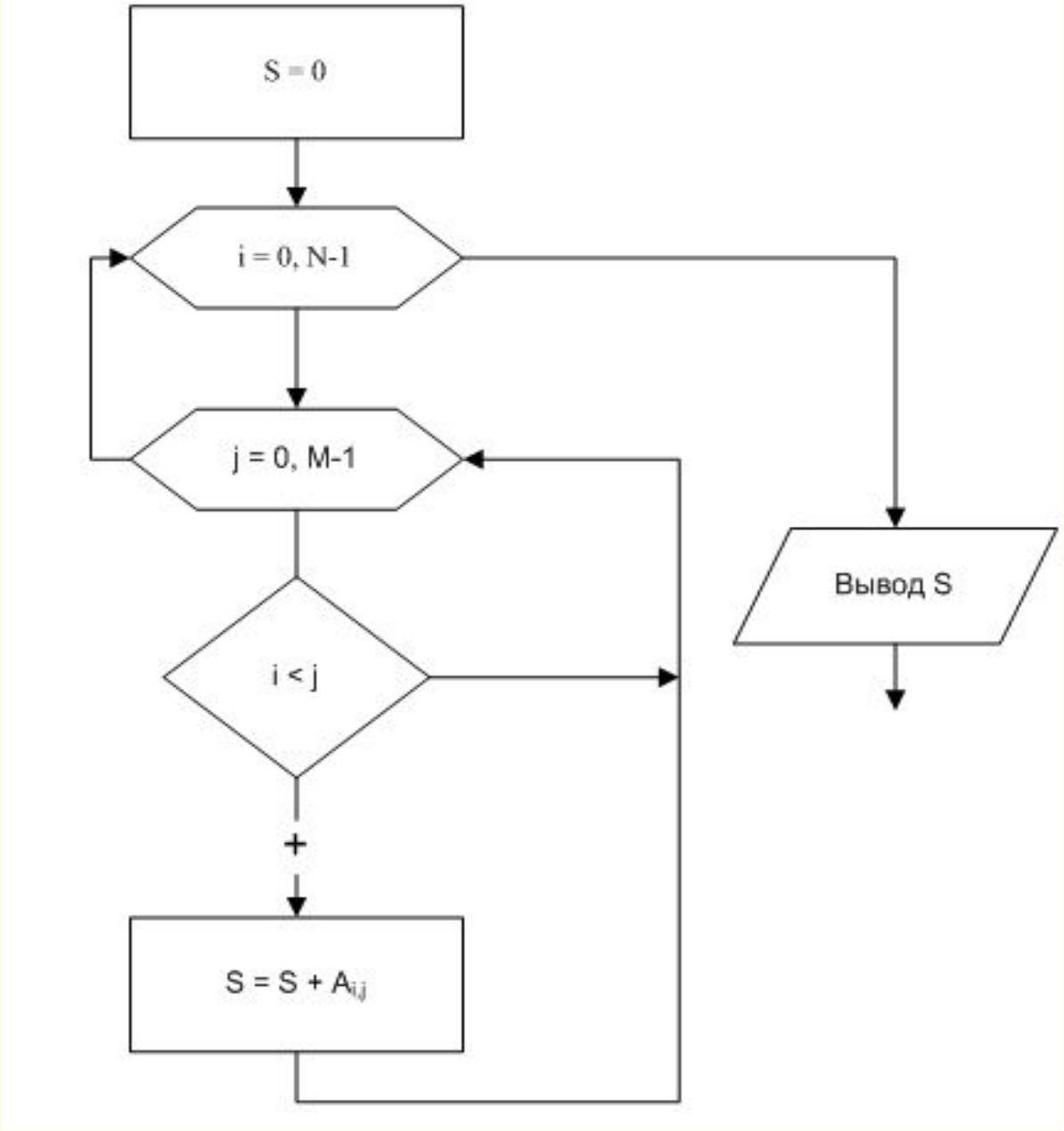
Рассмотрим некоторые свойства матриц:

- если номер строки элемента совпадает с номером столбца ($i = j$), это означает что элемент лежит на главной диагонали матрицы;
- если номер строки превышает номер столбца ($i > j$), то элемент находится ниже главной диагонали;
- если номер столбца больше номера строки ($i < j$), то элемент находится выше главной диагонали.
- элемент лежит на побочной диагонали, если его индексы удовлетворяют равенству $i + j + 1 = n$;
- неравенство $i + j + 1 < n$ характерно для элемента находящегося выше побочной диагонали;
- соответственно, элементу лежащему ниже побочной диагонали соответствует выражение $i + j + 1 > n$.



Найти сумму элементов матрицы, лежащих выше главной диагонали

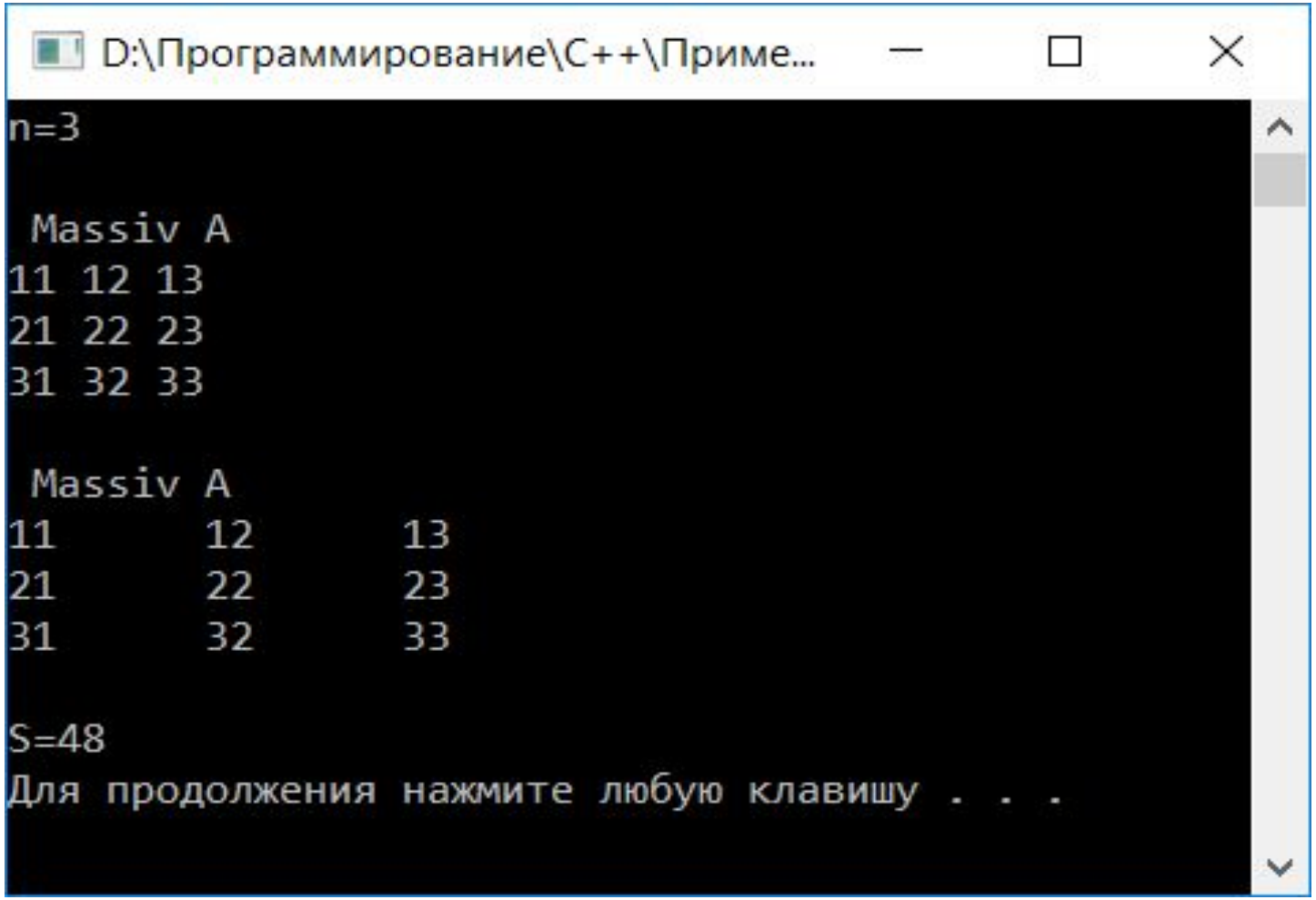



S = 0
i = 0, N-1
j = 0, M-1
i < j
+
S = S + A_{i,j}
Вывод S

Текст программы

```
#include….
#include <stdio.h>
using namespace std;
int main()
{  setlocale(LC_ALL, "");
float b, a[20][20], s;
// описание массива
int i, j, n; printf("n=");
scanf_s("%d", &n);
printf("\n Massiv A \n");
// ввод значений элементов массива
for (i = 0; i<n; i++)
   for (j = 0; j< n; j++)
{ scanf_s("%g", &b); a[i][j] =
b; printf("\n Massiv A\n");
// вывод значений элементов массива
for (i = 0; i<n; i++)
{   for (j = 0; j< n; j++)
printf ("%g\t", a[i][j]);
printf("\n"); }
// расчет суммы элементов массива,
// расположенных выше диагонали
for (s = 0, i = 0; i<n; i++)
for (j = i+1; j<n; j++)
 s += a[i][j];
printf("\nS=%g\n", s);
system("Pause");
return 0;  }
```

Результаты работы программы:



Использование генератора случайных чисел

Генератор случайных чисел - это объект, формирующий последовательность из псевдослучайных чисел.

Работа с генератором случайных чисел включает:

- описание объекта генератора случайных чисел;
- инициализацию случайным стартовым числом;
- формирование выходного диапазона случайных чисел;
- получение очередного случайного числа.

Написать программу заполнения двумерного массива случайными числами из диапазона от -100 до 100.

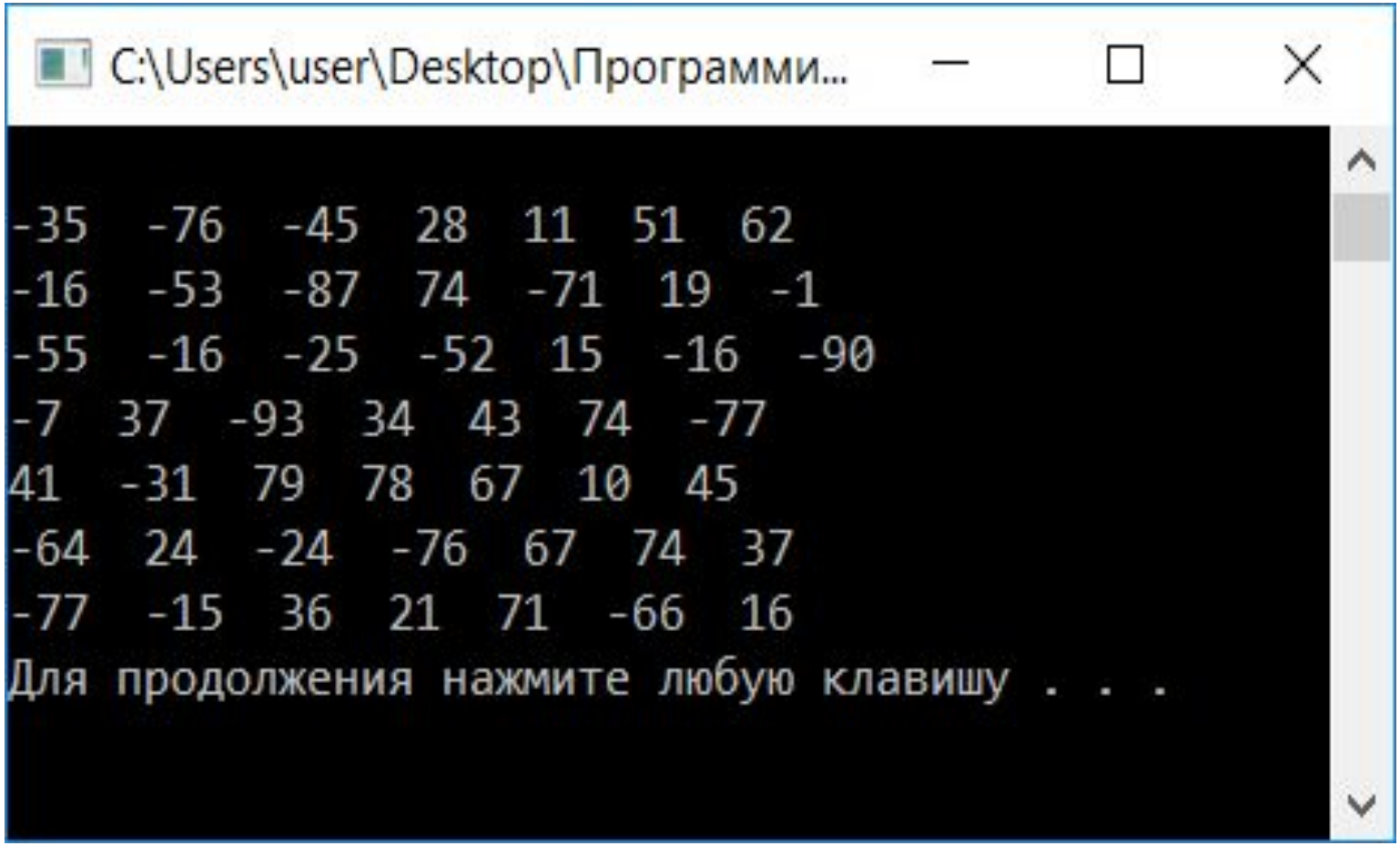
В программе используем генератор случайных чисел на основе алгоритма «*Вихрь Мерсенна*».

Использование генератора случайных чисел

```
#include"…"
#include<iostream>
#include<random>
using namespace std;
int  main()
{ // описание объекта – генератора
  // случайных чисел
random_devicerd;
// инициализация случайным
// стартовым числом
mt19937 gen(rd());
//задание выходного диапазона от
//  -100 до 100
uniform_int_distribution <>
dist(-100, 100);
      int mass[10][10];
      int n = 7;
```

```
for (int i = 0; i < n; i++)
{     cout << endl;
    for (int j = 0; j < n; j++)
      {
//генерация очередного случайного
// числа
       mass[i][j] = dist(gen);
      cout << mass[i][j] <<"  ";
      }     }
      cout << endl;
      system("Pause");
      return 0;
          }
```

Результаты работы программы:



Строки

Строка – последовательность символов. Если в выражении встречается одиночный символ, он должен быть заключен в **одинарные кавычки**. При использовании в выражениях строка заключается в **двойные кавычки**. Признаком конца строки является нулевой символ '\0'. В C\C++ в отличии от других языков программирования отсутствует тип данных строка, строки в Си можно описать с помощью массива символов (массив элементов типа char), в массиве следует предусмотреть место для хранения признака конца строки ('\0').

Обработка строк

Описание строки:

char s[20];

диалог: на экран выводится подсказка *" s= "* :

cout<< " s="<<endl;

ввод строки до пробела:

cin>>s;

char in_string[80]; // строковый массив для ввода

Ввод строки с пробелами до тех пор, пока не будет нажата клавиша <*Enter*>.

cin.getline(in_string, 80);

Пример работы со строками

```
#include <iostream>
#include <sstream>
#include <string>
using namespace std;
int main()
{    // объявление и инициализация строки
char string[] = "this is string - ";
cout << "Enter the string: ";
char in_string[80]; // строковый массив для ввода
cin.getline(in_string, 80);
// функция считывает все введённые символы с пробелами до тех пор, пока не
// будет нажата клавиша Enter
cout << string << in_string << endl; // вывод строкового значения
system("pause");
return 0;
}
```

Строки

В современном стандарте C++ определен класс с функциями и свойствами (переменными) для организации работы со строками (в классическом языке C строк как таковых нет, есть лишь массивы символов char):

#include <string>

Для работы со строками также нужно подключить using namespace std; В противном случае придётся везде указывать описатель класса std::string вместо string.

Строки

Основные возможности, которыми обладает класс string:

- ❖копирование одной строки в другую. Для встроенного типа приходится использовать функцию strcpy();
- ❖доступ к отдельным символам строки для чтения и записи. Во встроенном массиве для этого применяется операция взятия индекса или косвенная адресация с помощью указателя;
- ❖сравнение двух строк на равенство. Для встроенного типа используются функции семейства strcmp();
- ❖конкатенация (сцепление) двух строк, дающая результат либо как третью строку, либо вместо одной из исходных. Для встроенного типа применяется функция strcat(), однако чтобы получить результат в новой строке, необходимо последовательно задействовать функции strcpy() и strcat(), а также позаботиться о выделении памяти;
- ❖встроенные средства определения длины строки (функции-члены класса size() и length()). Узнать длину строки встроенного типа можно только вычислением с помощью функции strlen();
- ❖возможность узнать, пуста ли строка.

Строки

Инициализация строк при описании и **длина строки** (не включая завершающий нуль-терминатор):

```
string st ( "Моя строка\n" );
 cout << "Длина " << st << ": " << st.size() << " символов,
включая символ новой строки\n";
```

Строка может быть задана **пустой**:

```
string st2;
```

Для проверки того, **пуста ли строка**, можно сравнить ее длину с 0:

```
if ( ! st.size() ) // пустая или применить метод empty(),
возвращающий true для пустой строки и false для непустой:
if ( st.empty() ) // пустая
```

Третья форма создания строки **инициализирует объект типа string другим объектом того же типа**:

```
string st3( st ); Строка st3 инициализируется строкой st.
```

Строки

Для того чтобы убедиться, что **строки совпадают** воспользуемся оператором сравнения (==):

```
if ( st == st3 ) //
```

Чтобы **скопировать одну строку в другую** достаточно обычной операции присваивания:

```
st2 = st3; // копируем st3 в st2
```

Для **сцепления строк** используется операция сложения (+) или операция сложения с присваиванием (+=).

Пусть даны две строки:

```
string s1( "hello, " ); string s2( "world\n" );
```

Можно получить третью строку, состоящую из конкатенации первых двух, таким образом:

```
string s3 = s1 + s2;
```

Если необходимо добавить s2 в конец s1, пишем

```
s1 += s2;
```

Строки

Операция сложения может сцеплять объекты класса **string** не только между собой, но и со строками встроенного типа. Можно переписать пример, приведенный выше, так, чтобы специальные символы и знаки препинания представлялись встроенным типом char *, а значимые слова – объектами класса string:

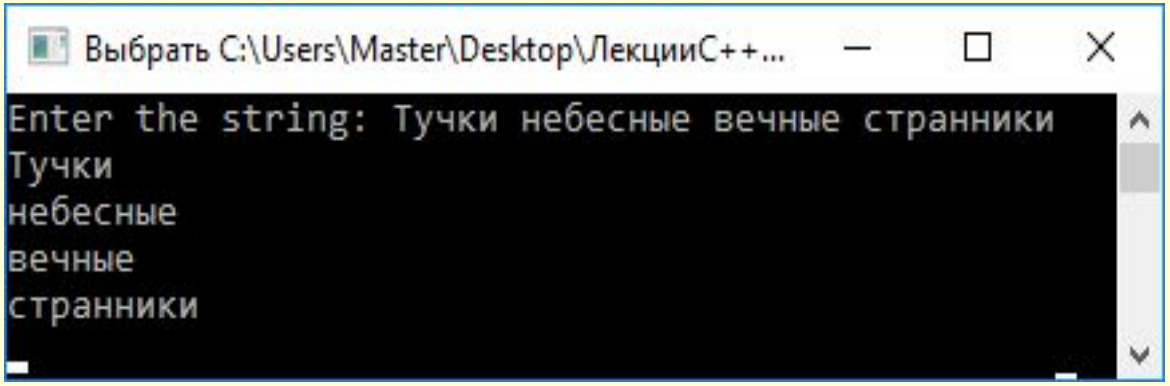
```
const char *pc = ", "; string s1( "hello" ); string s2( "world" );
string s3 = s1 + pc + s2 + "\n"; cout << endl << s3;
```

Подобные выражения работают потому, что компилятор "знает", как автоматически преобразовывать объекты встроенного типа в объекты класса string. Возможно и простое присваивание встроенной строки объекту string:

```
string s1; const char *pc = "a character array"; s1 = pc;
```

Выделение слов из строки string, введённой с клавиатуры

```
#include….
#include<iostream>
#include <string>
#include <vector>
#include <sstream>
 using namespace std;
int main()
{
setlocale(LC_ALL, "Russian");
cout << "Enter the string: ";
stringstr; getline(cin, str); // Ввод строки с клавиатуры
vector <string> vecstr;    string word; stringstream s(str);
while (s >> word) vecstr.push_back(word);
int vsize = vecstr.size();
for (int i = 0; i < vsize; i++)
cout <<  vecstr[i] << endl;    cin.get();
system("Pause");  return 0;
}
```



Структуры

Структура является совокупностью одного или более объектов(переменных, массивов, указателей, других объектов), которые для удобства работы с ними объединены под одним именем.

Определение структуры состоит из двух шагов:

- объявление структуры (задание нового типа данных определенного пользователем),
- объявление полей.

Структуры

```
struct student
    {    char fio[30];// определено поле fio
     char group[8];// определено поле group
     int year;
     int informatika, math, fizika, history;
    }
```

Определение переменных типа структура;

student Vasya, ES[50];

Для обращения к полям структуры надо указать имя переменной и через точку имя поля

Structura.Pole

Например,

Vasya.Year

ES[4].math

Задано **n** комплексных чисел, найти число наибольшего модуля.

```
#include "…"
#include <iostream>
#include <cmath>
using namespace std;
int main()
{struct complex
{ float x; float y;};
complex p[100];
int i,n,nmax;
float max;
cout<<"n=“;  cin>>n;
for(i=0;i<n;i++)
{ cout<< "Vvedite complex chislo\n";
cin>>p[i].x;     cin>>p[i].y;
cout<<p[i].x<<"+"<<p[i].y<<"i"<<endl;
}
max=pow(p[0].x*p[0].x+p[0].y*p[0].y,0
.5); nmax=0;
for(i=1;i<n;i++)
if pow(p[i].x*p[i].x+p[i].y*p[i].y,0.5)
>max) {
max=pow(p[i].x*p[i].x+p[i].y*p[i].y,
0.5); nmax=i;  }
cout<<"Nomer max modulya " <<nmax
<<
"\nEgo velichina "<<max<<endl;
return 0;  }
```

Файлы

Файлом называют способ хранения информации на физическом устройстве. Файл — это понятие, которое применимо ко всему — от файла на диске до терминала. В C++ отсутствуют операторы для работы с файлами. Все необходимые действия выполняются с помощью функций, включенных в стандартную библиотеку. Они позволяют работать с различными устройствами, такими, как диски, принтер, коммуникационные каналы и т.д. Эти устройства сильно отличаются друг от друга. Однако файловая система преобразует их в единое абстрактное логическое устройство, называемое **потоком**.

Файлы. Организация работы с файлами

Файл (file) – именованная совокупность данных, находящаяся на внешнем устройстве и имеющая определенные атрибуты (характеристики).

Поток (srteam) – абстрактный канал связи, создаваемый в программе для обмена данными.

Файл, рассматриваемый как последовательность строк символов, разделенных непробельными символами, называется *текстовым*. Его можно создавать и редактировать с помощью любого текстового редактора.

Текстовый поток — это последовательность символов. При передаче символов из потока на экран, часть из них не выводится (например, символ возврата каретки, перевода строки).

Двоичный поток — это последовательность байтов, которые однозначно соответствуют тому, что находится на внешнем устройстве.

Файловый ввод-вывод с использованием потоков средствами C++

Библиотека потокового ввода-вывода

fstream

Связь файла с потоком вывода

ofstream имя логического файла;

Связь файла с потоком ввода

ifstream имя логического файла;

Открытие файла

имя логического файла.open(имя физического файла);

Закрытие файла

имя логического файла.close();

*Заполнить файл значениями функции y = x * cos x*

```
#include …
#include <fstream>
#include <iostream>
using namespace std;
double fun(double x);
int main() {
setlocale(LC_ALL, "Russian");
double a, b, h, x; char s[20];
cout << " a,b,h: ";
cin >> a >> b >> h;
cout << "File name? "; cin >> s;
ofstream f;
f.open(s);
for (x = a; x <= b; x += h)
{  f.width(10);  f << x;
   f.width(15);  f << fun(x) << endl;  }
```