ОСНОВЫ C++ В КУРСЕ ИНФОРМАТИКИ

ФОКИН РОМАН РОМАНОВИЧ
АБИССОВА МАРИНА АЛЕКСЕЕВНА

Учебное пособие

WWW.SCIPRO.RU

Фокин Роман Романович
Абиссова Марина Алексеевна

ОСНОВЫ C++ В КУРСЕ ИНФОРМАТИКИ

Учебное пособие

Санкт-Петербург
2021

УДК 519.85
ББК 22.1
О75

Рецензент:

Копыльцов Александр Васильевич – доктор технических наук, проф., зав. кафедрой физики, ФГАОУ ВО "Санкт-Петербургский государственный университет аэрокосмического приборостроения"

Авторы

Фокин Роман Романович – доктор педагогических наук, доцент, профессор кафедры математики, ФГБВОУ ВО «Военно-космическая академия имени А.Ф. Можайского»

Абиссова Марина Алексеевна – кандидат педагогических наук, доцент кафедры вычислительных систем и информатики, ФГБОУ ВО «ГУМРФ имени адмирала С.О. Макарова»

Основы С++ в курсе информатики [Электронный ресурс]: учебное пособие – Эл. изд. - Электрон. текстовые дан. (1 файл pdf: 74 с.). - Фокин Р.Р., Абиссова М.А. 2021. – Режим доступа: http://scipro.ru/conf/computerscience.pdf. Сист. требования: Adobe Reader; экран 10'.

ISBN 978-1-312-08735-4

Рекомендовано к печати редакционно-издательской комиссией НОО "Профессиональная наука". Протокол № 07821 от 20 августа 2021 г.

Соответствует ФГОС ВО по направлениям подготовки бакалавров: 13.03.02 - Электроэнергетика и электротехника; 20.03.02 - Природообустройство и водопользование; 23.03.01 - Технология транспортных процессов; 23.03.03- Эксплуатация транспортно-технологических машин и комплексов.

Содержание

Введение

Пособие предназначено для студентов, изучающих в курсе информатики основы языка программирования Visual C++ для работы в среде Microsoft Visual Studio Professional 2013 в режимах пустого проекта и консольного приложения. Система программирования - это программное средство, в которую встроен какой-нибудь язык программирования. Visual Studio - это мультиязыковая система программирования. Кроме Visual C++ туда также встроены: Visual Basic (предназначен для прикладных программистов); Visual C# (как и Visual C++, предназначен для системных и прикладных программистов - это самый современный диалект языка C); Visual Fox Pro (система управления базами данных); Visual F# (современный язык искусственного интеллекта) и др. языки. Системные программисты разрабатывают программы для обслуживания вычислительных систем (компьютеров, сетевых систем, внешних устройств) - это операционные системы, драйверы, программы дефрагментации дисков, антивирусные программы и т.п. Прикладные программисты разрабатывают программы для обслуживания пользователей компьютеров: программы общего назначения (Word, Excel), специальные программы для научных расчетов (MatCad), для бухгалтерии (1С:Бухгалтерия) и т.п. В 70-х годах XX века Деннис Ритчи придумал язык C. В 80-х годах XX века Бьерн Страуструп добавил в C объектно-ориентированные технологии программирования, получился C++. В 90-х годах XX века Билл Гейтс добавил в C++ визуально-ориентированные технологии программирования, получился Visual C++.

Используемые сокращения: т. - тема; Л.Р. - лабораторная работа; ч. - часть; рис. - рисунок; табл. - таблица; бл.-сх. - блок-схема.

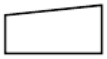
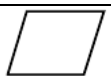
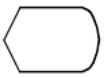
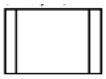
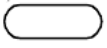
Т.1. Первое задание на С++

Пусть a, b, c - действительные переменные (ячейки памяти, предназначенные для хранения действительных чисел). Положить значение a равным 1.1, значение b ввести с клавиатуры во время работы программы. Вычислить c равным ab + sin b - 2.2

Во-первых, студент должен в специальной тетради для лабораторных работ составить блок-схему алгоритма и показать ее преподавателю. За это он получает первую подпись преподавателя. Конспект лекций ведется в другой тетради.

Таблица 1

Основные символы блок-схем по ГОСТ 19.701-90

Символ	Описание
	**Ручной ввод - символ отображает данные, вводимые вручную во время обработки с устройств любого типа (клавиатура, переключатели, кнопки, световое перо, полоски со штриховым кодом).**
	**Данные - символ отображает данные, носитель данных не определен.**
	**Линии потока - показывают последовательность действий при выполнении алгоритма.**
	**Дисплей - символ отображает данные, представленные в человекочитаемой форме на носителе в виде отображающего устройства (экран для визуального наблюдения, индикаторы ввода информации).**
	**Процесс - символ отображает функцию обработки данных любого вида (выполнение определенной операции или группы операций, приводящее к изменению значения, формы или размещения информации или к определению, по которому из нескольких направлений потока следует двигаться).**
	**Предопределенный процесс - символ отображает предопределенный процесс, состоящий из одной или нескольких операций или шагов программы, которые определены в другом месте (в подпрограмме, модуле).**
	**Решение - символ отображает решение или функцию переключательного типа, имеющую один вход и ряд альтернативных выходов, один и только один из которых может быть активизирован после вычисления условий, определенных внутри этого символа. Соответствующие результаты вычисления могут быть записаны по соседству с линиями, отображающими эти пути.**
	**Соединитель - символ отображает выход в часть схемы и вход из другой части этой схемы и используется для обрыва линии и продолжения ее в другом месте. Соответствующие символы-соединители должны содержать одно и то же уникальное обозначение.**
	**Терминатор - символ отображает выход во внешнюю среду и вход из внешней среды (начало или конец схемы программы, внешнее использование и источник или пункт назначения данных).**

Символ	Описание
	**Комментарий - символ используют для добавления описательных комментариев или пояснительных записей в целях объяснения или примечаний. Пунктирные линии в символе комментария связаны с соответствующим символом или могут обводить группу символов. Текст комментариев или примечаний должен быть помещен около ограничивающей фигуры.**

Символы, необходимые для составления блок-схем содержит табл. 1. Блок-схему для первого задания представляет рис. 1.

Во-вторых, студент должен в тетради для лабораторных работ составить текст программы и показать ее преподавателю. За это он получает вторую подпись преподавателя. Для данного задания текст программы такой:

```
#include <iostream>      // разрешение ввода и вывода
#include <locale> // разрешение национальных алфавитов
#include <cmath>         // разрешение математических функций
#include <string> // разрешение  работы со строками
using namespace std;     // разрешение стандартного пространства имен
/*      Ниже приведен составной оператор - описание главной функции
main,
        состоящий из заголовка функции и тела функции */
void main() // заголовок функции main() без аргументов
{ // операторы между { и } - это составной оператор-блок, это тело main
        setlocale(LC_ALL, "rus"); // разрешение русского алфавита при
выводе
        double a=1.1, b, c; /* создание действительных переменных a, b, c
        и инициализация a=1.1 */
        cout << "Задание 1" << endl << "b = "; /* вывод на экран строк
        "Задание 1",       endl, "b=" */
        cin >> b; // ввод с клавиатуры действительного числа в переменную b
        c = a*b + sin(b) - 2.2; // присвоение переменной c выражения по
заданию
        cout << "c= " << c << endl; // вывод на экран строки "c=", значения c,
строки endl
        system("pause"); // пауза перед завершением программы
}
                        // по правилам C++ это комментарий в конце строки
/*                      по правилам C++
                  это многострочный комментарий                          */
```

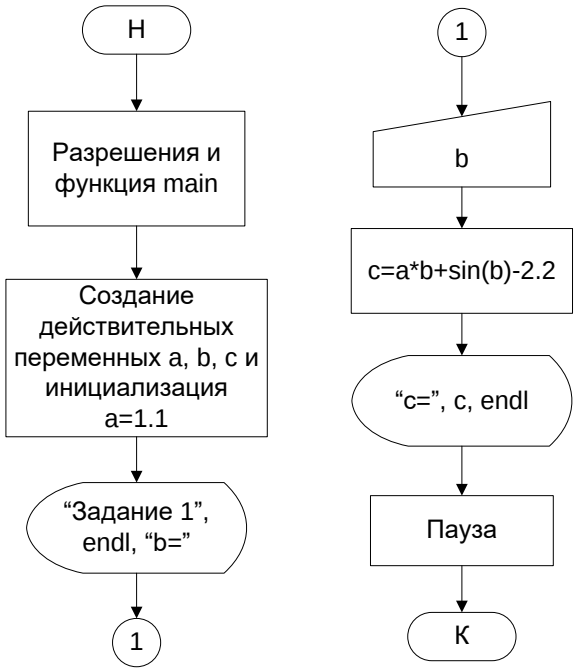


Рисунок 1. Блок-схема для первого задания

Язык программирования - это набор правил для написания программ. В С++ НЕЛЬЗЯ малые буквы заменять БОЛЬШИМИ и наоборот!!! Это одно из главных правил. Математическую функцию sin нужно писать малыми буквами. Написание Sin или SIN будет ошибочным. Аргументы функций всегда заключают в круглые скобки. Нужно писать sin(b), нельзя писать sin b. Функция main или главная в данном случае аргументов не имеет, поэтому мы пишем main() - пустые скобки. Функция sin возвращает действительное число, а функция main в данном случае не возвращает ничего, поэтому мы пишем void или отсутствие. Русский язык - это тоже набор правил для написания сообщений. Сообщение состоит из предложений. Предложение обычно завершается точкой. Программа на С++ состоит из дирректив, операторов и комментариев. Оператор обычно завершается (;) точкой с запятой. Комментарии машина не читает. Их пишет программист для себя. Все диррективы в нашей программе начинаются со знака (#) шарп или диез. Операторы, как и предложения русского языка, бывают простыми и составными. Составные включают в себя несколько простых. Фигурные скобки с содержимым {...} - это составной оператор блока. Здесь этот блок содержит много простых операторов. Составной оператор описания функции main состоит у нас из заголовка и тела функции main. Тело функции - это блок.

После предъявления преподавателю текста программы студент допускается к компьютеру.

В-третьих, студент должен показать преподавателю правильно работающую на компьютере программу. За это он получает третью подпись преподавателя и зачет за лабораторную работу. После запуска приведенной выше программы на черном экране мы увидим:

Задание 1
b=2.2 Enter
c=1.0285
Для продолжения нажмите любую клавишу...

Здесь и далее подчеркнутым шрифтом выделено то, что печатает пользователь в диалоге с работающей программой.

Схема 1. Создание нового проекта в среде Microsoft Visual Studio

1) Окно: Начальная страница - Microsoft Visual Studio

Вверху: **Файл → Создать → Проект**

2) Окно: Создать проект

Слева выбрать: **Visual C++**	**По центру выбрать:** **Пустой проект**	
Снизу ввести: **Имя:** **Проект1** **Если нужно, напечатать Проект2 и т.п.** **Расположение:** **D:\Prog\ЭТ11п2_Иванов_ИИ** **Выбрать свою папку по кнопке Обзор** **Имя решения:** **Проект1** **Если нужно, напечатать Проект2 и т.п.** **В конце работы с окном, снизу справа нажать ОК**		

3) Окно: Обозреватель решений - справа вверху

Файлы исходного кода Щелчок правой клавишей мыши → **Добавить → Создать элемент**

4) Окно: Добавление нового элемента - Проект1

Слева выбрать: **Visual C++**	**По центру выбрать:** **Файл C++ (.cpp)**	
В конце работы с окном, снизу справа нажать Добавить		

5) Окно: Исходный код.cpp - слева вверху

Здесь можно писать текст программы

Запуск программы - **Локальный отладчик Windows** или **Ctrl+F5**

Схема 2. Открытие проекта Проект1 в среде Microsoft Visual Studio

Легкий способ

(если Windows настроена для работы с расширениями файлов SLN)

В проводнике Windows открываем свою папку, например, **D:\Prog\ЭТ11п2_Иванов_ИИ**, в ней открываем папку **Проект1**, в ней делаем двойной щелчок по файлу **Проект1.sln**.

Универсальный способ

Из Windows запускаем "чистую" Visual Studio (без проектов).

1) Окно: Начальная страница - Microsoft Visual Studio

Вверху: **Файл → Открыть → Решение или проект**

2) Окно: Открыть проект

Открываем свою папку, например, **D:\Prog\ЭТ11п2_Иванов_ИИ**, в ней открываем папку **Проект1**, в ней открываем файл **Проект1.sln**.

Независимо от способа открытия

Если на экране нет окна Исходный код.cpp с текстом программы, то оно вызывается так:

Окно: Обозреватель решений - справа вверху

Последовательно открывать (щелкчком) пункты:

Файлы исходного кода → Исходный код.cpp

Нам нужно разобрать следующие вопросы:

Вопрос 1. Запустив Visual Studio, как нам получить электронный лист, где мы будем писать текст нашей программы. **Вопрос 2.** Отладка программы - это поиск ошибок в тексте программы. Никому никогда еще не удавалось написать программу сразу без ошибок.

По вопросу 1. До запуска Visual Studio мы должны выяснить, в какой папке студентам разрешается сохранять свою информацию, пусть для примера это будет D:\Prog Тогда в проводнике Windows войдем в эту папку. Там нужно создать папку для конкретного студента. Пусть это будет для примера Иванов Иван Иванович из группы ЭТ-11 подгруппы 2. Тогда он должен создать папку ЭТ11п2_Иванов_ИИ - используем только большие и малые русские буквы и нижнее подчеркивание. Если в данной группе есть еще один Иванов И.И., то он создает папку ЭТ11п2_Иванов_ИИ2 и т.д. Далее запускаем Visual Studio, при этом не берем с ней никаких указанных проектов - запускаем "чистую" Visual Studio. **Схема 1** показывает дальнейшие наши действия. **Схема 2** показывает наши действия, если нам надо открыть уже существующий проект, пусть это будет для определенности Проект1.

По вопросу 2. Бывают синтаксические и семантические ошибки. Синтаксические - это нарушение формальных правил языка C++. Например, **cout << "b = "** мы забыли поставить точку с запятой **;** в конце оператора, надо писать **cout << "b = ";** Такие ошибки нам помогает выявлять машина. Семантические ошибки - это когда программа работает, но не так, как надо. Формально в ней нет нарушений правил C++. Такие ошибки выявить очень сложно и для этого нет общих способов.

Пример поиска синтаксической ошибки:

Вместо void main() напечатаем oid main(). Запустим программу (Локальный отладчик Windows или Ctrl+F5).

Окно: Microsoft Visual Studio

Следующий проект устарел.

Выполнить его сборку?

Нажимаем кнопку **Да**.

Окно: Microsoft Visual Studio

Возникли ошибки сборки. Продолжить и запустить последний успешно построенный вариант?

Нажимаем кнопку **Нет**.

Окно: Список ошибок

Ошибки указаны по пунктам error.

Двойным щелчком выбираем любой пункт error
Окно: Исходный код.cpp
Слева появляется маркер напротив строки с ошибкой.

Данная лекция должна вам помочь выполнить лабораторную работу № 1. Каждый получит свой вариант 1, 2, 3, ... в соответствии с номером по журналу, формула для вычисления будет у каждого своя.

+ - * / для C++ это операции сложения, вычитания, умножения, деления соответственно, между ними имеются приоритеты и используются скобки, принятые в математике. Операции возведения в степень ^ в C++ нет, далее таблица 2 рассказывает про математические функции в C++.

Таблица 2

Математические функции в C++

Функция	**Описание**	**Пример**
abs(a)	модуль или абсолютное значение от **a**	abs(-3.0)=3.0 abs(5.0)= 5.0
sqrt(a)	корень квадратный из **a,** причём **a** не отрицательно	sqrt(9.0)=3.0
pow(a, b)	возведение **a** в степень **b**	pow(2, 3)=8
ceil(a)	округление **a** в большую сторону	ceil(2.3)=3.0 ceil(-2.3)=-2.0
floor(a)	округление **a** в меньшую сторону	floor(12.4)=12 floor(-2.9)=-3
fmod(a, b)	вычисление остатка от деления a на b	fmod(4.4, 7.5) = 4.4 fmod(7.5, 4.4) = 3.1
exp(a)	вычисление $\mathbf{e^a}$, где e≈2.71 - число Эйлера	exp(0)=1
sin(a)	**a** задаётся в радианах	
cos(a)	**a** задаётся в радианах	
log(a)	натуральный логарифм **a** (основание - число Эйлера)	log(1.0)=0.0
log10(a)	десятичный логарифм **a**	Log10(10)=1
asin(a)	арксинус **a**, где **-1.0 < a < 1.0**	asin(1)=1.5708

Т.2. Форма программы с кодом возврата и без него

Текст программы в форме без кода возврата приведен в 0. Чтобы получить аналогичную программу в форме с кодом возврата нужно: заменить **void main** на **int main** в заголовке функции и в теле функции **main** перед закрывающей его **}** фигурной скобкой добавить оператор **return 0;** вот и все! Текст программы станет таким:

```
#include <iostream>
#include <locale>
```

```
#include <cmath>
#include <string>
using namespace std;
int main() // изменили заголовок функции
{
	setlocale(LC_ALL, "rus");
	double a=1.1, b, c;
	cout << "Задание 1" << endl << "b = ";
	cin >> b;
	c = a*b + sin(b) - 2.2;
	cout << "c= " << c << endl;
	system("pause");
	return 0; // добавили оператор
}
```

Теперь наша программа в конце работы будет передавать операционной системе Windows нулевой "код возврата". Наша первоначальная программа в форме без кода возврата делала то же самое автоматически. Опытные программисты и системные администраторы манипулируют "кодами возврата" - у них они не обязательно нулевые. Но мы - новички! Зачем это нам? Чтоб знали! В некоторых учебниках C++ пишут программы в форме с кодом возврата.

Т.3. Простые типы данных

Тип **void** - отсутствие данных, этот экзотический тип данных используется как возвращаемый тип функции, которая ничего не возвращает (см. 0); **int** или **signed int** - целое или знаковое целое (синонимы), переменная этого типа занимает 4 байта памяти; **double** или **long float** - действительное двойной точности или длинное действительное (синонимы), 8 байтов; **bool** - Булевский (логический) тип, такая переменная может иметь значения **true** или **false** - это 1 или 0 соответственно, 1 байт; **char** - символ, 1 байт, например, 'a' - заключается в апострофы, '\0' - нулевой символ, это не литера '0' арабской цифры ноль, это символ, у которого ASCII-код равен нулю; **string** - строка, например, "abc" - заключается в кавычки, в данном случае занимает в памяти 4 байта, поскольку после символа 'c' в памяти записан нулевой символ '\0', сигнализирующий о конце строки, тип string работает только при наличии диррективы #include <string>. Выше указаны не все простые типы данных!!!

Т.4. Простые литералы

Это различные формы записи констант простых типов, ниже даны примеры.

Тип int

-3567 7693 +5831 - для переменных int диапазон чисел от -2^{31} до $2^{31}-1$ или от -2147483648 до 2147483647, т.е. диапазон приблизительно $\pm$ 2 млрд.

Для беззнаковых целых unsigned int диапазон чисел от 0 до 2^{32}-1 или от 0 до 4294967295, т.е. приблизительно от 0 до 4 млрд.
Для литералов типа int кроме 10-тичной также допустимы 8-ричная, 16-ричная, 2-ичная записи с помощью префиксов 0, 0x, 0b соответственно.
12 014 0xC 0b1100 - это то же самое число соответственно в 10-ной, 8-ной, 16-ной, 2-ной формах.

Тип double

-537.88 -0.53788e3 - между целой и дробной частью ставится точка, а не запятая, фактически это то же самое число в обычной и в инженерной формах, здесь десятичная мантисса m_{10}=-0.53788, десятичный порядок p_{10}=3, в принципе для double количество значащих цифр m_{10} не должно быть более 15 цифр, кроме того $-308 \le p_{10} \le +308$, для float длина m_{10} не должна быть более 7 цифр, кроме того $-37 \le p_{10} \le +37$

Тип bool

true равная 1 и false равная 0 - вот и все литералы типа bool

Тип char

'w' '\167' - тот же символ в обычной и в альтернативной форме, ее схема такая:
'\код символа' - код символа должен быть в 8-ной форме без префикса, '\0' - нулевой символ, пример альтернативной формы записи.

Тип string

Интересны управляющие строки, примеры: "\n" - переход в начало следующей строки; "\t" - соответствует клавише TAB; "\\" - состоит из символа \ обратного слеша; "\"" - состоит из символа " кавычки; "\'"- состоит из символа ' апострофа.

Рассмотрим следующий фрагмент C++ кода:

```
string st;
st="abc\\def\nghi\n";
cout << st;
system("pause");
```

В результате выполнения этого фрагмента мы увидим на экране:

```
abc\def
ghi
Для продолжения нажмите любую клавишу...
```

Переменная типа string может фактически содержать в себе несколько строк экрана. В стандартном пространстве имен std определена константа endl равная "\n", поэтому после подключения std можно писать endl вместо "\n". Что такое пространство имен std и как его подключать - см. 0 и 0.

Т.5. Идентификаторы

Это имена переменных и функций, которые придумывают сами пользователи Visual C++, т.е. мы. При этом следует придерживаться 4-х правил:
1) Имена могут состоять из:

больших и малых английских букв A...Z a...z
символа _ нижнего подчеркивания
арабских цифр 0...9

2) Имя не может начинаться с цифры
3) Имя не должно совпадать ни с одним из ключевых слов языка C++
4) Большие и малые буквы значимы в составе имени

SumQuadr sumQuadr Sumquadr sumquadr - это разные имена
x x25 _x x_25 - это допустимые имена
25x 2x5 7_x - это не допустимые имена.

В математическом моделировании часто используется обозначение величин греческими буквами, например: β, γ, δ. Как идентифицировать эти величины в программах C++. Нужно работать с переменными beta, gamma, delta соответственно.

Т.6. Подробнее об арифметических операциях и выражениях

По правилам C++ если оба числа, принимающие участие в некоторой операции имеют одинаковый тип, то результат будет иметь тот же тип, например, 14/3 сосчитается как 4. Поскольку 14 и 3 - целые типа int, то результат тоже будет целым типа int. Математически точный результат - это действительное число 4,666... или 4,(6) - т.е. 6 в периоде. При преобразовании действительного double в целое C++ не округляет, а просто отбрасывает дробную часть. Поэтому результат будет 4. Чтобы получить более точный результат деления, нужно писать выражение так 14.0/3.0 или, например, так 14/3.0 - почему?

По правилам C++ если числа, принимающие участие в некоторой операции имеют различные типы, то из этих типов выбирается самый сложный, и все эти числа преобразуются к этому сложному типу, результат будет иметь тот же сложный тип. В случае 14/3.0 число 14 имеет тип int, а число 3.0 имеет тип double. Число 14 будет автоматически преобразовано в double, т.е. в 14.0, следовательно, и результат при этом будет иметь тип double и сосчитается как 4.66666666666666 - мантисса из 15 цифр. Рассмотрим следующий фрагмент C++ кода:

```
int a=14, b=3;
double c;
c=a/b;
cout << c << endl;
```

При этом с сосчитается как 4.00000000000000. А надо 4.66666666666666 - как быть? Можно написать (a+0.0)/b Но, обычно пишут так (double)a/b, здесь (double)a - операция приведения a к типу double. Аналогично, допустимы операции приведения к другим простым типам.

В C++ есть экзотические арифметические операции: оператор a+=b; эквивалентен a=a+b; оператор a-=b; эквивалентен a=a-b; оператор a*=b; эквивалентен a=a*b; оператор a/=b; эквивалентен a=a/b; оператор a++; эквивалентен a=a+1; оператор a--; эквивалентен a=a-1; операция % означает

нахождение остатка от деления двух целых чисел, например, 17%3 даст 2 в качестве результата.

Т.7. Директивы

Директивы записываются в самом начале текста программы. Обычно они представляют собой разрешения на использование чего-либо в тексте программы - см. 0. Если бы, например, наша программа не использовала математические функции, то диррективу **#include <cmath>** она бы могла не содержать. Однако лучше в каждой программе писать все указанные в этом примере диррективы. Система программирования обрабатывает диррективы до начала трансляции - т.е. до перевода программы с С++ на машинные коды. Далее транслируется не исходный текст программы, а несколько измененный в соответствии с диррективами. В переводе с английского **include** означает **включить**. Например, включить текст некоторой главы старой Конституции в текст новой Конституции. Например, вместо диррективы **#include <iostream>** в текст программы включается текст из файла iostream.h - файлы с такими расширениями называются заголовочными. Если некоторый текст часто нужно включать в программу, то рационально пользоваться диррективой.

Т.8. Простые операторы

Они обязательно заканчиваются ; точкой с запятой, примеры:

system("pause"); - оператор вызова функции system с аргументом "pause". Вызов функции предполагает выполнение описанных в ней операторов. Их работа зависит от параметра типа string, его значение "pause" - это и есть аргумент функции. В данном случае аргумент один, но если у функции их много, то они указываются списком через запятую. Аргументов у функции может не быть вообще, тогда пишут () круглые скобки с пустым содержимым - см. 0.

double x=1.2, y, z, a, b; - оператор создания и инициализации переменных, в данном случае созданы 5 переменных x, y, z, a, b типа double, переменная x инициализирована числом 1.2, при этом значение y, z, a, b не определено. Далее мы узнаем, что переменные бывают глобальные и локальные. Чаще применяются локальные.

const double pi=3.14159; - оператор констант, он создает ячейку памяти с именем **pi** и записывает туда **3.14159**, при этом С++ следит, чтобы после этого в эту ячейку не было записано другого числа.

y=sin(x)-2.3; - оператор присваивания, мы также видим вызов функции sin с аргументом x равным 1.2 - см. выше оператор double. В данном случае говорят, что функция sin имеет возвращаемый тип double, поскольку значение sin - это число типа double. **z=y=sin(x)-2.3;** - в С++ допустимо и такое присваивание.

using namespace std; - оператор использования информации из некоторого хранилища. В Visual C++ есть хранилище std - стандартное. Это пространство имен - namespace. Там хранятся имена констант, переменных и функций, созданных не нами. Некоторые из них нам нужны, например, функция system,

строковая константа endl и т.п. Если бы в нашей программе не было бы этого оператора использования, то обращаться к упомянутым константе и функции соответственно пришлось бы так: std::endl и std::system("pause"); В программе операторы использования записываются сразу за директивами.

; - пустой оператор, он не выполняет никаких действий. Из-за него, если мы случайно напишем лишнюю ; точку с запятой, то ошибки не будет: using namespace std;;;

cin >> x; - оператор ввода из стандартного входного потока символов в переменную x. Это поток символов с консоли т.е. с клавиатуры.
Пусть x имеет тип double. В работе программы происходит пауза, а на черном экране мигает курсор, мы печатаем на клавиатуре, например, -3.48 и нажимаем Enter. Это число и будет введено в переменную x. Оператор **cin >> y >> z >> a >> b;** этот оператор обеспечивает ввод 4 элементов данных из стандартного входного потока символов в 4 переменные y, z, a, b. Разделителями элементов данных во входном потоке являются символы пробела, табуляции, конца строки. Если x, y, z, a, b - это 4 числа, то их можно, например, ввести: либо в столбик, нажимая Enter после печати каждого числа вот так:
2.5 Enter
3.4 Enter
4.3 Enter
5.2 Enter

либо в строчку, вводя пробел между числами, а после ввода последнего числа нажав Enter вот так:
2.5 3.4 4.3 5.2 Enter
При этом y, z, a, b будут равны соответственно 2.5, 3.4, 4.3, 5.2

cout << x << y << st << n << si; - оператор вывода на консоль - на экран. В данном случае на черный экран последовательно выводятся значения переменных x, y, st, n, si, т.е. в данном случае имеется 5 элементов вывода. Элементами вывода могут быть переменные, выражения, литералы любых простых типов.

return d; - оператор возврата значения d из некоторой функции с любым возвращаемым типом, кроме void - см. 0. Оператор **return;** допустим в теле функции с возвращаемым типом void. Операторы **return;** и **return d;** заканчивают работу операторов внутри тела функции.

Т.9. Составные операторы, блок

Их также называют конструкциями языка С++. Это конструкции блока, описания функций, ветвлений, циклов и т.п. После них ставить ; точку с запятой обычно не нужно, хотя это и не ошибка. Блоком мы уже пользовались, он имеет такую структуру: **{ Операторы }**. Операторами выхода из блока называют следующие простые операторы: **break** - оператор выхода из блока, непосредственно содержащий этот оператор, см. 0; **return** - выход из функции, содержащей этот оператор, см. подробнее о нем раздел ПРОСТЫЕ ОПЕРАТОРЫ; **goto** - если это переход на метку вне этого блока, см. 0.

Т.10. Конструкция описания функции

Общая схема конструкции состоит из 2-х компонентов:

Заголовок_функции
Блок_тела_функции // это описанный выше блок

Общая схема заголовка функции:

Возвращаемый_тип Имя_функции (Описание_параметров)

Если Возвращаемый_тип - это не void, то **Операторы** из **Блока_тела_функции** обязательно должны содержать хотя бы один оператор возврата **return d;** где d - возвращаемое функцией значение. Пример описания функции на языке С++:

```
/* Описание функции SinSumQuadr
Она вычисляет sin суммы квадратов параметров a и b */
double SinSumQuadr(double a, double b) // заголовок функции
// в теле функции SinSumQuadr используются локальные a, b, c
{ double c; c = sin(a*a + b*b); return c; } // тело функции
```

Математическое определение этой функции: $SinSumQuadr(a, b) = \sin(a^2 + b^2)$, приведенное выше описание функции на языке С++ позволяет компьютеру понять именно так алгоритм ее вычисления. Теперь мы можем вызывать эту функцию много раз с различными аргументами. Параметры a и b, а также переменная c при этом всякий раз заново создаются при вызове заголовка и входе в тело функции для вычисления ее значения и всякий раз уничтожаются при выходе из тела функции. В связи с их локализацией внутри функции переменные a, b, c называются локальными.

Теперь напишем главную функцию с именем **main**, которая в процессе своей работы будет вызывать 2 раза функцию **SinSumQuadr**. При запуске программы (проекта в целом) будет происходить вызов именно функции **main** в том виде, как она нами будет написана.

```
// Описание главной функции main
void main()
{
    setlocale(LC_ALL, "rus"); // разрешение русского алфавита при
выводе
    double a=1.3, b=2.8, c;
```

```
// в этом блоке используются локальные a, b, c функции main
        c = SinSumQuadr(a, b);
        cout << "1-й результат: " << c << endl;
        a=3.2; b=4.3;
        c = SinSumQuadr(a, b);
        cout << "2-й результат: " << c << endl;
        system("pause");
}
```

Локальные переменные a, b функции main и локальные переменные a, b функции SinSumQuadr - это разные переменные. Первые существуют все время, пока работает функция main, а вторые за это время 2 раза были созданы и уничтожены. Если две функции пишут два разных программиста, то они могут никак не согласовывать между собой имена своих локальных переменных. К моменту выполнения функции main компьютер уже должен знать, как устроена функция SinSumQuadr, следовательно, в тексте программы описание SinSumQuadr должно стоять до описания main. А в начале текста программы должно быть:

```
// Начало программы
#include <iostream>      // разрешение ввода и вывода
#include <locale> // разрешение национальных алфавитов
#include <cmath>         // разрешение математических функций
#include <string> // разрешение  работы со строками
using namespace std;     // разрешение стандартного пространства имен
```

При запуске такой программы мы бы увидели вот что:

```
1-й результат: -0.105028
2-й результат: -0.440061
Для продолжения нажмите любую клавишу...
```

Приведем полный текст программы, о которой выше шла речь:

```
// Начало программы
#include <iostream>      // разрешение ввода и вывода
#include <locale> // разрешение национальных алфавитов
#include <cmath>         // разрешение математических функций
#include <string> // разрешение  работы со строками
using namespace std;     // разрешение стандартного пространства имен
/* Описание функции SinSumQuadr
Она вычисляет sin суммы квадратов параметров a и b */
double SinSumQuadr(double a, double b) // заголовок функции
// здесь используются локальные a, b, c функции SinSumQuadr
{ double c; c = sin(a*a + b*b); return c; } // тело функции
// Описание главной функции main
void main()
{
```

```
	setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
	double a=1.3, b=2.8, c;
// здесь используются локальные a, b, c функции main
	c = SinSumQuadr(a, b);
	cout << "1-й результат: " << c << endl;
	a=3.2; b=4.3;
	c = SinSumQuadr(a, b);
	cout << "2-й результат: " << c << endl;
	system("pause");
}
```

Т.11. Конструкция описания программы

Общая схема конструкции состоит из 4-х компонентов:

Директивы
Операторы_использования
Операторы_создания_и_инициализации_глобальных_переменных
Описания и прототипы функций

Какие-то из первых 3-х компонентов могут отсутствовать. Что такое прототип функции? Прототип функции - это заголовок функции и ; точка с запятой. Правила языка С++ в случае отсутствия в тексте программы описания некоторой функции до ее вызова требуют наличия в тексте программы прототипа этой функции до ее вызова. Обратимся к предыдущему примеру программы. Прототип SinSumQuadr - это:

```
double SinSumQuadr(double a, double b); // Прототип функции SinSumQuadr
```

Если в тексте этой программы прототип SinSumQuadr поставить выше описания функции main, а ниже описания функции main - описание функции SinSumQuadr, то программа после запуска будет работать точно так же, как показано выше. Теперь о глобальных переменных. В предыдущем примере программы оператор **double a=1.3, b=2.8, c;** перенесем из описания функции main в компоненту, названную началом программы. Программа после запуска будет работать точно так же, как показано выше, но эти переменные a, b, c станут глобальными. Если разные функции пишут разные программисты, то они согласуют между собой именно глобальные переменные. Они нужны, чтобы все функции могли пользоваться некоторой единой информацией. Приведем полный текст программы после внесенных изменений, о которых выше шла речь:

```
// Начало программы
#include <iostream>      // разрешение ввода и вывода
#include <locale> // разрешение национальных алфавитов
#include <cmath>         // разрешение математических функций
#include <string> // разрешение  работы со строками
using namespace std;     // разрешение стандартного пространства имен
double a=1.3, b=2.8, c; // теперь это глобальные переменные
double SinSumQuadr(double a, double b); // Прототип функции SinSumQuadr
```

```
// Описание главной функции main
void main()
{
	setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
// в этом блоке используются глобальные a, b, c
	c = SinSumQuadr(a, b);
	cout << "1-й результат: " << c << endl;
	a=3.2; b=4.3;
	c = SinSumQuadr(a, b);
	cout << "2-й результат: " << c << endl;
	system("pause");
}
/* Описание функции SinSumQuadr
Она вычисляет sin суммы квадратов параметров a и b */
double SinSumQuadr(double a, double b) // заголовок функции
// в теле функции SinSumQuadr используются локальные a, b, c
{ double c; c = sin(a*a + b*b); return c; } // тело функции
```

Т.12. Булевы операции и выражения

Их результат имеет тип bool, т.е. он может быть true и false или 1 и 0 - что соответственно то же самое. Существуют операции над числовыми выражениями, результат которых - это true или false, например, d<0 в зависимости от значения d может быть true или false. Вот эти операции: == проверка на равенство, не путать с = оператором присваивания, != проверка на неравенство, < и > проверка на меньше и больше соответственно, <= и >= проверка на меньше либо равно и больше либо равно соответственно, Существуют операции над Булевскими выражениями: ! отрицание (не), && конъюнкция (и), || дизъюнкция (или), например, составим такое выражение: !(a==5 && b==7), если a и b равны 5, то оно имеет значение true.

Т.13. Конструкции ветвления if

Их две - неполная и полная - см. **Рисунок 2** и **Рисунок 3** соответственно. Общая схема неполной конструкции: **if (Условие) Оператор** Общая схема полной конструкции: **if (Условие) Оператор1 else Оператор2**

Здесь: **Условие** - это выражение типа bool. **Оператор, Оператор1, Оператор2** - это простые или составные операторы, включая ветвления if.

Пример 1.

```
int d;
cout << "d = ";
cin >> d;
if (d>=0)
	cout << "this is good!\n";		// это есть гуд!
```

else

cout << "this is not good!\n"; // это не есть гуд!

Здесь полная конструкция if, где Оператор1 и Оператор2 - простые операторы.

Пример 2.

int d;
cout << "d = ";
cin >> d;

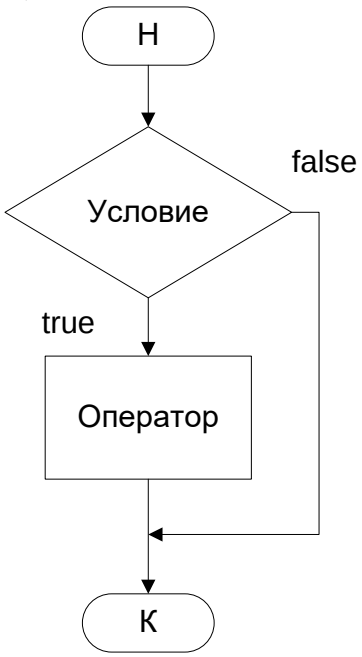


Рисунок 2. Неполная конструкция ветвления if

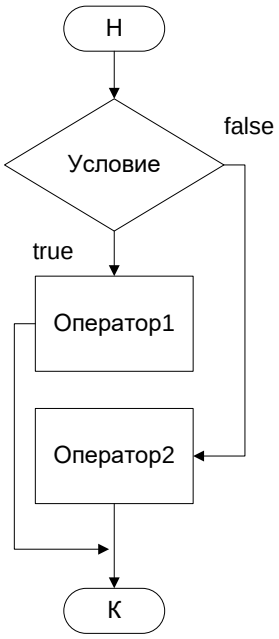


Рисунок 3. Полная конструкция ветвления if

if (d>0)

cout << "d is positive\n"; // d положительно

else

if (d=0)

cout << "d is zero\n"; // d нулевое

else

cout << "d is negative\n"\n"; // d отрицательное

Оператор2 для внешней полной конструкции if - это внутренняя полная конструкция if. Получилась составная конструкция с вложением.

Замечание 1. Примеры 1 и 2 демонстрируют хороший стиль записи кода C++ - стиль многострочный и с большими отступами, сделанными клавишей TAB. Конструкции C++ становятся более понятными при таком стиле записи. Писать все в одну строку - плохой стиль!

Замечание 2. Такие конструкции ветвления if приводят к известному еще с середины XX века (язык Algol-60) парадоксу двух if, рассмотрим такую составную конструкцию:

a=1; b=3; c=3; if (a>2) if (b>2) c=4; else c=5;

Как ее понимать? Здесь **else** относится к 1-му **if** или ко 2-му? 1-е предположение - **else** относится к 1-му **if**:

```
a=1; b=3; c=3;
if (a>2)
        if (b>2)
                c=4;
else
        c=5;
```

2-е предположение - **else** относится ко 2-му **if**:

```
a=1; b=3; c=3;
if (a>2)
        if (b>2)
                c=4;
        else
                c=5;
```

При 1-м предположении с стало бы 5. При 2-м предположении с стало бы 3 - именно оно и подтверждается практической проверкой в среде Microsoft Visual Studio 2013. Еще в языке Algol-60 было принято правило: **else** относится к ближайшемук нему **if**, отрадно, что оно сохранилось в Visual C++!

Т.14. Тернарная операция

Иногда она называется операцией ?: знак вопроса двоеточие, ее общая схема: **условие ? выражение1 : выражение2** Здесь: **условие** - это Булево выражение со значением true или false; **выражение1**, **выражение2** - это выражения любого типа, например, числовые. Пример: рассмотрим оператор присваивания: **Z = x<0 ? 1-cos(x) : sqrt(x+1);** Если **x<0** тогда **Z** будет присвоено значение **1-cos(x)** иначе - значение **sqrt(x+1)** Иногда использование в программах тернарных операций вместо составных операторов ветвления if упрощает написание программы.

Т.15. Простой оператор goto

Общая схема оператора goto такова: **goto Метка;** Метка - это идентификатор. При этом в теле данной функции обязательно должен быть Оператор, помеченный этой меткой следующим образом: **Метка: Оператор** Встретив оператор goto, компьютер далее будет выполнять не следующий за ним оператор, а помеченный оператор, далее - оператор, следующий за помеченным и т.д.

Пример:

```
int a;
cout << "a = ";
cin >> a;
if (a>7) goto M1; // такое называют условным переходом
b=3;
```

goto M2; // такое называют безусловным переходом
M1: b=4;
M2: cout << b << endl;

Если, например, в a введено 6, следовательно, a>7 - это false, значит оператор goto M1 не будет выполняться, далее следуют операторы b=3; goto M2; - они и будут выполняться, далее - помеченный M2 оператор cout << b << endl; и на экран будет выведено число 3.

Если, например, в a введено 8, следовательно, a>7 - это true, значит оператор goto M1 будет выполнен, далее - помеченный M1 оператор b=4; далее - следующий за ним оператор cout << b << endl; и на экран будет выведено число 4. Теоретически с помощью условных и безусловных переходов можно организовать любые сколь угодно сложные ветвления и циклы - это доказал советский академик А.А. Ляпунов, занимавшийся в частности математическим моделированием системы команд компьютера. Любой современный процессор имеет кодовые команды условных и безусловных переходов. Структурных конструкций ветвления и цикла C++ и других языков высокого уровня в процессорах нет. При трансляции (переводе) на машинный язык эти конструкции реализуются путем многочисленных условных и безусловных переходов. Приведенный выше фрагмент программы с goto эквивалентен следующему:

```
if (a>8)
        b=4;
else
        b=3;
cout << b << endl;
```

Его понять значительно легче! Если в программе много операторов goto, то понять ее очень трудно. Минимизировать количество операторов goto - хороший стиль программирования. Нидерландский ученый в области теоретического программирования Э. Дейкстра доказал теорему, что без операторов goto можно описать любой алгоритм. Хотя иногда стремление переделать программу так, чтобы непременно исключить из нее все операторы goto, приводит как раз к трудно понимаемой программе. Хотя Э. Дейкстра доказал, что это всегда можно сделать, но всегда ли нужно это делать?

Т.16. Конструкция ветвления switch

Эта конструкция имеtт следующую общую схему: **switch (ключевое_выражение) case-блок** Элемент **case-блок** имеет следующую общую схему: **{ case-фрагмент1 ... case-фрагментN default-фрагмент }** Здесь N≥1. Составляющая **default-фрагмент** может отсутствовать. Общая схема составляющей **case-фрагмент** такова:

case ключевое_значение: операторы

Общая схема составляющей **default-фрагмент** такова:

default: операторы

В последних двух общих схемах элемент **операторы** может отсутствовать, эти **операторы** могут содержать операторы выхода - **break**, **return**, **goto** - см. 0.

Пример 1.

```
char c;
cout << "input digit 0 or 1: "; // введите цифру 0 или 1:
cin >> c;
switch (c)
{
case '0':
        cout << "this is zero" << endl; // это ноль
case '1':
        cout << "this is one" << endl; // это один
default:
        cout << "this is not digit 0 or 1" << endl; // это не цифра 0 или 1
}
```

При вводе 1 мы увидим на черном экране:

input digit 0 or 1: 1 Enter
this is one
this is not digit 0 or 1

Дело в том, что в этом примере ни один из case-фрагментов не содержит операторов выхода. Компьютер сверху вниз ищет **case-фрагмент**, где **ключевое_значение** и **ключевое_выражение** равны, у нас это **case-фрагмент**, где **case '1'**, следовательно, выполняется оператор **cout << "this is one" << endl;** но конструкция **switch** предусматривает затем окончание всех проверок и выполнение всех последующих операторов из **case-блока**. Нужно было писать операторы выхода!

Пример 2.

```
char c;
cout << "input digit 0 or 1: "; // введите цифру 0 или 1:
cin >> c;
switch (c) {
case '0':
        cout << "this is zero" << endl; // это ноль
        break;
case '1':
        cout << "this is one" << endl; // это один
        break;
default:
        cout << "this is not digit 0 or 1" << endl; // это не цифра 0 или 1
}
```

В каждый case-фрагмент мы дописали оператор выхода **break**; который обеспечивает выход из всего **case-блока**. При вводе 1 мы теперь увидим на черном экране:

input digit 0 or 1: 1 Enter
this is one

При вводе 0 мы теперь увидим на черном экране:

input digit 0 or 1: 0 Enter
this is zero

При вводе буквы f мы как и раньше увидим на черном экране:

input digit 0 or 1: f Enter
this is not digit 0 or 1

Пример 3.

```
char c;
cout << "input digit 0...9: "; // введите цифру 0...9:
cin >> c;
switch (c) {
case '0':
case '1':
case '2':
case '3':
case '4':
        cout << "this digit less than 5" << endl; // эта цифра меньше 5
        break;
case '5':
case '6':
case '7':
case '8':
case '9':
        cout << "this digit more than 4" << endl; // эта цифра больше 4
        break;
default:
        cout << "this is not digit 0...9" << endl; // это не цифра 0...9
}
```

Это пример того, что в некоторых **case-фрагментах** вообще может не быть никаких операторов. При вводе 2 мы увидим на черном экране:

input digit 0...9: 2 Enter
this digit less than 5

Компьютер найдет **case-фрагмент**, где **case '2'**, но там никаких операторов нет. Компьютер пойдет далее, пока не выполнит оператор **cout << "this digit less than 5" << endl;** за ним стоит оператор **break**; который обеспечивает выход из всего **case-блока**.

При вводе буквы **w** мы увидим на черном экране:

input digit 0...9: w Enter
this is not digit 0...9

Написание подобных программ возможно с применением конструкций if вместо конструкций switch, но при этом их тексты будут трудно понимаемыми.

Т.17. Конструкции цикла while и do

Зачем нужны в программировании целые числа, например, типа int? Функционально они уступают действительным числам, например, типа double. Программисты называют целые числа (-37 или 48) точными, поскольку результаты арифметических действий над ними всегда абсолютно точны. Действительные числа (-7.57 или 6.85 - точность 2 цифры после точки) называют приближенными. Бухгалтерия дебет и кредит подсчитывает с точностью до копейки, поэтому финансовые программы работают только на целых числах. Если у чисел типа int (4 байта) разрядов слишком мало для выполнения необходимых расчетов, то используют тип long int (8 байтов). Расчеты надежнее, когда управляющие циклом переменные (синоним - счетчики) являются целыми - типов int или long int, другие вычисляемые циклически переменные могут быть действительными. В языках Pascal и Delphi разрешены лишь целые счетчики циклов, C++ разрешает любые.

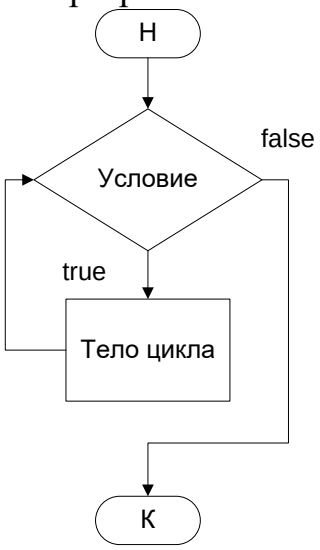


Рисунок 4. Конструкция цикла while

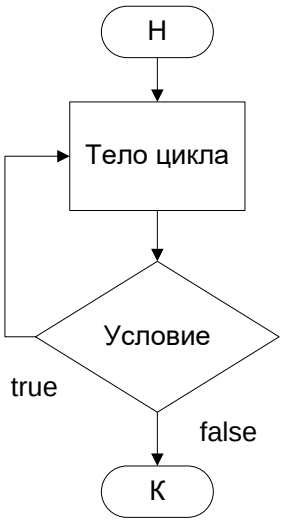


Рисунок 5. Конструкция цикла do

Общая схема конструкции цикла while такая: **while (Условие) Тело_цикла** Здесь **Условие** - это **выражение типа bool** и **Тело_цикла** - это **оператор** , простой или составной. Такая конструкция называется циклом с предусловием, поскольку Условие проверяется перед выполнением Тела цикла - см. **Рисунок 4**. Пример 1.

```
int s=0, i=1;
while (i<=10)
        {s+=i; i++;}
cout << s << "\t"<< i << endl;
```

Напоминаем: оператор **s+=i;** эквивалентен **s=s+i;** оператор **i++;** эквивалентен **i=i+1;** В переменной s циклически вычисляется сумма 1+2+...+9+10, которая равна 55, переменная **i** увеличивается от **1** до **11**, после чего **i<=10** становится равным **false**, и цикл завершается. На черный экран выведутся

числа **55** и **11** через табуляцию между ними. Аналогично написанному выше в теле цикла можно использовать операторы выхода: **break, return, goto** - см. 0.
Общая схема конструкции цикла do такая: **do Тело_цикла while (Условие);**
Это составной оператор, после которого **;** точка с запятой необходима!!!
Условие - это выражение типа bool. **Тело_цикла** - это **{Операторы}** - т.е. блок, содержащий операторы. Аналогично написанному выше в теле цикла можно использовать операторы выхода: **break, return, goto**. Такая конструкция называется циклом с постусловием, поскольку **Условие** проверяется после выполнения **Тела цикла** - см. **Рисунок 5** - цикл выполнится хотя бы 1 раз!
Пример 2.

```
int s=0, i=1;
do
        {s+=i; i++;}
while (i<=10);
cout << s << "\t" << i << endl;
```

Выполнится точно так же, как пример 1. В данном случае не важно, проверяется ли условие до или после операторов.

Пример 3.

```
int s=0, i=11;
do
        {s+=i; i++;}
while (i<=10);
cout << s << "\t" << i << endl;
```

Операторы **s+=i;** и **i++;** выполнятся только 1 раз, следовательно, **s** станет равным **11**, а **i** станет равным **12**. Затем при проверке **i<=10** станет равным **false**, следовательно, цикл завершится. На черный экран выведутся числа **11** и **12** через табуляцию между ними.

Т.18. Конструкция цикла for

Общая схема такая: **for (Инициализация_счетчика; Условие; Модификация_счетчика) Тело_цикла** Здесь **Тело_цикла** - это **оператор** , простой или составной. **Инициализация_счетчика** - это присваивание некоторой переменной (счетчику цикла) начального значения. Эту переменную можно тут же при инициализации и создавать, тогда она всякий раз создается при входе в цикл и уничтожается при выходе из него. Она называется локальной переменной цикла - см. ниже Пример 2. **Условие** - это **выражение типа bool**, цикл завершается, если при проверке оно оказывается равным **false**. **Модификация_счетчика** - это его инкремент (**счетчик++**) или его декремент (**счетчик--**). Аналогично написанному выше в теле цикла можно использовать операторы выхода: **break, return, goto** - см. 0. Конструкция **for** эквивалентен следующей конструкции:

```
{ Инициализация_счетчика;
while(Условие)
```

{Тело_цикла Модификация_счетчика;} } // блок вложен в блок

Пример 1.

```
int s=0, i;
for (i=1; i<=10; i++) s+=i;
cout << s << "\t" << i << endl;
```

Этот пример эквивалентен Примерам 1 из предыдущих двух тем, здесь Тело цикла - простой оператор.

Пример 2.

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

На черном экране печатается столбец чисел от 1 до 10, здесь тоже Тело цикла - простой оператор. Переменная i - локальная переменная цикла, она всякий раз создается при входе в цикл и уничтожается при выходе из него.

Пример 3.

```
double s=0;
int i, j;
for (i=1; i<=10; i++)
    for (j=1; j<=10; j++) s+=1.0/(i*i+j*j);
cout << s << endl;
```

Это пример вложенных циклов. Внешний цикл for по i имеет в качестве тела цикла внутренний цикл for по j. Вычисляется $s=\sum_{i=1}^{10}\sum_{j=1}^{10}\frac{1}{i^2+j^2}$ На черный экран будет выведено число 2.9681

Т.19. Одномерные массивы: создание и инициализация

В математике часто используют переменные с индексом, например, x_0, x_2, x_4, здесь 0, 2, 4 - значения индекса. В математике также пишут $\bar{x}=(x_0, x_1, x_2, x_3, x_4)$ - вектор $\bar{x}$ состоит из элементов x_0, x_1, x_2, x_3, x_4, т.е. он состоит из 5 элементов. Пусть эти элементы - действительные переменные. Как это реализовать на C++? Оператор **double x[5];** создает массив **x**, состоящий из 5 элементов - переменных типа **double**. Эти переменные - это **x[0], x[1], x[2], x[3], x[4]**. Общая схема такова:
Тип_элемента Имя_массива [Количество_элементов];
Количество_элементов - это **константа** целого типа - т.е., например, типов **int** или **long int**. Эта константа может быть предварительно создана оператором **const**, например:

```
const int five=5; double x[five];
```

Количество_элементов **не может быть** переменной - мы в данной книге будем рассматривать только так называемые **статические массивы**! Для **динамических массивов** Количество_элементов может быть переменной и даже любым выражением целого типа. При создании массив может быть инициализирован **составным литералом**, например: **double x[5] = { 5.2, -2.8, 4.3, 0.7, 3.1 };** При этом переменным **x[0], x[1], x[2], x[3], x[4]** будет присвоено **5.2, -2.8, 4.3, 0.7, 3.1** соответственно. Приведенный выше оператор можно было бы писать и так: **double x[ ] = { 5.2, -2.8, 4.3, 0.7, 3.1 };** В этом случае C++ сам

"додумается", что массив **x** состоит из 5 элементов. Если мы напишем такой оператор: **double x[5] = { 5.2, -2.8 };** то **x[0], x[1]** будет присвоено **5.2, -2.8** соответственно, а **x[2], x[3], x[4]** будет присвоен ноль. Если мы напишем такой оператор: **double x[5] = { };** то всем элементам массива **x** будет присвоен ноль.

Т.20. Работа с одномерными массивами

Пример 1. Напоминаем, оператор **s+=x[j];** эквивалентен **s=s+x[j];** это увеличение **s** на **x[j]**. Приведенный ниже фрагмент программы в переменной **s** постепенно копит сумму элементов массива **x**:

```
int j;
double s=0;
double x[5] = { 5.2, -2.8, 4.3, 0.7, 3.1 };
for (j=0; j<5; j++) s+=x[j];
cout << s;
```

При работе этого фрагмента будет выведено число **10.5**, поскольку оно равно **5.2 - 2.8 + 4.3 + 0.7 + 3.1**

Пример 2. Рассмотрим следующий фрагмент C++ программы:

```
double w[5], x[5] = { 5.2, -2.8, 4.3, 0.7, 3.1 };
// ниже используется табуляция \t и конец строки \n
cout << w[-6] << "\t" << w[-5] << "\t" << w[-4] << "\t" << w[-3] << "\t" << w[-2] << "\n";
```

При его работе будет выведено:

```
5.2    -2.8    4.3    0.7    3.1
```

По документам описания C++ индексы массива **w** должны находиться в диапазоне **0...4**. Индексы **-6...-2** лежат за границами этого диапазона, но C++ не запрещает их использование! Мы "нащупали" в памяти массив **x** путем использования массива **w** с "неправильными" индексами. Это пример стиля программирования с использованием побочных недокументированных эффектов языка программирования. Некоторые программисты гордятся, что умеют так программировать, C++ и Fortran этого не запрещают в отличие от большинства других языков программирования. За это многие любят C++ и Fortran. Такой стиль допустим при индивидуальном программировании "для себя", но не допустим при промышленном программировании, поскольку не способствует коллективной работе при разработке сложной программы. Такие программы будет понимать только автор. Аналогичный по работе фрагмент можно написать с использованием оператора цикла:

```
int j;
double w[5], x[5] = { 5.2, -2.8, 4.3, 0.7, 3.1 };
// ниже используется табуляция \t и конец строки \n
for (j=-6; j<=-2; j++) cout << w[j] << "\t";
cout << "\n";
```

К сожалению, в C++ имеются и недокументированные ошибки. Например, в C++ из Microsoft Visual Studio 2013 Professional попытка вычисления sqrt(8)

трактуется как ошибка, (это $\sqrt{8}$), а вот sqrt(8.0) нормально вычисляется. Ни в одном документе описания C++ про такую "ошибку" нет ни слова!

Т.21. Двумерные массивы: создание и инициализация

В математике часто используют переменные с двумя индексами, например, $a_{i,j}$ где i=0, 1, j=0, 1, 2. В этом случае матрицей A называют прямоугольную таблицу, составленную из этих переменных, причем первый индекс (i в данном случае) означает номер строки, а второй индекс (j в данном случае) означает номер столбца, при этом могут задаваться начальные значения этих переменных - элементов матрицы A:

$$A=\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \end{bmatrix}=\begin{bmatrix} 1.6 & 2.5 & 3.4 \\ 4.3 & 5.2 & 6.1 \end{bmatrix}$$

Здесь, например, $a_{0,0}$=1.6, $a_{1,2}$=6.1. Как это реализовать на C++?

double a[2][3]={{1.6, 2.5, 3.4}, {4.3, 5.2, 6.1}};

Такой оператор создает массив массивов **a**. Массив **a** состоит из 2 массивов, в каждом из которых по 3 элемента. Эти элементы - это переменные типа **double**, они инициализированы числами. Например, a[0][0]=1.6, a[1][2]=6.1. Массив **a** инициализирован составным литералом, который в свою очередь состоит из 2 составных литералов. По аналогии с одномерными массивами, можно было бы в индексах кое-что и пропустить, но это только левый индекс:

double a[ ][3]={{1.6, 2.5, 3.4}, {4.3, 5.2, 6.1}};

Чтобы "не запутать" компьютер, лучше всегда писать оба индекса. А вот такой оператор все 6 элементов массива **a** (переменных типа **double**) инициализировал бы нулями:

double a[2][3]={ };

Т.22. Работа с двумерными массивами

Пример 1. Оператор **swap(a, b);** производит взаимный обмен значений переменных **a** и **b**. Чтобы разрешить его использование, требуется дирректива **#include <algorithm>** Она, естественно, разрешает использование также многих других полезных алгоритмов. Например, оператора **sort** - он сортирует заданный одномерный массив, т.е. переставляет значения его элементов так, чтобы они были расположены по возрастанию. Подробнее про **sort** можно посмотреть в интернете, ключевые слова **C++ sort**. Рассмотрим ниже фрагмент программы C++:

```
double a[4][2]={{1.6, -3.1}, {4.3, -5.2}, {6.1, 7.1}, {2.2, 1.9}}; int j, k;
for (j=0; j<=3; j++) // цикл по j от 0 до 3, т.е. для каждой из 4 строк массива a
      if (a[j][1]<a[j][0])  /* если значение у 1-го элемента j-й строки массива
                              a  меньше, чем у нулевого */
            swap(a[j][1], a[j][0]);  // тогда меняются местами их значения
for (j=0; j<=3; j++) // цикл по j для каждой из 4 строк массива a
      {
```

```
for (k=0; k<=1; k++) /* цикл по k для каждого из 2-х элементов
                        j-й строки массива a */
    cout << a[j][k] << "\t"; /* выводим k-й элемент j-й строки
                                массива a и спецсимвол табуляции */
cout << endl; /* после вывода всех элементов j-й строки массива a
                 выводим спецсимвол конца строки */
}
```

При работе этого фрагмента на экран будет выведено:

```
-3.1  1.6
-5.2  4.3
6.1       7.1
1.9       2.2
```

Каждая из 4 строк массива **a** стала упорядоченной по возрастанию.

О выполнении лабораторных работ

1) К лабораторным работам нужно готовиться заранее, используя данное пособие. Задания на лабораторную работу по вариантам в соответствии с Вашим номером по журналу Вы получаете непосредственно перед выполнением лабораторной работы.

2) Нужно иметь 2 отдельные тетради не менее 48 листов каждая для лекций и лабораторных работ соответственно. На первой странице каждой из тетрадей необходимо указать: Дисциплину. Лекции это или Лабораторные работы. Свою группу. Фамилию, Имя, Отчество. На первой странице тетради для лабораторных работ необходимо начертить следующую таблицу:

Л. р.	**Блок-схема**	**Текст программы**	**Работа программы**
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

В пустых клетках этой таблицы преподаватель будет ставить свою подпись по мере выполнения Вами лабораторных работ. Также аналогичные подписи он ставит: на реальной блок-схеме в тетради; на реальном тексте программы в тетради; на описании работы программы в тетради. Итого: 6 подписей за каждую лабораторную работу.

3) Необходимо выяснить у персонала компьютерного класса, в какой папке студенты могут записывать свои проекты C++, файлы MS Word, MS Excel и т.п. В этой папке необходимо создать свою папку - см. 0. Необходимо выяснить правила установки Visual Studio на свой домашний компьютер - скорее всего это будет версия Community 2019 - русская версия.

Л.Р. 1. Линейные программы (ч. 1)

Вычислить и вывести на экран значение функции y. Исходные данные x, a и b ввести с клавиатуры.

$$y = \frac{\sqrt{b^2 - a^2}}{\sin x}$$

Блок-схема: представлена на **Рисунок 6**.

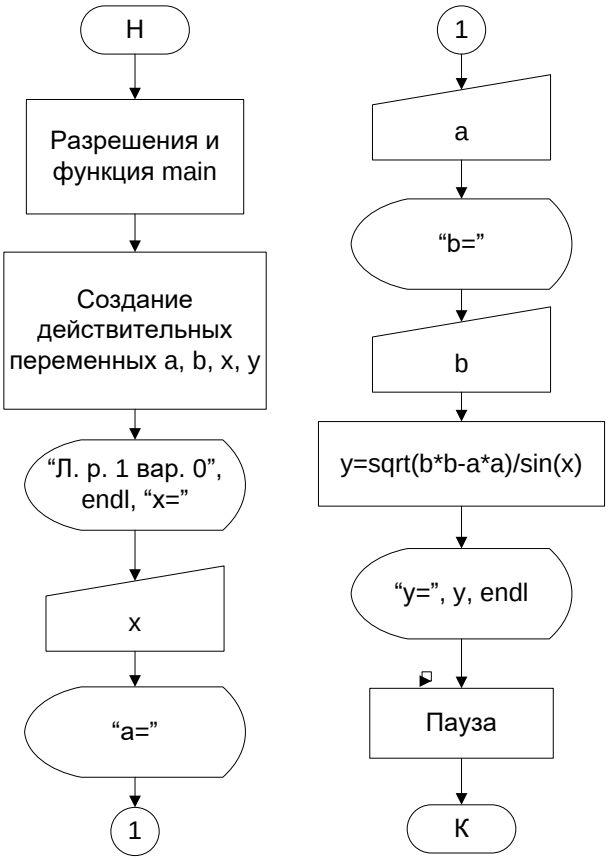


Рисунок 6. Бл.-сх. для Л.Р. 1

Текст программы в форме без кода возврата:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;      // разрешение стандартного пространства имен
/*      Ниже приведен составной оператор - описание главной функции main,
        состоящий из заголовка функции и тела функции       */
void main() // заголовок функции main() без аргументов
{ // операторы между { и } - это составной оператор-блок, это тело main
        setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
        double a, b, x, y; // создание действительных переменных a, b, x, y
        cout << "Л. р. 1 вар. 0" << endl << "действительное x="; /* вывод строк
        " Л. р. 1 вар. 0", endl, "действительное x=" */
        cin >> x; // ввод действительного числа в переменную x
        cout << "действительное a="; // вывод строки "действительное a="
        cin >> a; // ввод действительного числа в переменную a
        cout << "действительное b="; // вывод строки "действительное b="
        cin >> b; // ввод действительного числа в переменную b
        y = sqrt(b*b-a*a)/sin(x); /* присвоение переменной y
        выражения по заданию */
```

```
        cout << "y= " << y << endl; // вывод строки "y=", значения y, строки endl
        system("pause"); // пауза перед завершением программы
}
```

Текст программы в форме с кодом возврата:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;      // разрешение стандартного пространства имен
/*      Ниже приведен составной оператор - описание главной функции main,
        состоящий из заголовка функции и тела функции       */
int main()    // заголовок функции main() без аргументов
{ // операторы между { и } - это составной оператор-блок, это тело main
        setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
        double a, b, x, y; // создание действительных переменных a, b, x, y
        cout << "Л. р. 1 вар. 0" << endl << "действительное x="; /* вывод на экран
        строк " Л. р. 1 вар. 0", endl, "действительное x=" */
        cin >> x; // ввод с клавиатуры действительного числа в переменную x
        cout << "действительное a="; // вывод на экран строки "действительное
a="
        cin >> a; // ввод с клавиатуры действительного числа в переменную a
        cout << "действительное b="; // вывод на экран строки "действительное
b="
        cin >> b; // ввод с клавиатуры действительного числа в переменную b
        y = sqrt(b*b-a*a)/sin(x); /* присвоение переменной y
        выражения по заданию */
        cout << "y= " << y << endl; // вывод на экран строки "y=", значения y, endl
        system("pause"); // пауза перед завершением программы
        return 0; // передача Windows нулевого кода возврата
}
```

Работа представленных выше двух программ одинакова:

```
Л. р. 1 вар. 0
действительное x=1.5 Enter
действительное a=2.1 Enter
действительное b=5.7 Enter
y=5.31236
Для продолжения нажмите любую клавишу... Enter
```

Запустим программу еще раз с другими данными:

```
Л. р. 1 вар. 0
действительное x=1.5 Enter
действительное a=5.7 Enter
действительное b=2.1 Enter
y=-1.#IND
```

Для продолжения нажмите любую клавишу... Enter
Что такое -1.#IND ? В ячейку памяти y был записан такой код, поскольку она была вычислена с использованием недопустимой операции. Вычисление y производилось по формуле:
y=sqrt(b*b - a*a)/sin(x) (1.1)
У нас a=5.7, b=2.1, следовательно, b<a и b*b<a*a, значит b*b - a*a < 0. Но sqrt - это квадратный корень, его вычисление от отрицательного числа - это недопустимая операция.
Запустим программу еще раз с другими данными:
Л. р. 1 вар. 0
действительное x=0 Enter
действительное a=2.1 Enter
действительное b=5.7 Enter
y=1.#INF
Для продолжения нажмите любую клавишу... Enter
±1.#INF - это запись в ячейку памяти $\pm\infty$ (infinity - бесконечность). В данном случае 1.#INF равно +1.#INF - это $+\infty$. Такое обычно возникает при делении на нуль. Вычисление y производилось по формуле (1.1), теперь a=2.1, b=5.7, следовательно, b больше a, и квадратный корень вычисляется без проблем. Но x=0 влечет sin(x)=0, следовательно в формуле (1.1) мы делим на ноль.

Л.Р. 2. Линейные программы (ч. 2)

x=-1.2

y=3.5

z=2.3

$$\beta = z e^{x+y}$$

$$\gamma = \frac{\beta + 1}{x^2 + 1}$$

$$\delta = \frac{\gamma + 2}{y^2 + 2}$$

Здесь по трем переменным x, y, z, начальные значения которых заданы, необходимо вычислить по соответствующим формулам значения трех переменных β, γ, δ, причем вычислить именно в данном порядке.
Блок-схема: представлена на **Рисунок 7**. Будем вместо β, γ, δ писать соответственно beta, gamma, delta.
Текст программы в форме без кода возврата:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;      // разрешение стандартного пространства имен
/*      Ниже приведен составной оператор - описание главной функции main,
        состоящий из заголовка функции и тела функции       */
void main() // заголовок функции main() без аргументов
{ // операторы между { и } - это составной оператор-блок, это тело main
```

setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
double x=-1.2, y=3.5, z=2.3, beta, gamma, delta; /* создание действительных переменных с инициализацией некоторых из них */

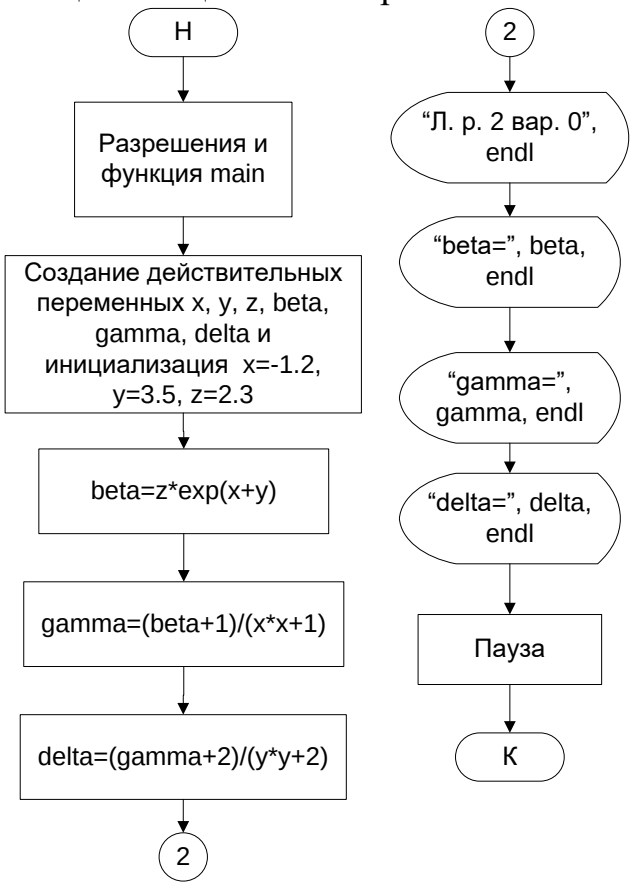


Рисунок 7. Бл.-сх. для Л.Р. 2

beta = z*exp(x+y); // вычисление beta
gamma = (beta+1)/(x*x+1); // вычисление gamma
delta = (gamma+2)/(y*y+2); // вычисление delta
cout << "Л. р. 2 вар. 0" << endl; // вывод строк "Л. р. 2 вар. 0", endl
cout << "beta=" << beta << endl; // вывод строки "beta=", значения beta, endl
cout << "gamma=" << gamma << endl; /* вывод строки "gamma=", значения gamma, строки endl */
cout << "delta=" << delta << endl; /* вывод строки "delta=", значения delta, строки endl */
system("pause"); // пауза перед завершением программы
}

Текст программы в форме с кодом возврата:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;      // разрешение стандартного пространства имен
```

```
/*      Ниже приведен составной оператор - описание главной функции main,
        состоящий из заголовка функции и тела функции       */
int main()    // заголовок функции main() без аргументов
{ // операторы между { и } - это составной оператор-блок, это тело main
        setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
        double x=-1.2, y=3.5, z=2.3, beta, gamma, delta; /* создание действительных
        переменных с инициализацией некоторых из них */
        beta = z*exp(x+y); // вычисление beta
        gamma = (beta+1)/(x*x+1); // вычисление gamma
        delta = (gamma+2)/(y*y+2); // вычисление delta
        cout << "Л. р. 2 вар. 0" << endl; // вывод строк "Л. р. 2 вар. 0", endl
        cout << "beta=" << beta << endl; // вывод строки "beta=", значения beta,
строки endl
        cout << "gamma=" << gamma << endl; /* вывод строки "gamma=", значения
gamma,      строки endl */
        cout << "delta=" << delta << endl; /* вывод строки "delta=", значения delta,
        строки endl */
        system("pause"); // пауза перед завершением программы
        return 0; // передача Windows нулевого кода возврата
}
```

Работа представленных выше двух программ одинакова:

```
Л. р. 2 вар. 0
beta=22.9406
gamma=9.81173
delta=0.828893
Для продолжения нажмите любую клавишу... Enter
```

Л.Р. 3. Разветвляющиеся программы

$$Z=\begin{cases}1-\cos x \text{ при } x<0\\ \sqrt{x+1} \text{ иначе}\end{cases}$$ где x = sin t.

Ввести t и вычислить Z, здесь x - вспомогательная переменная, все они действительные.

Блок-схема: представлена на **Рисунок 8**.

Текст программы в форме без кода возврата:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;      // разрешение стандартного пространства имен
// ниже - описание main, состоящее из заголовка и тела этой функции
void main() // заголовок main в стиле с кодом возврата
{ // между { и } - тело main
        setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
```

```
double Z, x, t; // создание действительных переменных Z1, x, m
cout << "Л. р. 3 вар. 0" << endl << "t="; // вывод " Л. р. 3 вар. 0", endl, "t="
cin >> t; // ввод в переменную t
```

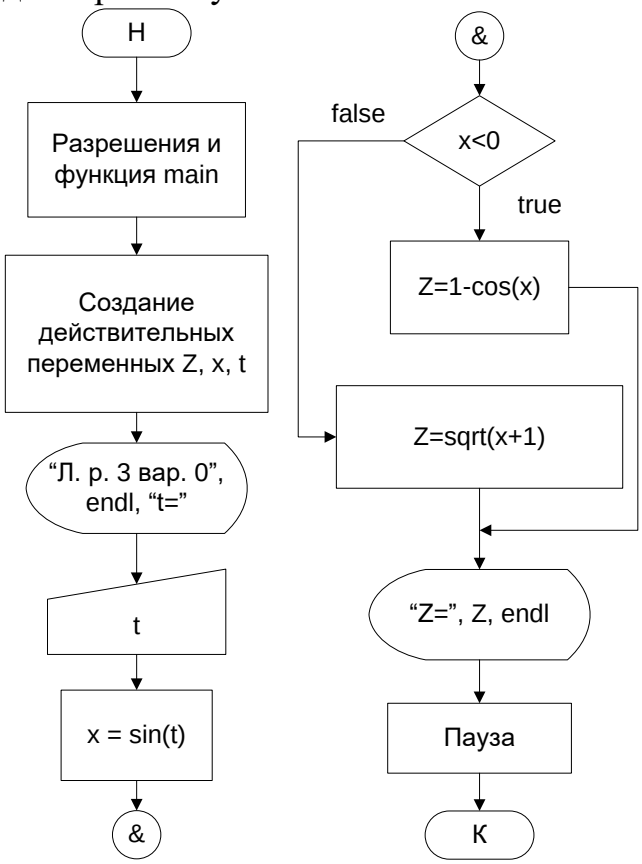


Рисунок 8. Бл.-сх. для Л.Р. 3

```
x = sin(t); // вычисление x
if (x<0) // начало полной конструкции ветвления if
        Z = 1-cos(x); // вычисление Z при x<0
else // иначе - начало альтернативной ветви
        Z = sqrt(x+1); // вычисление Z при x>=0
cout << "Z=" << Z << endl; // вывод строки "Z=", значения Z, строки endl
system("pause"); // пауза перед завершением программы
}
```

Альтернативный текст программы в форме с кодом возврата:

Программу было бы проще написать с использованием **тернарной** операции. В нашей программе вместо составного оператора if нужен следующий оператор присваивания: **Z = (x<0) ? (1-cos(x)) : (sqrt(x+1));**

Вот текст этой программы:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;      // разрешение стандартного пространства имен
```

```
// ниже - описание main, состоящее из заголовка и тела этой функции
int main() // заголовок main в стиле с кодом возврата
{ // между { и } - тело main
        setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
        double Z, x, t; // создание действительных переменных Z1, x, m
        cout << "Л. р. 3 вар. 0" << endl << "t="; /* вывод на экран строк
        " Л. р. 3 вар. 0", endl, "t=" */
        cin >> t; // ввод в переменную t
        x = sin(t); // вычисление x
        Z = (x<0) ? (1-cos(x)) : (sqrt(x+1)); /* Z присваивается результат
        тернарной операции */
        cout << "Z=" << Z << endl; // вывод на экран строки "Z=", значения Z, endl
        system("pause"); // пауза перед завершением программы
        return 0; // передача Windows нулевого кода возврата
}
```

Работа представленных выше двух программ одинакова:

Л. р. 3 вар. 0
t=3.5 Enter
Z=0.0608961
Для продолжения нажмите любую клавишу... Enter

Л.Р. 4. Декомпозиция разветвляющихся программ (ч. 1)

$F(d,Z)=\sqrt{d^2+Z^2}$	$d=\begin{cases} z^2(y+1) & \text{при } y<a^2 \\ z^2(y+2) & \text{при } a^2\le y\le b^2 \\ z^2(y+3) & \text{при } y>b^2 \end{cases} \quad Z=a^2-ab+b^2$

Ввести с клавиатуры a, b, y, z. Вычислить d, Z и F. Вывести на экран F.

Блок-схема: представлена на **Рисунок 9** - **Рисунок 13**. Блок-схема слишком сложна, необходима ее декомпозиция - разбивание на кусочки.

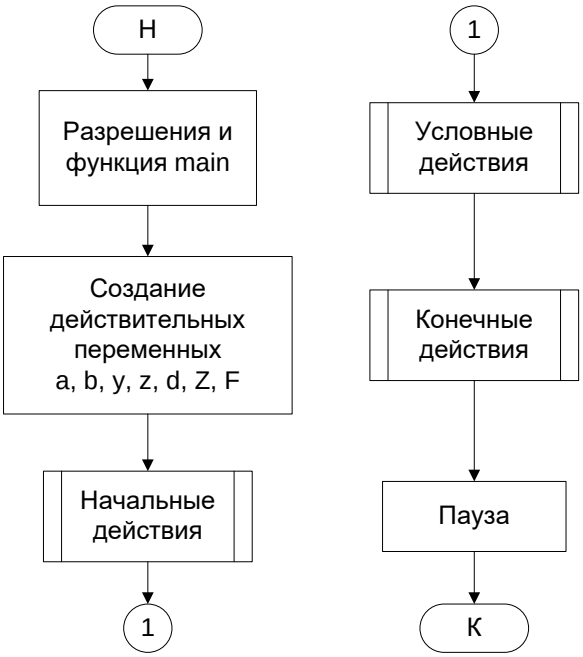


Рисунок 9. Главная бл.-сх.

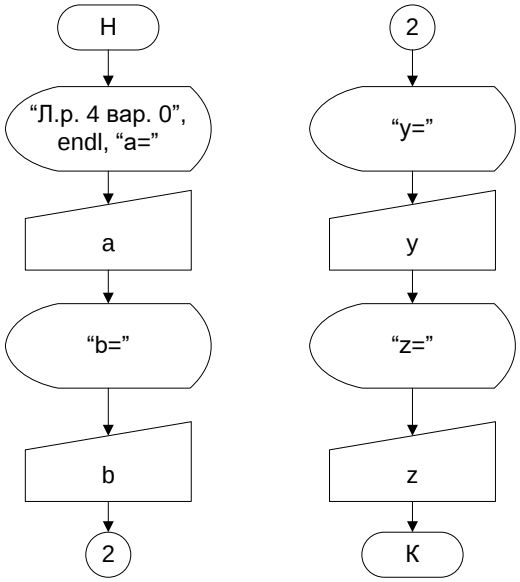


Рисунок 10. Бл.-сх. Начальные действия

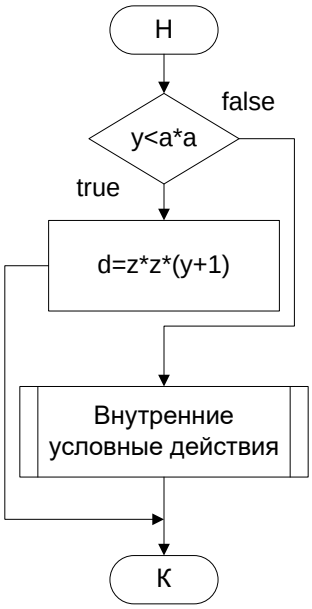


Рисунок 11. Бл.-сх. Условные действия

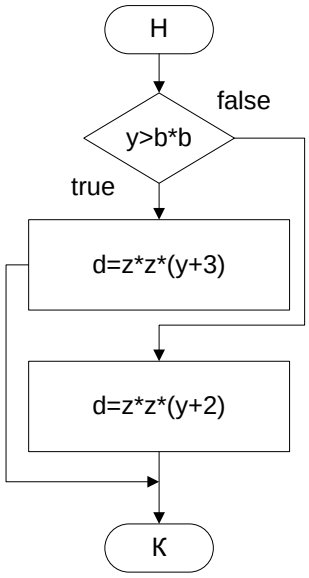


Рисунок 12. Бл.-сх. Внутренние условные действия

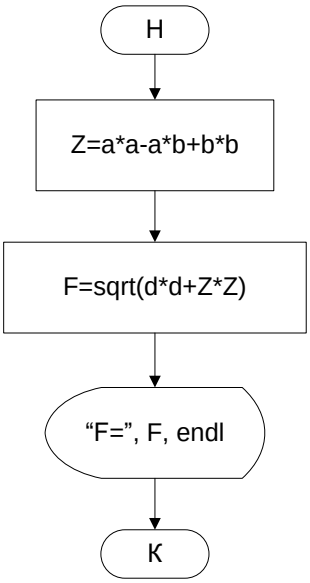


Рисунок 13. Бл.-сх. Конечные действия

Программу нужно разбить на отдельные подпрограммы, соответствующие блок-схемам - см. 0 - 0. В C++ подпрограммы называются **функциями**. Главной блок-схеме соответствует функция **main** - теперь понятно, почему она так называется. Остальные функции - пользовательские, поскольку мы придумываем их алгоритмы и имена. Русские буквы в идентификаторах C++ не допустимы. Пусть их заголовки будут как в таблице 3.

Все эти функции (включая main) возвращаемых значений не имеют (отсюда возвращаемый тип - void) и параметров не имеют, отсюда пустые скобки () в конце заголовка.

Если к заголовку функции справа приписать ; точку с запятой, то получится **прототип** функции. До использования функции (т.е. до ее вызова) в тексте программы должно быть ее описание или хотя бы ее прототип. Писать в начале программы прототипы чаще всего удобнее, а описание функций при этом удобнее писать после описания main. Заметим, что указанные в приведенных выше блок-схемах переменные должны быть видны из тела любой функции, следовательно, они должны быть глобальными, т.е. описываться до всех функций, включая main.

Таблица 3

Имена блок-схем и заголовки пользовательских функций

Имя блок-схемы	**Заголовок пользовательской функции**
Начальные действия	**void InitialActions ()**
Условные действия	**void ConditionActions ()**
Внутренние условные действия	**void InternalConditionActions ()**
Конечные действия	**void FinalActions ()**

Конструкция программы:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
```

```
#include <string>  // разрешение  работы со строками
using namespace std;       // разрешение стандартного пространства имен
Создание глобальных переменных
Прототипы пользовательских функций
void main () // заголовок main
{
        setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
        Операторы
}
```

Описания пользовательских функций

Например, математическая функция **sin(x)** имеет заголовок **double sin(double x)**, т.е. она имеет параметр **x** типа **double** и возвращает результат типа **double**, это **встроенная** в C++ функция, поэтому мы ее только вызываем, т.е. мы ее используем и никогда не пишем для нее ни описания, ни прототипа, ни заголовка - компьютер все это и без нас знает.

Текст программы с декомпозицией в форме без кода возврата:

```
#include <iostream>        // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;       // разрешение стандартного пространства имен
double a, b, y, z, d, Z, F; // создание ГЛОБАЛЬНЫХ действительных переменных
// ниже - прототипы пользовательских функций
void InitialActions();
void ConditionActions();
void InternalConditionActions();
void FinalActions();
// ниже - описание main, состоящее из заголовка и тела этой функции
void main() // заголовок main
{ // между { и } - тело main
        setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
        // ниже - вызовы пользовательских функции, выше - их прототипы
        InitialActions(); // вызов InitialActions
        ConditionActions(); // вызов ConditionActions
        FinalActions();// вызов FinalActions
        system("pause"); // пауза перед завершением программы
}
// ниже - описание InitialActions, состоящее из заголовка и тела этой функции
void InitialActions() // заголовок InitialActions
{ // между { и } - тело InitialActions
        cout << "Л. р. 4 вар. 0" << endl << "a="; /* вывод на экран строк
        " Л. р. 4 вар. 0", endl, "a=" */
        cin >> a; // ввод в переменную a
```

```
	cout << "b="; // вывод строки "b="
	cin >> b; // ввод в переменную b
	cout << "y="; // вывод строки "y="
	cin >> y; // ввод в переменную y
	cout << "z="; // вывод строки "z="
	cin >> z; // ввод в переменную z
}
// ниже - описание ConditionActions, состоящее из заголовка и тела этой функции
void ConditionActions() // заголовок ConditionActions
{ // между { и } - тело ConditionActions
	if (y<a*a) // если y<a*a
		d=z*z*(y+1); // вычисление d способом для y<a*a
	else // иначе
		InternalConditionActions(); // вызов InternalConditionActions
}
// ниже - описание InternalConditionActions, состоящее из заголовка и тела этой функции
void InternalConditionActions() // заголовок InternalConditionActions
{ // между { и } - тело InternalConditionActions
	if (y>b*b) // если y>b*b
		d=z*z*(y+3); // вычисление d способом для y>b*b
	else // иначе
		d=z*z*(y+2); // вычисление d способом для a<=x<=b
}
// ниже - описание FinalActions, состоящее из заголовка и тела этой функции
void FinalActions() // заголовок FinalActions
{ // между { и } - тело FinalActions
	Z=a*a-a*b+b*b; // вычисление Z
	F=sqrt(d*d+Z*Z); // вычисление F
	cout << "F=" << F << endl; // вывод строки "F=", значения F, строки endl
}
```

Различные программисты могут разрабатывать различные функции большой программы и даже различные блоки {...}. При этом руководитель проекта разрабатывает функцию main. Все эти программисты должны согласовывать между собой имена глобальных переменных и их роли. Внутри каждого блока {...} отдельный программист может создавать локальные переменные, не согласовывая их ни с кем. В приведенной выше программе локальных переменных нет.

Альтернативный текст программы без декомпозиции в форме с кодом возврата:

Программу было бы проще написать с учетом двух обстоятельств:

1) Можно было бы ввести a, b, y, z одним оператором

cin >> a >> b >> y >> z;

2) При вычислении d в приведенном выше тексте программы использовались два полных оператора if, причем внутренний был вложен во внешний. Можно было бы использовать вместо них две тернарные операции, причем внутренняя была бы вложена во внешнюю:

d = (y<a*a ? z*z*(y+1) : (y>b*b ? z*z*(y+3) : z*z*(y+2)));

Вот каким бы стал текст нашей программы:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;      // разрешение стандартного пространства имен
// ниже - описание main, состоящее из заголовка и тела этой функции
int main() // заголовок main в стиле с кодом возврата
{ // между { и } - тело main
        setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
        double a, b, y, z, d, Z, F; // создание локальных действительных
переменных
        cout << "Л. р. 4 вар. 0" << endl                    // написали
        << "Введите в следующем порядке" << endl      // оператор cout
        << "значения a, b, y, z:" << endl;                 // на 3 строчках
        cin >> a >> b >> y >> z; // ввод 4 чисел в столбик или в строчку
        d = (y<a*a ? z*z*(y+1) : (y>b*b ? z*z*(y+3) : z*z*(y+2))); // вычисление d
        Z=a*a-a*b+b*b; // вычисление Z
        F=sqrt(d*d+Z*Z); // вычисление F
        cout << "F=" << F << endl; // вывод на экран строки "F=", значения F, endl
        system("pause"); // пауза перед завершением программы
        return 0; // передача Windows нулевого кода возврата
}
```

Работа не альтернативной программы:

```
Л. р. 4 вар. 0
a=2.5 Enter
b=3.4 Enter
y=4.3 Enter
z=5.2 Enter
F=143.614
Для продолжения нажмите любую клавишу... Enter
```

Работа альтернативной программы:

```
Л. р. 4 вар. 0
Введите в следующем порядке
значения a, b, y, z:
2.5   3.4   4.3   5.2 Enter
F=143.614
```

Для продолжения нажмите любую клавишу... Enter
У альтернативной программы данные можно также вводить в столбик.

Л.Р. 5. Декомпозиция разветвляющихся программ (ч. 2)

Вычислить: $W = \begin{cases} P^2 - \min(Q, R, S) & \text{при } P < 0 \\ \pi \max(R, S^2 + 1)\sqrt{S^2 + 1} & \text{иначе} \end{cases}$ Исходные данные: P, Q, R, S.

Можно создать и инициализировать действительную переменную pi: **double pi=3.14159;** лучше создать константу: **const double pi=3.14159;**
Блок-схема: представлена на **Рисунок 14** – **Рисунок 18**. В данной задаче из-за сложности блок-схемы необходима ее декомпозиция. Главной блок-схеме соответствует функция **main**. Пусть заголовки остальных пользовательских функций будут как в таблице 4. Указанные в приведенных блок-схемах константа pi и переменные должны быть должны быть глобальными, т.е. описываться до всех функций, включая main. В приведенной ниже программе локальных переменных нет.

Таблица 4

Имена блок-схем и заголовки пользовательских функций

Имя блок-схемы	**Имя пользовательской функции**
Начальные действия	**void InitialActions ()**
Конечные действия	**void FinalActions ()**
Вычисление W способом 1	**void CalcWmet1 ()**
Вычисление W способом 2	**void CalcWmet2 ()**

Конструкция программы:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;      // разрешение стандартного пространства имен
Создание глобальных констант и переменных
Прототипы пользовательских функций
void main () // заголовок main
{
      setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
      Операторы
}
Описания пользовательских функций
```

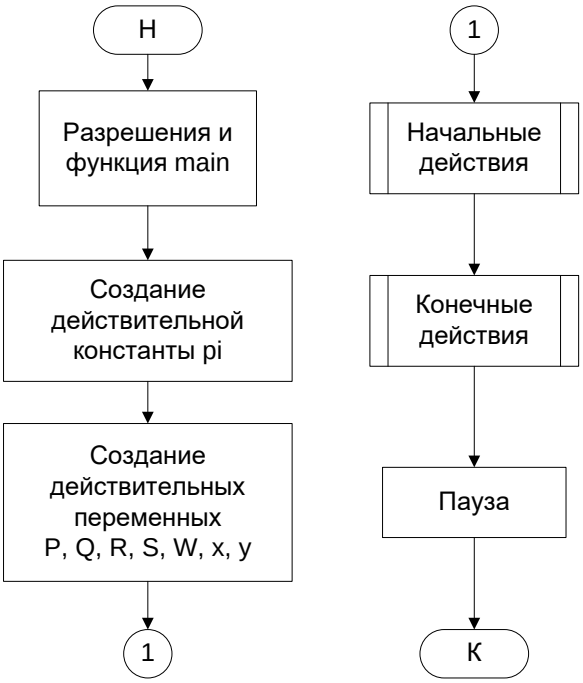


Рисунок 14. Главная бл.-сх.

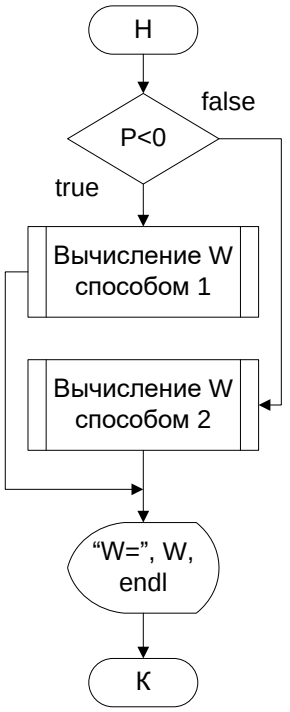


Рисунок 16. Бл.-сх. Конечные действия

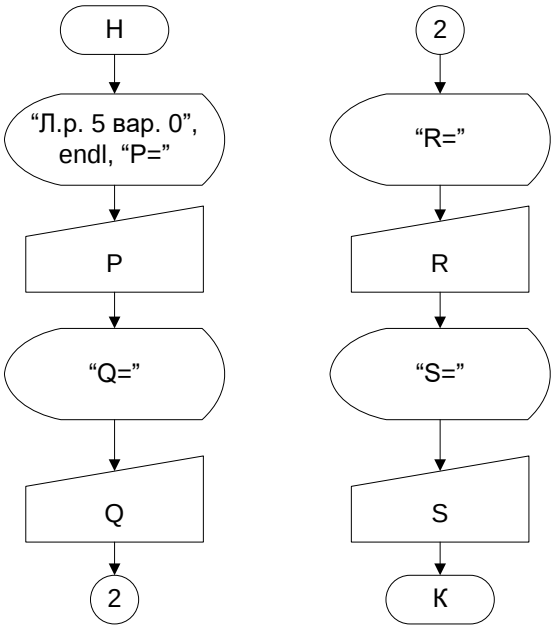


Рисунок 15. Бл.-сх. Начальные действия

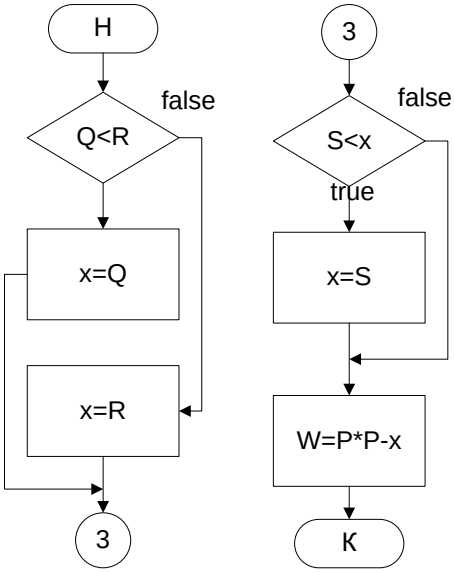


Рисунок 17. Бл.-сх. Вычисление W способом 1

Текст программы с декомпозицией в форме без кода возврата:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;      // разрешение стандартного пространства имен
const double pi=3.14159; // создание глобальных константы
double P, Q, R, S, W, x, y; // и переменных
```

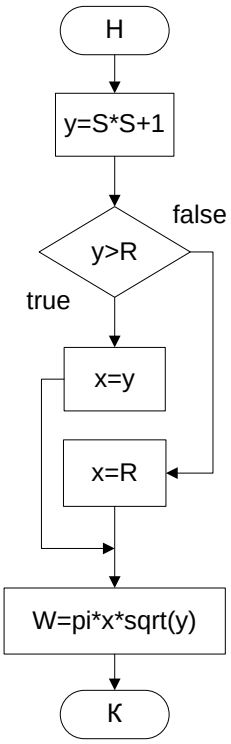


Рисунок 18. Бл.-сх.
Вычисление W способом 2

```
// ниже - прототипы пользовательских функций
void InitialActions();
void FinalActions();
void CalcWmet1();
void CalcWmet2();
// ниже - описание main, состоящее из заголовка и тела этой функции
void main() // заголовок main
{ // между { и } - тело main
        setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
        // ниже - вызовы пользовательских функции, выше - их прототипы
        InitialActions(); // вызов InitialActions
        FinalActions(); // вызов FinalActions
        system("pause"); // пауза перед завершением программы
}
// ниже - описание InitialActions, состоящее из заголовка и тела этой функции
void InitialActions() // заголовок InitialActions
```

```
{ // между { и } - тело InitialActions
        cout << "Л. р. 5 вар. 0" << endl << "P="; /* вывод строк
        " Л. р. 5 вар. 0", endl, "P=" */
        cin >> P; // ввод в P
        cout << "Q="; // вывод строки "Q="
        cin >> Q; // ввод в Q
        cout << "R="; // вывод строки "R="
        cin >> R; // ввод в переменную R
        cout << "S="; // вывод  строки "S="
        cin >> S; // ввод в переменную S
}
// ниже - описание FinalActions, состоящее из заголовка и тела этой функции
void FinalActions() // заголовок FinalActions
{ // между { и } - тело FinalActions
        if (P<0) // если P<0
                CalcWmet1(); // вызов CalcWmet1
        else // иначе
                CalcWmet2(); // вызов CalcWmet2
        cout << "W=" << W << endl; // вывод строки "W=", значения W, строки endl
}
// ниже - описание CalcWmet1, состоящее из заголовка и тела этой функции
void CalcWmet1() // заголовок CalcWmet1
{ // между { и } - тело CalcWmet1
        if (Q<R) x=Q; else x=R; // min(Q, R) -> x
        if (S<x ) x=S; // min(S, x) -> x
        W=P*P-x; // вычисление W при P<0
}
// ниже - описание CalcWmet2, состоящее из заголовка и тела этой функции
void CalcWmet2() // заголовок CalcWmet2
{ // между { и } - тело CalcWmet2
        y=S*S+1;
        if (y>R) x=y; else x=R; // max(R, y) -> x
        W=pi*x*sqrt(y); // вычисление W при P>=0
}
```

Альтернативный текст программы без декомпозиции в форме с кодом возврата:

1) Ввод P, Q, R, S одним оператором: **cin >> P >> Q >> R >> S;**

2) Тернарные операции: оператор **x=(Q<R) ? Q : R;** вместо **if (Q<R) x=Q; else x=R;** оператор **x=(S<x) ? S : x;** вместо **if (S<x) x=S;** оператор **x=(y>R) ? y : R;** вместо **if (y>R) x=y; else x=R;**

Вот каким бы стал текст нашей программы:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
```

```
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;      // разрешение стандартного пространства имен
// ниже - описание main, состоящее из заголовка и тела этой функции
int main() // заголовок main в стиле с кодом возврата
{ // между { и } - тело main
        setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
        const double pi=3.14159; // оператор констант
        double P, Q, R, S, W, x, y; // создание действительных переменных
        cout << "Л. р. 5 вар. 0" << endl                    // написали
        << "Введите в следующем порядке" << endl      // оператор cout
        << "значения P, Q, R, S:" << endl;                 // на 3 строчках
        cin >> P >> Q >> R >> S; // ввод 4 чисел в столбик или в строчку
        if (P<0) // начало внешнего if
        { // блок Вычисление W способом 1 для внешнего if
                x=(Q<R) ? Q : R; // min(Q, R) -> x
                x=(S<x) ? S : x; // min(S, x) -> x
                W=P*P-x; // вычисление W при P<0
        }
        else // иначе внешнего if
        { // блок Вычисление W способом 2 для внешнего if
                y=S*S+1;
                x=(y>R) ? y : R; // max(R, y) -> x
                W=pi*x*sqrt(y); // вычисление W при P>=0
        }
        cout << "W=" << W << endl; // вывод строки "W=", значения W, строки endl
        system("pause"); // пауза перед завершением программы
        return 0; // передача Windows нулевого кода возврата
}
```

Работа не альтернативной программы:

Л. р. 5 вар. 0
P=2.5 Enter
Q=3.4 Enter
R=4.3 Enter
S=5.2 Enter
W=466.462
Для продолжения нажмите любую клавишу... Enter

Работа альтернативной программы:

Л. р. 5 вар. 0
Введите в следующем порядке
значения P, Q, R, S:
2.5 3.4 4.3 5.2 Enter
W=466.462

Для продолжения нажмите любую клавишу... Enter
У альтернативной программы данные можно также вводить в столбик.

Л.Р. 6. Циклические программы (ч. 1)

$$F(x,y,z)=e^{\pi z}\left(\frac{x^2+3xy+y^2}{z^2+2z+5}\right)$$

а) z=2.5 y=-1.2 -0.2≤x≤3.8 с шагом Hx=0.4
б) z=-2.1 x=1.5 -1.1≤y≤1.7 с шагом Hy=0.3
В задании а начальным значением x считаем -0.2. В задании б начальным значением y считаем -1.1. Нужно вывести на экран все вычисленные значения F(x, y, z). В программе нужно представить описание функции F(x, y, z). В этой работе можно создать и инициализировать переменную pi так: **double pi=3.14159;** но лучше использовать оператор констант C++: **const double pi=3.14159;** При этом pi может быть как глобальной, так и локальной внутри тела функции F. Мы выберем последнее. При описании функции F нам потребуется создать параметры x, y, z типа double. Нам потребуется создать глобальные переменные x, y, z, Hx, Hy типа double. Начиная с этой лабораторной работы мы будем писать программы только с кодом возврата, поскольку наш опыт в программировании возрос.
Блок-схема: представлена на **Рисунок 19** – **Рисунок 22**. В данной задаче из-за сложности блок-схемы необходима декомпозиция.

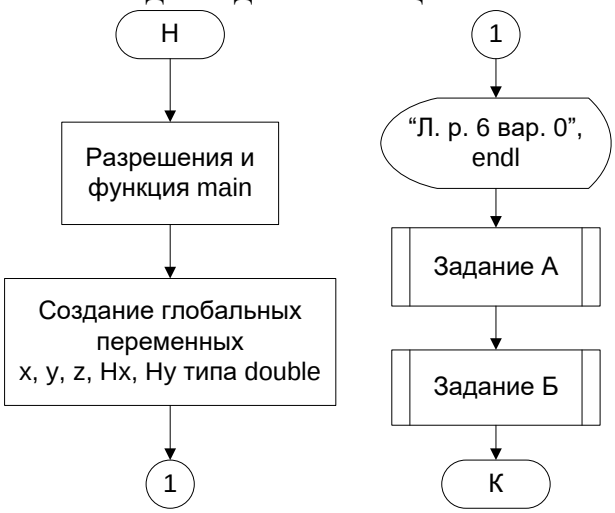


Рисунок 19. Главная бл.-сх.

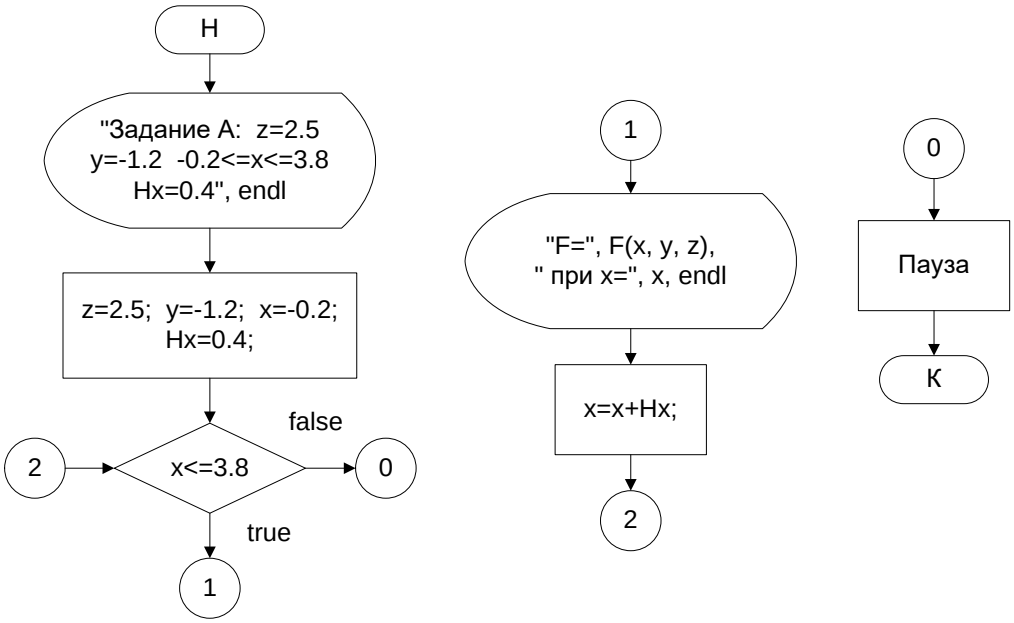


Рисунок 20. Бл.-сх. Задание А

Конструкция программы:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;      // разрешение стандартного пространства имен
```

Создание глобальных переменных

Прототипы пользовательских функций

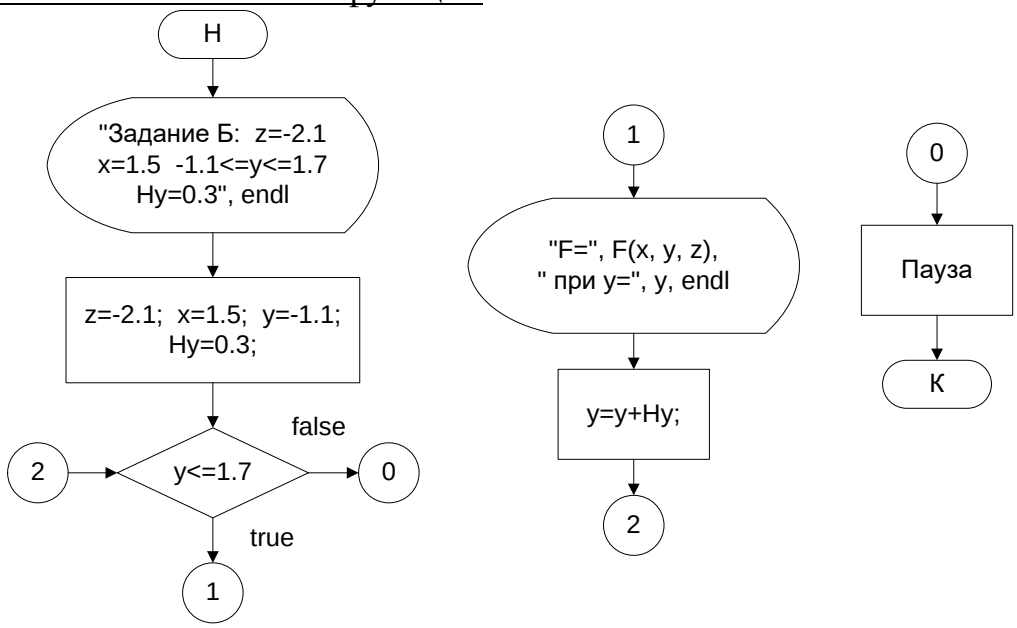


Рисунок 21. Бл.-сх. Задание Б

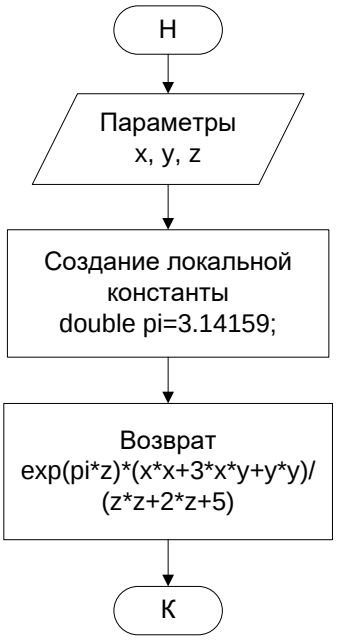


Рисунок 22. Бл.-сх. F(x, y, z)

```
int main () // заголовок main в стиле с кодом возврата
{
        setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
        Операторы
        return 0;
}
```

Описания пользовательских функций

Главной блок-схеме соответствует функция main. Другие блок-схемы и пользовательские функции показаны в таблице 5.

Таблица 5

Имена блок-схем и заголовки пользовательских функций

Имя блок-схемы	**Заголовок пользовательской функции**
Задание А	**void TaskA ()**
Задание Б	**void TaskB ()**
F(x, y, z)	**double F(double x, double y, double z)**

Текст программы с декомпозицией в форме с кодом возврата:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;      // разрешение стандартного пространства имен
double x, y, z, Hx, Hy; // создание глобальных переменных
// ниже - прототипы пользовательских функций
void TaskA();
void TaskB();
double F(double x, double y, double z);
// ниже - описание main, состоящее из заголовка и тела этой функции
int main() // заголовок main в стиле с кодом возврата
```

```
{ // между { и } - тело main
	setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
	cout << "Л. р. 6 вар. 0" << endl; // вывод строк "Л. р. 6 вар. 0", endl
	// ниже - вызовы пользовательских функций
	TaskA(); // вызов TaskA
	TaskB(); // вызов TaskB
	return 0; // возврат нулевого кода возврата в Windows
}
// ниже - описание TaskA, состоящее из заголовка и тела этой функции
void TaskA () // заголовок TaskA
{ // между { и } - тело TaskA
	cout << "Задание А:  z=2.5  y=-1.2  -0.2<=x<=3.8  Hx=0.4" << endl;
	z=2.5;  y=-1.2;  x=-0.2;  Hx=0.4; // присвоение начальных значений
	while (x<=3.8) // условие цикла, пока x<=3.8
	{// блок между { и } - это тело цикла
		cout << "F=" << F(x, y, z) << " при x=" << x << endl;
		x+=Hx; // модификация цикла, эквивалентно x=x+Hx;
	}
	system("pause"); // пауза
}
// ниже - описание TaskB, состоящее из заголовка и тела этой функции
void TaskB () // заголовок TaskB
{ // между { и } - тело TaskB
	cout << "Задание Б:  z=-2.1  x=1.5  -1.1<=y<=1.7  Hy=0.3" << endl;
	z=-2.1;  x=1.5;  y=-1.1;  Hy=0.3; // присвоение начальных значений
	while (y<=1.7) // условие цикла, пока y<=1.7
	{// блок между { и } - это тело цикла
		cout << "F=" << F(x, y, z) << " при y=" << y << endl;
		y+=Hy; // модификация цикла, эквивалентно y=y+Hy;
	}
	system("pause"); // пауза
}
// ниже - описание F состоящее из заголовка и тела этой функции
double F(double x, double y, double z) // заголовок F
{ // между { и } - тело F
	const double pi=3.14159; // создание локальной константы pi
	return exp(pi*z)*(x*x+3*x*y+y*y)/(z*z+2*z+5); // возврат значения F
}
```

Работа программы:

```
Л. р. 6 вар. 0
Задание А:  z=2.5  y=-1.2  -0.2<=x<=3.8  Hx=0.4
F=348.744 при x=-0.2
F=120.475 при x=0.2
```

F=-57.0673 при x=0.6
F=-183.883 при x=1
F=-259.973 при x=1.4
F=-285.336 при x=1.8
F=-259.973 при x=2.2
F=-183.883 при x=2.6
F=-57.0673 при x=3
F=120.475 при x=3.4
F=348.744 при x=3.8
Для продолжения нажмите любую клавишу... Enter
Задание Б: z=-2.1 x=1.5 -1.1<=y<=1.7 Hy=0.3
F=-0.000390086 при y=-1.1
F=-0.00018588 при y=-0.8
F=6.54507e-005 при y=-0.5
F=0.000363906 при y=-0.2
F=0.000709486 при y=0.1
F=0.00110219 при y=0.4
F=0.00154202 при y=0.7
F=0.00202897 при y=1
F=0.00256305 при y=1.3
F=0.00314425 при y=1.6
Для продолжения нажмите любую клавишу... Enter
6.54507e-005 - инженерная форма записи действительного числа, означает **$6.54507*10^{-5}$**

Об альтернативных текстах программы

1) У нас счетчики цикла - это **x** в функции **TaskA** и это **y** в функции **TaskB**. Но **x** и **y** имеют тип **double**, следовательно, управление этими циклами носит приблизительный характер! В **TaskA** в цикле **x** меняется от **-0.2** до **3.8** с шагом **Hx=0.4**. Фактически (см. выше работу программы) цикл выполнился **11 раз** при **x** равном **-0.2, ..., 3.8**, но из-за накапливающихся погрешностей **x** мог бы вместо **3.8** стать **3.80001**. Цикл выполняется, **пока x<=3.8**, из-за **x** равного **3.80001** наш цикл выполнился бы **10 раз**. Если Ваша будущая программа предназначена управлять кораблем, самолетом или считать деньги, то это очень важно.

2) Можно было бы вместо **while** использовать другую конструкцию цикла, например, **for**. Перепишем цикл для **TaskA** с учетом пунктов 1 и 2:

```
for (int k=1; k<=11; k++) // цикл выполнится ровно 11 раз
{ // блок между { и } - это тело цикла, оно не изменилось
        cout << "F=" << F(x, y, z) << " при x=" << x << endl;
        x+=Hx; // модификация x, но он теперь не счетчик цикла
}
```

Л.Р. 7. Циклические программы (ч. 2)

$$F(x,y,z)=(\sin x)\left(\frac{x^2+3xy+y^2}{z^2+2z+5}\right) \quad x = 1...N$$

Найти сумму и максимальный элемент из положительных членов последовательности.

О глобальных переменных нашей программы. Значения переменных **y, z, N будут вводиться**. Эти переменные **x, N - типа int**, переменные **y, z - типа double**. Еще нам понадобятся **переменные типа double**: 1) **Ex** - для вычисления значения текущего элемента последовательности, рассматриваемого нами при конкретном **x**; 2) для получения результатов соответственно **Sum** - для суммы и **Max** - для максимального элемента, их **инициализация Sum=0, Max=-1**.

В программе нужны описания функций с заголовками **double F(int x, double y, double z)**. и **double Max2(double x, double y)** - она будет возвращать максимум из двух чисел - x и y.

Сущность алгоритма: счетчик цикла (переменная x) будет изменяться от 1 до N, на каждом витке цикла: 1) вычисляется **Ex=F(x, y, z);** 2) Если **Ex>0**, то увеличивается сумма **Sum+=Ex;** и обновляется максимум **Max=Max2(Max, Ex);** после окончания цикла **выводятся Sum и Max**. Если окажется, что Max остался -1, значит у последовательности не было положительных членов.

Блок-схема: представлена на **Рисунок 23** – **Рисунок 26**. В данной задаче из-за сложности блок-схемы необходима ее декомпозиция. Блок-схеме **Работа с Ex** будет соответствовать функции с заголовком **void ExWork()**, главной блок схеме - int main (), остальные блок-схемы имеют те же имена, что и соответствующие им функции.

Конструкция программы:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;      // разрешение стандартного пространства имен
Создание глобальных переменных
Прототипы пользовательских функций
int main () // заголовок main в стиле с кодом возврата
{
        setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
        Операторы
        return 0;
}
Описания пользовательских функций
```

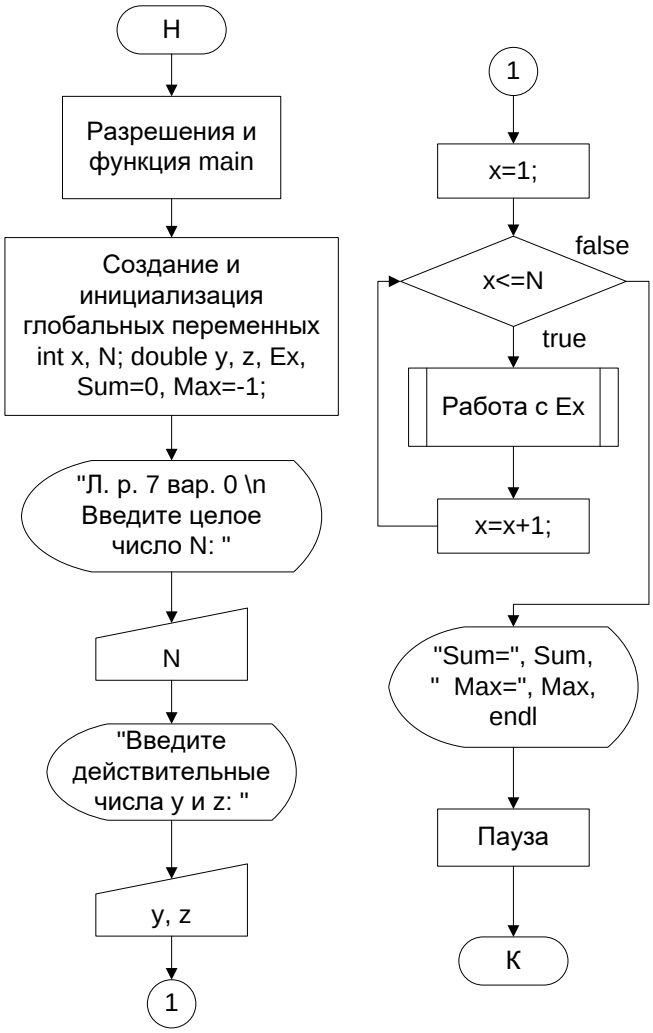


Рисунок 23. Главная бл.-сх.

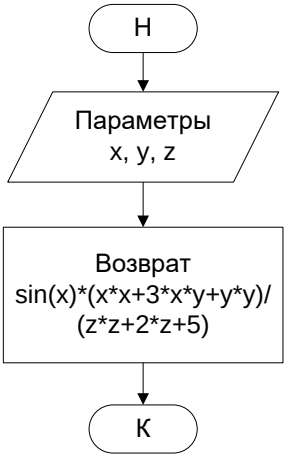


Рисунок 24. Бл.-сх. F(x, y, z)

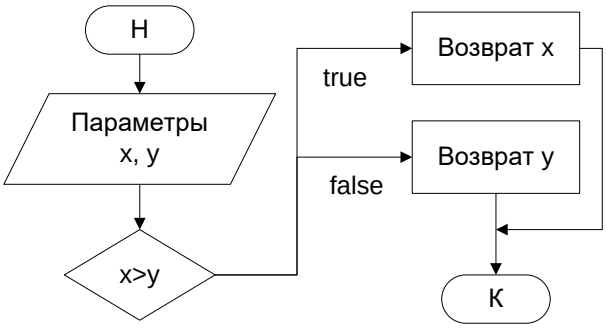


Рисунок 25. Бл.-сх. Max2(x, y)

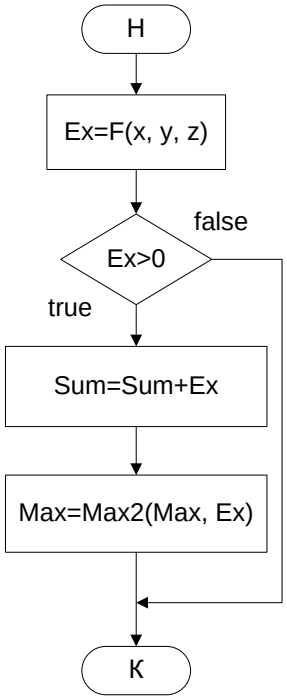


Рисунок 26. Бл.-сх. Работа с Ex

Текст программы с декомпозицией в форме с кодом возврата:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;      // разрешение стандартного пространства имен
// ниже - создание глобальных переменных
int x, N; double y, z, Ex, Sum=0, Max=-1;
// ниже - прототипы пользовательских функций
void ExWork();
double F(double x, double y, double z);
double Max2(double x, double y);
// ниже - описание main, состоящее из заголовка и тела этой функции
int main() // заголовок main в стиле с кодом возврата
{ // между { и } - тело main
```

```
	setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
	cout << "Л. р. 7 вар. 0\nВведите целое число N: "; // вывод строки
	cin >> N; // ввод N
	cout << "Введите действительные числа y и z: ";
	cin >> y >> z; // ввод y и z
	// ниже - вызов пользовательской функции
	for (x=1; x<=N; x++) ExWork();
	cout << "Sum=" << Sum << "  Max=" << Max << endl;
	system("pause"); // пауза
	return 0; // возврат нулевого кода возврата в Windows
}
// ниже - описание ExWork, состоящее из заголовка и тела этой функции
void ExWork () // заголовок ExWork
{ // между { и } - тело ExWork
	// ниже - вызов пользовательских функций
	Ex=F(x, y, z);
	if (Ex>0) { Sum+=Ex; Max=Max2(Max, Ex); }
}
// ниже - описание F состоящее из заголовка и тела этой функции
double F(double x, double y, double z) // заголовок F
{ // между { и } - тело F
	return sin(x)* // один оператор записан на 3 строчках
	(x*x+3*x*y+y*y)/   // пробелы после знака деления / очень важны!!!
	(z*z+2*z+5); // эти 3 строчки - оператор возврата значения F
}
// ниже - описание Max2 состоящее из заголовка и тела этой функции
double Max2(double x, double y)// заголовок Max2
{ // между { и } - тело Max2
	return x>y ? x : y; // оператор возврата значения Max2
}
```

Работа программы:

```
Л. р. 7 вар. 0
Введите целое число N: 18 Enter
Введите действительные числа y и z: 2.1  3.4 Enter
Sum=37.5642  Max=12.2388
Для продолжения нажмите любую клавишу... Enter
```

Об альтернативных текстах программы

Дадим описание функции **double Max2(double x, double y)** без использования тернарной операции.

```
// ниже - описание Max2 состоящее из заголовка и тела этой функции
double Max2(double x, double y) // заголовок Max2
{ // между { и } - тело Max2
	if (x>y) return x; // если x>y, возвращается x
```

```
    else return y; // иначе, возвращается y
}
```

Л.Р. 8. Программы с циклами и ветвлениями

№ п/п	Последовательность			Параметры	
	Общий член a_k	**N**	**Способ обработки**	**Вычисляемые**	**Заданные**
0	$\frac{(-1)^{k+1} e^{-t(k+2)}}{x^k (k+5)!}$	**8**	**Найти сумму по нечетным и произведение по четным номерам**	**I.** $x = \begin{cases} \frac{A^2}{B^2 + C^2} & \text{если } A + B > C \\ 1 & \text{иначе} \end{cases}$ **II. $t=t_0+hi$, i=1...M**	**A=3.1, B=1.5, C=1.7, t_0=0.8, h=0.2, M=7**

Здесь рассматривается конечная последовательность a_1, ..., a_N с общим членом a_k, который зависит от k, x, t. Рассматриваются i=1...M, t зависит от i, следовательно, мы должны рассмотреть в цикле M последовательностей для каждого i.

Кратко о будущей программе:

Вот какие глобальные константы и переменные нам нужны:

```
const int M=7, N=8;
const double A=3.1, B=1.5, C=1.7, t0=0.8, h=0.2;
int i, k;
double x, t, Sum, Prod;
```

Переменные Sum и Prod нужны для вычисления результирующих суммы и произведения соответственно.

Блок-схема: представлена на **Рисунок 27** – **Рисунок 33**. В данной задаче из-за сложности блок-схемы необходима ее декомпозиция. Главной блок-схеме соответствует функция **main**. Заголовки остальных пользовательских функций см. в таблице 6.

Таблица 6

Имена блок-схем и заголовки пользовательских функций

Имя блок-схемы	Заголовок пользовательской функции
Вычисление x	**void Calc_x ()**
Работа с последовательностью	**void SecuWork ()**
Вычисление суммы	**void CalcSum ()**
Вычисление произведения	**void CalcProd ()**
a(k)	**double a(int k)**
fact(k)	**double fact(int k)**

десь используется математическое определение k! (k факториала), как рекурсивной функции - т.е. функции, которая при вычислении вызывает саму себя. Обычное определение k! при k≥0 такое: 0!=1, k!=1*2*...*k при k>0, рекурсивное определение такое:

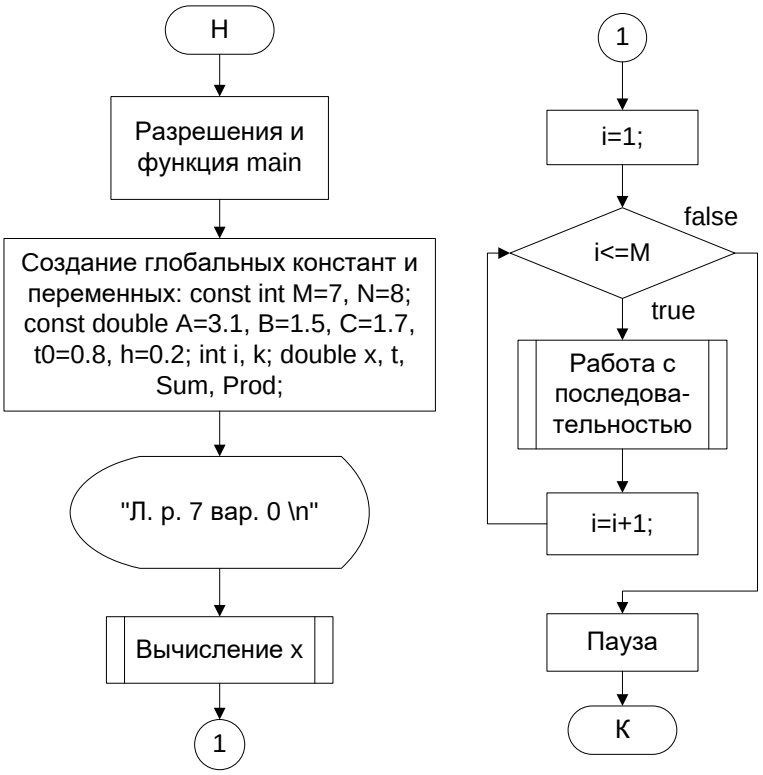


Рисунок 27. Главная бл.-сх.

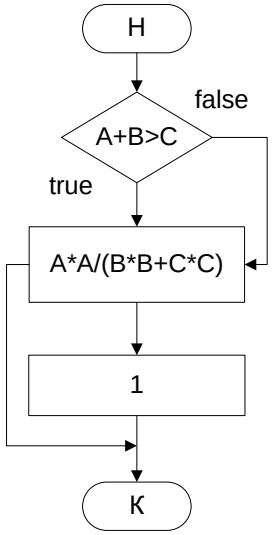


Рисунок 28. Бл.-сх. Вычисление x

$$k!=\begin{cases}1, & \text{если } k=0\\ (k-1)!*k, & \text{иначе}\end{cases}$$

На примере 5! компьютер будет работать по рекурсивному определению так: 5! - это 4!*5, 4! - это 3!*4, 3! - это 2!*3, 2! - это 1!*2, 1! - это 0!*1, 0! - это 1, далее компьютер пройдет по этой цепочке назад и получит, что 5! - это 120.

double fact(int k) - таков будет заголовок соответствующей функции, параметр k - целого типа (int), поскольку в математике k! определен только для целых не отрицательных k.

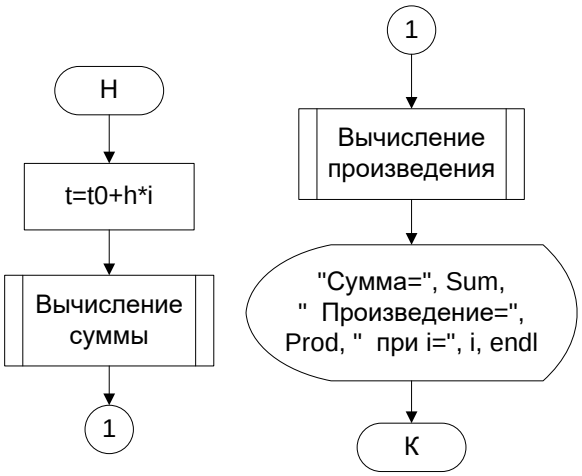


Рисунок 29. Бл.-сх. Работа с последовательностью

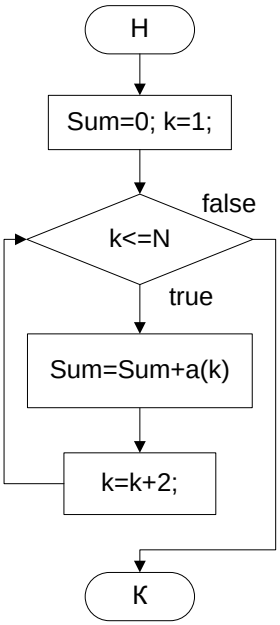


Рисунок 30. Бл.-сх. Вычисление суммы

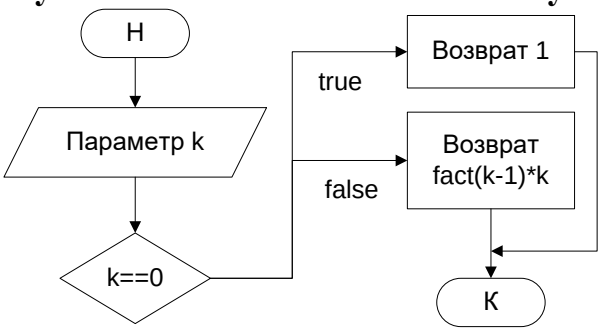


Рисунок 31. Блок-схема fact(k)

Но почему результат функции fact - действительного типа (double)? Диапазон типа int - это приблизительно ± 2 млрд., диапазон типа double - это приблизительно $\pm 10^{308}$, что значительно шире, k! очень быстро растет, диапазон int выдерживает только 12!, диапазон double - 170!, 171! уже находится вне диапазона double. Мы поставили double как тип результата для большей

универсальности в применении функции fact в будущем, в данной работе хватило бы int.

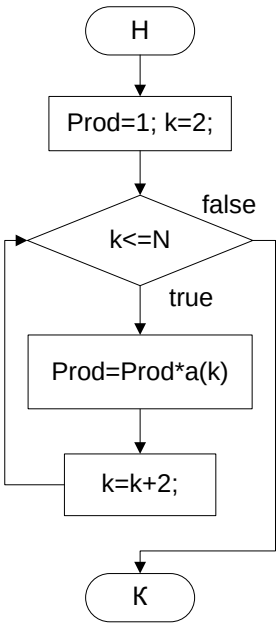


Рисунок 32. Блок-схема Вычисление произведения

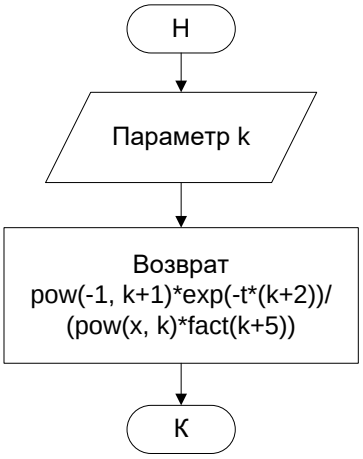


Рисунок 33. Бл.-сх. a(k)

Конструкция программы:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;      // разрешение стандартного пространства имен
Создание глобальных констант и переменных
Прототипы пользовательских функций
int main () // заголовок main в стиле с кодом возврата
{
        setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
        Операторы
        return 0;
```

```
}
```

Описания пользовательских функций

Текст программы с декомпозицией в форме с кодом возврата:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;      // разрешение стандартного пространства имен
// ниже - создание глобальных констант и переменных
const int M=7, N=8;
const double A=3.1, B=1.5, C=1.7, t0=0.8, h=0.2;
int i, k;
double x, t, Sum, Prod;
// ниже - прототипы пользовательских функций
void Calc_x ();
void SecuWork ();
void CalcSum ();
void CalcProd ();
double a(int k);
double fact(int k);
// ниже - описание main, состоящее из заголовка и тела этой функции
int main() // заголовок main в стиле с кодом возврата
{ // между { и } - тело main
        setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
        cout << "Л. р. 8 вар. 0\n"; // вывод строки
        Calc_x ();
        for (i=1; i<=M; i++) SecuWork ();
        system("pause"); // пауза
        return 0; // возврат нулевого кода возврата в Windows
}
// ниже - описание Calc_x, состоящее из заголовка и тела этой функции
void Calc_x () // заголовок Calc_x
{ // между { и } - тело Calc_x
        x = A+B>C ? A*A/(B*B+C*C) : 1;
}
// ниже - описание SecuWork, состоящее из заголовка и тела этой функции
void SecuWork () // заголовок SecuWork
{ // между { и } - тело SecuWork
        t=t0+h*i;
        CalcSum();
        CalcProd();
        cout   << "Сумма=" << Sum << "  Произведение=" << Prod
               << "  при i=" << i << endl;
```

```
}
// ниже - описание CalcSum, состоящее из заголовка и тела этой функции
void CalcSum () // заголовок CalcSum
{ // между { и } - тело CalcSum
        Sum=0;
        for (k=1; k<=N; k+=2)  Sum +=a(k);
}
// ниже - описание CalcProd, состоящее из заголовка и тела этой функции
void CalcProd () // заголовок CalcProd
{ // между { и } - тело CalcProd
        Prod=1;
        for (k=2; k<=N; k+=2)  Prod *=(a(k));
}
// ниже - описание a состоящее из заголовка и тела этой функции
double a(int k) // заголовок a
{ // между { и } - тело a
        return pow(-1, k+1)*exp(-t*(k+2))/(pow(x, k)*fact(k+5));
}
// ниже - описание fact состоящее из заголовка и тела этой функции
double fact(int k) // заголовок fact
{ // между { и } - тело fact
        return k==0 ? 1 : fact(k-1)*k;
}
```

Работа программы:

```
Л. р. 8 вар. 0
Сумма=3.70104e-005  Произведение=5.58367e-045  при i=1
Сумма=2.03071e-005  Произведение=2.06476e-047  при i=2
Сумма=1.11431e-005  Произведение=7.63522e-050  при i=3
Сумма=6.11483e-006  Произведение=2.82340e-052  при i=4
Сумма=3.35566e-006  Произведение=1.04405e-054  при i=5
Сумма=1.84154e-006  Произведение=3.86077e-057  при i=6
Сумма=1.01063e-006  Произведение=1.42766e-059  при i=7
Для продолжения нажмите любую клавишу... Enter
```

Об альтернативных текстах программы

Дадим описание функции **double fact (int k)**, исходя из обычного математического определения факториала 0!=1, k!=1*2*...*k при k>0.

```
// ниже - описание fact состоящее из заголовка и тела этой функции
double fact(int k) // заголовок fact
{ // блок между { и } - тело fact
        double f=1;  // создание локальных
        int i;           // вспомогательных переменных
        if (k==0) return 1; // если k равно 0, возвращается 1
        else // иначе
```

```
    { // оператор - блок между { и }
        for ( i=1; i<=k; i++ ) f*=i; // в f накапливается произведение 1*2*...k
        return f; // возвращается значение f
    }
}
```

Заметим, что, если вместо **if (k==0)** написать **if (k=0)**, то k будет присвоен 0 и общий результат этого оператора будет 0, т.е. false, следовательно, C++ это ошибкой не сочтет, программа будет работать, но не так, как нам нужно. Это пример семантической (смысловой) ошибки, которую очень трудно заметить!

В США в начале 60-х годов XX века был взрыв баллистической ракеты на старте с многочисленными человеческими жертвами из-за примерно такой же семантической ошибки в управляющей программе, написанной на языке Fortran. С тех пор появилось понятие НАДЕЖНОСТИ программирования. C, C++, Fortan - ненадежные языки, надежные - Pascal, Delphi, C#, Java, JS (Java Script), VB (Visual Basic), VBA (Visual Basic for Application). Надежные языки обычно имеют более жесткий синтаксис (жесткие правила), в результате большинство семантических ошибок становятся синтаксическими и автоматически обнаруживаются компьютером. Однако жесткие правила некомфортны для программиста. Слишком сложные, запутанные правила языка с многочисленным количеством исключений из каждого правила тоже не способствуют надежности. В идеале оптимальным является простой и жесткий синтаксис. Pascal, Delphi имеют малую популярность из-за слишком жестких и сложных правил. Исходя из всех плюсов и минусов, по популярности в настоящее время первое место делят Java и VB, на втором стоит C# - у них у всех как раз простой и жесткий синтаксис. Fortan с простым и мягким синтаксисом более всех популярен для математического моделирования, C++ со сложным и мягким синтаксисом очень любят системные программисты.

Л.Р. 9. Применение одномерных массивов

Найти сумму и минимальный элемент из отрицательных элементов массива A, состоящего из 10 элементов.

О глобальных переменных нашей программы. Пусть одномерный массив A будет таким:

double A[10]={-0.3, 3.2, -2.7, -2.3, 7.1, -6.7, 4.2, -2.8, 8.1, -1.2}; // массив из чисел типа double

Его 10 элементов A[0], A[1], A[2], ..., A[9] - это, соответственно, числа: -0.3, 3.2, -2.7, ..., -1.2 Сумма отрицательных элементов - это (-0.3)+(-2.7)+(-2.3)+(-6.7)+(-2.8)+(-1.2)=(-16) Минимальный из отрицательных элементов - это (-6.7) Такие у нас будут ответы.

int k // переменная для индексов массива

double Sum=0, Min=0 // переменные для подсчета суммы и минимума

Все эти переменные могли бы быть и локальными для тела main.

Также нужно описание функции с заголовком **double Min2(double x, double y)** Она будет возвращать минимум из двух чисел x и y.

Сущность алгоритма: Счетчик цикла k будет изменяться от 0 до 9. На каждом витке цикла: Если **A[k]>0**, то он добавляется к сумме **Sum+=A[k];** и обновляется минимум **Min=Min2(Min, A[k]);** после окончания цикла **выводятся Sum и Min**. Если бы массив A содержал другие числа и оказалось бы, что Min остался нулем, значит в массиве A не было ни одного отрицательного элемента.
Блок-схема: представлена на **Рисунок 34** и **Рисунок 35**.

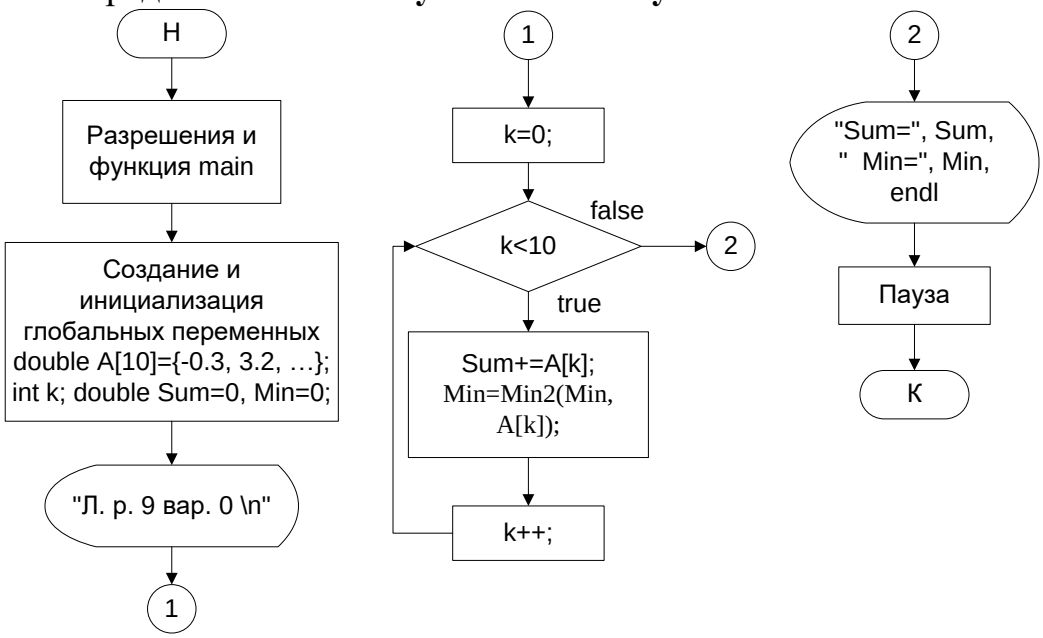


Рисунок 34. Главная бл.-сх.

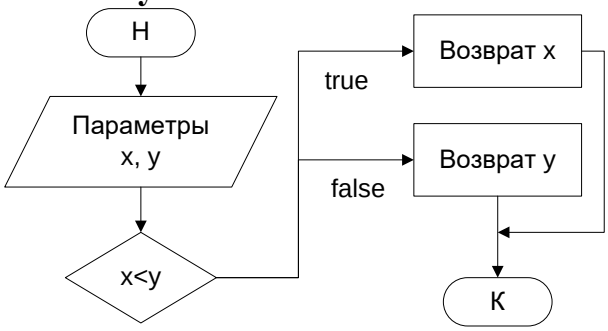


Рисунок 35. Бл.-сх. Min2(x, y)

Конструкция программы:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;      // разрешение стандартного пространства имен
```

Создание глобальных переменных

Прототип пользовательской функции Min2

```
int main () // заголовок main в стиле с кодом возврата
{
        setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
```

Операторы

```
        return 0;
}
```

Описание пользовательской функции Min2

Текст программы:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;      // разрешение стандартного пространства имен
// ниже - создание глобальных переменных
double A[10]={-0.3, 3.2, -2.7, -2.3, 7.1, -6.7, 4.2, -2.8, 8.1, -1.2};
int k; // переменная для индексов массива
double Sum=0, Min=0; // переменные для подсчета суммы и минимума
// ниже - прототип пользовательской функции Min2
double Min2(double x, double y);
// ниже - описание main, состоящее из заголовка и тела этой функции
int main() // заголовок main в стиле с кодом возврата
{ // между { и } - тело main
        setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
        cout << "Л. р. 9 вар. 0\n"; // Вывод заголовка работы
// Операторы
        for (k=0; k<10; k++) if (A[k]<0) { Sum+=A[k]; Min=Min2(Min, A[k]);}
        cout << "Sum=" << Sum << "  Min=" << Min << endl;
        system("pause"); // пауза
        return 0; // возврат нулевого кода возврата в Windows
}
// ниже - описание Min2 состоящее из заголовка и тела этой функции
double Min2(double x, double y) // заголовок Min2
{ // между { и } - тело Min2
        return x<y ? x : y; // оператор возврата значения Min2
}
```

Работа программы:

```
Л. р. 9 вар. 0
Sum=-16  Min=-6.7
Для продолжения нажмите любую клавишу... Enter
```

Об альтернативных текстах программы

Создадим массив A другой командой - без инициализации:

```
double A[10]; // массив из чисел типа double
```

Вместо заполнения числами массива A с помощью инициализации, можно в цикле для k=0, 1, 2, ..., 9 ввести соответствующее число в ячейку A[k]

```
cout << "Введите 10 действительных чисел:\n";
for (k=0; k<10; k++) cin >> A[k];
```

Эти 2 строки нужно вставить в программу между строками с комментариями **Вывод заголовка работы** и **Операторы**.
Тогда работа программы была бы немного другой:
Л. р. 9 вар. 0
Введите 10 действительных чисел:
-0.3 3.2 -2.7 -2.3 7.1 Enter
-6.7 4.2 -2.8 8.1 -1.2 Enter
Sum=-16 Min=-6.7
Для продолжения нажмите любую клавишу ... Enter
Между числами необходимо вводить или пробелы, или табуляции, или Enter, после последнего числа - только Enter. В данном случае мы могли бы вводить в массив A и другие числа.

Л.Р. 10. Применение двумерных массивов

Заданы матрицы A и B одинаковой структуры (3 строки × 3 столбца). Вычислить и вывести матрицу C той же структуры, если C=2*A+3*B. По правилам линейной алгебры $c_{i,j}=2*a_{i,j}+3*b_{i,j}$ для любой строки i и столбца j.

$$A=\begin{bmatrix}1.0 & 2.0 & 3.0\\ 4.0 & 5.0 & 4.0\\ 3.0 & 2.0 & 1.0\end{bmatrix} \quad B=\begin{bmatrix}0.5 & 0.4 & 0.3\\ 0.2 & 0.1 & 0.2\\ 0.3 & 0.4 & 0.5\end{bmatrix}$$

О глобальных переменных:. Нужны двумерные массивы A, B. C:
double A[3][3]={{1.0, 2.0, 3.0}, {4.0, 5.0, 4.0}, {3.0, 2.0, 1.0}};
double B[3][3]={{0.5, 0.4, 0.3}, {0.2, 0.1, 0.2}, {0.3, 0.4, 0.5}};
double C[3][3];
Номера строк i= 0, 1, 2, номера столбцов j= 0, 1, 2. Для строки i=0 и столбца j=1, например, A[0][1]=2.0, B[0][1]=0.4, следовательно, C[0][1]=2*2.0+3*0.4=5.2
int i, j // для номеров строк и столбцов
Сущность алгоритма: Внешний цикл по i= 0, 1, 2 будет работать в целом с i-ми строками A, B, C. Внутренний цикл по j= 0, 1, 2 будет работать с j-ми элементами A, B, C из i-й строки, он будет: вычислять C[i][j]=2*A[i][j]+3*B[i][j]; выводить C[i][j]; если j=3, то выводить endl (конец строки), иначе выводить "\t" (табуляцию).
Блок-схема: представлена на **Рисунок 36** – **Рисунок 38**. Блок-схемы **Работа с i-ми строками A, B, C** и **Вывод после числа** реализованы функциям с заголовками **void WorkRiABC()** и **void CoutAfterNumber()** соответственно.

Конструкция программы:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;      // разрешение стандартного пространства имен
```

Создание глобальных переменных
Прототипы пользовательских функций

```
int main () // заголовок main в стиле с кодом возврата
{
        setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
        Операторы
        return 0;
}
```

Описание пользовательских функций

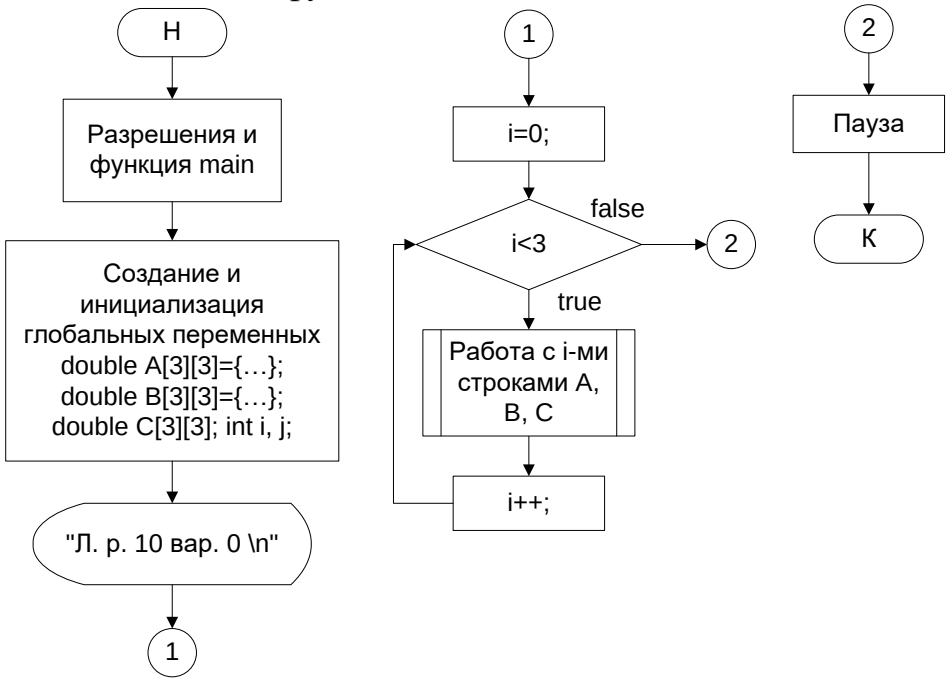


Рисунок 36. Главная бл.-сх.

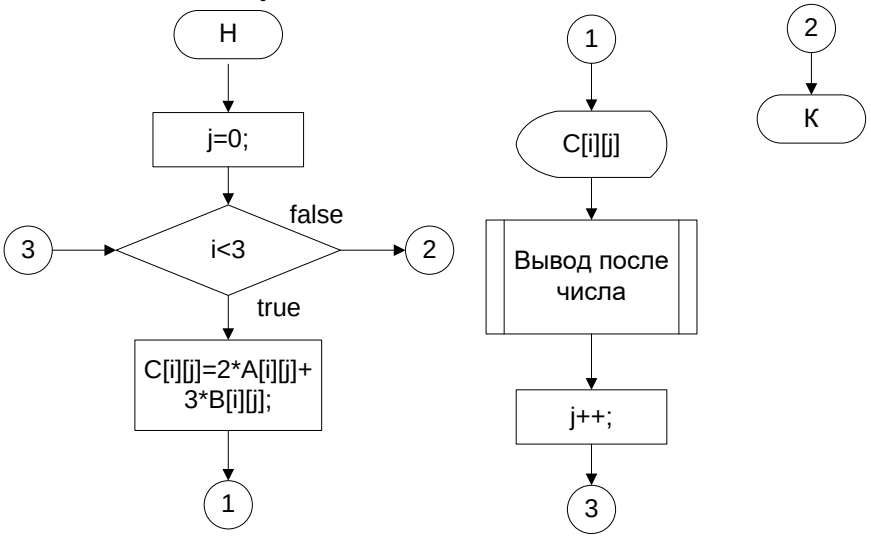


Рисунок 37. Бл.-сх. Работа с i-ми строками A, B, C

Текст программы с декомпозицией:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
```

```
using namespace std;      // разрешение стандартного пространства имен
// ниже - создание глобальных переменных
double A[3][3]={{1.0, 2.0, 3.0}, {4.0, 5.0, 4.0}, {3.0, 2.0, 1.0}};
double B[3][3]={{0.5, 0.4, 0.3}, {0.2, 0.1, 0.2}, {0.3, 0.4, 0.5}};
double C[3][3];
int i, j; // для номеров строк и столбцов
```

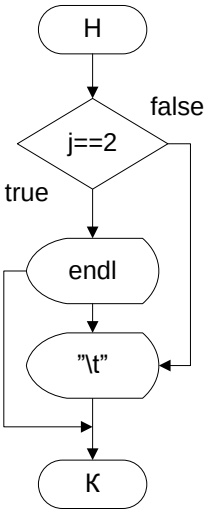


Рисунок 38. Бл.-сх. Вывод после числа

```
// ниже - прототип пользовательской функции WorkRiABC
void WorkRiABC();
// ниже - прототип пользовательской функции CoutAfterNumber
void CoutAfterNumber();
// ниже - описание main, состоящее из заголовка и тела этой функции
int main() // заголовок main в стиле с кодом возврата
{ // между { и } - тело main
	setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
	cout << "Л. р. 10 вар. 0\n"; // Вывод заголовка работы
	for (i=0; i<3; i++) WorkRiABC(); // Операторы
	system("pause"); // пауза
	return 0; // возврат нулевого кода возврата в Windows
}
// ниже - описание WorkRiABC состоящее из заголовка и тела этой функции
void WorkRiABC() // заголовок WorkRiABC
{ // между { и } - тело WorkRiABC
	for (j=0; j<3; j++)
	{
		C[i][j]=2*A[i][j]+3*B[i][j];
		cout << C[i][j];
		CoutAfterNumber();
	}
}
```

```
// ниже - описание CoutAfterNumber состоящее из заголовка и тела этой функции
void CoutAfterNumber() // заголовок WorkRiABC
{ // между { и } - тело CoutAfterNumber
        if (j==2) cout << endl; else cout << "\t";
}
```

Работа программы:

```
Л. р. 10 вар. 0
3.5     5.2     6.9
8.6     10.3    8.6
6.9     5.2     3.5
Для продолжения нажмите любую клавишу... Enter
```

Текст программы без декомпозиции:

```
#include <iostream>       // разрешение ввода и вывода
#include <locale>  // разрешение национальных алфавитов
#include <cmath> // разрешение математических функций
#include <string>  // разрешение  работы со строками
using namespace std;      // разрешение стандартного пространства имен
// ниже - создание глобальных переменных
double A[3][3]={{1.0, 2.0, 3.0}, {4.0, 5.0, 4.0}, {3.0, 2.0, 1.0}};
double B[3][3]={{0.5, 0.4, 0.3}, {0.2, 0.1, 0.2}, {0.3, 0.4, 0.5}};
double C[3][3];
int i, j; // для номеров строк и столбцов
// ниже - описание main, состоящее из заголовка и тела этой функции
int main() // заголовок main в стиле с кодом возврата
{ // между { и } - тело main
        setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
        cout << "Л. р. 10 вар. 0\n"; // Вывод заголовка работы
        // ниже приведены Операторы
        for (i=0; i<3; i++) for (j=0; j<3; j++)
        {
                C[i][j]=2*A[i][j]+3*B[i][j];
                cout << C[i][j];
                if (j == 2) cout << endl; else cout << "\t";
        }
        system("pause"); // пауза
        return 0; // возврат нулевого кода возврата в Windows
}
```

Программа получилась более лаконичная и менее понятная, ее работа точно такая же.

Заключение

В C++ предусмотрены также трехмерные, четырехмерные и т.п. многомерные массивы. Вот пример оператора, который создает трехмерный массив **y** - массив, состоящий из 3 массивов, каждый из которых тоже состоит из 4 массивов, каждый из которых состоит из 5 переменных типа **double**:
double y[3][4][5]; Данная книга не содержит полного описания языка C++. Она ориентирована только на основы C++ в курсе информатики. Приведенный ниже список рекомендуемой литературы предназначен тем, кого интересует тематика современного программирования и информатики. Пособие [1] посвящено заданиям по вариантам к лабораторным работам по C++, о которых говорилось выше. Пособия [2; 3] содержат более полное описание C++. Пособие [4] посвящено информатике в целом, а пособия [5-8] - современным технологиям прогаммирования: пособие [5] - сценарному программированию; пособия [6-8] - визуальному программированию на различных языках.

Список рекомендуемой литературы

1. Журавлев А.Е. Информатика: Методические указания по выполнению лабораторных работ - СПб.: ГУМРФ, 2017. - 19 с.
2. Васильев А.Н. Программирование на C++ в примерах и задачах: Учебное пособие - М.: ЭКСМО, 2016. – 368 с.
3. Павловская Т.А., Щупак Ю.А. C/C++. Структурное и объектно-ориентированное программирование: Практикум: Учебное пособие - СПб.: Питер, 2011. - 352 с.:
4. Николаева Н.А., Балса А.Р. Информатика: Учебное пособие - М.: ЭКСМО, 2016. – 368 с.
5. Слепцова Л.Д. Программирование на VBA в Microsoft Office 2010/ - М.: ООО «И.Д. Вильямс», 2010 – 432с.
6. Зиборов В.В. MS Visual Basic 2012 на примерах. - СПб.: БХВ-Петербург, 2013. - 443с.
7. Зиборов В.В. MS Visual C# 2012 на примерах. - СПб.: БХВ-Петербург, 2013. - 480с.
8. Зиборов В.В. MS Visual C++ 2010 в среде .NET. Библиотека программиста. — СПб.: Питер, 2012. — 320 с

ISBN 978-1-312-08735-4

Усл. печ. л. 2.5.
Объем издания 2.0 MB
Оформление электронного издания:
НОО Профессиональная наука, mail@scipro.ru
Дата размещения: 30.08.2021 г.
URL: http://scipro.ru/conf/computerscience.pdf