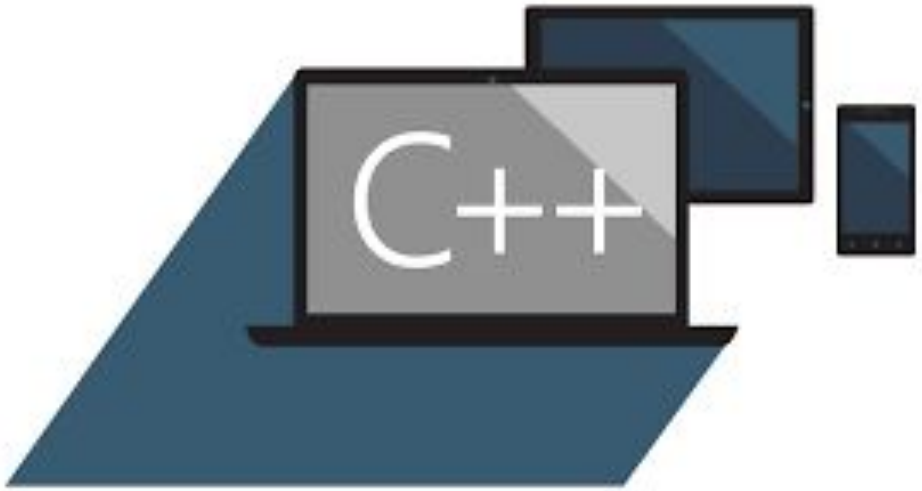
Структура программы на языке C++



Сама по себе программа на языке C++ представляет собой текстовый файл, в котором представлены конструкции и операторы данного языка в заданном программистом порядке.

- Программа
- Машинный язык
- Язык ассемблера
- Высокоуровневые языки программирования

Компилятор – это программа, которая читает код и создаёт автономную (способную работать независимо от другого аппаратного или программного обеспечения) исполняемую программу, которую процессор понимает напрямую. При запуске программы весь код компилируется целиком, а затем создаётся исполняемый файл и уже при повторном запуске программы компиляция не выполняется.

Интерпретатор — это программа, которая напрямую выполняет код, без его предыдущей компиляции в исполняемый файл. Интерпретаторы более гибкие, но менее эффективны, так как процесс интерпретации выполняется повторно при каждом запуске программы.

- **Преимущества высокоуровневых языков программирования**
- 1. Легче писать/читать код.
- 2. Требуется меньше инструкций для выполнения определенного задания.
- 3. Вы не должны заботиться о таких деталях, как загрузка переменных в регистры процессора.
- 4. Высокоуровневые языки программирования портативнее под различные архитектуры

Прежде чем приступить к написанию программ, необходимо изучить структуру программ на языке программирования C++.

Структура программ это разметка рабочей области (области кода) с целью чёткого определения основных блоков программ и синтаксиса.

Структура программ несколько отличается в зависимости от среды программирования.

Мы ориентируемся на IDE Microsoft Visual Studio

C и C++

- C
- 1972 году Деннисом Ритчи в *Bell Telephone Laboratories*
- C++
- разработан Бьёрном Страуструпом в *Bell Labs* в качестве дополнения к C в 1979 г.

Введение в разработку программного обеспечения

- **Схема разработки ПО** (программного обеспечения):

- **Шаг №1: Определите проблему, которую хотели бы решить**
- чётко сформулировать идею

Шаг №2: Определитесь, как вы собираетесь решить эту проблему

- **хорошие решения имеют следующие характеристики**:
- простота;
- хорошая документация (с инструкциями и комментариями);
- модульный принцип: любая часть программы может быть повторно использована или изменена позже, не затрагивая другие части кода;
- надёжность: соответствующая обработка ошибок и экстренных ситуаций.

Режимы конфигурации «Debug» и «Release»

- **Конфигурация сборки** (англ. «*build configuration*») — это набор настроек проекта, которые определяют принцип его построения. Конфигурация сборки состоит из: имени исполняемого файла, директории исполняемого файла, в каких директориях IDE будет искать код и заголовочные файлы, информации об отладке и параметров оптимизации вашего проекта.

- Интегрированная среда разработки имеет две конфигурации сборки: «Release» (Релиз) и «Debug» (Дебаг/Отладка).
- **Конфигурация «Debug»** предназначена для отладки вашей программы. по умолчанию.
- **Конфигурация «Release»** используется для сборки программы для её дальнейшего выпуска.

Стейтменты

- **Стейтмент** (англ. **«statement»**) — это наиболее распространённый тип инструкций в программах. Это и есть та самая инструкция, наименьшая независимая единица в языке C++.
- **Все стейтменты в C++ заканчиваются точкой с запятой.**

```
1 int x;
2 x = 5;
3 std::cout << x;
```

x = 5 — это стейтмент присваивания (англ. «assignment statement»).

int x — это стейтмент объявления (англ. «statement declaration»).

std::cout << x; — это стейтмент вывода (англ. «output statement»).

Комментарии

- **Комментарий** — это строка (или несколько строк) текста, которая вставляется в исходный код для объяснения того, что делает этот код.
- **Однострочные комментарии** — //
- **Многострочные комментарии /* */**

Как правильно писать комментарии?

- на уровне библиотек/программ/функций комментарии отвечают на вопрос **«ЧТО?»**:

```
1 // Эта программа вычисляет оценку студента за семестр на основе его оценок за модули
2
3 // Эта функция использует метод Ньютона для вычисления корня функции
4
5 // Следующий код генерирует случайное число
```

Как правильно писать комментарии?

- внутри библиотек/программ/функций комментарии отвечают на вопрос **«КАК?»**: «Как код выполняет задание?».

```
/* Для расчёта итоговой оценки студента, мы добавляем все его оценки за уроки и домашние задания,
   а затем делим получившееся число на общее количество оценок.
   Таким образом мы получаем средний балл студента. */
```

Как правильно писать комментарии?

- на уровне стейтментов (однострочного кода) комментарии отвечают на вопрос **«ПОЧЕМУ?»**:«Почему код выполняет задание именно таким образом, а не другим?».
- Плохой комментарий на уровне стейтментов объясняет, что делает код.

Комментирование кода

- **Причина №1**: Вы работаете над новой частью кода, которая пока что не рабочая, но вам нужно запустить программу.
- **Причина №2**: Вы написали код, который компилируется, но работает не так как нужно и сейчас у вас нет времени с этим разбираться.
- **Причина №3**: Поиск корня ошибки. Если программа производит не те результаты (или вообще происходит сбой), полезно будет поочерёдно «отключать» части вашего кода, чтобы понять какие из них рабочие, а какие создают проблемы.
- **Причина №4**: Тестирование нового кода.

Структура программ для Microsoft Visual Studio

```
1. // struct_program.cpp: определяет точку входа для консольного приложения.
2. #include "stdafx.h"
3.         //здесь подключаем все необходимые препроцессорные директивы
4. int main() {       // начало главной функции с именем main
5.          //здесь будет находится ваш программный код
6. }
```

В строке 1 говорится о точке входа для консольного приложения, это значит, что данную программу можно запустить через командную строку Windows указав имя программы, к примеру, такое **struct_program.cpp.**

Строка 1 является однострочным комментарием, так как начинается с символов //

Структура программ для Microsoft Visual Studio

1. **// struct_program.cpp:** определяет точку входа для консольного приложения.
2. **#include "stdafx.h"**
3. //здесь подключаем все необходимые препроцессорные директивы
4. **int main()** { // начало главной функции с именем main
5. //здесь будет находится ваш программный код
6. **}**

В строке 2 подключен заголовочный файл **"stdafx.h".**

Данный файл похож на контейнер, так как в нем подключены основные препроцессорные директивы (те, что подключил компилятор, при создании консольного приложения), тут же могут быть подключены и вспомогательные (подключенные программистом).

include — директива препроцессора, т. е. сообщение препроцессору.

Строки, начинающиеся с символа **#** обрабатываются препроцессором до компиляции программы.

Структура программ для Microsoft Visual Studio

```
1. // struct_program.cpp: определяет точку входа для консольного приложения.
2. #include "stdafx.h"
3. //здесь подключаем все необходимые препроцессорные директивы
4. int main() { // начало главной функции с именем main
5. //здесь будет находится ваш программный код
6. }
```

С 4-й по 6-ю строки объявлена функция **main**.

Строка 4 – это заголовок функции, который состоит из типа возвращаемых данных (в данном случае int), этой функцией, и имени функции, а также круглых скобок, в которых объявляются параметры функции.

int — целочисленный тип данных

Между фигурными скобочками размещается основной программный код, называемый еще телом функции. Это самая простая структура программы.

Программа на языке C++ состоит из:

1. директив препроцессора,
2. указаний компилятору,
3. объявлений переменных и/или констант,
4. объявлений и определений функций.

Препроцессор *— это компьютерная программа, принимающая данные на входе и выдающая данные, предназначенные для входа другой программы (например, компилятора).*

Структура программы на С++

#include <имя библиотеки 1> #include <имя библиотеки 2>	**Заголовочные файлы (подключение библиотек)**
// прототипы функций (заголовки)	**Объявление функций**
// глобальные идентификаторы (типы, переменные и т.д.)	**Объявление глобальных идентификаторов**
int main() { *// описание переменных* *// раздел операторов* **}**	**Главная функция программы**
// реализация функций	**Реализация объявленных функций**

Структура программы на С++

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

 int main( )
 {
 тело функции
 }
```

Раздел подключений библиотек

Директивы препроцессору

Раздел главной функции программы

Директива препроцессора – это инструкция, которая включает в текст программы файл, содержащий описание множества функций, что позволяет правильно компилировать программу.

Это важно

- все директивы препроцессора начинаются со знака **#**;
- после директивы препроцессора точка с запятой **не ставится**.

Синтаксис подключения заголовочных файлов:

Директива **#include** позволяет включать в текст программы указанный файл.

Имя файла может быть указано двумя способами:

```
#include <some_file.h>
#include "my_file.h”
```

- Если файл является стандартной библиотекой и находится в папке компилятора, он заключается в угловые скобки **<>**.
- Если файл находится в текущем каталоге проекта, он указывается в кавычках **""**.

Синтаксис подключения заголовочных файлов:

#include <имя заголовочного файла>

Более старые заголовочные файлы подключаются так (этот стиль подключения библиотек унаследован у языка программирования С):

#include <имя заголовочного файла.h>

Различие состоит в том, что после имени ставится расширение **.h.**

Заголовочные файлы

Стандартная Библиотека — коллекция классов и функций, написанных на базовом языке.

Основные заголовочные файлы:

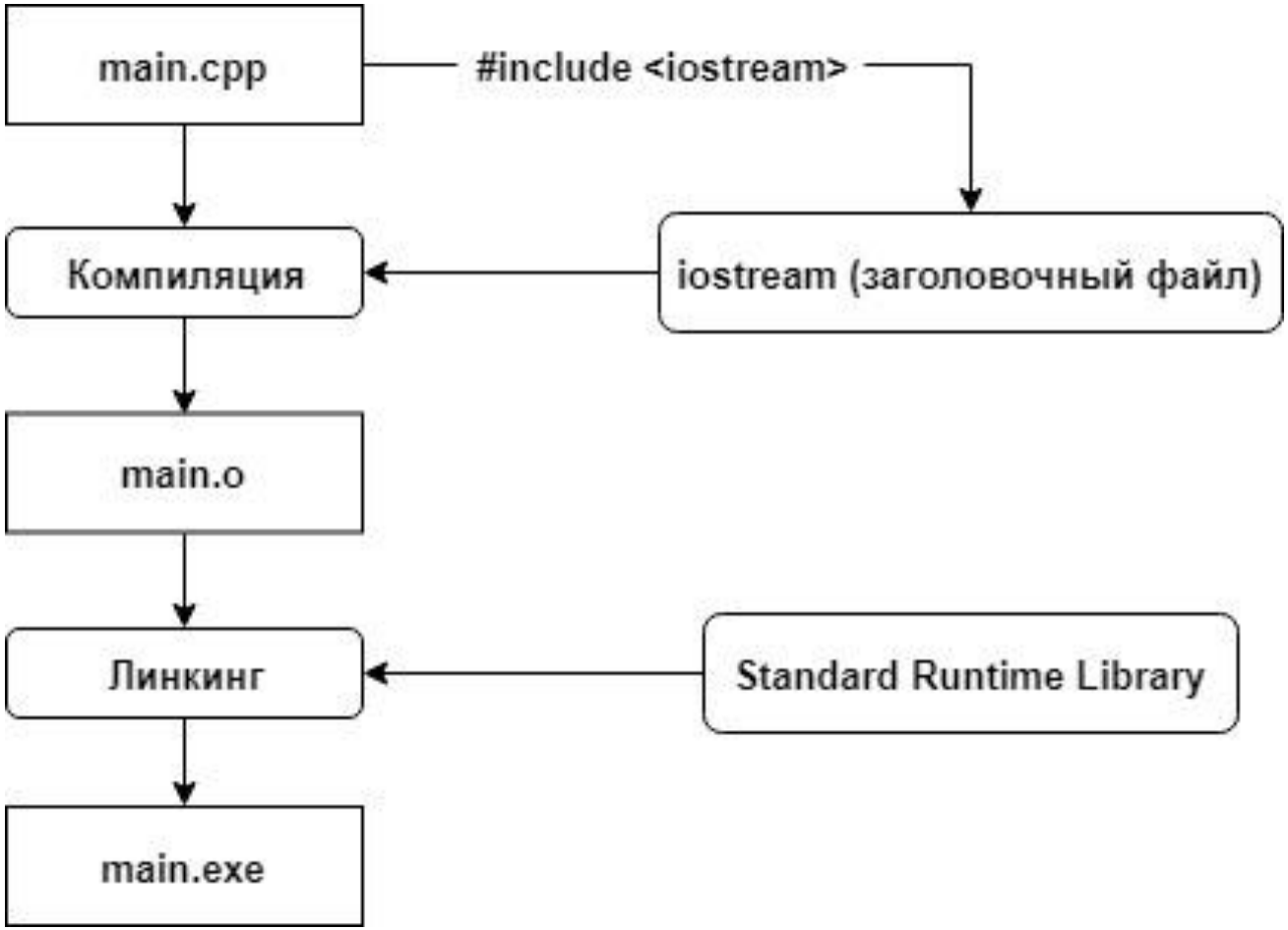
- **iostream** – потоки ввода/вывода
- **fstream** – файловые потоки
- **sstream** – строковые потоки

Заголовочные файлы

Директива **#include <iostream>** - используется для присоединения внешнего файла, в данном случае - iostream - для поддержки системы ввода-вывода.

include - включать(анг)

iostream
input output stream
входящий исходящий поток(анг)


main.cpp
#include <iostream>
Компиляция
iostream (заголовочный файл)
main.o
Линкинг
Standard Runtime Library
main.exe

Пространства имен (namespace)

Директива **using** открывает доступ к пространству имен (англ. namespace) **std**, в котором определяются **средства стандартной библиотеки языка C++.**

- **using namespace std**
- **using namespace standart** - использование имен стандартных(анг)

Пространства имен (namespace)

Пространство имен (*namespace*) — окружение, созданное для логической группировки уникальных имен.

- Необходимо чтобы избежать конфликтов имен идентификаторов.
- Функциональные особенности стандартной библиотеки объявляются внутри пространства имен **std**.

Вызов объекта: **std :: *имя объекта*;**

Пример пространства имен

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

int main()
{
setlocale(LC_CTYPE, "rus");
int a,b,c;
	int main()
	{
	setlocale(LC_CTYPE, "rus");
	int a,b,c;

cout << “Введи данные\n"; cin >>a>>b>>c;	std:: cout << “Введи данные\n"; std:: cin >>a>>b>>c;

system("pause");
return 0;
}
	system("pause");
	return 0;
	}

Функция main()

- Выполнение программы начинается со специальной стартовой функции main.
- В момент запуска программы, управление передается данной функции.
- Функция **main** обязательно должна быть определена в одном из модулей программы. Модуль, содержащий функцию main принято называть главным модулем.

Функция main()

Стандарт предусматривает два формата функции:

//без параметров

тип main( ){/* … */}

//с двумя параметрами

тип main(int argc, char* argv[]){/* … */}

Если программу запускать через командную строку, то существует возможность передать какую-либо информацию этой программе.

- Параметр argc имеет тип int, и содержит количество параметров, передаваемых в функцию main. Причем argc всегда не меньше 1, даже когда функции main не передается никакой информации, так как первым параметром считается имя приложения.
- Параметр argv[] представляет собой массив указателей на строки. Через командную строку можно передать только данные строкового типа.

Функция main()

Функция **main** может возвращать определенное значение, или не возвращать ничего.

- Если функция не возвращает никакого значения, то она должна иметь тип **void** (такие функции иногда называют процедурами)
- Функция может возвращать значение, тип которого в большинстве случаев аналогично типу самой функции.

```
int main()
{
        ...
}
```

или

```
void main()
{
        ...
}
```

Структура функции

заголовок

```
int main(void)
```

```
{
cout << “Hellow Word \n”;
return(0);
}
```

Тело функции

Заголовок функции

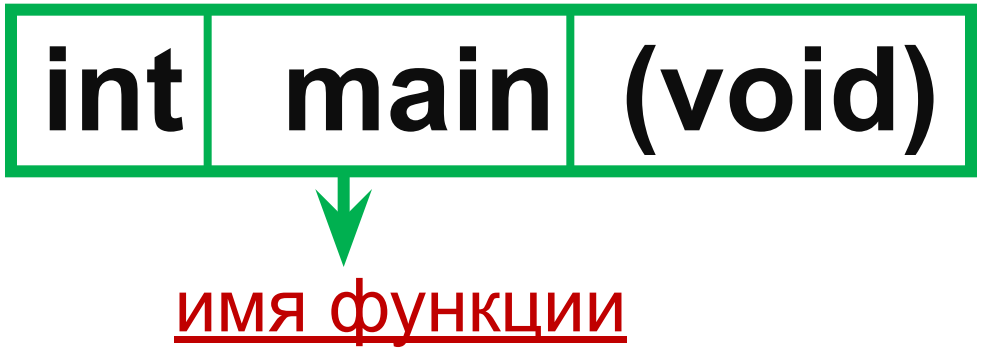
тип значения которое возвращает функция

В нашем случае это int.
То есть, когда функция main закончит свою работу, она должна вернуть в программу которая её вызвала, какое-то **целое значение**.

Если не нужно чтобы программа возвращала какое-то значение, то пишем тип **void**.
Если бы функция main не должна была бы ничего возвращать, то её заголовок выглядел бы так.

void main(void)

Заголовок функции



В нашем случае это имя **main**. Главная функция всегда имеет имя **main.**

Но могло быть и какое-нибудь другое.

Заголовок функции

типы и количество аргументов (параметров) функции

В нашем случае там написано **void**, это значит то функция не принимает никаких аргументов.

(void) — это перечень аргументов функции. Слово **void** указывает, что у данной функции нет аргументов

Директива #define

Директива **#define** служит для поиск и замена одного набора символов на другой.

- Идентификаторы, заменяющие текстовые или числовые константы, называют именованными константами.
- Идентификаторы, заменяющие фрагменты программ, называют макроопределениями

Директива **#define** имеет две синтаксические формы:

```
#define идентификатор текст

#define идентификатор (список параметров) текст
```

Пример:

```
#define WIDTH  80

#define LENGTH (WIDTH+10)
```

Эти директивы изменят в тексте программы каждое слово WIDTH на число 80, а каждое слово LENGTH на выражение (80+10) вместе с окружающими его скобками.

Объявление переменных

Язык СИ++ требует **явного объявления всех переменных** используемых в программе вместе с указанием соответствующих им типов.

Объявления переменной имеет следующий формат:

<спецификатор типа> имя_1, имя_2, ..., имя_n;

Спецификатор типа – одно или несколько ключевых слов, определяющие тип объявляемой переменной.

Например:

```
unsigned int n;
int b,f2,f3;
int c;
long d;
```

Переменные. Инициализация и Присваивание

Переменная - это ячейка в памяти компьютера, которая имеет имя и хранит некоторое значение.

- Значение переменной может меняться во время выполнения программы.
- При записи в ячейку нового значения старое стирается.

Типы переменных

- int – целое число в интервале [-32768…32767] (2 байта)
- float – вещественное число, *floating point* (4 байта)
- char – символ, *character* (1 байт)

Тип переменной

- область допустимых значений
- допустимые операции
- объём памяти
- формат хранения данных
- для предотвращения случайных ошибок

Начальные значения:

```
int a, b = 1, c = 55;
```

Что в переменной **a**?

ТИПЫ ДАННЫХ ЯЗЫКА СИ++

- В языке СИ++ имеется 4 базовых арифметических типа и 2 модификатора (знака и длины)

Модификаторы типов

- Модификатор знака unsigned/signed
- Модификатор длины long, short

АРИФМЕТИЧЕСКИЕ ТИПЫ ДАННЫХ ЯЗЫКА СИ++

Тип данных	Размер (байт)	Диапазон значений	Эквивалентные названия типа
char	1	–128 ... +127	signed char
int	2/4	зависит от системы	signed, signed int
unsigned char	1	0...255	нет
unsigned int	2/4	зависит от системы	Unsigned
short int	2	–32768 ... 32767	short, signed short int
unsigned short	2	0 ... 65535	unsigned short int
long int	4	–2147483648 ... 2147483647	long, signed long int
unsigned long int	4	0 ... 4294967295	unsigned long
float	4	±(3.4E–38 ... 3.4E+38)	нет
double	8	±(1.7E–308 ... 1.7E+308)	нет
long			

- В языке C++ объявлять и определять переменные можно в любом месте программы, однако область их действия начинается с момента
- объявления и заканчивается ближайшей за этой переменной закры

Объявление переменных

- Оператор объявления (описания) переменных имеет следующий синтаксис:

```
<тип>   <имя1>,<имя2>,...;
```

выделение места в памяти

Объявление переменных:

тип – целые

список имен переменных

```
int a, b, c;
```

```
int  x,y,z;
double a,b;
float t;
```

Имена переменных

МОЖНО использовать

- латинские буквы (A-Z, a-z)

заглавные и строчные буквы **различаются**

- цифры

имя не может начинаться с цифры

- знак подчеркивания _

НЕЛЬЗЯ использовать

- ~~русские буквы~~
- ~~скобки~~
- ~~знаки +, =, !, ? и др.~~

Какие имена правильные?

AXby R&B 4Wheel Вася “PesBarbos”
TU154 [QuQu] _ABBA A+B

Определение переменных/ Инициализация

- Оператор объявления (описания) переменных имеет следующий синтаксис:

```
<тип> <имя1>=<нач.значение1>,<имя2>=<нач.значение2>,...;
```

```
1  #include <iostream>
2
3  int main()
4  {
5      int age;
6      age = 28;
7      std::cout<<"Age = " << age;
8      return 0;
9  }
```

```
1  #include <iostream>
2
3  int main()
4  {
5      int age = 28;
6      std::cout<<"Age = " << age;
7      return 0;
8  }
```

Инициализация по умолчанию

```
1  #include <iostream>
2
3  int x;
4  int main()
5  {
6      int y;
7      std::cout <<"X = " << x << "\n";
8      std::cout <<"Y = " << y;
9
10     return 0;
11 }
```

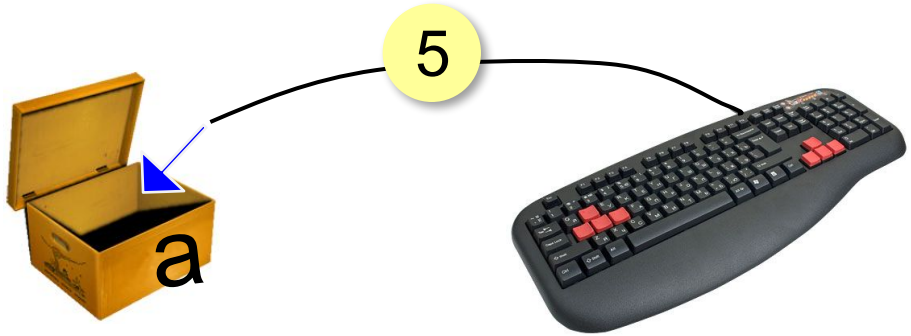
Управляющие символьные последовательности

Последовательность	Значение
\a	Звуковой сигнал
\b	Стирание предыдущего символа
\f	Перевод страницы
\n	Новая строка
\r	Возврат в начало текущей строки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\?	Знак вопроса
\’	Одинарная кавычка
\”	Двойная кавычка
\\	Обратная косая черта
\ddd	ASCII-код символа в восьмеричной системе
\xdd	ASCII-код символа в шестнадцатеричной системе

Ввод и вывод данных в поток

Ввод данных

ввести значение **a** из входного потока




```
cin >> a;
```

1. Программа ждет, пока пользователь введет значение и нажмет *Enter*.
2. Введенное значение записывается в переменную **a**.

Ввод и вывод данных в поток

Вывод данных

```
cout << a;                    //вывод значения
                              //переменной a

cout << a << endl;            //...и переход
                              //на новую строку

cout << "Привет!";            //вывод текста

cout << "Ответ: " << c;

//вывод текста и значения переменной c

cout << a << "+" << b << "=" << c;
```

Форматированный ввод-вывод в C++

- флаги форматированного ввода-вывода;
- манипуляторы форматирования

• Использование флагов

- • Флаги позволяют установить параметры ввода-вывода, которые будут действовать во всех последующих операторах ввода-вывода до тех пор, пока не будут отменены.

Для установки флага вывода

```
cout.setf(ios::flag)
```

Для снятия флага конструкция:

```
cout.unsetf(ios::flag)
```

Для установгвки нескольких флагов конструкция:

```
cout.setf(ios::flag1 | ios::flag2 | ios::flag3)
```

• Использование флагов

right	Выравнивание по правой границе
left	Выравнивание по левой границе (по умолчанию)

```
int r=-25;
cout.setf(ios::right);
cout.width(15);
cout<<«r=»<<r<<endl;
```

манипуляторы форматирования

Манипуляторы вставляются в операторы cin(cout) и устанавливают параметры текущего оператора ввода-вывода.

```
#include <iomanip>
...
a = 123;
cout << setw(5) << a;
```

манипуляторы для управления потоками

5

␣␣123

5 знаков

set width – установить ширину поля

```
cout<<«p=»<<fixed<<p<<endl;
```

p=146.673

Фиксированная форма вывода вещественных чисел (по умолчанию)

```
double p=1234.6578;
cout.setf(ios::fixed);
cout<<«p=»<<setw(15)<<setprecision(3)<<p<<endl;
```

p=1234.658

Определяет количество цифр (n-1) в дробной части числа

Именованные константы

```
const double pi=3.14159;  //объявление именованной конс
```

Преобразование типов

- • неявное преобразование типов;
- • явное преобразование типов;
- • функциональное преобразование типов.

Неявное преобразование типов.

- если оба операнда одного типа, то и результат будет того же типа, *например, в выражении 2/3 оба операнда целого типа, значит, и результат будет целого типа, в этом случае он будет равен целому частному от деления 2 на 3, т. е. 0*;
- если операнды имеют разный тип, то тип результата будет совпадать с более широким типом операнда, *например, в выражении 2/3.0 первый операнд — целого типа, второй — вещественного типа, значит, и результат будет вещественного типа, т. е. 0.6667.*

Явное преобразование типов и функциональное преобразование типов.

```
<тип> <выражение>
(<тип>) <выражение>
```

Если *применить ко всему выражению операцию* явного преобразования типов `double (2/3)` *или операцию* функционального преобразования типов `(double)(2/3)`, *то сначала будет вычислено выражение в скобках (2/3), равное* 0, *а затем результат будет преобразован к типу* double *и станет равным* 0.0.

Операторы C++

- арифметические операции;
- операции инкремента и декремента;
- операции присваивания;
- операции отношения;
- логические операции.

Арифметическое выражения

\+ сложение;
\- вычитание;
* умножение;
/ деление;
% — нахождение остатка от деления нацело.

```
          3   1 2   4       5    6
a = (c + b*5*3 - 1) / 2 * d;
```

Приоритет (*старшинство*):

1)скобки
2)умножение и деление
3)сложение и вычитание

$$a = \frac{c + b \cdot 5 \cdot 3 - 1}{2} \cdot d$$

Деление

Результат деления целого на целое – **целое** число (остаток отбрасывается):

Что запишется в **x**?

```
int a = 3, b = 4;
float x;
x = 3 / 4;
x = 3. / 4;
x = 3 / 4.;
x = a / 4;
x = a / 4.;
x = a / b;
x = float(a) / 4;
x = a / float(b);
```

Остаток от деления

% – остаток от деления

```
int a, b, d;
d = 85;
b = d / 10;
a = d % 10;
d = a % b;
d = b % a;
```

Для отрицательных чисел:

```
int a = -7;
b = a / 2;
d = a % 2;
```

! В математике не так!

остаток ≥ 0

-7 = (-4)*2 + 1

Инкремент и декремент

- Постфиксная форма

```
X=1;
Y=X++;        (Y=1,    затем  X=2)

Z=5;
T=Z--;        (T=5,    затем Z=4)
```

- Префиксная форма

```
X=1;
Y=++X;        (X=2,    затем  Y=2)
```

Инкремент и декремент

- Постфиксная форма
- Префиксная форма

```
++a;          c++;
--b;          d--;
```

Сокращенная запись операций

```
int a, b;
...
a ++;      // a = a + 1;
a --;      // a = a - 1;
a += b;    // a = a + b;
a -= b;    // a = a - b;
a *= b;    // a = a * b;
a /= b;    // a = a / b;
a %= b;    // a = a % b;
```

Приоритет операций

1) * , /, %;
2) +, -;
3) ! — логическое отрицание;
4) && — логическое И;
5) || — логическое ИЛИ;
6) =, <>, <, >, >=, <=.

подключить математическую библиотеку

```
#include <cmath>
```

abs(x) – модуль целого числа
fabs(x) – модуль вещественного числа
sqrt(x) – квадратный корень
sin(x) – синус угла, заданного **в радианах**
cos(x) – косинус угла, заданного **в радианах**
exp(x) – экспонента e^x
ln(x) – натуральный логарифм
pow(x,y) – x^y: возведение числа x в степень y
floor(x) – округление «вниз»
ceil(x) – округление «вверх»

```
float x;
x = floor(1.6);// 1
x = ceil(1.6); // 2
```

```
x = floor(-1.6);//-2
x = ceil(-1.6); //-1
```

Объявление переменных

- **Глобальные переменные** описываются вне функций и действуют от конца описания до конца файла.
- **Локальная переменная** описывается внутри функции и действует от конца описания до конца функции.

Что происходит дальше?

текст программы на Си или Си++

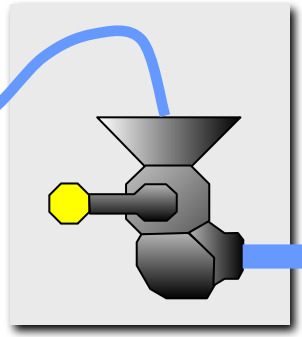
first.cpp

```
main()
{

}
```

исходный файл

транслятор

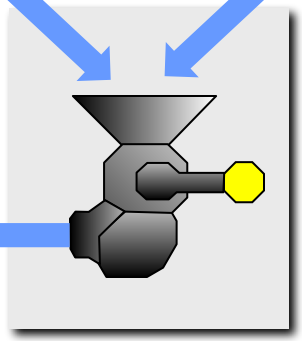


first.o

ЪБzЦ2?|ё3БКа
n/36ШпIC+И-
Ц3_5МуРЧб
s6bд^:/@:лЖ1_

объектный файл

стандартные функции



редактор связей (компоновка)

first.exe

MZPo:ЄРэ_е3"!_
`кn,ЦbЄ-Щр1
G_БАС,
_Ощях¤9жФ

исполняемый файл

- **по исходному файлу можно восстановить остальные**
- **исполняемый файл можно запустить**

Спасибо за внимание!

LOGO