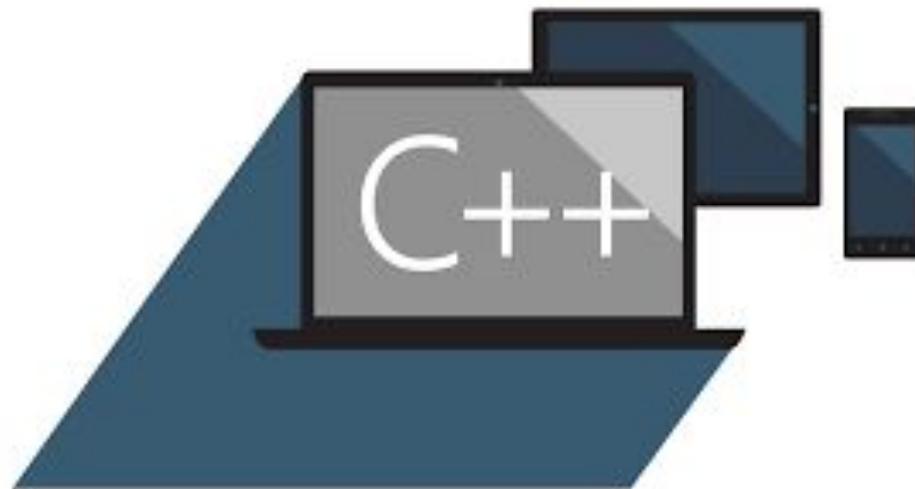




Структура программы на языке C++

Лекция 4





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



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

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 находиться 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 находиться 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 скобками 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 C++

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

Структура программы на C++

```
#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 C):

#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)

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

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

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

```
  cout << "Введи данные\n";  
  cin >>a>>b>>c;
```

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

standart

```
#include <iostream>
```

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

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

```
  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 << "Hello Word \n";  
return(0);  
}
```

Тело функции

Заголовок функции

```
int main (void)
```

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

int	main	(void)
------------	-------------	---------------



имя функции

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

Но могло быть и какое-нибудь другое.

Заголовок функции

```
int main (void)
```



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 поиска 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;

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?

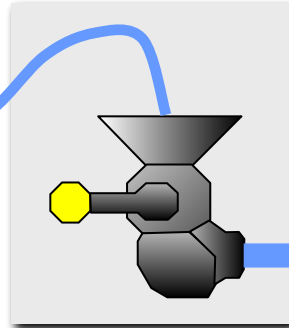
текст программы на Си или Си++

first.cpp

```
main()  
{  
  
}
```

исходный файл

транслятор



first.o

```
ЪБzЦ2?|ё3БKa  
n/36Шп|C+И-  
Ц3_5MyPЧ6  
s6bd^:/@:лЖ1_
```

объектный файл

стандартные
функции

first.exe

```
MZPo:ЄPэ_e3"!_  
`кп,ЦбЄ-Щр1  
G_БАС,  
_Ощяхα9жФ
```

исполняемый файл

редактор
связей
(компоновка)



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



Спасибо за внимание!



LOGO