2. СТРУКТУРА ПРОГРАММЫ

Оглавление

§2.1 Основные определения, составляющие структуру программы

Рассмотрим основные элементы, которые будут составлять структуры любой компьютерной программы.

Компьютерная программа — это последовательность инструкций, которые сообщают компьютеру, что ему нужно сделать.

Стэйтмент (англ. «statement») — это тип инструкции в компьютерных программах. Он является наименьшей независимой единицей в языках программирования. Стэйтмент в программировании — это то же самое, что и «предложение» в русском языке. Мы пишем предложения, чтобы выразить какую-то идею. В языках программирования мы пишем стэйтменты, чтобы выполнить какое-то задание. Все стэйтменты в C++ и C# **заканчиваются точкой с запятой**.

Есть много разных видов стэйтментов в языках программирования. Самые распространенные из них представлены в таблице 2.1.

Таблица 2.1. Виды стэйтментов

Наименование стэйтмента	**Объявление**	**Описание**
Стэйтмент объявления (англ. «statement declaration»)	`int x;`	Он сообщает компилятору, что `x` является целочисленной переменной. Все переменные в программе должны быть объявлены, прежде чем использованы.
Стэйтмент присваивания (англ. «assignment statement»)	`x = 50;`	Здесь мы присваиваем значение 50 переменной `x`.
Стэйтмент вывода (англ. «output statement»)	`cout << x;`	Мы выводим значение переменной `x` на экран

Компилятор также способен обрабатывать выражения.

Выражение (англ. «expression») — это объект, который создается (составляется) для проведения вычислений и нахождения соответствующего результата. Например, в математике выражение 2 + 3 имеет значение 5. Выражения в языках программирования могут содержать:

- отдельные цифры и числа (например, 2, 45);
- буквенные переменные (например, x, y);
- операторы, в т.ч. математические (например, +, -);
- вызовы функций.

Выражения могут состоять как из единичных символов — цифр или букв (например, 2 или x), так и из различных комбинаций этих символов с операторами (например, 2 + 3, 2 + x, x + y или (2 + x) * (y - 3)).

Для наглядности разберем простой корректный стэйтмент присваивания `x = 2 + 3;`. Здесь мы вычисляем результат сложения чисел 2 + 3, который затем присваиваем переменной `x`.

В языках программирования стэйтменты объединяются в блоки — функции.

Функция — это последовательность стэйтментов. Каждая программа в C++ должна содержать главную функцию `main()` (в C# - функцию `Main()`). Именно с первого стэйтмента, находящегося в функции `main()`, и начинается выполнение всей программы. Функции, как правило, выполняют конкретное задание. Например, функция `max()` может содержать стэйтменты, которые определяют большее из заданных чисел, а функция `calculateGrade()` может вычислять среднюю оценку студента по какой-либо дисциплине.

Библиотека — это набор скомпилированного кода (например, функций), который был «упакован» для повторного использования в других программах. С помощью библиотек можно расширить возможности программ.

§2.2 Структура программы на C++

C++ - компилируемый язык. Это означает, что для запуска программы сначала необходимо транслировать ее из текстовой формы, понятной для человека, в форму, понятную для машины. Эту задачу выполняет особая программа, которая называется компилятором. То, что вы пишете и читаете, называется исходным кодом или **исходным текстом** программы, а то, что выполняет компьютер, называется **выполняемым**, **объектным** или **машинным кодом**.

Обычно файлы с исходным кодом программы на языке C++ имеют суффикс .cpp (например, hello_world.cpp) или .h (например, std_lib_facilities.h), а файлы с объектным кодом имеют суффикс .obj (в Windows) или .o (в Unix). Следовательно, простое слово код является двусмысленным и может ввести в заблуждение: его следует употреблять с осторожностью и только в ситуациях, когда недоразумение возникнуть не может. Если не указано иное, под словом код подразумевается «исходный код» или даже «исходный код за исключением комментариев», поскольку комментарии предназначены для людей и компилятор не переводит их в объектный код.

Компилятор читает исходный код и пытается понять, что вы написали. Он проверяет, является ли программа грамматически корректной, определен ли смысл каждого слова. Обнаружив ошибку, компилятор сообщает о ней, не пытаясь выполнить программу.

Рассмотрим стандартную программу «Hello, world!»:

Листинг 2.1.

```
1  #include <iostream>
2  int main()
3  {
4  std::cout << "Hello, World!" << std::endl;
5  return 0;
6  }
```

Эту программу можно разделить на 2 части: директивы препроцессора, которые начинаются с символа #, и основную часть программы, которая начинается с определения сигнатуры главной функции `int main()`.

§2.2.1 Директива препроцессора #include

Как и предполагает название, **препроцессор** – это инструмент, который запускается перед фактическим началом компиляции. **Директивы препроцессора** – это команды препроцессору, они всегда начинаются со знака «решетка» (#). В строке 1 листинга 2.1. директива `#include <имя файла>` указывает препроцессору взять содержимое файла (в данном случае – `iostream`) и включить его в строку, где расположена директива.

`iostream` – это стандартный файл заголовка, который включается потому, что она содержит определение объекта потока вывода данных `cout`, используемого в строке 4 для вывода на экран сообщения `"Hello, World!".` Компилятор смог откомпилировать строку 4 только потому, что мы заставили препроцессор включить определение объекта потока `cout` в строке 1.

§2.2.2 Тело программы – функция main()

После директив препроцессора следует тело программы, расположении в функции `main()`. Исполнение программы C++ всегда начинается здесь, поэтому функцию `main()` часто называют **точкой входа в программу**. По стандартному соглашению, перед функцией `main()` указывается тип `int`. Тип `int` в данном случае – это тип данных возвращаемого значения функции `main()`.

Всё, что находится между открывающей фигурной скобкой в строке 3 и закрывающей фигурной скобкой в строке 6, — считается содержимым функции `main()`.

Рассмотрим подробнее строку 4, выполняющую задачу программы, представленной в листинге 2.1.

`cout` («console-out» - вывод на консоль, произносится как «си-аут») – оператор, выводящий на экран строку `"Hello, World!"`. `cout` – это поток, определенный в стандартном пространстве имен (поэтому и указывается `std::cout`), а то, что мы делаем – это помещаем текст строки `Hello, World!` в данный поток, используя оператор вывода в поток `<<`.

Потоком (stream) называется последовательность символов, записываемая или читаемая из устройства ввода-вывода некоторым способом. Термин «поток» подразумевает, что символы поступают и передаются последовательно на протяжении определенного времени.

Оператор `std::endl` используется для завершения строки, а ввод его в поток эквивалентен переносу строку. Оператор вывода в поток используется при необходимости вывода в поток каждого нового элемента.

Функции в языке C++ должны возвратить значение, если противное не указано явным образом. `main()` – это функция, всегда и обязательно возвращающая целое число. Традиционно программисты возвращают значение 0 в случае успешного выполнения программы и -1 в случае ошибки.

Язык C++ чувствителен к регистру. Поэтому готовьтесь к неудаче компиляции, если напишете `Int` вместо `int`, `Void` вместо `void` и `Std::Cout` вместо `std::cout`.

§2.2.3 Концепция пространства имен

Пространства имен – это множество, в рамках которого определяются различные идентификаторы (имена типов, функций, переменных и т. д.). Пространства имен используются для организации кода в виде логических групп и с целью избежания конфликтов имен, которые могут возникнуть, особенно в таких случаях, когда база кода включает несколько библиотек. Все идентификаторы в пределах пространства имен доступны друг другу без уточнения.

Чтобы избежать конфликта имен, разрешается также объявлять собственные пространства имен через ключевое слово `namespace`. Всё, что объявлено внутри пользовательского пространства имен, — принадлежит только этому пространству имен. Для того, чтобы указать компилятору место поиска идентификатора в определенном пространстве имен необходимо использовать название необходимого

пространства имен вместе с оператором разрешения области видимости (`::`) и требуемым идентификатором.

Причина использования в программе синтаксиса оператора `std::cout`, а не просто `cout`, в том, что используемый элемент (`cout`) находится в стандартном пространстве имен (`std`).

При вызове оператора `std::cout` вы указываете компилятору использовать именно тот объект, который доступен в стандартном пространстве имен `std`. В этом пространстве объявлена вся стандартная библиотека C++.

Многие программисты находят утомительным регулярный ввод в коде спецификатора при использовании оператора `cout` и других подобных элементов, содержащих в том же пространстве имен. Объявление `using namespace std;`, представленное в листинге 2.2, позволит избежать этого повторения.

Листинг 2.2.

```
1  #include <iostream>
2  int main()
3  {
4  //Указать компилятору пространство имен для поиска
5  using namespace std;
6  cout << "Hello, World!" << endl;
7  return 0;
8  }
```

Сообщив компилятору, что предполагается использование пространство имен `std`, можно не указывать пространство имен явно в строке 6.

§2.2.4 Комментарии в коде C++

Строка 4 листинга 2.2 содержит текст на человеческом языке, но программа все равно компилируется. Она также не влияет на вывод программы. Такая строка называется комментарием.

Комментарий – это способ размещения произвольного текста внутри кода программы. Комментарии игнорируются компилятором и обычно используются программистами для объяснений в коде.

Различают два вида комментариев:

1. Символ `//` означает, что следующая строка – комментарий. Например:

```
// Это комментарий
```

2. Текст, содержащийся между символами /* и */, также являются комментариями, даже если он занимает несколько строк. Например:

```
/* Это комментарий,
Занимающий 2 строки /*
```

§2.2.5 Функции в C++

Функция — это последовательность стэйтментов для выполнения определенного задания. Часто ваши программы будут прерывать выполнение одних функций ради выполнения других. Вы делаете аналогичные вещи в реальной жизни постоянно. Например, вы читаете книгу и вспомнили, что должны были сделать телефонный звонок. Вы оставляете закладку в своей книге, берете телефон и набираете номер. После того, как вы уже поговорили, вы возвращаетесь к чтению: к той странице, на которой остановились.

Программы в C++ работают похожим образом. Иногда, когда программа выполняет код, она может столкнуться с вызовом функции.

Вызов функции — это выражение, которое указывает процессору прервать выполнение текущей функции и приступить к выполнению другой функции. Процессор «оставляет закладку» в текущей точке выполнения, а затем выполняет вызываемую функцию. Когда выполнение вызываемой функции завершено, процессор возвращается к закладке и возобновляет выполнение прерванной функции.

Функция, в которой находится вызов, называется **caller**, а функция, которую вызывают — **вызываемая функция**, например,

Листинг 2.3.

```
1  #include <iostream> // для cout и endl
2  // Объявление функции doPrint(), которую мы будем вызывать
3  void doPrint() {
4     cout << "In doPrint()" << endl;
5  }
6  // Объявление функции main()
7  int main()
8  {
9  using namespace std;
10 cout << "Starting main()" << std::endl;
```

11	doPrint(); // прерываем выполнение main() вызовом функции doPrint(). main() в этом случае является caller-ом
12	cout << "Ending main()" << endl;
13	return 0;
14	}

Результат выполнения программы:

Starting main()

In doPrint()

Ending main()

В строке 3 находится **объявление функции** (function declaration), которое в основном указывает компилятору, что вы хотите создать функцию по имени `doPrint()`. Тип возвращаемого значения `void` указывает компилятору, что функция **не возвращает никакое значение**, а лишь производить манипуляции над данными. Далее, в строках 3-5, следует **определение функции** (function definition), т.е. ее реализация.

Программа, представленная в листинге 2.3., начинает выполнение с первой строки функции `main()`, где выводится на экран следующая строка: Starting main(). Вторая строка функции `main()` вызывает функцию `doPrint()`. На этом этапе выполнение стэйтментов в функции `main()` приостанавливается и процессор переходит к выполнению стэйтментов внутри функции `doPrint()`. Первая (и единственная) строка в `doPrint()` выводит текст In doPrint(). Когда процессор завершает выполнение `doPrint()`, он возвращается обратно в `main()` к той точке, на которой остановился. Следовательно, следующим стэйтментом является вывод строки Ending main().

Обратите внимание, для вызова функции нужно указать её имя и список параметров в круглых скобках (). В примере выше параметры не используются, поэтому круглые скобки пусты. Поэтому круглые скобки при объявлении функции опускать нельзя.

§2.2.6 Простые операторы ввода std::cin и вывода std::cout

Ваш компьютер позволяет взаимодействовать с выполняющимися на нем приложениями разными способами, а также позволяет этим приложениям взаимодействовать с вами разными способами. Вы можете взаимодействовать с

приложениями, используя периферийные устройства – клавиатуру или мышь. Рассмотрим самую простую форму ввода и вывода на языке C++ - использование консоли для отображения или ввода информации.

Для записи простых тестовых данных на консоль используется оператор `std::cout` (произносится как standard see-out – «стандарт си-аут») и оператор `std::cin` (произносится как standard see-in – «стандарт си-ин») для чтения текста и чисел с консоли (как правило, с клавиатуры). Фактически при отображении слов Hello World на экране в листинге 2.1 вы уже встречались с оператором `cout`:

```
std::cout << "Hello World" << std::endl;
```

Здесь оператор `cout` сопровождается оператором вывода `<<` (позволяющим вставить данные в поток вывода), который подлежит выводу строковым литералом "Hello World" и символом новой строки в форме оператора `std::endl` (произносится как standard endline (стандарт енд-лайн)).

Применение оператора `cin` сопровождается указанием переменной, в которую следует поместить вводимые данные:

```
std::cin >> ПЕРЕМЕННАЯ;
```

Оператор сопровождается оператором извлечения значения (данные извлекаются из входного потока) и переменной, в которую следует поместить данные. Если вводимые данные, разделенные пробелом, стоит хранить в двух переменных, то можно использовать один оператор:

```
std::cin >> ПЕРЕМЕННАЯ1 >> ПЕРЕМЕННАЯ2;
```

Обратите внимание на то, что оператор `cin` применяется для ввода как текстовых, так и числовых данных, как представлено в листинге 2.4.

Листинг 2.4.

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4  int main()
5  {
6  // Объявление переменной для хранения целого числа
7  int InputNumber;
8  cout << "Enter an integer: ";
9  // Сохранить введенное пользователем целое число
```

```
10  cin >> InputNumber;
11  // Аналогично с текстовыми данными
12  cout << "Enter your name: ";
13  string InputName;
14  cin >> InputName;
15  cout << InputName << " entered " << InputNumber << endl;
16  return 0;
17  }
```

Результат выполнения программы:

Enter an integer: 2020

Enter your name: Zaid

Zaid entered 2020

В строке 7 переменная `InputNumber` объявляется как способная хранить данные типа `int`. В строке 8 пользователя просят ввести число, используя оператор `cout`, а введенное значение сохраняется в целочисленной переменной с использованием оператора `cin` в строке 10. То же самое повторяется при сохранении имени пользователя, которое, конечно, не может содержаться в целочисленной переменной. Для этого используется другой тип — `string`, как можно заметить в строках 13 и 14. Именно поэтому, чтобы использовать тип `string` далее в функции `main()`, в строке 2 была включена директива `#include <string>`. И наконец, в строке 15 оператор `cout` используется для отображения введенного имени и числа с промежуточным текстом, чтобы получить вывод **Zaid entered 2020**.

§2.2.7 Пример обработки синтаксических ошибок

Компиляторы довольно придирчивы к синтаксису. Пропуск какой-нибудь детали, например, директивы `#include`, двоеточия или фигурной скобки. приводит к ошибке. Кроме того, компилятор точно так же абсолютно нетерпим к опечаткам. Продемонстрируем это рядом примеров, в каждом из которых сделана одна небольшая ошибка. Каждая из этих ошибок является довольно типичной.

Листинг 2.5.

```
1  // пропущен заголовочный файл
2  int main ( ) {
3  cout << " Hello, World! \n";
4  return 0; }
```

Мы не сообщили компилятору о том, что представляет собой объект cout, поэтому он сообщает об ошибке. Для того чтобы исправить программу. следует добавить директиву #include.

Листинг 2.6.

```
1  #include <iostream>
2  int main ( ) {
3  cout << " Hello, World! \n";
4  return 0; }
```

Компилятор нередко будет вас раздражать. Иногда будет казаться, что он придирается к несущественным деталям (например, к пропущенным точкам с запятыми) или к вещам, которые вы считаете абсолютно правильными. Однако компилятор, как правило, не ошибается: если уж он выводит сообщение об ошибке и отказывается создавать объектный код из вашего исходного кода, то это значит, что ваша программа не в порядке: иначе говоря, то, что вы написали, не соответствует стандарту языка C++.

§2.3 Структура программы на C#

§2.3.1 Инструкции и блоки кода

Базовым строительным блоком программы являются инструкции (statement). Инструкция представляет некоторое действие, например, арифметическую операцию, вызов метода, объявление переменной и присвоение ей значения. В конце каждой инструкции в C# ставится точка с запятой (;). Данный знак указывает компилятору на конец инструкции. Например:

```
Console.WriteLine("Привет");
```

Данная строка представляет вызов метода `Console.WriteLine()`, который выводит на консоль строку. В данном случае вызов метода является инструкцией и поэтому завершается точкой с запятой.

Набор инструкций может объединяться в блок кода. Блок кода заключается в фигурные скобки, а инструкции помещаются между открывающей и закрывающей фигурными скобками:

```
{
    Console.WriteLine("Привет");
    Console.WriteLine("Добро пожаловать в C#");
```

```
}
```

§2.3.2 Тело программы – класс Program и метод Main

Точкой входа в программу на языке C# является метод `Main`. При создании проекта консольного приложения в Visual Studio, например, создается следующий метод `Main`:

Листинг 2.7.

```
1  using System; // подключаемое пространство имен
2  namespace HelloApp // объявление нового пространства имен
3  {
4  class Program // объявление нового класса
5  {
6      static void Main()
7      {
8          Console.WriteLine("Hello World!");
9      }
10 } // конец объявления нового класса
11 } // конец объявления нового пространства имен
```

В начале файла идет директива `using`, после которой идет название подключаемого пространства имен. Пространства имен представляют собой организацию классов в общие блоки. Например, на первой строке `using System;` подключается пространство имен System, которое содержит фундаментальные и базовые классы платформы .NET.

И так как C# имеет Си-подобный синтаксис, каждая строка завершается точкой с запятой, а каждый блок кода помещается в фигурные скобки.

Далее начинается уже собственно наше пространство имен, которое будет создавать отдельную сборку или исполняемую программу: сначала идет ключевое слово `namespace`, после которого название пространства имен. По умолчанию Visual Studio дает ему название проекта. Далее внутри фигурных скобок идет блок пространства имен.

Пространство имен может включать другие пространства или классы. В данном случае у нас по умолчанию сгенерирован один класс - `Program`. Классы объявляются похожим способом - сначала идет ключевое слово `class`, а потом название класса, и далее блок самого класса в фигурных скобках.

Класс может содержать различные переменные, методы, свойства, прочие инструкции. В данном случае у нас объявлен один метод `Main`. В программе на C# метод `Main` является входной точкой программы, с него начинается все управление. Он обязательно должен присутствовать в программе.

Слово `static` указывает, что метод `Main` - статический, а слово `void` - что он не возвращает никакого значения. Позже мы подробнее разберем, что все это значит.

По умолчанию метод `Main` размещается в классе `Program`. Название класса может быть любым. Но метод `Main` является обязательной частью консольного приложения. Если мы изменим его название, то программа не скомпилируется.

По сути, и класс, и метод представляют своего рода блок кода: блок метода помещается в блок класса. Внутри блока метода `Main` располагаются выполняемые в программе инструкции. По умолчанию он содержит одно действие: `Console.WriteLine("Hello World!");` - оно выводит на консоль строку "Hello World!".

§2.3.3 Консольный ввод-вывод данных

Есть другой способ вывода на консоль сразу нескольких значений:

Листинг 2.9.

```
1  using System;
2  namespace HelloApp
3  {
4      class Program
5      {
6          static void Main(string[] args)
7          {
8              string name = "Tom";
9              int age = 34;
10             double height = 1.7;
11             Console.WriteLine("Имя: {0}  Возраст: {2}  Рост: {1} м", name, height, age);
12             Console.ReadKey();
13         }
14     }
15 }
```

Этот способ подразумевает, что первый параметр в методе `Console.WriteLine` представляет выводимую строку ("Имя: {0} Возраст: {2} Рост: {1}м"). Все

последующие параметры представляют значения, которые могут быть встроены в эту строку (name, height, age). При этом важен порядок подобных параметров. Например, в данном случае вначале идет name, потом height и потом age. Поэтому у name будет представлять параметр с номером 0 (нумерация начинается с нуля), height имеет номер 1, а age - номер 2. Поэтому в строке "Имя: {0} Возраст: {2} Рост: {1} м" на место плейсхолдеров {0}, {2}, {1} будут вставляться значения соответствующих параметров.

Кроме Console.WriteLine() можно также использовать метод Console.Write(), он работает точно так же за тем исключением, что не осуществляет переход на следующую строку.

Кроме вывода информации на консоль мы можем получать информацию с консоли. Для этого предназначен метод `Console.ReadLine()`. Он позволяет получить введенную строку.

Теперь изменим весь этот код на следующий:

Листинг 2.10.

```
1  using System;
2  namespace HelloApp
3  {
4      class Program
5      {
6          static void Main(string[] args)
7          {
8              Console.Write("Введите свое имя: ");
9              string name = Console.ReadLine();
// вводим имя
10             Console.WriteLine($"Привет {name}");
// выводим имя на консоль
11             Console.ReadKey();
12         }
13     }
14 }
```

В строке 8 листинга 2.9 выводится сообщение-приглашение к вводу. В строке 9 определяется строковая переменная `name`, в которую пользователь вводит информацию с консоли. Затем в строке 10 выводится введенное имя на консоль.

Чтобы ввести значение переменной `name` внутрь выводимой на консоль строки, применяются фигурные скобки `{}`. То есть при выводе строки на консоль выражение `{name}` будет заменяться на значение переменной `name` – введенное имя.

Однако, чтобы можно было вводить таким образом значения переменных внутрь строки, перед строкой указывается знак доллара $.

Однако минусом этого метода является то, что `Console.ReadLine` считывает информацию именно в виде строки. Поэтому мы можем по умолчанию присвоить ее только переменной типа string. Как нам быть, если, допустим, мы хотим ввести возраст в переменную типа int или другую информацию в переменные типа double или decimal? По умолчанию платформа .NET предоставляет ряд методов, которые позволяют преобразовать различные значения к типам int, double и т.д. Некоторые из этих методов:

- `Convert.ToInt32()` (преобразует к типу int)
- `Convert.ToDouble()` (преобразует к типу double)
- `Convert.ToDecimal()` (преобразует к типу decimal)

Пример ввода значений:

Листинг 2.11.

```
1  using System;
2  namespace HelloApp
3  {
4      class Program
5      {
6          static void Main(string[] args)
7          {
8              Console.Write("Введите имя: ");
9              string name = Console.ReadLine();
10             Console.Write("Введите возраст: ");
11             int age = Convert.ToInt32(Console.ReadLine());
12             Console.Write("Введите рост: ");
13           double height = Convert.ToDouble(Console.ReadLine());
14             Console.Write("Введите размер зарплаты: ");
15                                          decimal   salary   =
Convert.ToDecimal(Console.ReadLine());
16           Console.WriteLine($"Имя: {name}  Возраст: {age}  Рост:
{height}м  Зарплата: {salary}$");
17             Console.ReadKey();
18         }
```

```
19      }
20  }
```

C# является регистрозависимым языком. Это значит, в зависимости от регистра символов какое-то определенные названия может представлять разные классы, методы, переменные. Например, название обязательного метода Main начинается именно с большой буквы: "Main". Если мы назовем метод "main", то программа не скомпилируется, так как метод, который представляет стартовую точку в приложении, обязательно должен называться "Main", а не "main" или "MAIN".