Информатика: семестр 3

Программирование на C++

Конспекты лекций

ООО "Тупой скат"

6 сентября 2017 г.

Оглавление

Лекция 1

Введение в C++

1.1 Этапы компиляции

C++ — компилируемый язык. Перевод с человеческого языка на машинный код происходит не во время исполнения, а заранее. Сборка программы при этом проходит в несколько этапов:

1. Препроцессинг: my.cpp → my.cpp′
   Здесь происходит подстановка всех макросов, а акже удаление комментариев:

   Код 1.1: Пример макроса

   ```
   1 #include "mylib"
   2 #define A 123456
   ```

2. Компиляция: my.cpp′ → main.o
   Он содержит переведенную на машинный код инструкции, но все еще содержит сигнатуру функций. Это означает, что объектный файл может содержать названия функций, их входные данные, но не саму реализацию.

3. Компоновка (линковка): my.o → my.exe
   Здесь уже все файлы связываются и функции, не реализованные в п.3, получают реализацию. Программа может быть разделена на модули так, что разные файлы используют функции других файлов. При этом, если один файл ссылается на ресурсы других файлов, эти ресурсы необходимо объявить в отдельном файле: header–файле (хидер на жаргоне). Хидер содержит лишь сигнатуры функций и переменных, однако не входит в не конечный результат компиляции. Во время компановки эти хидеры будут показывать, где искать функции.
   Подобное деление лежит в основе принципа раздельной компиляции: программы делятся на компоненты, которые собираются по отдельности. Это уменьшает время компиляции.

1.2 Синтаксис

Код 1.2: Демонстрация синтаксиса

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

В языке C++ есть особенности, отличающие его от Python. Так, при создании переменной необходимо явно указать ее тип (строка 4).
В первой строке — подключение библиотеки для работы ввода/вывода.
В строке 5 стоит оператор ввода из стандартного ввода (это консоль) и спец. оператор >>. Результат считывания уходит в переменную x. std — это пространство имен, о них будет сказано позже.
В строке 6 — вывод, по аналогии с вводом. При этом строки можно выводить конвейером (как видно ‘cout << "x="<< x‘ — эти строки будут сшиты при выводе).
В C++ табуляция не имеет значения, перевод строки после (6) и дальнейшие команды показывают это.
Строка 8 содержит код возврата — он будет возвращен по окончании программы (0 означает успех).

1.3 Арифметические операции и сравнение

x = y — арифметическая операция (не присваивание, как в питоне). Можно писать такой код

Код 1.3: Неясное поведение

```
1 y=(x=3)+(z=5)
```

В коде выше все работает, однако такое поведение программы будет неочевидным. В C++ все еще работают x += 1. Помимо этого имеются два оператора: x++ ≈ ++x ≈ x+=1. Разница следующая: y = (x++) запишет в y **старый** x, в то время как в y=(++x) будет записан **новый** x. Операции ** (возведение в степень) *нет*. Примеры некоторых арифметических действий:

1. x*y — при применении к целым числам результат будет целым числом.

2. x/y — при применении к целым числам будет применено целочисленное деление. Проблему можно решить, если добавить в конце ".". Так, 2/3 станет 2/3. — точка превратит число в вещественное.

3. x > y, x >= y, x == y. Это тоже арифметические операции, возвращают, правда, значение типа *bool*: **true** или **false**.

1.4 Циклы, условия и конструкции

В C++ три вида циклов, их синтаксис отличается от Python:

Код 1.4: Виды циклов

```
1 // 1 type
2 while(condition)
3 {
4 some code;
5 }
```

Условие надо обязательно кидать в скобки, иначе C++ не сможет отличить его от тела.

Код 1.5: Виды циклов

```
1 // 2 type
2 while(condition)
3 {
4 if(condition)
5 {
6 std::cout << "Exiting..\n";
7 break;
8 }
```

Здесь мы выходим из цикла по команде **break**.

Код 1.6: Виды циклов

```
1 // 3 type
2 do {
3 ...
4 } while(condition);
```

Как обычный while, но сначала выполняет тело цикла, а затем проверяет условие.

Код 1.7: Виды циклов

```
// 4 type
for(op1;condition;op3)
{
some code;
}
// EQUAL:
op1;
while(condition)
{
some code;
op3;
}
// Example:
for(i=0; i < 10; i++)
{
some code;
}
```

Это очень похоже на то, что в питоне было под видом

Код 1.8: Аналог цикла for в Python

```
 for i in range(0, 10, 1):
     some code
```

Переменные существуют локально, поэтому можно переменную i сделать только для цикла. Более того, переменная "живет"до тех пор, пока не будет завершен соответствующий блок:

Код 1.9: Виды циклов

```
for(int i=0; i < 10; i++)   // "i" won't be seen outside
{
some code;
}

// else === elif
if(condition)
{
..
} else if(condition2)
{
..
}
else {
..
}
```

1.5 Тернарный оператор

Тернарный оператор заменяет условия (иногда полезно):

Код 1.10: Пояснение к тернарному оператору

```
1  z = (a > 0) ? x : y;
2  // EQUALS TO:
3  if(a > 0)
4  {
5  z = x;
6  }
7  else
8  {
9  z = y;
10 }
```

Такой подход здорово уменьшает код, однако при неправильном использовании может сделать его нечитаемым.

1.6 Оператор switch

Код 1.11: Оператор switch

```
1  switch(x)
2  {
3    case 0: op1; // if x=0
4         op2;
5        break;
6    case 1: op1; // if x=1
7         op2;
8         break;
9    default: op1; // By default
10 }
```

break — обязательно в конце, так как **case 0** — это метка (такое поведение берет корни из Assembler) для перехода. Как только значение x будет подходящим, программа прыгнет в соответствующую метку, но что будет после выполнения основного кода? Дальше она просто пойдет по остальным, ведь метка не накладывает никаких условий для входа (она просто обозначает кусок программы). Именно поэтому необходим **break** — он завершает выполнение **switch**.