Информатика: семестр 3

Программирование на C++

Конспекты лекций

Лектор: Хирьянов Т.Ф.

17 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**.

Лекция 2

Типы в C++

2.1 Строгая статическая типизация в C++

Проверка типов происходит в момент компиляции.

Код 2.1: Неправильное присваивание

```
1 int x;
2 x = "Hello"; // won't work
```

Помимо этого, присваивание является арифметической операцией (делает дело и возвращает число).

2.2 Представление целых чисел в C++

Существует три способа хранить целые отрицательные числа в памяти компьютера:

1. Прямой ход. Сделаем первый бит знаковым. 0 — знак минус, 1 — знак плюс. Недостатком будет то, что данное число можно интерпретировать как беззнаковое. Пример: $10010011_2 = -19_{10}$. Есть еще одна проблема. 0 трактуется дважды: $-0 = 0$

2. Обратный ход. Мы инвертируем исходное число: 10010011 $\rightarrow$ 11101100. Просто поменяли 0 на 1 и 1 на 0 (это называется побитовое отрицание).

3. Дополнительный код. Обратный код, к которому прибавлена единица. В чем profit: 11101101 $= -19$. Попробуем к нему прибавить число 31:

31	00011111
+ -19	11101101
12	00001100

   Обратите внимание: здесь ушла единица, которая возникла слева при сложении

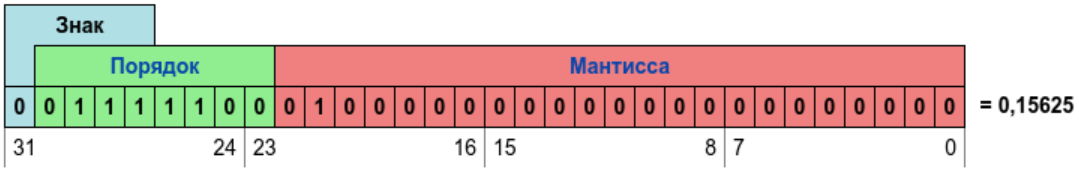
(от увеличения старшего разряда). В этом и заключается вся прелесть дополнительного кода: можно использовать тот же сумматор для двух чисел.

Помимо этого есть *тупоконечное* и *остроконечное* представление. Когда число сохраняется в память, его биты уходят в память в определенном порядке. В тупоконечном порядке сначала в памяти идут старшие разряды, т.е., если мы будем "читать"содержимое ОЗУ слева направо сверху вниз, то все будет идти в правильном порядке. В остроконечном порядке - наоборот. В компьютерах Intel, например, используется остроконечная система.

2.3 Хранение чисел с плавающей точкой

Мы можем условиться: первые биты брать как целую часть, дальше - дробную. Поскольку любое число можно представить бесконечной периодической двоичной дробью, мы сможем представить все числа.

IEEE 754



Стандарт для хранения чисел с плавающей точкой. Можно почитать здесь: вики.
Числа одинарной точности с плавающей запятой обеспечивают относительную точность 7-8 десятичных цифр в диапазоне от 10^{-38} до примерно 10^{38}. Для вычисления показателя степени из восьмиразрядного поля порядка вычитается смещение порядка равное $127_{10} = 7F_{16} = 01111111_2$ (то есть, $01111100_2 - 01111111_2 = 124_{10} - 127_{10} = -3_{10}$). Так как в нормализованной двоичной мантиссе целая часть всегда равна единице, то в поле мантиссы записывается только её дробная часть. Для вычисления мантиссы к единице добавляется дробная часть мантиссы из 23-х разрядного поля дробной части мантиссы $1,01000000000000000000000_2$. Число равно произведению мантиссы со знаком на двойку в степени порядка $= 1,01_2 * 2_{10}^{-3_{10}} = 101_2 * 2_{10}^{-5_{10}} = 5_{10} * 2_{10} - 5_{10} = 0,15625_{10}$.
(Взято из другой статьи)

2.4 Явное и неявное преобразование типов

Код 2.2: Неявное преобразование

```
1 double x;
2 int a = 2, b = 5;
```

```
3 x = b/a; // NOT DOUBLE without casting
4
5 x = double(b)/double(a); // explicit casting
6
7 x = b; // auto convert b to double, then will write to x
```

В коде выше используется неявное преобразование: дает возможность писать код гибче, однако требует осторожнее (так, в строке 3 используется *целочисленное* деление).

Код 2.3: Не всегда удается преобразовать типы

```
1 std::string s;
2 s = std::string(x); //won't work
```

2.5 Типы целых чисел

В C++ есть несколько типов целых чисел:

1. **char** - код символа (по умолчанию 1 байт)
2. **int** - целое число (сколько байт - не известно).

Не стоит быть уверенным, что **char** - 1 байт, на разных процессорах по-разному. В C++ так устроено: язык заточен под процессор, на котором он будет выполняться. Для обоих типов можно указать **signed** и **unsigned** - знаковый тип или нет.

Код 2.4: Пример доп. указателей к типам

```
1  signed int x;
2  unsigned int y;
3  long int z;
4  // size
5  sizeof(int); // about 4
6  long int // 4 bytes, too
7  unsigned long long int // 8 bytes now
8  short int // 2 bytes, wow
9
10 int8_t x8;
11 int16_t x16;
12 int32_t x32;
13 int64_t x64;
14 // They all has fixed size, dude
```

. Стоит быть внимательным: сравнение для знаковых и беззнаковых может пойти не так, как хотелось бы. Пример: $-1 > 50$ - верное равенство, если -1 имеет тип **signed**, а 50 - **unsigned**. Проблема заключается в том, что -1 соответствует максимальному числу из **unsigned**.

2.6 Логические операции

Работают над переменными типа **bool**.

не	$!A$
и	$A\&\&B$
или	$A\|\|B$

Здесь почти то же самое, что в Python, лишь синтаксис другой.

2.7 Битовые операции C++

Как можно поменять две переменные без третьей? Для этого могут помочь логические операции. В C++ их несколько:

1. Инверсия – x – работает над *любым* числом, не обязательно **bool**. Заменяет все ноли и единички в представлении числа на противоположные.
2. Битовое **"и"** – $x\&y$ – умножает биты в двоичном представлении двух чисел. Иногда битовое "и"называют *наложением маски*.
3. Битовое **"или"** – $x|y$ – складывает биты в двоичном представлении двух чисел.
4. xor - x∧y. С его помощью можно как раз осуществить обмен переменных:
   x = x ∧ y
   y = x ∧ y
   x = x ∧ y

2.8 Автоматические типы переменных

Код 2.5: Автоматический тип переменной

```
#include<iostream>

int get_x(); // 'auto x' will be int

double get_x(); // now 'auto x' will be double
// and so on...

int main()
{
  auto x = get_x();
  return 0;
}
```

Код будет присваивать переменной x различные типы, однако этот трюк будет работать, только если компилятору сказать использовать современный стандарт (например, C++11)!

2.9 Простейшие массивы

В C++, как и во многих языках, существуют массивы.

Код 2.6: Пример объявления простейшего массива

```
1  const int N_max = 1000;
2
3  int main()
4  {
5    int A[N_max];
6    int B[N_max] = {1, 2, 3};
7
8    // Let's iterate!
9    for(int i = 0; i < N_max; i++)
10     A[i] = i*i;
11
12 }
```

При объявлении массива надо указать его размер. По массиву можно итерироваться через обращение к элементу (точно так же, как в Питоне): A[1] – второй элемент и так далее; нумерация идет с *нуля*.
Подробнее о массивах будет рассказано на следующей лекции.