3.2. Видимость и инициализация переменных

3.2.1. Видимость переменной

Ссылаться на переменную (и использовать) можно только после того, как она была объявлена:

```
cout << x;
int x = 5;
```

Такой код даже не скомпилируется. Компилятор выдаст ошибку «'x' was not declared in this scope». Если же поменять строчки местами, программа заработает.

Другой пример:

```
{
  int x = 5;
  {
    cout << x;
  }
  cout << x;
}
cout << x;
```

Здесь переменная `x` определена внутри операторных скобок, и в трех местах кода имеет место попытка вывести ее на экран. При этом программа не компилируется: компилятор указывает на ошибку при попытке вывести на экран `x` за пределами первых операторных скобок. Если эту строчку закомментировать, программа успешно скомпилируется:

```
{
  int x = 5;
  {
    cout << x;
  }
  cout << x;
}
//cout << x;
```

Таким образом, переменные в C++ видны только после своего объявления и до конца блока, в котором были объявлены. Например, следующий код с условным оператором не скомпилируется:

```
if (1 > 0) {
  int x = 5;
}
cout << x;
```

Это связано с тем, что переменная объявлена внутри тела условного оператора.

То же самое имеет место для цикла while:

```
while (1 > 0) {
  int x = 5;
}
cout << x;
```

И для цикла for:

```
for (int i = 0; i < 10; ++i) {
  int x = 5;
}
cout << x;
```

Может возникнуть вопрос: видна ли переменная `i`, которая была объявлена как счетчик цикла:

```
for (int i = 0; i < 10; ++i) {
  int x = 5;
}
cout << i;
```

Оказывается, что она также не видна.

Еще один пример:

```
string s = "hello";
{
  string s = "world";
  cout << s << endl;
}
cout << s << endl;
```

Здесь переменная `s` определена как внутри операторных скобок, так и вне их. Программа компилируется и выводит:

```
world
hello
```

Однако использование одинаковых имен, хоть не вызывает ошибку компиляции, считается плохим стилем, так как усложняет понимание кода и увеличивает вероятность ошибиться.

3.2.2. Инициализация переменной

Пусть функция PrintInt объявляет переменную типа int и выводит на экран:

```
void PrintInt() {
  int x;
  cout << x << endl;
}
```

Пусть также определена функция PrintDouble, в которой определяется переменная типа double и ей сразу присваивается значение. Переменная также выводится на экран.

```
void PrintDouble() {
  double pi = 3.14;
  cout << pi << endl;
}
```

Пусть в main сначала вызывается функция PrintInt, а за ней — PrintDouble:

```
PrintInt();
PrintDouble();
```

Такая программа компилируется без ошибок. Интерес представляет то, какое значение `x` будет выведено на экран, поскольку значение этой переменной установлено не было. Можно предположить, что по умолчанию значение переменной `x` будет равной 0. Программа выводит:

```
0
3.14
```

Теперь рассмотрим другую ситуацию, когда после функции PrintDouble еще раз вызывается PrintInt:

```
PrintInt();
PrintDouble();
PrintInt();
```

В результате работы программы:

```
0
3.14
1074339512
```

Вместо нуля при втором вызове PrintInt выводится какое-то большое странное число, а не ноль.

Этот пример показывает, что значение неинициализированной переменной не определено. У этого есть рациональное объяснение. Автор C++ руководствовался принципом «zero overhead principle»[1]:

[1] «Не платить за то, что не используется.»

```
int value; // мне все равно, что будет в переменной value
int value = 0; // мне необходимо, чтобы в value был ноль
```

Отсюда следует вывод: переменные нужно инициализировать при их объявлении. Так получится защититься от ситуации, что в переменной неожиданно окажется мусор, а не ожидаемое значение.

3.2.3. Инициализация переменной при объявлении: примеры

Следующая функция печатает, является ли переданное в качестве параметра число четным:

```
void PrintParity(int x) {
  string parity;

  if (x % 2 == 0) {
    parity = "even";
  } else {
    parity = "odd";
  }

  cout << x << " is " << parity;
}
```

В этом случае, казалось бы, не получается инициализировать переменную при ее объявлении. Однако этого можно добиться с помощью тернарного оператора:

```
void PrintParity(int x) {
  string parity = (x % 2 == 0) ? "even": "odd";
  cout << x << " is " << parity;
}
```

Следующая функция выводит на экран, является ли число положительным, отрицательным или нулем:

```
void PrintPositivity(int x) {
  string positivity;

  if (x > 0) {
    positivity = "positive";
  } else if (x < 0) {
    positivity = "negative";
  } else {
```

```
    positivity = "zero";
  }

  cout << x << " is " << positivity;
}
```

В этом случае также хочется добиться инициализации переменной при ее объявлении.

Для этого можно вынести часть кода в отдельную функцию:

```
string GetPositivity(int x){
  if (x > 0) {
    return "positive";
  } else if (x < 0) {
    return "negative";
  } else {
    return "zero";
  }
}
```

В этом случае переменная positivity может быть инициализирована в месте ее объявления:

```
void PrintPositivity(int x){
  string positivity =  GetPositivity(x);
  cout << x << " is " << positivity;
}
```