Московский государственный университет им. М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Алгоритмы и алгоритмические языки
Лекция 6

25 сентября 2019 г.

Переменные

- Переменная = тип + имя + *значение*
- Каждая переменная является *объектом* программы
- Ключевые слова (C89 — 32, C99 — C89 + 5) не могут быть именами переменных
- Объявление переменной: `type name [, name, name];`
  Можно задать класс памяти и начальное значение переменной
  `int a, b; unsigned c = 2019;`

Область действия (scope) переменных

Переменная может быть объявлена

1. внутри функции или блока (локальная),
2. в объявлении функции (параметр функции),
3. вне всех функций (глобальная).

Область действия (видимости)

локальной переменной — блок, в котором она объявлена (C99 — начиная со строки объявления),

глобальной переменной — программный файл, начиная со строки объявления.

В одной области действия нельзя объявлять более одной переменной с одним и тем же именем.

Классы памяти и области действия: пример

```
#include <stdio.h>
int count;              /* global */
void func (void) {
  int count;            /* auto */
  count = count - 2;
}
static int mult = 0;    /* static */
int sum (int x, int y) {
  count++;
  return (x + y) * (++mult);
}
int main (void) {
  register int s = 0;   /* register */
  count = 0;
  s += sum (5, 7);
  func ();
  printf ("Sum is %d, func is called %d times\n", s, count);
  return 0;
}
```

Инициализация переменных

При объявлении переменной `int x = 42;`

- автоматические переменные инициализируются каждый раз при входе в соответствующий блок;
- если нет инициализации, значение соответствующей переменной **не определено**!
- глобальные и статические инициализируются только один раз в начале работы программы;
- если нет инициализации, они обнуляются компилятором;
- внешние переменные инициализируются только в том файле, в котором они определяются;
- при инициализации переменной типа с квалификатором `const` она является константой и не может изменять свое значение.

Литералы

Литералы задают константу (фиксированное значение).

- символьные константы `'c'`, `L't'`, `'\0x4f'`, `'\040'`
  тип символьной константы — `int`!
- целые константы `100`, `-34l`, `1000U`, `999llu`
- константы с плавающей точкой `11.123F`, `4.56e-4f`, `1.0`, `-11.123`, `3.1415926l`, `-6.626068e-34L`
  тип вещественной константы без суффикса — `double`!
- шестнадцатеричные константы `0x80` (128)
  вещественные 16-ричные: `0x3.ABp3` $3\frac{171}{256} \times 8 = 29.34375$
- восьмеричные константы `012` (10)
- строковые константы `"a"`, `"Hello, World!"`, `L"Unicode string"`
- специальные символьные константы `\n`, `\t`, `\b`

Операции над целочисленными данными

Арифметические

одноместные: изменение знака (`-`), одноместный плюс (`+`)
двухместные: сложение (`+`), вычитание (`–`), умножение (`*`), деление нацело (`/`), остаток от деления нацело (`%`)
$(a/b) \times b + (a\%b) == a$

Отношения (результат 0/1 типа `int`)

больше (`>`), больше или равно (`>=`), меньше (`<`), меньше или равно (`<=`)

Сравнения (результат 0/1 типа `int`)

равно (`==`), не равно (`!=`)

Логические

отрицание (`!`), конъюнкция (`&&`), дизъюнкция (`||`)
ложное значение — 0, истинное — любое ненулевое
«ленивое» вычисление `&&` и `||`

Операции присваивания

Побочные эффекты: изменение объекта, вызов функции

`lvalue = rvalue`

- `lvalue` — выражение, указывающее на объект памяти
- `rvalue` — выражение, генерирующее значение
  Пример: `a = b = c = d = 0;`

Укороченное присваивание: `lvalue op= rvalue`, где `op` — двухместная операция
Пример: `a += 15;`

Инкремент и декремент: `++` и `--`
префиксные и постфиксные

Последовательное вычисление: операция запятая `,`
Пример: `a = (b = 5, b + 2);`

Точки следования

Побочные эффекты: изменение объекта, вызов функции

Точка следования (sequence point): момент во время выполнения программы, в котором все побочные эффекты предыдущих вычислений закончены, а новых — не начаты

- первый операнд `&&`, `||`, `,`
- окончание *полного* выражения (full expression)
- между вычислением фактических параметров и вызовом функции

Между двумя точками следования изменение значения переменной возможно не более одного раза[1].

```
(a=2) + (a=3)
i++ + ++i
```

[1] В последних стандартах терминология несколько иная (sequenced before, unsequenced, indeterminately sequenced): точка следования влечёт частичный порядок, его отсутствие делает возможным любые варианты.

Форматный ввод-вывод

```
#include <stdio.h>
int main (void) {
  int s = 0;
  int a, b;
  scanf ("%d%d", &a, &b);
  s += a + b;
  printf ("Sum␣is␣%d\n", s);
  return 0;
}
```

Спецификаторы ввода-вывода

спецификатор	печатает/считывает
`%d`, `%ld`, `%lld`	число `int`, `long`, `long long`
`%u`, `%lu`, `%llu`	число `unsigned`, `unsigned long`, `unsigned long long`
`%f`, `%Lf`	печатает `double`, `long double`
`%f`, `%lf`, `%Lf`	считывает `float`, `double`, `long double`
`%c`	символ (`char`)

`%4d`: вывести число типа `int` минимум в четыре символа

`%.5f`: вывести число типа `double` с пятью знаками

`%%`: напечатать знак процента

Функция `scanf` возвращает количество удачно считанных элементов

Пример Си-программы

```
/* Solving a quadratic equation */
#include <stdio.h>
#include <math.h>
int main (void) {
  int a, b, c, d;
  /* Input coefficients */
  if (scanf ("%d%d%d", &a, &b, &c) != 3) {
    printf ("Need␣to␣input␣three␣coefficients!\n");
    return 1;
  }
  if (!a) {
    printf ("That's␣not␣quadratic!\n");
    return 1;
  }
 <...>
```

Пример Си-программы

```
<...>
  d = b*b - 4*a*c;
  if (d < 0)
    printf ("No solutions\n");
  else if (d == 0) {
    double db = -b;
    printf ("Solution: %.4f\n", db/(2*a));
  } else {
    double db = -b;
    double dd = sqrt (d);
    printf ("Solution 1: %.4f, solution 2: %.4f\n",
            (db+dd)/(2*a), (db-dd)/(2*a));
  }
  return 0;
}
```