2. Типы данных, операции и выражения

Владимир Верстов

Б. Керниган, Д. Ритчи. Язык программирования C

Имена переменных

- Имя может состоять из букв, цифр и символа “_”
- Имя должно начинаться с буквы или символа “_”
- Буквы в разных регистрах различаются: x и X - разные переменные
- Принято записывать имена переменных строчными буквами, а констант - прописными
- Значащим считается как минимум 31 символ из имени
- Ключевые слова: if, else, return, int, float, double, while, for и др. зарезервированы - их нельзя использовать в качестве имен переменных

Типы данных и их размеры

- **char** - 1 байт, содержит 1 символ из локали системы
- **int** - целое число, обычно имеет типовой размер для целого числа в системе
- Для **int** есть модификаторы - **short** и **long**. Всегда соблюдается правило: **short <= int <= long**
- Наличие **знака** у **int** регулируют модификаторы: **signed** и **unsigned**
- **float** - число с плавающей точкой
- **double** - число с плавающей точкой с двойной точностью
- Для **double** есть модификатор **long**

Типовые размеры данных
limits.h

Тип данных	Размер, байт
char	1
short	2
int	4
long	8
float	4
double	8
long double	16

Целочисленные константы

Тип данных	**Пример**
int	1234
unsigned int	1234u или 1234U
long	1234L или 1234l
unsigned int	1234UL или 1234ul

Способы записи целых чисел

Способы записи числа 31

- 31 - десятичная
- 037 - восьмеричная (начинается с 0)
- 0x1F или 0X1F - шестнадцатеричная (начинается с 0x или 0X)
- К константам в восьмеричной и шестнадцатеричной системе счисления тоже применены суффиксы U и L

Символьная константа

- … - это целое число в форме одного символа в одинарных кавычках, например 'x' или '\n'

- Например, `'0' == 48`

- Запись битовой последовательности длиной 1 байт:

  - '\ooo', где ooo - от 1-й до 3-х восьмеричных цифр

  - '\xhh', где hh - 2 шестнадцитиричные цифры (0…9, A…F)

Управляющие последовательности (Escapes)

Последовательность	**Значение**
\a	подача звукового сигнала
\b	backspace (назад и затереть)
\f	прогон страницы
\n	конец строки
\r	возврат каретки
\t	горизонтальная табуляция
\v	вертикальная табуляция
\\	обратный слэш (косая черта)
\?	вопросительный знак
\’	одинарная кавычка
\”	двойная кавычка
\ooo	восьмеричное число
\xhh	шестнадцатиричное число

Константное выражение

- … - это выражение, которое содержит только константы. Такие выражения могут быть вычислены (и чаще всего вычисляются во время компиляции)

- Пример:

```
#define MAXLINE 1000

char line[MAXLINE + 1];
```

Строковая константа

- … или **литерал** - это последовательность из нескольких (или ни одного) символов в двойных кавычках, например: “hello, world”

- литерал в C - массив символов, который завершается символом \0

Длина строки

```c
#include "stdio.h"

int strlen(char s[]);

int main() {
   printf("%d\n", strlen("hello, world"));
   return 0;
}

/* Функция реализована библиотеке string.h */
int strlen(char s[]) {
   int i;
   i = 0;
   while (s[i] != '\0') {
      ++i;
   }
   return i;
}
```

Перечисления

- … - это список целочисленных констант. По умолчанию первая константа получает значение 0, вторая 1 и т.д.
- Примеры:

```
enum boolean { NO, YES };

enum escapes { BELL = '\a', TAB = '\t', VTAB = '\v' };

enum months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL,
   AUG, SEP, OCT, NOV, DEC };
```

Объявления переменных

```
int lower, upper, step;
char c, line[1000];
```

```
int lower;
int upper;
int step;
char c;
char line[1000];
```

```
char escape = '\\';
int i = 0;
int limit = MAXLINE + 1;
```

Неизменяемые (immutable) переменные

- Изменение значений переменных или параметров функций не допустимо, если указан модификатор **const**
- Примеры:

```
const double e = 2.7182818281845905;
const char message = "Error";

int strlen(const char s[]);
```

Арифметические операции

Оператор	**Операция**
+	Сложение
-	Вычитание
*	Умножение
/	Деление. При делении целых чисел дробная часть отбрасывается
%	Получение остатка. Неприменимо к float и double

Отношения и логические операции

Оператор	**Операция**
>	больше
>=	больше или равно
<=	меньше или равно
<	меньше
==	равно
!=	не равно
&&	логическое И
||	логическое ИЛИ
!	логическое НЕ или операция отрицания

Значение логического выражения

- По определению числовое значение логического выражения или сравнения равно 1, если выражение истинно, и 0 — если ложно

- Операция ! превращает ненулевой операнд в 0, а нулевой в 1

- Пример:

```
if (!valid)

if (valid == 0)
```

Преобразование типов

- Если операнды некой операции имеют разные типы, то они преобразуются к одному типу
- Основное правило преобразования - “узкие” типы приводятся к более “широким” без потери данных
- Если требуется приведение “широкого” типа к “узкому”, то компилятор выдает предупреждение о возможной потере данных
- Выражения не имеющие смысла запрещены. Например, использование float и double как индексов в массиве

Преобразование строки в целое число

```
/* Для общего случая функция реализована в
   в библиотеке stdlib.h */
int atoi(char s[]) {
   int i, n;
     n = 0;
     for (i = 0; s[i] >= '0' && s[i] <= '9'; ++i)
          n = 10 * n + (s[i] - '0');
     return n;
}
```

Функция для работы с символами

```
/* Преобразование символа к нижнему регистру
   Для общего случая tolower из ctype.h */
int lower(int c) {
    if (c >= 'A' && c <= 'Z' )
        return c + 'a' - 'A';
    else
        return c;
}

/* Функция определяет является ли с цифрой:
   1 - да, 0 -нет
   Для общего случая isdigit из ctype.h */
int digit(int c) {
   return c >= '0' && c <= '9';
}
```

Правила приведения типов

- Если один из операндов long double, то другой преобразуется в long double
- В противном случае, если один из операндов double, то другой преобразуется в double
- В противном случае, если один из операндов float, то другой преобразуется в float
- В противном случае char и short преобразуются в int
- Затем, если среди операндов есть long, то другой преобразуется в long

Особенности приведения типов

- Возникает множество сложностей при преобразовании знаковых типов. Например:
  -1L < 1U, но -1L > 1UL

- При приведении большего целого в меньший путем отбрасывания старших битов

- float не приводится к double

- При приведении double к float значение числа усекается или округляется

- float используется при необходимости экономить память

- При приведении float или double к целому отбрасывается дробная часть

Явное приведение типов

- Операция приведение типов

```
(имя-типа) выражение
```

- Приведение типа создает новое значение нужного типа, не изменяя исходную переменную

```
#include "math.h"

int n = 4;

double root = sqrt((double) n);
```

Инкремент и декремент

- Начальные значения переменных:

```
int n = 5;
int x;
```

- Префиксная форма инкремента

```
x = ++n; // x = 6, n = 6
```

- Постфиксная форма инкремента

```
x = n++; // x = 5, n = 6
```

- Префиксная форма декремента

```
x = --n; // x = 4, n = 4
```

- Постфиксная форма декремента

```
x = n--; // x = 5, n = 4
```

Поразрядные операции

Оператор	**Операция**
&	поразрядное И
|	поразрядное ИЛИ
^	поразрядное исключающее ИЛИ
<<	сдвиг влево
>>	сдвиг вправо
~	одноместное поразрядное дополнение до 1

Пример

```
A = 186;
B = 107;
X = A & B; // X = 42
```

Разряд	7	6	5	4	3	2	1	0
A	1	0	1	1	1	0	1	0
B	0	1	1	0	1	0	1	1
X	0	0	1	0	1	0	1	0

Таблица истинности

A	Операция	B	Результат
0	&	0	0
0	&	1	0
1	&	0	0
1	&	1	1
0	|	0	0
0	|	1	1
1	|	0	1
1	|	1	1
0	^	0	0
0	^	1	1
1	^	0	1
1	^	1	0

Примеры

A	Операция	B	Результат
0101 0011	&	1001 0010	0001 0010
0101 0011	|	1001 0010	1101 0011
0101 0011	^	1001 0010	1100 0001
0001 0011	<<	2	0100 1100
0101 0011	>>	3	0000 1010
	~	1001 0010	0110 1101

Извлечение бит

```
/* Извлечение n-бит,
   начиная с позиции p,
   из числа x */

unsigned getbits(unsigned x, int p, int n) {
   return (x >> (p+1-n)) & ~(~0 << n);
}
```

Операции с присваиванием

- Пример

```
int i = 1;
// ...
i = i + 2; // i = 3
i += 2; // i = 5
```

- Псевдокод

```
выражение1 = выражение1 операнд выражение2
выражение1 операнд= выражение2
```

- Допустимые операции: + - * / % << >> & ^ |

Подсчет установленных бит

```
int bitcount (unsigned x) {
   int b;
   for (b = 0; x != 0; x >>= 1)
      if (x & 01)
         b++;
   return b;
}
```

Условные выражения

```
int a, b, c;
// ...
if (a > b)
   c = a;
else
   c = b;

c = a > b ? a : b;
```

```
if (выражение1)
   выражение2
else
   выражение3

выражение1 ? выражение2 : выражение3
```

Приоритет операций

Операции	**Порядок вычисления**
() [] -> .	Слева направо
! ~ ++ -- + - * & (тип) sizeof	Справа налево
* / %	Слева направо
+ -	Слева направо
<< >>	Слева направо
< <= > >=	Слева направо
== !=	Слева направо
&	Слева направо
^	Слева направо
|	Слева направо
&&	Слева направо
||	Слева направо
?:	Справа налево
= += -= *= /= %= &= ^= |= <<= >>=	Справа налево
,	Слева направо

Приоритеты операций

- Главное правило: забыли или запутались в приоритетах — используйте скобки ()