Типове, оператори и изрази

Пламен Танов
Ненко Табаков
Мартин Вачовски
Технологично училище „Електронни системи“
Технически университет – София

версия 0.1

Типове в C

Типът дава следната информация за променливата:

- стойностите, които може да присвоява
- действията, които могат да се извършват
- паметта, която заема

Имена на променливи

- букви (вкл. _) и цифри. Задължително започва с буква (вкл. _)!
- разлика между малки и главни букви
- обикновено променливите са с малки букви, константите - с главни, различните думи в името на една п-ва се отделят с _
- имената им не могат да съвпадат с тези на запазените думи (if, else и т.н.)
- първите 31 символа са значими, 6 (case unsensitive) - за extern п-ви

```
int fahr, celsius;
float value, new_value;
char c;
```

Типове и размер

- **`char`** един байт, съдържащ символ от локалната кодова таблица
- **`int`** целочислен тип, размерът му зависимост от системната архитектура (най-малко 16 бита)
- **`float`** число с плаваща запетая и единична точност
- **`double`** число с плаваща запетая и двойна точност

Допълнителни определители

- **short** най-малко 16 бита
- **long** най-малко 32 бита
- **short** ≤ **int** ≤ **long**

```
short int sh;                    short sh;
long int counter;          ≡     long counter;
long double distance;            long double distance;
```

- **signed** със знак
- **unsigned** без знак (2^n)

```
unsigned char c;          //0 ÷ 255
signed char q;            //-128 ÷ 127    (-2^(n-1) ÷ 2^(n-1)-1)
unsigned int count;       //0 ÷ (2^n-1)
unsigned long int stars;
```

<limits.h>, <float.h> (в директорията include на компилатора) - съдържат константи с големините и други характеристики на компилатора

Константи

Към дефиницията на всяка променлива може да се прилага модификаторът **`const`**, който показва, че стойността на променливата няма да се променя. Това предпазва от нежелана промяна на дадена променлива и до създаването на по-ефективен код:

```
const double E  = 2.71828182845904523536;
const double PI = 3.14159265358979323846;
const char[] MESSAGE = "warning: ";
```

В самия код може да се изписват константи:

```
int                :  1234
long               :  123456789L, 1234567891
unsigned           :  1234U
unsigned long      :  123456789UL

float              :  123.4f, 123.4F
double             :  123.4, 1e-2
long double        :  123.4L
```

Константи

В самия код може да се изписват константи и в различни бройни системи:

```
octal          : 037;
hexadecimal    : 0x1f, 0XFF, 0XFU
```

Символни константи:

```
char     : '1' ≡ 49 ≡ '\061' ≡ '\x31'
string   : "hello, world" ≡ "hello, " "world"
"" - празен стринг

"1" ≠ '1'
```

Примери

```
char c;
c = getchar();
if (c == '1') {
    printf("one");
}
```

```
char c;
c = getchar();
if (c == 49) {
    printf("one");
}
```

```
char c;
c = getchar();
if (c == '\061') {
    printf("one");
}
```

```
long x;
unsigned long ul;

x  = 123L;
ul = 0XFUL;
```

Изброени константи

Това е списък от наименовани целочислени стойности

```
enum boolean { NO, YES }; //NO = 0, YES = 1
enum escapes { BELL = '\a', BACKSPACE = '\b', TAB = '\t'};

enum months { JAN = 1, FEB, MAR, APR, MAY, JUN,
JUL, AUG, SEP, OCT, NOV, DEC }; // FEB = 2, MAR = 3, т.н.

enum months m;
enum boolean is_running;
...

if (m == MAY) { ...
```

Декларация на променливи

- Всяка променлива трябва да се декларира преди да бъде ползвана
- Чрез декларацията се оказва типът на променливата

```
#define END 3

int lower = 0, upper, step;
char c, line[1000];

char esc = '\\';
int i = 0;
const int LIMIT = END + 1;
float eps = 1.0e-5;

const char MESSAGE[] = "warning: ";
```

Аритметични оператори

- `+`
- `-`
- `*`
- `/` целочислено или реално деление
- `%` деление по модул, само между целочислени

```
if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)
    printf("%d is a leap year\n", year);
else
    printf("%d is not a leap year\n", year);
```

Сравнителни оператори

- >
- >=
- <
- <=
- == проверка за равенство
- != проверка за разлика

Логически оператори

- `&&` логическо И (AND)
- `||` логическо ИЛИ (OR)

изпълняват се от ляво на дясно и изчислението спира веднага след като резултатът бъде установен:

```
for (i=0; i < lim-1 && (c=getchar()) != '\n' && c != EOF; ++i)
    s[i] = c;
```

- `!` логическо НЕ

```
if (!valid) {    //вместо if (valid == 0)
```

Преобразуване на типовете$_1$

- Преди да се изпълни произволен оператор неговите операнди трябва да са от един и същи тип, това обикновено става като „по-малкият“ тип се преобразува към „по-големия“:

$$\texttt{float + int} \rightarrow \texttt{float + float}$$

При **`signed`** типове важи правилото:

- Ако някой от операндите е **`long double`**, преобразуваме и другия към **`long double`**.
- Иначе, ако някой от операндите е **`double`**, преобразуваме и другия към **`double`**.
- Иначе, ако някой от операндите е **`float`**, преобразуваме и другия към **`float`**.
- Иначе, преобразуваме **`char`** и **`short`** към **`int`**.
- След което ако някои от операндите е **`long int`**, преобразуваме и другия към **`long int`**.

Преобразуване на типовете$_2$

- По друг начин казано за **signed** променливи: ако някой от операндите е **long double**, преобразуваме и другия към **long double**. Ако няма операнд от този тип - последователно се проверява за наличието на тип **double** или **float** и другият операнд се преобразува към него. В противен случай (когато няма нито един **long double, double, float**) **char** и **short** се преобразуват към **int**. На края, ако някои от операндите е **long int**, преобразуваме и другия към **long int**.

Пример

```
long double + double → long double + long double
long double + int → long double + long double
long double + char → long double + long double

float + double → double + double
float + int → float + float
float + long int → float + float

char + int → int + int
short int + int → int + int
char + long int → int + long int → long int + long int
short int + long int → int + long int → long int + long int

и т.н.
```

Преобразуване на типовете 3

- Преобразуването между **signed** и **unsigned** типове зависи от системната архитектура:
  -1L < 1UL, когато **1UL** се преобразува към **signed long**
  -1L > 1UL, когато **-1L** се преобразува към **unsigned long**

- Когато се подават аргументи към функция:
  char и short → int; float → double:

```
double sqrt(double);   //от <math.h>
...
root2 = sqrt(2);       //автоматично 2 → 2.0
```

- Явно преобразуване:

```
int n = 2;
root2 = sqrt((double)n);     //явно 2 → 2.0
```

Оператори за увеличаване и намаляване

++ – увеличава стойността на променливата с 1

-- – намалява стойността на променливата с 1

действие	нова стойност на **nc**
`nc = 5;`	5
`++nc;`	6
`nc++;`	7
`--nc;`	6
`nc--;`	5

Пример

```
k = 5;
z = k++;

(k = 6, z = 5)
```

```
k = 5;
z = ++k;

(k = 6, z = 6)
```

```
arr[i] = i++; //ГРЕШНО !!!

(a + j)++; //ГРЕШНО !!! - ++ и -- само на променливи!
```

Пример

```
/* изтрива всички съвпадения на c от s */
void squeeze(char s[], char c) {
    int i, j;
    for (i = j = 0; s[i] != '\0'; i++)
       if (s[i] != c)
          s[j++] = s[i];
    s[j] = '\0';
}
```

≡

```
if (s[i] != c) {
    s[j] = s[i];
    j++;
}
```

```
/* добавя t, към края на s (трябва да има място в s!) */
void strcat(char s[], char t[]) {
    int i, j;
    i = j = 0;
    while (s[i] != '\0')          /* търси края на s */
       i++;
    while ((s[i++] = t[j++]) != '\0'); /* копира t */
}
```

Оператор за присвояване

```
a = (a) OP (x);      ≡     a OP= x;

a = a + 2;                 a += 2;
a = a * (y + 2);     ≡     a *= y + 2;
a = a & 2;                 a &= 2;
```

```
yyval[yypv[p3+p4] + yypv[p1]] += 2;

//много по-разбираемо от колкото в тази форма:

yyval[yypv[p3+p4] + yypv[p1]] =
                yyval[yypv[p3+p4] + yypv[p1]] + 2;
```

Побитови оператори

- Между **`char`**, **`short`**, **`int`** и **`long`** (без значение **`signed`** или **`unsigned`**)

- **`&`** побитово И
- **`|`** побитово ИЛИ
- **`^`** побитово изключващо ИЛИ (сума по модул от 2)
- **`<<`** преместване вляво
- **`>>`** преместване вдясно
- **`~`** инвертиране (т.е. числото

1	1	0	0	1	0	1	0

става:

0	0	1	1	0	1	0	1

)

Примери

- Преместване вляво с 2 (**<<2**):
- **00010100** → **01010000**

- Преместване вдясно с 2 (**>>2**):
- **00010100** → **00000101** (без знак)
- **10010100** → **11100101** (със знак)
- **10010100** → **00100101** (със знак)

не е строго дефинирано при **signed** променливи дали се добавят 0 или 1 (зависи от системната архитектура)!

Примери

```
char set_bit(char c, int bit_no) {
    return c |= (1<<bit_no);
}

int is_bit_set(char c, int bit_no) {
    return (c &= (1<<bit_no)) != 0;
}

char unset_bit(char c, int bit_no) {
    return c &= ~(1<<bit_no);
}
```

Условни изрази

```
expr1 ? expr2 /* expr1 != 0 */ : expr3 /* expr1 == 0 */
```

```
if (a > b)
    z = a;
else
    z = b;
```

≡

```
z = (a > b) ? a : b;
```

Приоритет

```
() [] -> .
! ~ ++ -- + - * (type) sizeof
* / %
+ -
<< >>
< <= > >=
== !=
&
^
|
&&
||
?:
= += -= *= /= %= &= ^= |= <<= >>=
,
```

Унарните **&**, **+**, **-** и ***** (пример: **-k**) имат по-висок приоритет от еквивалентните им бинарни форми (пример: **a-b**).

Примери

```
x = f() + g();
//не е определено дали f() или g() ще се извика първо

printf("%d %d\n", ++n, power(2, n)); // ГРЕШНО !!!

++n;
printf("%d %d\n", n, power(2, n));  //вярно

arr[i] = i++; //ГРЕШНО !!!

(a + j)++; //ГРЕШНО !!! - ++ и -- само на променливи!
```