ТИПОВЕ В C

Ненко Табаков
Пламен Танов
Технологическо училище “Електронни системи”
Технически университет – София

Версия 0.2

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 променливи

```
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;
long int counter;
long double distance;
```

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

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

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

`<limits.h>`, **`<float.h>`** - 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  = 2.71828182845904523 54;
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' ≡ '\0x31'
string  : "hello, world" ≡ "hello, " "world"
"" - празен стринг

"1" ≠ '1'
```

ПРИМЕР

```
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 DIRECTIVE 3

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

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

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

АРИТМЕТИЧНИ ОПЕРАТОРИ

- `+` - събиране
- `-` - изваждане
- `*` - умножение
- `/` - целочислено или реално делене
- `%` - делене 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);
```

ОПЕРАТОРИ ЗА СРАВНЕНИЕ

- **>** – по - голямо
- **>=** – по – голямо или равно
- **<** – по - малко
- **<=** – по – малко или равно
- **==** – еднакво
- **!=** – различно

ЛОГИЧЕСКИ ОПЕРАТОРИ

- `&&` – логическо И
- `||` – логическо ИЛИ

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 ТИПОВЕ

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

  `float + int` $\rightarrow$ **`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`**, преобразуваме и другия към **`long`**.

ПРЕОБРАЗУВАНЕ НА ТИПОВЕ

- Преобразуването между **`signed`** и **`unsigned`** типове зависи от системната архитектура:
  `-1L < 1U`, когато **`1U`** 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 в 2.0

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

ОПЕРАТРОРИ ЗА УВЕЛИЧАВАНЕ И НАМАЛЯВАНЕ

- `++` – увеличава стойността на променливата с 1
- `--` – намалява стойността на променливата с 1

```
nc = 5;
++nc;
nc++;

--nc;
nc--;
```

стойността на nc е 5
стойността на nc е 6
стойността на nc е 7

стойността на nc е 6
стойността на nc е 5

ПРИМЕР

```
k = 5;
z = k++;
//(k = 6, z = 5)


k = 5;
z = ++k;
//(k = 6, z = 6)



arr[n] = ++n; //ГРЕШНО !!!
```

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';
}
```

```
/* 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 */
}
```

ПОБИТОВИ ОПЕРАТРОРИ

- 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

ПРИМЕР

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)!

ПРИМЕР

```
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);
}
```

ОПЕРАТОРИ ЗА ПРИСВОЯВАНЕ

```
a = (a) OP (x);       еквивалентно на       a OP= x;

a = a + 2;                                  a += 2;
a = a * (y + 2);   еквивалентно на          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;
```

УСЛОВНИ ИЗРАЗИ

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


if (a > b)
   z = a;          еквивалентно на      z = (a > b) ? a : b;
else
   z = b;
```

ПРИОРИТЕТИ

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

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

ПРИМЕР

```
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 !!!
```