Reprezentarea intregilor. Operatori pe biti

11 octombrie 2005

Tipuri de bază (fundamentale)

Un *tip*: determină mulțimea valorilor pe care le poate lua o variabilă, și operațiile care pot fi efectuate pe aceste valori.
– reprezentate pe un număr *finit* de octeți ⇒ set *finit* de valori (chiar dacă în matematică, domeniile pentru întregi și reali sunt nelimitate)
⇒ Atenție la depășiri !!!

Limbajul C are doar câteva tipuri de bază.
– `char`: caractere, reprezentate pe 1 octet (8 biți)
  `signed char`: -128..127 `unsigned char`: 0..255
  `char` poate corespunde la oricare din ele, conform standardului
– `int`: numere întregi
– `float`: numere reale (virgulă mobilă), în precizie simplă
– `double`: numere reale, în dublă precizie

Domeniul de valori pentru întregi și reali e dependent de arhitectură (de obicei, corespunde natural cu dimensiunea regiștrilor procesorului)

Reprezentarea binară a numerelor

În memoria calculatorului, numerele se reprezintă în binar (baza 2).

Valoarea unui *întreg fără semn*, cu k cifre binare (biți):
$c_{k-1}c_{k-2}\dots c_1c_0\ _{(2)} = c_{k-1} * 2^{k-1} + \dots + c_1 * 2^1 + c_0 * 2^0$
c_{k-1} = bitul *cel mai semnificativ* (superior)
c_0 = bitul *cel mai puțin semnificativ* (inferior)
Exemplu (pe 8 biti): $11111111_{(2)} == 255$

Întregi cu semn: reprezentati în *complement de 2*
dacă bitul superior e 1, numarul se consideră negativ
valoarea: translatată cu 2^k în jos față de interpretarea fără semn.
$1c_{k-2}\dots c_1c_0\ _{(2)} = -2^{k-1} + c_{k-2} * 2^{k-2} + \dots + c_1 * 2^1 + c_0 * 2^0$
Exemple (pe 8 biți): $11111111_{(2)} == -1$; $10000000_{(2)} == -128$

Adunarea se poate face normal, bit cu bit, indiferent de reprezentare.
Test de depasire: `int x, y, s; scanf("%d%d", &x, &y); s = x + y;`

```
if (x < 0 && y < 0 && s >= 0 || x > 0 && y > 0 && s < 0)
  printf("depasire");
```

Tipuri întregi

Tipul `int` poate primi ca prefix calificatori care specifică:
– *dimensiunea*: `short`, `long` (în C99 și `long long`)
– *semnul*: `signed` (implicit, în caz de omisiune), `unsigned`
Cele două se pot combina; `int` poate fi omis: (ex. `unsigned short`)

Standardul prevede urmatoarele limite:
– `int, short`: $\geq$ 2 octeți, minim $[-2^{15}, 2^{15}-1]$ = [-32768, 32767]
– `long`: $\geq$ 4 octeți, acoperă minim $[-2^{31}$ (-2147483648) , $2^{31}-1]$
– `long long`: $\geq$ 8 octeți, acoperă minim $[-2^{63}, 2^{63}-1]$
– `unsigned` păstrează dimensiunea; între 0 și $2^{8b}-1$ (b = nr. octeți)
– `sizeof(short)` $\leq$ `sizeof(int)` $\leq$ `sizeof(long)` $\leq$ `sizeof(long long)`

Constante definite în `<limits.h>` (valori minime cerute de standard)

```
SHRT_MIN, INT_MIN    -32767          SHRT_MAX, INT_MAX   32768
LONG_MIN          -2147483647        LONG_MAX         2147483647
USHRT_MAX, UINT_MAX   65535          ULONG_MAX        4294967295
```

Constante de tipuri întregi

Constante întregi

– în baza 10: scrise obișnuit; ex. `-5`
– în baza 8: cu prefix cifra zero; ex. `0177` (127 zecimal)
– în baza 16: cu prefix `0x` sau `0X`; ex. `0xA9` (169 zecimal)
– sufix `u` sau `U` pentru `unsigned`, ex. `65535u`
– sufix `l` sau `L` pentru `long` ex. `0177777L`

Constante de tip caracter

– caracter tipăribile, între ghilimele simple: `'0'`, `'!'`, `'a'`
– caracter speciale:

`'\0'`	null	`'\n'`	linie nouă
`'\a'`	alarm	`'\r'`	carriage return
`'\b'`	backspace	`'\f'`	form feed
`'\t'`	tab	`'\''`	apostrof (ghilimea)
`'\v'`	vertical tab	`'\\'`	backslash

– caractere scrise în octal (max. 3 cifre), ex: `'\14'`
– caractere scrise în hexazecimal (prefix `x`), ex. `'\xff'`

Tabela de caractere ASCII

ASCII = American Standard Code for Information Interchange
Caracterele sunt memorate ca și cod numeric = indicele în acest tabel
ex. '0' == 48, 'A' = 65, 'a' = 97, etc.

```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
-----------------------------------------------------
0x0   \0                       \a \b \t \n \v \f \r
0x10:
0x20:     !  "  #  $  %  &  '  (  )  *  +  ,  -  .  /
0x30:  0  1  2  3  4  5  6  7  8  9  :  ;  <  =  >  ?
0x40:  @  A  B  C  D  E  F  G  H  I  J  K  L  M  N  O
0x50:  P  Q  R  S  T  U  V  W  X  Y  Z  [  \  ]  ^  _
0x60:  `  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o
0x70:  p  q  r  s  t  u  v  w  x  y  z  {  |  }  ~
```

– caracterele $<$ 0x20 (spațiu): caractere de control
– caracterele cu cod $>$ 0x7f (127): nu fac parte din setul ASCII (diacritice, diverse variante nationale standardizate de ISO, etc.)

Funcţii de clasificare a caracterelor (în `ctype.h`)

```
int isalnum(int c)   (isalpha(c) || isdigit(c))
int isalpha(int c)   ('A' <= c && c <= 'Z' || 'a' <= c && c <= 'z')
int isblank(int c)   (c == ' ' || c == '\t')
int iscntrl(int c)   // caracter de control, valoare: 0 - 31
int isdigit(int c)   ('0' <= c && c <= '9')
int isgraph(int c)   // caracter tipăribil, exceptând spaţiu
int islower(int c)   ('a' <= c && c <= 'z')
int isprint(int c)   // caracter tipăribil, inclusiv spaţiu
int ispunct(int c)   // tipăribil şi nu alnum şi nu spaţiu
int isspace(int c)   (c == ' ' || c == '\t' || c == '\n' ||
/* "spatii albe" */    c == '\v' || c == '\f' || c == '\r')
int isupper(int c)   ('A' <= c && c <= 'Z')
int isxdigit(int c)  // cifră hexazecimală: 0-9, A-F, a-f
int tolower(int c)   // A - Z -> a - z, restul neschimbat
int toupper(int c)   // a - z -> A - Z, restul neschimbat
```

Dimensiunea tipurilor

Operatorul `sizeof` dă numărul de *octeţi* de memorie ocupaţi de operand (un tip de date sau o expresie – în particular o variabilă).
– dacă operandul e un tip, trebuie pus între (paranteze)
– dacă e o expresie, rezultatul e dat de tipul său (fără evaluarea ei)

```
#include <stdio.h>
int main (void)
{
  printf("char\t\t%d\n",sizeof(char));
  printf("short\t\t%d\n",sizeof(short));
  printf("int\t\t%d\n",sizeof(int));
  printf("long\t\t%d\n",sizeof(long));
  return 0;
}
```

C99 defineste *tipuri intregi de dimensiune data* in `<stdint.h>` :

`int8_t,uint8_t, int16_t,uint16_t, int32_t,uint32_t, int64_t,uint64_t`

Operatori aritmetici

Operatori aritmetici

– operatorii uzuali binari: `+, -, *, /` pentru numere întregi și reale
ATENȚIE: pentru întregi, / înseamnă împărțire cu rest

– operatorul % (numai pentru întregi): modulo (restul la împărțire)

`9/-5==-1  9%-5==4  -9/5==-1  -9%5==-4  -9/-5==1  -9%-5=-4`

(restul are semnul deîmpărțitului)

– operatorul unar - (minus; nu există plus unar).

În expresii aritmetice, caracterele sunt considerate ca și întregi
(indicele caracterului respectiv în tabela ASCII)
Exemple: `'7' - '0' == 7,  'a' + 5 == 'f'`
(cifrele, literele mari, literele mici: 3 grupuri continue in tablela ASCII)
Conversii: `lit_mica = lit_mare + 'a' - 'A';  cifra = valoare + '0';`
Precedența: - unar, apoi * / %, apoi + -

Operatori pe biți

– oferă acces direct la reprezentarea binară a datelor în memorie,
cu posibilități apropiate limbajului de asamblare
– pot fi aplicați doar operanzilor de tipuri întregi, cu sau fără semn

`&`	ȘI bit cu bit	`~`	complementare bit cu bit
`|`	SAU bit cu bit	`<<`	deplasare la stânga
`^`	SAU exclusiv bit cu bit	`>>`	deplasare la dreapta

De obicei testam, setam, resetam sau inversam bitul k dintr-un numar.
Cream un numar (2^k) care are 1 pe bitul k si in rest 0 : `1 << k`

Testare	`n & (1 << k)`	nenul daca bitul k din `n` e 1
Setare	`n = n | (1 << k)`	bitul k din `n` devine 1, ceilalti neschimbati
Resetare	`n = n & ~(1 << k)`	bitul k din `n` devine 0, ceilalti neschimbati
Inversare	`n = n ^ (1 << k)`	bitul k din `n` se schimba, ceilalti nu

Exemple cu operatori pe biti

- `n & 0xF` are ultimii 4 biţi mai puţin semnificativi la fel ca `n`, restul 0
  (pentru `n` fără semn, echivalent cu `n % 16`; `n & 1 == n % 2`)
- `n | 0200` are bitul 7 pe 1, şi toţi ceilalţi biţi la fel ca `n`
- `n ^ 1` are ultimul bit schimbat faţă de `n`, toţi ceilalţi la fel
- `~0 == -1` : intreg *cu semn* cu toţi biţii 1
- `~0u == UINT_MAX` : intreg *fara semn* (`unsigned`) cu toţi biţii 1
- `n < 0` (pentru `signed n`) – echivalent cu testarea bitului de semn
- `~0xf` are ultimii 4 biţi pe 0 şi restul pe 1
- `n << 3` are biţii lui `n` deplasaţi 3 poziţii la stânga, şi ultimii 3 biţi 0
- `n >> 2` are biţii lui `n` deplasaţi 2 poziţii la dreapta; primii 2 biţi devin:
  – 0, daca `n` este `unsigned`, sau `signed` dar nenegativ
  – dependent de implementare (tipic: 1), daca `n` este `signed` negativ
  ATENTIE la efectul deplasarii la dreapta in functie de semn !
- << şi >> : ca şi înmulţiri/împărţiri cu puterile lui 2 (mai eficiente)

Operatori pe biţi (cont.)

Operatori compuşi de atribuire (pt. cei binari): `&=` `|=` `^=` `<<=` `>>=`
`n &= a` e o notatie prescurtata pentru `n = n & a` etc.

Exemple: extragerea unei porţiuni din reprezentarea unui număr:
creem o *mască* (un tipar) în care biţii respectivi sunt pe 0 (sau 1)

`~0 << k`	are ultimii k biţi pe 0, restul pe 1
`~(~0 << k)`	are ultimii k biţi pe 1, restul pe 0
`~(~0 << k) << p`	are k biţi pe 1, începând de la bitul p, şi restul 0

`(n >> p) & ~(~0 << k)`
are pe ultimele poziţii cei k biţi ai lui `n` începând cu bitul p, şi în rest 0

`n & (~(~0 << k) << p)`
are cei k biţi începând cu bitul p la fel ca ai lui `n`, şi restul biţilor pe 0