Introducere în programare
2016-2017

Laborator 4

1. Tablouri şi pointeri, legătura pointerilor cu tablourile
2. Exercitii cu siruri de caractere
3. Să se definească un tip de date pentru reprezentarea numerelor complexe şi să se scrie funcții pentru diferite operații cu astfel de numere.
4. Se dă o listă de persoane şi punctajele obținute de acestea la un concurs. Să se ordoneze descrescător persoanele în funcție de punctaj şi să se afişeze lista ordonată.
5. Să se exemplifice folosirea reuniunilor (union) pentru citirea şi afişarea datelor despre mai multe persoane de ambele sexe.
6. Scrieți o funcție getbits(x, p, n) care să returneze (cadrat la dreapta) câmpul de lungime n biți al lui x, care începe la poziția p.
7. Să se verifice dacă un număr p dat este "deosebit" sau nu. Spunem că p este "deosebit" dacă există q astfel încât p=q+s(q), în care s(q) este suma cifrelor lui q. 13
8. Scrieți o funcție bitcount(n) care să contorizeze numărul de biți pe 1 dintr-un argument întreg.

```
#include int bitcount(unsigned n)
 {
for (int b=0; n; n =>> 1) // este echivalent cu n=n>>1
if (n & 01) b++;
return b;
}
void main()
{
long x;
printf("\nDati x: ");
scanf("%ld",&x);
printf("Avem %d biti pe 1 in %ld",bitcount(x),x);
}
```

9. Să se elimine dintr-o listă de numere reale acele numere care au partea zecimală egală cu zero.

8. Scrieți o funcție `getbits(x, p, n)` care să returneze (cadrat la dreapta) câmpul de lungime `n` biți al lui `x`, care începe la poziția `p`.

Presupunem că bitul 0 este cel mai din dreapta şi că `n` şi `p` sunt valori pozitive sensibile. De exemplu, `getbits(x,4,3)` returnează 3 biți în pozițiile 4, 3 şi 2, cadrați la dreapta.

```
#include <stdio.h>
unsigned getbits(unsigned x, unsigned p, unsigned n)
{
      return ((x >> (p+1-n)) & ~(~0 << n));
}
void main()
{
      int x=13; // 00001101
      printf("\nx=%d",x);
      int y=getbits(x,4,3);
      printf("\ny=%d",y); // 3, adica 011
}
```

Funcția `getbits` are argumente de tip `unsigned`, iar tipul returnat este tot `unsigned`. `unsigned`, alături de `signed`, `short` şi `long` se numesc **modificatori de tip**. Un astfel de modificator schimbă semnificația tipului de bază pentru a obține un nou tip.

Fiecare dintre aceşti modificatori poate fi aplicat tipului de bază `int`.

Modificatorii `signed` şi `unsigned` pot fi aplicați şi tipului `char`, iar `long` poate fi aplicat lui `double`.

Când un tip de bază este omis din declarație, se subînțelege că este `int`. Exemple:

```
long               x;     → int este subînțeles
unsigned char      ch;
signed   int       i;     → signed este implicit
unsigned long int  l;     → nu era nevoie de int
```

Acest exemplu pune în evidență şi un alt aspect al limbajului C, cum ar fi **operatorii pentru manipularea biților**, din care noi am folosit doar câțiva.

Operatorii pentru manipularea biților sunt:

- `&` {I, bit cu bit
- `|` SAU inclusiv, bit cu bit
- `^` SAU exclusiv, bit cu bit
- `<<` deplasare la stanga a bitilor (despre care am mai vorbit)
- `>>` deplasare la dreapta a biților
- `~` complement față de 1 (este un operator unar)

Acestia nu se pot aplica lui `float` sau lui `double`.

Operatorul "{I, bit cu bit" "&" este folosit adesea pentru a masca anumite mulțimi de biți.

De exemplu, `c = n & 0177` pune pe zero toți biții lui `n`, mai puțin bitul 7, (cel mai tare).

Operatorul "SAU, bit cu bit" "`|`" este folosit pentru a pune pe `1` biți:

`x = x | MASK` pune pe `1`, în `x`, biții care sunt setați pe `1` în `MASK`.

Trebuie să distingeți cu grijă operatorii pe biți "&" şi "`|`" de conectorii logici "&&" şi "`||`". De exemplu, dacă `x` este `1` şi `y` este `2`, atunci `x & y` este 0, dar `x && y` este `1`!

Operatorul unar "~" dă complementul față de `1` al unui întreg, adică el converteşte fiecare bit de 1 în 0 şi invers.

Un exemplu de utilizare ar fi: `x&~077`. Aici se maschează ultimii 6 biți ai lui `x` pe 0. De notat că `x&~077` este independent de lungimea cuvântului şi deci, preferabil, de exemplu, lui `x&0177700`, care vede pe `x` ca o cantitate de lungime 16 biți.

În funcția `getbits`, `x` este declarat `unsigned` pentru a ne asigura că la shiftarea spre dreapta, biții vacanți vor fi umpluți cu 0 şi nu cu biții de semn (independent de implementarea de C !). ~0 este cuvântul cu toți biții pe `1`. Prin operatia `~0 << n` creăm o mască cu zerouri pe cei mai din dreapta `n` biți şi `1` în rest; complementându-l apoi cu `~`, facem o mască cu `1` pe cei mai din dreapta `n` biți.

Propunem cititorului să rescrie `getbits` pentru a număra biții de la stânga la dreapta. Un alt exercițiu ar putea fi scrierea unei funcții `right_rot(n,b)` care să rotească întregul `n` la dreapta cu `b` poziții.