Programare Procedurala

Laborator 7

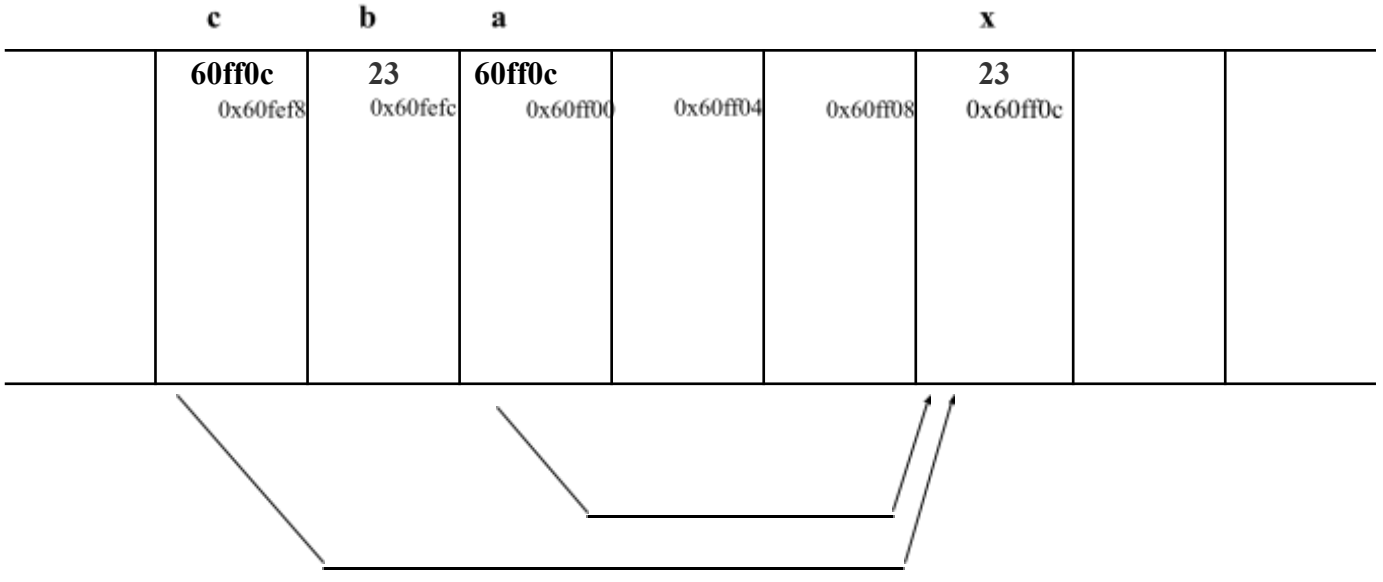
Poineri

- P**ointer** = variabilă care reţine **o adresă de memorie**:
  - adresa unui **tip de date**: elementar, structura, sir de caractere;
  - adresa unei **funcţii:** adresa la care punctul curent de execuţie va sări, în cazul în care acea funcţie este apelată;
  - adresa la care se afla o adresa de memorie;
  - adresa unei zone cu conţinut necunoscut (pointer către void).

Exemple

```
1) int x,……, *a,b,*c;
x=23;
a=&x;        // a primeste adresa lui x
b=*a;        // b memoreaza valoarea ce se afla la adresa continuta in a
c=a;         // c are acelasi continut cu a, adica o adresa (a lui x).

printf("%d\t%d\t%d",x,*a,*c);
printf("%x  %x  %x  %x  %x",&a,&b,&c,c,&x);
```




```
2) int n;

    scanf("%d",&n);
    printf("%x\t%x",&n,&n+1);        // Concluzii?

    int *p;
    p=p+3;                           /* se incrementeaza adresa continuta de p
                                     cu 3*sizeof(int);  */


3) int x,*a,*b;
x=135;
```

```
a=&x;
b=a;
(*a)++;            // rezultatul?         printf("%d   %d", x,*a);
*a++;             // precedenta operatorilor?
printf("%d   %d    %x   %p   %x   %x   %x", *a,*b,a,a,b,&a,&b);
```

- **Pointeri către void:**

```
1) void *p=NULL;  // pointer către void, inițializat la NULL
```

2) Un pointer către void trebuie convertit la un pointer către un tip de date înante de a fi folosit, utilizand operatorul de cast: `(int *)`

```
*p=23;            // invalid use of void expression

// Atunci, convertim întâi la int:

p = malloc(sizeof(int));      // alocăm spațiu pentru un int (va urma)
*((int*)p) = 5;

printf("%d\n",*((int*)p));
```

3) Dimensiunea unui pointer depinde de arhitectura și sistemul de operare pe care a fost compilat programul. Dimensiunea se determină cu `sizeof(void *)` și nu este în mod necesar egală cu dimensiunea unui tip de date întreg.

- **Pointeri la funcții**

Capcana apelului fără paranteze:

```
int Exemplu();

int main()
{
      int x;
    scanf("%d",&x);
    if(Exemplu)
        printf("Mesaj 1");
    else
        printf("Mesaj 2");

    return 0;
}
```

- **Greșeală clasică:**

1)
```
int *p, x = 100 ;
 *p = x;
```

De ce? Deoarece pointerului p nu i s-a atribuit inițial nicio adresă, el conţine una necunoscută. Deci, la atribuirea *p = x valoarea din x este scrisă într-o locaţie de memorie necunoscută. Acest tip de problemă scapă de obicei neobservată când programul este mic, deoarece cele mai mari şanse sunt ca p să conţină o adresă "sigură" – una care nu intra în zona de program, de date ori a sistemului de operare.

Dar, pe măsură ce programul se măreşte, creşte şi probabilitatea ca p să conţină ceva vital. În cele din urmă, programul se va opri.

Așadar, corest este:

```
int *p, x = 100 ;
 p=&x;
```

Alt exemplu:

```
  char *p;

  strcpy(p, "abc");
  printf("%s",p);
```

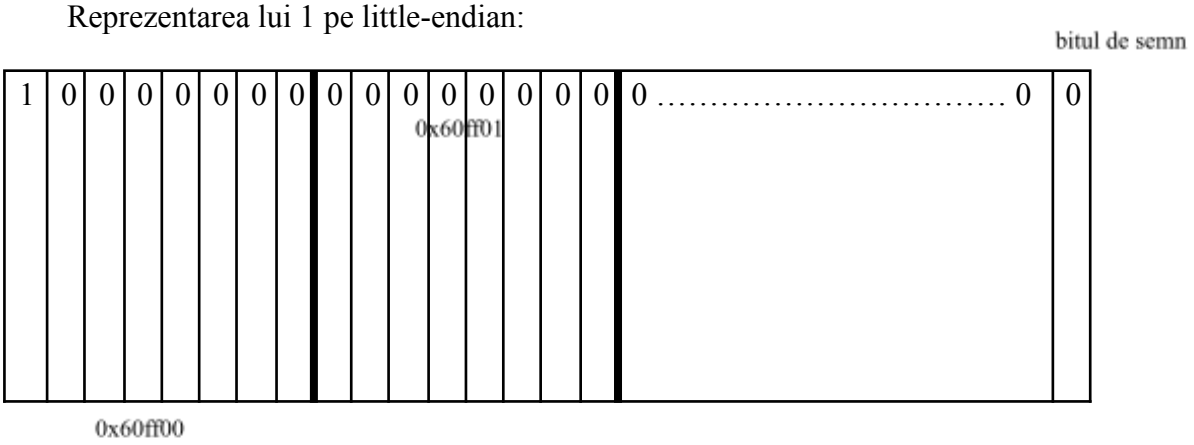
Soluții

```
char *p="abc";
```
sau
```
char p[20];
strcpy(p, "abc");
```

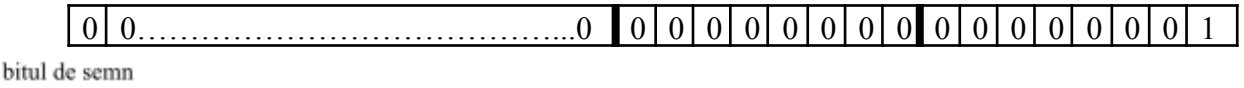
Probleme

1. Scrieți codul pentru a determina dacă lucrați pe un calculator **little-endian** sau **big-endian**.

   Reprezentarea lui 1 pe little-endian:



   Reprezentarea lui 1 pe big-endian:



2. Fie un număr întreg x. Folosiți pointeri și conversii pentru a extrage, pe rând, fiecare din cei 4 octeți.

3. Folosiți un tablou de pointeri la funcții pentru a afișa radicalul, inversul și sinusul unui număr real citit de la tastatură.

4. Se citesc de la tastură un număr natural **n** ce reprezintă dimensiunea unei matrice pătratice și elementele unei matrice pătratice. Folosind pointeri, să se afișeze elementele matricei,

elementul de la intersecţia diagonalelor (pentru n impar), iar apoi elementele de pe cele două diagonale.