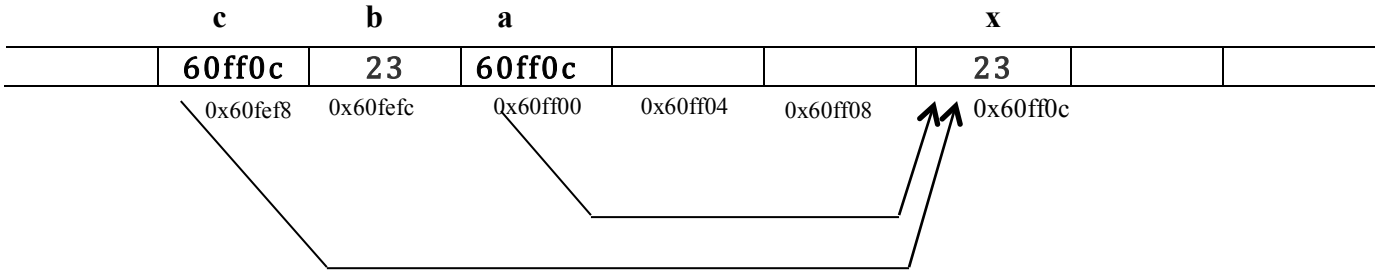
Programare procedurală
– Laborator 8 –

Pointeri

- **Pointerii** sunt consideraţi un tip de date derivate în limbajul C. Totuşi, de obicei, prin pointer se înţelege o variabilă de tip pointer. Un pointer reţine **o adresă de memorie**:
    - adresa unui **tip de date**: elementar, structură, şir 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 adresă 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 nu trebuie convertit explicit către un anumit tip de pointer, deoarece acest lucru se realizează implicit. Totuşi, un pointer de tip void trebuie atribuit unui pointer cu tip, deoarece altfel nu poate fi utilizat, neavând aritmetică.

```
*p=23;                                  // invalid use of void expression
// Atunci, convertim întâi la int:
p = (int *) malloc(sizeof(int));        // alocăm spațiu pentru un int (va urma)
*p = 5;
printf("%d\n",*p);
```

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

- **Greşeală clasică:**

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

De ce este greşeală? Deoarece pointerului p nu i s-a atribuit iniţial 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.

Aşadar, corect este:

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

Alt exemplu ce conţine greşeli:

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

Corect este:

```
char *p="abc";          //prima varianta

char p[20];
strcpy(p, "abc");   //a doua varianta

char *p = (char *)malloc(4);//a treia varianta – cu alocare dinamica folosind functia malloc
strcpy(p,"abc");
```

- **Pointeri la structuri**

Putem defini pointeri la structuri în acelaşi mod în care am defini un pointer la orice altă variabilă.

Declarare

struct struct_name *struct_pointer;

Pentru a accesa membrii unei structuri prin intermediul pointerilor, folosim operatorul ->

struct_pointer->member_name;

Instructiunea de mai sus este echivalenta cu:

```
(*struct_pointer).member_name;
```

Pentru a trimite un pointer la o structura ca argument unei functii, folosim următoarea sintaxa:

```
return_type function_name(struct struct_name *param_name);
```

Exemplu

```
struct Book {
      int book_id;
      char title[50];
      char author[50];
      char subject[100];
};
void printBook(struct Book *book);

int main () {
      struct Book *book_pointer, book1;
      book_pointer = &book1;
      //specificatia pentru book1:
      book1.book_id = 2000;
      strcpy(book1.title, "Defence Against the Dark Arts");
      strcpy(book1.author, "Severus Snape");
      strcpy(book1.subject, "Defence Against the Dark Arts");

      printBook(&book1);
      return 0;
}

void printBook(struct Book *book) {
      printf("Book title: %s\n", book->title);
      printf("Book author: %s\n", book->author);
      printf("Book subject: %s\n", book->subject);
      printf("Book identifier: %d\n", book->book_id);
}
```

Accesarea membrilor structurilor prin intermediul pointerilor utilizând alocarea dinamică

Putem aloca memorie dinamic utilizand functia malloc() din header-ul "stdlib.h"
Sintaxa:

```
      ptr = (cast_type*)malloc(byte_size);
```

Exemplu:

```
int main() {
      struct Book *book_ptr;
      int i, n;

      printf("Enter number of books: ");
      scanf("%d, &n);

      //alocam memorie pentru n structuri Book cu book_ptr
      //indicand spre adresa de baza
      book_ptr = (struct Book*)malloc(n * sizeof(struct Book));

      for(i = 0; i < n; ++i) {
            printf("Enter book id, title, author and subject");
            scanf("%d", &(book_ptr+i)->book_id);
            scanf("%s", &(book_ptr+i)->title);
```

```
            scanf("%s", &(book_ptr+i)->subject);
        }
        //afisam informatiile despre cartile citite anterior
        for(i = 0; i < n; ++i) {
              printBook1(book_ptr+i);
        }

        return 0;
}
```

Probleme

1. Scrieţi codul pentru a determina dacă lucraţi pe un calculator **little-endian** sau **big-endian**. Puteţi realiza acest lucru determinând cum sunt memoraţi octeţii din scrierea binară al lui 1. În reprezentarea little-endian octetul cel mai semnificativ este pus ultimul iar octetul cel mai puţin semnificativ este pus primul. În reprezentarea big-endian octetul cel mai semnificativ este pus primul iar octetul cel mai puţin semnificativ este pus ultimul. ul

   Reprezentarea lui 1 pe little-endian:
   00000001 00000000 00000000 00000000

   Reprezentarea lui 1 pe big-endian:
   00000000 00000000 00000000 00000001

2. Fie un număr întreg x. Folosiţi pointeri şi conversii pentru a extrage, pe rând, fiecare din cei 4 octeţi.

3. Se citesc de la tastatură un număr natural n ce reprezintă dimensiunea unei matrice pătratice și elementele unei matrice pătratice. Folosind pointeri realizaţi următoarele: (i) afișaţi elementele matricei; (ii) considerând n impar, afișaţi elementul de la intersecţia diagonalelor; (iii) afișaţi elementele de pe cele două diagonal; (iv) interschimbaţi elementele linei i cu elementele liniei j.

4. Se citesc de la tastatură un număr natural n şi apoi n numere complexe (reprezentate cu ajutorul unei structuri) într-un tablou unidimensional alocat dinamic. Scrieţi o funcţie care foloseşte pointeri pentru a determina şi afișa numărul complex cu cel mai mare modul.

5. Scrieţi un program care citeşte de la tastatură un tablou unidimensional format din numere întregi, iar apoi construieşte un tablou format din numerele strict negative din tabloul iniţial şi un alt tablou format din numerele strict pozitive. Toate cele 3 tablouri vor fi alocate dinamic! Implementaţi o soluţie care să utilizeze funcţia malloc/calloc pentru alocarea dinamică a tablourilor formate din valorile strict pozitive/strict negative şi o soluţie care să utilizeze funcţia realloc pentru cele două tablouri!

6. Scrieţi o funcţie care construieşte un tablou alocat dinamic format din poziţiile pe care apare valoarea maximă într-un tablou unidimensional dat. Scrieţi un program de test pentru această funcţie!

7. Scrieţi un program care citeşte de la tastatură un tablou unidimensional format din numere reale şi construieşte un tablou de numere reale inserând între oricare două valori aflate pe poziţii consecutive în tabloul iniţial media lor aritmetică. Ambele tablouri vor fi alocate dinamic!