Laborator C

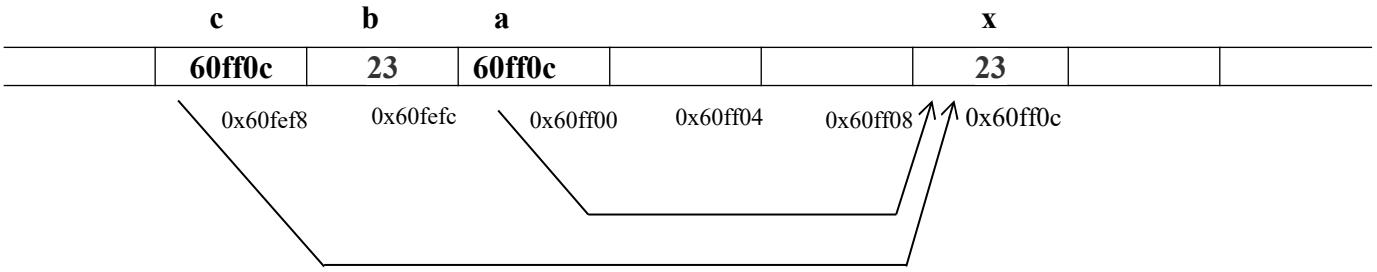
Pointeri

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


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.

- **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, corect 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");
```

Pointeri la structuri

Putem defini pointeri la structuri in acelasi mod in care am defini un pointer la orice alta variabila.

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 urmatoarea 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 utilizand alocarea dinamica:

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)->author);
         scanf("%s", &(book_ptr+i)->subject);
     }
     //afisam informatiile despre cartile citite anterior
```

```
    for(i = 0; i < n; ++i) {
         printBook1(book_ptr+i);
    }

    return 0;
}
```

Siruri de caractere

Limbajul C nu dispune de un tip de date nativ pentru reprezentarea sirurilor de caractere de lungime variabila. In acest scop se utilizeaza structuri de date de tip tablou de caractere.

Intrucat sirurile de caractere prelucrate in programe au in general lungime variabila, s-a stabilit o conventie prin care ultimul caracter utilizat al unui sir este urmat de un caracter cu valoarea zero ('\0'), numit terminator de sir.

```
char sir [10];
```

Exemplul de mai sus declara un tablou de 10 de elemente de tip caracter. Un asemenea tablou se poate folosi pentru memorarea unui sir de caractere de lungime variabila, dar de maxim 9 de caractere, intrucat ultimul element este rezervat pentru terminatorul de sir.

Daca sirul de mai sus contine valoarea "TEST", continutul memoriei rezervate tabloului este urmatorul:

Index	0	1	2	3	4	5	6	7	8	9
Continut	T	E	S	T	**\0**	-	-	-	-	-

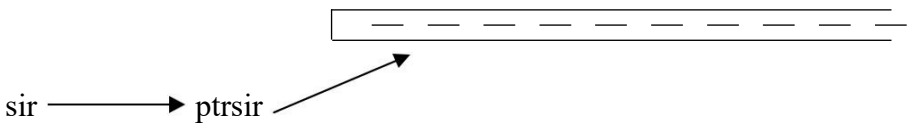
Valoarea elementelor tabloului corespunzatoare indicilor de la 5 la 9 nu este cunoscuta. Evident, aceste tablouri care memoreaza siruri de caractere se pot aloca si dinamic:

```
char *sir = (char*)malloc (10 * sizeof (char));
/* ...operatii cu variabila sir... */
free (sir);
```

Daca se doreste doar accesarea si prelucrarea elementelor unui sir de caractere care a fost alocat anterior, static (declaratie de tablou) sau dinamic, se poate utiliza si o variabila de tip pointer catre caracter:

```
char sir [10];
char *ptrsir;
ptrsir = sir;
```

/*Ambele variabile indica spre acelasi sir de caractere*/



Orice sir de caractere care se prelucreaza in program trebuie sa dispuna insa de o declaratie de alocare de memorie (static sau dinamic).

Compilatoarele de C permit initializarea tablourilor de caractere in momentul declararii acestora cu un sir de caractere:

```
char sir [10] = "Un sir";
```

In urma acestei declaratii, variabila *sir* va avea urmatorul continut:

Index	0	1	2	3	4	5	6	7	8	9
Continut	U	n		S	i	r	**0**	-	-	-

Lista funcțiilor pentru siruri de caractere:
http://www.cplusplus.com/reference/cstring/

Probleme

1. 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.
2. Implementati o functie care numara aparitiile unui cuvant intr-un text dat.
   int count_occurrences(const char* text, const char* word);
3. Folosind functia count_occurrences implementati o alta functie ce citeste doua cuvinte si inlocuieste intr-un text introdus de la tastatura toate aparitiile primului cuvant prin cel de-al doilea.