9. Pointeri.

9.1. Operatori specifici pointerilor.

Un *pointer* este o variabilă care are ca valori adrese ale altor variabile, sau mai general adrese de memorie.
Un pointer este asociat unui tip de variabile, deci avem pointeri către **`int, char, float`**, etc.
În general o variabilă pointer **`p`** către tipul **`T`** se declară: **`T *p;`**
Un tip pointer la tipul **`T`** are tipul **`T*.`**

Exemple:

```
int j,*pj;/*pj este o variabila de tip pointer la întregi*/
char c, *pc;
```

Se introduc doi noi operatori:

- *operatorul de adresare* **`&`** - aplicat unei variabile furnizează adresa acelei variabile

```
pj=&j; /* inițializare pointer */
pc=&c;
```

Aceste inițializări pot fi făcute la definirea variabilelor pointeri:

```
int j, *pj=&j;
char c, *pc=&c;
```

O greşeală frevent comisă o reprezintă utilizarea unor pointeri neinițializați.

```
int *px;
*px=5;/* greşit, pointerul px nu este inițializat (legat
         la o adresă de variabilă întreagă) */
```

Pentru a evita această eroare vom inițializa în mod explicit un pointer la **`NULL`**, atunci când nu este folosit.

- *operatorul de indirectare (dereferențiere)* **`*`** – permite accesul la o variabilă prin intermediul unui pointer. Dacă **`p`** este un pointer de tip **`T*,`** atunci **`*p`** este obiectul de tip **`T`** aflat la adresa **`p`**.

În mod evident avem:

```
*(&x) = x;
&(*p) = p;
```

Exemplu;

```
int *px, x;
x=100;
px=&x; // px contine adresa lui x
printf("%d\n", *px);  // se afiseaza 100
```

Dereferențierea unui pointer neinițializat sau având valoarea **`NULL`** conduce la o eroare la execuție.

9.2. Pointeri generici (pointeri void).

Pentru a utiliza un pointer cu mai multe tipuri de date, la declararea lui nu îl legăm de un anumit tip.

```
void *px; // pointerul px nu este legat de nici un tip
```

Un pointer nelegat de un tip nu poate fi dereferențiat.
Utilizarea acestui tip presupune conversii explicite de tip (cast). Exemplu:

```
int i;
void *p;
. . .
p=&i;
*(int*)p=5; // ar fi fost gresit *p=5
```

Exemplul 17*: Definiți o funcție care afişează o valoare ce poate aparține unuia din tipurile:* `char, int, double`.

```
#include <stdio.h>
enum tip {caracter, intreg, real};
void afisare(void *px, enum tip t) {
   switch(t) {
      case caracter:
         printf("%c\n", *(char*)px); break;
      case intreg:
         printf("%5d\n", *(int*)px); break;
      case real:
         printf("%6.2lf\n",*(double*)px); break;
   }
}

int main(){
   char c='X';
   int i=10;
   double d=2.5;
   afisare(&c, caracter);
   afisare(&i, intreg);
   afisare(&d, real);
}
```

9.3. Pointeri constanți şi pointeri la constante.

In definițiile:

```
const int x=10,
*px=&x;
```

x este o constantă, în timp ce **px** este un pointer la o constantă. Aceasta însemnă că **x**, accesibil şi prin **px** nu este modificabil (operaţiile **x++** şi **(*px)++** fiind incorecte, dar modificarea pointerului **px** este permisă (**px++** este corectă).
Un pointer constant (nemodificabil), se defineşte prin:

```
int y, * const py=&y;
```

In acest caz, modificarea pointerului (**py++**) nu este permisă, dar conţinutul referit de pointer poate fi modificat (**(*py)++**).
Un pointer constant (nemodificabil) la o constantă se defineşte prin:

```
const int c=5, *const pc=&c;
```

In cazul folosirii unor parametri pointeri, pentru a preveni modificarea conţinutului referit de aceştia se preferă definirea lor ca pointeri la constante. De exemplu o functie care compară două şiruri de caractere are prototipul:

```
int strcmp(const char *s, const char *d);
```

9.4. Operaţii aritmetice cu pointeri.

Asupra pointerilor pot fi efectuate următoarele operaţii:

- adunarea / scăderea unei constante
- incrementarea / decrementarea
- scăderea a doi pointeri de acelaşi tip

Prin incrementarea unui pointer legat de un tip **T**, adresa nu este crescută cu **1**, ci cu valoarea **sizeof(T)** care asigură adresarea următorului obiect de acelaşi tip.
În mod asemănător, **p + n** reprezintă de fapt **p+n*sizeof(T)** .

Doi pointeri care indică elemente ale aceluiaşi tablou pot fi comparaţi prin relaţia de egalitate sau ne egalitate, sau pot fi scăzuţi.
Pointerii pot fi comparaţi prin relaţiile **==** şi **!=** cu constanta simbolică **NULL** (definită în **stdio.h**) .

9.5. Legătura între pointeri şi tablouri.

Între pointeri şi tablouri există o legătură foarte strânsă. Orice operaţie realizată folosind variabile indexate se poate obţine şi folosind pointeri.
În C numele unui tablou este un pointer constant la primul element din tablou: **x=&x[0]**
Numele de tablouri reprezintă *pointeri constanţi*, deci nu pot fi modificaţi ca pointerii adevăraţi.
Exemplu:

```
int x[10], *px;
px=x;  /* sunt operatii permise */
px++;
x=px;  /* sunt operatii interzise, deoarece x este */
x++;   /* pointer constant */
```

Prin urmare *variabilele indexate* pot fi transformate în *expresii cu pointeri* şi avem echivalențele:

Tabel 8.2. Corespondența între tablouri şi pointeri

Adresă		Valoare	
Notație indexată	Notație cu pointeri	Notație indexată	Notație cu pointeri
`&x[0]`	`x`	`x[0]`	`*x`
`&x[1]`	`x+1`	`x[1]`	`*(x+1)`
`&x[i]`	`x+i`	`x[i]`	`*(x+i)`
`&x[n-1]`	`x+n-1`	`x[n-1]`	`*(x+n-1)`

În C avem următoarea echivalență ciudată! Dacă **x** este un tablou de întregi

x[i] ≡ i[x]

Într-adevăr: **x[i]=*(x+i)=*(i+x)=i[x]**

9.6. Parametri tablouri.

Dacă în lista de parametri a unei funcții apare numele unui tablou cu o singură dimensiune se va transmite adresa de început a tabloului. Aceasta ne permite să nu specificăm dimensiunea tabloului, atât la definirea, cât şi la apelul funcției.

Exemplul 18: *Scrieți o funcție care calculează produsul scalar a doi vectori* **x** *şi* **y**, *având câte* **n** *componente fiecare*.
Antetul funcției va fi:

```
double scalar(int n, double x[], double
y[])
```

Funcția poate fi declarată cu parametri formali pointeri în locul tablourilor:

```
double scalar(int n, double *x, double *y)
{ double P=0;
  for (int i=0; i<=n; i++)
     P=P+x[i]*y[i];
  return P;
}
```

echivalentă cu:

```
double scalar(int n, double *x, double *y)
{ double P=0;
  for (int i=0; i<=n; i++)
     P=P+*(x+i)**(y+i);
  return P;
}
```