Alocarea dinamică a memoriei. Tipul pointer

Prin **alocarea dinamică a memoriei** se înţelege alocarea unei variabile într-o zonă specială de memorie numită **Heap**, iar alocarea se face **în timpul executării programului** şi nu la început ca la alocarea statică a memoriei. Mecanismul alocării dinamice a memoriei necesită **pointeri.**

Avantajele alocării dinamice a memoriei

- Apelând la Heap se măreşte memoria disponibilă. Cele 3 zone de memorie internă sunt:

Segmentul de date	Codul sursă şi variabilele globale
Segmentul de stivă	Apeluri de funcţii şi variabile locale
Heap	Pointeri

- Programul va utiliza atâta memorie cât are nevoie, deoarece memoria se alocă şi se eliberează pe parcursul executării programului în funcţie de necesităţi.
- Anumite structuri de date, cum ar fi **listele liniare**, se implementează cu uşurinţă în Heap.

Obs. Orice variabilă de memorie ocupă un număr de octeţi succesivi. **Adresa variabilei** este dată de adresa primului octet ocupat.

Pointerul este o varibilă de memorie în care se memorează o adresă de memorie (adresa unei variabile). Un pointer se asociază unui tip de date (int, float, struct etc), numit **tip de bază**, şi el va reprezenta adresa unei variabile de acest tip.

Declarare: **tip_baza *nume_variabila;**
// **pointerul "nume_variabila" reprezintă adresa unei variabile de tipul "tip_baza"**

Exemple:

```
char *s, *t; // pointeri catre char (ocupă câte 1 Byte)
int *x; // pointer catre int (ocupă 4 Bytes)
float *a, *b; // pointeri catre float (ocupă câte 4 Bytes)
void *p; //pointer fara tip
```

Pointerul 0 sau NULL înseamnă **nicio adresă / adresă inexistentă / adresă nealocată** şi este util pentru iniţilizarea valorii unui pointer.

Exemplu: int *p=NULL;

Pointerii sunt utilizaţi pentru alocarea dinamică a memoriei.

Operatori specifici

a) **Operatorul de adresare**: **&** - furnizează **adresa** unei variabile de memorie. Nu se poate aplica unei constante sau unei expresii. Operaţia se numeşte **referenţierea** unei variabile de memorie.

b) **Operatorul de redirectare**: ***** - furnizează **valoarea** variabilei de memorie care se găseşte la adresa memorată de pointer. Operaţia se numeşte **dereferenţierea** unui pointer. Nu se aplică la pointeri de tip void.

Ex 1.

```
int x=10, *p;
p=&x; // p= adresa variabilei x;
cout<<x<<" "<<*p;  //10 10
// *p = valorea/ obiectul de la adresa p
cout<<p; // adresa lui x=constanta hexazecimala
```

Operatori pentru alocarea dinamica a memoriei

1) Operatorul **new** - alocă spaţiu în Heap pentru o variabilă de tipul referit de pointer.

```
tip *p;
p=new tip; // alocă spaţiu la adresa p pentru o varibilă de tipul "tip"
```

Ex 2.

```
int *p;
p=new int; // alocă spaţiu la adresa p pentru o variabilă de tip int
*p=54;
```

2) Operatorul **delete** - elibereaza spaţiul de memorie de la adresa p.

```
delete p;
```

Obs. O variabilă alocată în Heap ocupă spaţiul alocat fie până la eliberarea spaţiului cu **delete**, fie până la sfârşitul executării programului.

Ex 3. Suma a două nr întregi cu pointeri

```
int *x, *y, s;
x=new int;
y=new int;
*x=7; *y=10;
cout<< *x + *y; // 17
delete x; delete y;
```

Ex 4.

```
float *a, *b; // pointeri către float
a=new float;
*a=3;
b=a; // atribuire între pointeri, care este echivalentă cu:  b=new float; *b=*a;
cout<< *b; // 3
```

Obs. Este permisă operaţia de atribuire între doi pointeri de acelaşi tip. În plus, unui pointer fără tip (void) i se poate atribui orice pointer cu tip.

Ex 5.

```
struct elev
      {  char nume[30];
         int varsta;
      }  *p;  // p= adresa unei variabile de tip elev

p=new elev; // alocă spaţiu la adresa p pentru o variabilă de tip elev
cin>> p->nume>>p->varsta;
cout<< p->nume<<" "<<p->varsta;
delete p;  // se elibereaza spatiul de la adresa p
```

Tema:

1) Interschimbarea valorilor a 2 numere întregi – folosind doi pointeri către int.
2) Maximul dintre 3 numere reale – folosind trei pointeri către float.