

**Introducere**

Variabilele pointer sunt variabile care primesc ca valoare adrese de memorie.

Din punct de vedere al conținutului zonei de memorie adresată prin variabilele pointer, se disting două categorii:

- **pointeri de date** – conțin ca valori adrese de variabile sau constante;
- **pointeri de funcții** – conțin adrese de cod executabil ale funcțiilor.

În această laborator vor fi prezentate noțiuni referitoare la pointeri de date.

De asemenea, vor fi prezentate variabilele referință a căror semnificație este apropiată de pointerii de date, ele primind ca valoare adrese de memorie ale unor obiecte.

**8.1. Variabile pointeri****8.1.1. Declararea variabilelor pointeri de date**

Sintaxa declarării unei variabile pointer de date este:

**tip \* nume\_variabila;**

unde: – **tip** poate fi orice tip de date, fundamental sau definit de utilizator;

– **nume\_variabilă** este un identificator.

Simbolul \* precizează că variabila declarată este un pointer. Ea va putea primi ca valoare numai adrese de date de tipul **tip**.

Exemple de declarare a variabilelor pointeri:

```
int * p_i;           // pointer la tipul int
float * p_f;         // pointer la tipul float;
int* tab_p[10];      // tablou de pointeri către întregi
float ** p_p;        // pointer la pointer
void * p_v;          // pointer la tipul void – poate lua ca valoare adresa oricărui tip de date
```

Declarația unei variabile pointer, ca în cazul oricărei variabile, are ca efect alocarea de memorie, adresa putând fi determinată cu operatorul &, numărul de octeți alocați fiind determinat de tipul date (2 sau 4 octeți) și se poate determina cu ajutorul operatorului sizeof.

```
int * p_i;           // pointer la tipul int
&p_i;               // expresie care returnează adresa la care este alocată variabila
sizeof(p_i);        // expresie care returnează numărul de octeți alocați pentru variabilă
```

Declarația unei variabile pointer poate fi făcută cu sau fără inițializare.

```
int * p_i;           // declarație fără inițializare
float * p_r = 0;     // declarație cu inițializare
```

### 8.1.2. Operații de atribuire cu pointeri

Variabilele pointer pot prelua, prin atribuire, valoarea adreselor unor variabile sau constante alocate în memorie.

Atribuirea se poate face numai dacă tipul pointerului și tipul datei a cărei adresă o preia sunt identice.

Variabilelor pointer nu li se pot atribui valori constante, cu excepția valorii 0.

Exemple de atribuire de valori variabilelor pointeri:

```
int var1;  
int *p1;  
p1=&var1;  
double var2;  
double *p2;  
p2=&var2;  
p2=&var1;           // eroare, se atribuie adresa unui int unui pointer de tip double  
p1=0;               // atribuire corectă, valoarea 0 nu face referire la nici o adresă de memorie  
p1=NULL;            // atribuire corectă, valoarea NULL este o constantă cu valoarea 0 definită în  
                    // mai multe fișiere header, cum ar fi stdlib.h, alloc.h, stdio.h etc.  
p1=0xFF00;          // eroare, nu se admite atribuirea decât a constantei 0 sau a unei adrese a unui obiect
```



#### Observații

- Constanta întreagă 0 se poate atribui oricărei variabilă pointer. Această valoare este desemnată în mod frecvent prin constanta NULL. Adresele funcțiilor și obiectelor au valori diferite de 0. Valoarea 0 indică faptul că nu se face referire la nici o adresă de memorie.
- Variabilele pointer neinițializate, ca orice variabilă, iau valoarea 0 dacă sunt declarate global sau static și valori reziduale în cazul variabilelor automate.
- Este important ca orice variabilă pointer să fie inițializată cu o valoare validă, 0 sau adresa unui obiect alocat în memorie, înainte de a fi utilizată, deoarece la compilare sau execuție nu se fac verificări privind validitatea adreselor utilizate. Utilizarea incorectă a adreselor poate produce efecte necontrolate în evoluția programului.
- La atribuire este important ca tipul variabilei pointer și tipul datei de la adresa preluată să fie identice, în caz contrar se obține un mesaj de eroare.

### 8.1.3. Referirea obiectelor prin variabile pointer

Se consideră secvența:

```
tip nume_obiect;  
tip*nume_pointer;  
nume_pointer = &nume_obiect;
```

Referirea obiectului indirect, prin intermediul variabilei **nume\_pointer** se face cu sintaxa:

```
*nume_pointer
```

Această expresie se citește: obiectul de la adresa **nume\_pointer**.

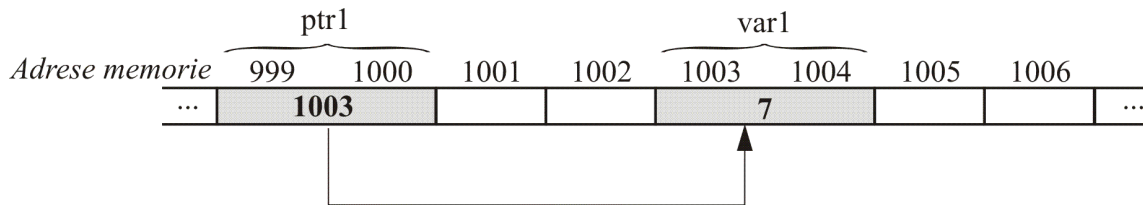
Expresia `*nume_pointer` poate fi folosită atât ca rvalue, cât și ca lvalue, adică poate fi folosită atât pentru a obține valoarea memorată la adresa indicată de pointer, cât și pentru atribuire.

Tipul pointerului determină modul de interpretare a datelor de la adresa indicată.

Se ia ca exemplu secvența:

```
int var1=7;
int * ptr1 = &var1;
```

Modul de interpretare al acesteia este descris în fig. 8.1.



**Fig. 8.1. Exemplu de utilizare a variabilei pointer**

Dacă variabila `var1` este alocată la adresa 1003, variabila `ptr1` preia ca valoare valoarea 1003.

În continuare este prezentat un exemplu de folosire a variabilelor pointeri.

```
#include <stdio.h>
main()
{
    int var=100;          // declarație de variabilă int
    int *ptr;             // declarație de variabilă int * (pointer de int)
    printf("\nVariabila var se afla la adresa: %p\n", &var);
    printf("\nsi are valoarea var= %d\n", var);

    ptr=&var;
    printf("\nVariabila ptr are valoarea: %p\n", ptr);
    printf("\nsi adreseaza obiectul: %d\n", *ptr);

    *ptr=200;             // se modifică valoarea obiectului de la adresa memorată în ptr;
                        // ptr conține adresa variabilei var, deci valoarea 200 este
                        // preluată de var; prin *ptr se face referire în mod indirect la
                        // variabila var, operația se mai numește "indirectare"

    printf("\nAcum var are valoarea %d\n", var);
}
```

#### 8.1.4. Pointeri generici

Se pot defini pointeri la tipul `void`. Ei constituie o categorie aparte de pointeri. Aceștia pot primi ca valoare adresa oricărui tip de date și se numesc pointeri generici.

Exemplu de folosire a pointerilor generici:

```
int vi=10;
double vr = 1.5;
```

```

void * pv;
pv = &vi;           // corect, pv poate prelua adresa oricărui tip de date
pv = &vr;           // corect, pv poate prelua adresa oricărui tip de date

```

Pentru referirea obiectelor prin intermediul unui pointer generic, este necesar să se precizeze, prin conversia pointerului, tipul obiectului referit, astfel încât compilatorul să aibă informația necesară despre reprezentarea obiectului referit.

```

int vi=10;
double vr = 1.5;
void * pv;
pv = &vi;
printf("%d", *(int*)pv);
pv = &vr;
*(double*)pv = 9.5;
printf("%lf", *(double*)pv);

```

### 8.1.5. Operații aritmetice cu pointeri

Cu variabilele pointer, în afară de operațiile de atribuire, se pot face și operații de adunare, scădere, inclusiv incrementare și decrementare și operații de comparare.

Valorile a doi pointeri pot fi comparate folosind operatorii relaționali, spre exemplu:

```

int*p1, *p2;
...
if(p1<p2)
    printf("p1<p2");
if(p1==0)
    printf("pointerul p1 nu adreseaza nici un obiect");

```

Operațiile de adunare și scădere sunt interpretate diferit față de operațiile efectuate cu date numerice.

Având variabila pointer

**tip \* ptr;**

Expresia:

**ptr +n**

returnează ca valoare adresa memorată în ptr la care se adaugă

**n\* sizeof(tip)**

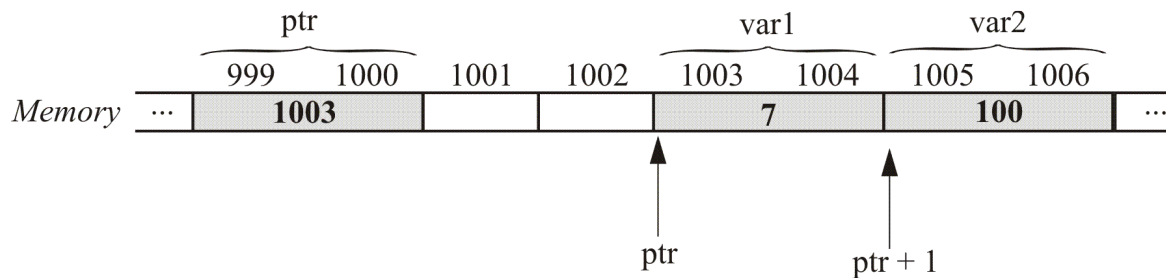
Pentru situația descrisă prin secvența:

```

int var1, var2, *ptr;
ptr = &var1;
ptr = ptr +1;

```

expresia (ptr + 1) reprezintă deplasarea în memorie cu un întreg, adică cu un număr de octeți egal cu sizeof(int) (vezi Fig. 8.2.).



**Fig. 8.2. Exemplu pentru operația de adunare a unui întreg la un pointer**

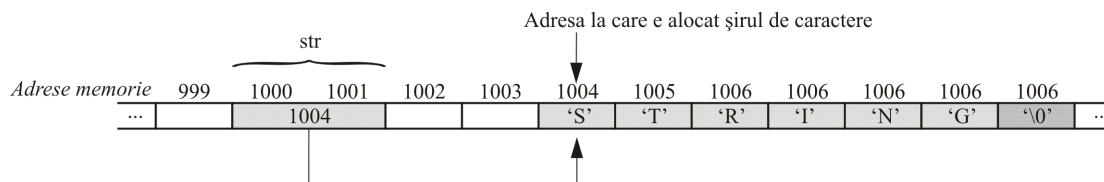
Exemplu de expresii ce conțin operații aritmetice cu pointeri:

```
float *pf;
pf +4;      // adresa se modifică cu sizeof(float)*4, adică cu 20 de octeți
pf++;      // pf își modifică valoarea, crește cu 4 octeți
pf--;      // pf își modifică valoarea, scade cu 4 octeți
```

### 8.1.6. Tablouri și pointeri

Numele unui tablou fără index este un pointer constant de tipul elementelor tabloului și are ca valoare adresa primului element al tabloului.

```
char *str="STRING";
```



**Fig.7.3. Legătura dintre pointeri și tablouri de date**

Având în vedere semnificația operațiilor de adunare și scădere cu variabile pointer, se pot referi elementele tabloului fie prin index, fie prin operații cu pointeri ca în exemplele următoare.

Exemple referitoare la legătura dintre pointeri și tablouri de date:

```
float tab[20], *ptr;
ptr=tab;      // atribuire validă, pointerul ptr va conține adresa
              // primului element al tabloului
&tab[0] == tab ;      // expresie adevărată
&tab[2] == tab+2 ;    // expresie adevărată
tab[0] == *tab ;      // expresie adevărată
tab[2] == *(tab+2) ;  // expresie adevărată
tab++ ;          // eroare, tab este un pointer constant, deci nu se poate incrementa
ptr++ ;          // corect, ptr este un pointer la float, nu a fost declarat constant
```

Tablourile multidimensionale reprezintă tablouri cu elemente tablouri, astfel încât numele tabloului (fără index) este un pointer de tablouri.

```

float mat[10][10];           // mat reprezintă un tablou de pointeri float
float *p;                   // p este un pointer float
p=mat;                      // eroare, tipurile pointerilor diferă, unul este pointer
                             // de float, celălalt e pointer de pointer de float
p=(float*)mat;              // corect, s-a folosit o conversie explicită de tip
mat == &mat[0][0];          // expresie adevărată
mat+1 == &mat[1][0];        // expresie adevărată
*(mat+9) == mat[9][0];      // expresie adevărată

```

Având în vedere aceste corespondențe, se pot scrie secvențe de cod echivalente, fie folosind referirea elementelor de tablou prin index, fie folosind expresii cu pointeri, ca în exemplele următoare.

Secvența următoare:

```

int tab[7]={9,11,25,7,44,2,14};
printf("\n\nafisarea elementelor tabloului tab);
for( i = 0 ; i < 7 ; i++)
    printf("\ntab[%d] = %d", i, tab[i]);

```

poate fi rescrisă astfel:

```

int tab[7] = {9,11,25,7,44,2,14};
printf("\n\nafisarea elementelor tabloului tab);
for( i = 0 ; i < 7 ; i++)
    printf("\ntab[%d] = %d", i, *(tab + i));    // expresia tab+i reprezintă ca valoare
                                                // adresa elementului de index i

```

Sau secvența:

```

printf("\n\ncitire de valori de la tastatura pentru elementele tabloului tab");
for( i = 0; i<7 ; i++)
{
    printf("\ntab[%d]=",i);
    scanf("%d",&tab[i]);
}

```

poate fi înlocuită cu:

```

printf("\n\ncitire de valori de la tastatura pentru elementele tabloului tab");
for( i = 0; i<7 ; i++)
{
    printf("\ntab[%d]=",i);
    scanf("%d", tab + 1);    // în funcția scanf() se precizează adresa la care
                             // se depune valoarea citită de la tastatură
}

```

În mod similar se poate proceda și în cazul matricelor. Astfel, secvența de citire a elementelor matricei de la tastatură următoare:

```

int mat[3][4], i, j;
//se citesc de la tastatura elementele lui mat
printf("Introduceti elementele matricei mat:\n");
for(i=0 ; i<3 ; i++)
    for(j=0 ; j<4 ; j++)
    { printf("mat1[%d][%d]=", i, j);
      scanf("%d", &mat1[i][j]);
    }

```

se poate scrie folosind operații cu pointeri:

```
printf("Introduceți elementele matricei mat:\n");
for(i=0;i<3;i++)
    for(j=0;j<4;j++)
        { printf("mat[%d][%d]=", i, j);
          scanf("%d",*(mat+i)+j);           // expresia *(mat+i)+j reprezintă adresa
                                              // elementului de pe linia i, coloana j
        }
}
```



### Observații

- Expresia (mat) reprezintă adresa la care e alocată matricea; ca tip de dată este un pointer de pointer de int (dublă indirectare);
- Expresia (mat+i) reprezintă adresa liniei de index i; ca tip de dată este un pointer de pointer de int (dublă indirectare);
- Expresia \*(mat+i) reprezintă adresa primului element de pe linia de index i; ca valoare este egală cu valoarea expresiei (mat+i) dar ca tip de dată este un pointer de int (simplă indirectare);
- Expresia \*(mat+i)+j reprezintă adresa elementului de index j al liniei de index i ; ca tip de dată este un pointer de int (simplă indirectare).

Secvența de afișare a elementelor matricei:

```
printf("\nmat:");
for(i=0;i<3;i++)
{ printf("\n");
  for(j=0;j<4;j++)
    printf("%6d",mat[i][j]);
}
```

este echivalentă cu secvența:

```
printf("\nmat:");
for(i=0;i<3;i++)
{ printf("\n");
  for(j=0;j<4;j++)
    printf("%6d",*(*(mat+i)+j));
}
```

## 8.2. Variabile referință

C++ oferă posibilitatea de a declara identificatori ca referințe de obiecte (variabile sau constante). Referințele, ca și pointerii, conțin adrese. Pentru a declara o referință la un obiect se folosește simbolul &, folosind sintaxa:

**tip & id\_referinta = nume\_obiect;**

unde:

- **tip** este tipul obiectului pentru care se declară referința,
- simbolul **&** precizează că **id\_referinta** este numele unei variabile referință
- **nume\_obiect** este obiectul a cărui adresă va fi conținută în **id\_referinta**.

Exemplu de declarare și folosire comparativă a variabilelor pointeri și referință:

```

int n;                // se declară n de tip întreg
int *p=&n;            // se declară pointerul p cu inițializare adresa lui n
int &r=n;             // se definește r referința lui n
n=20;                // n primește valoarea 20
*p=25;               // n primește valoarea 25
r=30;                // n primește valoarea 30

```

În exemplul anterior, atât p, cât și r, acționează asupra variabilei n.

Atunci când se accesează o variabilă prin referința sa, nu este necesar să se folosească adresa, acest lucru realizându-se automat.

Spre deosebire de pointeri, care la un moment dat pot primi ca valoare adresa unei alte variabile, referințele nu pot fi modificate, ele fiind practic o redenumire a variabilei a căror adresă o conțin (se creează un alias al respectivei variabile).

În utilizarea referințelor, trebuie avute în vedere următoarele restricții:

- referințele trebuie inițializate în momentul declarării;
- referințelor nu li se pot modifica locațiile la care se referă;
- nu sunt permise referințe la câmpuri de biți, referințe la referințe și pointeri la referințe, deci nici tablouri de referințe.

```

int &r ;               // eroare, se declară o referință fără inițializare
int &r=20 ;           // corect, se declară o referință la o constantă
const int i = 20;
int &r = i ;          // eroare, se declară o referință întreagă la o constantă întreagă
const int i = 20;
const int &r = i ;    // corect, se declară o referință constantă întreagă la o
                      // constantă întreagă

```



## Exemple

/\*\*\*\*\*\*

**Exemplul 8.1.** Se exemplifică legătura dintre tablouri și pointeri. Se folosește în paralel referirea elementelor unui tablou unidimensional prin indecși și, respectiv, folosind operații cu pointeri.

\*\*\*\*\*/

```

#include <stdio.h>
#include <conio.h>

```

```

void main()
{ float vect[]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
  float *p;
  int i;
  p=vect;
  printf("\n");
  //se afiseaza elementele tabloului vect - referirea elementelor se face prin index
  for(i=0 ; i<12 ; i++)
    printf("%6.2f", vect[i]);
  printf("\n");
  //se afiseaza elementele tabloului vect - referirea elementelor se face prin adrese
  for(i=0 ; i<12 ; i++)
    printf("%6.2f", *(vect+i));

```



```

printf("\n");
//se afiseaza elementele tabloului vect - referirea elementelor se face prin adrese, folosind
//pointerul p (p nu isi modifica valoarea)
for(i=0 ; i<12 ; i++)
    printf("%6.2f", *(p+i));
printf("\n");
//se afiseaza elementele tabloului vect - folosind pointerul p, referirea se face prin index
//(p nu isi modifica valoarea)
for(i=0 ; i<12 ; i++)
    printf("%6.2f", p[i]);
printf("\n");
//se afiseaza elementele tabloului vect - folosind pointerul p (p isi modifica valoarea prin
//incrementare)
for(i=0 ; i<12 ; i++)
    printf("%6.2f", *p++);
getch();
}

```

/\*\*\*\*\*

**Exemplul 8.2.** Se exemplifică legătura dintre tablouri și pointeri. Se folosește în paralel referirea elementelor unui șir de caractere folosind operații cu pointeri.

\*\*\*\*\*/

```

#include <stdio.h>
#include <conio.h>

void main(void)
{ char sir[25] = "siruri si pointeri";
  int i;
  // Se afiseaza, caracter cu caracter, toate caracterele in format %c si %d.
  for(i=0 ; i<25 ; i++)
      printf("\n %c - %d", *(sir+i), *(sir+i));
  printf("\n");
  // Se afiseaza, caracter cu caracter, toate caracterele, pana la intalnirea terminatorului de sir,
  // in format %c si %d .
  for(i=0 ; *(sir+i) != 0 ; i++)
      printf("\n %c - %d", *(sir+i), *(sir+i));
  *sir = 'S';
  *(sir+7) = 'S';
  *(sir+10) = 'P';
  printf("\nSirul este: %s", sir);
  getch();
}

```

/\*\*\*\*\*

**Exemplul 8.3.** Se exemplifică legătura dintre tablouri și pointeri. Se folosește în paralel referirea elementelor unui tablou multidimensional (matrice) prin indecși și, respectiv, folosind operații cu pointeri.

\*\*\*\*\*/

```

#include <stdio.h>
#include <conio.h>

```

```

void main()
{ int mat[3][4]={ {1, 2, 3, 4},
                  {5, 6, 7, 8},
                  {9, 10, 11, 12}};

  int *p, i, j;
//afisarea elementelor matricii - referire prin cei doi indecsi
  for(i=0 ; i<3 ; i++)
  { printf("\n");
    for(j=0 ; j<4 ; j++)
      printf("%5d", mat[i][j]);
    getch();
  }
//afisarea de informatii despre matrice
  printf("\nsizeof(*mat) = %d", sizeof(*mat));
  printf("\n**mat = %d", **mat);
  printf("\n*(mat+1) = %d", *(mat+1));
  printf("\n*(mat+2) = %d", *(mat+2));
//atribuire prin conversie explicita a pointerului
  p = (int*)mat;
// afisarea elementelor tabloului, folosind pointerul p
  for(i=0 ; i<3 ; i++)
    printf("%5d", *(p+i));
  getch();
// afisarea elementelor tabloului, folosind pointerul p cu specificarea liniei si coloanei elementului
  for(i=0 ; i<3 ; i++)
  { printf("\n");
    for(j=0 ; j<4 ; j++)
      printf("%5d", *(p+i*4+j));
    getch();
  }
  for(i=0 ; i<3 ; i++)
  { printf("\n");
    for(j=0 ; j<4 ; j++)
      printf("%5d", p[i*4+j]);
    getch();
  }
}

```



### Întrebări. Exerciții. Probleme.

1. Se citesc de la tastatură elementele unui tablou unidimensional. Să se înlocuiască toate valorile negative cu valoarea 0. Se afișează elementele tabloului. Pentru referirea elementelor se vor folosi operații cu pointeri.
2. Se citesc de la tastatură elementele a două tablouri unidimensionale de aceeași dimensiune. Se calculează elementele unui al treilea tablou ca sumă a elementelor de același index ale primelor două și se afișează. Pentru referirea elementelor se vor folosi operații cu pointeri.
3. Se citesc de la tastatură două șiruri de caractere. Se determină un al treilea șir prin concatenarea primelor două și se afișează. Pentru referirea elementelor se vor folosi operații cu pointeri.