UNIVERSITATEA TEHNICĂ A MOLDOVEI

PROGRAMAREA CALCULATOARELOR
TIPURI DE DATE POINTERI. POINTERI ȘI TABLOURI
Prelegere

Kulev Mihail, dr., conf. univ.

Stimați studenți și stimată audiență!

Mă numesc Kulev Mihail sunt doctor, conferențiar universitar la Universitatea Tehnică a Moldovei. Din cadrul cursului PROGRAMAREA CALCULATOARELOR Vă propun spre atenția Dumneavoastră prelegerea cu tema:

” TIPURI DE DATE POINTERI. POINTERI ȘI TABLOURI”

UNIVERSITATEA TEHNICĂ A MOLDOVEI

PROGRAMAREA CALCULATOARELOR

TIPURI DE DATE POINTERI. POINTERI ȘI TABLOURI

Prelegere

Kulev Mihail, dr., conf. univ.

UNIVERSITATEA TEHNICĂ A MOLDOVEI

TIPURI DE DATE POINTERI. POINTERI ȘI TABLOURI

Conținutul prelegerii

1. Tipuri de date pointeri.

Vom defini noțiunea de pointer, vom afla pentru ce sunt folosite pointeri, modalități de declarare și de inițializare a pointerilor și vom studia doi operatori unari folosiți pentru lucrul cu pointeri.

2. Operații cu pointeri.

Vom studia și demonstra exemple referitor la operațiile cu pointeri: de incrementare și decrementare, adunarea și scăderea unui întreg la (din) un pointer, scăderea și compararea pointerilor.

3. Pointeri și tablouri în limbajul C.

Vom studia legatura strînsă între pointeri și tablouri, modalități de accesare elementelor tabloului utilizînd indexare și aritmetica pointerilor. Vom afla ce reprezintă pointer spre tablou, tablou de pointeri și pointer la pointer.

Exemple de cod.

Pe parcursul studierii întrebărilor menționate vom prezenta exemple de cod

în limbajul C.

1. Tipuri de date pointeri

Pointeri în limbajul C:

- sunt tipuri speciale de variabile (derivate din tipuri de date regulate) care **au ca valori adrese** ale unor alte variabile (adrese ale unor locaţii de memorie)
- permit **calcule cu adrese**
    - specifice limbajelor de asamblare
- sunt folosiţi în scopul scrierii unor **programe mai eficiente** atât din punct de vedere al **timpului de execuţie,** cât şi din punct de vedere al utilizării **memoriei calculatorului**.
- sunt în mod special utili la **alocarea dinamică a memoriei şi la apelul funcțiilor** (vom studia aceste aspecte la lecțiile ulterioare).

Pointerii reprezintă una din cele mai puternice caracteristici ale limbajului C,

dar şi periculoase. Dacă pointerii nu sunt iniţializaţi corect sau dacă

conţin valori incorecte pot determina blocarea calculatorului, sau să conducă

la erori greu de depistat.

Adresa care este stocată intr-o variabilă pointer poate fi:

- Adresa unei date de un anumit tip definit în program (pointer simplu)
- Adresa unei adrese de memorie etc. (pointer la pointer sau pointer dublu, triplu etc.)
- Adresa unui tablou alocat static sau dinamic (pointer simplu, pointer la pointer, pointer la tablou)
- Adresa unei zone cu conținut necunoscut (pointer de tip ”void*” sau pointer generalizat, generic)
- Adresa unei funcții (pointer la funcție).

Există o constantă de tip pointer cu numele **NULL** (valoarea zero) si care este compatibilă la atribuire cu orice tip pointer.

Deoarece adresele de memorie sunt numere întregi pozitive, tipurile pointer sunt diferite de tipurile întregi și au utilizări diferite.

In limbajul C tipurile pointer se folosesc pentru:

- Utilizarea mai eficientă de variabile simple și de tablouri.
- Acces la date alocate dinamic care nu pot fi adresate printr-un nume.
- Parametri (argumente) de funcții.

Declararea unei variabile de tip pointer:

tip*nume; // se creează o variabilă capabilă să conţină o adresă de memorie.

- ***tip*** este *tipul de bază* care specifică tipul variabilei simple sau tipul vectorului la care indică (punctează, pointează) pointerul;
- ***nume*** este numele variabilei pointer care poate conține adresa unei variabile sau unui vector de *tip de bază*;
- ***** (*asterix)* arată că variabila ***nume*** este variabila de tip pointer (***tip****).

*De menționat că spații înainte și/sau după asterixului * nu contează.*

Dimensiunea variabilei de tip pointer depinde de arhitectura și

sistemul de operare, pe care a fost compilat programul. Dimensiunea

unui pointer se determină cu sizeof(int*) = sizeof(double*) =

sizeof(void *) = ... și nu depinde de tipul variabilei asociate.

Exemple de declarare:

```
int *pi;    //  pi este pointer la tipul int, sau pointer de tip int* și poate conține
//adresa unei variabile simple sau adresa unui vector de tip int.

float* pf;  //  pf este pointer la tipul float, sau pointer de tip float* și

// poate conține adresa unei variabile simple sau adresa unui vector de tip float.

char * pc;  // pc este pointer la tipul  char și conține adresa unui caracter sau adresa
//unui șir de caractere.

void* pv;   // pv este pointer la tipul void și conține adresa unei zone de memorie cu
//conținut de tip necunoscut.

int** pp;   // pp este pointer la tipul pointer la int și conține adresa unui pointer la
//tip întreg sau adresa unui vector de pointeri la tipul întreg.

Atunci când se declară mai multe variabile pointer de același

tip, nu trebuie omis asteriscul, care arată că este un pointer.

int *p, m; int *a, *b ; // a și b de tip pointer.
```

Operatori unari & și *

Pentru lucrul cu pointeri limbajul C oferă doi operatori unari:

Operatorul de referențiere (de luare adresei) & determină adresa de memorie a variabilei pe care o precede.

Ex.: **&x** - adresa variabilei **x** (de fapt adresa primului octet al locaţiei de memorie în care este stocată variabila **x**)

Operatorul de dereferențiere (indirectare) * determină valoarea de la adresa pe care o precede (acces inderect la conținutul memoriei utilizând pointer).

Ex.: ***p** - valoarea de la adresa la care pointează pointerul **p**.

Operatorul * poate fi folosit în partea stângă a unei instrucţiuni

de atribuire pentru a atribui indirect unei variabile o nouă

valoare folosind un pointer (*p este aliasul denumirii variabilei).

Operatori unari & și *

Pentru ca dereferențierea să aibă loc cu succes, pointer-ul trebuie să indice o adresă de memorie validă, la care programul are acces. Această adresă poate fi adresa unei variabile declarate în prealabil sau adresa unui bloc de memorie alocat dinamic.

Este indicată inițializarea pointerilor cu constanta NULL, compatibilă cu orice tip de pointer, care indică, prin convenție, un pointer neinițializat.

O eroare frecventă este utilizarea unei variabile pointer, care nu a primit

o valoare (o adresă de memorie) prin atribuire sau prin inițializare

la declarare. Efectul este accesul la o adresă de memorie imprevizibilă,

chiar în afara spațiului de memorie ocupat de programul ce conține eroare.

Operatori unari & și *

Exemplu utilizării operatorilor unari & și * :

```
# include <stdio.h>
int main( ) {
int a, d, *b;
a = 55;    // variabila a primeşte valoarea 55
     // b primeşte adresa lui a`
*b = 135;
b=&d;
 // atribuirea lui a valorii 135 folosind pointerul b
printf("\n Valoarea lui a = %d \n", a);
printf("\n Adresa lui a = %p \n", b);
```

Exemplu de atribuire a pointerilor

Unui pointer poate fi atribuit un alt pointer printr-o instrucţiune de atribuire.

```
# include <stdio.h>
int main (){
int a = 1234;
int *b,  *c;
b = &a;
c = b;
printf("\n Valoarea lui a=%d \n", a);
printf("\n Adresa lui a este %p\n",c);
}
```

Exemple de utilizare a pointerilor

```
double a, *pd = NULL; // *pd = &a; (inițializarea pointerului pd)
a=8.406;
pd=&a;
int n=1, m=2, s[10];
int *pi;     // pi este un pointer la tipul întreg
pi = &n;     // pi pointează acum la variabila n
m = *pi;     // m are valoarea de la adresa lui n, adică 1
*pi = 0;     // n are acum valoarea 0
pi = &s[0]; // pi pointează acum la s[0], adică are ca
             // valoare adresa elementului s[0]
*pi=5*m;     // s[0] are acum valoarea 5*1=5
```

Un pointer este constrâns să pointeze la un anumit tip de date!!!

```
# include <stdio.h>
int main(void){
int *p;

float a, b;
p = &a; // greșit - p indică către un float
// fiind pointer la tip întreg
b = *p;
}
```

2. Operații cu pointeri

a) Operaţii de incrementare şi decrementare a pointerilor

Operatorul de incrementare ++ aplicat unui operand de tip pointer spre tipul oarecare, măreşte adresa, care este valoarea operandului cu numărul de octeţi necesari pentru a păstra o valoare de tipul respectiv.

Operatorul de decrementare ` aplicat unui operand de tip pointer spre tipul oarecare, micşorează valoarea operandului cu numărul de octeţi necesari pentru a păstra o valoare de tipul respectiv. De obicei, decrementările şi incrementările adreselor sunt mai rapide ca execuţie când se au în vedere prelucrări de tablouri.

Exemplu: incrementare și decrementare a pointerului

```
int tab[10];
int *p;
int i=0;
p=&tab[i];
p++;
//p conține adresa lui tab[1]
 //cu p se pot face referiri la orice
  // element de tablou
p=&tab[10]; // adresa după ultimul
  //element.
p--;
 // p contine adresa ultimului element
  //tab[9]
p=&tab[5];
p--;
// p contine adresa lui tab[4]
```

Operaţiile de incrementare/decrementare

pot fi aplicate asupra pointerului însuşi sau asupra obiectului pe care îl punctează. Trebuie procedat cu prudenţă dacă se încearcă să se incrementeze obiectul pe care îl punctează un pointer. De exemplu:

int *p;

instrucţiunea:

*p++;

obţine mai întâi valoarea pe care o punctează *p* şi apoi p este incrementat pentru a puncta elementul următor.

Pentru a incrementa obiectul pe care îl punctează *p* trebuie să folosim instrucţiunea:

(*p)++;

Parantezele fac ca valoarea punctată de *p* să

fie incrementată.

b) Adunarea şi scăderea unui întreg la (din) un pointer

Dacă **p** este un pointer spre tipul oarecare şi **n** un întreg, atunci se pot utiliza expresiile: **p+n** şi **p-n**.

Expresia **p+n** are ca valoare, valoarea lui **p** mărită cu produsul **r*n**, unde prin **r** am notat numărul de octeţi necesari pentru a păstra în memorie o dată de tipul respectiv.

Analogic, **p-n** este valoarea lui **p** micşorată cu **r*n.**

Exemplu:

float x,*pf=&x;

Dacă pf are valoarea 1024, atunci expresia (pf – 5)

va avea valoarea 1024 – 4*5 = 1024 - 20, adică 1004.

c) Scăderea pointerilor

int *pin, *pfin, *pmed;

Adunarea pointerilor nu este permisă!!!

pmed=(pin+pfin)/2; // greșit

pmed=pin+(pfin-pin)/2; //este o instrucțiune corectă (la pointerul pin se adaugă un întreg)

int num , *ptr1 ,*ptr2 ;

num=ptr2 - ptr1;   // rezultatul este un **întreg**,

//egal cu **numărul de obiecte (locații de memorie**)

//de același tip dintre valorile pointerilor

d) Compararea pointerilor

Doi pointeri de acelaşi tip pot fi comparaţi cu operatorii <, <=, ==, >= şi >. Un caz particular îl reprezintă comparaţia cu zero, care este permisă, zero fiind asimilat cu pointerul NULL.

```
int a,b, *p=&a,*q=&b;
double bb, *pp=&bb;
if(p<=q) printf("p<=q\n“);
else printf("p>q\n“);
 p=NULL;
if(p==NULL) printf("p==NULL\n“);  //p==NULL
if(p<pp) printf("???\n“); //error: no conversion
//from 'double *' to 'int *'
```

3. Pointeri și tablouri în limbajul C

Pointerii sunt strict legaţi de tablouri

- numele unui tablou este un pointer constant, care are ca valoare adresa primului element cu indexul 0 al tabloului respectiv.
- orice operaţie care se face folosind indicii tablourilor poate fi făcută prin folosirea pointerilor.

Dacă folosim numele unui tablou fără indice, generăm de fapt un pointer constant către primul element al tabloului. Acest lucru permite ca această valoare să fie atribuită unei variabile pointer şi astfel devine posibilă accesarea elementelor tabloului folosind pointerul. De exemplu:

```
int v[40], *p; p = v;
```

Variabila **p** a fost iniţializată cu adresa primului element din vector **v[0]**.

Pentru a accesa al treilea element din vector se poate scrie:

```
v[2] sau p[2] utilizînd indecsare
```

și

```
*(p + 2) sau  *(v+2) utilizînd pointeri (aritmetica pointerilor)
```

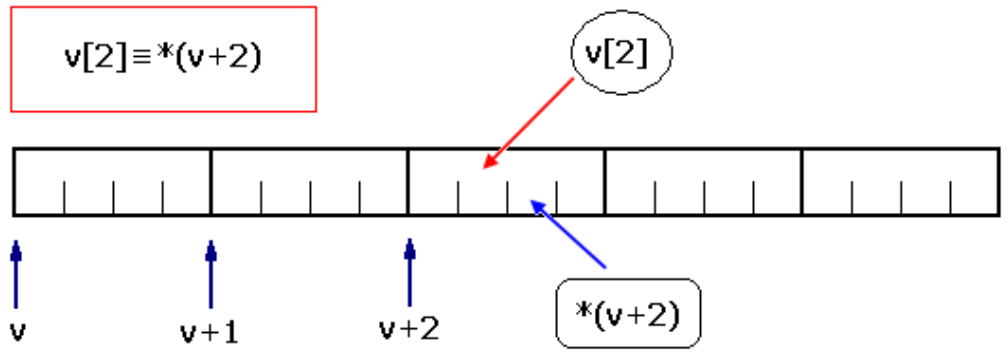
Indicii unui tablou încep de la 0. Pentru a avea acces la al treilea element, trebuie folosit indicele 2

pentru vector sau să adăugaţi 2 la pointerul **p**, deoarece **p** indică efectiv spre primul element din vector

(spre **v[0]**). Limbajul C oferă două metode de acces la elementele unui tablou: aritmetica pointerilor

sau indicii tablourilor. Prima este mai rapidă.

Echivalenţa v[2] cu *(v+2)



Există următoarele echivalenţe de notaţie pentru un vector cu *n* elemente:

Adresă		Valoare	
Notaţie indexată	Notaţie cu pointeri	Notaţie indexată	Notaţie cu pointeri
&a[0]	a	a[0]	*a
&a[1]	a+1	a[1]	*(a+1)
&a[i]	a+i	a[i]	*(a+i)
&a[n-1]	a+n-1	a[n-1]	*(a+n-1)

! În limbajul C avem următoarea echivalenţă interesantă: a[i] = i[a]

Într-adevăr: a[i]=*(a+i)=*(i+a)=i[a]

Parcurgerea unui tablou unidimensional utilizînd indexare și pointeri

Varianta 1

```
for (i=0; i < n; ++i)
    suma += a[i];
```

Varianta 2

```
for (i=0; i < n; ++i)
    suma += *(a+i);
```

Varianta 3

```
for (p=a; p < &a[n]; ++p)
    suma += *p;
```

Varianta 4

```
p=a;
for (i=0; i < n; ++i)
    suma += p[i];
```

Exemplu de afișare tabloului utilizînd indexare și pointeri:

```
# include <stdio.h>
int main(void){
  int *p, a[5] = {1,2,3,4,5};
  int i;
  p = a; //atribuie lui p adresa de început a tabloului
  printf("\nAfişarea tabloului folosind indicii tabloului \n");
  for (i =0; i <5; i++)
    printf("%d\t", a[i] );
    printf ("\nAfişarea tabloului folosind pointeri, nu indici de tabel \n");
    for (i = 0; i <5; i ++)  //for(p=a;p<&a[5];p++)
    printf("%d\t",*(p + i) );
    // printf("%d\t", *p );
    return 0;}
```

Exemplu: adunarea elementelor unui vector folosind pointeri

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
  float x[100], *y,v;
  double s=0.0,med;
  int i;
  //genereaza    aleator    elementele
//vectorului  x
  for(i=0;i<100;i++)
    {
       v=rand()/pow(2,15);
       x[i]=1000*v-500;
       //printf("\n %5.3f \t",x[i]);
    }
     printf("\n");
//se aduna elementele sirului
//x;   pointerul   y   a   fost
//initializat cu adresa
/primului element al sirului x

for(y=x;y!=&x[100];)
    s+=*y++;
  med=s/100;
  printf("\n   s=%lg\t   media=
%lg \n",s,med);

  return 0;
}
```

Tablori bidimensionale. Pointer la tablou.

int a[2][3];

Expresii echivalente cu a[i][j]

***(a[i] + j)**

***((*(a + i)) + j)**

(*(a + i))[j]

***(&a[0][0] + m*i +j)**

coloane / linii	j = 0	j= 1	j = 2
i = 0	a[0][0]	a[0][1]	a[0][2]
i = 1	a[1][0]	a[1][1]	a[1][2]

----

a[i] sau *(a+i) – adresa liniei i sau *(a+0) = *a

int (*p)[3]; // pointer la tablou de tip integer cu trei elemente

p=a; // a este pointer constant la tablou de tip integer cu trei elemente

// a – adresa tabloului bidimensional

// a și *a sunt pointeri de tipuri diferite

Tablouri de pointeri

Pointerii pot fi organizaţi sub forma de tablouri ca oricare alt tip de date. Declararea unui tablou de pointeri de tip int de 10 elemente este:

```
int *x[10]; int num;
```

Pentru a atribui adresa unei variabile de tip întreg cu numele **num** elementului al doilea al tabloului de pointeri, se va scrie:

```
x[1] = &num;
```

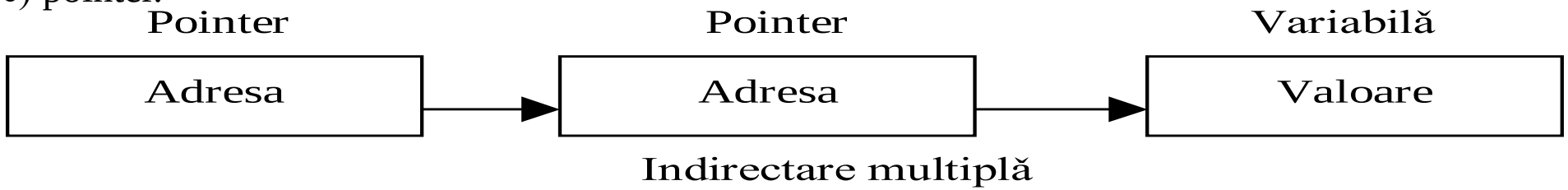
iar pentru a obţine valoarea **num** se va scrie:

```
*x[1]
```

Tablourile de pointeri sunt folosite de obicei pentru a păstra pointeri la liniile tabloului bidimensional alocat dinamic și pointeri către şiruri de caractere.

Pointer la pointer

Este posibil să avem un pointer care să puncteze un alt pointer. Această situaţie este denumită indirectare multiplă sau pointer la (către) pointer.



Indirectare multiplă

Când un pointer punctează un alt pointer, primul pointer conţine adresa celui de al doilea pointer, care punctează locaţia ce conţine obiectul, valoarea dorită. Pentru a declara un pointer către un alt pointer, trebuie plasat un asterix suplimentar în faţa numelui pointerului.

De exemplu: **int **pp;**

pp este un pointer către un alt pointer de tip int*, şi nu este un pointer către un întreg.

Accesarea valorii printr-un pointer către un alt pointer, cere ca operatorul asterix să fie aplicat de două ori. De exemplu:

```
p = &ch; // asignează lui p adresa lui ch
pp = &p; //asignează lui pp adresa lui p
```

**pp = ‘c’; // asignează lui ch valoarea ‘c’ folosind indirectare multiplă.

Probleme propuse spre rezolvare cu utilizarea pointerilor pentru parcurgerea tablourilor

1. Fie dat tabloul bidimensional (matrice) cu dimensiunele **n*m**. Să se formeze tabloul unidimensional (vector), elementele căruia vor fi elementele de pe perimetrul matricei. Utilizați pointeri pentru parcurgerea tablourilor.

2. Fie dat tabloul unidimensional cu **n** elemente. Să se scrie un program care inserează un element nou în tablou. Utilizați pointeri pentru parcurgerea tablourilor.

3. Să se scrie un program care citeşte de la tastatură un număr pozitiv **n** împreună cu alt număr pozitiv **max**. Apoi programul va iniţializa un vector cu numere aleatoare în intervalul [**0..max-1**]. Sortaţi vectorul folosind metoda preferată, afişându-i conţinutul atât înainte, cât şi după ce sortarea a avut loc. Utilizați pointeri pentru parcurgerea vectorului.

4. Pentru tabloul bidimensional dat din n linii și m coloane să se determine

suma elementelor negative din fiecare linie și numărul elementelor pozitive

din fiecare coloană. Utilizați pointeri pentru parcurgerea tablourilor.

Tutoriale online pentru tema prelegerii:

1. https://ocw.cs.pub.ro/courses/programare/laboratoare/lab08
2. http://andrei.clubcisco.ro/cursuri/1pc/curs/1/Curs%205_6.pdf
3. https://staff.fmi.uvt.ro/~victoria.iordan/Programare_MI/Curs8.pdf
4. http://www.cs.ucv.ro/staff/gmarian/Programare/cap7_Pointeri.pdf
5. https://www.math.uaic.ro/~necula/down_files/cpp2017/pointeri_i_2017.pdf
6. http://www.phys.ubbcluj.ro/~vasile.chis/cursuri/info/c07ppt.pdf
7. http://www.euroinformatica.ro/documentation/programming/Documentatie/curs%20scoala%20C++/curs%20scoala%20C++/

VĂ MULȚUMESC PENTRU ATENȚIE!

MULTĂ SĂNĂTATE ȘI SUCCESE!