PROIECTAREA ALGORITMILOR

Adrian Runceanu

Curs 2

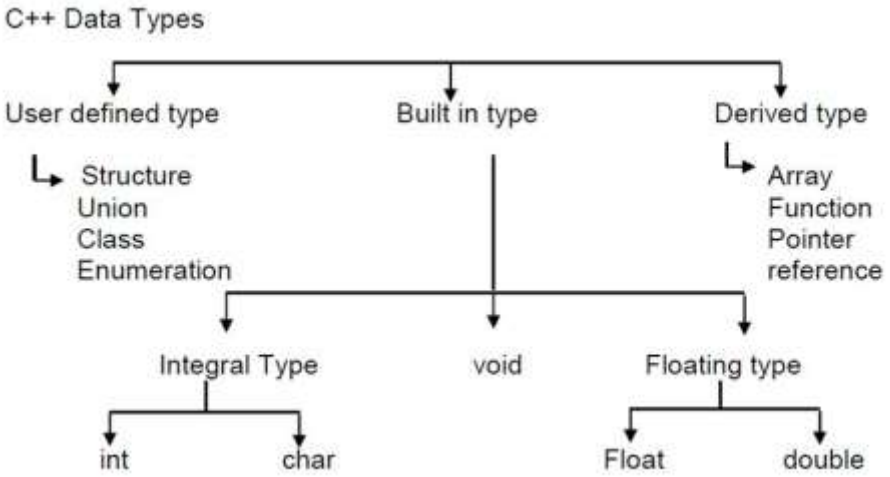
Alocarea dinamică de memorie în C++

Conţinutul cursului

1. Tipuri de date
2. Conceptul de pointer
3. Operatori specifici pointerilor
4. Aritmetica pointerilor
   - 4.1. Pointeri şi tablouri
   - 4.2. Echivalenţe de scriere
   - 4.3. Expresii compacte cu pointeri
5. Legătura dintre pointeri şi tablouri

1. Tipuri de date

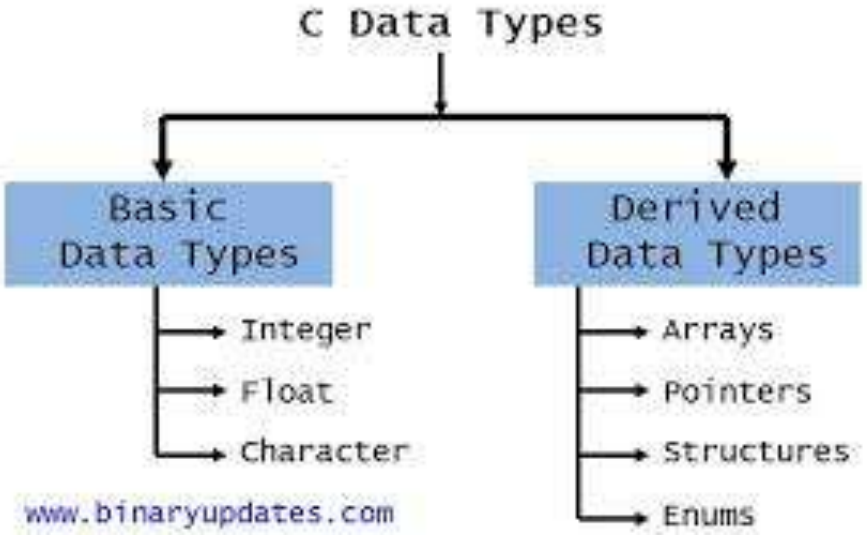
- Informaţia care stă la dispoziţia calculatorului constă dintr-o mulţime de date care descriu lumea reală prin abstractizare.
- O abstractizare este de fapt o simplificare a faptelor prin reţinerea caracteristicilor relevante ale obiectelor reale.
- O ***dată*** este *orice entitate asupra căreia poate opera calculatorul*.
- Astfel apare noţiunea de ***tip de date.***



https://slideplayer.com/slide/5755418/

1. Tipuri de date

- Prin ***tip de date*** se înţelege *o mulţime de valori **D** care formează **domeniul** tipului, împreună cu o **mulţime de operatori** pe acest domeniu*.
- În cazul când elementele domeniului sunt valori compuse din mai multe componente atomice, tipul de date obţinut se numeşte *tip de date structurat* sau *structură de date*.



1. Tipuri de date

- Din punct de vedere al implementării lor, putem vorbi despre:

  1. date implementate ***static***

  2. date implementate ***dinamic***

- Criteriul de clasificare este dat de *modul de alocare a memoriei interne*.

1. Tipuri de date

1. *Pentru structurile statice* memoria se alocă la începutul programului şi rămâne alocată cât timp programul se execută.
   - Astfel, caracteristicile variabilelor statice sunt bine definite, cunoscute şi fixe.
   - *Structura, tipul şi adresa de memorie nu se pot modifica în timpul execuţiei programului*.
   - De asemenea, variabilele statice sunt referite prin numele lor, fiecărui nume asociindu-i-se o adresă fizică de memorie.

1. Tipuri de date

- Unei variabile statice *i se poate modifica doar valoarea, nu şi adresa din memoria internă*.
- Scopul definirii tipurilor de date este acela de:
  - a fixa domeniul valorilor pe care le pot lua aceste variabile
  - de a preciza structura, dimensiunea şi amplasarea zonelor de memorie ce le sunt asociate
- Deoarece toate aceste elemente sunt fixate de la început de catre compilator, astfel de variabile şi structurile de date aferente lor se numesc ***statice***.

1. Tipuri de date

Din motive de memorare a datelor în calculator, în limbajele de programare, datele sunt formate din două clase:

a) date elementare

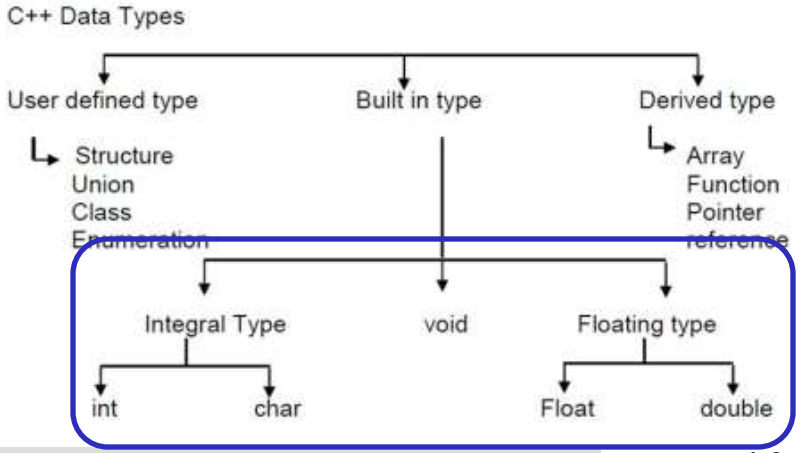
b) date compuse

1. Tipuri de date

a) *Datele elementare* (aşa numitele tipuri scalare) sunt reprezentate în structura internă a sistemului prin şiruri de biţi asupra cărora acţionează mecanismul de adresare şi care pot constitui operanzii direcţi ai operaţiilor sistemului de calcul.

Date elementare se considera scalarii predefiniţi (built in type):

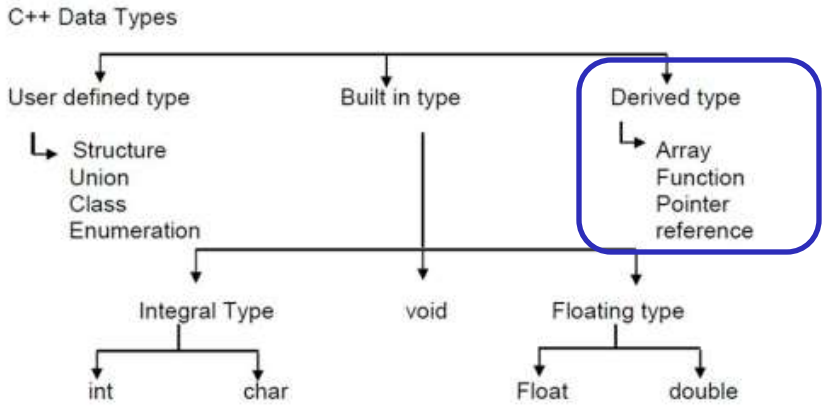
1) întregi
2) reali
3) caracter



1. Tipuri de date

b) Datele compuse sunt determinate de o descriere a tipului componentelor lor şi prin indicarea metodelor de structurare a acestora.
Sunt considerate date compuse:

1) tablourile
2) înregistrările(structurile)
3) şirurile de caractere
4) fişierele, etc



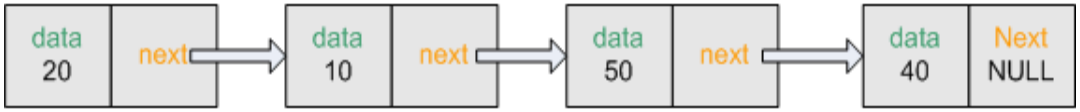
1. Tipuri de date

2. Pentru structurile dinamice *se alocă memorie în timpul execuţiei programului, iar când nu mai este necesară, memoria se eliberează*.

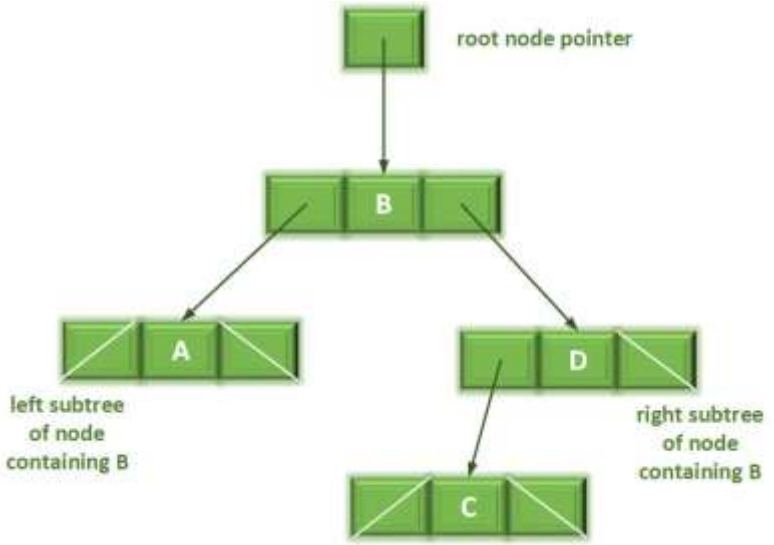
- Totuşi, variabilele dinamice au un tip bine precizat încă din faza de compilare, însă ele pot fi alocate dinamic, pot fi utilizate prin adresa lor din *heap* şi pot fi distruse dacă nu mai sunt utile.
- *Aceste variabile pot fi referite printr-o variabilă de tip referinţă (pointer) ce conţine adresa variabilei dinamice*.

1. Tipuri de date

Structurile dinamice cuprind:
listele şi ***arborii***



Linked list

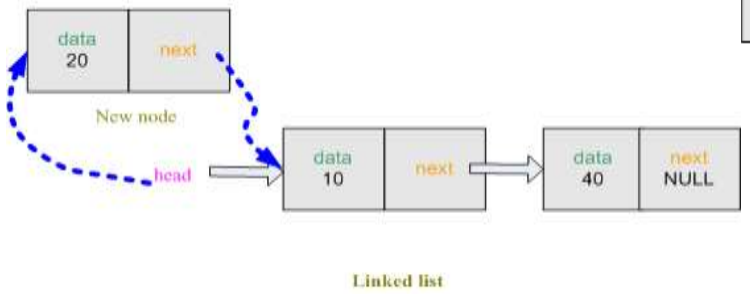


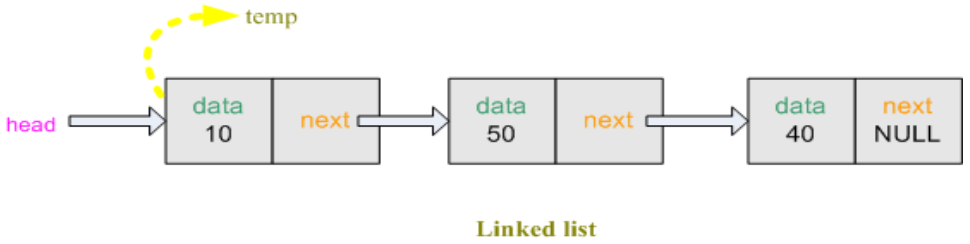
https://www.codeproject.com/Articles/24684/How-to-create-Linked-list-using-C-C

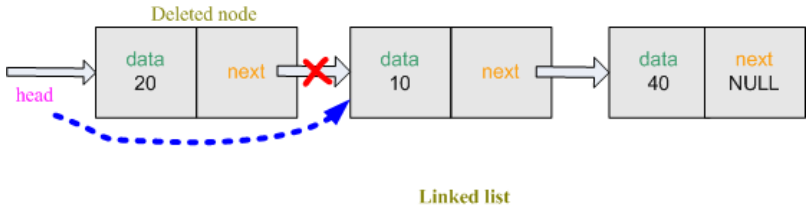
http://www.infocodify.com/cpp/cpp_binary_tree

1. Tipuri de date

1. Lista este *o structură de date definită cu ajutorul unor relaţii de ordine asupra înregistrărilor*.



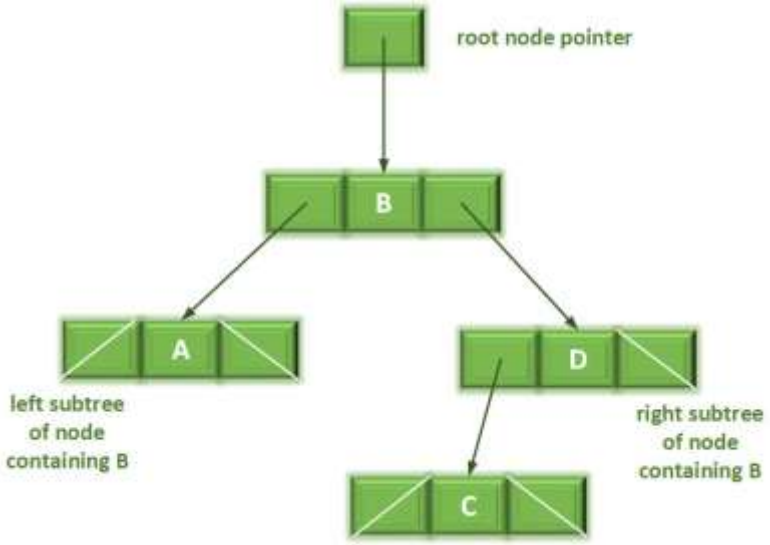


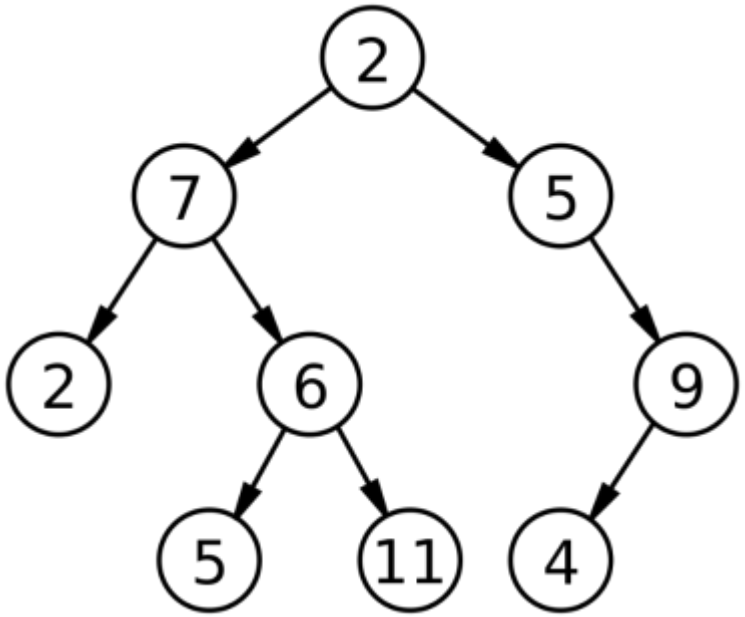


https://www.codeproject.com/Articles/24684/How-to-create-Linked-list-using-C-C

1. Tipuri de date

2. Arborele este *o structură de date definită cu ajutorul unor relaţii de ordine asupra listelor*.





http://www.infocodify.com/cpp/cpp_binary_tree

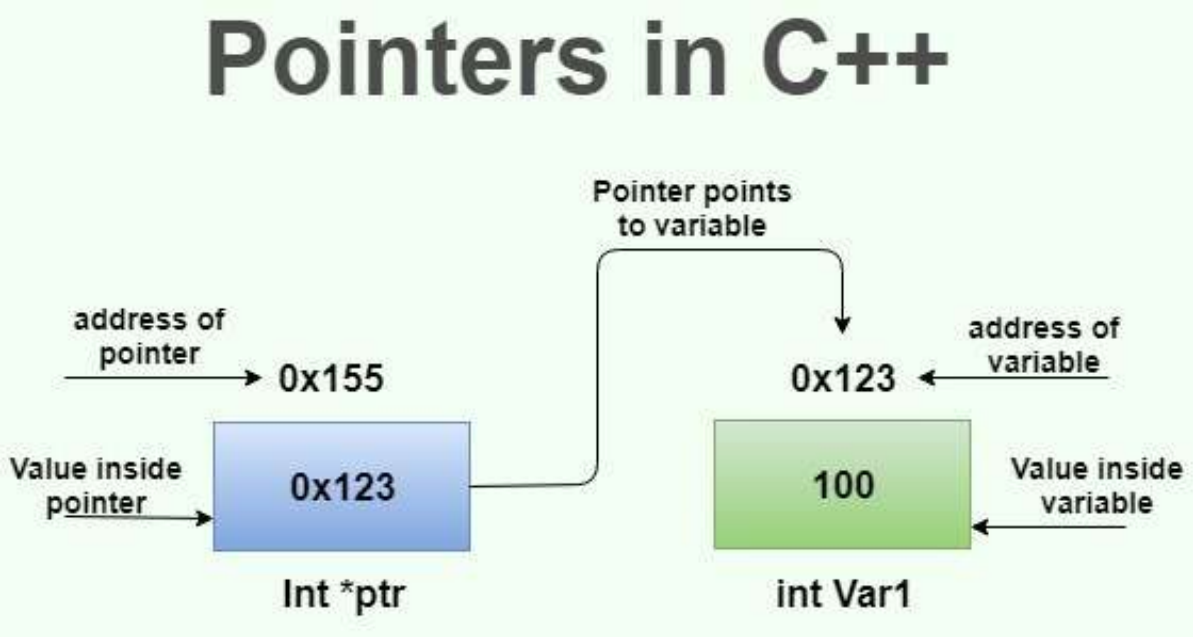
https://en.wikipedia.org/wiki/Tree_(data_structure)

Conţinutul cursului

1. Tipuri de date
2. Conceptul de pointer
3. Operatori specifici pointerilor
4. Aritmetica pointerilor
   - 4.1. Pointeri şi tablouri
   - 4.2. Echivalenţe de scriere
   - 4.3. Expresii compacte cu pointeri
5. Legătura dintre pointeri şi tablouri

2. Conceptul de pointer

- Un **pointer (variabila referință)** este o variabilă care are ca valoare adresa unui obiect cu care operează limbajul C/C++ (variabilă, constantă, funcţie).

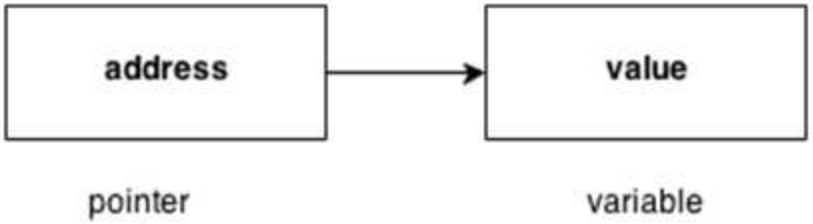


https://simplesnippets.tech/cpp-pointers-concept-with-example/

2. Conceptul de pointer

Folosirea pointerilor este determinată de două motive şi anume:

1. este singura modalitate de a efectua anumite calcule
2. folosirea pointerilor duce la generarea unui cod sursă de program mai eficient şi mai compact



2. Conceptul de pointer

Astfel putem enumera 2 (două) situaţii în care limbajul C/C++ foloseşte pointerii, şi anume:

- atunci când *se transmite unei funcţii o matrice sau un şir de caractere*

- sau când se transmit parametri prin referinţă, adică atunci când vrem să modificăm variabilele respective

2. Conceptul de pointer

Declararea unei variabile de tip pointer se face astfel:

```
tip *nume_pointer;
```

Unde:

- **tip** reprezintă oricare dintre tipurile limbajului C/C++ şi indică tipul variabilei a cărei adresă este memorată de variabila pointer.
- **nume_pointer** reprezintă numele ales de utilizator pentru a identifica pointerul respectiv

•operatorul '*' este obligatoriu la declarare

2. Conceptul de pointer

Exemple:

```
char *pc;     // pointer de tip caracter
int  *pi;     // pointer de tip întreg
float *pf;    // pointer de tip real
```

Observaţie:

- Pot exista şi pointeri fără tip, care se declară astfel:

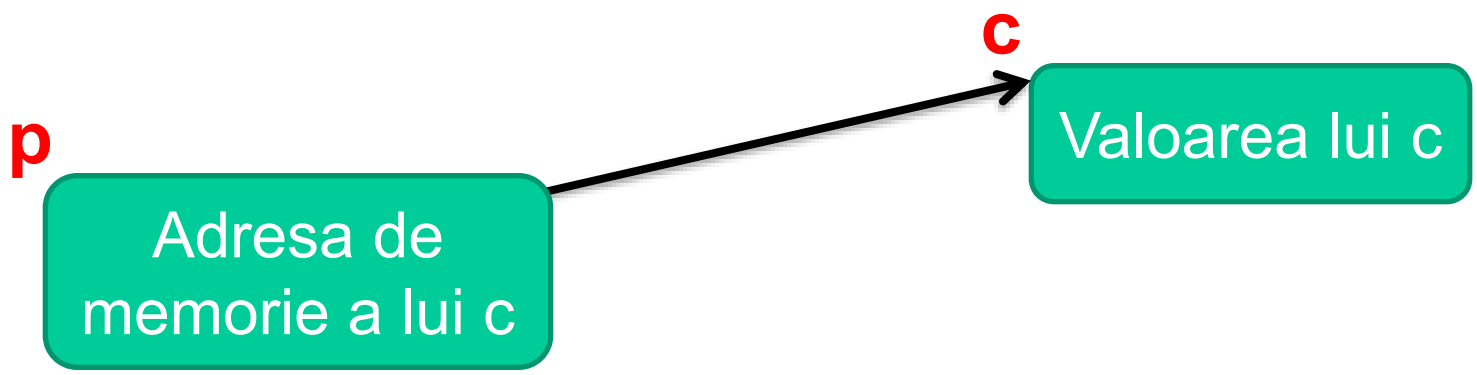
```
void *nume_pointer;
```

- Aceştia se utilizează pentru ca în program să primească *valoarea oricărui tip de dată*.

2. Conceptul de pointer

Un pointer este un grup de octeți (adesea doi sau patru) care poate conține o adresă.

Astfel dacă c este un char și p un pointer care trimite către el (se refera la variabila c), atunci am putea reprezenta situația astfel:



Conţinutul cursului

1. Tipuri de date
2. Conceptul de pointer
3. Operatori specifici pointerilor
4. Aritmetica pointerilor
   - 4.1. Pointeri şi tablouri
   - 4.2. Echivalenţe de scriere
   - 4.3. Expresii compacte cu pointeri
5. Legătura dintre pointeri şi tablouri

3. Operatori specifici pointerilor

Operatorii folosiţi pentru lucrul cu pointerii sunt operatorii unari:

1. Operatorul ‘ & ‘ (operator de adresă) – *se aplică unei variabile* furnizând adresa acelei variabile.

2. Operatorul ‘ * ‘ (operator de redirectare) - *se aplică unui pointer* şi furnizează obiectul referit de acel pointer.

3. Operatori specifici pointerilor

1. Operatorul ‘ & ‘ (operator de adresă) – *se aplică unei variabile furnizând adresa acelei variabile.*

Exemplu: Următorul program arată cum se poate folosi operatorul de adresă pentru a afişa adresele unor variabile de tipuri diferite.

3. Operatori specifici pointerilor

```
#include <iostream>

using namespace std;

int main()
{
        int indice=1;
        float salariu_dolari=20000.0;
        long salariu_lei=30000000L;
        cout<<"\nAdresa variabilei indice este "<<&indice;
        cout<<"\nAdresa variabilei salariu_dolari este
"<<salariu_dolari;
        cout<<"\nAdresa variabilei salariu_lei este "<<&salariu_lei;
}
```

3. Operatori specifici pointerilor

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6      int indice=1;
7      float salariu_dolari=20000.0;
8      long salariu_lei=30000000L;
9      cout<<"\nAdresa variabilei indice este "<<&indice;
10     cout<<"\nValoarea variabilei salariu_dolari este "<<salariu_dolari;
11     cout<<"\nAdresa variabilei salariu_lei este "<<&salariu_lei;
12
13 }
14
```

Execute Mode, Version, Inputs & Arguments

GCC 9.1.0

CommandLine Arguments

Result

CPU Time: 0.00 sec(s), Memory: 3504 kilobyte(s)

```
Adresa variabilei indice este 0x7ffeb173ef88
Valoarea variabilei salariu_dolari este 20000
Adresa variabilei salariu_lei este 0x7ffeb173ef90
```

3. Operatori specifici pointerilor

Iniţializarea pointerilor se poate face cu o valoare de adresă de memorie, prin folosirea operatorului de adresă '**&**':

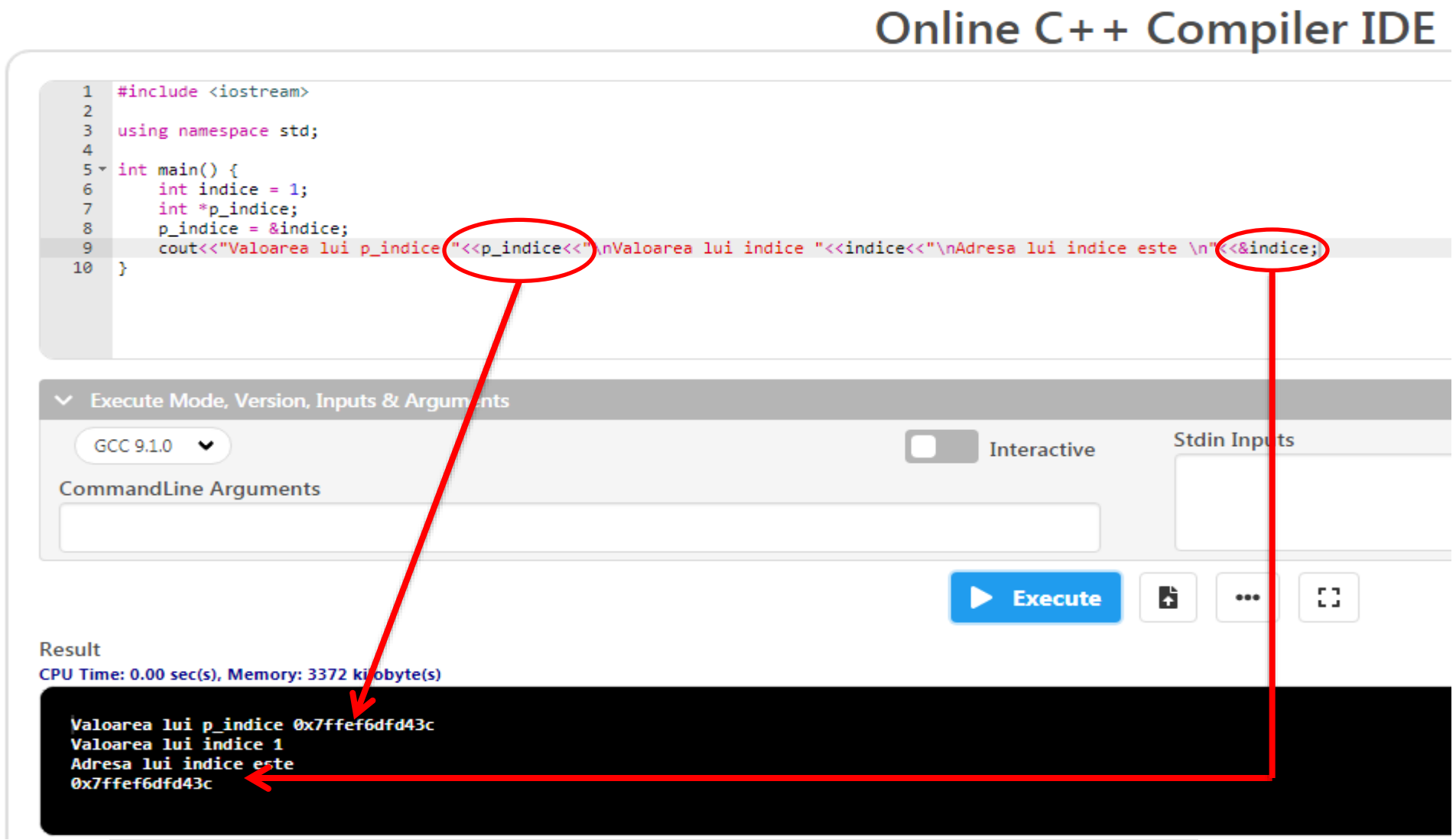
```
nume_pointer = &variabila;
```

Exemplu: Următorul program declară o variabilă de tip pointer şi atribuie adresa unei variabile de tip întreg şi apoi afişează valoarea variabilei pointer împreună cu adresa variabilei de tip întreg.

3. Operatori specifici pointerilor

```
#include <iostream>
using namespace std;
int main()
{
    int indice = 1;
    int *p_indice;
    p_indice = &indice;
    cout<<"Valoarea lui p_indice
    "<<p_indice<<"\nValoarea lui indice
    "<<indice<<"\nAdresa lui indice este \n"<<&indice;
}
```

3. Operatori specifici pointerilor



3. Operatori specifici pointerilor

2. Operatorul ‘ * ‘ (operator de redirectare) - *se aplică pointerilor şi furnizează obiectul referit de acel pointer.*

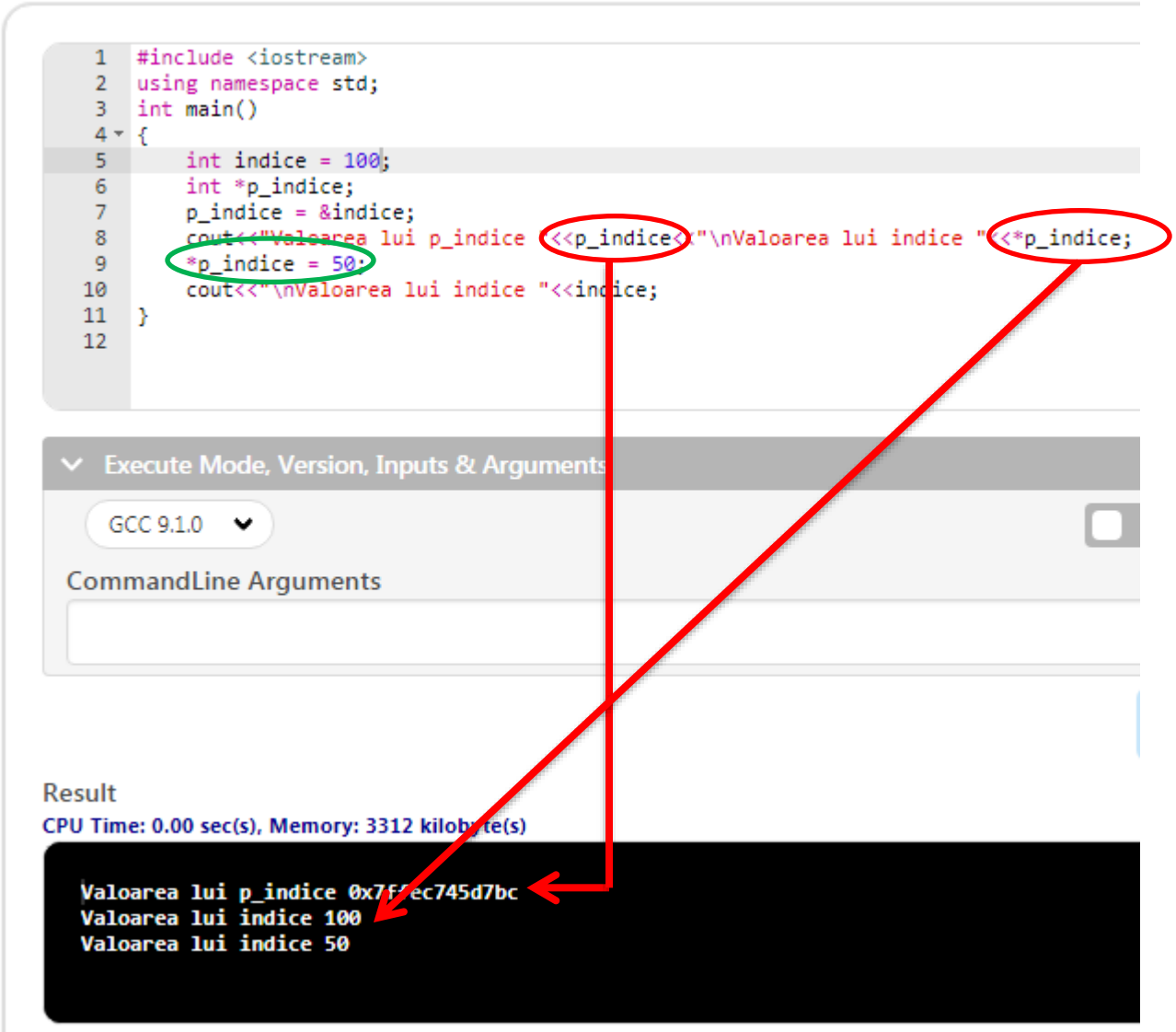
Exemplu:

Următorul program atribuie pointerului adresa unei variabile de tip întreg, afişează adresa sa împreună cu valoarea memorată la adresa pe care o are pointerul, apoi modifică valoarea variabilei prin intermediul pointerului şi o afişează.

3. Operatori specifici pointerilor

```
#include <iostream>
using namespace std;
int main()
{
    int indice = 1;
    int *p_indice;
    p_indice = &indice;
    cout<<"Valoarea lui p_indice "<< p_indice <<
    "\nValoarea lui indice " << *p_indice;
    *p_indice = 50;
    cout<<"\nValoarea lui indice “ << indice;
}
```

3. Operatori specifici pointerilor



Conţinutul cursului

1. Tipuri de date
2. Conceptul de pointer
3. Operatori specifici pointerilor
4. Aritmetica pointerilor
   - 4.1. Pointeri şi tablouri
   - 4.2. Echivalenţe de scriere
   - 4.3. Expresii compacte cu pointeri
5. Legătura dintre pointeri şi tablouri

4. Aritmetica pointerilor

4.1. Pointeri şi tablouri

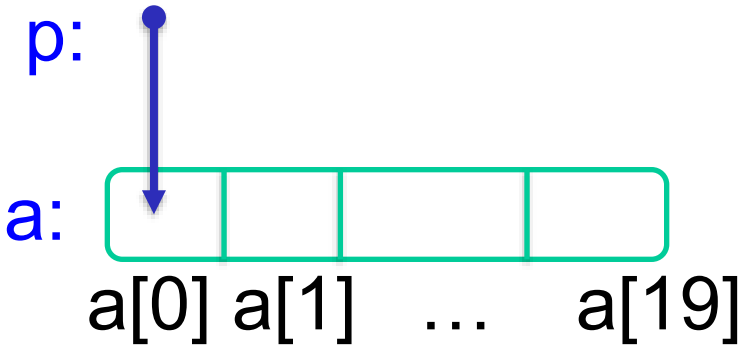
• În limbajul C++ există o stânsă legătură între pointeri şi tablouri, astfel încât orice problemă care se poate rezolva cu tablouri, poate fi rezolvată şi cu pointeri.

• Putem spune că un pointer care reţine adresa unui tablou, *reţine de fapt adresa primului element al tabloului*.

4. Aritmetica pointerilor

Exemplu:

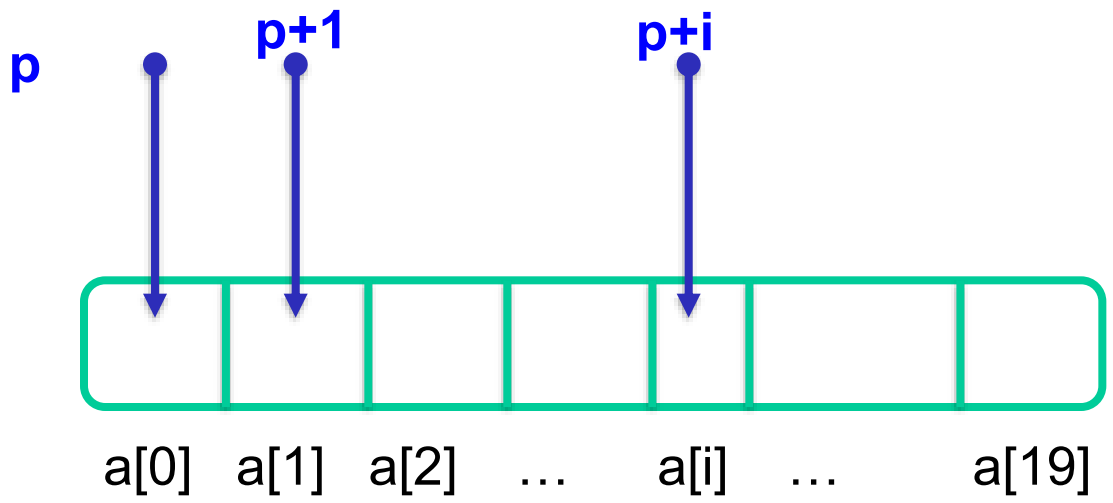
```
int a[20], *p;

p=&a[0];
```



4. Aritmetica pointerilor

Dacă p este un pointer către un element al unui tablou oarecare, atunci:

- *p+1 este un pointer către elementul următor al tabloului,*
- *p+i se referă la elementul al i-lea,*
- *iar p-i la elementul al i-lea înainte de p.*



4. Aritmetica pointerilor

Observaţie:

➢În expresia **p+i** ne apare faptul că exemplificăm pe un tablou cu elemente de tip întreg. De unde putem concluziona că:

➢*Dacă p este un pointer către un obiect (tip_obiect *p), atunci p+i va fi un pointer către al i-lea obiect aflat după obiectul referit de p.*

➢Numele unei variabile de tip tablou este de fapt adresa primului element (adică adresa elementului 0 al tabloului).

Deci:

p = &a[0]; ⇔ **p = a;**

4. Aritmetica pointerilor

Dacă p şi q sunt doi pointeri către elemente ale aceluiaşi tablou (adică **p = &a[i]** şi **q = &a[j]**), atunci putem efectua următoarele operaţii:

1) Comparaţii:

p == q// adică se compară i cu j
p != q
p < q
p <= q
p > q
p >= q

2) p – q // adică numărul de elemente ale tabloului, care se află între elementul referit de p şi elementul referit de q

4. Aritmetica pointerilor

Exemplu: Următorul program afişează cu ajutorul unor tablouri de tipuri diferite, diferenţa între doi pointeri.

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      char s[]="Proiectarea Algoritmilor", *cp, *cq;
6      int tab_integer[] = {1, 2, 3, 4,5 }, *ip, *iq;
7      long tab_long[] = {666, 777, 888}, *lp, *lq;
8      float tab_float[] = {1.2, 3.44, 5.78, 6.1234, 2.333}, *fp, *fq;
9      double tab_double[] = {1.111, 2.222, 3.333, 4.444}, *dp, *dq;
10     cp = cq = s;
11     cp++;
12     cout<<"cp - cq = " << cp - cq;
13     ip = iq = tab_integer;
14     ip++;
15     cout<<"\nip - iq = " << ip - iq;
16     lp = lq = tab_long;
17     lp++;
18     cout<<"\nlp - lq = " << lp - lq;
19     fp = fq = tab_float;
20     fp++;
21     cout<<"\nfp - fq = " << fp - fq;
22     dp = dq = tab_double;
23     dp++;
24     cout<<"\ndp - dq = " << dp - dq;
25     return 0;
26 }
```

Result

CPU Time: 0.00 sec(s), Memory: 3412 kilobyte(s)

```
cp - cq = 1
ip - iq = 1
lp - lq = 1
fp - fq = 1
dp - dq = 1
```

4. Aritmetica pointerilor

4.2. Echivalenţe de scriere

p = &a[0]	⇔	p = a
p + 1	⇔	&a[1]
*(p + 1)	⇔	a[1]
p + i	⇔	&a[i]
a + 1	⇔	&a[1]
*(a + 1)	⇔	a[1]
a + i	⇔	&a[i]
*(a + i)	⇔	a[i]

Deci, putem scrie:

a[i] ⇔ *(a+i) ⇔ *(i +a) ⇔ i[a]

4. Aritmetica pointerilor

- Din echivalenţele de mai sus, putem desprinde o regulă a compilatorului C++, şi anume:

- Orice expresie de forma a[i] este transformată imediat într-o expresie ***(a+i)**, adică o expresie în care apare **un pointer** (adică a) şi **un deplasament** (adică i).

4. Aritmetica pointerilor

Exemplu:

Următorul program prezintă ultima regulă obţinută mai sus şi anume **a[i] ⇔ i[a]**.

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int tablou_integer[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
6      int i = 7;
7      cout<<"tablou_integer[i] = "<<tablou_integer[7];
8      cout<<"\ni[tablou_integer] = " << i[tablou_integer];
9      return 0;
10 }
```

Result

CPU Time: 0.00 sec(s), Memory: 3372 kilobyte(s)

```
tablou_integer[i] = 7
i[tablou_integer] = 7
```

4. Aritmetica pointerilor

Observaţie:

- Succesiunea operatorilor ‘**&***‘ aplicată unui pointer cu deplasament are ca efect (evaluarea se face de la dreapta la stanga):
  - la primul pas, selectarea conţinutului (valorii) de la respectiva adresă
  - iar la al doilea pas, selectarea adresei respectivului conţinut
- Deci aplicând această succesiune de operatori de obţine tot obiectul.
- Nu acelaşi lucru de poate spune despre succesiunea ‘***&**‘.

4. Aritmetica pointerilor

Exemplu: Următorul program arată echivalenţa dintre un tablou si un pointer care îl referă.

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int tablou_integer[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
6      int *p = tablou_integer;
7      cout<<tablou_integer + 3<<"  "<<&*(tablou_integer + 3)<<"\n";
8      cout<<p + 3<<" " << &*(p + 3);
9      return 0;
10 }
```

Result

CPU Time: 0.00 sec(s), Memory: 3468 kilobyte(s)

```
0x7fffe39378dc  0x7fffe39378dc
0x7fffe39378dc 0x7fffe39378dc
```

4. Aritmetica pointerilor

4.3. Expresii compacte cu pointeri

➢Cu ajutorul pointerilor se pot scrie expresii compacte în care pointerii apar împreună cu operatorii de ***incrementare (++)*** şi ***decrementare (--)***.

➢Astfel pot apărea următoarele situaţii:

EXPRESIA	OPERATIA	CE SE MODIFICĂ
*p++	postincrementare	pointerul
*p--	postdecrementare	pointerul
*++p	preincrementare	pointerul
*--p	predecrementare	pointerul
++*p	preincrementare	obiectul
--*p	predecrementare	obiectul
(*p)++	postincrementare	obiectul
(*p)--	postdecrementare	obiectul

4. Aritmetica pointerilor

- ***Expresiile din prima grupă modifică pointerul şi nu obiectul la care se referă acesta.***
- Acest tip de expresii îmbunătăţesc viteza de execuţie a unui program şi se recomandă utilizarea acestora în ciclurile repetitive.
- ***Expresiile din a doua grupă modifică obiectul referit, astfel încât operatorii ++ şi -- sunt aplicaţi de către compilator asupra obiectului şi nu asupra pointerului.***
- Deci, nu putem utiliza în aceste expresii un pointer la o structură, uniune sau o funcţie.

Exemplu: Următorul program prezintă modul în care acţionează operatorii ++ şi -- asupra expresiilor compacte cu pointeri.

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int a = 1, b = 2, c = 0;
6      int *x = &a, *y = &b;
7
8      cout<<++(*x)<<"\t";
9      cout<<(*y)--<<"\t";
10     cout<<++c<<"\n";
11
12     a++;     b = 2;  c-=(*x) + b++;
13     cout<<++(*x)<<"\t";
14     cout<<(*y)--<<"\t";
15     cout<<++c<<"\n";
16
17     x = y;
18     cout<<++(*x)<<"\t";
19     cout<<(*y)--<<"\t";
20     cout<<++c<<"\n";
21
22     cout<<a<<"\t"<<b<<"\t"<<c<<"\n";
23     return 0;
24 }
25
```

Result

CPU Time: 0.00 sec(s), Memory: 3436 kilobyte(s)

```
2   2   1
4   3   -3
3   3   -2
4   2   -2
```

Conţinutul cursului

1. Tipuri de date
2. Conceptul de pointer
3. Operatori specifici pointerilor
4. Aritmetica pointerilor
   - 4.1. Pointeri şi tablouri
   - 4.2. Echivalenţe de scriere
   - 4.3. Expresii compacte cu pointeri
5. Legătura dintre pointeri şi tablouri

5. Legătura dintre pointeri şi tablouri

- În primul rând, **numele unui tablou este un pointer** deoarece el *are ca valoare adresa primului său element*.
- Totuşi există o diferenţă între numele unui tablou şi o variabilă de tip pointer, deoarece dacă unei variabile de tip pointer i se poate atribui o adresă, acest lucru nu se poate realiza pentru numele unui tablou, el fiind întotdeauna un pointer spre primul element al tabloului, deci un **pointer constant**.

5. Legătura dintre pointeri şi tablouri

Exemple:

1) Fie **int a[100]** un tablou.

Următoarele secvenţe sunt echivalente, deoarece fiecare din ele reprezintă *adresa primului element din tablou*:

1) **a**
2) **&a[0]**
3) **&*a**

- Prima expresie reprezintă cele spuse înainte conform definiţiei.
- În a doua expresie, **&a[0]** reprezintă adresa elementului de index 0 al tabloului. Cum 0 reprezintă poziţia primului element al tabloului, rezultă că a doua expresie reprezintă adresa de început a tabloului.
- Compunerea operatorilor **&** şi ***** îşi anulează semnificaţia, deci a treia declaraţie este echivalentă tot cu **a**.

5. Legătura dintre pointeri şi tablouri

2) Fie declaraţiile:

int a[10],b[10],*c;

……

a=b; // greşit deoarece a şi b sunt pointeri constanţi

c=a; // corect deoarece c este un pointer variabil

b=c; // greşit deoarece chiar dacă c este un pointer variabil, pointerul din partea stânga a atribuirii, b, este constant

5. Legătura dintre pointeri şi tablouri

Tablouri de pointeri şi pointeri la tablouri

- Pointerii fiind variabile, pot fi folosiţi pentru a forma alte tipuri de date compuse.
- Spre exemplu se pot forma tablouri de pointeri.
- Sintaxa generală de utilizare este:

Tip *tablou[dim];

5. Legătura dintre pointeri şi tablouri

Exemple:

Declaraţia ***char *s[25];*** reprezintă un tablou de 25 pointeri la caracter.

Sortarea unor şiruri de caractere:

```
#include <iostream>
#include <cstring>
using namespace std;
void sort_lines( char *sir[ ], int n)
{
   int i, sort = 0;
   char *temp;
```

5. Legătura dintre pointeri şi tablouri

```
while (!sort)    {
sort = 1;
for(i = 0; i < n - 1; i++)
        if(strcmp(sir[i], sir[i + 1]) > 0)                {
                temp = sir[i];
                sir[i] = sir[i + 1];
                sir[i + 1] = temp;
                sort = 0;
        }
   }
}
```

5. Legătura dintre pointeri şi tablouri

```
int main(){
    int i;
    char *sir[ ] = {"manual", "carte", "mare",
       "12345", "4542" };
    for(i = 0; i < 5; i++)  cout<<sir[i]<<" ";
    cout<<"\n";
    sort_lines(sir, 5);
    for(i = 0; i < 5; i++)  cout<<sir[i]<<" ";
    cout<<"\n";
    return 0;
}
```

```cpp
#include <iostream>
#include <cstring>
using namespace std;

void sort_lines( char *sir[ ], int n)
{
    int i, sort = 0;
    char *temp;
    while (!sort)
    {
        sort = 1;
        for(i = 0; i < n - 1; i++)
            if(strcmp(sir[i], sir[i + 1]) > 0)
            {
                temp = sir[i];
                sir[i] = sir[i + 1];
                sir[i + 1] = temp;
                sort = 0;
            }
    }
}
int main()
{
    int i;
    char *sir[ ] = {"manual", "carte", "mare", "12345", "4542" };
    for(i = 0; i < 5; i++)  cout<<sir[i]<<" ";
    cout<<"\n";
    sort_lines(sir, 5);

    for(i = 0; i < 5; i++)  cout<<sir[i]<<" ";
    cout<<"\n";
    return 0;
}
```

Result

CPU Time: 0.00 sec(s), Memory: 3480 kilobyte(s)

```
manual carte mare 12345 4542
12345 4542 carte manual mare
```

5. Legătura dintre pointeri şi tablouri

Pentru a arăta că folosim ***pointeri la tablouri*** (şi nu tablouri la pointeri) avem pentru declaraţie sintaxa următoare:

Tip (*p) [dim];

Diferenţa dintre pointerii la tablouri de un tip şi tablourile de pointeri la acelaşi tip este modul de stabilire a unităţii de deplasare.

5. Legătura dintre pointeri şi tablouri

Exemplu:
În definiţia

```
int (*x)[50];      // x este un pointer la un tablou de 50 întregi
int *x[50];        // x este un tablou de 50 pointeri la întregi
```

Se observă diferenţa de semnificaţie care are loc o dată cu folosirea parantezelor.

Aplicatie

Se dau doua matrici. Sa se afiseze suma matricelor. Matricele sunt alocate in Heap.

```
#include <iostream>
using namespace std;
int m, n, i, j, (*c)[10], (*a)[10], (*b)[10];
void *Citeste_matrice (int m, int n){
        int i, j;
//(*t)[10]=new int [][];
        for (i=0; i<m;i++)
                for (j=0; j<n; j++)
                        cin>>*(*(a + i) +j);
        return a;
}
```

```
void Tipareste_matrice (int m, int n, int (*t) [10])
{
        int i, j;
        cout<<"Matricea rezultat\n";
        for (i=0; i<m; i++)
        {
                for (j=0; j<n ; j++) cout <<t[i][j]<<" ";
                cout<<endl;
        }
}
```

```
void *Suma_matrice ( int m, int n, int (*a)[10],
int (*b)[10])
{
        int i, j, (*c)[10]=new int[10][10];
        for  (i=0; i<m; i++)
                for (j=0; j<n; j++)
                        c[i][j] = a[i][j] + b[i][j];
        return c;
}
```

```
int main ( )
{
        cin>>m>>n;
        cout<<"Prima matrice \n";
        a =(int (*) [10])    Citeste_matrice(m,n);
        Tipareste_matrice(m,n,a);
        cout<<"A doua matrice\n";
        b =(int (*) [10]) Citeste_matrice(m,n);
        Tipareste_matrice(m,n,b);
        c =(int (*) [10]) Suma_matrice(m,n,a,b);
        Tipareste_matrice(m,n,c);
        return 0;
}
```

Test grila

Ce se afiseaza dupa executia urmatoarelor programe C++ ?

1. Ce se afiseaza dupa executia urmatorului program C++ ?

```
#include <iostream>
using namespace std;
void function(int x)
{       x = 30;         }
int main()
{
        int y = 20;
        function(y);
        cout<<y;
        return 0;
}
```

```
1  #include <iostream>
2  using namespace std;
3  void function(int x)
4  {    x = 30; }
5  int main()
6  {
7      int y = 20;
8      function(y);
9      cout<<y;
10     return 0;
11 }
```

a) 20
b) 30
c) Compiler error
d) Runtime error

2. Ce se afiseaza dupa executia urmatorului program C++ ?

```
#include <iostream>
using namespace std;
void function(int *ptr)
{       *ptr = 30;      }
int main()
{
        int y = 20;
        function(&y);
        cout<<y;
        return 0;
}
```

```
1  #include <iostream>
2  using namespace std;
3  void function(int *ptr)
4  {    *ptr = 30;  }
5  int main()
6  {
7      int y = 20;
8      function(&y);
9      cout<<y;
10     return 0;
11 }
```

a) 20
b) 30
c) Compiler error
d) Runtime error

3. Ce se afiseaza dupa executia urmatorului program C++ ?

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5   int *ptr;    int x;
6   ptr = &x;    *ptr = 0;
7   cout<<" x = "<<x<<'\n';
8   cout<<" *ptr = "<<*ptr<<'\n';
9   *ptr += 5;
10  cout<<" x = "<<x<<'\n';
11  cout<<" *ptr = "<<*ptr<<'\n';
12  (*ptr)++;
13  cout<<" x = "<<x<<'\n';
14  cout<<" *ptr = "<<*ptr<<'\n';
15  return 0;
16 }
```

```
#include <iostream>
using namespace std;
int main()
{
 int *ptr;    int x;
 ptr = &x;    *ptr = 0;
 cout<<" x = "<<x<<'\n';
 cout<<" *ptr = "<<*ptr<<'\n';
 *ptr += 5;
 cout<<" x = "<<x<<'\n';
 cout<<" *ptr = "<<*ptr<<'\n';
 (*ptr)++;
 cout<<" x = "<<x<<'\n';
 cout<<" *ptr = "<<*ptr<<'\n';
 return 0;
}
```

a)
x = 0
*ptr = 0
x = 5
*ptr = 5
x = 6
*ptr = 6

b)
x = garbage value
*ptr = 0
x = garbage value
*ptr = 5
x = garbage value
*ptr = 6

c)
x = 5
*ptr = 5
x = garbage value
*ptr = garbage value
x = 0
*ptr = 0

d)
x = 0
*ptr = 0
x = 0
*ptr = 0
x = 0
*ptr = 0

4. Ce se afiseaza dupa executia urmatorului program C++ ?

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5  float arr[5] = {12.5, 10.0, 13.5, 90.5, 0.5};
6  float *ptr1 = &arr[0];
7  float *ptr2 = ptr1 + 3;
8  cout<<*ptr2<<" ";
9  cout<<ptr2 - ptr1;
10 return 0;
11 }
```


```
#include <iostream>
using namespace std;
int main()
{
float arr[5] = {12.5, 10.0, 13.5, 90.5, 0.5};
float *ptr1 = &arr[0];
float *ptr2 = ptr1 + 3;
cout<<*ptr2<<" ";
cout<<ptr2 - ptr1;
return 0;
}
```

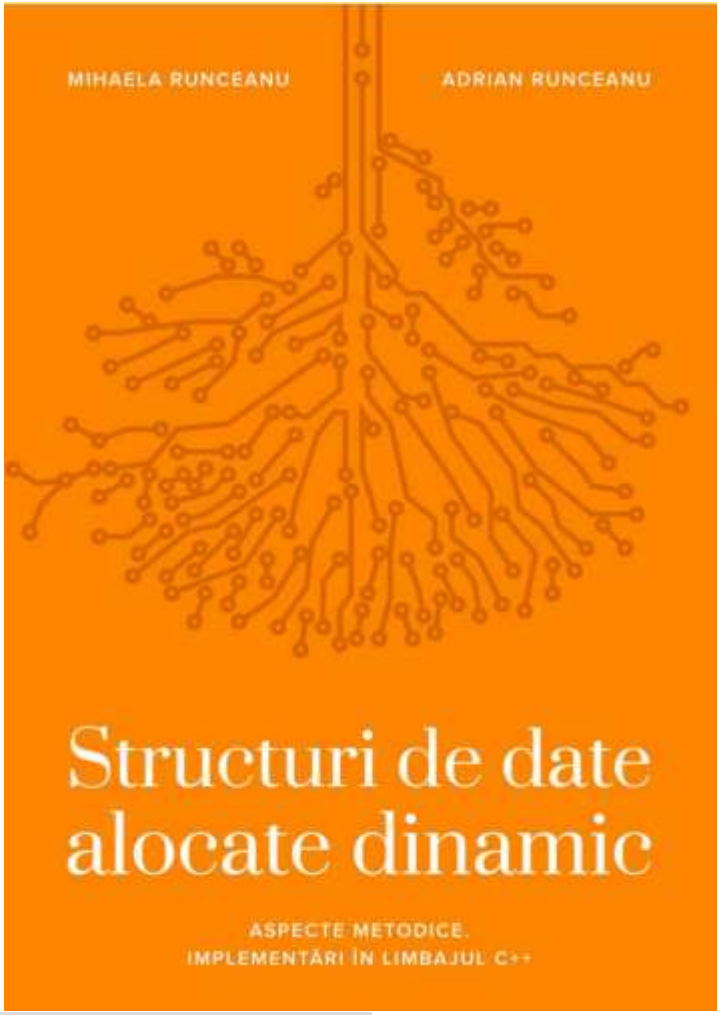

Presupunem ca tipul de date **float** este memorat(stocat) pe 4 octeti.

a) 90.5 3
b) 90.5 12
c) 10.5 12
d) 0.5 3

Bibliografie

Mihaela Runceanu, Adrian Runceanu - **STRUCTURI DE DATE ALOCATE DINAMIC. Aspecte metodice. Implementări în limbajul C++**, 2016, Editura Academica Brancusi din Targu Jiu

https://www.researchgate.net/publication/308938197_STRUCTURI_DE_DATE_ALOCATE_DINAMIC_Aspecte_metodice_Implementari_in_limbajul_C/download



Întrebări?