Fundamentele Programarii

Expresii. Propozitii. Operatori.

Expresii

- O **expresie** este o compunere de funcţii matematice scrisă în stil operatorial, operatorii fiind funcţii, iar operanzii argumentele acestor funcţii.
- O expresie este cel mai simplu bloc al unui program
- Expresia calculeaza o valoare obtinuta dintr-un numar de operanzi

Exemple:

- `34` // expresie literala
- `Nume_favorit` // nume de variabila
- `1.4+3.2` // adunarea
- `2*5` // produsul
- `a>b` // expresie relationala
- `a=b` // atribuirea

Propozitii

- O **propozitie** este o linie completa de cod care realizeaza o anumita actiune.
- De obicei, sunt terminate cu ";"
- De obicei, contin expresii
- C++ are multe tipuri de propozitii: expresii, declarari, propozitia nula, atribuiri, instructiuni, iteratii, s.a.

Exemple:

- `int x;` // declarare
- `Nume_favorit=5;` // atribuire
- `x=1.4+3.2;` // atribuirea lui x a valorii obtinute prin adunarea celor doua constante literale
- `x=2*5 ;` // atribuirea lui x a valorii obtinute prin produsul celor doua constante literale
- `if (a>b)`
  `cout<<"a este mai mare decat b";` // instructiunea if
- `;` // propozitia nula

Operatori

- C++ contine un set bogat de operatori:
  - ❑ unari (-),
  - ❑ binari (operatorul de multiplicare *),
  - ❑ ternari (operatorul conditional ?:)

- Operatorii uzuali pot fi grupati in modul urmator:

- De atribuire
- Artimetici
- De incrementare/decremenetare
- Relationali /de comparare
- Logici
- Altii

Operatorul de atribuire

TS=TD

- **TD** este o expresie care are o valoare
- Valoarea lui **TD** este retinuta in **TS**

- Valoarea lui **TD** trebuie sa fie compatibila cu **TS**
- Lui **TS** trebuie sa putem sa ii asociem o valoare (nu poate fi o constanta)

- Putem atribui mai multor variabile o anumita valoare printr-o singura expresie

- In C++ avem conceptul de

l-value si r-value

- In **l-value** (ce este la stanga operatorului =) se retine o adresa, iar in **r-value** (ce este la dreapta operatorului =) se retine valoarea de la adresa respectiva (ceea ce se afla la adresa respectiva) (Exemplu: scor=90)

Operatorul de atribuire

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

int main(){

    int num1{ 10 }; //initializare, nu atribuire
    int num2{ 20 }; //initializare, nu atribuire


    cout << "num1 este " << num1 << endl;
    cout << "num2 este " << num2 << endl;

    //atribuirea unei valori unei anumite variabile este atunci cand schimbam valoarea variabilei respective
    //ce a fost initializata la inceput

    num1 = 100;
    cout << "num1 este " << num1 << endl;
    cout << "num2 este " << num2 << endl;


    return 0;
}
```

Operatorul de atribuire

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

int main(){

    int num1{ 10 }; //initializare, nu atribuire
    int num2{ 20 }; //initializare, nu atribuire


    cout << "num1 este " << num1 << endl;
    cout << "num2 este " << num2 << endl;

    //atribuirea unei valori unei anumite variabile este atunci cand schimbam valoarea variabilei respective
    //ce a fost initializata la inceput

    num1 = num2;
    cout << "num1 este " << num1 << endl;
    cout << "num2 este " << num2 << endl;


    return 0;
}
```

Operatorul de atribuire

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

int main(){

    int num1{ 10 }; //initializare, nu atribuire
    int num2{ 20 }; //initializare, nu atribuire


    cout << "num1 este " << num1 << endl;
    cout << "num2 este " << num2 << endl;

    //atribuirea unei valori unei anumite variabile este atunci cand schimbam valoarea variabilei respective
    //ce a fost initializata la inceput

    num1 = num2 = 1000;
    cout << "num1 este " << num1 << endl;
    cout << "num2 este " << num2 << endl;


    return 0;
}
```

Stanga<-----Dreapta

num1 = num2 = 1000

num2 = 1000

num1 = 1000

Operatorul de atribuire

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

int main(){

    int num1{ 10 }; //initializare, nu atribuire
    int num2{ 20 }; //initializare, nu atribuire


    cout << "num1 este " << num1 << endl;
    cout << "num2 este " << num2 << endl;

    //atribuirea unei valori unei anumite variabile este atunci cand schimbam valoarea variabilei respective
    //ce a fost initializata la inceput

    num1 = "Simona";
    cout << "num1 este " << num1 << endl;
    cout << "num2 este " << num2 << endl;


    return 0;
}
```

Compilatorul va genera o eroare!

Operatorul de atribuire

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

int main(){

    const int num1{ 10 }; //initializare, nu atribuire
    int num2{ 20 }; //initializare, nu atribuire


    cout << "num1 este " << num1 << endl;
    cout << "num2 este " << num2 << endl;

    //atribuirea unei valori unei anumite variabile este atunci cand schimbam valoarea variabilei respective
    //ce a fost initializata la inceput

    num1 = 100;
    cout << "num1 este " << num1 << endl;
    cout << "num2 este " << num2 << endl;




    return 0;
}
```

Compilatorul va genera o eroare!

Operatorul de atribuire

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

int main(){

    int num1{ 10 }; //initializare, nu atribuire
    int num2{ 20 }; //initializare, nu atribuire


    cout << "num1 este " << num1 << endl;
    cout << "num2 este " << num2 << endl;

    //atribuirea unei valori unei anumite variabile este atunci cand schimbam valoarea variabilei respective
    //ce a fost initializata la inceput

    100 = num1;   //100 este o constanta  literala,   deci nu are o adresa
    cout << "num1 este " << num1 << endl;
    cout << "num2 este " << num2 << endl;

    return 0;
}
```

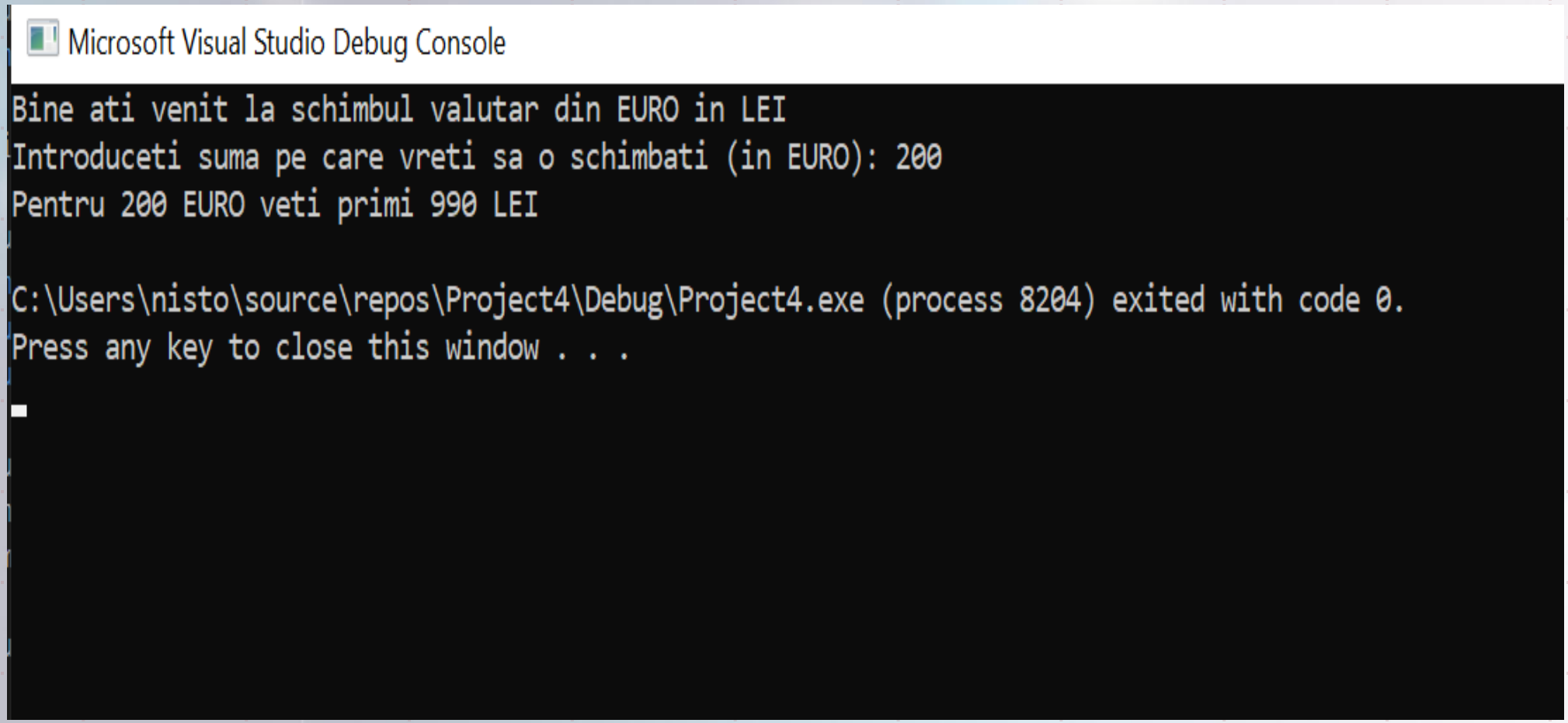
Compilatorul va genera o eroare!

Operatorii artimetici

- + adunarea
- - scaderea
- * produsul
- / (div) impartirea
- % (modulo) restul functioneaza doar pentru numere intregi

- Atentie!
- 4/5 = 0
- 4.0/5 = 0.8
- 4/5.0 = 0.8
- 4.0/5.0 = 0.8
- (float) 4/5 = 0.8   //in stilul C
- (double) 4/5 = 0.8 //in stilul C
- static_cast<double> (4)/5 = 0.8 //in stilul C++11

Operatorii aritmetici



Operatorii de incrementare (++) si decremetare (--)

- i++ incrementarea operandului cu 1 (i+1)
- i-- decrementarea operandului cu 1   (i-1)

- Preincrementare: ++numVar
- Postincrementare: numVar++

- Predecrementare: --numVar
- Postdecrementare: numVar--

- **Vezi programul “OperatoriIncrementareDecrementare”!**

Testarea pentru egalitate

`==` si `!=`

- Compara valorile a doua expresii
- Rezultatul este o variabiala de tip bool (adevarat sau fals, 1 sau 0)
- De obicei este folosit in cadrul instructiunilor **`if, while`**

- **Exemple:**

- `expr1 == expr2`
- `expr1 != expr2`

- `100 == 200` `//intotdeauna returneaza fals`
- `num1 !=num2`

Testarea pentru egalitate

== si !=

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

int main() {
    bool rezultat{ false };
    rezultat = (100 == 50 + 50);
    cout << "rezultat = " << rezultat << endl;

    int num1{ 4 };
    int num2{ 10 };
    rezultat = (num1 != num2);
    cout << "rezultat = " << rezultat << endl;

    cout << (num1 == num2) << endl;
    cout << boolalpha;
    cout << (num1 == num2) << endl;
    cout << (num1 != num2) << endl;
    return 0;
}
```

rezultat = 1

rezultat = 1

0

false

true

- Este un manipulator definit in spatiul de nume standard
- Toate variabilele de tip bool ce vor fi afisate dupa
  cout<< boolalpha;
  vor aparea in consola cu valoarea **true** sau **false** (in locul lui **1** sau **0**).
- Daca vrem sa revenim la afisarea variabilelor de tip bool cu **1** si **0** vom folosi
  cout<<noboolalpha;

Vezi programul “TestEgalitate”!

Operatorii relationali

Expr1 op Expr2

- `>` mai mare
- `>=` mai mare sau egal
- `<` mai mic
- `<=` mai mic sau egal

Operatorii relationali

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

int main() {
    double num1, num2;

    cout << boolalpha;

    cout << "Dati doua numere reale: ";
    cin >> num1 >> num2;

    cout << num1 << " <= " << num2 << " este " << (num1 <= num2) << endl;
    cout << num1 << " < " << num2 << " este " << (num1 < num2) << endl;
    cout << num1 << " >= " << num2 << " este " << (num1 >= num2) << endl;
    cout << num1 << " > " << num2 << " este " << (num1 > num2) << endl;

    return 0;
}
```

num1= 3.4 num2=5.6

3.4. <= 5.6 este true

3.4 < 5.6 este true

3.4 >= 5.6 este false

3.4>5.6 este false

Ce va afisa programul daca num1= 5 si num2=4.999999999999?

Operatorii logici

Operator	Sens
not !	negatia
and &&	si logic
or ||	sau logic

De evitat de folosit cuvintele cheie! Vom folosi operatorii corespunzatori.

- **!** este un operator unar
- **&&** si **||** sunt operatori binari
- **!** are prioritate in fata lui **&&**
- **&&** are prioritate in fata lui **||**

Expresia a	not a !a
true	false
false	true

Expresia a	Expresia b	a and b a && b
true	true	**true**
true	false	false
false	true	false
false	false	false

Expresia a	Expresia b	a or b a || b
true	true	**true**
true	false	**true**
false	true	**true**
false	false	False

Exemple

```
cout << boolalpha;

cout << endl << (!(3 < 4) || (2 >= 1) && (5 < 7))<<endl;

cout << endl << !((3<4) || (2>5) && (3<3))<<endl;
```

true

false

Operatorii logici

Short-Circuit Evaluation

- Cand se evalueaza o expresie logica, C++ se opreste o data ce gaseste rezultatul expresiei

```
expr1 &&  expr2 && expr3

expr1 || expr2 || expr3
```

- Exemplu:

```
cout << (2 < 1) && (5 < 7)<<endl;
cout << (4<5)  || (6>3) || !(4>5)<< endl;
```

0

1

Operatorii de atribuire compusi

Operator	Exemplu	Interpretare
+=	`TS += TD;`	`TS = TS + TD;`
-=	`TS -= TD;`	`TS = TS - TD;`
*=	`TS *= TD;`	`TS = TS * TD;`
/=	`TS /= TD;`	`TS = TS / TD;`
%=	`TS %= TD;`	`TS = TS % TD;`

Operatorii de atribuire compusi

Exemple:

```
a+=1;          // a=a+1;
a/=5;          // a=a/5;
a*=b+c;        // a=a*(b+c);
a+=b*c;        // a=a+(b*c);
a%=5;          // a=a%5;
```

Precedenta operatorilor (nu este lista completa)

Operator	Asociativitatea
`[ ]  ( )`	De la stanga la dreapta
`++  --  !  - (unar)  sizeof`	**De la dreapta la stanga**
`*  /  %`	De la stanga la dreapta
`+  -`	De la stanga la dreapta
`<  <=  >  >=`	De la stanga la dreapta
`==  !=`	De la stanga la dreapta
`&&`	De la stanga la dreapta
`||`	De la stanga la dreapta
`=  ?:`	**De la dreapta la stanga**

https://en.cppreference.com/w/cpp/language/operator_precedence

Precedenta si asociativitatea operatorilor

- Presupunem ca avem o expresie cu doi operatori distincti cu nivel de precedenta diferit

`expr1 op1 expr2 op2 expr3` // precedenta

- Presupunem ca avem o expresie cu doi operatori, fie aceiasi, fie diferiti dar cu acelasi nivel de precedenta

`expr1 op1 expr2 op1 expr3` // asociativitatea

- Pentru a evita orice dubiu putem folosi intotdeauna parantezele!

Precedenta operatorilor

Exemplu:

```
Rezultat =  num1+num2*num3;
Rezultat =  (num1+(num2*num3));
```

```
Rezultat1 =  num1+num2-num3;
Rezultat1 = (( num1+num2)-num3);
```

```
int x = 2;
int y = 5;
int z;
z = x++ + --y;
cout << "x=" << x << endl;
cout << "y=" << y << endl;
cout << "z=" << z << endl;
```

x=3
y=4
z=6

Precedenta operatorilor

Exemplu:

```
int i = 2;
cout << i++ + --i << endl;
cout << i << endl;
```

2
2

```
int x = 2;
cout << -x++ % 2 - --x * 2 << endl;
cout << x << endl;
```

-3
2

Nu este ok! Comportament nedefinit!

```
int y = 3;
cout << (++y / 2 == y-- - (1 < 3) * 2 + y - 3)<<endl<<endl;
cout << "y= " << y << endl;
cout << "Membrul din stanga (evaluat prima data): "<< ++y / 2 << endl;
cout << "Membrul din dreapta: "<<y-- - (1 < 3) * 2 + y - 3<<endl<<endl;
cout << "y= "<<y << endl;
cout <<"Membrul din dreapta (evaluat prima data): "<< y-- - (1 < 3) * 2 + y-3<<endl;
cout << "Membrul din stanga: " << ++y / 2;
```

2
3
1
1

https://test-master.space