

Tutoriat 1

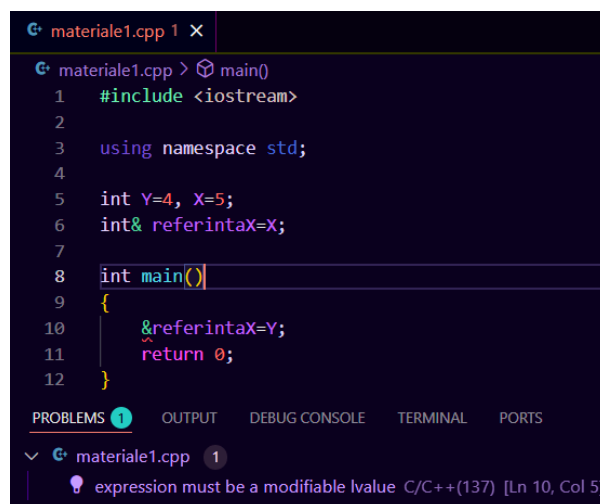
Referințe (caracterul &)

Definiție = reprezintă aliasurile unor variabile deja existente;

Sintaxa: tip_date &nume_referință = variabilă;

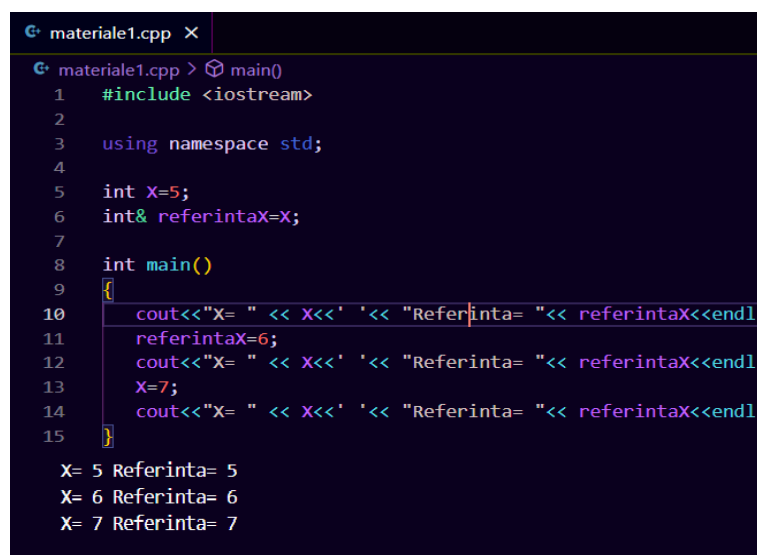
Mențiuni:

-odată asignată o valoare referinței, nu se mai poate modifica (asta se face la declarare);



```
materiale1.cpp 1 X
materiale1.cpp > main()
1  #include <iostream>
2
3  using namespace std;
4
5  int Y=4, X=5;
6  int& referintaX=X;
7
8  int main()
9  {
10     &referintaX=Y;
11     return 0;
12 }
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS
materiale1.cpp 1
expression must be a modifiable lvalue C/C++(137) [Ln 10, Col 5]
```

-dacă modifici valoarea prin intermediul referinței, se modifică automat și variabila (și invers);

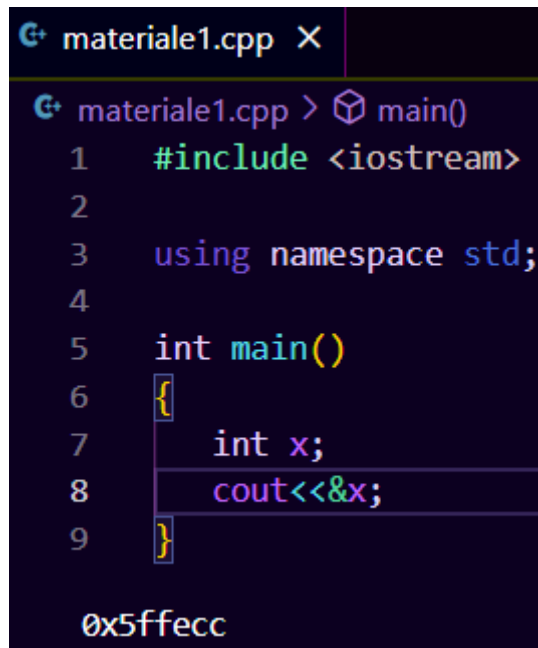


```
materiale1.cpp X
materiale1.cpp > main()
1  #include <iostream>
2
3  using namespace std;
4
5  int X=5;
6  int& referintaX=X;
7
8  int main()
9  {
10     cout<<"X= " << X<<' '<< "Referinta= "<< referintaX<<endl;
11     referintaX=6;
12     cout<<"X= " << X<<' '<< "Referinta= "<< referintaX<<endl;
13     X=7;
14     cout<<"X= " << X<<' '<< "Referinta= "<< referintaX<<endl;
15 }
X= 5 Referinta= 5
X= 6 Referinta= 6
X= 7 Referinta= 7
```

-se poate pune referința și la ceea ce returnează o funcție, dar și la parametrii acesteia, pentru a nu mai copia valoarea variabilelor;

```
int& /*nu copiem ceea ce returnam*/ functie(int& val /*nu copiem parametrul*/){  
    int& referinta; // o referinta declarata in functie  
}
```

-& arată zona din memorie a unei variabile;



```
materiale1.cpp X  
materiale1.cpp > main()  
1  #include <iostream>  
2  
3  using namespace std;  
4  
5  int main()  
6  {  
7      int x;  
8      cout<<&x;  
9  }  
  
0x5ffecc
```

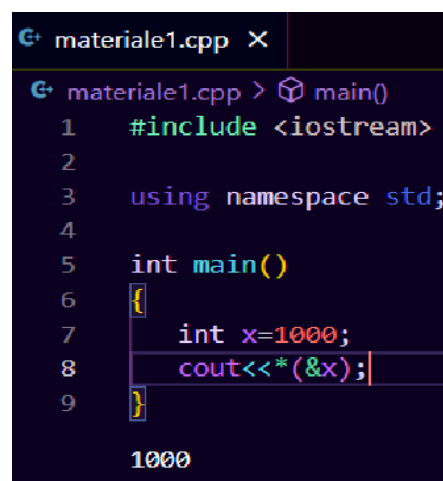
Pointeri (*)

Definiție = conțin adresa unei zone de memorie.

Sintaxa: tip_date* nume_pointer = zona_memorie;

Mențiuni:

-arată valoarea de la o zonă din memorie;



```
materiale1.cpp X  
materiale1.cpp > main()  
1  #include <iostream>  
2  
3  using namespace std;  
4  
5  int main()  
6  {  
7      int x=1000;  
8      cout<<*&x;  
9  }  
  
1000
```

- se declară pointeri cu ajutorul *;
- tipul pointerului trebuie să fie același cu tipul datei spre care pointează;
- poate fi NULL (sau nullptr, același lucru, dar mai nou);
- se folosește new și delete pentru alocarea/dealocarea memoriei (pentru array-uri se folosește new tip[lungime] și delete[]). Se numește alocare dinamica;

```
material1.cpp X
material1.cpp > main()
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      int* x=new int[10]{1,1,1};
8      cout<<x[0]<<' '<<x[1]<<' '<<x[2]<<' '<<x[3]; //1 1 1 0
9      delete[] x;
10 }
```

Definiție = valoarea unei variabile nu poate fi modificată după inițializare;

Sintaxa:

1. tip_data const nume_constanta = valoarea;
2. const tip_data nume_constanta = valoarea; // nu merge mereu
3. #define nume_constanta valoarea;

Mențiuni:

- se initializează la declarare;
- de obicei se scrie la dreapta tipului, dar merge câteodată și înainte;

```
material1.cpp ×
material1.cpp > main()
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      const int x=5;
8      int const y=5;
9  }
```

-se poate aplica **const** variabilelor, pointerilor, parametrilor sau funcțiilor;

```
//nu am nume la parametrul functiei
//sa stiti ca se poate
//altii(eu) n-au stiut
class A{
    //constul la functie se pune pentru
    //a "asigura" faptul ca nu modificam
    //parametrii unei clase
    int const functie(int const * const) const {}
    //primul const din paranteze arata ca valoarea
    //nu poate fi schimbata
    //cel de al doilea const arata ca spatiul din
    //memorie nu poate fi schimbat
};
```

-“const int const * const” e același lucru cu “int const * const”, deoarece primul și al doilea const se aplică lui int;