Exercitii Tutoriat 1

**à pointeri, referinte, alocare dinamica**

**à introducere in POO – clase & obiecte , modificatori de acces si compunere**

**Cerinta: Pentru fiecare dintre programele de mai jos, supneți dacă sunt corecte. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți ce nu este corect , de ce și ce ați corecta.**

**Exercitiul 1**

#include<iostream>

using **namespace std;**

int main()

{

    int a=3,b=2;

    int \*ptr\_a=nullptr, \*ptr\_b;

    int& ref\_a=a;

    ptr\_a=&ref\_a;

    ref\_a=b;

    ptr\_b=&b;

    b=5;

    ptr\_b=&ref\_a;

    ref\_a=10;

    b=2;

    cout<<"ref\_a: "<<ref\_a<<endl;

    cout<<"a: "<<a<<endl;

    cout<<"b: "<<b<<endl;

    cout<<"\*ptr\_a: "<<\*ptr\_a<<endl;

    cout<<"\*ptr\_b: "<<\*ptr\_b<<endl;

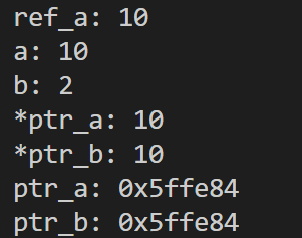
    cout<<"ptr\_a: "<<ptr\_a<<endl;

    cout<<"ptr\_b: "<<ptr\_b<<endl;

    return 0;

}

**Compileaza: output:**



**Exercitiul 2**

#include<iostream>

using namespace std;

int main()

{

    int a=3,b=5,c=8;

    int &ref\_a=a, &ref\_b, &ref\_c=c;

    ref\_a=ref\_b;

    ref\_c=ref\_a;

    cout<<ref\_b<<' '<<ref\_c<<' '<<ref\_a;

    return 0;

}

**Eroare la linia 8 : “reference variable "ref\_b" requires an initializer”**

**Explicatie: Referintele trebuie mereu initializate in momentul declararii. Aici ref\_b nu e initializata**

**Modificare: Initializam referinta: &ref\_b=b;**

**Exercitul 3**

#include<iostream>

using namespace std;

int f(int& x)

{

    x++;

    x\*=2;

    return x;

}

int\* g(int x)

{

    x+=7;

    x/=2;

    return x;

}

int main()

{

    int x=5;

    f(x);

    cout<<x<<endl;

    g(x);

    cout<<x<<endl;

    return 0;

}

**Eroare la linia 16 : la return x; in functia g**

**Explicatie:**

**-x este de tip intreg (int), iar subprogramul returneaza un int\* adica o adresa de memorie à incompatibilitatea tipului de date returnat.**

**-Nu putem face conversie de la int la int\* si invers.**

**Modificare: declaram antetul lui g : int g(int x); à adica il obligam sa returneze un int, nu un int\***

**Exercitiul 4**

#include<iostream>

using namespace std;

int f(int& x)

{

    x++;

    x\*=2;

    return x;

}

int g(int x)

{

    x+=7;

    x/=2;

    return x;

}

int main()

{

    int x=5;

    f(x);//x = 12

    cout<<x<<endl;

    g(x);

    cout<<x<<endl;

    return 0;

}

**Compileaza :**

**12**

**12**

**Explicatie:**

**-se apeleaza f ce are ca parametru un int&**

**-orice modificari vom face asupra parametrului x din f , vor fi vizibile si in main è x=5, il transmitem prin referinta in functia f . Dupa calcule … x=12 in f, dar si in main**

**-apoi apelam functia g ce are ca parametru un int**

**-avand in vedere ca nu e transmis prin referinta, x-ul din main nu va suferi modificari**

**-x-ul LOCAL din g va avea valorea 9, dar dupa ce incheiem subprogramul, acel x LOCAL inceteaza sa existe. Astfel, in urma apelului functiei g, x isi va pastra vechea valoare.**

**-x LOCAL != x din MAIN in acest caz**

**Exercitiul 5**

#include<iostream>

using namespace std;

int\* g(int\* x)

{

    \*x=8;

    return x;

}

int main()

{

    int x=5,\*x\_x=&x;

    g(x\_x);

    cout<<x<<' '<<\*x\_x<<endl;

    return 0;

}

**Compileaza: 8 8**

**Explicatie:**

**-avem un int x=5 si un pointer x\_x catre acesta**

**-apelam functia g care cere un parametru de tip int\* care se potriveste cu tipul lui x\_x (int\*)**

**-in functia g punem LA ADRESA de memorie a lui x (care e de fapt x\_x din main) valoarea 8**

**-la final, returnam un pointer**

**-tot ce am modificat asupra lui x din g se va resfrange asupra lui x\_x si in main**

**-deci acum la adresa de memorie indicata de x\_x avem 8, implicit valaorea lui x din main este tot 8**

**Exercitiul 6**

#include<iostream>

using namespace std;

int\* g(int\* x)

{

    x=56;

    return x;

}

int main()

{

    int x=5,\*x\_x=&x;

    g(x\_x);

    cout<<x<<' '<<\*x\_x<<endl;

    return 0;

}

**Eroare la linia 7 : a value of type "int" cannot be assigned to an entity of type "int \*"**

**Explicatie:**

**-problema apare atunci cand se incerca executia liniei x=56;**

**-x este de tip int\* à adica x simplu, fara \*, se refera la o adresa de memorie (gen : 0x8f7a2e)**

**-practic, incercam sa ii dam lui x o adresa de memorie la care nu are acces**

**Modificare: in loc de x=56; scriem \*x=56; . Asemanator exercitiului anterior, se va afisa 56 56**

**Exercitiul 7**

#include<iostream>

using namespace std;

int& f(int x)

{

    x++;

    x\*=2;

    return x;

}

int main()

{

    int x=5;

    int& x\_x = f(x);

    cout<<x\_x<<endl;

    return 0;

}

**Eroare la linia 16 : cout<<x\_x<<endl;**

**Explicatie: f cere ca tip returnat un int&, adica o referinta la int. Atunci cand subprogramul se incheie, variabila locala x nu mai exista. In consecinta, referinta care ar trebui returnata este nula.**

**Modificare: antentul functiei f va fi int& f(int&x)**

**De ce? daca x va fi transmis prin referinta in f atunci f va primi x-ul “original” (care exista pana cand se incheie main-ul) , nu o copie a sa. Deci, referinta returnata nu va fi nula.**

**Exercitiul 8**

#include<iostream>

using namespace std;

class Test

{

    int value;

public:

    Test (int v = 0) {value = v;}

    int getValue() { return value; }

};

int main()

{

    Test t;

    cout << t.getValue();

    return 0;

}

**Compileaza: 0**

**Exmplicatie:**

**-pentru t se va apela constructorul clasei Test**

**-t.value va avea valoarea 0**

**-putem accesa metoda getValue(), deoarece este publica**

**-mai multe despre constructori in tutoriatul 2**

**Exercitiul 9**

#include <iostream>

using namespace std;

class Test

{

public:

    int x=2;

    void f(Test c) {cout << c.x;}

};

int main()

{

    Test d;

    f(d);

    return 0;

}

**Eroare : linia 15 la f(d)**

**Explicatie:**

**-Metoda f nu este statica, deci pentru a fi apelata, trebuie apelata de o instanta a clasei**

**-Mai multe despre static in tutoriatele urmatoare**

**Modificare: d.f(d);**

**Exercitiul 10**

#include <iostream>

using namespace std;

class Fraction

{

private:

    int den;

    int num;

public:

    Fraction() { num = 1; den = 1; }

    int& Den() { return den; }

    int& Num() { return num; }

    void print() { cout << num << "/" << den; }

};

int main()

{

    Fraction f1;

    f1.Num() = 7;

    f1.Den() = 9;

    f1.print();

    return 0;

}

**Compileaza: 7/9**

**Interbare: Daca metodele membre Num() si Den() ar fi avut antetul de mai jos, programul ar fi compilat? Daca da, de ce? Daca nu, de ce?**

    int Den() { return den; }

    int Num() { return num; }

**Exercitiul 11**

#include <iostream>

using namespace std;

class cls

{

public:

    cls(int i=0, int j=0) { x=i; y=j; }

    int x;

    int y;

};

int main()

{

    cls a, b, c[3]={cls(1,1), cls(2,2), a};

    cout << c[2].x;

    return 0;

}

**Compileaza: 0**

**Explicatie: Asemanator cu exercitiul 12**