Tutoriat 1
Supraîncarcarea operatorilor

1. Operatori unari: -, ++, --, !

```
class A {
private:
    int x;
    bool y;
public:
    A(int x = 2, bool y = true): x(x), y(y){}

    int getX() const {
        return x;
    }

    void setX(int x) {
        A::x = x;
    }

    bool isY() const {
        return y;
    }

    void setY(bool y) {
        A::y = y;
    }

    A operator - (){
        this -> x = - x;
        return A(this -> x);
    }

    A operator ! (){
        this -> y =  !y;
        return A(this -> y);
    }

    A operator ++ (){
        // prefix
        ++ this -> x;
        return A(this -> x, this -> y);
    }

    A operator ++ (int){
        // sufix
```

```
        A copy(this -> x, this -> y);
        this -> x ++;
        return copy;
    }

};
int main()
{
    A a(3, false);
    cout<<a.getX()<<" "<<a.isY()<<endl; // 3 0
    -a;
    cout<< a.getX()<<" "<<a.isY()<<endl; // -3 0
    !a;
    cout<< a.getX()<<" "<<a.isY()<<endl; // -3 1
    ++a;
    cout<< a.getX()<<" "<<a.isY()<<endl; // -2 1
    a++;
    cout<< a.getX()<<" "<<a.isY()<<endl; // -1 1
    return 0;
}
```

2. Operatori binari: +, -, /, *

```
class A {
private:
    int x;
public:
    A(int x = 2): x(x){}

    int getX() const {
        return x;
    }

    void setX(int x) {
        A::x = x;
    }

    // two objects of type A: this and ob
    A operator + (const A& ob) {
        A a;
        a.x = this -> x + ob.x;
        return a;
    }

};
int main()
{
    A a(3), b(5);
    cout<<a.getX()<<endl; // 3
```

```
    cout<<b.getX()<<endl; // 5
    A sum = a + b;
   cout<< sum.getX(); // 8
    return 0;
}
```

3. Operatori relationali: <, > , <=, >=, ==

```
class A {
private:
    int x;
public:
    A(int x = 2): x(x){}

    int getX() const {
        return x;
    }

    void setX(int x) {
        A::x = x;
    }

    // overloaded < operator
    bool operator <(const A& ob) {
        if(this -> x < ob.x){
            return true;
        }

        return false;
    }

};
int main()
{
    A a(3), b(5);
    cout<<a.getX()<<endl; // 3
    cout<<b.getX()<<endl; // 5

   cout<<  (a < b)<<endl; // 1
    //cout<<  (a > b); // eroare (trebuie supraincarcat si > )
    cout<<  (b < a); // 0
    return 0;
}
```

4. Operatorul de asignare =

```
class A {
private:
```

```
    int x;
public:
    A(int x = 2): x(x){}

    int getX() const {
        return x;
    }

    void setX(int x) {
        A::x = x;
    }

    A& operator = (const A& a){
        this -> x = a.x;
        return *this;
    }

};
int main()
{
    A a(3), b(5);
    cout<<a.getX()<<endl; // 3
    cout<<b.getX()<<endl; // 5

   a = b;
    cout<<a.getX()<<endl; // 5
    cout<<b.getX()<<endl; // 5
    return 0;
}
```

5. Input/Output (vezi Tutoriat 1 -> Noțiuni introductive -> funcții friend)

6. Functia call ()

```
class A {
private:
    int x;
public:
    A(int x = 2): x(x){}

    int getX() const {
        return x;
    }

    void setX(int x) {
        A::x = x;
    }
```

```
    // overload function call ()
    A operator()(int e, int f) {
        cout<< " Call () operator overloading"<<endl;
        A a; // x = 2
        cout<<a.x << " "<< e<< " "<< f<<endl;
        a.x =  a.x + e + f; //operatie random
        return a;
    }

};
int main()
{
    A a(3);
    cout<<a.getX()<<endl;

    A c = a(6,7);
    cout<<c.getX()<<endl; // Call () operator overloading
                          // 2  6 7
                          // 2 + 6 + 7 = 15
    return 0;
}
```

7. Operatorul []

```
const int n = 5;

class A {
private:
    int arr[n];

public:
    A() {
        for(int i = 0; i < n; i++) {
            arr[i] = i;
        }
    }

    int &operator[](int i) {
        if( i > n ) {
            cout << "Index out of bounds" <<endl;
            // return first element.
            return arr[0];
        }

        return arr[i];
    }
};

int main() {
```

```
    A a;

    cout << "Value of A[1] : " << a[1] <<endl;  // 1
    cout << "Value of A[3] : " << a[3]<<endl;  // 3
    cout << "Value of A[6] : " << a[6]<<endl;  // Index out of bounds

    return 0;
}
```

8. Opeartorul cast

```
class A
{
    int x;
public:

    A(int val=0): x(val){}

    // Note that conversion-type-id "int" is the implied return type.
    // Returns by value so "const" is a better fit in this case.
    operator int() const
    {
        return this -> x;
    }
};

int main()
{
    A a(10);
    cout << int(a); // 10
    return 0;
}
```