Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică şi Microelectronică

RAPORT

Lucrarea de laborator nr.1
la disciplina POO

Tema: Supraîncărcarea operatorilor

Varianta 10

A efectuat: Plămădeală Vladislav

A verificat: Lisnic Inga

Chişinău 2020

Scopul lucrării:

- Studierea necesităţii supraîncărcării operatorilor;
- Studierea sintaxei de definire a operatorilor;
- Studierea tipurilor de operatori;
- Studierea formelor de supraîncărcare;

Varianta 10
Sarcina:

a)Să se creeze clasa *Bool* – variabile logice.
Să se definească operatorii "+" – SAU logic, "*" – ŞI logic, "^" – SAU EXCLUSIV, ca funcţii prietene, iar operatorii "==" şi "!=" – ca metode ale clasei.
Operatorii trebuie să permită realizarea operaţiilor atît cu variabilele clasei date, cît şi cu variabilele de tip predefinit *int*. (Dacă numărul întreg este diferit de zero, se consideră că variabila este adevăr, altfel – fals.)

Codul sursă:

```
//
class Bool{
private:
	bool state;
public:
	Bool(){
		state = false;
	}
	Bool(bool s){
		state = s;
	}
	bool getState(){
		return state;
	}
	friend bool operator+(Bool &a, Bool &b){
		if(a.getState() || b.getState()){
			return true;
		}
		else return false;
	}
	friend bool operator+(Bool &a, int n){
		bool s;
```

```
            if(n != 0) s = true;
            else s = false;
            if (a.getState() || s) return true;
            else return false;
        }
        friend bool operator*(Bool &a, Bool &b){
            if(a.getState() && b.getState()){
                return true;
            } else return false;
        }
        friend bool operator*(Bool &a, int n){
            bool s;
            if(n != 0) s = true;
            else s = false;
            if (a.getState() && s) return true;
            else return false;
        }
        friend bool operator^(Bool &a, Bool &b) {
            if ((a.getState() && b.getState()) || (!a.getState() && !b.getState())) {
                return false;
            } else return true;
        }
        friend bool operator^(Bool &a, int n){
            bool s;
            if(n != 0) s = true;
            else s = false;
            if((a.getState() && s) || (!a.getState() && !s)) return false;
            else return true;
        }
        bool operator==(Bool &b){
            return this->getState() == b.getState();
        }
        bool operator==(int n){
            bool s;
            if(n != 0) s = true;
            else s = false;
            return this->getState() == s;
        }
        bool operator!=(Bool &b){
            return !this->getState() == b.getState();
        }
        bool operator!=(int n){
            bool s;
            if(n != 0) s = true;
            else s = false;
            return !this->getState() == s;
        }
};
```

Funcţia main:

```
//
int main() {
        Bool v1(true);
        Bool v2(true);
        Bool v3(false);
        bool state;
        cout << "v1 = true, v2 = true, v3 = false"<<endl;
        state = v1 + v2;
        cout <<"\nv1 + v2 = "<< state;
        state = v1 + v3;
        cout << "\nv1 + v3 = "<<state;
        state = v1 * v2;
        cout << "\nv1 * v2 = "<<state;
        state = v1 * v3;
        cout << "\nv1 * v3 = "<<state;
        state = v1 ^ v2;
        cout << "\nv1 ^ v2 = "<<state;
        state = v1 ^ v3;
        cout << "\nv1 ^ v3 = "<<state;
        state = v1 == v2;
        cout << "\nv1 == v2 = "<<state;
        state = v1 == v3;
        cout << "\nv1 == v3 = "<<state;
        state = v1 != v2;
        cout << "\nv1 != v2 = "<<state;
        state = v1 != v3;
        cout << "\nv1 != v3 = "<<state;
        return 0;
}
```

La efectuarea programului:

```
v1 = true, v2 = true, v3 = false

v1 + v2 = 1
v1 + v3 = 1
v1 * v2 = 1
v1 * v3 = 0
v1 ^ v2 = 0
v1 ^ v3 = 1
v1 == v2 = 1
v1 == v3 = 0
v1 != v2 = 0
v1 != v3 = 1
Process finished with exit code 0
```

b) Să se creeze o clasă *Set* – mulţimea numerelor întregi, utilizînd memoria dinamică.
Să se definească operatorii de lucru cu mulţimile: "+" – uniunea, "*" – intersecţia, "-" scăderea, ca metode ale clasei, iar "+=" – înserarea unui nou element în mulţime, "==" – comparare la egalitate, ş. a. ca funcţii prietene.
Să se definească operatorii "<<" şi ">>".
Să se definească funcţia de verificare a apartenenţei unui element la o mulţime

---

Codul sursă:

//

```
class Set{
private:
	vector<int> v;
	void remove_dup(std::vector<int> &v){
		auto end = v.end();
		for (auto it = v.begin(); it != end; ++it) {
			end = std::remove(it+1, end, *it);
		}
		v.erase(end, v.end());
	}
public:
	Set(int n){
		v.reserve(n);
	}
	Set(){
		v.reserve(10);
	}
	Set(vector<int> n){
		v = n;
		sort(v.begin(), v.end());
	}
	Set operator+(Set&b){
		vector<int> sum(this->v);
		sum.insert(sum.end(), b.v.begin(), b.v.end());
		sort(sum.begin(), sum.end());
		remove_dup(sum);
		return Set(sum);
	}
	Set operator*(Set &b){
		vector<int> inter;
		set_intersection(this->v.begin(), this->v.end(),
					b.v.begin(), b.v.end(),
					back_inserter(inter));
		return Set(inter);
	}
	Set operator-(Set &b){
		vector<int> diff(b.v.size());
```

```
        vector<int>::iterator it;
        it = set_difference(this->v.begin(), this->v.end(),
                            b.v.begin(), b.v.end(),
                            diff.begin());
        remove_dup(diff);
        return Set(diff);
    }
    friend void operator+=(Set&a,  int b){
        a.v.push_back(b);
        sort(a.v.begin(), a.v.end());
    }
    friend bool operator==(Set&a, Set&b){
        return equal(a.v.begin(), a.v.end(), b.v.begin(), b.v.end());
    }
    friend ostream  & operator <<(ostream &out, const Set&a){
        out <<"[";
        for (auto i = 0; i < a.v.size(); ++i) {
            out << a.v[i];
            if(i != a.v.size() - 1) out <<", ";
        }
        out << "]\n\n";
        return out;
    }
    friend istream & operator >>(istream &in, Set&a){
        int n;
        cout << "Introduceti numarul pentru adaugare la set: ";
        in >> n;
        a.v.push_back(n);
        sort(a.v.begin(), a.v.end());
        return in;
    }
    bool contains(int n){
        return find(this->v.begin(), this->v.end(), n) != v.end();
    }

};
```

Funcţia main:

```
int main(){
    vector<int> v1{1, -5, 3, 45, 30, 989, -205, 0};
    vector<int> v2{-5, 3, 50, 989, -604, 78, 20};
    Set a(v1);
    Set b(v2);
    cout<<"v1: { ";
    for(int a: v1)
        cout <<a<<" ";
    cout<<"}\n";
    cout<<"v2: { ";
    for(int a: v2)
        cout <<a<<" ";
    cout<<"}\n";
    cout <<"\nSetul A, bazat pe v1:\n";
    cout << a;
    cout <<"Setul B, bazat pe v2:\n";
    cout << b;
    cout << "A in reuniune cu B:\n" << a+b;
    cout << "A in intersectie cu B:\n" << a*b;
    cout << "A \\ B:\n"<< a-b;
```

```
        cout << "A += 33:\n" << a;
        bool s = a == b;
        cout << "A == B: "<< s;
        cout << "\n\nB <<\n";
        cin >> b;
        cout << "Setul B cu numarul adaugat:\n";
        cout << b;
        s = a.contains(77);
        cout << "Control daca numarul 77 se afla in A: " << s;
        s = b.contains(50);
        cout << "\n\nControl daca numarul 50 se afla in B: " << s;

        return 0;
}
```

La efectuarea programului:

```
v1: { 1 -5 3 45 30 989 -205 0 }
v2: { -5 3 50 989 -604 78 20 }

Setul A, bazat pe v1:
[-205, -5, 0, 1, 3, 30, 45, 989]

Setul B, bazat pe v2:
[-604, -5, 3, 20, 50, 78, 989]

A in reuniune cu B:
[-604, -205, -5, 0, 1, 3, 20, 30, 45, 50, 78, 989]

A in intersectie cu B:
[-5, 3, 989]

A \ B:
[-205, 0, 1, 30, 45]

A += 33:
[-205, -5, 0, 1, 3, 30, 45, 989]

A == B: 0

B <<
Introduceti numarul pentru adaugare la set: 33
Setul B cu numarul adaugat:
[-604, -5, 3, 20, 33, 50, 78, 989]

Control daca numarul 77 se afla in A: 0

Control daca numarul 50 se afla in B: 1
```