Универзитет „Св. Кирил и Методиј“ - Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Објектно ориентирано програмирање

Аудиториски вежби 6

Содржина

1. Преоптоварување на оператори

1.1. Пример 1

Да се напише класа `Array` за работа со еднодимензионални полиња од целобројни елементи. За полето се чуваат информации за неговиот вкупен капцитет, тековниот број на елементи. Резервацијата на меморијата да се врши динамички.

Во класата да се преоптоварат следните оператори:

- `+=` за додавање нови броеви во полето и притоа ако е исполнет капацитетот на полето (низата) да се зголеми за 100%.
- `-=` која ги брише сите појавувања на целобројниот аргумент во полето, а притоа капацитетот да не се промени.
- `<<` за печатење на елементите од полето.

Да се тестира класата во main функција.

Решение `oop_av61a.cpp`

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

class Array {
private:
    int *x;
    int size;
    int capacity;
public:
    Array(const int capacity = 5) {
        x = new int[capacity];
        size = 0;
        this->capacity = capacity;
    }

    // copy constructor
    Array(const Array &a) {
        size = a.size;
        capacity = a.capacity;
        x = new int[capacity];
        for (int i = 0; i < size; ++i) {
            x[i] = a.x[i];
        }
    }

    // asignment operator =
    Array& operator=(const Array &a) {
        if (this == &a) return *this;
        size = a.size;
        capacity = a.capacity;
        delete [] x;
        x = new int[capacity];
        for (int i = 0; i < size; ++i) {
```

```
            x[i] = a.x[i];
        }
        return *this;
    }

    // destructor
    ~Array() {
        delete [] x;
    }
    int getSize() {
        return size;
    }

    int getCapacity() {
        return capacity;
    }

    const int *getX() {
        return x;
    }
    //na nizata x od dadeniot objekt dodadi go elementot n
    Array & operator+= (int n) {
        if (capacity == size) {
            int *y = new int[2 * capacity];
            for (int i = 0; i < size; ++i) {
                y[i] = x[i];
            }
            delete [] x;
            x = y;
            capacity = capacity * 2;
        }
        x[size] = n;
        size++;
        return *this;
    }

    //od nizata x od dadeniot objekt izbrishi go elementot n
    Array & operator-= (int n) {
        int newSize = 0;
        for (int i = 0, j = 0; i < size; ++i)
            if (x[i] != n) {
                x[j++] = x[i];
                newSize++;
            }
        size = newSize;
        return *this;
    }
//friend ostream & operator<<(ostream &o, Array &a);


};

ostream& operator<<(ostream &o, Array &a) {
    for (int i = 0; i < a.getSize(); ++i) {
        o << a.getX()[i] << " ";
    }
    for (int i = a.getSize(); i < a.getCapacity(); ++i) {
        o << "- ";
    }
    o << endl;
    return o;
}
int main() {

    Array a;
    a += (6);
    a += (4);
    a += (3);
    a += (2);
    a += (1);

    Array b(a);

    b -= (2);
    b -= (3);
```

```
    cout << " a: " << a;
    cout << " b: " << b;

    return 0;
}
```

1.2. Пример 2

Да се дополни првиот пример така што ќе се преоптоварат и операторите:

- `[]` за пристап до елемент и промена на вредноста на елемент од полето.
- `==` за споредба на два објекти од класата `Array`.

Да се тестира класата во main функција.

Решение `oop_av61b.cpp`

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

class Array {
private:
    int *x;
    int size;
    int capacity;
public:
    Array(const int capacity = 5) {
        x = new int[capacity];
        size = 0;
        this->capacity = capacity;
    }

    // copy constructor
    Array(const Array &a) {
        size = a.size;
        capacity = a.capacity;
        x = new int[capacity];
        for (int i = 0; i < size; ++i) {
            x[i] = a.x[i];
        }
    }

    // asignment operator =
    Array& operator=(const Array &a) {
        if (this == &a) return *this;
        size = a.size;
        capacity = a.capacity;
        delete [] x;
        x = new int[capacity];
        for (int i = 0; i < size; ++i) {
            x[i] = a.x[i];
        }
        return *this;
    }

    // destructor
    ~Array() {
        delete [] x;
    }
    int getSize() {
        return size;
    }

    int getCapacity() {
        return capacity;
```

```
    }

    const int *getX() {
        return x;
    }
    //na nizata x od dadeniot objekt dodadi go elementot n
    Array & operator+= (int n) {
        if (capacity == size) {
            int *y = new int[2 * capacity];
            for (int i = 0; i < size; ++i) {
                y[i] = x[i];
            }
            delete [] x;
            x = y;
            capacity = capacity * 2;
        }
        x[size] = n;
        size++;
        return *this;
    }

    //od nizata x od dadeniot objekt izbrishi go elementot n
    Array & operator-= (int n) {
        int newSize = 0;
        for (int i = 0, j = 0; i < size; ++i)
            if (x[i] != n) {
                x[j++] = x[i];
                newSize++;
            }
        size = newSize;
        return *this;
    }

    int& operator[](int index) {
        int pom = -1;
        if (index >= 0 && index < size )
            return x [index];
        else {
            cout << " Nadvor od opseg " << endl ;
            return pom;
        }
    }

    bool operator==(Array &a) {
        if (this->size != a.size ) return false;
        for (int i = 0; i < size; i++)
            if (x[i] != a.x[i]) return false;

        return true;
    }

//friend ostream & operator<<(ostream &o, Array &a);


};

ostream& operator<<(ostream &o, Array &a) {
    for (int i = 0; i < a.getSize(); ++i) {
        o << a[i] << " "; //povik na operatorot []
    }
    for (int i = a.getSize(); i < a.getCapacity(); ++i) {
        o << "- ";
    }
    o << endl;
    return o;
}
int main() {

    Array a;
    a += (6);
    a += (4);
    a += (3);
    a += (2);
    a += (1);
```

```
    Array b(a);
    b -= (2);
    b -= (3);

    a[0] = 9; //primena na operatorot []

    cout << " a: " << a;
    cout << " b: " << b;

    if (a == b) cout << "Isti se";
    else cout << "Ne se isti";


    return 0;
}
```

2. Задачи

2.1. Комплексен број

Да се дефинира класа за работа со комплексни броеви. За секој комплексен број се чуваат податоци за реалниот и имагинарниот дел.

Да се преоптоварат операторите +, -, *, /, +=, -=, *=, /= за извршуваање на содветните операции со комплексни броеви.

Да се имплементира операторот << за печатење на комплексен број.

Да се имплементира операторот + за собирање на комплексен и децимален број (како и децимален и комплексен број).

Решение `oop_av62.cpp`

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

class Complex {
private:
    float a, b;
public:
    Complex(const float a = 0, const float b = 0) {
        this->a = a;
        this->b = b;
    }
    Complex operator+(const Complex &c) {
        Complex res(a + c.a, b + c.b);
        return res;
    }

    friend Complex operator-(const Complex &c1, const Complex &c2); //globalna funkcija

    Complex operator*(const Complex &c) {
        return Complex(a * c.a - b * c.b, b * c.a - a * c.b);
    }

    Complex operator/(const Complex &c) {
        float m = c.a * c.a + c.b * c.b;
        float r = (a * c.a - b * c.b) / m;
```

```
        return Complex(r, (b * c.a + b * c.b) / m);
    }

    Complex &operator+=(const Complex &c) {
        a += c.a;
        b += c.b;
        return *this;
    }

    Complex &operator-=(const Complex &c) {
        a -= c.a;
        b -= c.b;
        return *this;
    }
    Complex &operator*=(const Complex &c) {
        a = a * c.a - b * c.b;
        b = b * c.a - a * c.b;
        return *this;
    }
    Complex &operator/=(const Complex &c) {
        *this = *this / c;
        return *this;
    }

    bool operator==(const Complex &c) {
        return a == c.a && b == c.b;
    }

    float getA() const {
        return a;
    }
    float getB() const {
        return b;
    }

    Complex operator+(float n) {
        Complex res(a + n, b);
        return res;
    }
    friend Complex operator+(float n, Complex &c);

    friend ostream &operator<<(ostream &x, const Complex &c) {
        x << c.a;
        if (c.b >= 0) {
            x << "+";
        }
        x << c.b << "j";
        return x;
    }
};

Complex operator-(const Complex &c1, const Complex &c2) {
    return Complex(c1.a - c2.a, c1.b - c2.b);
}
Complex operator+(float n, Complex &c) {
    Complex res(c.a + n, c.b);
    return res;
}


int main() {
    Complex c1(2, -6);
    Complex c2(3, 5);
    Complex c = c1 + c2;
    cout << c1 << " + " << c2 << " = " << c << endl;
    c = c1 - c2;
    cout << c1 << " - " << c2 << " = " << c << endl;
    c = c1 * c2;
    cout << c1 << " * " << c2 << " = " << c << endl;
    c = c1 / c2;
    cout << c1 << " / " << c2 << " = " << c << endl;
    if (c == c1) {
        cout << "Numbers are equal" << endl;
    }

    c = c1 + 2;
```

```
    cout << c1 << " + " << 2 << " = " << c << endl;
    c = 2 + c1;
    cout << 2 << " + " << c1 << " = " << c << endl;

    return 0;
}
```

2.2. Ученици

Да се напише класа за опис ученици. За секој ученик се чува името (динамички алоцирана низа од знаци), просекот (децимален број) и школска година (цел број). За оваа класа да се имплементираат:

- Конструктори и деструктор
- Оператор ++ за зголемување на запишаната школска година за еден
- Оператор << за печатење на ученик со сите негови податоци
- Оператор > за споредување на два ученика според просекот

Потоа треба да се креира класа за опишување на паралелка што содржи динамички алоцирана низа од ученици, како и број на елементи во низата. За оваа класа да се имплементираат:

- Конструктори и деструктор
- Оператор += за додавање на нов студент во паралелката
- Оператор ++ за зголемување на запишаната школска година за еден за сите студенти во низата
- Оператор << за печатење на сите ученици во паралелката
- Метод `nagrada` што ги печати само оние ученици кои имаат просек 10.0.
- Метод `najvisokProsek` што го печати највисокиот просек во паралелката

Решение `oop_av63.cpp`

```
#include <iostream>
#include <string.h>
#define MAX 100
using namespace std;
class Ucenik
{
private:
    char *ime;
    float prosek;
    int godina;
public:
    Ucenik (const char* ii = "", float pp = 0, int gg = 0)
    {
```

```
        ime = new char[strlen(ii) + 1];
        strcpy (ime, ii);
        prosek = pp;
        godina = gg;
    }
    Ucenik (const Ucenik& u)
    {
        ime = new char[strlen(u.ime) ];
        strcpy (ime , u.ime);
        prosek = u.prosek;
        godina = u.godina;
    }
    ~Ucenik() {
        delete [] ime;
    }

    Ucenik& operator= (const Ucenik& u)
    {
        if (this != &u)
        {
            delete  []  ime;
            ime = new char[strlen(u.ime)] ;
            strcpy (ime, u.ime) ;
            prosek = u.prosek ;
            godina = u.godina ;
        }
        return *this ;
    }
    Ucenik& operator++() { //prefiksen operator
        godina++ ;
        return *this ;
    }
    Ucenik operator++(int) { //postfiksen operator
        Ucenik u(*this) ;
        godina++ ;
        return u;
    }
    float getProsek() {
        return prosek;
    }
    // globalna funkcija za preoptovaruvanje na operatorot <<
    // ovaa funkcija e prijatelska na klasata Ucenik
    friend ostream& operator<< (ostream& o, const Ucenik& u)
    {
        return o << "Ime:" << u.ime << ", godina:" << u.godina << ",prosek:" << u.prosek
<< endl;
    }
    friend bool operator> (const Ucenik& u1, const Ucenik& u2);
};

//globalna funkcija za preoptovaruvanje na operatorot >
bool operator> (const Ucenik& u1, const Ucenik& u2)
{
    return (u1.prosek > u2.prosek);
}

class Paralelka
{
private:
    Ucenik* spisok;
    int vkupno;
public:
    Paralelka (Ucenik* s = 0, int v = 0)
    {
        vkupno = v;
        spisok = new Ucenik [vkupno];
        for (int i = 0; i < vkupno ; i ++)
            spisok[i] = s[i];
    }

    Paralelka (const Paralelka &p)
    {
        this -> vkupno = p.vkupno;
        this -> spisok = new Ucenik[vkupno];
        for (int i = 0; i < vkupno; i ++)
            spisok[i] = p.spisok[i];
```

```
    }

    ~Paralelka() {
        delete [] spisok;
    }
    Paralelka& operator+= (Ucenik u) {
        Ucenik* tmp = new Ucenik[vkupno + 1];
        for (int i = 0; i < vkupno; i++)
            tmp[i] = spisok[i];
        tmp [vkupno ++] = u;
        delete [] spisok;
        spisok = tmp;
        return * this ;
    }

    Paralelka& operator++() {
        for (int i = 0; i < vkupno; i++)
            spisok[i]++;
        return *this;
    }
    Paralelka operator++(int) {
        Paralelka p(*this);
        for (int i = 0; i < vkupno; i++)
            spisok[i]++;
        return p;
    }

    friend ostream& operator<< (ostream& o, const Paralelka& p)
    {
        for (int i = 0; i < p.vkupno; i ++)
            o << p.spisok[i];
        return o;
    }

    void nagradi()
    {
        for (int i = 0; i < vkupno; i++)
            if (spisok[i].getProsek() == 10.0)
                cout << spisok[i];
    }

    void najvisokProsek()
    {
        Ucenik tmpU = spisok[0];
        for (int i = 0; i < vkupno; i++)
            if ( spisok[i] > tmpU)
                tmpU = spisok[i];
        cout << "Najvisok prosek vo paralelkata:" << tmpU.getProsek() << endl;
    }
};

int main ()
{
    Ucenik u1("Martina Martinovska", 9.5, 3);
    Ucenik u2("Darko Darkoski", 7.3, 2);
    Ucenik u3("Angela Angelovska", 10, 3);

    Paralelka p;
    p += u1;
    p += u2;
    p += u3;

    cout << p;
    cout << "Nagradeni:" << endl;
    p.nagradi();
    cout << endl;
    p.najvisokProsek();
    cout << endl;

    u2++;
    cout << p;
    cout << endl;
    p++;
    cout << p;

    return 0;
```

```
}
```

3. Изворен код од примери и задачи

https://github.com/finki-mk/OOP/

Source code ZIP