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

Object oriented programming

Exercises 6

Version 1.0, 21 March, 2017

Table of Contents

1. Operators overloading

1.1. Example 1

In the class `Array` implement the following operators:

- `+=` adding new elements in the array
- `-=` deletes occurrences of the given integer argument
- `<<` printing the elements of the array.

Test the class in a main function.

Solution `oop_av61a_en.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];
        }
    }

    // assignment 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;
    }

    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;
    }

    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. Example 2

Extend the first example with overloading the following operators:

- `[]` for mutable access of element
- == for comparison of two objects of class `Array`.

Test the class in the main function.

Solution `oop_av61b_en.cpp`

```
#include <iostream>
#include <cstdlib>
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];
        }
    }

    // assignment 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;
    }

    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;
    }

    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 << " Out of range " << endl ;
            exit(EXIT_FAILURE);
        }
    }

    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] << " "; // using the operator []
    }
    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; // using the operator []

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

    if (a == b) cout << "Equal";
    else cout << "Not equal";

    return 0;
}
```

2. Problems

2.1. Complex number

Define a class for complex numbers. For each complex number keep information for the real and for the imaginary part.

Overload the operators `+`, `-`, `*`, `/`, `+=`, `-=`, `*=`, `/=` for executing the appropriate operations for complex numbers.

Implement the operator << for printing.

Also implement the operator + for addition of complex number and real number and vice versa.

Solution `oop_av62_en.cpp`

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

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

    friend Complex operator-(const Complex &c1, const Complex &c2); // as global function

    Complex operator*(const Complex &c) {
        return Complex(real * c.real - imag * c.imag, imag * c.real - real * c.imag);
    }

    Complex operator/(const Complex &c) {
        float m = c.real * c.real + c.imag * c.imag;
        float r = (real * c.real - imag * c.imag) / m;
        return Complex(r, (real * c.real + imag * c.imag) / m);
    }

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

    Complex &operator-=(const Complex &c) {
        real -= c.real;
        imag -= c.imag;
        return *this;
    }
    Complex &operator*=(const Complex &c) {
        real = real * c.real - imag * c.imag;
        imag = imag * c.real - real * c.imag;
        return *this;
```

```
    }
    Complex &operator/=(const Complex &c) {
        *this = *this / c;
        return *this;
    }

    bool operator==(const Complex &c) {
        return real == c.real && imag == c.imag;
    }

    float getReal() const {
        return real;
    }
    float getImag() const {
        return imag;
    }

    Complex operator+(float n) {
        return Complex(real + n, imag);
    }
    friend Complex operator+(float n, Complex &c);

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

Complex operator-(const Complex &c1, const Complex &c2) {
    return Complex(c1.real - c2.real, c1.imag - c2.imag);
}
Complex operator+(float n, Complex &c) {
    return Complex(c.real + n, c.imag);
}

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. Students

Implement a class for students. Each student has a name (dynamically allocated char array), average (real number) and academic year (integer). Implement the following:

- Constructors and destructor

- operator ++ that will increment the academic year for +1
- operator << for printing a student with all the information
- operator > for comparing two students by their average.

Then implement a class for a group of students that keeps dynamically allocated array of students and their number. For this class implement:

- Constructors and destructor
- operator += for adding new student in the group
- operator ++ for increasing the school year for +1
- operator << for printing all the students in the group
- method `reward` that print only students that have an average higher than 9.0.
- method `highestAverage` that will print the highest average of the group.

Solution `oop_av63_en.cpp`

```
#include <iostream>
#include <string.h>
#define MAX 100
using namespace std;

class Student
{
private:
    char *name;
    float average;
    int academicYear;
public:
    Student(const char* n = "", float a = 0, int ay = 0) {
        name = new char[strlen(n) + 1];
        strcpy(name, n);
        average = a;
        academicYear = ay;
    }

    Student(const Student& u) {
        name = new char[strlen(u.name) ];
        strcpy(name , u.name);
        average = u.average;
        academicYear = u.academicYear;
    }

    ~Student() {
        delete [] name;
    }

    Student& operator=(const Student& u) {
        if (this != &u) {
            delete [] name;
            name = new char[strlen(u.name)];
            strcpy(name, u.name);
            average = u.average;
            academicYear = u.academicYear;
        }
        return *this;
    }
    Student& operator++() { // prefix operator
```

```
        academicYear++;
        return *this;
    }
    Student operator++(int) { // postfix
        Student u(*this);
        academicYear++;
        return u;
    }
    float getAverage() {
        return average;
    }
    friend ostream& operator<<(ostream& o, const Student& u) {
        return o << "Name: " << u.name << ", academicYear: " << u.academicYear << ",
average: " << u.average << endl;
    }
    friend bool operator>(const Student& s1, const Student& s2);
};

bool operator>(const Student& s1, const Student& s2) {
    return s1.average > s2.average;
}

class Group
{
private:
    Student* students;
    int count;
    void copy(const Group &g) {
        this -> count = g.count;
        this -> students = new Student[count];
        for (int i = 0; i < count; i ++)
            students[i] = g.students[i];
    }
public:
    Group(Student* s = 0, int c = 0) {
        count = c;
        students = new Student [count];
        for (int i = 0; i < count; i ++)
            students[i] = s[i];
    }

    Group(const Group &g) {
        copy(g);
    }

    ~Group() {
        delete [] students;
    }
    Group& operator+=(Student s) {
        Student* tmp = new Student[count + 1];
        for (int i = 0; i < count; i++)
            tmp[i] = students[i];
        tmp [count ++] = s;
        delete [] students;
        students = tmp;
        return *this;
    }

    Group& operator++() {
        for (int i = 0; i < count; i++)
            students[i]++;
        return *this;
    }
    Group operator++(int) {
        Group g(*this);
        for (int i = 0; i < count; i++)
            students[i]++;
        return g;
    }

    friend ostream& operator<<(ostream& o, const Group& p) {
        for (int i = 0; i < p.count; i ++)
            o << p.students[i];
        return o;
    }
```

```
    void reward() {
        for (int i = 0; i < count; i++)
            if (students[i].getAverage() > 9.0)
                cout << students[i];
    }

    void highestAverage() {
        Student tmpU = students[0];
        for (int i = 0; i < count; i++)
            if (students[i] > tmpU)
                tmpU = students[i];
        cout << "Highest average in the group:" << tmpU.getAverage() << endl;
    }
};

int main() {
    Student s1("Martina Martinovska", 9.5, 3);
    Student s2("Darko Darkoski", 7.3, 2);
    Student s3("Angela Angelovska", 10, 3);

    Group group;
    group += s1;
    group += s2;
    group += s3;

    cout << group;
    cout << "Reward:" << endl;
    group.reward();
    cout << endl;
    group.highestAverage();
    cout << endl;

    s2++;
    cout << group;
    cout << endl;
    group++;
    cout << group;

    return 0;
}
```

3. Source code of the examples and problems

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

Source code ZIP