**Programe simple cu obiecte în C++**

## Obiective

Specificarea, proiectarea şi implementarea problemelor cu obiecte în C++.

## Aspecte teoretice

Probleme simple cu obiecte în C++.

## Probleme

Se cunosc informații despre mai mulți studenți: vârsta și numele (un șir de caractere alocat dinamic). Să se selecteze toți studenții a căror nume conține un șir de caractere dat și cu vârsta peste o limită dată.

Iterația 1. Să se definească și să se testeze clasa Student (constructor implicit, constructor cu parametri, constructor de copiere, destructor, operator=, setVarsta, setNume, getVarsta, getNume). Se recomandă următoarea structură a aplicației:
- Student.h, Student.cpp
- operations.h, operations.cpp
- tests.h, tests.cpp
- application.cpp

Iterația 2: Se definește clasa Repo (contructori, destructor, addElem, getAll) și se fac teste similare celor din iterația 1.
Se recomandă următoarea structură a aplicației:
- Student.h, Student.cpp
- Repo.h, Repo.cpp
- tests.h, tests.cpp
- application.cpp

Student.h

```h
#ifndef STUDENT_H
#define STUDENT_H

#include <iostream>
#include <string.h>

class Student{
private:
    char* name;
    int age;
public:
    Student();
    Student(char* n, int a);
    Student(const Student &s);
    ~Student();
    char* getName();
    int getAge();
    void setName(char *n);
    void setAge(int a);
    Student& operator=(const Student &s);
    bool operator==(const Student &s);
};

#endif
```

Student.cpp

```cpp
#include "Student.h"
#include <iostream>
#include <string.h>

// Constructor
// In: -
// Out: an empty object of type Student
Student::Student(){
    this->name = NULL;
    this->age = 0;
}

// Constructor with parameters
// In: a name (string) and an age (integer)
// Out: an object of type Student that contains the given info
Student::Student(char* n, int a){
    this->name = new char[strlen(n) + 1];
    strcpy_s(this->name, sizeof this->name, n);
    this->age = a;
}

// Copy constructor
// In: an object s of type Student
// Out: another object of type Student that contains the same info as s
Student::Student(const Student &s){
    this->name = new char[strlen(s.name) + 1];
    strcpy_s(this->name, sizeof this->name, s.name);
    this->age = s.age;
}

// Desonstructor
// In: an object of type Student
// Out: -
Student::~Student(){
    if (this->name){
        delete[] this->name;
        this->name = NULL;
    }
}

// getter
// In: an object of type Student
// Out: name of the student
char* Student::getName(){
    return this->name;
}

// getter
// In: an object of type Student
// Out: age of the student
int Student::getAge(){
    return this->age;
}

// setter
// In: an object of type Student and a name (string)
// Out: the same object with a new name
void Student::setName(char *n){
    if (this->name){
        delete[] this->name;
    }
    this->name = new char[strlen(n) + 1];
    strcpy_s(this->name, sizeof this->name, n);
}
```

```
// setter
// In: an object of type Student and an age (integer)
// Out: the same object with a new age
void Student::setAge(int a){
    this->age = a;
}

// assignment operator
// In: two objects of type Student (the current one, this, and s)
// Out: the new state of the current object (this)
Student& Student::operator=(const Student &s){
    this->setName(s.name);
    this->setAge(s.age);
    return *this;
}

// comparator
// In: two objects of type Student (this and s)
// Out: true or false
bool Student::operator==(const Student &s){
    return ((strcmp(this->name, s.name) == 0) && (this->age == s.age));
}
```

Operations.h

```
#include "Student.h"
#include "Repo.h"

void filterStudents(Student students[], int n,
char* s, int a, Student studFilter[], int &m);

void filterStudentsWithRepo(Repo &rep, char* s,
 int a, Student studFilter[], int &m);
```

Operations.cpp

```
#include "operations.h"

// filter all the students of a given name and older than a given limit
// In: an array of students and their number (integer), a name (String), an age (integer)
// Out: an array of filtered students and their number (integer)
void filterStudents(Student students[], int n, char* s, int a, Student studFilter[], int &m){
    m = 0;
    for(int i = 0; i < n; i++){
        if ((strcmp(s, students[i].getName()) == 0) && (students[i].getAge() >= a)){
            studFilter[m++] = students[i];
        }
    }
}

// filter all the students of a given name and older than a given limit
// In: an array of students and their number (integer), a name (String), an age (integer)
// Out: an array of filtered students and their number (integer)
void filterStudentsWithRepo(Repo &rep, char* s, int a, Student studFilter[], int &m){
    for(int i = 0; i < rep.getSize(); i++){
        Student crtStudent = rep.getItemFromPos(i);

        if ((strcmp(s, crtStudent.getName()) == 0) && (crtStudent.getAge() >= a)){
            studFilter[m++] = crtStudent;
        }
    }
}
```

Repo.h

```
#ifndef REPO_H
#define REPO_H
#include "Student.h"
using namespace std;

class Repo{
private:
    Student students[10];
    int noStudents;

public:
    Repo();
    ~Repo();
```

Repo.cpp

```
#include "Repo.h"

Repo::Repo(){
    this->noStudents = 0;
}
Repo::~Repo(){}
void Repo::addItem(Student &s){
    this->students[this->noStudents++] = s;
}
Student Repo::getItemFromPos(int pos){
    return this->students[pos];
}
int Repo::getSize(){
    return this->noStudents;
}
```

```
    void addItem(Student &s;
    Student getItemFromPos(int pos);
    int getSize();
};
#endif
```

Tests.h

```
void testFilterStudents();
void testFilterStudentsWithRepo();
void testFilterStudentsWithRepoWithStl();
```

Tests.cpp

```
#include "tests.h"
#include "operations.h"
#include "assert.h"

#include <iostream>
using namespace std;

void testFilterStudents(){
    Student s1("Ana", 19);
    Student s2("Maria", 19);
    Student s3("Ana", 20);
    Student s4("Ana", 18);
    Student studs[4] = {s1, s2,s3, s4};
    Student results[4];
    int m = 0;
    filterStudents(studs, 4, "Ana", 19, results, m);
    assert ((m == 2) && (results[0] == s1) && (results[1] == s3));
}

void testFilterStudentsWithRepo(){
    Student s1("Ana", 19);
    Student s2("Maria", 19);
    Student s3("Ana", 20);
    Student s4("Ana", 18);
    Repo rep;
    rep.addItem(s1);
    rep.addItem(s2);
    rep.addItem(s3);
    rep.addItem(s4);
    Student results[10];
    int m = 0;
    filterStudentsWithRepo(rep, "Ana", 19, results, m);
    assert ((m == 2) && (results[0] == s1) && (results[1] == s3));
}
```

App.cpp

```
#include <iostream>
#include "tests.h"
using namespace std;

int main(){
    cout << " start... " << endl;
    testFilterStudents();
    testFilterStudentsWithRepo();
    testFilterStudentsWithRepoWithStl();
    cout << " good job!! " << endl;
    return  0;
}
```