WRITTEN EXAMINATION– ROW 2

1. Define the classes **Complex** and **Vector** such that the following C++ code is correct and its results are the ones indicated in the comments. **(2p)**

```
void complex() {
        Complex a{}, b{ 1, 2 }, c{ 6, 4 }, d{ b };
        assert(a.getReal() == 0 && a.getImaginary() == 0);
        assert(c.getImaginary() == 4);
        assert(b == d);
        Complex res1 = c / 2;
        cout << res1 << "\n"; // prints: 3+2i
        try {
                Complex res2 = b / 0;
        }
        catch (runtime_error& e) {
                assert(strcmp(e.what(), "Division by zero!") == 0);
        }

        Vector<string> v1{ std::vector<string>{"hello", "bye"} };
        v1.printAll(std::cout); // prints: hello, bye,

        Vector<Complex> v2{ std::vector<Complex>{a, b, c, d} };
        v2.printAll(std::cout); // prints: 0+0i, 1+2i, 6+4i, 1+2i,
}
```

2. Determine the result of the execution of the following C++ programs. If there are any errors, indicate the exact place where the errors occur. **Justify your answers. (4 x 0.75p)**

```
// a)
int main()
{
        vector<int> v{ 1, 2, 3, 4, 5 };
        vector<int>::iterator it =
std::find(v.begin(), v.end(), 4);
        v.insert(it, 8);
        it = v.begin() + 2;
        *it = 10;
        vector<int> x;
        std::copy_if(v.begin(), v.end(),
back_inserter(x), [](int a) { return a % 2 ==
0; });
        for (auto a : x)
                cout << a << " ";

        return 0;
}
```

```
// b)
class Ex1 {
public:
        Ex1() { cout << "Exception1 "; }
        Ex1(const Ex1& ex) { cout << "Copy_ex1 ";
}
};

class Ex2 : public Ex1 {
public:
        Ex2() { cout << "Exception2 "; }
        Ex2(const Ex2& ex) { cout << "Copy_exc2
"; }
};
void except(int x) {
        if (x < 0)
                throw Ex1{};
        else if (x == 0)
                throw Ex2{};
        cout << "Done ";
}
int main()
{       try {
                cout << "Start ";
                try {
                        except(0);
                }
                catch (Ex1& e) {}
                except(-2);
        } catch (Ex1 e) {}
        return 0;
}
```

```
// c)
```

```
// d)
```

```
class B
{
public:
        void f() { cout << "B.f "; }
        virtual ~B() {}
};

class D1 : public B
{
public:
        virtual void f() { cout << "D1.f "; }
        virtual ~D1() {}
};

class D2 : public D1
{
public:
        void f() { cout << "D2.f "; }
};

int main()
{
        B* b1 = new B{};  b1->f(); delete b1;
        B* b2 = new D1{}; b2->f(); delete b2;
        B* b3 = new D2{}; b3->f(); delete b3;
        D1* d = new D2{}; d->f();  delete d;
        return 0;
}
```

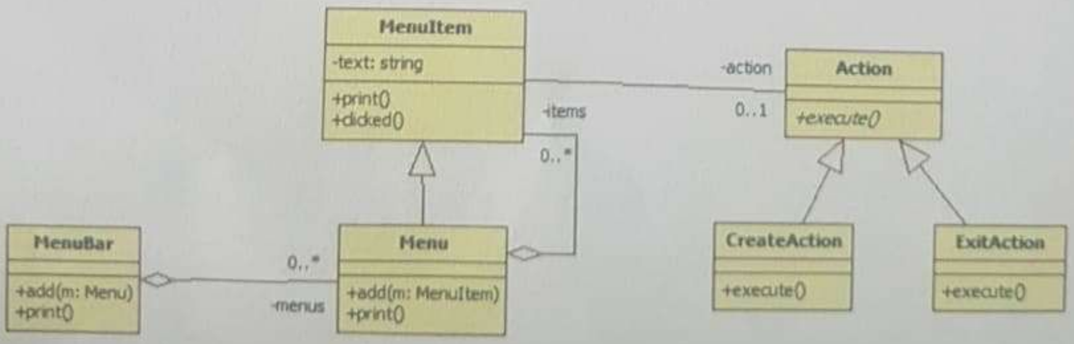
```
class Vector {
        int* elems;
        int size;

public:
        Vector() : size{ 0 } { elems = new
int[10]; }
        void add(int elem) {
                elems[size++] = elem; }
        int& operator[](int pos) {
                if (pos < 0 || pos >= size)
                        throw std::runtime_error{
"Index out of bounds." };
                return elems[pos];
        }
        ~Vector() { delete [] elems; }
};

int main()
{
        Vector v1;
        v1.add(0);
        v1.add(1);
        Vector v2 = v1;
        try {
                v1[0] = 2;
                cout << v1[0] << " " << v1[1] << "
";

                cout << v2[0] << " " << v2[1] << "
"; }
        catch (std::runtime_error& e) { cout <<
e.what(); }
        return 0;
}
```

3. Write a C++ application which demonstrates the construction of menus, as follows:
   a. The class *Action* is abstract. The *execute* function will print "Create file" for a *CreateAction* and "Exit application" for an *ExitAction*. **(0.75p)**
   b. A *MenuItem* has a text and an associated action. When a menu item is displayed (function *print()*), its text is displayed. When a *MenuItem* is clicked (function *clicked()*), its text is displayed and its associated action (if this is valid), is executed. **(1p)**
   c. A *Menu* can contain several menu items. However, a menu can also be a menu item for another menu (a submenu). When a menu is displayed, besides showing the menu's text, all its submenus are displayed. **(0.75p)**
   d. A *MenuBar* contains several *Menus*. When a menu bar is displayed, all its elements are displayed. **(0.5p)**
   e. The main application will show a menu bar containing the menus *File* and *About*. None of these have any actions associated. *File* has *New* and *Exit* as submenus and New has *Text* and *C++* as submenus. Exit has an *ExitAction* associated, while *Text* and *C++* each have a *CreateAction*. Simulate the clicking of the following sequence: File -> New -> C++ and then Exit and show the actions that are executed. Take memory management into consideration and implement it correctly. **(1p)**



Quiz

Write an application which simulates a quiz, as follows:

1. The information about the questions is in a text file. Each ***Question*** has **an id** (int), **a text** (string), the **correct answer** (string) and **a score** (int). These are read when the application starts and are also stored in the file by the program.
2. Another file contains information about the participants. Each ***Participant*** has **a name** (string) and **a score** (int). This file is manually created and it is read when the application starts. The initial score is 0 for each participant.
3. When the application is launched, a new window is created for the presenter (with the title "Presenter") and also one for each participant, having as title the participant's name. **(0.5p)**
4. The window of the presenter will show all the questions, with their id, text, correct answer and associated score, sorted by id. **(0.75p)**
5. The windows of the participants will show all the questions, with their id, text and associated score, sorted descending by score. **(0.75p)**
6. Only the presenter can add questions, by inputting the question's id, text and correct answer and pressing a button "Add". This operation fails if the text is empty or if there is another question with the same id. The user will be informed if the operation fails. **(1p)**
7. The participants can answer questions, by selecting the question, inputting the answer in a text edit and pressing a button "Answer":
   a. A participant cannot answer the same question twice. When a question is answered, it will have a green background in the participant's list and when clicking it, the "Answer" button will be disabled. **(1p)**
   -1
   b. The score of each participant is shown in his/her window. When a question was answered correctly, the score of the participant increases by the score of the question that was answered. **(1p)**
8. When a modification is made by the presenter, all the participants will see the modified list of questions. **(2.5p)**
9. When the application is finished, the questions file will be updated. **(0.5p)**

Observations — 0.5

1. **1p - of**
2. Specify and test the following functions (repository / controller): **(1p)** — 1
   a. Function which adds a question. **(0.5p)**
   b. Function which updates a participant's score. **(0.5p)**
3. Use a layered architecture. If you do not use a layered architecture, you will receive 50% of each functionality.
4. If you do not read the data from file, you will receive 50% of functionalities 3, 4, 5, 6 and 9.

Non-functional requirements

1. Use STL to represent your data structures.
2. Use objects *Question* and *Participant* to represent the necessary data.
3. Use a class *Repository* to manage your questions and your participants.

You are allowed to use Qt Designer.
You are allowed to use the following sites for documentation, but nothing else:
- http://doc.qt.io/qt-5/
- http://en.cppreference.com/w/
- http://www.cplusplus.com/

commit @ scs.ubbcluj.ro