WRITTEN EXAMINATION – ROW 2

1. Given the test function below, specify and implement the function **fct**. **(2p)**

```
void testFct()
{
	vector<int> v1{ 4, 2, 1, -4};
	assert(fct<int>(v1) == 3);
	vector<int> v2;
	try {
		fct<int>(v2);
		assert(false);
	}
	catch (std::exception&) { assert(true); }

	vector<double> v3{ 2, 10.5, 5, -10 };
	assert(fct<double>(v3) == 7.5);

	vector<string> v4{ "y", "q", "a", "m" };
	assert(fct<string>(v4) == "yqam");
}
```

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.**

```
// a)

#include <iostream>
using namespace std;

class B {
public:
    virtual void f() { cout << "B.f "; }
    virtual ~B() { cout << "~B "; }
};
class D : public B {
private:
    B& b;
public:
    D(B& _b) : b(_b) { cout << "D "; b.f(); }
    void f() override { cout << "D.f "; }
    ~D() { cout << "~D "; }
};

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

```
// b)

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

string except(int x)
{
	if (x < 0)
		throw string{ "Negative " };
	return "Positive ";
}

int main()
{
    cout << "One ";
    try {
		cout << except(3);
		cout << except(-2);
		cout << except(5);
    }
    catch (string& ex) { cout << ex << " "; }
    return 0;
}
```

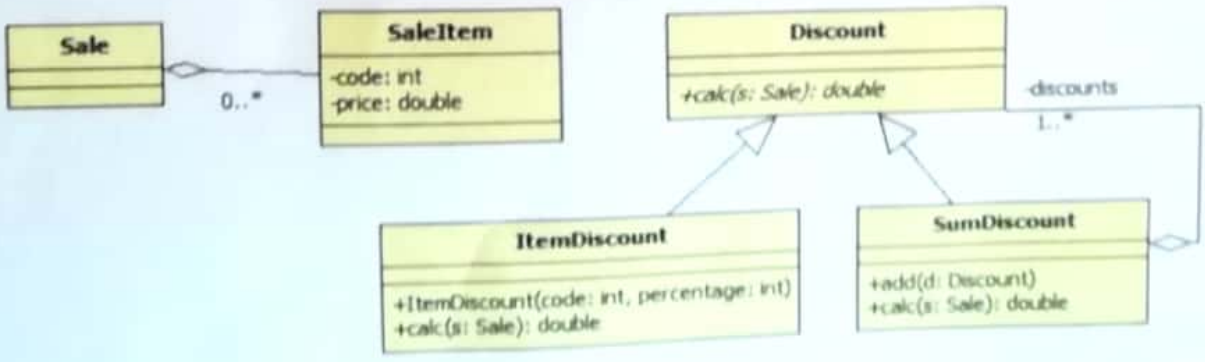
```
// c)
#include <iostream>
#include <string>
```

```
// d)
#include <iostream>
#include <string>
```

```
using namespace std;
template <typename T, typename U>
U fct2(T a, T b, U x, U y)
{
	cout << a << " ";
	cout << b << " ";
	if (a == b)
		return x + y;
	return x;
}

class A {
	int a;
public:
	A(int _a) : a{ _a } {}
};

int main()
{
    cout << fct2<int, int>(10, 10, 5, 5) << " ";
    cout << fct2<double, int>(10, 10.5, 5, 5) <<
" ";
    cout << fct2<int, string>(-2, -2, "Good ",
"luck!");
    cout << fct2<int, A>(1, 1, A{ 2 }, A{ 3 });
    return 0;
}
```

```
#include <vector>
using namespace std;
int main()
{
    vector<string> str{ "a", "b", "c", "d" };
    str.erase(str.begin() + 2);
    vector<string>::iterator it =
str.begin();
    str.insert(it + 2, "b");
    str.insert(str.end() - 1, "a");
    str.pop_back();
    it = str.begin();
    while (it != str.end())
    {
        cout << *it << " ";
        it++;
    }
    return 0;
}
```

3. Write a C++ application which simulates how sale discounts are applied, as follows:
   a. A *Sale* contains several *SaleItem*. Each *SaleItem* has a code and a price. **(0.5p)**
   b. The class *Discount* is abstract. Its role is to help compute the discount for a given sale. **(0.5p)**
   c. The class *ItemDiscount* computes a discount for those elements in a sale that have a given code, by discounting the item's price by the given percentage. **(0.75p)**
   d. The class SumDiscount computes a sum of discounts, as shown on the diagram. The discounts may be applied for different products. **(1p)**
   e. The main application will create a sale with 3 items. Two of the items have a discount of 10% and 15%, respectively. The application will print the total discount for the given sale. **(1p)**
   f. Take memory management into consideration and implement it correctly. **(0.25p)**



1p - of

WRITTEN EXAMINATION – ROW 1

1. Given the test function below, specify and implement the function **fct. (2p)**

```
void testFct()
{
	vector<int> v1{ 4, 2, 1, 6, 3, -4 };
	assert(fct<int>(v1) == 6);
	vector<int> v2;
	try {
		fct<int>(v2);
		assert(false);
	}
	catch (std::exception&) { assert(true); }

	vector<double> v3{2, 10.5, 6.33, -100, 9, 1.212};
	assert(fct<double>(v3) == 10.5);

	vector<string> v4{ "y", "q", "a", "m" };
	assert(fct<string>(v4) == "y");
}
```

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)**

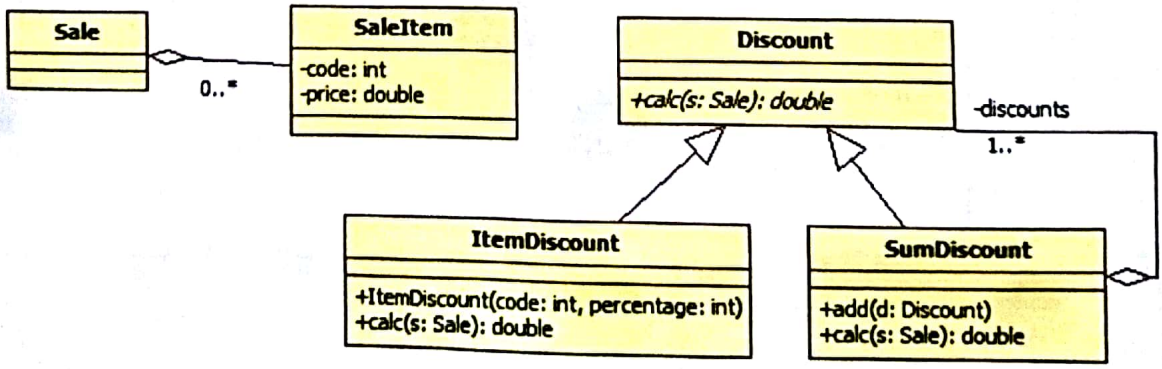
```
// a)

#include <iostream>
using namespace std;

int except(bool ex)
{
    if (ex)
       throw 10;
    cout << "Finished function." << endl;
}


int main()
{
    cout << 1 << " ";
    try{
       cout << except(true) << " ";
       cout << except(5 < 5) << " ";
    }
    catch (int& ex) { cout << ex << " "; }
    cout << 40 << " ";
    return 0;
}
```

```
// b)

#include <iostream>
using namespace std;

class B {
public:
	virtual void f() { cout << "B.f "; }
	virtual ~B() { cout << "~B "; }
};
class D : public B {
private:
	B& b;
public:
	D(B* b) : b(*b) {}
	void f() override { b.f(); cout << "D.f "; }
	virtual ~D() { cout << "~D "; }
};

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

```cpp
// c)
#include <iostream>
#include <string>
using namespace std;

template <typename T>
class elem {
private:
    T x;
public:
    elem(T _x) : x{ _x } {}
    static T add(T a, T b) { return a + b;
}
    elem& operator+=(const T& a)
{ x += a; return *this; }
    T get() { return x; }
};

int main()
{
    cout << elem<string>::add("Answer to
", "life ");
    elem<int> e{ 3 };
    e += 39;
    cout << e.get();
    return 0;
}
```

```cpp
// d)
#include <iostream>
#include <string>
#include <vector>
using namespace std;

int main()
{
    vector<string> str{ "a", "b", "c", "d" };
    vector<string>::iterator it = str.end();
    it--;
    *it = "a";
    it--;
    str.erase(it);
    str.insert(it, "b");
    for (it = str.begin(); it != str.end(); it++)
       cout << *it << " ";
    return 0;
}
```

3. Write a C++ application which simulates how sale discounts are applied, as follows:
   a. A *Sale* contains several *SaleItem*. Each *SaleItem* has a code and a price. **(0.5p)**
   b. The class *Discount* is abstract. Its role is to help compute the discount for a given sale. **(0.5p)**
   c. The class *ItemDiscount* computes a discount for those elements in a sale that have a given code, by discounting the item's price by the given percentage. **(0.75p)**
   d. The class SumDiscount computes a sum of discounts, as shown on the diagram. The discounts may be applied for different products. **(1p)**
   e. The main application will create a sale with 3 items. Two of the items have a discount of 10% and 15%, respectively. The application will print the total discount for the given sale. **(1p)**
   f. Take memory management into consideration and implement it correctly. **(0.25p)**



1p - of

Issue Tracker

Write an application which simulates the development and testing of a software application, as follows:

1. The information about the development team is in a text file. Each member of the team - ***User*** has a **name** (string) and **a type** (string), which indicates whether the user is *a tester* or *a programmer*. This file is manually created and it is read when the application starts.
2. Another file contains information about the issues reported by the testers. Each ***Issue*** has a **description** (string), **a status** (can be *open* or *closed)*, **the reporter** – the name of the person who reported it and **the solver** – the name of the person who solved it. These are read when the application starts and are also stored in the file by the program.
3. When the application is launched, a new window is created for each user, having as title the user's name and type (tester or programmer). **(0.5p)**
4. Each window will show all the issues, with their description, status, reporter and solver, sorted by status and by description. **(1p)**
5. Only testers can report issues, by inputting the issue's description and pressing a button "Add". The issue's reporter will automatically be set – this will be the name of the tester who added it. This operation fails if the description is empty or if there is another issue with the same description. The user will be informed if the operation fails. **(1.25p)**
6. Both programmers and users can remove issues. An issue can only be removed if its status is *closed*. **(1p)**
7. Only programmers can resolve issues, by selecting the issue and pressing a button "Resolve". This button is activated only if the status of selected issue is *open*. When an issue is resolved, the name of the issue's solver is automatically updated to the name of the programmer who solved it. **(1.25p)**
8. When a modification is made by any user, all the other users will see the modified list of issues. **(2p)**
9. When the application is finished, the issues file will be updated. **(0.5p)**

Observations

1. **1p - of**
2. Specify and test the following functions (repository / controller): **(1.5p)**
   a. Function which adds an issue. **(0.5p)**
   b. Function which removes an issue. **(0.5p)**
   c. Function which updates an issue's status and programmer. **(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 and 6.

Non-functional requirements

1. Use STL to represent you data structures.
2. Use objects *User* and *Issue* to represent the necessary data.
3. Use a class *IssueRepository* to manage your users and your issues.
4. Create a custom defined exception class to handle the constraints for the requirements 5 and 6 and use objects of this class.

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/