Examination Guide

Contents

1. Objectives

- Understand and be able to use OOP concepts, as well as C++.
- Solve small/medium scale problems using OOP.
- Explain class structures as fundamental, modular building blocks.
- Understand the role of inheritance, polymorphism, dynamic binding and generic structures in building reusable code
- Use classes written by other programmers, use libraries (STL).
- Write small/medium scale C++ programs with simple graphical user interfaces in Qt.

2. Course Contents

1. C Programming Language Fundamentals
2. Functions, Modular Programming and Memory Management in C
3. Object-Oriented Programming, Classes and Objects in C++
4. Templates, C++ Standard Template Library
5. Inheritance, UML diagrams
6. Polymorphism
7. I/O Streams, Exception handling
8. RAII, Smart Pointers in STL
9. Qt framework
10. Signals and slots in Qt
11. Model/View Architecture
12. Design Patterns – Observer, Adapter
13. Design Patterns – Iterator, Composite, Strategy
14. Revision

3. EVALUATION

3.1. During the semester

Lab grade – **L (40% of the final grade)**:

- 50%: Lab assignments: weighted arithmetic mean of 10 lab assignments, with 0 for the labs you did not present.
- 50%: 3 practical tests during the labs (10%, 15%, 25%)
- Additional work: laboratory bonus (LB) – **this is added to the final grade**.

The precondition to attend the examination in the regular session is: L ≥ 5 (no rounding).

Seminar activity – **SA**: 0 – 0.5 – optional bonus for your activity during the seminars. **This is added to the final grade.**

3.2. During the examination session

Written examination – **W (30% of the final grade)**: on your examination date.

Practical examination – **P (30% of the final grade)**: on your examination date, after the written examination.

Final Grade: G = 0.4 x L + 0.3 x P + 0.3 x W + SA + LB

To pass the examination all grades (L, W, P) must be ≥ 5 (no rounding).

3.3. During the retake session

- The final grade in the retake session is computed by the same algorithm presented above.
- During the retake session you can hand in laboratory work, but are limited to a maximum laboratory grade (**L**) of 5.00. *All lab assignments should be properly solved in order to receive 5.*
- You can choose to retake the written, practical, or both examinations in case you have failed/not attended during the regular examination session.
- If you want to increase the grade you obtained during the regular session, you may participate to the examinations during the retake session. Your final grade will be the largest one between those obtained.

4. EXAMINATION DATES

06.06.2022 – group 913

08.06. 2022 – group 917 (backup date for groups 913, 911)

11.06. 2022 – group 911 (backup date for group 916)

12.06. 2022 – group 916 (backup date for groups 914, 917)

14.06. 2022 – group 914 (backup date for group 912)

16.06. 2022 – group 912 (backup date for group 915)

17.06. 2022 – group 915

Important rules and observations:

- Attending the examination this year (normal or retake session) is only possible if you fulfill the attendance criterion (5 for seminar activities, 12 for laboratory activities).
- A grade < 5 for the laboratory activity implies that you are not allowed to take the examination during the normal session. You can take the exam during the retake session (see Section 3.3 for more details).
- Make sure you've fulfilled your financial obligations towards the University, otherwise we are not allowed to grade you.
- To take the exam on the backup date, you need a very good reason. In such situations, write me a message on Microsoft Teams **at least 48h beforehand!**
- Re-check the date/time of the exam beforehand.
- Arrive at least 5 minutes earlier and bring a photo ID.
- Both the written and the practical exam take place in the same day, first written, then practical, with a break in between.
- Bring your laptop, if possible. If not, let me know in advance that you will need a university computer. In this case you will have the possibility to use Qt and Qt Creator.
- Do not plagiarize! Plagiarism involves examination failing (you cannot retake the examination, not even during the retake session).

Suggestions:

- Make sure you rest the night before the examination, but do not forget to set your alarm clock (several alarms).
- You will be at the faculty many hours in the exam day, so make sure you have something to eat and drink.
- Try to pay attention to the explanations I give at the beginning of each exam, about the requirements.

- Do not stress too much. You have more chances to take the examination (retake, then you still have the following years, if necessary). *"It's not the end of the world if you fail an exam."* (Istvan Czibula).

5. Written Examination

This will last approximately 1.5 hours.

5.1. OOP concepts – C++

- templates, STL (containers, algorithms, iterators, smart pointers);
- dynamic memory allocation;
- constructors, destructors, inheritance, virtual methods, abstract class, operator overloading, static, friend elements;
- input/output streams;
- exceptions;
- UML, design patterns.

Given the test function below, define the class **Stack**. Enumerate the data member(s), including their types, the prototypes of all methods (including constructors, destructors, if necessary), and implement the method allowing the operation shown in bold in the code below.

```
void testStack()
{
	Stack<std::string> s{ 2 };
	assert(s.getMaxCapacity() == 2);
	try	{
		s = s + "examination";
		s = s + "oop";
		s = s + "test";
	}
	catch (std::exception& e) {
		assert(strcmp(e.what(), "Stack is full!") == 0);
	}
	assert(s.pop() == "oop");
	assert(s.pop() == "examination");
}
```

Define the classes “ToDo” and “Activity”: enumerate the data members, including their types, the prototypes of all methods (including constructors, destructors, if necessary), and implement the method allowing the operation shown in bold in the code below, such that the following C++ code is correct and its results are the ones indicated in the comments.

```
void ToDoList()
{
	ToDo<Activity> todo{};
	Activity tiff{ "go to TIFF movie", "20:00" };
	todo += tiff;
	Activity project{ "present project assignment", "09.20" };
	todo += project;

	// iterates through the activities and prints them as follows:
	// Activity present project assignment will take place at 09.20.
	// Activity go to TIFF movie will take place at 20.00.
	for (auto a : todo)
		std::cout << a << '\n';

	// Prints the activities as follows:
	// Activity go to TIFF movie will take place at 20.00.
	// Activity present project assignment will take place at 09.20.
	todo.reversePrint(std::cout);
}
```

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.

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

class D : public B
{
public:
	D() { std::cout << "D{}"; }
	void print() override { std::cout <<
"D"; }
	virtual ~D() { std::cout << "~D()"; }
};
```

```
int main()
{
	B* b[] = { new B{}, new D{} };
	b[0]->print();
	b[1]->print();
	delete b[0];
	delete b[1];
	return 0;
}
```

```cpp
class Person
{
public:
	Person() { std::cout << "Person{}"; }
	virtual void print() = 0;
	virtual ~Person() { std::cout <<
"~Person()"; }
};

class Student : public Person
{
public:
	Student() { std::cout << "Student{}";
}
	void print() override { std::cout <<
"Student"; }
	virtual ~Student() { std::cout <<
"~Student()"; }
};
```

```cpp
int main()
{
	Person* p = new Person{};
	delete p;
	Person* s = new Student{};
	s->print();
	delete s;

	return 0;
}
```

```cpp
class E
{
public:
	E() { std::cout << "E{}"; }
	virtual ~E() { std::cout << "~E()"; }
};

class DE : public E
{
public:
	static int n;
	DE() { n++; }
};

int DE::n = 0;



int fct2(int x)
{
	if (x < 0)
	{
		throw E{};
		std::cout << "number less than
0" << std::endl;
	}
	else if (x == 0)
	{
		throw DE{};
		std::cout << "number equal to 0"
<< std::endl;
	}
	return x % 10;
}
```

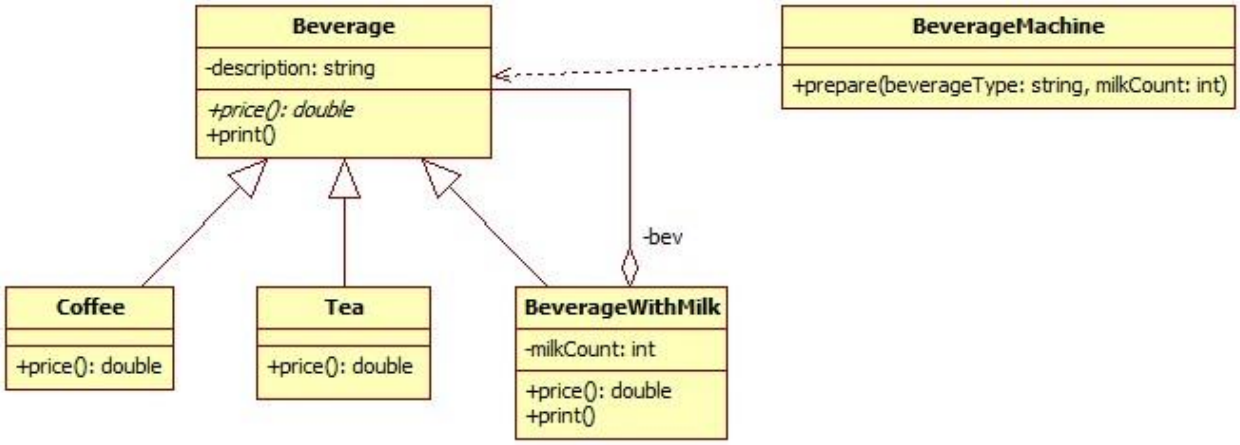
```cpp
int main()
{
	try
	{
		int res = 0;
		res = fct2(-5);
		std::cout << DE::n;
		res = fct2(0);
		std::cout << DE::n;
		res = fct2(25);
		std::cout << DE::n;
	}
	catch (E&)
	{
		std::cout << DE::n;
	}

	return 0;
}
```

```cpp
#include <deque>
#include <string>
#include <iostream>

int main()
{
	std::deque<std::string> d;
	d.push_back("A");
	d.push_front("B");
	d.push_back("C");
	d.push_front("D");

	auto itBegin = d.begin();
	auto itEnd = d.end();
	itBegin++;
	itEnd--;

	while (itBegin != itEnd)
	{
		std::cout << *itBegin << " ";
		itBegin++;
	}

	return 0;
}
```

5.2. Inheritance, polymorphism, UML, design patterns – C++

Write a program that simulates a beverage machine and corresponds to the UML diagram given below.

- All the beverages (*Beverage*) have a *description* (string) and a *price*, given by the pure virtual function *price()* in the class *Beverage*. The *print()* method in the class Beverage will print at the console the description of the beverage and its price.
- *Coffee* and *Tea* are two concrete beverages which the machine can provide. These two classes will initialize the description by calling the base class constructor with the descriptions “Coffee” and “Tea”, respectively. The prices of these beverages are 4.5 RON for coffee and 3.5 RON for tea.
- The machine allows adding milk to a beverage, thus resulting *BeverageWithMilk*. Each milk costs 0.5 RON. Printing a beverage with milk involves printing the beverage and then printing the number of portions of milk that were added. The price of a beverage with milk is computed as the price of the beverage, to which the price of all the portions of milk is added.
- When the program starts, a new object of type *BeverageMachine* is created, to simulate buying the following beverages, by calling the method *prepare*, with the given parameters, as follows:
  - “Tea”, 0 – tea, with no milk
  - “Coffee”, 1 – coffee, with milk
  - “Coffee”, 2 – coffee, with double milk
- The *prepare* method will show the prepared beverage, including its price.
- Take memory management into consideration and implement it correctly.


Beverage
-description: string
+price(): double
+print()
BeverageMachine
+prepare(beverageType: string, milkCount: int)
-bev
Coffee
+price(): double
Tea
+price(): double
BeverageWithMilk
-milkCount: int
+price(): double
+print()

6. PRACTICAL EXAMINATION

This will last approximately 2.5 hours.

Below you will find a problem statement similar to what you can expect to receive during the practical examination. The problem statement will in general follow the requirements set out between Lab 5 and Lab 14, will require reading and writing from/to a file, data validation, a graphical user interface (using Qt), the implementation of layered architecture, Observer, Model/View architecture, simple drawing in Qt and may require writing specifications and tests.

Observations:

1. Solving the following problem statement completely should be possible for you in a time span of 2.5 hours.

2. You are encouraged to bring your own laptop to the exam. You are free to use your preferred IDE. Make sure your IDE is set up correctly and it works! Make sure that Qt works! If you cannot bring a laptop, you can use university computers (Qt and Qt Creator). For this, please let me know at least one day in advance that you will be using a university computer.

3. You are allowed to use Qt Designer, if you want to.

4. Only functional features are scored (source code is not scored).

5. The problem must be started from an empty workspace. You are allowed to use the following sites for documentation, but nothing else:
- http://doc.qt.io/qt-6/
- http://en.cppreference.com/w/
- http://www.cplusplus.com/

6.1. Problem statement

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). **(1p)**
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 / service):
    a. Function which adds an issue. **(0.5p)**
    b. Function which updates an issue's status and programmer. **(0.5p)**
3. The application must use layered architecture in order for functionalities to be graded.
4. If you do not read the data from file, you will receive 50% of functionalities 3, 4, 5 and 6.

6.2. Advice for the practical examination

1. Implement a problem similar to the one in the example (Section **Problem statement**). Time yourself while solving it, make sure you can implement at least some of the functionalities in the allotted time (2.5 hours). This way you can detect where your difficulties are and you can improve yourself.
2. Build your application incrementally: one step at a time, **compile frequently**.
3. Only add one function in one step, so that you can easily revert to a functional version, in case something doesn't work.
4. **Do not ignore the errors**, solve them before continuing with writing source code. **Read the error text!**
5. Do not ignore warnings, sometimes these can indicate errors in the program.
6. Use STL containers and algorithms.
7. **Do not implement functionalities that are not required**. By doing this, you might waste valuable time and **the source code will not be graded, only the functional requirements**!
8. If there are issues that you cannot solve, try finding alternative implementations such that you can still test your code.
9. For the practical examination, build an empty Qt project and make sure that it compiles and that you can execute it (that you have everything set up properly).

7. BIBLIOGRAPHY

1. B. Stroustrup. The C++ Programming Language, Addison Wesley, 1998.
2. Bruce Eckel. Thinking in C++, Prentice Hall, 1995.
3. A. Alexandrescu. Programarea moderna in C++: Programare generica si modele de proiectare aplicate, Editura Teora, 2002.
4. S. Meyers. Effective C++: 55 Specific Ways to Improve Your Programs and Designs (3rd Edition), Addison-Wesley, 2005.
5. S. Meyers. More effective C++: 35 New Ways to Improve Your Programs and Designs, Addison-Wesley, 1995.
6. B. Stroustrup. A Tour of C++, Addison Wesley, 2013.
7. C++ reference (http://en.cppreference.com/w/).
8. Qt Documentation (http://doc.qt.io/qt-6/).
9. E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Longman Publishing, 1995.