LINKÖPING UNIVERSITY
Department of Computer and Information Science
Software and Systems
Christoffer Holm 2021-08-18

Computer examination in **TDDD38** Advanced Programming in C++

Date 2021-08-18

Time 08-14

Department IDA

Course code TDDD38

Exam code DAT1

Staff

Teacher on call: Christoffer Holm, 013-28 13 59 (christoffer.holm@liu.se)
Will answer questions through Microsoft Teams or E-mail.
Examiner: Klas Arvidsson, 013-28 21 46 (klas.arvidsson@liu.se)
Administrator: Anna Grabska Eklund, 013-28 23 62

Grading

The exam consists of three parts. Complete solutions/answers to part I and part II are required for a passing grade. It is also required that you have submitted to the “Examination rules” submission in Lisam, which confirms that you swear to follow the rules.

The third part is designated for higher grades. It consists of two assignments. To get grade 4 you must solve one of these assignments. To get grade 5 you need to solve both.

Communication

- You can ask questions to Christoffer Holm (christoffer.holm@liu.se) through the chat in Microsoft Teams or by E-mail.
- General information will be published when necessary in Microsoft Teams through the team called `Team_TDDD38_Exam_2021-08-18`. Be sure to check there from time to time. A suggestion would be to turn on notifications in Microsoft Teams so you don’t miss any important information.
- All communication with staff during the exam can be done in both English and Swedish.
- All E-mails must be sent from your official LiU E-mail address.
- In case of emergency call the teacher on call.

Rules

- You must sit in a calm environment without any other people in the same room.
- All types of communication is forbidden, the exception being questions to the course staff.
- All forms of copying are forbidden.
- You must report any and all sources of inspiration that you use. You may use cppreference.com without citing it as a source.
- When using standard library components, such as algorithms and containers, try to choose “best fit” regarding the problem to solve. Avoid unrelated/unnecessary computations and unnecessary data structures.

- C style coding is to be avoided.
- All concepts discussed during the course are OK to use.
- Your code must compile. Commented out regions of non-compiling code are acceptable if they clearly demonstrate the idea. Write a comment describing why that piece of code is commented out.
- You must be ready to demonstrate your answers to the staff after the exam if asked to.
- Failure to follow these rules will result in a *Failed* grade.

Submission

Submission will be done through Lisam on this page:
`https://studentsubmissions.app.cloud.it.liu.se/Courses/Lisam_TDDD38_2020HT_ZA/submissions`
You can also find this page by going to `https://lisam.liu.se`, navigating to the TDDD38 course page and clicking on "Submissions" in the left-hand side menu. There your should see the following submissions:

- 2021-08-18: Examination rules
- 2021-08-18: Partial submission (10:00)
- 2021-08-18: Partial submission (12:00)
- 2021-08-18: Final submission part I
- 2021-08-18: Final submission part II
- 2021-08-18: Final submission part III

Partial submission: On the marked times you must send in the current state of all your solutions (all files). *Failure to do so within 5 minutes of the marked time will result in a failing grade.* We do not expect complete or even compiling solutions at this point.
Suggestion: Set an alarm so you don't forget.

Final submission: When you are done with the exam, you must send in your solutions through "Final submission part I" and "Final submission part II". If you have attempted Part III you must also make a submission to "Final submission part III".

- Your solution(s) to part I should be source code files (.cc, .cpp, .h, .hh, .hpp).
- Your solution to part II should be a PDF document.
- Your solution(s) to part III should be one source code file per assignment and one PDF for your answers to all the questions presented in the assignments.
- The final submission must be submitted no later than 14:00.

When you have submitted your final submission in Lisam, make sure to send *all* of your files to `christoffer.holm@liu.se` and `klas.arvidsson@liu.se` by E-mail. This includes any .doc, .docx, .odt and .txt files. The subject line must be `TDDD38: Exam 2021-08-18` .

Submitting through Lisam: Attach the files to your submission and press the submit button (it doesn't matter which one if there are multiple). You can select multiple files by holding Ctrl and clicking the files you want to attach.

You will be prompted "Do you really want to submit?". Double check that you have everything, and then press "Submit" on the popup.

You will then see a popup: “You will be redirected automatically when everything is finished” Once that has finished you will redirected to the submission page. You should also get a confirmation E-mail.

Agree to the examination rules

Before starting to work on Part I you must submit the message **“I have read and understood the rules of the examination, and I swear to follow those rules”** to the submission called “2021-08-18: Examination rules” in Lisam (see above).

Do this before starting the exam!

Part I

Introduction

This part of the exam deals with practical programming skills. You will discuss your solution to this part in part II of the exam.

Note that your code should compile on Ubuntu 18 with g++ version 7 or later with the flags: `-std=c++17 -Wall -Wextra -Wpedantic`. You can test your code on ThinLinc if you don't have access to Ubuntu 18 or g++ version 7 on your local machine.

The problem

You have been hired by a company to help them improve the code in one of their libraries. This company has a lot of digital forms that need to be filled in by employees. To help them with this they developed a small library that handle general forms. However, the code is not of the highest quality and it is hard to add new features. This is where you come in: your job is to improve the code however you can.

The library along with an example program can be found in `forms.cc`. The library consists of two entities:

Field which the types of fields that can exist in a form. There are currently three types of fields available (see further down).

Form represents one of the digital forms, which is essentially a collection of fields associated with names for those fields.

Each field can be read from a stream (to fill out the form) and printed to a stream. There are three different fields right now:

DATA which represents a single data value (`double`).

TEXT represents a single line of text.

COLLECTION represents an arbitrary collection of unique data values (`double`), meaning there are no duplicate values in a collection.

Each of these fields are handled differently when filling out the form and when printing the result.

The assignment

You must identify **suitable** parts of the given code that can be improved, and then demonstrate how to make those improvements. Your improvement must involve:

- Two different STL algorithms and one STL container **OR** two different STL containers
- Classes and Polymorphism
- A Class Template **OR** A Function Template

Note: `std::vector` does not count towards the STL container requirement, *unless* it is substantially different from how it is used in the given code.
Note: It is not required that you rewrite *everything*. It is enough that you rewrite parts of the code to demonstrate your ideas and understanding.

It is up to you to show that you understand these concepts. Remember that more advanced features does not necessarily imply better code.

Note: If you have trouble showing all of these concepts in one solution, you are allowed to create different solutions based on the given code. If you do this, place each solution in its own separate file and write a comment that describe which concepts you are covering in that file.

Suggestions and hints

Suggestion: Try to quickly analyze which parts will be easier and which will be harder to rewrite and plan your time accordingly. If you want to try for higher grades our recommendation is that you are done with Part I and Part II within 3 to 4 hours.

Hint: There are a lot of comments in the code. Some of these comments contain a wishlist. These are improvements that the author would like the code to contain. You are free to use these whishlists as inspiration, but there may be other parts you wish to improve which is also OK.

Hint: Some parts may be improved by completely rewriting them. Your solution doesn't have to use code from the given file, as long as your solution performs the same (or very similar) work as the given program but in a better way.

There are more hints and suggestions in the given file.

Part II

Rules

The answer to this part must be written as a text. You need to use a program where you can insert headers, text and code examples. You can for example use Microsoft Word or OpenOffice. It is also OK to use a pure text format (for example markdown). The important part is that the formatting clearly separates headers, text and code examples (and that you can export it as a pdf). The entire text should be possible to read and understand without reading your solution to part I. This means that you have to insert relevant pieces of code from your solution into the document. You document should be around 500 to 2000 words long.

The assignment

You must answer ***ALL*** of the following questions about your solution to part I. Remember to demonstrate **suitable** usage of these concepts in each question. More advanced features does not necessarily imply better code. You **must** write one header per question.

1. Describe the class hierarchy of your solution. You should do **one** of these:
   - describe the classes and their relationships textually
   - draw a UML diagram (photos of hand drawn diagrams or digitally drawn diagrams are both OK)
2. Discuss how and why your usage of polymorphism is better than the given code. Describe the reasoning behind each virtual function, each class and the encapsulation. Discuss how these things improve the design of the program.
3. Describe a place where you used a template. Discuss why that template is an improvement compared to the given code.
4. Discuss your usage of a STL container. Discuss why you decided to use that container in favor of others.
5. Either discuss two STL algorithms *or* a second STL container. Discuss what benefits this gives to the code and why it was an improvement compared to the given code.

Part III

Introduction

You only have to write this part if you want a higher grade. However it can also help you compensate any potential flaws in part I and/or part II.

In this part two programming assignments are presented, each paired with a question.

- To get a grade 4 you need to solve one of the assignments.
- To get a grade 5 you need to solve both assignments.

We count a solution as solved if you have fulfilled the requirements specified in the assignment and if you have answered the question.

Write your answers to the questions in a separate document that you then submit as a PDF with your code to "2021-08-18: Final submission part III". Note that your answers can be short as long as they actually answer the question.

Note: We don't expect perfect solutions. If you are *close enough* we might still grade the assignment as solved. So if you feel that you are close to a solution you can still submit it. But if you do, make sure to write comments on what you have tried and why you think it didn't work.

Note: Any solution that doesn't compile will **not** be considered solved. **So make sure to comment out any code that causes compile errors.**

Assignment 1

In C++ it can be quite tedious if you want to utilize random elements in your programs. This is because you need a random number generator together with some kind of distribution that determines what type of numbers are generated. In this assignment your job is to make this easier by creating a function `random` where we generate numbers based on some parameter (what this parameter does depends on the data type).

There are three different behaviours of `random` depending on what type the template argument `T` is:

- If `T` is a floating point number (`float`, `double` or `long double`) then `random` should take a parameter `upper` of type `T`. In this case the function should generate a floating point number between `0.0` and `upper` with a uniform distribution.
- If `T` is an *integral type* then it should take a parameter `upper` of type `T` and generate an integer between `0` and `upper`. This should also be generated with a uniform distribution.
- If `T` is a container with random access iterators then it should take a parameter `base` which is a container of type `T`. `random` will then return a new container of type `T` which contains the same elements as `base`, but *shuffled* (a random order).

You may only create three overloads of `random`, one for each of the cases above. There are a few testcases as well as some examples on how to generate numbers given in `assignment1.cc`.

Hint: Use type traits (`<type_traits>`) to determine if it is a floating point or integral type.

Hint: Use `std::shuffle` to shuffle a container with random access iterators.

Question: Is it necessary to introduce a priority order between the different cases?

- If yes, what is that priority?
- If no, why not?

Assignment 2

In the `<type_traits>` header there are a lot of useful utilities for retrieving type information during compile time. One of these is `std::common_type` which takes two types and return the type that can best represent values from both types. For example, the most common type of `int` and `double` is `double`, since `double` can store integers, but not the other way around.

In this assignment you are going to create the much less useful (maybe even completely useless) sister trait `uncommon_type` which does the complete opposite. Of the two types passed to it, `uncommon_type` will return the one that *cannot* represent values of both types. So if `std::common_type<int, double>::type` is `double`, then the complete opposite should be returned from `std::uncommon_type<int, double>::type`, namely `int`.

In order to implement it you can take the result of `std::common_type` and simply use the type that wasn't returned. In order to do this `std::is_same` and `std::conditional` might be useful.

Note that `uncommon_type` should be a `struct` that takes two template parameters, `T1` and `T2`. The "returned" type should be "stored" in the type alias `uncommon_type<T1, T2>::type`.

In `assignment2.cc` there are some testcases given.

Question: What is the difference between `std::conditional` and `std::enable_if`? Why do you think the STL contain both of these similar type traits?