LINKÖPING UNIVERSITY
Department of Computer and Information Science
Software and Systems
Christoffer Holm 2021-01-14

Computer examination in

TDDD38 Advanced Programming in C++

Date 2021-01-14

Time 8-14

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_2021-01-14`. 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:

- 2021-01-14: Examination rules
- 2021-01-14: Partial submission (10:00)
- 2021-01-14: Partial submission (12:00)
- 2021-01-14: Final submission part I
- 2021-01-14: Final submission part II
- 2021-01-14: 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 assigments.
- 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 `COURSE: Exam 2021-01-14` where `COURSE` is replaced with either `TDDD38` or `726G82`.

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 "2021-01-14: 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

In `message.cc` there is a given program. The program implements a small messaging system which allows the user to convert data of different types to strings that can be sent to another program. The reciever can then convert that string back into the data. This process of converting data to strings is called *serialization*. The process of converting the strings back into data is called *deserialization*. The given program only deals with *serialization* of specific messages.

There are different types of messages that can be serialized: empty messages, messages containing integers, floating-point values and strings as well as a message that consists of a sequence of other messages. This will allow the user to convert different types of data into strings. Other than containing the data, the messages also contain information about what type of data it is.

Unfortunately this serialization system doesn't really utilize the full power of C++ since it is written by someone who isn't particularly comfortable with the language. Your job is to demonstrate to the author how this system can be improved by using modern C++. This is done by introducing classes, polymorphism, the STL and templates.

The focus for this assignment is for you to demonstrate that you can apply the language features discussed during the course to make the code better in any way you see fit. Some examples of concepts you could focus on when making the code better are: readability, maintainability, usability, code safety and efficiency. Note that this list is for inspiration, it is not a requirement. You don't have to use these concepts if you find other ways to improve the code. Just make sure to clearly discuss your intentions in part II.

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:

- At least two different STL Algorithms, one of which **must** use a lambda function.
- Classes and Polymorphism
- A Class Template **or** two Function Templates

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 piece of your solution where you use templates and explain why you made those changes. If you have multiple places in the code to choose from, it is up to you to describe the one you think demonstrates the usage of templates best.
4. Describe two places in your solution where you use different STL algorithms and explain why you made those changes.

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 "2021-01-14: 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

The *greatest common divisor* (GCD) of two numbers is the largest number that can evenly divide both those numbers. For example `gcd(3, 6) == 3`.

But GCD is not limited to just two numbers, we can calculate the GCD of an arbitrary amount of numbers. For example: `gcd(6, 120, 36, 24, 366) == 6` since `6` is the largest number that can evenly divide *all* the numbers in the list. The trick for making this work is realizing that `gcd(a, b, c) == gcd(gcd(a, b), c)`.

In this assignment you will create a class template `GCD` that takes *at least* two (but arbitrarily many more) `int` values as (non-type) template parameters. `GCD` must contain a `static` constant called `value` that contains the GCD of all the passed in template parameters.

In this assignment there are some restrictions:

- You **must** solve this assignment without using any functions (besides `main`).
- You cannot include *anything*.
- `GCD` must be able to take two *or more* integer values and calculate their GCD.

To help you with the implementation there is a sample implementations of gcd given in `assignment1.cc`. There are also a few testcases given.

Hint: Partial template specializations allows you to specialize behaviour based on the values of non-type parameters.

Hint: Non-type template parameters can be stored in a variadic pack (Namely: `int... pack`).

Suggestion: Start with solving the case for exactly two numbers.

Question: What is the difference between *template specialization* and *partial template specializations*? Can this calculation be performed in compile-time without the use of class templates? Explain.

Assignment 2

`std::get` takes an index as a template parameter and return the corresponding element from the passed in object. It can retrieve values from `std::tuple`, `std::pair`, `std::array` and `std::variant`. For example: suppose we have: `std::tuple<int, int> t {1, 2}`, then the expression `std::get<1>(t)` will return 2.

In this assignment your job is to make an even more general version that will work for all the cases mentioned above as well as for all containers (including C-arrays).

Create a function template `get` that takes two template parameters, an integer `N` and an arbitrary type `T`. `get` should only take one function parameter `t`, namely a reference to the object that we wish to access the `N`:th element from. Depending on `T`, the behaviour of `get` will differ:

1. If `std::get<N>(t)` is valid, do that.
2. If `t` has an `operator[]`, use `t[N]`.
3. Otherwise retrieve the value of the `N`:th position by using iterators.

Notice that for some containers there are overlap between these three different cases. For example, `std::vector` work with both case 2 and case 3. So you will have to disambiguate by introducing a priority.

There are testcases given in `assignment2.cc`.

Question: What are forwarding references? Can they be used to improve your solution? Explain.