LINKÖPING UNIVERSITY
Department of Computer and Information Science
Software and Systems
Christoffer Holm 2021-03-17

Computer examination in
TDDD38 Advanced Programming in C++

Date 2021-03-17

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_2021-03-17`. 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:

- 2021-03-17: Examination rules
- 2021-03-17: Partial submission (10:00)
- 2021-03-17: Partial submission (12:00)
- 2021-03-17: Final submission part I
- 2021-03-17: Final submission part II
- 2021-03-17: 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 2021-03-17` 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 "2021-03-17: 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

Event systems are quite common in for example: web applications, games and low-level programs (such as operating systems).

The idea with event systems is to trigger a certain action whenever a specific event occurs, for example whenever the program prints something to the terminal. Other things that could trigger events are: when the user clicks on a button, when the program exits or when the operating system recieves a network message.

In the given code `event.cc` there is a simple event-system framework implemented. In this system there are three structs:

`Action` represents what actions can be taken whenever an event is triggered. This could either be: print something to the terminal (called `LOG`), call a function (called `CALLBACK`) or trigger multiple different actions (called `MULTI`).

`Event` represents an event that can be triggered by the system. Each event consists of an action and a unique identifier. These identifiers are used by the system to uniquely identify a certain event.

`System` implements the entire system. The idea is that we register events that should trigger on certain occasions. In the given code there are three events that could occur: `IO` which triggers whenever the user reads or writes data (with the given `read` and `write` functions) from some stream, `USER` which triggers whenever the user calls the `trigger_user` function, and finally `EXIT` which triggers when the systems is destroyed.

The user must register actions to the system, which the system then will trigger under the specified circumstances (`IO`, `USER` or `EXIT`). The system also has functionality to remove events either by a predicate or by its unique identifier.

In this assignment the `IO` event only triggers when the user calls the `read` or `write` functions.

This system works, but it is not particularly scalable. Your job is to improve the code with modern C++ and better usage of the standard library.

Note: this is a very simplified system so you don't really have to know how "real" event-systems work. It should be enough to just read the source code and make judgements based on the code itself.

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:

- A STL container **OR** two unique STL algorithms (not `std::for_each`)
- Classes and Polymorphism
- A Class Template **AND** a 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 place where you used a class template. Discuss why that class template is an improvement compared to the given code.
4. Describe a place where you used a function template. Discuss why that function template is an improvement compared to the given code.
5. Discuss your usage of either a STL container *or* two STL algorithms. Discuss why these changes are improvements 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 "2021-03-17: 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

If you want to iterate through a container in a random order you can use `std::shuffle` together with some random number generator (for example `std::mt19937`). However that approach will then destroy your original order. In this assignment you will implement a `random_iterator` that iterates through an arbitrary container in random order. The goal is to make the following work:

```
std::vector<int> v { 1, 3, 5, 7, 9, 11, 13 };
for (int i : shuffled(v)) {
    std::cout << i << " ";
}
std::cout << std::endl;
```

In such a way that each run of the program produces a different order.

To do this you must create two class templates: `random_iterator` and `Shuffled`. You must also create a function template `shuffled`. All three of these templates must take exactly one template parameter, `Container` which represents an arbitrary container type. The function template `shuffled` takes a reference to a container as parameter and returns a `Shuffled` object.

`Shuffled` contains a reference to a container. It has a constructor that initializes the reference, and it also contains two functions: `begin()` and `end()`. Both of which returns a `random_iterator` which operates on the container that `Shuffled` refers to.

`random_iterator` must be a forward iterator. It must implement `operator*`, both versions of `operator++`, `operator==` and `operator!=`. **Note:** You **don't** have to implement the type aliases (`value_type, difference_type` etc.) that iterators normally have.

`random_iterator` must contain *at least* two data member: a container reference `container` and an `std::vector` of indicies called `left`. `left` contains which indicies are left to visit in the iteration. This means that the iterator returned from `Shuffled::begin()` must contain *all* possible indicies in the container, while `Shuffled::end()` returns a `random_iterator` where `left` is empty.

This wouldn't be a good random iterator if we didn't initially shuffle `left`. To do this you can use `std::iota` and `std::shuffle` with the given random number generator `gen`.

When incrementing the iterator you remove the *last* index in `left`. When dereferencing the iterator you access the *last* index in `left`. It is OK if `operator*()` is $\mathcal{O}(n)$ so you can use `std::next` to add the index to the begin iterator of the container to access the element. Two iterators `a` and `b` are the same if `a.left == b.left`.

There are some tests given in `assignment1.cc`.

Question: What is the purpose of the type aliases `value_type` and `iterator_category` that iterators normally have? Is there anything you *can't* do with the `random_iterator` since it doesn't have them?

Assignment 2

The range-based for-loop and `std::for_each` are both designed to iterate over containers. However, if a container *contains* containers those solutions will only iterate the first "layer".

In this assignment you will create a function template `iterate_leaves` that takes an arbitrary container and a callable object. This function template will iterate through all elements in a nested structure of containers. To demonstrate what this means, here is an example:

```
std::vector<std::set<std::list<int>>> v {
    std::set<std::list<int>> {
        std::list<int>{1, 2},
        std::list<int>{3, 4, 5}
    },
    std::set<std::list<int>> {
        std::list<int>{6, 7},
        std::list<int>{8, 9, 10, 11}
        std::list<int>{12}
    }
};
```

Now, if we call `iterate_leaves(v, fun)` this will call `fun` on each of the `int`s, in order. So we will call `fun` on `1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12` in that order.

Note that `iterate_leaves` must work for any containers nested in any order.

To do this, create a helper function that takes an arbitrary container through the template parameter `T`. The behaviour of this helper function should differ depending on `T`:

1. If the passed in function `fun` can be called on `T` directly, then do so.
2. If `T` is `std::pair`, then call `iterate_leaves` on the `second` field of the passed in `std::pair`.
3. Otherwise, if `T` is a container then call `iterate_leaves` on each element in the container.

Notice that there might be overlap between these cases, so you will have to induce a priority.

There are some testcases given in `assignment2.cc`.

Hint: You can assume that if `T` has an iterator then it is a container.

Hint: Make sure to *forward declare* `iterate_leaves` so you can call it in the helper functions.

Question: How did you induce a priority on the different cases? Explain why it works.