LINKÖPING UNIVERSITY
Department of Computer and Information Science
Software and Systems
Christoffer Holm 2022-01-13

Computer examination in
TDDD38 Advanced Programming in C++

Date 2022-01-13

Time 8-13

Department IDA

Course code TDDD38

Exam code DAT1

Examiner

Klas Arvidsson (klas.arvidsson@liu.se)

Administrator

Anna Grabska Eklund, 28 2362

Teacher on call

Christoffer Holm (christoffer.holm@liu.se)
Will primarily answer exam questions using the student client.
Will only visit the exam rooms for system-related problems.

Allowed Aids (tillåtna hjälpmedel)

An English-* dictionary may be brought to the exam.
No other printed or electronic material are allowed.
The cppreference.com reference is available in the exam system, except for the language section.

Grading

The exam has a total of 25 points.
0-10 for grade U/FX
11-14 for grade 3/C
15-18 for grade 4/B
19-25 for grade 5/A

Special instructions

- All communication with staff during the exam can be done in both English and Swedish.
- Don't log out at any time during the exam, only when you have finished.
- Given files are found in subdirectory `given_files` (write protected). The exam will be available as a `PDF` in this directory at the start of the exam.
- Files for examination must be submitted via the Student Client.
- When using standard library components, such as algorithms and containers, try to chose "best fit" regarding the problem to solve. Avoid unrelated/unnecessary computations and unnecessary data structures.
- C style coding may cause point reduction where C++ alternatives are available.
- Your code should compile. Commented out regions of non-compiling code may still give some points. Resource leaks and undefined behavior is important to fix.
- Questions marked as *Discussion* is meant to be answered textually (`txt` or `PDF`). The answers to these questions must be passed in separately from the code.

Available commands

`e++20` is used to compile with "all" warnings as *errors*.
`w++20` is used to compile with "all" warnings. **Recommended.**
`g++20` is used to compile *without* warnings.
`valgrind --tool=memcheck` is used to find memory leaks.

C++ reference

During the exam you will have *partial* access to `http://www.cppreference.com/` with the chromium browser. You can start the broweser by either running `chromium-browser` in the terminal or choose an appropriate option in the start menu. Do note that everything except cppreference will be unavailable. If you are unable to access a page that should be available (it might have been blocked by mistake) then you can send a message through the exam client. Since it is an experimental feature there might be some quirks.

1. Keeping a schedule of what to do at what times is a common organizational tool. Often we find that we keep multiple schedules: one for work or studies and one for our private life. But in order for schedules to be as efficient as possible we want to view them all at once. [4p]

   In this assignment you will use STL (the standard library) to create a program that takes two schedules and merge them into one. There are example schedules in the files `first.txt` and `second.txt`.

   Each schedule consists of events of the format: `<start hour> <end hour> <description>` where `<start hour>` and `<end hour>` are integers between 0 and 24. They represent at which hour this event starts and ends, respectively.

   `<description>` is a string that is terminated by a newline character.

   An event is represented by the type `Event` (which is defined in `given_files/program1.cc`), and can be read from a stream with `operator>>` and printed to a stream with `operator<<`. The body for `operator>>` and `operator<<` are given in `given_files/program1.cc`, but you will have to declare the function headers yourself.

   Note: due to overloading rules, both `operator<<` and `operator>>` must reside in the namespace `std`.

   There are step-by-step instructions for how this program should be implemented given in `given_files/program1.cc`. Make sure to follow these steps as closely as possible.

   In this assignment you are to use standard algorithms to implement the program described above. Hand-written loops and `std::for_each` will deduce points.

2. **Discussion:** Answer the following questions: [4p]

   a) When overloading `operator<<` what are some things you must consider? Is it an ordinary function? A member function? Is there anything in particular we have to think about when it comes to parameters and return type? Explain clearly what each parameter represents and why they are necessary.

   b) Explain what overload resolution is. Your explanation must describe how overload resolution deals with multiple functions of the same name. In what order does the compiler prioritize candidate functions? Does the data type of parameters matter? Does the data type of return types matter? Does templates parameters matter?

   Note: Both (a) and (b) are worth 2 points each. Your answers should be around 150-500 words long per question (this is not a hard limit). Make sure to make your explanations *clear*. It is important that you communicate your assumptions.

3. Databases are a central part of software development. A database stores the rows of a table where each column is of a specified data type. [4p]

   In this assignment you will implement a very simple database class template, called `database` that takes an arbitrary number of template parameters, each representing the data type of individual columns in the database.

   Each row in the database is associated with a unique integer identifier. This identifier will be used as a shorthand when dealing with individual rows in the database.

   Each row is stored as `std::tuple` objects.

   `database` must have the following functions:

   - `insert`, takes an argument for each column and insert these values into the database. This function assigns a new identifier to this newly inserted row and return that identifier.

     Generation of identifiers is done with a counter. Each insertion will assign the identifier to the row and then increase the counter by 1. The counter starts at 0.

     Note: The counter must NEVER decrease.
   - `get`, takes a row identifier and return the columns for that row. It must return all the columns as an `std::tuple` reference. If the specified row does not exist it must throw an `std::out_of_range` exception.
   - `remove`, takes a row identifier and removes that row from the database. If the row identifier doesn't exist the function doesn't do anything.
   - `filter`, takes a callable object as a *forwarding reference*. This callable object must take two parameters: an `int` (a row identifier) and a `std::tuple` (the columns of a row). The callable object return a `bool`.

     `filter` will return a `std::vector` containing the row identifiers for all rows where calling the callable object return `true`.

     Requirement: The returned `std::vector` should store the identifiers in sorted order.

   `database` must also contain a type alias called `row_type` which is an alias for `std::tuple` instantiated with the column types.

   Hint: Think carefully about which containers you should use to implement the database. It should be possible to find rows based on the identifiers. The implementation of `filter` becomes a bit simpler if you can iterate through the rows in sorted order.

   There are testcases given in `given_files/program3.cc`.

4. Formatting documents can be a hard task. They usually contain different types of elements such as text, lists, tables, images etc. These elements should be printed in a nice format and in the correct order. Some well-known formatting tools for documents are: HTML and CSS, LaTeX, Microsoft Word etc. [5p]

   In this assignment you will use dynamic polymorphism to create a very simple framework for formatting documents in the terminal. This framework will only have 3 different types of elements:

   Label A title followed by some text.

   Grid A list of integers printed in a grid pattern (all grid cells in this grid have the same width).

   List A list of integers printed such that each line looks like this: `- [integer]`

   There must be a base class called `Element` that has a string data member called `name` and a pure-virtual function `print` that takes an `std::ostream` object (this function returns nothing). The `print` function will be called whenever we want to print this element to the specified stream.

   The constructor of `Element` takes a string and assign it to `name`.

   `Label` is a subclass of `Element` that has a string data member called `text` (both `text` and the inherited `name` should be set through the constructor).

   `Label` overrides `print` such that it prints: `<name>: "<text>"`, where `<text>` is the content of the `text` member, and `<name>` is the content of the inherited member `name`.

   `Element` have a subclass called `Collection`. It contains:

   - a vector of `int`s called `items`,
   - a variable called `column_width` (always initialized to `0`),
   - a protected virtual function `print_item` that takes an `std::ostream` object and an `int`. This function will print the passed in `int` to the `ostream`. This function must prepend spaces before the `int` until the total number of characters printed is equal to `column_width` (here `std::setw` from `<iomanip>` might come in handy).
   - A function called `insert` that takes an `int`. This function will insert the `int` to the end of the `items` vector. If the number of digits in the `int` is greater than `column_width`, then `column_width` is set to that number of digits. Do this by converting the passed in `int` to a string with `std::to_string` and compare its size to `column_width`, if the string size is greater, than set `column_width` to that value.

   `Collection` have the same constructor as `Element`.

   `Grid` is a class template that takes an `int` value called `width` as a *template parameter*. `width` represents how many grid cells there are per row in the grid.

   `Grid` overrides the `print` function. An implementation is given in `given_files/program4.cc`.

   Finally there is the `List` class. It behaves the same way as a `Grid` with `width = 1`, with the exception that `List` overrides `print_item`. `List::print_item` should print the same thing as `Collection::print_item`, but it should start with "`- [`" and end with "`]`".

   There is a partial main program given in `given_files/program4.cc`. Make sure to fix the parts inside the comments.

   Note: `Grid` must be a class template and `width` must be a template parameter to `Grid`.

5. **Discussion:** Discuss the advantages and disadvantages of the class hierarchy in the previous assignment. Focus on *readability*, *scalability* and/or *usability*: i.e. how easy is the code to understand for the reader, how easy is the code to change and how easy are the classes to use. Your answer should be around 200-1000 words. Give code examples. You don't have to cover all three aspects, but make sure to bring up at least one advantage and one disadvantage. [3p]

   Hint: You could compare this to other ways to achieve the same thing.

6. Passing callable objects (functions, lambdas and function objects) to functions is a very common pattern in many programming languages (including C++). Usually this pattern occurs in the interface of algorithms. [5p]

   In this assignment you will create a simple algorithm called `enumerate`. It behaves similarly to `std::for_each` but with some key differences.

   Instead of taking iterators to the container, `enumerate` will take the container as a constant reference. It will (just like `std::for_each`) take a callable object and apply it on each element in the container.

   `enumerate` will therefore take exactly two parameters; a container and a callable object. It does not return anything.

   `enumerate` will have different implementations based on the number of parameters which can be passed to the callable object. There are 3 different implementations:

   - If the callable object takes one parameter then `enumerate` will iterate through the container (in a general way) and pass each element to the callable object.
   - If the callable object takes two parameters then `enumerate` will iterate through the container and pass each element AND that elements index to the callable object. This means that you will have to keep track of the index of each element, which can be done with a simple counter.
   - Lastly, if the callable object takes three parameters then `enumerate` will also pass the containers size to the callable object. So for each element it will pass the element, its index and the containers total size.

   Some callable objects can be called with a variable amount of parameters. This occurs when there are default parameters to the callable object. Say for example that we pass a callable object that can take 1, 2 or 3 parameters. In that case all the implementations of `enumerate` will be valid, which would lead to ambiguous calls. You need to fix this issue by inducing a priority order on the implementations:

   1. If the callable object *can* be called with three parameters, then `enumerate` should always do so.
   2. If the callable object can be called with two parameters (but not three), then corresponding implementation of `enumerate` should be called.
   3. In all other cases `enumerate` should call the implementation where the callable object takes one parameter.

   There are testcases given in `given_files/program6.cc`.

   Note: It should be possible to pass any container to `enumerate`, including C-arrays and `std::map`. However, the implementation taking three parameters should only be possible to call if the container have a `size` function. This means that for example C-arrays can not be used with all three implementations.

   Hint: Create a helper function template with one overload for each implementation. Use SFINAE to test whether the callable object is callable with 1, 2 or 3 parameters. It might be helpful to add extra parameters to the helper functions in order to induce the priority order.

   Hint: `enumerate` should take (at least) two template parameters.