LINKÖPING UNIVERSITY
Department of Computer and Information Science
Software and Systems
Christoffer Holm 2019-01-11

Computer examination in

TDDD38 Advanced Programming in C++

Date 2019-01-11

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.

Available commands

`e++17` is used to compile with "all" warnings as *errors*.
`w++17` is used to compile with "all" warnings. **Recommended.**
`g++17` is used to compile *without* warnings.
`valgrind --tool=memcheck` is used to find memory leaks.

C++ reference

During the exam you will have *experimental* 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.

Theory questions

1. Answers may be given in either Swedish or English. Write all your answers in one text file and submit as assignment 1.

   (a) What is *placement new*? [1p]

   (b) Explain why the program below prints `1`. [1p]

   ```
   #include <iostream>
   int main()
   {
     int f();
     std::cout << f << std::endl;
   }
   ```

   (c) Give an example of a *forwarding reference*. [1p]

   (d) What does the *as if* rule state? [1p]

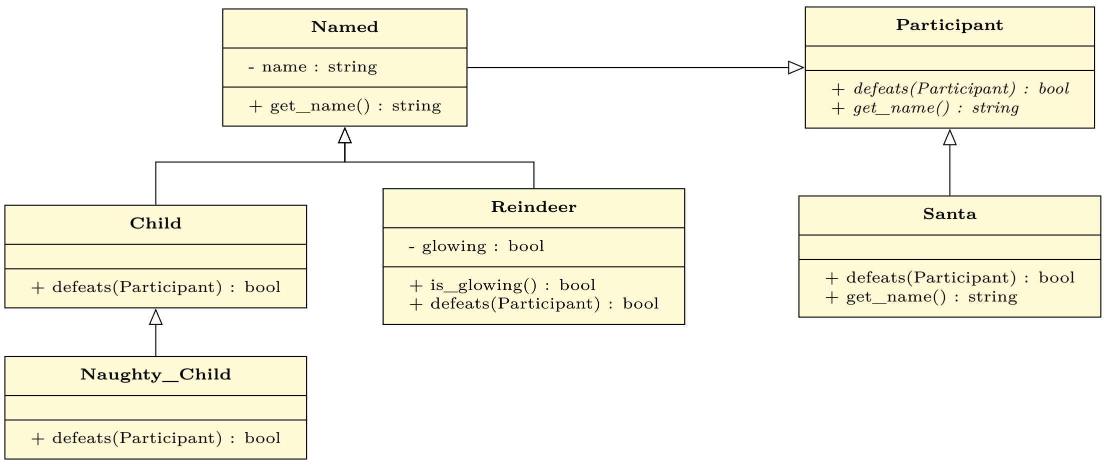
   (e) What is an inline namespace? [1p]

Practical questions

2. Santa and his reindeers have been challenged to a post-christmas snowball fight by all the children. Everyone in the fight has varying skills and targets for their snowballs so it is quite a mess to determine who will hit who. [5p]

   Your job in this assignment is to write an object-oriented program that determines who can hit who given a set of participants.

   You are to implement the following UML diagram:



   Santa is a masterful snowball fighter (he has been practicing for many christmases) so none of the children are capable of hitting him. Santa does not have the heart to throw at the nice children so he makes sure to only throw snowballs at the naughty children.

   The reindeers find this unfair and decide to team up against Santa which leaves them open to the children. However he children have a hard time spotting the reindeers that do not have glowing noses, so it is only Rudolf that gets hit by snowballs.

   The naughty children will also throw snowballs at the nice children; they are naughty for a reason after all. Summarized this gives us the following requirements:

   - `Santa::defeats` should return `true` if the passed in participant is of type `Naughty_Child`.
   - `Child::defeats` should only return `true` if the passed in participant is a `Reindeer` with `glowing` set to `true` (`glowing` should be accessible with `Reindeer::is_glowing()`).
   - `Naughty_Child::defeats` should behave the same way as `Child::defeats`, but should also return `true` if the argument is *exactly* of type `Child`.
   - `Reindeer::defeats` should return `true` if the passed in participant is of type `Santa`.
   - `Named::get_name()` should return the value of `Named::name`.
   - `Santa::get_name()` should return `"Santa"`.
   - Each class should have a constructor that takes appropriate values to initialize all data members.

   It should *not* be possible to copy any of the objects. The base class `Participant` should not contain any logic.

   There is a partial main program given in `given_files/program2.cc`. You should modify the given code so that it works correctly with your implementation of the class hierarchy.

3. Santa has began a process to modernise his operations until next christmas. Part of this process is to automatically calculate whether or not a child is naughty or nice by analyzing their wishlists. [5p]

   In this assignment you are to implement an algorithm that calculate a naughty/nice score for a given `vector<string>` (a *wishlist*). You are to use the standard library exclusively - no handwritten loops allowed. Limit your use of `std::for_each` as much as possible.

   To calculate the score of a wishlist, Santa first calculates the score for individual entries and then sum the scores of each individual entry.

   To calculate the score of an entry follow this algorithm:

   1. If an entry is empty, then the score is `0`.
   2. The entry should be transformed into all *lowercase* characters.
   3. The difference between each consecutive character should be calculated and stored in a `vector<int>` called `result`.

      Example: Given the entry `"book"` we get the vector `{13, 0, -4}` since:

      - `'o' - 'b'== 13`
      - `'o' - 'o'== 0`
      - `'k' - 'o'== -4`

      Note: for each consecutive pair of characters calculate the second minus the first.
   4. Create a `vector<double>` called `weights` with the same size as `result`
   5. Fill `weights` with the fractions `1 / weights.size()`, `2 / weights.size()` etc. up to `1`.

      Example: given the `result` vector from above, the `weights` vectors should become `{0.333333, 0.666666, 1}`.
   6. Lastly multiply the elements from `result` and `weights` pairwise and the take the sum of each product. The result should be converted to an `int`.

      Example: `"book"` will result in the product `13*0.333333 + 0*0.666666 + (-4)*1` which should result in a score of `0` after truncation.

   To calculate the final result of a given wishlist apply the algorithm above to each entry and then sum all the scores together.

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

4. The north pole operation relies heavily on lists of various sorts. The elves are tired of having to write these lists manually each day so they have hired you to make their list creation process easier. [5p]

   In this assignment you are going to write an *OutputIterator* (*LegacyOutputIterator* on cppreference) that writes a list point before printing the actual data to some stream. The user should be able to specify what type of list format will be printed through a policy class.

   Create a class template `print_list_iterator` that takes a policy class as a template parameter. It must also contain the five common member types of iterators (`value_type`, `iterator_category`, `reference`, `pointer` and `difference_type`). All of these can be an alias for `void` except `iterator_category` which should be `std::output_iterator_tag`.

   `print_list_iterator` should have one constructor that takes an `ostream` reference and a `Policy` object, both of these should be stored as data members.

   Whenever the `print_list_iterator` is written to (assigned to) it should first call the function `Policy::print_point` which is defined inside the policy, and then it should print the data to the stream followed by a newline. Note that it should be possible to write *any* type of data that is printable to streams.

   Whenever the iterator is incremented it should call the function `Policy::increment` that is defined in the policy class.

   For this assignment you must implement two policy classes:

   enumerate Represents an enumeration list (a list that numbers each entry). The constructor should take two optional arguments; `start` and `step` (both should have default value `1`). `start` defines at what number the list starts counting and `step` defines how many steps there should be between each numbered entry.

   This policy should keep track of the current entry number. Each call to `increment` should add `step` to the current entry number.

   `print_point` should print the current entry number followed by a period and a space; it should *not* modify anything in the class.

   description Represents a description list (a list where each list point is specified by the user). The user specifies the available list points inside a container and then passes this container to the `description` class.

   This should be a template class where the container type should be specified as a template argument. The constructor only takes one argument, a reference to some container.

   The class must keep track of the current list point in the container. `increment` should step the current list point to be the next element in the container. If the end is reached, the list points should start over from the beginning of the container. Finally, `print_point` should print the current list point.

   You must overload the operators required on *OutputIterator* according to `https://en.cppreference.com/w/cpp/named_req/OutputIterator`.

   There are testcases given in `given_files/program4.cc`. You should not have to modify these testcases at all.

5. Santa and his elves recieves a lot of data from children around the world. Their current software for handling this data is hard to use. Children send a large variety of data, and as it stands right now the elves must implement a new `read` function each time some new combination of data is sent, which can be quite time consuming. [5p]

   Your job in this assignment is to make their `read` functionality more general using templates.

   You are to implement a template function `read` that takes an `std::istream&` and a reference to an object `t` of arbitrary type `T`.

   The behaviour of `read` varies depending on `T`:

   1. If `T` is a `std::string` then the data should be read from the stream into `t` using `std::getline`;
   2. If `T` is a `std::tuple` with arbitrarily many fields then `read` should read a value into each field in the tuple `t` recursively using `read`. You may assume that each field in the tuple has a unique type.
   3. If there is an overload of `operator>>` that takes `T`, then the data should be read with that operator into `t`.
   4. Otherwise, if none of the above holds `read` should assign the default value of `T` to `t`.

   Partial credits will be given if cases 1, 2, 4 or cases 1, 3, 4 are correct. There are testcases given in `given_files/program5.cc`.

   Hint: If you are having trouble that the compiler choses the wrong overload or if you have problems with ambiguity, try to add an extra argument to your helper functions to control the overload resolution.