LINKÖPING UNIVERSITY
Department of Computer and Information Science
Software and Systems
Christoffer Holm 2019-04-25

Computer examination in

TDDD38 Advanced Programming in C++

Date 2019-04-25

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 *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.

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) Given [1p]

   ```
   int&& foo(int&& x)
   {
     return x;
   }
   ```

   what is the *value category* of the expression `foo(5)`? Explain why.

   (b) Give a code example of a *template-template parameter*. [1p]

   (c) When talking about inheritance and polymorphism, what is the *diamond problem*? [1p]

   (d) What is *ADL* (argument dependent lookup)? [1p]

   (e) What is meant with the `constexpr` keyword? [1p]

Practical questions

2. A customer has contacted you because they want a program that takes an arbitrary text and replaces each word with a synonym if possible. However, there is a slight problem. The customer clearly doesn't know what a synonym is. They believe that two words are synonyms if they are spelled exactly the same but for one character difference. In other words; they think that two words are synonyms if they differ at exactly one letter in their spelling. After a short discussion you realize that the customer actually want what they ask for. [5p]

   In this assignment you are to use standard algorithms to implement the program described above. No Hand-written loops are allowed. Usage of `for_each` will result in point deductions.

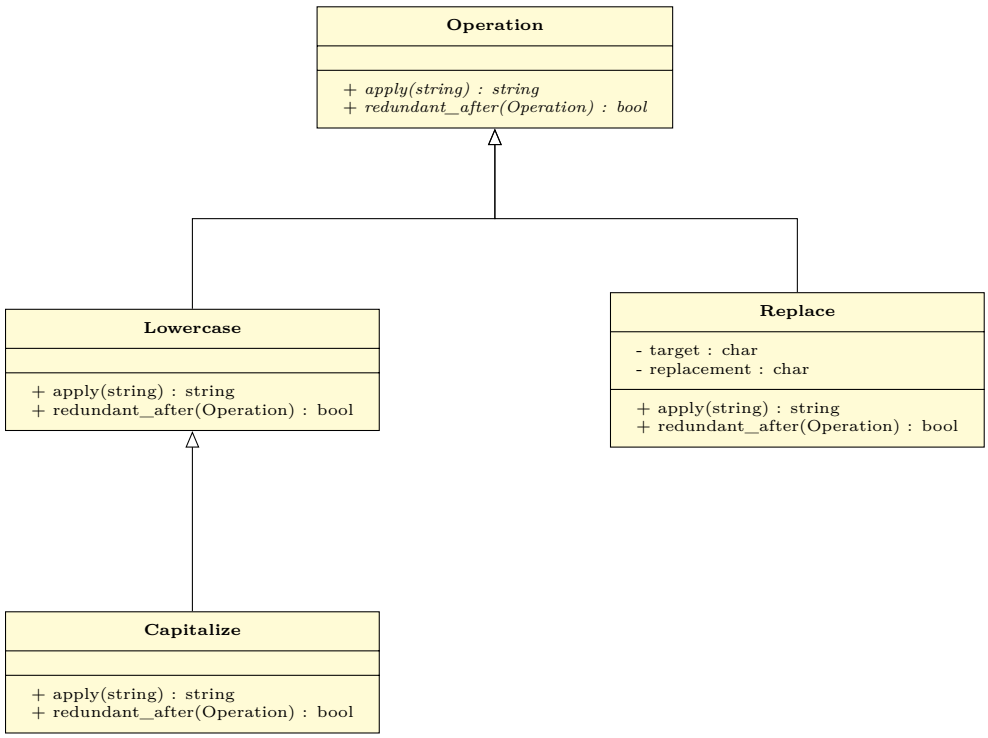
   A skeleton for the program is given in `given_files/program2.cc`. The file contains a list of "synonyms" and further instructions on how the program should be implemented.

   Here are some examples you can try:

   - "`wash the fish`" should give "`cash the dish`"
   - "`I am leasing this ring`" should give "`I am leading this king`"
   - "`make films when happy`" should give "`jake files then happy`"

3. In this assignment you are to implement a polymorphic class hierarchy that represents a sequence of string transformations. Each class in the hierarchy contains logic for one specific operation on a given string. Some of these operations are redundant if they occur after another specific operation, so there should be functionality to check whether an operation is redundant given the previous operation (the one that comes before it). [5p]

   The following class hierarchy should be implemented:



There are four classes in total:

`Operation` base class for the various operations

`Lowercase` will transform each letter in the given string into lowercase letters. It is redundant if it occurs directly after a Lowercase operation (note that it is NOT redundant if it occurs after a Capitalize operation).

`Capitalize` will first perform the operation of the `Lowercase` class, and after that it will capitalize (turn it into an upper letter) the first character in the string. It is redundant if it occurs directly after another Capitalize operation.

`Replace` has a constructor that takes two parameters; `target` and `replacement`. Will replace each occurrence of the character `target` with the character `replacement` in the given string. It is redundant if it comes directly after another Replace operation which shares the same `target` character.

Hint: The functions `tolower`, `toupper` and `std::replace` might be useful.

It should *not* be possible to copy any of these classes and `Operation` should not contain any logic.

Each class should have exactly two functions:

`apply` Will take a string and apply the corresponding operation on it and return the resulting string.

`redundant_after` Takes a reference to an `Operation` object `op` and returns `false` if this object is redundant given that it occurs directly after `op`, and `true` otherwise.

There is a partial main program given in `given_files/program3.cc`. You should modify the given code so that it works correctly with your implementation of the class hierarchy.

4. In this assignment you are to create a function template `get_size` that calculates the size of a object and a variadic function template `total_size` that calculates the total sum of all parameters size. [5p]

   The function template `get_size` should take a type `T` as a template parameter and take an object of type `T` as a parameter.

   - If `T` is a container (i.e. it has iterators) then `get_size` should call `get_size` on each element and return the sum.
   - Otherwise, `get_size` should simply return the result of `sizeof` on the object (for example; `return sizeof(t);`).

   The function template `total_size` should take an arbitrary count of parameters with arbitrary types. `total_size` should apply `get_size` on each parameter and should return the sum of all the sizes.

   There are testcases given in `given_files/program4.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.

5. In this assignment you are to implement an *OutputIterator* (called *LegacyOutputIterator* on cppreference) `sorted_insert_iterator` that insert elements in sorted order into a container. Sorting should be done according to a comparator given by the user. [5p]

   `sorted_insert_iterator` should be a class template with two template parameters;

   `Container` an arbitrary container that is sorted.

   `Compare` a function object that takes two parameters, `left` and `right` which has type `Container::value_type` and returns `true` if `left` should be inserted *before* `right` in the container and `false` otherwise.

   `Compare` should have `std::less` as default type.

   The class template `sorted_insert_iterator` must 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 for `iterator_category` which should be `std::output_iterator_tag`.

   `sorted_insert_iterator` should have one constructor that takes a reference to a container of the type `Container` and a function object of type `Compare`, both of which should be stored as data members.

   You must overload all the operators required by *OutputIterator*, which can be found here `https://en.cppreference.com/w/cpp/named_req/OutputIterator`.

   You should also create a function template `sorted_inserter` that takes a container and a comparator, and returns a `sorted_insert_iterator` based on those parameters. Note that the comparator parameter should be optional (i.e. should be default constructed if no argument is given by the user).

   There are testcases given in `given_files/program5.cc`. You should not have to modify these testcases at all.