LINKÖPING UNIVERSITY
Department of Computer and Information Science
Software and Systems
Eric Elfving 2018-05-31

Computer examination in

TDDD38 Advanced Programming in C++

Date 2018-05-31

Time 14-19

Department IDA

Course code TDDD38

Exam code DAT1

Examiner

Eric Elfving (eric.elfving@liu.se)

Administrator

Anna Grabska Eklund, 28 2362

Teacher on call

Eric Elfving (013-28 2419)
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.

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.

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 wrong in this code and how is it fixed: [1p]

   ```
   template <typename T>
   T identity(T&& value)
   {
       return std::move(value);
   }
   ```

   (b) What is the *as-if* rule? [1p]

   (c) What happens when an exception is thrown from a function declared as `noexcept`? [1p]

   (d) What is wrong here: [1p]

   ```
   struct A
   {
     A(int value) : y{value}, x{y+1} {}
     int x, y;
   };
   ```

   (e) What is the difference between a *static* and a *dynamic* type? [1p]

Practical questions

2. Quite often programmers find themselves needing to iterate over multiple containers at once. As it stands right now there is no syntax in C++ that will help us achieve this in a readable way. We have to rely on index based loops, multiple iterators etc. This can lead to hard-to-read code. [5p]

   In other programming languages (such as Python, Javascript, Haskell etc.) there is an operation called *zip* which helps the programmer solve this kind of problem. Zip takes two containers (sequences of values) and merge them into a new container (sequence of values) which contains pairs of these values. For example if we zip the lists `1,2,3` and `4,5,6` we get the new list `(1,4),(2,5),(3,6)`.

   In this assignment you are to create a template-class called `zip` which takes two template-parameters `left_container` and `right_container` representing two arbitrary container types. The class `zip` must have the member functions `begin` and `end`, both of which must return iterators of type `zip_iterator`.

   You are to implement the class `zip_iterator`. `zip_iterator` must be a *ForwardIterator* with both post- and prefix increment operators, `==`, `!=` and a dereference operator which returns `std::pair<A&, B&>` where `A` is the value type of `left_container` and `B` is the value type of `right_container`. Note that `operator->` is not required in this assignment.

   If the containers differ in size then `zip` should stop as soon as the smallest container reaches the end.

   If everything is implemented correctly you should be able to use structured-bindings and range based for-loops to iterate through two containers. It would look like this:

```
for (auto [left, right] : zip(v1, v2)) {
  //...
}
```

   A test program is given in `given_files/program2.cc`.

3. When a container is sorted there are ways to make searching through said container a lot faster for example by using binary search. [5p]

   In this assignment you are going to implement a function-template `sorted_search` which takes three parameters; a container, a value to search for in the container and a policy representing the strategy used when searching. The function `sorted_search` must use the policy to search for the given value and must return an iterator pointing to the found element. If no element was found then it must return the end iterator of the given container.

   In what order and how the elements are compared is determined by the policy. Note that the container is always assumed to be sorted. The policy is a template-class which contains the following member functions:

   first Takes a container reference and returns an iterator pointing to where `sorted_search` should begin searching.

   done Takes a container reference and an iterator. Returns `true` if the search should be aborted (if for example the iterator has reached the end of the container) and `false` otherwise.

   next Takes an iterator as parameter and returns an iterator pointing to the next element in the search sequence.

   compare Takes an iterator and a value. Returns `true` if the value is equal to the element pointed to by the iterator, otherwise return `false`.

   The function `sorted_search` must use the policy member functions while searching.

   You are to implement two different policies:

   Linear searches through the container in sequence, from the first element until either the value has been found or the end of the container is reached.

   Binary uses binary search to find the value. This policy should keep two data members, `left` and `right` which determines the current search interval. When `left` and `right` are equal, the search is done.

   If no policy is given when calling `sorted_search` the default should be `Linear`.

   There are testcases and sample implementations of both the linear- and binary search algorithms in `given_files/program3.cc`.

   Hint: `Binary::compare` should do most of the work for the binary search policy.

4. A recurring problem in software development is handling the communication between different modules in the program. Modules often communicate by calling functions and reading the return values. Sometimes the calling module must perform some action depending on the value returned from another module. The naïve way to do this is with some kind of `if` or `switch` statement. However, this does not scale at all as the program gets larger. A more sophisticated way to solve this problem is by using events. [5p]

   In this assignment you are to create a class hierarchy which represent events sent from one module to another. Each event represent an action that the calling module should take. To get two-way communication between the modules, each calling module should supply the event with a context. In a real-world application the context would contain many members, but for the purposes of this assignment they are kept as simple as possible.

   There are two context classes:

   Context represents a base context which simply keeps track of an identifier. This identifier must be unique for each instance and should be generated automatically in the default constructor. A function `reset` must exist that generates a new identifier such that no other instance of a context class has the same identifier. The `print` function should print the type (`Context` in this case), and the id to an arbitrary output stream (given as an argument). The context should be printed according to this format: `Context [id]`.

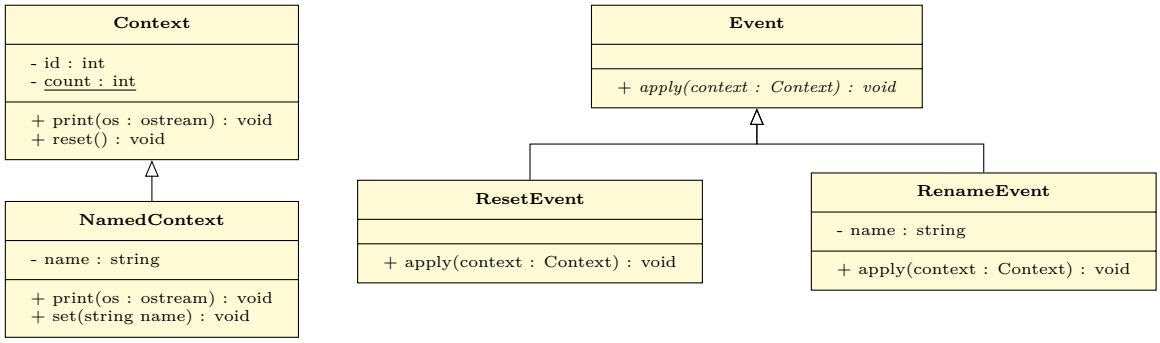
   NamedContext is a context which has a string representing a name for the context. This name is passed into the class through a suitable constructor. There should also be a setter for the name. The `print` function must print the type, the id and the name to an arbitrary output stream according to this format: `Named Context [id] "Name"`.

   `Event` is an abstract class with two subclasses:

   ResetEvent Represents an event which generates a new identifier for the given context. The `apply` function must reset the supplied context (i.e. call `reset`) and print the context before and after the reset.

   RenameEvent Represents an event which changes the name of a `NamedContext` instance. This event must store which name the context should be changed to. The `apply` function should print the context before and after the renaming (regardless of what type of context it is). If the supplied context is a `NamedContext` then it must change the name of the context to the one stored in the event.

   It should not be possible to copy any of the classes.



   There is a skeleton for a main-program given in `given_files/program4.cc`. Modify this main-program so that it works for your code. It is a requirement that the `apply_events` function only operates on events marked as `const`.

5. In this assignment you are to implement functions that operate on log files. A log file is represented as a vector of pairs. Each pair contains a timestamp (`int`) and a message (`string`). Three functions should be implemented: [5p]

   compress A reoccurring message in a log file is more often than not unnecessary. This function should take a log file and remove all consecutive duplicates of messages in the log file. The duplicated message with the smallest timestamp should be kept, but all other consecutive duplicates should be removed. Example:

   ```
   Before compress:    After compress:
   1 Duplicate         1 Duplicate
   2 Duplicate         3 Not a duplicate
   3 Not a duplicate   4 Duplicate
   4 Duplicate
   5 Duplicate
   ```

   diff Takes two log files and returns a pair of iterators to the first pair of entries which differs between the log files. Example:

   ```
   Log 1:              Log 2:
   1 This              1 This
   2 is a              2 is another
   3 log file          3 log file
   ```

   Log 1 and Log 2 first differs at the second entry.

   merge Takes two log files and merges them into a single log file, sorted with respect to the timestamp and compressed (here you can use the `compress` function). Example:

   ```
   Log 1:              Log 2:
   1 This              1 This
   2 is a              3 is another
   4 log file          4 log file

   Log 1 and Log 2 merged:

   1 This
   2 is a
   3 is another
   4 log file
   ```

   The focus in this assignment is the usage of STL, a full solution shouldn't require any standard iteration statements. Make sure to use suitable algorithms to solve each problem. A testprogram is given in `given_files/program5.cc`.

   In your main program you must sort all the log files with respect to their timestamps before you call the functions, since the log entries might be added in an incorrect order. Do not sort a log file more than once.