LINKÖPING UNIVERSITY
Department of Computer and Information Science
Software and Systems
Christoffer Holm 2019-08-21

Computer examination in

TDDD38 Advanced Programming in C++

Date 2019-08-21

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, specifically you will *not* have access to anything in the language section of cppreference. You can start the browser 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.

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) Give an example of a *fold-expression*. [1p]

   (b) What does the `override` keyword do? [1p]

   (c) Given [1p]

   ```
   struct A
   {
     void foo() &&;
   };
   ```

   give an example of a *valid* call to `A::foo`.

   (d) What is the difference between a *type template parameter* and a *nontype template parameter*? [1p]

   (e) What is *slicing*? [1p]

Practical questions

2. You have gotten your hands on quite a lot of books recently. You have decided to let your friends and family read these books so they can give you recommendations. In this assignment you are to simulate which books your friends and family might recommend. Among your friends and family there are four types of people: [5p]

   - those who likes everything (called `Reader`)
   - academics who only like non-fiction (called `Academic`)
   - children who only want to read books for children (called `Child`)
   - detectives who like fiction but does not want to read crime fiction since they do not want to bring home their work (called `Detective`)

   You are to create a class hierarchy of the various types of books and people in this scenario. The following classes must be implemented:

   Book abstract base class of all book types. This class contains the title of the book and a pure virtual function `for_children`.

   Non_Fiction derived class of `Book` which overrides `for_children` to always return `false`.

   Fiction derived class of `Book` that stores a `bool` variable that determines whether or not this is a childrens book. This class must override `for_children` such that it returns the value of the `bool` variable.

   Crime_Fiction is exactly the same as `Fiction`, but as we all know; crime dramas will always become best sellers, so during construction this class should append the string `" (best seller)"` to the end of the title.

   Reader base class for the different kinds of people among your friends and family. Contains the persons name, a virtual function `likes` that takes an object of type `Book` and returns `true` as well as a function `read` which takes an object of type `Book` and prints a message whether or not this person recommends the passed in book. The message should have the following format: `<name of person> recommends reading <title of book>` if the person likes the book and `<name of person> does not recommend reading <title of book>` if they did not like the book.

   Academic derived class of `Reader` that overrides `likes` such that it returns `true` if the passed in `Book` is of type `Non_Fiction` and `false` otherwise.

   Child derived class of `Child` that override `likes` such that it returns `true` only if `for_children` returns `true` for the given `Book`-parameter.

   Detective derived class of `Reader` that override `likes` if the `Book`-parameter is *exactly* of type `Fiction`. Recall; the detective does not want to bring home their work, so no crime fiction for the detective.

   Note: It should not be possible to copy any of these classes. There is a skeleton for a test program given in `given_files/program2.cc`.

3. In this assignment you are to implement a *ForwardIterator* (called *LegacyForwardIterator* on cppreference) `append_iterator` that iterates over two containers in sequence as if the second container has been appended to the end of the first container. [5p]

   You are to implement a class `append` that stores references to two containers of the exact same (arbitrary) type (for example two `std::vector<int>` or two `std::set<double>` etc.) that provides an iterator range with iterators of type `append_iterator` through the member functions `begin()` and `end()`.

   `append_iterator` takes one template-parameter; the iterator type of a container (the same type as the iterator ranges). `append_iterator` should fulfill all requirements of the *ForwardIterator* and it should store the begin- and end-iterator of two containers. The begin-iterator of the first container will act as the current element. Note that `operator->` is not required in this assignment.

   Note: The constructor of `append_iterator` should only be callable by the `append` class.

   Dereferencing an `append_iterator` should simply dereference the current element. The increment operator should step the current element and if the current element has reached the end of the first container it should immediately proceed to the first element in the second container. The comparison operators should compare all the iterators stored in both `append_iterator`s.

   Example: Suppose we have two containers `{1,2,3}` and `{4,5,6}` then the `append_iterator` should traverse through these two containers as such: `1 2 3 4 5 6`, i.e. it should act as if the two containers have been appended together.

   Note: It is **not** ok to create a new container inside the `append` class.

   Hint: To handle the case whenever the first container is empty, you might have to add some logic in the constructor of `append_iterator`.

   The end-iterator of type `append_iterator` should be represented such that the current element is equal to the end-iterator of the second container. I.e. the other three iterators should be the same as they are for the begin-iterator.

   If everything is implemented correctly, then the following should output `1 2 3`:

```
vector<int> v1 {};
vector<int> v2 {1,2,3};
for (auto i : append(v1, v2))
{
  cout << i << " ";
}
```

   A test program is given in `given_files/program3.cc`. Note that you must compile with C++17 for this to work correctly.

4. A customer has contacted you because they want a program that takes an arbitrary text and replaces each word with a synonym if possible. This time around[1] the customer actually knows what a synonym is. The customer has given you a file `given_files/SYNONYMS` that contains a list of synonyms. Each line in the file is of the following format: `<word to be replaced>=<word to replace with>` where each occurance of the word on the left-hand-side is to be replaced with the right-hand-side word. [5p]

   Note: you can assume that no whitespace characters (such as space) occur in this file. You are **not** allowed to use `std::getline` to read the lines.

   Your program should do the following:

   1. Open the file `SYNONYMS` for reading.
   2. read each line in `SYNONYMS` and separate the line at `=`, inserting the resulting word pair into an `std::map` called `synonyms` (i.e. the key should be the left-hand-side word and the value should be the right-hand-side word)
   3. read a text word-by-word from `cin` into a `std::vector` called `words`. If a word has an entry in `synonyms` the word should be replaced with the corresponding value in `synonyms`, otherwise leave the word as-is.
   4. Print all words in `words` separated by a space.

   In this assignment you are to use standard algorithms to implement the program described above. No hand-written loops are allowed, nor is `std::for_each`. There is a skeleton and some testcases given in `given_files/program4.cc`.

[1] There was a similar assignment in the exam given 2019-04-25

5. *Binary search* is a way to search for values in a sorted container. It is usually faster than *Linear search* since binary search utilizes that the container is sorted while linear search does not. However, binary search is more limited than linear search. Binary search can only be used with containers that have *random access* (i.e. containers with *RandomAccessIterator*). [5p]

   In this assignment you will create a function `search` that finds the position of a specified value inside a **sorted** container. This function must use binary search if the container has *RandomAccessIterator*s and linear search otherwise.

   You are to implement a function template `search` that takes two template-parameters `It` and `T` representing an iterator type and a value type respectively. It should take three parameters; an iterator range and a value to search for. This function should find the passed in value inside the iterator range and return an iterator to the found element. If no element is found then the end-iterator should be returned instead. If `It` is an *RandomAccessIterator* then binary search must be used, otherwise linear search must be used.

   There are two partially implemented functions `binary_search` and `linear_search` and some testcases given in `given_files/program5.cc`. You may rewrite these to work with iterators and use them in your implementation of `search`.

   Hint: You can use that iterators contains a type-alias `iterator_category` to detect if an iterator is *RandomAccess*.