LINKÖPING UNIVERSITY
Department of Computer and Information Science
Software and Systems
Christoffer Holm 2025-03-20

Computer examination in
TDDD38 Advanced Programming in C++

Date 2025-03-20

Time 8-13

Department IDA

Course code TDDD38

Exam code DAT2

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 `~/Desktop/given_files` (write protected). The exam will be available as a `PDF` in this directory at the start of the exam.
- Files you want assessed 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/` as well as a local mirror of it with a working search function, but only through the desktop icon "Web access". Do note that not everything on cppreference will be available (in particular the pages under the "Language" section will be blocked). 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.

1. The *Hamming distance* is a way to measure the *distance* between two strings. In particular it is defined as the number of *positions* in both strings that contain *different* characters. If one string is longer than the other then each additional character will add one to the total distance. [5p]

   Example: Suppose we have the strings `"crate"` and `"bribe"`. Here we see that the first, the third and the fourth positions differ, which gives us a Hamming distance of 3.

   Example: Take the strings `"pass"` and `"taster"`. We see that the two strings differ at the first and the fourth positions. Additionally, there are two extra characters in `"taster"` (i.e. the last two), which gives us a total Hamming distance of 4.

   In this assignment you will implement a program that takes a word from a user and finds the five *closest* words found in `wordlist.txt` with respect to Hamming distance.

   Implement the following steps:

   1. Implement the function `hamming_distance()` which takes two strings and returns the Hamming distance between them.

      Hint: To calculate the Hamming distance between two strings of equal length you can view the operation as an *inner product* where the product/multiplication operation is replaced with the inequality operator.
   2. Read all words in `wordlist.txt` into an appropriate container called `wordlist`.
   3. Read a word from `std::cin` into the variable `word`.
   4. Find the top five words in `wordlist` with the smallest Hamming distance to `word`. If there are multiple words with the same Hamming distance then their internal order doesn't matter.
   5. Print the top five words to `std::cout`, one word per line.

   Some example outputs are given in `given_files/program1.cc`. Note that your output may differ slightly since we do not care about the internal ordering between words that have the same Hamming distance from `word`.

   Requirement: In this assignment you may not use loops, `std::for_each`, `std::transform` or recursion. You must use STL algorithms and containers.

2. **Discussion:** What is pointer (or iterator) invalidation in the context of containers? Give an example of when it can occur. Explain *why* this might cause issues. Are there any containers that are preferred if we want to avoid pointer invalidation? [2p]

3. **If you passed the midterm test then skip this assignment (you get full points).** [4p]

   In this assignment you will write a type trait called `Reverse` that takes one template parameter pack and produces a new template parameter pack which is in the *reverse* order of the original pack. The result should be given as a type alias named `type`.

   In `given_files/program3.cc` the helper struct template `Pack` is given. `Pack` will be used to represent template parameter packs.

   Requirement: You must use partial template specializations in your solution.

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

   Hint: One way to implement this is to make a helper struct template called `Helper` that takes two template parameters `T` and `U` and then implement the following specializations:

   - `Helper<Pack<T, Ts...>, Pack<Us...>>` which moves `T` from the beginning of the first pack to the beginning of the second pack and then recursively use itself again.
   - `Helper<Pack<>, Pack<Us...>>` which sets `type` to `Pack<Us...>`.

   The initial case for this helper would be that the first template parameter is of the form `Pack<Ts...>` and the second template parameter is `Pack<>`.

4. **Discussion:** 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? How does implicit type conversion plays into it? [3p]

5. Sorting is a staple in computer science. It is without a doubt something commonly taught and discussed. The STL library provides the ever useful `std::sort()` for all our sorting needs... or does it? [4p]

   Notice that `std::sort()` requires the iterators of the container to be *random access*, which means it doesn't work for all containers. But today that changes: in this assignment you will provide a sorting function that "works" on most STL containers (with the exception of container adaptors), using a very liberal definition of "works".

   You must implement a function template `sort()` that takes two type template parameters: `Container` which represents an arbitrary STL container (including C-arrays), and `Comp` which represents the type of a *callable object* used as a comparator for the sorting operation. Note that `Comp` should have `std::less` as its default type (i.e. if the type cannot be deduced).

   Hint: `std::less` takes the *value type* of `Container` as its template parameter.

   Additionally `sort()` takes two function parameters: `container` which is an lvalue reference to a `Container` object, and `comp` which is an object of type `Comp`. Note that if the `comp` parameter is omitted by the caller, then it should be set to a default-initialized `Comp` object.

   The `sort()` function should return nothing.

   There are in partiular two types of STL containers that cannot be sorted using `std::sort()`:

   - Containers that have a member function called `sort()` (which takes a callable comparator object as an optional parameter).
   - Associative containers (i.e. `std::map`, `std::set` and all their variants). The main reason these cannot be sorted is because the order is not possible to change by the user. Therefore we will count these as "already sorted", meaning nothing is done for them (however, it should still be *possible* to call `sort()` on them).

   Use SFINAE (or C++20 concepts) to implement `sort()` as described above. There are testcases given in `given_files/program5.cc` which should work without modifications.

   Hint: To implement `sort()` you need to handle these three cases (appearing in priority order), preferably by making three overloads of a helper function:

   1. If `Container` is an associative container, then do nothing. You can detect whether it is an associative container by checking for the existence of the type `Container::key_type`.
   2. If the container has a member function called `sort()` which takes `comp` as a parameter, then use that to sort the container.
   3. Otherwise, use `std::sort()`. Notice that you can, and must, make this case work for C-arrays as well.

   Note: For STL containers there is no overlap between any of these cases, but you should still induce the priority defined above to increase the likelyhood that it works for custom containers as well.

6. **Discussion:** What is *placement new*? How is it different from `new`? When should it be used? [2p]

7. During the course we have discussed sum types and how they can be used to somewhat emulate dynamic typing. However, one of the drawbacks we found was that we always have to bring our dynamically typed objects into the statically typed domain once again if we want to perform any operations. [5p]

   In this assigmment we will explore how one could go about designing a class that more closely mimics languages such as Python or Javascript.

   Create a class called `Variable` which represents a variable that can store numbers (`double`), text (`std::string`) and lists (`std::vector<Variable>`). The class only has one data member: a `std::variant` called `value`.

   Note that all exceptions must be of the exception type `Variable_Error` which is given in `given_files/program7.cc`. The `Variable` class must implement the following functions and operators:

   - A constructor for each of the three types that a variable can contain. These should simply set the initial state of the `value` member.
   - An assignment operator for each of the three types that a variable can contain.
   - `operator+(other)` which has different behaviour depending on what is currently stored in `*this` and `other`.
     - If they are storing different types then an exception should be thrown with an appropriate message.
     - If both store numbers then it should return a new variable containing the sum of those two numbers.
     - If both store text then it should return a new variable that contains the concatenation of the two strings (i.e. how addition usually behaves for strings).
     - If both store lists then it should return a new variable containing the two list concatenated together (by starting with all elements of the first list and then inserting all the elements of the second list).
   - `operator-(other)` which should only work if both `*this` and `other` are storing numbers. If they don't, throw an exception with an appropriate message.

     This operator should return a new variable containing the difference of the two numbers.
   - A `size()` function that returns an `std::size_t`. This function throws an exception for variables containing numbers, but for text and lists it should return the number of characters/elements stored in the text/list.
   - A `print()` function that takes an arbitrary `std::ostream` and prints the content of the variable to that stream.

     If the variable contains a number, we simply print that number.

     If the variable contains text, then print the text surrounded by `"`.

     Finally, if the variable contains a list, then call `print()` on all the elements in the list. The printed elements should be separated by spaces. The whole list should be surrounded by `[` and `]` (see testcases in `given_files/program7.cc` for examples of the format).

   There are testcases given in `given_files/program7.cc`. All of them should pass without any modifications.