LINKÖPING UNIVERSITY
Department of Computer and Information Science
Software and Systems
Christoffer Holm 2024-05-31

Computer examination in

TDDD38 Advanced Programming in C++

Date 2024-05-31

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/`, 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. *Note:* The search functionality should work, but only if you do it through cppreference. You *cannot* search on DuckDuckGo.

1. Unix-style functional composition is an abstraction that allows us to create chains of function calls without having to manually pass the result of one function to the other. For example, instead of doing: [4p]

   ```
   fun3(fun2(fun1(1)))
   ```

   functional composition allows us to use this syntax instead (similar to unix terminals, or C++20 ranges):

   ```
   payload(1) | fun1 | fun2 | fun3
   ```

   This type of constructing will henceforth be referred to as a *composition chain*.

   In this assignment you will implement a small framework for dealing with composition chains. To do this you must implement the following:

   - A class/struct template called `Payload` that represent a value being passed through a composition chain. It contains a data member of arbitrary type `T`, and an implicit conversion operator from `Payload<T>` to `T` (i.e. `operator T()`).
   - An ordinary function template `payload()` that takes a parameter of arbitrary type and returns a `Payload<T>` object containing the passed in function parameter.

     Requirement: The parameter to `payload()` must be a forwarding reference.
   - An `operator|()` that takes a `Payload<T>` object and an arbitrary callable object of arbitrary type `Callable`. It should return a `Payload` containing the value returned by passing the payload to the callable object. Note that the type returned from `Callable` can be different from the type passed in as a parameter.

     Note: We assume that the callable object takes exactly *one* parameter which is of the type `T`.

     Hint: You can either use `decltype` to deduce the return type *OR* the type trait `std::invoke_result`.

   In `given_files/program1.cc` there are a few testcases that should produce no output if everything is correct.

2. **If you passed the midterm test then skip this assignment (you get full points).** [5p]

   It is quite common to have containers that store key-value pairs as `std::pair` (`https://en.cppreference.com/w/cpp/utility/pair`) objects. It is known that one can iterate through such pairs in a concise way using range-based for-loops and structured-bindings, i.e.:

```
for (auto&& [key, value] : v)
{
    // 'key' contains the key
    // 'value' contains the value
}
```

   But if we only want to iterate one of them (the keys for example), then we have an unused variable and that isn't the best practice.

   In this assignment you will implement a function template called `iterate()` that solves this problem. `iterate()` takes two iterators and a callable object and applies the callable object to each element in the iterator range. The "twist" however is that based on the parameters of the callable object it will either iterate the keys, values or both for those iterators that have `std::pair` as their element. See `given_files/program2.cc` for examples on how this function is used.

   To implement this, do the following:

   - Implement `iterate()` so that it iterates through each element, and calls `apply()` (see below) on the current element and the callable object.
   - Implement a helper function called `apply()` that takes an element `e` and a callable object `callable` (and potentially more parameters determined by you). This function have several overloads:
     1. If the element `e` is an `std::pair` and `callable` can be called with `p.first` as its only parameter, then do that.
     2. If the element `e` is an `std::pair` and `callable` can be called with `p.second` as its only parameter, then do that.
     3. If the element `e` is an `std::pair` and `callable` can be called with `p.first` and `p.second` as its two parameters, then do that.
     4. If the `e` is not a pair, then just call `callable(e)`.

     Note that some of these cases may overlap, for example if the `std::pair` has the same type for both fields, in those cases the priority of the overloads is given be the order above.

   There are testcases given in `given_files/program2.cc`.

   Hint: Use SFINAE or C++20 concepts to detect the various cases of `apply()`.

3. **Discussion:** Explain what an *xvalue* is. When does it arise? Give examples. Discuss *why* xvalues were introduced to the language. How does the classification of xvalues help us produce higher quality code? [2p]

4. In this assignment the user will enter a text with multiple lines (they press `ctrl+D` when they are done). Your job is to report whether there are any words that appear on every line (ignoring blank lines). [5p]

   In `given_files/program4.cc` a few suggestions for steps to take are given.

   Note: The user may enter words in any order and may repeat multiple words per line. We only care about how many unique words there are per line and if there are any words that appear on every non-empty lines. Words on a line are separated by an arbitrary number of whitespace characters (except newlines), and lines are separated by newline characters.

   Note: The user may enter blank lines which are ignored by the program.

   Note: Your program is allowed to be case sensitive (i.e. uppercase and lowercase letters are treated as separated letters).

   Requirement: This assignment must be solved using STL algorithms and containers. You may **NOT** use any loops, recursion or `std::for_each`.

   Hint: Recall that `std::istream_iterator<T>` (`https://en.cppreference.com/w/cpp/iterator/istream_iterator`) works as long as `T` has an operator>>, use this fact together with `std::getline()` (`https://en.cppreference.com/w/cpp/string/basic_string/getline`) to read entire lines.

   Hint: Mathematically, what this assignment requests is for you to calculate $L_1 \cap L_2 \cap ... \cap L_n$ where L_i is the set of all words on line i. This operation is similar to a sum, but with set intersection instead of addition.

Example runs (bold is user input)

```
Enter your text:
STL algorithms are extremely useful

there are many algorithms in the STL

STL algorithms are time savers
STL algorithms provide useful solutions that are optimized

discover that many STL algorithms are useful for many tasks
<ctrl + D>
The words common with all lines are:
STL
algorithms
are
```

```
Enter your text:
STL algorithms offer great efficiency
Optimize tasks with standard functions
Enhance productivity using these utilities
No word appeared on every line.
```

5. **Discussion:** Why is it important to use the member initialization list in constructors? Give an example where it is absolutely necessary to initialize data members using the member initialization list. Give another example where the order of initialization in the member initialization list matters. [2p]

6. **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]

7. JSON is a common dynamic storage format. In this assignment we will work with a reduced version of JSON (see below). In this assignment you will implement a reduced framework for storing and printing JSON objects. [4p]

   A JSON object consist of key-value pairs, where the key is a string and the value, called a *field*, can be any of these:

   - A `double` value,
   - A `std::vector` that contains a collection of fields, i.e. it can be an arbitrary collection of any of the things mentioned in this list (referred to as `JSON_List` from now on),
   - Another JSON object.

   Example: Below we have a JSON object:

```
{
  "a" : 5,
  "b" : [1, [2, 3], 4],
  "c" : {
    "d" : 0,
    "e" : 6
  }
}
```

   It has three fields, `"a"` `"b"` and `"c"`. Note that `"a"` contains a `double`, while `"b"` contains a list of fields, where the first and last elements are `double` values, and the second element is another list. `"c"` contains another JSON object with two `double` fields, `"d"` and `"e"`.

   In `given_files/program7.cc` there is a given class representing JSON objects called `JSON`. Your job is to implement the class `JSON_Field`. It needs to fulfill the following requirements:

   - At a single point in time it can store one of the following: `double`, `JSON_List` (see the given file), `JSON`. These members must share the same memory location.
   - It must have a default constructor, and a constructor for each type it can store. Note that it must still be possible to copy and move the `JSON_Field` object, however the compiler generated defaults are enough.
   - It must have an assignment operator that can take any of the types it can store and set the stored value to what was assigned.
   - A print function that takes a `std::ostream` reference, and an indentation (needed by given `JSON::print()` that handles all indentation for you). This function will vary its behaviour depending on what is currently stored, as such:
     - if `double`: just print the value to the stream.
     - If `JSON_List`: print all the values separated by a comma and surrounded by `[` and `]`. You must pass the `indent` value to each element's `print()` function since it might be a `JSON` which is the only case that uses it.
     - If `JSON`: call `JSON::print()` on the stored object.

   The example in `given_files/program7.cc` should work without any modification.

   Requirement: All the potential values that are stored in `JSON_Field` **must** share the same memory location, if they don't you will earn zero points for this assignment (i.e. they must be stored in a union or `std::variant` (`https://en.cppreference.com/w/cpp/utility/variant`)).