LINKÖPING UNIVERSITY
Department of Computer and Information Science
Software and Systems

2022-11-09

Optional Test in **TDDD38** Advanced Programming in C++

Date 2022-11-09

Time 13:15-15:00

Course code TDDD38

Examiner:
Klas Arvidsson (klas.arvidsson@liu.se)
Teacher on call:
Christoffer Holm (christoffer.holm@liu.se)

Allowed Aids (tillåtna hjälpmedel)

An English-* dictionary may be brought.
No other printed or electronic material are allowed.
The cppreference.com reference is available in the exam system, except for the language section.

Instructions

- All communication with staff during the test can be done in both English and Swedish.
- Don't log out at any time during the test, only when you have finished.
- Given files are found in subdirectory `~/Desktop/given_files` (write protected). The test will be available as a `PDF` in this directory at the start of the test.
- Files you want assessed must be submitted via the Student Client.
- C style coding should be avoided where C++ alternatives are available.
- Your code must compile and work as intended.
- Each assignment can be graded as either *Passed* or *Failed*.
- To get a passing grade for the test you need to fully solve *one* assignment.
- There are two assignments, you may attempt both, but if one of your assignments is passed then the other one will be ignored.

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 test 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. In this assignment you will explore how the interface of a stream can be used in a more abstract way than what we are used to. More specifically: you will create a simple stack class which supports the common stack operations. But instead of creating member functions for these operations we will use operator overloading to make the stack look and behave more like a stream.

   Create a class template `Stack` that takes one template parameter `T` that represent what data type is stored in the stack. `Stack` must have a constructor that allows the user to initialize the content of the stack with an arbitrary amount of `T` parameters.

   Use variadic templates to implement the constructor. The parameters *must* be taken as forwarding references and must be properly forwarded to other function calls.

   As mentioned earlier, the operations of the stack will be implemented as operator overloads, more specifically:

   - `operator<<` must be implemented as a member function of `Stack` and must take a `T` parameter. This operation will take the parameter and add it to the stack (think "push").
   - `operator>>` must also be implemented as a member function. It takes a reference to a `T` variable as its only parameter, and will assign the value at the top of the stack to that `T` variable. It will then remove the top value from the stack (think "pop").

   It must be possible to chain these operations, so think carefully about what return type they should have.

   Finally, the stack must be printable to a normal `std::ostream` using the "normal" `operator<<`. When printing the stack the first element to be printed should be the top of the stack and the final value should be the bottom. All elements should be separated with a space (see `given_files/program1.cc`).

   There are a few testcases for `T = int` given in `given_files/program1.cc`, but you should also add tests for other types (for example `std::string`).

   Hint: The internal storage for the stack can either be `std::vector` or a normal C-array (in this case you may assume that the stack never has more than `100` elements).

2. The way the ternary conditional operator works is as follows:

   Given `(condition ? a : b)` it works such that if `condition` is `true` then `a` will be returned from the expression, otherwise `b` will be returned.

   Note that `a` and `b` doesn't have to be variables, but can be full expressions.

   As you might see, this can easily be replaced with an if-statement or a function (`std::max`) and *should* be replaced with those things in all but a select few cases. However; one peculiar property of the ternary conditional operator arises whenever `a` and `b` are different types.

   Since this counts as an expression it must return a singular type, but which one? The type of `a` or the type of `b`? The compiler will choose the type that can most accurately represent both types (if such a type exists).

   Example: The type of `(true ? 1.5 : 2)` will be `double` since *both* `1.5` and `2` can be represented accurately as `double`. `int` is not a valid alternative since `1.5` cannot be represented accurately as `int`.

   With this knowledge in mind, you are to implement a class-template called `common_type` which takes two template-parameters `T1` and `T2`. There should be a type alias called `type` defined inside `common_type` which is an alias for the type which can most accurately represent both `T1` and `T2`.

   You should also specialize `common_type` such that if any or both of `T1` and `T2` are `void`, then `common_type::type` is also `void`.

   Note: `common_type::type` should *not* be a reference-type; `std::remove_reference` (defined in `<type_traits>`) might be useful to make sure that it is not a reference-type.

   There are some testcases given in `given_files/program2.cc`. Using the already defined `std::common_type` for this assignment is not acceptable.