LINKÖPING UNIVERSITY
Department of Computer and Information Science
Software and Systems
Christoffer Holm 2025-06-04

Computer examination in
TDDD38 Advanced Programming in C++

Date 2025-06-04

Time 14-19

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

   In this assignment you will write two type traits: `Add_Type` and `Remove_Type`. Both take two type template parameters, a type `T` and a type parameter pack `P`. Note that there is a given template class `Pack` in `given_files/program1.cc`. `P` is assumed to be an arbitrary instantiation of `Pack`.

   `Add_Type` contains a type alias called `type` which is a `Pack` where the first type is `T` followed by all the types contained in `P`. I.e. `Add_Type` adds `T` to the beginning of the pack `P`.

   `Remove_Type` contains a type alias called `type` which is a `Pack` containing all types contained in `P`, except for any occurances of the type `T`. I.e. `Remove_Type` removes all instances of `T` in `P`.

   There are testcases given in `given_files/program1.cc`.

   Hint: To implement `Remove_Type` you could have specializations for the following cases:

   - If `P = Pack<>` then `type = Pack<>`.
   - If `P = Pack<T, Rest...>` then recursively apply `Remove_Type` on `Pack<Rest...>`.
   - Otherwise, if `P = Pack<First, Rest...>` then recursively apply `Remove_Type` on `Pack<Rest...>` and use `Add_Type` to add `First` back to the beginning of the returned pack.

   Note: Implementing only `Add_Type` will give few partial points, the majority of the points are allocated for `Remove_Type`.

2. For many applications `new` and `delete` aren't appropriate for memory management. In those situations it is common to create a custom memory management scheme. [4p]

   In this assignment you will implement a class called `Pool` that represents a custom memory management scheme. The idea is that `Pool` has two member function templates used for allocation and deallocation of objects, called `allocate()` and `deallocate()`. Both functions takes a type `T` as template parameter. `allocate()` returns a pointer to a default-constructed `T` object, and `dellocate()` takes a `T` pointer and deallocates it (in terms of the custom memory scheme, see below for details).

   This scheme works by creating a memory region (a dynamically allocated `char` array) that is divided into a `count` number of blocks, each being `block_size` bytes large (meaning the dynamically allocated `char` array is `count*block_size` bytes large). `Pool` also contains a vector of `char` pointers called `free`. The idea is that `free` contains all blocks that are currently unused. When an object is allocated the *last* block in `free` is removed from `free` and the object is constructed within that block. When deallocating an object, the destructor of the object is called and then the block is appended into `free` (i.e. inserted at the *end* of the vector). Note that you have to *reinterpret* the `T` pointer into a `char` pointer before inserting it into `free`.

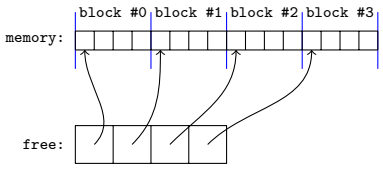
   There is a testprogram given in `given_files/program2.cc`.

   Requirement: The `allocate()` function must use *placement new* to construct default-initialized objects of type `T` within the blocks. Note that we assume that any objects that are allocated will fit inside the block, so no bounds checks are needed. The `deallocate()` function takes a `T` pointer and calls the *destructor* of the object, which can be done with `std::destroy_at()`.

   Requirement: The `Pool` may not have any memory leaks in terms of `new` and `delete`.

   Hint: You can use `new[]` and `delete[]` to allocate and deallocate dynamic arrays.

   Note: Blocks are represented as pointers to the first byte in each block. The memory scheme is summarized in the diagram below. Note that in the diagram below both `block_size` and `count` are set to `4`, so there are `4` bytes per block, and there are `4` blocks so the memory region is in total `4*4=16` bytes large.



3. **Discussion:** What is the difference between using parentheses when initializing a variable, and using curly braces? Give a list of what steps each initialization tries. Are there any other differences that are not covered by the steps? [2p]

4. **Discussion:** Describe what a *view* in C++20 is, and explain how it is semantically different from STL algorithms. Relate your discussion to one or more of the following concepts: lazy/eager evaluation, the functional paradigm or memory usage. [2p]

5. The concept of *adding two things together* has many different interpretations when dealing with a broad definition of *things*. [5p]

   In this assignment you will create a function template `add()` that takes two parameters `a` and `b` of the arbitrary types `T` and `U` respectively. The function template returns a new object that represent `a` and `b` added together, see below for details on what that means.

   For this assignment there are three cases to take into account:

   1. If `a` and `b` can be added using `operator+`, then use that to add them together and return the result.
   2. If both `a` and `b` are containers (i.e. they have iterators) then create a copy of `a` and then pairwise add all elements from `b` to corresponding element in the copy of `a` using the `add()` function.

      Example: Suppose `a = { 1, 2, 3 }` and `b = { 4, 5, 6 }`. Then the result should be `{ 1 + 4, 2 + 5, 3 + 6 } = { 5, 7, 9 }`, which is represented as the same type of container as `a`.
   3. If `a` is a container and `b` is not, then make a copy of `a` and recursively add `b`, using `add()`, to each element in the copy of `a`.

      Example: Suppose `a = { 1, 2, 3 }` and `b = 4`. Then the result, which is the same type of container as `a`, should be `{ 1 + 4, 2 + 4, 3 + 4} = { 5, 6, 7 }`.

   Note: There might be overlap between some of these cases for certain types. The list has been ordered according to their priority which needs to be explicitly implemented.

   There are testcases given in `given_files/program5.cc`.

   Requirement: use SFINAE or C++20 concepts/require clauses to solve this assignment.

   Hint: Declare the `add()` function with `auto` as return type, and then implement helper functions for each of the cases above. Place the implementation of `add()` after all the helper functions.

6. **Discussion:** Briefly explain the difference between `std::list` and `std::vector`. Give one advantage and one disadvantage of each of the two containers. Describe a scenario when one is preferable over the other in terms of your given advantages and disadvantages. [2p]

7. Usually companies are structured into several teams, and it is very common that some employees are part of more than one team. In this assignment you are tasked with finding all unique employees of a company by analyzing the structure of each team. [5p]

   The file `given_files/teams.txt` contains the structure of each team. This file is structured like this:

   - The first line of the file specifies the number of teams the company has.
   - After that each team is listed in order.
   - The first line of each team specifies the number of members in that team.
   - Then all the team members are listed, one member per line. **Note:** You may assume that names does *not* contain any whitespace characters.

   Note: You shouldn't read the file line-by-line. Reading it word-by-word is sufficent for the assignment.

   You must implement the steps given below. You are not allowed to merge any steps, they must be performed separately and the distinction between steps must be clear in your source code.

   1. Read all the teams from `given_files\teams.txt` into an appropriate container structure called `teams`. **Note:** each team must be stored as its own element in `teams`.
   2. Find all members of the team by taking the *set union* of all the teams and store the result in a container called `employees`. **Hint:** this is similar to taking the sum of all elements, but instead of addition we do *set union*.
   3. Print the list of all employees.
   4. Find the largest team (i.e. the team with the most number of members).
   5. Print all employees that are part of the largest team.

   See `given_files/program7.cc` for the expected output of `given_files/teams.txt`.

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

   Hint: Making an informed choice of containers for the structure of `teams` will simplify the assignment greatly.