LINKÖPING UNIVERSITY
Department of Computer and Information Science
Software and Systems
Christoffer Holm 2025-08-21

Computer examination in
TDDD38 Advanced Programming in C++

Date 2025-08-21

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. `std::any` is a powerful but expensive type in the standard library. We can reduce some of the cost by limiting the sizes of the largest types that can be stored in them. In this assinnment you will implement a class called `Bounded_Any` that serves as a compromise between `std::any` and `std::variant`. The idea is that `Bounded_Any` can store *any* type that is smaller than (or as big as) a specified maximum size. [5p]

   In this assignment you must do the following:

   - Implement `Bounded_Any` as a class template, with a size `N` as its template parameter.
   - Finnish the implementation of the given functions `default_dtor` and `retrieve`.
   - Ensure that no memory issues such as segmentation faults or memory leaks can occur when using the class for its intended purpose.

   The `Bounded_Any` class has the following requirements:

   - It must store three things:
     - a C-array of `char` with the size `N` called `memory`. This data member is where all the objects will be stored.
     - a `std::type_index` (`https://en.cppreference.com/w/cpp/types/type_index.html`) of the name `type`. This variable can store objects returned from `typeid` (`https://www.en.cppreference.com/w/cpp/language/typeid.html`), so it will be used to remember what type is currently stored.
     - A function pointer to functions which have the following signature: `void(char*)`. This pointer should be called `dtor` and always points to a function that will properly call the destructor of the currently stored object (will either be `empty_dtor` or an instantiation of `default_dtor`).
   - There must be a default constructor that sets `type` to `typeid(void)` and `dtor` to `&empty_dtor`. This state will henceforth be known as the *empty* state of the `Bounded_Any`.
   - It should not be possible to *copy* a `Bounded_Any`, trying should result in compile errors.
   - The destructor of `Bounded_Any` must call the destructor of the currently stored value.
   - There must be an assignment operator template that assigns values of arbitrary types to the `Bounded_Any`. If the `sizeof` of the type is larger than `N` an exception should be thrown and no changes should be made to the data members. Otherwise a copy of the value is placed into `memory` member and `type` as well as `dtor` are updated accordingly. Remember to call the destructor of the current value before assigning the new value.
   - A `clear()` function must exist, which destroys the currently stored value and puts the `Bounded_Any` into its *empty* state.
   - There must be a member function template `has_type()` that takes a type `T` and indicates whether the currently stored value is of type `T` or not.
   - You must implement a member function template `get()` which takes a type `T` and uses the `retrieve()` function template to extract a `T` object from `memory` **if** the currently stored type is `T`. Otherwise an exception is thrown.
   - Finally there must exist a member function `has_value()` that returns `false` if the `Bounded_Any` is *empty* and `true` otherwise.

   Requirement: No other data members are allowed besides those described above.

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

2. In mathematics a *norm* is the generalized concept of *length.* The most common norm in geometry (and related fields) is the so called *Euclidean norm* which is defined as [4p]

$$||\mathbf{x}||^2 = x_1^2 + x_2^2 + \ldots + x_n^2$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is an n-dimensional vector. In C++ there are many different ways one might represent a "vector", one might do it using a `struct` with `x` and `y` components, any container or by a simple *arithmetic* value (e.g. an `int` or `double`). In this assignment you will implement a function template `norm2()` that calculates the Euclidean norm *squared* (for simplicty) of a value of type `T` (the template parameter).

To implement this you need to handle the following three cases:

1. If `T` is an *arithmetic type* then just return the value multiplied with itself. You can use `std::is_arithmetic_v` (`https://en.cppreference.com/w/cpp/types/is_arithmetic.html`) to detect whether a type is arithmetic.

   Hint: it might be helpful to use `std::enable_if_t` (`https://en.cppreference.com/w/cpp/types/enable_if.html`).

2. If `T` is a *container* or *range* then calculate the sum of each element squared.

   Example: If the container is `{ 1, 2, 3 }` then applying `norm2()` on it should give: `1*1 + 2*2 + 3*3 == 14`

   Note: To detect whether something is a container or range, it is sufficient to check if it has iterators.

3. If `T` has data members `x` and `y` then return `x*x + y*y`.

Note: There might be overlap between cases here (see for example `Weird_Vector` in `given_files/program2.cc`). In those cases the priority is given by the order of the list above, meaning case #1 have priority over case #2 which in turn has priority over case #3.

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

3. **Discussion:** Study the following `struct`: [2p]

```
struct X
{
  double get_value() const;
  int value;
  union
  {
    std::string str;
    double num;
  };
};
```

Explain a possible memory layout of an object of type `X`, consider size, alignment and data member ordering. You must clearly state any assumptions made.

Can a default destructor be generated by the compiler? If yes, explain what it does. If no, give a possible explanation for why the compiler is unable to generate it.

Note: You can assume the following properties for the various data types:

type	size	alignment
`int`	4	4
`double`	8	8
`std::string`	32	8

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

In this assignment you will write a type trait called `Replace` that takes three template parameters: `From`, `To` and `P`.

`P` represents a parameter pack, and the other two are arbitrary types. The idea for `Replace` is that it will produce a new parameter pack where each instance of the type `From` in the parameter pack `P` has been replaced with the type `To`.

In `given_files/program4.cc` the helper struct template `Pack` is given. `Pack` will be used to represent parameter packs.

Example: Suppose `P = Pack<int, char, int>`, `From = int` and `To = float`, then the result of `Replace` on `P` should be `Pack<float, char, float>`.

If `From` doesn't appear in `P`, then the result should be `P`.

The result from `Replace` is placed in a type alias (`using`) called `type`.

There are testcases given in `given_files/program4.cc`.

Hint: To implement this you could use variadic recursion with the following cases (specializations) for `Replace`:

- If `<From, To, Pack<From, Ts...>>` then `type` should be the result of prepending (i.e. adding to the beginning) `To` to result of applying `Result` to `Pack<Ts...>`.
- If `<From, To, Pack<T, Ts...>>` then `type` should instead be the result of prepending `T` to the result of applying `Result` to `Pack<Ts...>`.
- In any other case `type` should be `Pack<>`.

In order to properly implement this you also need to implement a way to *prepend* types to the beginning of a `Pack`. It is recommended to implement this as a type trait called `Prepend` that takes an arbitrary type `T` and a parameter pack `P`. It is probably a good idea to make a specialization of `Prepend` for the case when `P = Pack<Ts...>`, that way you can set `type = Pack<T, Ts...>`.

5. `given_files/schedule.txt` contains a timetable for a train station. Each line represents a train and consists of its arrival time followed by its departure time. In this assignment you are going to write a program that analyzes this timetable to find at what point in time the maximum number of trains are at the station simultaneously and how many they are. [5p]

   Your solution must implement these steps:

   1. Create a container of `std::pair<int, bool>` elements and call it `events`.
   2. Read each time (integer) from `given_files/schedule.txt`. For the first, third, fifth etc. element, pair together the time with the value `true`, and insert them into `events`. For every second, fourth, sixth etc. element, instead pair them with `false`.

      Example: Suppose the timetable is `1 2 3 4`, then, after applying this step, `events` should be equal to `{ 1, true }, { 2, false }, { 3, true }, { 4, false }`. Meaning the bools should alternate between `true` and `false`.

      These `bool` values represent whether the event is the arrival (`true`) or the departure (`false`) of a train.
   3. Sort the events based on the time. In the case of multiple events sharing the same time, then *arrivals* should be sorted before *departures*.
   4. Create a new container of `int`, called `trains`. For each element in `events`, insert either `1` or `-1` into `trains`. `1` is inserted if the corresponding event is an arrival, and `-1` if it is a departure.
   5. Each element in `trains` now corresponds to the change in how many trains are at the station at each event. Your job is to calculate the running sum at each event. I.e. if `trains = { 1, 1, -1, 1, 1, -1, -1, -1 }` then after this step it should instead be `trains = { 1, 2, 1, 2, 3, 2, 1, 0 }` (i.e. for each position we calculate the sum of all elements in the original container up to and including that position). Another way to look at it is to assume there are zero trains from the beginning, and then at each event we calculate how many trains there are after the recorded change.
   6. Find the *index* in `trains` where the most trains occur (if there are multiple, any of those points will do). Then use this index to find the corresponding time in `events`.
   7. Print the point in time when the most number of trains occur, and how many trains there are at that time. Look at `given_files/program5.cc` for an example of the expected output.

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

6. **Discussion:** What is a forwarding reference? Give an example of a forwarding reference and explain how we know it is a forwarding reference and not another type of reference. How is `std::forward()` related to forwarding references? [2p]

7. **Discussion:** Explain the difference between copy semantics and move semantics. How does the compiler determine whether an object should be copied or moved? Give at least one example where the compiler prefers moving a value. Explain a benefit we get from this language feature. [2p]