TDDE18/726G77 – Examination

2025-01-17

Rules

- All code sent for assessment must compile and be well tested.
- Electronic devices are not allowed. Phones must be switched off and placed in a coat or bag.
- Outdoor clothes and bags must be placed in the designated area.
- Students may leave no earlier than one hour after the exam start.
- Fill in invigilators designated list if you need to leave the room.
- All contact between students are strictly prohibited during the exam.
- Books and notes may be reviewed by invigilators during the exam.
- Questions regarding specific assignments or regarding the exam in general are submitted via the communication client.
- System questions can be answered by an assistant if you raise your hand.
- Assignments sent in after the end of the exam will be disregarded.
- You can correct flaws and ask for new assessment until an assignment has grade “Pass” or “Fail”. An assignment can be graded as “Fail” if no significant improvement took place since last attempt.
- Assignments that are graded by points can be handed in again for re-assessment until the maximum points have been reached or it marked as “Final”. When graded as “Final” you keep the point from your previous attempt, but will not be permitted to hand in again.

Aiding material	One C++-book One A4-page with any notes on both sides

Examination

The exam consists of two parts, Part I and Part II. Both are assessed live which means that handed in assignments will be assessed during the exam. Any flaws that the assessment revealed can be fixed and then the assignment can be handed in again for re-assessment.

All assignments in Part I are graded *Passed*, *Try again* or *Failed*. The grade *Passed* will be set if the assignment is fully completed according to specification and without any incorrect practices. The grade *Try again* will be set if there are some issues that need to be fixed. *Failed* is set if no sufficient improvement was made on fixing the given feedback.

Assignments in Part II are graded based on points. Each assignment is worth 10. You may resubmit solutions as long as *significant* improvement is made. If no significant improvement is made you will get one last attempt before the assignment gets marked "Final" and the maximum points earned for that assignment is locked in.

Grading guidelines

For a passing grade you need to pass all assignments in Part I *and* earn at least 10 points in Part II.

For a higher grade you need to pass all assignments in Part I and earn points in Part II based on the table(s) below.

Points earned	**Grade**
10p	3
16p	4
22p	5

Table 1: Grading TDDE18

Points earned	**Grade**
10p	G
19p	VG

Table 2: Grading 726G77

Bonus points

Bonus points earned from the labs are all counted towards the point total of Part II and is added on top of the points earned during the exam.

The bonus is only applicable during the first exam given after the bonus was earned.

Computer environment

Log on

When instructed, log in as normal using you LiU-ID.

Desktop environment

Upon successful log in you will enter the desktop environment. The communication client should start automatically. Note that the network is inaccessible. Networked application features may thus malfunction.
It is important that you leave the communication client running during the entire exam. We may send out public corrections and hints. Notify assistant if it does not start automatically within 5 minutes after log in or after selecting the fish on the desktop.

Terminal commands

`e++17` is used to compile with "all" warnings *as errors*.
`w++17` is used to compile with "all" warnings. **Recommended**.
`g++17` is used to compile `without` warnings.
`valgrind --tool=memcheck` is used to check for 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.

Given files

Any given files reside in the folder `given_files` on your desktop. This folder is write protected, thus you don't have to worry about accidentally changing the given files. To modify a given file, you must first copy it to your work folder, use your desktop as a work folder. You are expected to know how to do this, it is part of the course.

Log off

When your assignment and exam grade is satisfactory (and correct) in your communication client it is safe to leave. If you run out of time you have to leave without knowing the result of your last attempt, contact the examiner by email after the exam to know the result. Terminate all open programs and log out.

Part I

In this part you are presented with three assignments. Your solutions to these assignments must fulfill all specified requirements, follow good programming practice and consist of valid C++ code.

Assignment #1 – STL

In the given file `assignment1.cc` there is a `struct` that represent a board game and a container with board games that represents a collection.

```
struct Game
{
    string name;
    int fun;
    int storage;
};
```

Write code which check whether the board game collection is optimal. If it is not optimal then the name of all non-optimal board games must be printed. If the collection is optimal then to output should be done. A board game is optimal if `fun` divided by `storage` of a game is greater than or equal to `3`.

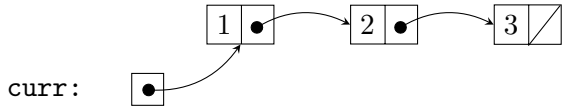
Functional requirements: Execution of the program must give the following output in the terminal:

```
Descent Journeys in the Dark: first edition
Arcs
```

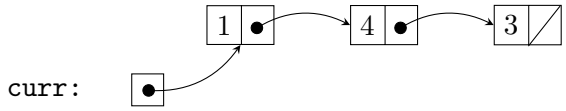
Non-functional requirements:

- You must use one or more appropriate STL-algorithms together with an appropriate lambda function so that your solution can be applied to an arbitrary vector of games.
- You may not modify the given code, you may only make additions.

Assignment #2 – Memory



Write code that creates the linked-structure above. Then write code that modifies the structure so that it looks like this:

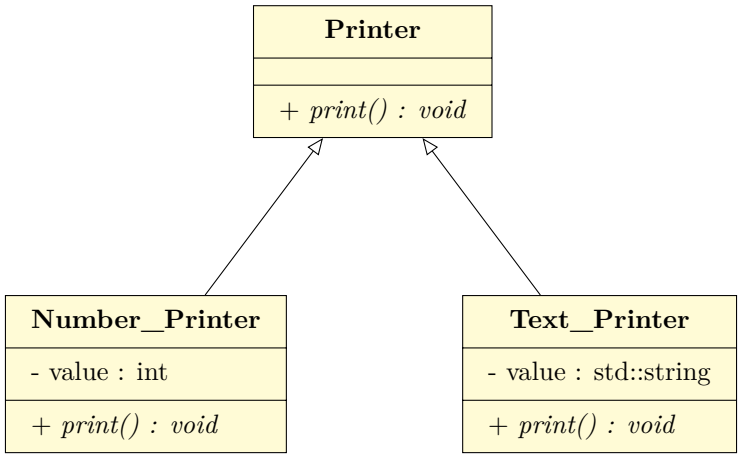


Non-functional requirements:

- Each element must be dynamically allocated and the code may not generate any memory leaks.
- A maximum of two variables of pointer type may be used in the main program.
- At the end of your program there must exist a variable `curr` that points at the first element in the structure.

Assignment #3 – Polymorphism

Implement the following class hierarchy:



`Number_Printer::print()` prints the integer `Number_Printer::value` to `std::cout`.

`Text_Printer::print()` prints the string `Text_Printer::value`, enclosed with `"` characters, to `std::cout`.

You must also create a constructor for `Text_Printer` and `Number_Printer` that initializes their data members.

Non-functional requirements:

- All your classes must be declared and defined in *one* source file.
- The given file `assignment3.cc` has a main-program that should work withot modifications.

Functional requirements: Execution of the program should give the following output to the terminal:

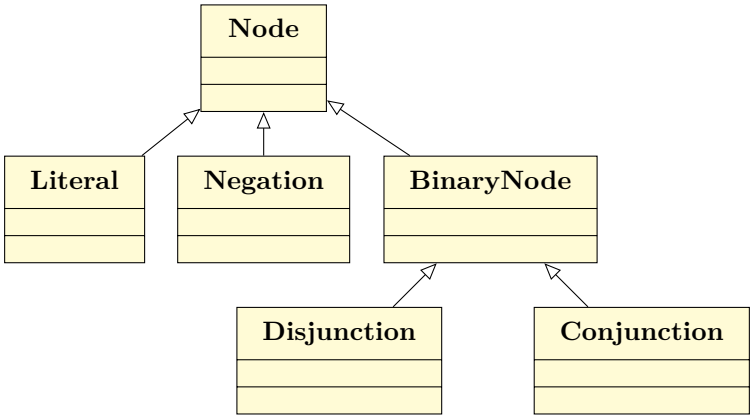
```
5
"My text"
```

Part II

Each of the following three assignments are worth 10 points each, meaning you can earn up to 30 points in Part II. Remember that the bonus points earned from the course is added on top of the points earned here.

Assignment #4 – Polymorphism

Trees is a common way to represent logical and arithmetic expressions in programming. In this assignment you must extend the given code in `assignment4.cc` with those classes that are required for the program to compile and work as intended, the beginning of a UML diagram is:



A short description of each class:

Node Base class in the hierarchy. All nodes that have a specific behaviour during evaluation must implement the member function `eval()`.

Literal Store a `bool`, during evaluation this class returns the stored truth-value.

Negation Stores a reference (or pointer) to a different node. At evaluation this class will return the inverse (negation) of the truth-value calculated from evaluating its referenced node.

BinaryNode Is a node that refers to two other nodes.

Disjunction At evaluation this class will combine the values retrieved from evaluating both referenced nodes by applying *logical or*. Then it returns the result.

Conjunction Similar to `Disjunction` but with *logical and* instead.

Functional requirement: Execution of the program must give the following output:

```
true
```

Assignment #5 – STL

In this assignment you will create a program that keeps track of the stock of a warehouse or store. In `assignment5.cc` there is a given `struct` named `Product` that contains three data members: the name of the product (`name`), the price of one unit of said product (`price`) and how many units are currently in stock (`stock`).

Your assignment is to:

1. modify the given function `read_products()` so that it no longer uses any loops. This function is called in the main-program where the rest of the steps will operate on the vector that is returned from this function (called `products`).
2. Order the vector so that it is partitioned into two parts: the first part contains all products that are in stock (i.e. all products where `stock` is greater than zero), the second part contains all products that are **not** in stock.

   Each part must be internally sorted in such a way that the most expensive product within a part appears first and the cheapest appear last (within the part).

   Note: both parts must still be stored in the `products` vector, you must keep track of the boundary between the two parts yourself.
3. Print both parts according to the example below.

Requirement: In this assignment the aim is to use STL algorithms. Don't use any loops or recursion. You should not create any new containers, instead all work should be done on the given vector.

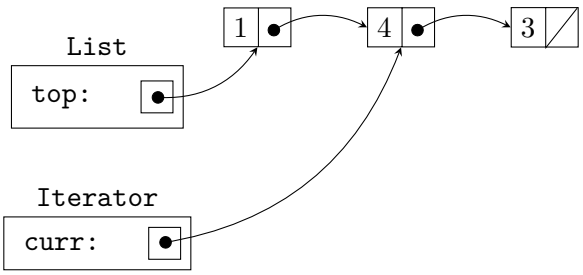
Example run

```
Products in-stock:
 - Fryer (249 SEK)
 - Mat (199.5 SEK)
 - Mop (129.9 SEK)
 - Broom (49.9 SEK)
 - Soap (39.9 SEK)
 - Wrap (29.5 SEK)
 - Bags (25.5 SEK)
 - Towel (19.8 SEK)
 - Brush (15 SEK)
 - Sponge (12.5 SEK)
Product out-of-stock:
 - Iron (399 SEK)
 - Pot (299.5 SEK)
 - Trashcan (149 SEK)
 - Toilet (99.9 SEK)
 - Laundry (89.9 SEK)
 - Vacuum (79 SEK)
 - Cloth (79 SEK)
 - Cleaner (45 SEK)
 - Detergent (35.5 SEK)
 - Spray (25 SEK)
```

Note: There will be severe point deductions for each usage of loops or recursion. Inappropriate choice of algorithm will impose strong point deductions.

Assignment #6 – Memory

An iterator is a class that keeps track of specific location in a container. How this is implemented depens on what specific container is used. In our case the container is implemented as a linked list. Then the iterator keeps track of a specific location by having a pointer to the node that contains the value. In the image below we see an iterator object that keeps track of the second value in a list:



In this assignment you will extend a given implementation of a linked list with a simplified version of an iterator[1]. You must modify the given class `List` and write your own class `Iterator`.

The `Iterator` class will internally store a pointer to a `Node` from the list. This class has an appropriate constructor and the following member operators:

- The dereference operator (`int& operator*()`) – This operator returns a reference to the value of the node that the iterator is currently pointing to.
- Prefix increment operator – Moves the iterator one step forward in the list and returns a reference to the iterator (i.e. to the iterator we just moved). For this assignment you don't have to implement the postfix variant.
- Comparison operator for equality – Compares two iterators and checks whether they are pointing to the exact same node. This is done by comparing the addresses of the nodes they are currently storing.

You also have to implement the following member functions in `List`:

- `begin()` – Create an `Iterator` object that points to the *first* node in the list.
- `end()` – Create an `Iterator` object that points to nothing (`nullptr`).

In this assignment we do not care about any of the other special member functions. So you do **not** have to implement copy or move operations for any of the classes.

Non-functional requirements:

- The program must execute without any memory leaks.
- You are not allowed to modify the `main()` function.

Functional requirements: The execution of the program should result in the following output:

```
1 12 7 2 3
```

[1]This implementation will not fulfill all requirements for it to count as a complete iterator, since we will just implement a curated selection of functions that a "real" iterator would have. Typically an iterator is implemented as an inner class of a container. In this assignment we implement the iterator outside of the class to avoid circular dependencies.