TDDE18/726G77 – Examination

2024-09-19

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 tha 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 13 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**
13p	3
19p	4
25p	5

Table 1: Grading TDDE18

Points earned	**Grade**
13p	G
22p	VG

Table 2: Grading 726G77

Bonus points

Each of the five labs that were handed in within the deadline and fully solved with only one round of feedback grants you **one** bonus point. If all labs were granted a bonus point an additional bonus point will be added as well. Meaning a maximum of six bonus points can be earned during the course.

Bonus points 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

```
struct Date
{
  int month;
  int day;
};

std::vector<Date> dates {
  Date{1, 3}, Date{5, 18}, Date{4, 13}, Date{1, 7}, Date{4, 27}
};
```

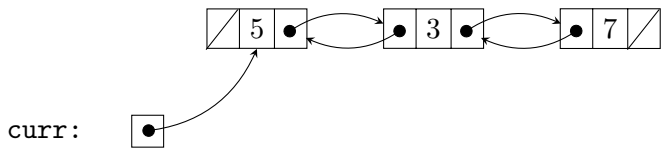
Write code that sorts the vector `dates` so that the months are increasing while all the days within a month are decreasing.

Expected result:

```
Date{1, 7}, Date{1, 3}, Date{4, 27}, Date{4, 13}, Date{5, 18}
```

Requirement: You must use one of the STL sorting algorithms together with an appropriate lambda to make this work for *any* vectors of dates, not just the given `dates` vector.

Assignment #2 – Memory



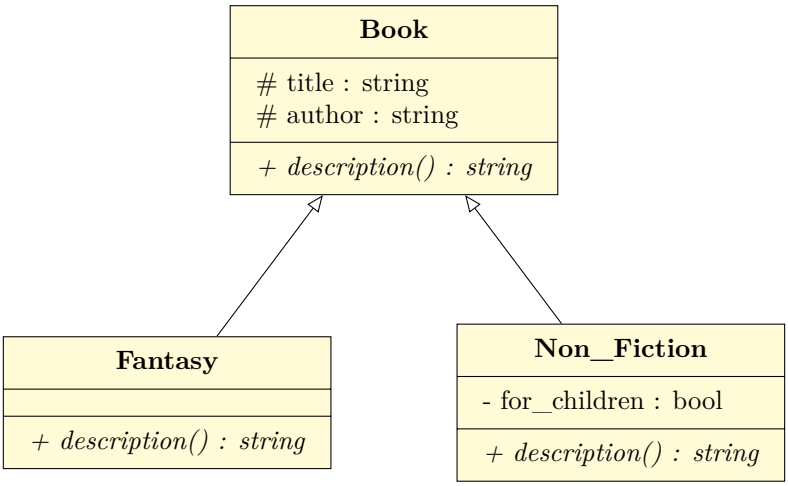
Write code that move element 5 so that it instead lies between the existing elements with value 3 and 7 respectively. The operation described above may only use the pointer variable `curr`. You can assume node are declared as:

```
struct Node
{
  Node* prev;
  int val;
  Node* next;
};
```

Requirement: You are not allowed to move the values, you may only move the pointers. All of the elements must be dynamically allocated and the code may not generate any memory leaks.

Assignment #3 – Polymorphism

Implement the following class hierarchy:



Where the `description()` has the following behaviour for each class:

Book returns a string consisting of the title followed by the author, separated by a comma.

Fantasy returns a string containing the title followed by the word "by" followed by the author.

Non_Fiction does the same as `Book` but adds the string `", for children"` at the end if the `for_children` variable is `true`.

Requirement: The given file `assignment3.cc` has a given main program that should work without modification, and should produce the following output:

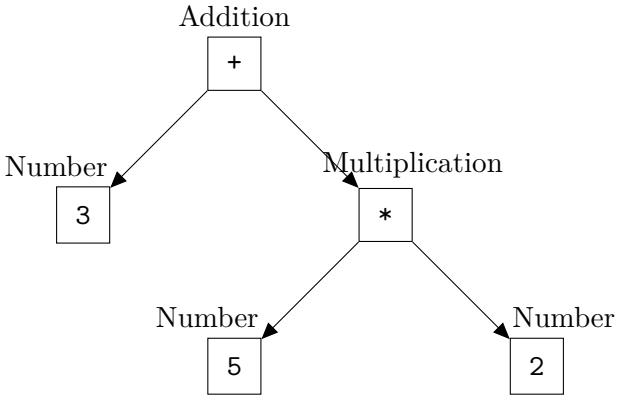
Requirement: All your classes should be declared and defined in *one* source file.

```
1984, George Orwell
Lord of the Rings by J.R.R. Tolkien
C++ Primer, Stanley Lippman
Grumpy Monkey, Suzanne Lang, for children
```

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



Arithmetic expressions can be represented as *trees* consisting of *nodes*. Each node can have none, or two *children* (represented by arrows in the diagram above). Note that there are different types of nodes, we have numbers, addition and multiplication. Number nodes have no children, while addition and multiplication have two. This means that the classes representing addition and multiplication stores two *references* (not pointers) to other nodes. The diagram shows the tree representation of the expression:

$$3 + 5 \cdot 2$$

We can evaluate the expression by asking the addition node (calling a member function) to evaluate itself. The addition node will then ask its children (in this case the number 3 and the multiplication) to evaulate themselves. Then the addition node takes the result from its two children and return their sum. The value node will return 3, while the multiplication asks both of its children to evalute themselves and returns their product. In this case the multiplication will result in 10, which means the addition becomes $10 + 3 = 13$.

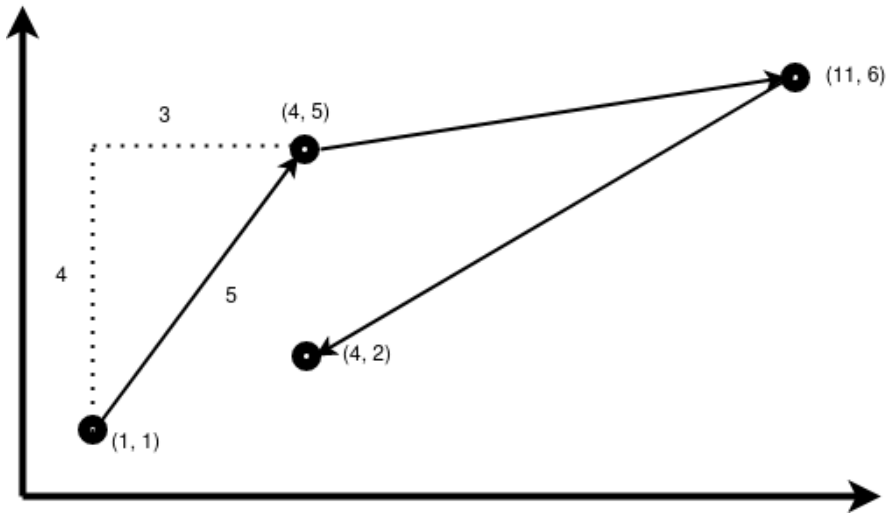
In the given file `assignment4.cc` there is a given program. Your assignment is to implement the class hierarchy required to make this program work.

Hint: Think hard about what classes you should implement, what data members and member functions they should have and whether you need to introduce virtual member functions and so on.

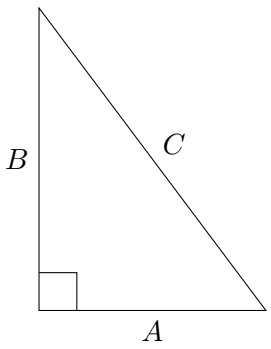
Requirement: The testprogram must work without any modifications to `main`.

Note: You will earn points by demonstrating various different concepts relating to classes, inheritance and polymorphism so it is better to use too many features than too few.

Assignment #5 – STL



This assignment is based around the given file `WAYPOINTS.txt`. In the file there are four *waypoints* which are represented by an x- and a y-coordinate which are separated with `-`. Your assignment is to create a program with *appropriate* STL algorithms that reads this file and prints the total distance travelled if we visited each waypoint in order. We assume that we travel between waypoints using straight lines. The diagram above shows the path defined by `WAYPOINTS.txt`.



To measure the distance between two points we use Pythagoras theorem. This means that the distance between the first pair of waypoints in the diagram are given by: $\sqrt{(4-1)^2+(5-1)^2}$ where $4-1$ is the difference between the x-coordinates of the points, while $5-1$ is the difference between the y-coordinates. In general the hypotenuse is given by:

$$C = \sqrt{A^2 + B^2}$$

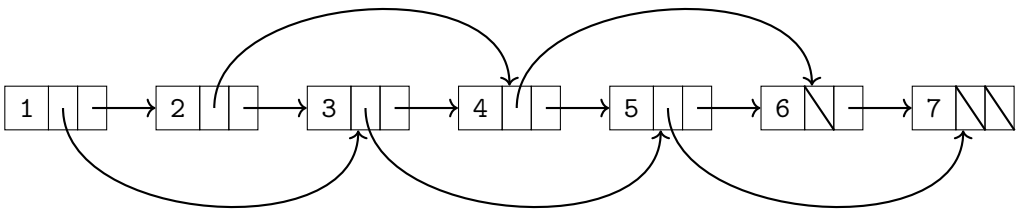
So your job is to calculate the distances between each consecutive pair of points in `WAYPOINTS.txt` and then sum them all together to get the total distance (should be ≈ 20.133). The output of the program should just be the total distance.

You may **not** use any loops or recursion. The aim is to use *appropriate* STL algorithms to solve the problem. The problem should work for any set of valid waypoints, not just the ones given.

Hint: The function `std::sqrt` in `<cmath>` calculates the square root of a number.

Assignment #6 – Memory

In `given_files/assignment6.cc` there is a given *partial* implementation for a slightly optimized linked list that we call `Jump_List`. The difference between a normal linked list and our `Jump_List` is that each node have two pointers instead of one. A `next` pointer that points to the next element. But there is also a `jump` pointer which points to the element two steps ahead. This allows us to search through the list much quicker since we in general can skip over half the elements (see picture).



Your assignment is to implement the following member functions for `Jump_List`:

- Copy constructor
- Copy assignment operator
- Move constructor
- Move assignment operator
- Destructor
- The member function `push_front()` that takes a value and puts it at the beginning of the list. Note that the `jump` pointer should be updated *if* that element exists.

Remember that the list must always maintain the given structure for the `at()` function to work correctly.

Requirement: There may be no memory leaks.

Requirement: You must extend the test program so that it tests all the special member functions.

Hint: The copy assignment operator can reuse the copy constructor by creating a copy as a local variable.

Hint: You may implement the copy logic iteratively or recursively, that is up to you, however a recursive solution is generally much simpler.