TDDE18 - Examination

2023-01-13

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 assessed as "Fail" if no significant improvement took place since last attempt.

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

Information

Grading guidelines - TDDE18

The exam consists of five assignments. Solutions that compile and fulfill the specification as well as follow good conventions and style are assessed “Pass”. Other solutions are assessed “Try again” or (rarely) “Fail”. Grading is based on the number of assignments with a passing grade you solve during the first four hours of the exam. See Table 1. *For grade 3 you always have the full exam time.*

Time	Solved assignments	Grade
3 h	3	5
4 h	4	5
4 h	3	4
2 h	2	4
5 h	2	3

Table 1: Grading TDDE18

Grading guidelines - 726G77

The exam consists of five assignments. Solutions that compile and fulfill the specification as well as follow good conventions and style are assessed “Pass”. Other solutions are assessed “Try again” or (rarely) “Fail”. Grading is based on the number of assignments with a passing grade you solve during the first four hours of the exam. See Table 2. *For grade G you always have the full exam time.*

Time	Solved assignments	Grade
2.5 h	2	VG
3.5 h	3	VG
4.5 h	4	VG
5 h	2	G

Table 2: Grading 726G77

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.

Assignment 1 - Complex numbers

Complex numbers are a mathmatical object that consists of two parts: a *real* and an *imaginary* part. We represent complex numbers as such: $a + bi$ where a is the real part and b is the imaginary.

In the given file `complex.cc` there is a main program that use the `Complex` class. Your job is to implement this class. It has two data members of type `double` that represent the real and the imaginary part of the number respectively.

The `Complex` class must have an *appropriate* constructor and the following operator overloads:

- Output stream operator (`operator<<`), see the example for how it should behave.
- The binary operator `operator+` that produces a new complex number by adding two complex numbers together. Adding two complex numbers means adding each component (real and imaginary) together. Suppose we have two complex numbers u and v then $u + v$ is defined as such (where $\text{Re}(u)$ denotes the real part, and $\text{Im}(u)$ denotes the imaginary part):

$$\begin{aligned}\text{Re}(u+v) &= \text{Re}(u) + \text{Re}(v)\\ \text{Im}(u+v) &= \text{Im}(u) + \text{Im}(v)\end{aligned}$$

- The binary operator `operator*` which produces a new complex number by multiplying two complex numbers together.

  This is a bit more complicated than addition, but it should follow this pattern:

$$\begin{aligned}\text{Re}(u \cdot v) &= \text{Re}(u) \cdot \text{Re}(v) - \text{Im}(u) \cdot \text{Im}(v)\\ \text{Im}(u \cdot v) &= \text{Re}(u) \cdot \text{Im}(v) + \text{Im}(u) \cdot \text{Re}(v)\end{aligned}$$

Example output:

```
u = 1+2i
v = 3+4i
u + v = 4+6i
u * v = -5+10i
```

Assignment 2 - Macros

In the context of programming, a *macro* is similar to a variable, but instead of keeping track of a value it keeps track of a piece of text. Then, whenever a macro name is found in the code it will replace that name with the specified macro definition.

This concept is however not exclusive for programming, it can be used in text processing to make short-hand for common terms or phrases. For example:

Suppose we have to write `Linköping University` a lot in our texts, then we can define a macro, let's call it `LiU` and let its definition be `Linköping University`. Then we can for example write `I study at LiU` Which *expands* to `I study at Linköping University`.

One problem with macros however is that their definitions might refer to other macros. For example, suppose we have the following macros:

```
IDA = Department of Computer and Information Science
LiU = Linköping University
ORGANIZATION = IDA / LiU
```

Then the text `The course TDDE18 is given at the ORGANIZATION` should expand to `The course TDDE18 is given at Department of Computer and Information Science / Linköping University`

I.e. we had to recursively expand macros inside the macro. This can occur in an arbitrary number of steps.

In this assignment you will implement macros by following these steps:

1. create a function called `define_macros()` which takes an `std::ifstream&` called `ifs`.

   Each line in `ifs` has the following format: `<MACRO NAME>:<DEFINITION>`
   See `MACROS.txt` for examples.

   You must create an appropriate container which can associate a macro name with its definition. It should be possible to find a macro definition by just supplying its name.

   `define_macros()` should return the constructed container.

   Hint: `std::getline` can be used to extract the name (look for `:`) and then you can extract the definition by also using `std::getline` (look for `\n`).

2. create a function called `expand()` which takes a string containing a line and the macros container returned from calling `define_macros()`. This function will expand all macros within the line and return the fully expanded line as a string.

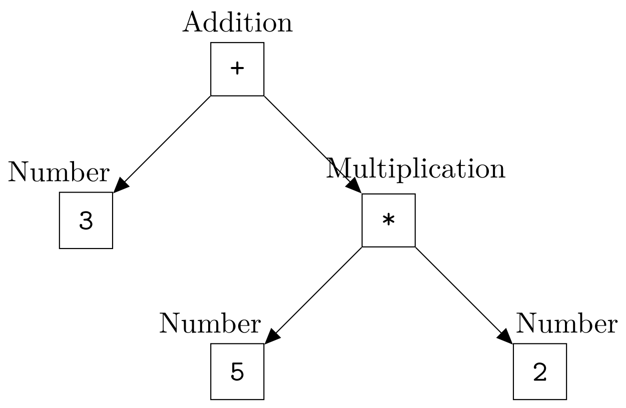
   To implement this function you should go through the line word-by-word. Check if the word is a macro by looking for it in your container. If it is a macro, then call `expand()` on its definition and put the result in the output string. If a word isn't a macro, then just add it to the output string (followed by a space).

   Hint: Use `std::istringstream` and `operator>>` to go through the line word-by-word.

   Note: It is important that we call `expand()` on the macro definitions here rather than in `define_macros()` since we don't know all the macros until after the `define_macros()` is done.

There is a main program and an example run given in `macro.cc`.

Assignment 3 - Math with classes



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:

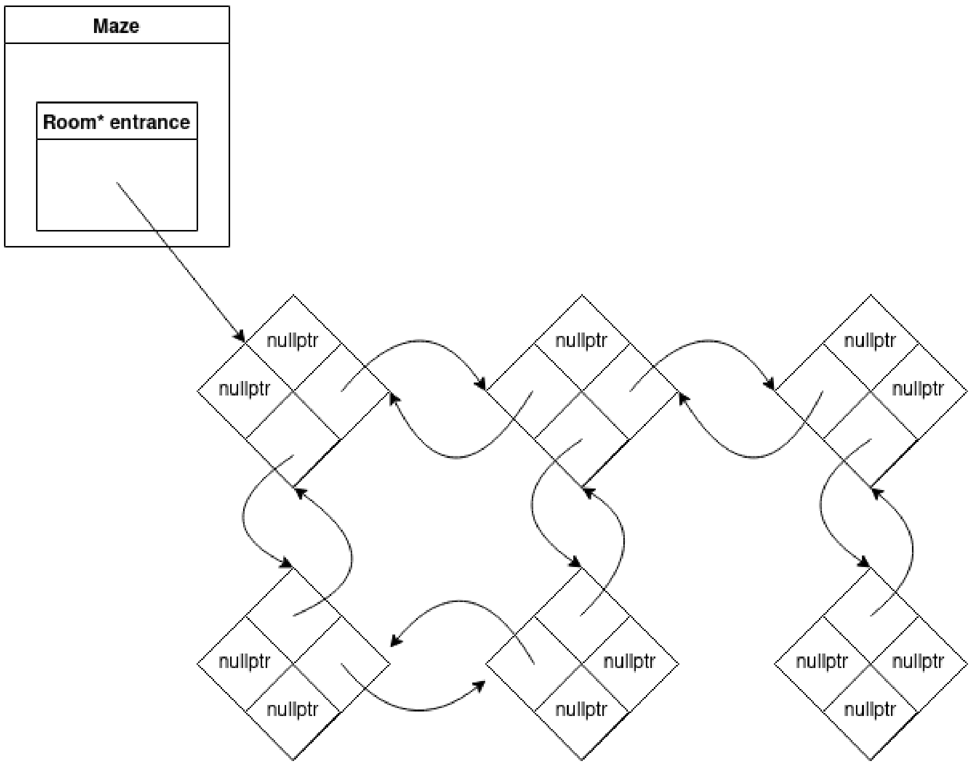
$$3 + 5 \cdot 2$$

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 `expression.cc` there is a given program. Your assignment is to implement the class hierarchy required to make this program work. You must implement 5 small classes.

- `Node` is the base class that represents an arbitrary node. It has no data members and is *pure-virtual* (abstract). It has the member function `int eval()` which doesn't have an implementation.
- `Number` inherits from `Node`. It stores one `int` data member and implements `int eval()`. The overriden `eval()` should return the data member value.
- `Binary` is a pure-virtual class that inherits from `Node`. It has two data members of type `Node&` (these represent its left- and right child).
- `Addtion` inherits from `Binary` and implements `eval()`. This member function evaluates the children (by calling `eval()` on them) and returns their sum.
- `Multiplication` inherits from `Binary` and implements `eval()`. This member function evaluates the children (by calling `eval()` on them) and returns their product.

Assignment 4 - Maze



In the given file `maze.cc` there is a start of an implementation for a linked maze (`Maze`). The class have a data member `entrance` of type `Room*` which leads to the first room of the maze. Each room have up to four neighbouring rooms.

All the memory is managed by the `Maze`, which includes the rooms. But the problem is that `Maze` doesn't have a destructor, so it has memory leaks. It is your job in this assignment to implement the destructor of `Maze` so that it doesn't have any memory leaks.

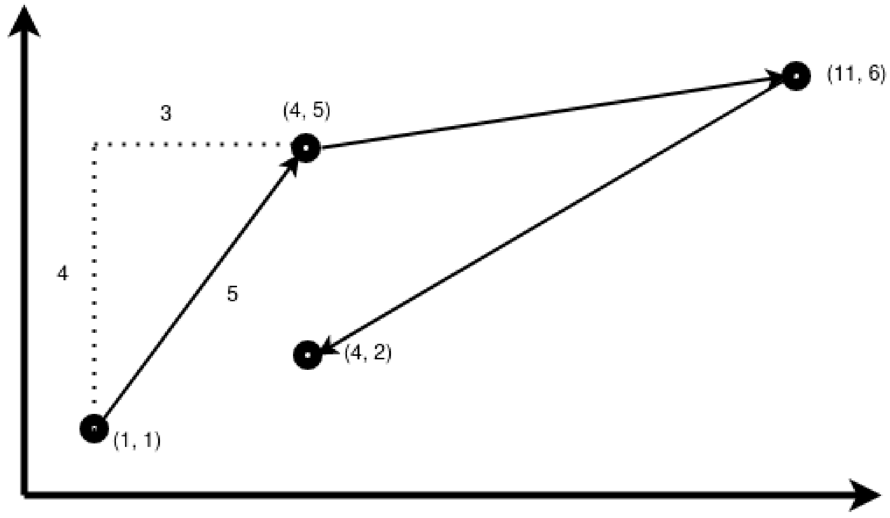
Note that the maze (you can see it in the diagram above) have *cycles*, meaning there are multiple ways to enter the same room. Because of this you have to make sure that a room is deleted *only once* (even though there are more than one way to find it), and that *every* room gets deleted.

Test if your solution works without memory leaks using valgrind:

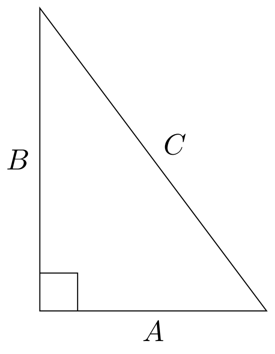
```
valgrind --leak-check=full ./a.out
```

Hint: You need to keep track of which rooms have been visited somehow. A suggestion for how to implement this is to create a helper function that you *recursively* call on all neighbours. This function should only run if the specified room has not yet been visited.

Assignment 5 - Measure distance



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.