TDDE18 - Examination

2022-01-14

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 exam end 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.
- Correctly compiling code, complete fulfillment of requirements, and use of good programming conventions and style are requirements for “Pass” grade on an assignment.

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

Information

Grading guidelines - TDDE18

The exam consists of five assignments. Solutions that fulfill specification and follow good conventions 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 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 fulfill specification and follow good conventions 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 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. Network applications 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 in the start menu.

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 pages

During the exam, you will have access to `http://www.cppreference.com/` in the browser Chromium. Note that only this site is accessible, and that some features on the site may be blocked.

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 - Date formatting

All over the world people have the need to communicate specific days. This is usually done with the help of calendar dates, i.e. a specific year, month and date combined communicates a unique day. But people write these dates in various different ways depending on where in the world they are.

In this assignment you will create a class called `Date` that represents a calendar date. This class contains 3 integer data members `year`, `month` and `date`. Write a constructor that allow these members to be initialized. `Date` also contains a pure-virtual function called `to_string` that is used to get a string representation of the date.

Create a subclass of `Date` called `YMD_Date`. This class represents a date that is printed on the format `yyyy-mm-dd`. For example `2022-01-14`. This subclass has no extra data members and should have a similar constructor as `Date`. `YMD_Date` overrides `to_string` and return the date on the specified format. Notice that `date` and `month` should always be represented by two digits.

Create another subclass of `Date` called `MDY_Date`. This class represents a date that is printed on the format `day m/d/yyyy`. For example `Thursday 2/19/2020`. `MDY_Date` adds a string data member called `day`, which represents which weekday it is that day. This, and the other data members should all be set by a constructor. `MDY_Date` overrides `to_string` such that it returns a string representation of the date on the specified format. Note that `date` and `month` are printed as they are, no extra zeroes should be added.

There is a partial test program given in `given_files/dates.cc`. You must complete that program.

Your program should not contain any memory leaks nor should it contain slicing.

Assignment 2 - XML validation

XML (eXtensible Markup Language) is a common way to structure data in a both human and machine readable format. An XML file consists of *elements* which are blocks of text surrounded by *tags*.

A tag is an arbitrary string (consisting of only letters) that is surrounded by `<` and `>`. It can look like this `<tag>`. There is also a special tag called a *closing tag*, which looks like this `</tag>`, i.e. it is a tag that starts with `</`. Each tag must be closed with a closing tag. An element looks like this:

```
<tag>
Some text here,
which can span multiple lines.
</tag>
```

An element can also contain elements themselves (look at `valid.xml` for a complete example).

In this assignment you will write a program that checks whether or not a given XML file is valid. In this assignment you may assume that each tag is on its own line with nothing else but the tag on that line. If the given XML file is invalid then an appropriate error message will be printed.

There are detailed instructions on how to write this program in `given_files/xml.cc`. There are also four test files in `given_files` called `valid.xml`, `invalid1.xml`, `invalid2.xml` and `invalid3.xml`.

Example runs:

```
$ ./a.out valid.xml
Valid XML file!
```

```
$ ./a.out invalid1.xml
Invalid tag <123>
```

```
$ ./a.out invalid2.xml
Tag <name> not closed
```

```
$ ./a.out invalid3.xml
There are unclosed tags
```

Assignment 3 - Schedules

Keeping a schedule of what to do at what times is a common organizational tool. Often we find that we keep multiple schedules: one for work or studies and one for our private life. But in order for schedules to be as efficient as possible we want to view them all at once.

In this assignment you will use STL (the standard library) to create a program that takes two schedules and merge them into one. In `given_files/schedule.cc` there are two example schedules.

Each schedule is a `std::vector` containing events. An event is a `std::string` with the format `"<start time>-<end time> <description>"`, where `<start time>` and `<end time>` are time-points on the format `"HH:MM"` (example: `"08:00"`). These timepoints indicate when this event starts and ends respectively.

`<description>` is a text describing this event.

There are step-by-step instructions for how this program should be implemented given in `given_files/schedule.cc`. Make sure to follow these steps as closely as possible.

In this assignment you are to use standard algorithms to implement the program described above. No hand-written loops are allowed. It is important that you choose appropriate algorithms to solve the problem, `std::for_each` is usually not a good choice.

Assignment 4 - K-Cats

In this assignment you will implement a special data structure called *K-Cats*. It is exactly like a stack but with these differences:

- Instead of pushing to the top, a K-Cats will push to the bottom.
- Instead of popping from the top, K-Cats will pop from the bottom.

In `given_files/kcats.h` there is an incomplete definition of the class `KCats`. It is your job to complete this definition and then implement *all* member functions in a new file called `kcats.cc`.

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

Requirements

1. The K-Cats consists of node elements, just like a stack. You must implement these node elements yourself.
2. The K-Cats must have correct memory usage.
3. The public interface may not be changed in `kcats.h`.
4. `pop` will return the removed element.
5. Correct usage of `const` is required.
6. Your solution should have correct header and implementation files.
7. The copy constructor and the copy assignment operator must be private and should **not** do anything special (you are also allowed to delete them if you like).
8. The move constructor and the move assignment operator must be correctly implemented.
9. You are not allowed to modify the given test program at all.
10. You must send in all three files; `kcats.h`, `kcats.cc` and `kcats_test.cc`.

Example run:

```
$ ./a.out
1 5 72 35 2 99
99 2 35 72 5 1
```

Hint: Even though a K-Cats operates at the bottom rather than the top, its implementation does not have to reflect this.

Assignment 5 - Counter

Counters are very common in a lot of different programs. In this assignment you will implement a class called `Counter` which can be printed and increased in different ways.

A `Counter` has a name (represented as a string) and a value (represented as an integer). When a counter is printed it will print the name inside brackets followed by an equality sign, followed by the value.

Example: A counter that is named `"My Counter"` and has the value `1` will be printed as `[My Counter] = 1`

It should be possible to add integer values to the counter value with `+`, `+=` and `++`.

There is a test program given in `given_files/counter.cc`.

Requirements

1. It should be possible to print a `Counter` to a `std::ostream` with `operator<<`.
2. It should be possible to increase a counter with `operator+=` and `operator++` (pre- and postfix). All three of these operators must be member functions of `Counter`.
3. There should be two implementations of `operator+`. Both of which are ordinary functions outside of the class.
4. `operator+=` and `operator+` should add an `int` value to the counter.
5. All plus operators should have the same behaviour as if they where performed on the value of the counter.
6. No code duplication allowed. Reuse operators as much as possible when implementing the others.
7. You are not allowed to change the given code.

0.1 Example run:

```
$ ./a.out
[My Counter] = 1
[My Counter] = 2
[My Counter] = 2
[My Counter] = 6
[My Counter] = 11
[My Counter] = 11
```

Hint: Implement `operator+=` first and implement all other arithmetic operators with that.