TDDE18 - Examination

2020-01-15

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 fulfilment 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 fulfil 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

Tabell 1: Grading TDDE18

Grading guidelines - 726G77

The exam consists of five assignments. Solutions that fulfil 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

Tabell 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 normal desktop environment (Mate-session in Linux Mint). 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 - Special cards

The board game Gloomhaven contain several special cards. These cards have three parts: a top effect, a bottom effect and an initiative. An example of such a card can be seen to the right.

In `assignment1.cc` there is a given main program, your assignment is to implement all classes that are used by this program. You must create 3 classes according to the following descriptions.

`Effect` is an aggregate (a `class` or `struct` with only public data members) with 2 data members. The data members are `name` and `description`, both of which should be of type `std::string`.

`Card` is a class with 3 data members. The data members `top` and `bottom` are objects of type `Effect`. The data member `initiative` is an integer value. `Card` must implement operators for comparison and for printing to an output stream. It is enough to implement the "less than" operator for comparison. Cards are compared based on `initiative`.

`Hand` is a class with one data member; a container that contains an arbitrary amount of `Card` objects. `Hand` has two member functions, `draw` and `print`. `draw` adds a new card to the container and `print` will print all the cards in the container to an arbitrary `std::ostream`. `print` must use the output stream operator of the cards.

Example run:

```
$ ./a.out
Card1:
Top: Aid from the Ether
Initiative: 91
Bottom: Summon Mystic Ally

Top: Aid from the Ether
Initiative: 91
Bottom: Summon Mystic Ally

Top: Ice Lance
Initiative: 25
Bottom: Ride the Wind

card1 is less than card2: false
card2 is less than card1: true
```

Assignment 2 - Parts of a game board

The game board of the board game Gloomhaven consists of several pieces connected together. Such a piece is called a `Tile` and it consists of several hexagons. Each hexagon has an arbitrary amount of things stacked on top of it. In this assignment we represent a hexagon with `std::string` where each `char` is one thing placed on that hexagon. The thing that is at the top of the stacked things is the last character in the string.

Your job is to create the class `Tile` that works as demonstrated in the given main program located in `program2.cc`.

The class `Tile` is a piece of the game board and have two data members: a string `name` and a container of hexagons called `hexes`. `hexes` stores an arbitrary amount of hexagons. Each hexagon should be associated with an x- and a y-coordinate. `std::map` is an appropriate starting point. `Tile` must work according to these requirements:

- It should work without any changes to the main program.
- A `Tile` should be created with only a name.
- It should be possible to add new hexagons with a function that takes an x/y-coordinate and a `std::string` with the function `create_hex`.
- Adding new things on top of a hexagon should be done with the function `push` that takes one coordinate and one `char`. The passed in thing (`char`) should then be placed on top of the hexagon at the passed in coordinate.
- Individual hexagons should be possible to print with the function `print_hexagon`. `print_hexagon` prints the hexagon at the passed in coordinate.
- When printing a hexagon then only the thing at the top of the hexagon should be printed.
- The entire `Tile` should be printed with the function `print_tile`.

Example run:

```
Hexagon(0, 0): [X]
b1:
Hexagon(0, 0): [O]
Hexagon(0, 1): [H]
```

Assignment 3 - Adversaries with a common base class

In Gloomhaven there are several different types of monsters. In this assignment you are going to use polymorphism and inheritance to represent 2 different monsters, bandit archers and bandit guards.

You must create the three classes: `Adversary`, `BanditGuard` and `BanditArcher` according to the following descriptions:

`Adversary` has one integer data member that represents the monsters health. It has the member functions `description` and `to_string`. `description` has no default behaviour. `to_string` returns the monsters health as a string according to the following format: "`hp: x`".

`BanditGuard` extends the behaviour of `Adversary`. It adds an integer data member for mobility. Objects of this data type implements the member function `description` which returns the string "`Bandit guard`". `to_string` return the bandits health and mobility as a string on the following format: "`hp: x movement: y`".

`BanditArcher` extends the behaviour of `BanditGuard`. It adds an integer data member representing how far the archer can shoot with its bow. `description` returns the string "`Bandit archer`" and `to_string` return the bandits health, mobility and range on the following format: "`hp: x movement: y range: z`".

In `assignment3.cc` there is a main program given. You are going to have to make some changes in this program to solve the problem of slicing and potential memory leaks.

Example run:

```
Adversaries:
=============
Bandit archer:
hp: 1 movement: 3 range: 2

Bandit guard:
hp: 2 movement: 4

Bandit guard:
hp: 3 movement: 3
```

Assignment 4 - Count words

While playing Gloomhaven it can be interesting to analyze the discussion around the table. In this assignment you are going to take a text from `std::cin` and print a table of each unique word and the number of times it occured. However, there is a catch; you are not allowed to use `std::map` (which probably would have been a perfect container for this assignment). Instead you will have to solve this problem with appropriate STL algorithms and other containers.

Requirements

- You must use appropriate algorithms.
- Neither `std::map` nor `std::for_each` can be used.
- You cannot use any loops (`for`, `while` or `do-while`), you have to use algorithms instead.
- The user must indicate that the input text is complete by pressing `ctrl+D`.

Example run:

```
$ ./a.out
I found loot over here
I too found loot

I: 2
found: 2
here: 1
loot: 2
over: 1
too: 1
```

Assignment 5 - Building a game board

In the given code `assignment5.cc` there are parts of a linked data structure implemented. This linked structure represents a map in the game Gloomhaven. A map consists of many `Tiles` that are connected before the game begins. This way we can create many different maps for the game. In the given file there is a class `GMap` that represents a map, but it does not work correctly. It is your job to complete the implementation.

`GMap` contains a pointer to the first tile and member functions necessary for adding new tiles as the first tile in the linked structure. It is also possible to remove the first tile, after removal then the second tile becomes the first tile.

`Tile` contains a string with the name of the tile. It also contains a pointer to the next tile.

The following problems you will have to fix:

- The class does not take responsibility to cleanup any resources it owns.
- Missing implementation and testcases for the move operations (move constructor move assignment operator). **Hint:** `std::move` can be used to test move operations.
- `GMap` does not implement copying in the correct way. It should **not** be possible to copy a map.

There is no example run that demonstrates the desired output since it is your job to rewrite the main program to test the class in an appropriate way.