TDDE18 - Examination

2024-01-12

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 - Classes and operators

Numbers are a very useful construction in all walks of life. What is lesser known however is that there are a numerous different types of numbers, one of which is called *ordinals*. An ordinal is a number that represents *order*. Example of ordinals is words such as *first*, *second*, etc.

In mathematics the ordinals are defined recursively by first defining `0` to be an empty list, and then the `n`:th ordinal is defined as a list of all the numbers *preceding* it (i.e. all non-negative integers it comes after). This means that `1` is defined as `{0}`, the ordinal `2` is defined as `{0, 1}` and so on.

Increment (also known as the *successor function*) of ordinals is defined as simply adding yourself to the list thus giving us the next ordinal. Addition is defined as repeated increment.

In this assignment you must create a class called `Ordinal` which represents a so-called ordinal. It should fulfill the following requirements:

- Contains a list of `unsigned` values.
- It should be possible to default-initialize such that the internal list is empty (i.e. it represents the ordinal `0`).
- There should be a member function called `successor()` that returns the next ordinal.
- Both variants of `operator++` must be defined. These operators transforms the current ordinal to the next one (the successor). One way to implement this is to just add the current size of the list to the end of the list.
- Define `operator+` when adding two ordinals `lhs` and `rhs`. This is implemented by repeatedly applying the increment operator on a copy of `lhs` as many times as there are elements in the list of `rhs`.
- It should be possible to print the ordinals as a comma separated list with `operator<<` (see example below for exact format).

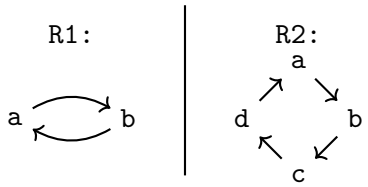
There is a test program given in `given_files/assignment1.cc`.

Running example

```
0:
1: 0
2: 0, 1
2: 0, 1
3: 0, 1, 2
4: 0, 1, 2, 3
```

Assignment 2 - Classes

In computer science there is a theoretical concept called *relations*. This is a type of table which associates two objects with each other. Often these relations are drawn as diagram, like this:



Here we see two example relations: `R1` och `R2`. Each arrow represent that an object is related to another. In `R1` we see that `a` relates to `b` and the other way around. While in `R2` we see that `a` relates to `b`, while `b` relates to `c` and so on.

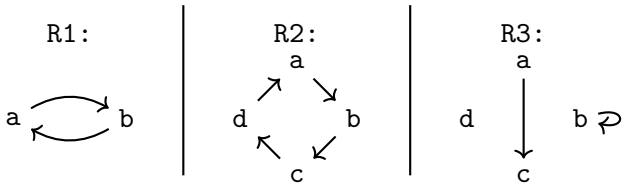
In this assignment you must create a class called `Relation` which implement these types of relations. It must fulfill the following requirements:

- It must contain a vector. In the vector we store a pair of characters (`char`) which represents the relation. `R1` from the diagram above would therefore contain the pairs `'a'`, `'b'` and `'b'`, `'a'`.
- There must be an `add_relation()` member function that takes a pair and adds it to the vector. Note that there *may* be multiple arrows for each pair of characters.
- There must be a `to_string()` member function that returns a string representation of the whole relation (see running example below).

Besides these requirements there must also be a member function called `compose()` that takes a `Relation` object as a parameter and returns a new `Relation` object which is the so called *composition* of the current relation with the one that got passed in to the function.

Two relations `A` and `B` are composed to a *new* relation `C` by iterating all pairs of arrows from `A` and `B`: if the arrow from `A` points to an object which is the start point of another arrow in `B` then we add to `C` an arrow from the start object in `A` which points to the end point of the arrow in `B` (there is pseudo code for this in `given_files/assignment2.cc`).

Example: Below we can see the composition (`R3`) of `R1` and `R2`:



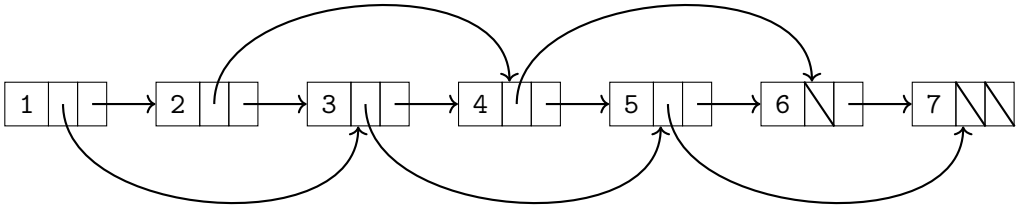
Note that there is an arrow from `a` to `b` in `R1` and an arrow from `b` to `c` in `R2`. This means that we in `R3` add an arrow from `a` to `c`.

Running example

```
r1: a->b b->a
r2: a->b b->c c->d d->a
r3: a->c b->b
```

Assignment 3 - Memory management

In `given_files/assignment3.cc` there is a given 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 special member functions for `Jump_List`:

- Copy constructor
- Copy assignment operator
- Move constructor
- Move assignment operator

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.

Assignment 4 - Polymorphism

You and your friends have decided to develop a simple adventure RPG game. An important aspect of this game is to collect various items and store them in a bag. In this assignment you will create a class hierarchy that represent different items and weapons for the game.

In this assignment you must implement the following classes:

`Item` acts as a base class for the whole hierarchy. This class should have a data member named `weight` of type `double`, with a corresponding getter-function `get_weight()` which returns the weight. There should also be an appropriate constructor that initializes `weight`.

It should have two virtual functions: `print()` which is pure-virtual and takes an output stream, and `print_info()` which takes an output stream.

`print_info()` should print `(<weight> kg)` to the passed in output stream where `<weight>` is replaced with the actual value of `weight`. Note that this function should not be possible to call outside the class hierarchy.

Shovel inherits from `Item`. This class should have a default-constructor that initializes `Item::weight` to `3.0`. `Shovel` should also override `print()` such that it prints the string `"a shovel "` to the passed in stream. It should then call `print_info()`.

Weapon inherits from `Item`. It should have the data member `damage` which is of type `double`. `Weapon` should have an appropriate construct that initializes `damage` and `Item::weight`. There should also be a `get_damage()` getter which returns the value of `damage`.

This class should override `print_info()` so that it prints `"dealing <damage> damage "` to the passed in output stream, where `<damage>` is replaced with the actual value of the `damage` member. Then it should call the base class implementation of `print_info()`.

Sword inherits from `Weapon`. It has the string data member `name` which is initialized with an appropriate constructor. Note that the constructor must initialize all data members from the base classes too.

`Sword` overrides `print()` such that it prints `"A sword named '<name>' "` to the output stream, where `<name>` is replaced with the content of the `name` member. It should then call `print_info()`.

Bow inherits from `Weapon`. It should have the same constructor as `Weapon` and override `print()` so that it prints the string `"A bow "` to the stream and then calls `print_info()`.

There is a partial test program given in `given_files/assignment4.cc` which you must modify so that it works as intended.

Requirement: `Weapon::get_damage()` **cannot** be virtual nor appear in `Item`.

Assignment 5 - STL

It is a surprisingly common problem in programming to get a list and remove all the duplicates. In this assignment you will a slightly more difficult version of this problem which is to remove all duplicates while still maintaining the entered order of the unique values.

Example: Assume that the user enters `can you can a can as a canner can can a can`

then we see that the words `can` and `a` are repeated multiple time, so the program should remove all occurrences of these words except the first occurrence and then print the following: `can you a as canner`. Note that the words are still printed in the same order they were entered.

Requirement: To solve this problem you must use STL algorithms and appropriate containers. No loops, recursion or `std::for_each` are allowed.

Requirement: You may only create *one* container in your solution.

Hint: Create a container that contains an `std::pair<int, std::string>` where the first value represent which index the word had in the entered order and the second value is the word. Using this you can change the order freely and then sort the vector again based on the indices in the first position of the pairs.

Running example (bold is user input)

```
$ ./a.out
can you can a can as a canner can can a can <ctrl+D>
can you a as canner
```

```
$ ./a.out
I bet they bet on black <ctrl+D>
I bet they on black
```

```
$ ./a.out
d a b z b c a <ctrl+D>
d a b z c
```