Linköping university/ITN

16 March, 2022
08:00 – 13:00

TNG033: computer examination

Aid: A C++ book, with errata. Comments and notes in the margin of the book, directly relating to the text and examples on the relevant pages. Pencil and white paper.

Access to cppreference.com, TNG033: lectures (login: `TNG033` password: `TNG033ht13`), C++ notes website (login: `TND012` password: `TND012ht2_12`) and TND012: lectures (login: `TND012` password: `TND012ht2_12`).
To access the websites above, use **Google Chrome**.

Search in cppreference.com: https://en.cppreference.com/w/Special:Search .

No other help material is allowed during the exam.

Questions related to the exam exercises are answered by the course staff. You can request to the exam watchers to contact the course staff by phone.

Questions related to the functioning of the computers, software, or the procedure for computer examinations are answered by LiU-IT. You can request to the exam watchers to contact the IT-support team, by phone (call 013-282828).

Not allowed: Solutions of previous exams/duggor. Code for lab exercises or course exercises. USB. Extra pages containing notes or programming code. Private laptops.

Credits: Next to each question, you can find how many points are awarded.

IT-support: It is your responsibility to communicate to the LiU-IT any IT-problems you experience during the exam. To this end, you can request to the exam watchers to contact the IT-support team, by phone (call 013-282828).

You cannot claim that any exam outcomes are due to IT-problems which were not reported to the LiU-IT support during the exam.

Instructions: This is an **individual examination**. Use a computer in the lab room to log in with your username and password. Follow the instructions given by LiU-IT. You can find the exam files in the folder `Exam workspace`. Start by making a backup copy of the contents of this folder, which can be useful in case you need to get back any of the original files distributed with the exam.

In the folder `code` (located in `Exam workspace`), there is a CMake file which you can use to create a Visual Studio solution project for the exercises in the exam (as for the labs in the course).

You must write your name and personal number in the beginning of every source code file, in a comment line. Otherwise, your exam is not graded.

Follow the specifications, design your solutions with care, and use C++ properly.

Delivering your solutions: **Deliver one file for each exercise**. Copy the folder `code` with your files for the exercises into the folder `Exam hand in`.

Executable files (`.exe`), Visual Studio files, and CMake related files should not be delivered. The `build` folder should not be copied.

Do not turn off the computer at any time. Log out when you have finished the exam.

Exam points: *3p*

Good luck!

Additional information

If you find any open questions in the presented exercises then feel free to make your own decisions. However, your decisions should be reasonable and must not contradict any of the conditions explicitly written for the exercise. Please, write comments in the programs that clarify your assumptions/decisions, if any.

The final grade in this course can be either **3**, or **4**, or **5**. Below you can find how your course grade is decided. For more information, please read the course info available from the course website.

This final computer exam has three parts. Each part awards at most 10 points. Note that the exam's **Part III** only awards points if the programs run and produce sensible output.

Final Grade	Requirements
3	Minimum of 8 points in exam's **Part I**.
4	Fulfill the requirements for grade 3 and you must have at least • 6 points in **Part II** or • 5 points in **Part III**.
5	Fulfill the requirements for grade 3 and you must have at least 8 points in **Part III**.

If you have already been approved in the course but want to improve your grade ("*plussa*") then you only need to do the exercises in **Part II** and/or **Part III** of the final computer exam.

To get full points in an exercise, the following conditions must be satisfied.

- Readable and well indented code.
- No global variables can be used (but, global constants are accepted).
- The compiler must not issue warnings when compiling your code.
- Only ISO C++ statements are accepted.
- Your programs must respect the given specifications.
- Your programs must pass all test examples shown in this paper.
- Respect for the submission instructions.

It is advisable not to ignore the compiler warnings. Two help documents are distributed with this exam.

- `CMake-short-guide.pdf`: a short guide for CMake. Pay special attention to the paths indicated in this guide for the `build` and `code` folders.
- `aid-VS.pdf`: a short guide for Visual Studio IDE (how to set the compiler to compile C++17, etc).

Note that DrMemory does not produce reliable diagnostics for code compiled with Microsoft Visual C++ compiler.

Part I

Exercise 1 [2p + 3p = 5p]

In the file `exerc1.cpp`, you can find the definition of a (simple) class `List` to represent a singly linked list such that each node stores an integer, similar to lab 1. Notice that a list may contain repeated values and it does not have to be sorted. In addition, the class is equipped with a copy constructor. Study the given class and add to it the functionality described below.

a. A constructor such that, given the number of values $\boldsymbol{n \geq 0}$ to store in the list, a starting value, and an increment (or decrement) step, creates a list storing $\boldsymbol{n}$ values satisfying the following conditions:

- The list's first value is the given start value.
- The difference between $(i + 1)$th and the ith values in the list (i.e. two consecutive elements) is equal to the given step.

Some examples are given below (and you can also find several examples in the file `exerc1.cpp`). First, argument is number of values n, second argument is the start value, and the third argument is the step.

```
List L0;            // The list is empty!
List L1(4);         // L1 = 0 -> 1 -> 2 -> 3
List L2(4, 2);      // L2 = 2 -> 3 -> 4 -> 5
List L4(4, 2, -2);  // L3 = 2 -> 0 -> -2 -> -4
```

As you can see in the examples above, if the constructor has no arguments then an empty list is created. If the starting value is not provided then it's assumed to be zero. And by default, the step is set to one.

After solving this exercise, your program should be ready to pass `Test 1` in the `main`.

b. A member function `symmetric_difference` such that `L1.symmetric_difference(L2)` returns a sorted list with the elements that are found in either `L1` or `L2`, but not in both lists. Assume that both lists `L1` and `L2` are sorted increasingly. Neither `L1` nor `L2` are modified by this member function.

For instance, if list $\boldsymbol{L1 = 2 \rightarrow 3 \rightarrow 4 \rightarrow 5}$ and $\boldsymbol{L2 = 2 \rightarrow 4 \rightarrow 6 \rightarrow 8}$ then

```
L1.symmetric_difference(L2);
```

returns a new list $\boldsymbol{3 \rightarrow 5 \rightarrow 6 \rightarrow 8}$.

After solving this exercise, your program should be ready to pass `Test 2` in the `main`.

Add your code to the file `exerc1.cpp`. The `main` function must be delivered unchanged. The expected output is available in the file `ex1_out.txt`.

Exercise 2 [5p]

Define a class `Log2` that represents logarithms of base 2. An instance of `Log2` should store an integer value (`unsigned long`). The class `Log2` should have the following functionality.

- It should be possible to create instances of `Log2` as shown in the examples below.

```
Log2 l1{ 1 }; //l1 = log2 1
Log2 l2{ 5 }; //l2 = log2 5
```

- Implicit conversion from integer values to instances of `Log2` class should not be supported.
- Instances of class `Log2` should be copyable and assignable.
- A member function `eval` that returns the value of the logarithm represented by the object. For instance, `l1.eval()` should return 0.0. The function `std::log2`, available in the `cmath` library, can be useful to implement this member function.
- The following operators should be overloaded.
  - `operator+` for adding two `Log2` instances and a corresponding add-and-assign `operator+=`.
  - `operator++`, in both prefix and postfix form, to add one to a `Log2` object. For instance, `++l2` should be mathematically equivalent to the expression $1 + \log_2 5$.

Recall the following properties of logarithms[1].

- $\log_b x + \log_b y = \log_b(x \times y)$
- $1 = \log_b b$ (thus, $1 + \log_b x = \log_b b + \log_b x = \log_b(b \times x)$)

In addition, the class should satisfy the following requirement: member functions to be allowed and the compiler can provide should not be implemented.

Add your code to the file `exerc2.cpp`. This file already contains a skeleton for class `Log2`, a stream insertion operator, and a `main` function to test the class. The `main` must be delivered unchanged. The expected output is available in the file `ex2_out.txt`.

[1] In this exercise, the base of the logarithms is 2 (i.e. $b = 2$).

Part II

Exercise 3 [2p]

Give your answer to the following question in the file `theory_exerc3.txt`. Answers should be given in your **own words**. Be clear and concise. Answers can be given in Swedish or English.

Consider the following declaration of a member function for a class `C`.

```
void C::fun(const C &x) const;
```

Describe briefly what does it imply the use of each of the `const`-qualifiers in the declaration above. Motivate your answer.

Exercise 4 [1 + 1 + 2 + 2 + 2 = 8p]

Write a program that performs the steps described below, by the given order. The components and algorithms of the standard template library (STL) should be used. No hand-written loops (`for`, `while`, `do-while`, or range-based loops) can be used. `std::for_each` cannot be used, either.

1. Read the file `numbers.txt` and store the integers read in a vector `V`. Then, display the number of values read and the values stored in `V`.
2. Compute the median of the values in `V` and display it. Vector `V` can be modified, if needed. Note that the median of, for instance, 1 3 5 **7** 8 9 11 is 7, while the median of 1 3 **4 6** 7 8 is $(4 + 6)/2 = 5$.
3. Copy the values stored in `V` into two suitable containers `C1` and `C2` such that `C1` contains all odd values and `C2` contains all even values. Both `C1` and `C2` should be sorted and have no repeated values.
4. Display the values in the container `C1` in decreasing order. Then, display the values in container `C2` in increasing order.
5. Copy all even values in `C2` which are smaller than any of the odd values in `C1` to another vector `V1`. Then, display all values in `V1`.

Add the code to the file `Exerc4.cpp`. The expected output is available in the file `ex4_out.txt`.

Part III

Exercise 5 [2p]

Give your answer to the following question in the file `theory_exerc5.txt`. Answers should be given in your **own words**. Be clear and concise. Answers can be given in Swedish or English.

Indicate in which case(s), it's required to make the destructor virtual. Motivate your answer.

Exercise 6 [8p]

Design a polymorphic class hierarchy for representing documents. The documents hierarchy consists of classes `Document`, `Email`, and `Report`, although other types of documents may be added in the future (e.g. letter, CV).

Class `Document` is an abstract base class. This class should store the document's text (`string`). Moreover, it should be possible to associate a number of keywords with a document. Each keyword is a `string`, like "*exam*". This class should offer the following functionality.

- Default constructor.
- It should be possible to specify the document's text when creating a `Document` object.
- Functions `getText` and `setText` that return and set the document's text, respectively.
- A function `add_keyword` that associates a new keyword with the document. Assume that keywords are always given in lower case letters (e.g. "*exam*").
- A stream insertion operator (`operator<<`) that writes the document to a given stream. This function should start by writing the type of document (e.g. "*Email*", "*Report*") and its associated keywords, if any. Obviously, this function may have different output depending on the type of document.
- A function `list_documents_by_key` that displays all documents associated with a given keyword, if any.
- Instances of class `Document` cannot be copied nor assigned.

Class `Email` is a subclass of `Document`. Instances of `Email` should store the sender, the receiver, and email's subject, in addition to the text message. When creating a new email, the sender, the receiver, and the subject must be specified, although the text message may be left unspecified. Thereafter, neither the email's subject, nor the sender, nor the receiver can be modified. The stream insertion operator should also write the email's sender, receiver, subject, and text message, after the information indicated above for documents.

Class `Report` is a subclass of `Document`. A report has an author, a title, besides the report's text. When creating a new report, the author and the title must be specified. Thereafter, it should not be possible to change the report's author. The stream insertion operator should also write the report's author, title, and the first line of the report's text, after the information indicated above for documents.

Design your classes with care and follow the given specifications. For each of the classes above, no other public member functions other than those described are allowed[2]. Make use of good programming practices in your code. Unnecessary code duplication will incur point deductions.

Add your code to the file `exerc6.cpp` that also contains a `main` function for testing. Notice that you need to add some code in the `PHASE 4` part of the `main` (clearly marked with "`//ADD CODE`"). The `main` should be delivered unmodified, otherwise.

The expected output is available in the file `ex6_out.txt`. Note that we cannot guarantee that your solution is correct just because it passes the given tests.

[2] You can add non-public member functions, though.