Linköping University/ITN

12 January, 2022
08:00 – 13:00

TNG033: computer examination
Programming in C++

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.

Forbidden to post anywhere information about the exam until 12 of January 2022, 17:00.

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`. 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.

Tentavisning:

Time and place are e-mailed to students registered on the exam, when the grading is finished.

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. Study the given class and add to it the functionality described below.

a. A constructor such that given two vectors of integers with same size[1], `V1` and `V2`, creates a list with the elements in `V1` such that each element `V1[i]` ($0 \leq i < V1.size()$) appears consecutively `V2[i]` times in the list. Note that the elements of the list should appear by the same order as in `V1`. For example,

```
List L(std::vector<int>{3, 5, 7}, std::vector<int>{2, 1, 2});
```

creates the list $\boldsymbol{L = 3 \rightarrow 3 \rightarrow 5 \rightarrow 7 \rightarrow 7}$.

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

b. A member function `unique` such that `L.unique()` removes all consecutive duplicate elements from list `L`. Only the first element in each group of equal elements is left in the list.

For instance, if list $\boldsymbol{L = 1 \rightarrow 1 \rightarrow 6 \rightarrow 6 \rightarrow 5 \rightarrow 10 \rightarrow 10}$, then after executing

```
L.unique();
```

list $\boldsymbol{L = 1 \rightarrow 6 \rightarrow 5 \rightarrow 10}$.

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`.

[1] Your function can assume that both argument vectors, `V1` and `V2`, have the same size.

Exercise 2 [5p]

Define a class `DayOfWeek` to represent the seven days of a week, i.e. *Sunday*, *Monday*, to *Saturday*. Each instance `d` of this class stores an integer index such that 0 corresponds to *Sunday*, 1 corresponds to *Monday*, …, and 6 corresponds to *Friday*. This idea is already implemented in the code given for the class, see file `exerc2.cpp`.

Add to class `DayOfWeek` the following (basic) functionality.

- A constructor that creates an instance of class `DayOfWeek` from a given string (`std::string`). An example is shown below.

  ```
  DayOfWeek tng033_exam_day{"Wednesday"};
  ```

  If the given string is not one of those given below then the instance of class `DayOfWeek` created should correspond to *Sunday*.

  ```
  "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"
  ```

- Instances of class `DayOfWeek` should be copiable and assignable.
- Pre-increment and pre-decrement operators. For instance, `++tng033_exam_day` evaluates to the week's day *Thursday*, while `--tng033_exam_day` evaluates to the week's day *Tuesday*.
- Addition of an instance `d` of `DayOfWeek` and an integer $n \geq 0$ such that expressions `d+`n and n`+d` represent the day of the week n days after `d`. For instance, `tng033_exam_day + 5` evaluates to the week's day *Monday*.
- Subtraction of an integer $n \geq 0$ from an instance `d` of `DayOfWeek` such that the expression `d-`n represents the day of the week n days before `d`. For instance, `tng033_exam_day - 5` evaluates to the week's day *Friday*. Note that expressions of the form n`-d` should not compile.

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 `DayOfWeek`, 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`.

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 each of the `const`-qualifiers in the declaration above. Motivate your answer.

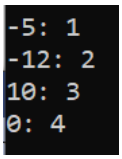
Exercise 4 [2 + 2 + 1 + 1 + 1 + 1 = 8p]

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) can be used.

The text file `integers.txt` contains numbers that correspond to integers and the same integer may occur several times in the file, though written in different ways. For instance, $+0$, -0, and 00 represent integer zero, while $+10$ and 0010 represent integer ten.

1. Read all numbers in `integers.txt` and create a table that, for each integer v in the file, stores all different formats of v occurring in the file. The table should be sorted increasingly by the integer values. Use a `std::map` to represent the table.
2. Write the table created in step 1 to `std::cout`.
3. Copy the table created in step 1 into a vector (`std::vector`). Do not use `std::for_each`.
4. Sort the vector created in step 3 increasingly by the number of different ways (formats) the integers values occur in the file.
5. Write the sorted vector to `std::cout`.
6. Write to `std::cout` each integer value v in the file and the number of different ways v occurs in the file, sorted increasingly by the latter. For the given file, this step should produce the following output (e.g. -5 occurs in the file in one format, while -12 appears written in the file in two different ways). Do not use `std::for_each` nor an extra container such as a vector, a map, a list, or a set.

   [**Hint**: convert each element in the sorted vector to a string].

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.

Why should one explicitly state in the code that a member function overrides a virtual function from a base class? Motivate your answer by giving a concrete (simple) example.

Exercise 6 [8p]

Design a polymorphic class hierarchy for representing Boolean expressions. The truth values `T` (representing *true*) and `F` (representing *false*) are Boolean expressions. By using Boolean operators, such as "∧" (also known as `AND`[2]), "∨" (also known as `OR`[3]), or "¬" (also known as *negation*[4] or *not*), one can build Boolean expressions such as `T ∧ F`, `(T ∧ F) ∨ T`, `¬ (T ∧ F) ∨ (T ∧ F)`, `¬T ∧ F`. The table given at the end of this exercise description shows how these operators are evaluated.

`Boolean_expression` is the base class of all classes representing Boolean expressions. Boolean expressions should have the following basic functionality.

- A public member function `evaluate` that evaluates the expressions and returns a `bool`.
- A public member function `clone` shall be the only public way to make a copy of a Boolean expression. This function shall create a dynamically allocated object and return a pointer to the new object. Note that Boolean expressions should not be assignable, either.
- An overloaded stream insertion operator that writes a Boolean expression to a given output stream.
- It should not be possible to create instances of `Boolean_expression` which are not instances of one of its sub-classes.

`Truth_value` is a sub-class of `Boolean_expression` representing truth values. Instances of this class store a value of type `bool`. When an instance of this class is initialized, a `bool` value should be given which is stored in the object and, thereafter, it cannot be modified. For instance, `Truth_value ff{false};` creates an instance of this class named `ff`.

`Not` is a sub-class of `Boolean_expression` representing a negated Boolean expression. This class should have a constructor that, given a boolean expression e, creates a Boolean expression representing $\neg e$.

`Binary_bool_expression` is a sub-class of `Boolean_expression` representing expressions obtained by applying a binary Boolean operator to two Boolean expressions. It should not be possible to create instances of this class that are not instances of one of its sub-classes.

`_AND_` is a sub-class of `Binary_bool_expression` and it should have a constructor that, given two boolean expressions e_1 and e_2, creates a Boolean expression representing $e_1 \wedge e_2$.

`_OR_` is a sub-class of `Binary_bool_expression` and it should have a constructor that, given two boolean expressions e_1 and e_2, creates a Boolean expression representing $e_1 \vee e_2$.

Design your classes with care and follow the given specifications. For each of the classes above, no other public member functions other than those explicitly required in this exercise should be added[5]. In addition, special member functions to be allowed, and the compiler can generate, should not be implemented.

---

[2] Other possible used designations are "`&&`" or "`*`".
[3] Other possible used designations are "`||`" or "`+`".
[4] Other possible used designation is "`!`".
[5] You can add non-public member functions, though.

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. Feel free to add any other tests, but the solution you hand in must contain the `main` function delivered with this exam (thus, delete any extra test code you may have added).

Note that we cannot guarantee that your solution is correct just because it passes the given tests.

e_1	e_2	$e_1 \wedge e_2$	$e_1 \vee e_2$	$\neg e_1$
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T