Linköping University
Department of Science and Technology/ITN
Aida Nordman

Introduction to Programming (C++)

TND012

Laboration 6

Course goals

- To define new data types.
- To divide the program code into header files (`.h`) and source files (`.cpp`).
- To write programs with basic file processing.
- To structure programs with functions.

Preparation

This lab has two exercises. Exercise 1 is mandatory while exercise 2 is optional. Exercise 2 is an extension of exercise 1 and is slightly more complex. Thus, you need to complete exercise 1 before you start with exercise 2.

Note that you are strongly advised to solve exercise 2 as well. The difficulty of exercise 2 is similar what you will experience in Part III of the exam. Recall that you must complete Part III of the exam if you want to get the highest grade (i.e. 5).

Your source files should be placed in the folder `TND012\Labs\Lab6`.

Preparation for exercise 1

1. Review the concepts and code examples discussed in lecture 10-12. For instance, you should review the employee examples presented in these lectures.
2. Review the exercises of lesson 6.
3. Read both exercises, exercise 1 and exercise 2.
4. Download the file `Code6.zip` and unzip it into `TND012\Labs\Lab6`. For this exercise you are going to work with the files: `movies.h`, `movies.cpp`, `movies_database.cpp`, and `movies.txt`.
5. Follow the steps a. to h.1 (including task h.1), as described in steps for exercise 1 before your lab session in week 41.
6. **Only if you plan to work on exercise 2**, steps h.2, h.3, and h.4 must be completed before the lab session in week 41 as well.

Preparation for exercise 2

1. Follow the preparation instructions given above for exercise 1.
2. After doing all steps in exercise 1, make a copy of all files (i.e. `movies.h`, `movies.cpp`, `movies_database.cpp`) before you proceed.

3. Review how to make a linear search in a sorted array. This was discussed in lecture 5 and lecture 8 (see slide 15).
4. Implement also options 1, 2, and 3 described in exercise 2 before your lab session on week 41.

Presenting solutions and deadline

The exercise in this lab is compulsory and you should demonstrate your solution during your lab session latest on **week 42**. Note that the course ends on week 42.

We remind you that your code for the lab exercises cannot be sent by email to the staff and that at most two labs can be presented in a lab session.

Before presenting your code make sure that it is readable, well-indented, and that **the compiler issues no warnings**[1]. Moreover, **you are not allowed to use global variables**, i.e. all variables must be either declared inside the **`main`** or inside a function. Otherwise, your lab won't be approved.

If you have any specific question about the exercise, then send us an e-mail. Be short and concrete, otherwise you won't get a quick answer. You can write your e-mail in Swedish. Add the course code and your study programme to the e-mail's subject, e.g. "**`TND012/ED:`** …".

Exercise 1

You have been hired to create a database for a small movie rental shop. For each movie, the following information should be stored in the database.

- Movie name (`string`).
- Number of copies owned by the shop.
- Movie type (`string`), i.e. "`action`", "`drama`", "`comedy`", "`horror`".

The aim of this exercise is to write a program which reads in a text file containing information about movies (described below), then loads the file into an array, and sorts the array alphabetically by movie names. Finally, the program prints the list of sorted movies on the screen and also saves the sorted list into a new text file (named `movies_sorted.txt`).

The text file `movies.txt` contains data about several movies and is structured as follows.

```
Movie name
Type Number_of_copies
Movie name
Type Number_of_copies
…
```

Note that the name of a movie may consist of several words like *"Blade Runner"*. Assume that the file `movies.txt` contains at most 100 different movies.

[1] Follow the instructions given in the appendix of lab 1.

The file `movies.h` contains the declaration (prototypes) of the basic functions available to manipulate a movie. Movies are represented by a new data type `Movie`. The file `movies.cpp` should contain the implementation of every function declared in `movies.h`. Finally, the file `movies_database.cpp` contains the declaration of several functions and a `main` function to perform the actions described above.

In this exercise, you are **not** allowed to change or add function declarations in the header file `movies.h`. Additionally, the `main` function in `movies_database.cpp` must **not** be changed. The existing function declarations in `movies_database.cpp` must remain unchanged as well but you are allowed to add your own functions there.

Steps for exercise 1

Perform each of the steps below in the given order and do not skip any step.

a. Since your program is composed of several files you first need to create a Code::Blocks project which takes care of compiling all `.cpp` files, linking, and generating the executable. Therefore, follow the steps given in the appendix. After creating a project, you press `F9` and Code::Blocks will compile your program, link it, and generate an executable.
However, the skeleton you downloaded from the webpage cannot yet be compiled.

b. Copy `movies.txt` into the same folder as the Code::Blocks project.

c. Add a new datatype named `Movie` to the header file `movies.h`. This datatype should represent a movie as described above.

d. You should now be able to compile the program (press `F9`). Fix the compilation errors, if any. Obviously, the program does nothing useful, yet.

e. Carefully read the comments in front of each function declaration in `movies.h`. These comments indicate what each function is supposed to do. For example, function `get` reads the data for a movie from a given file. In case you are wondering about how these functions are used, you need to wait until step h.

f. Your next task is to add the implementation for the functions `get`, `put`, and `larger_than` in `movies.cpp`. Follow the specifications (i.e. see the comments before each function). Remember to add the implementation of each function, one at a time, and then compile the file `movies.cpp`. If there are any compilations errors then you should correct the code before you proceed with the implementation of the next function.

   - Note that the file `movies.cpp` does not contain any `main` function. Thus, you cannot generate an executable from it. However, you can compile the current file, i.e. `movies.cpp`, with the key combination `Ctrl-Shift-F9`.

g. In `movies_database.cpp`, you will find a small application program that reads in a text file of movies, named `movies.txt`, and creates a database of movies (i.e. an array of movies). The database is then sorted alphabetically by movie name. Finally, the sorted database is displayed on the screen and is also saved into a new text file named `movies_sorted.txt`. Read and understand the algorithm implemented in the `main` function. Note that the `main` function in `movies_database.cpp` is already finalized and cannot be modified.

h. As you probably noticed, the `main` function calls the functions `read_from_file`, `sort_movies`, `write_to_file`, and `display_DB`. These functions have been

declared in the file `movies_database.cpp` and each function declaration is preceded by a comment specifying what the function is supposed to do.

1. Carefully read the comments in front of function `read_from_file` and implement it. Function `get` declared in `movies.h` can be useful here. Compile and test the program (press `F9`). Fix any compilation or execution errors.
2. Carefully read the comments before function `sort_movies` and implement it. The Boolean function `larger_than` declared in `movies.h` can be useful here. Compile and test the program (press `F9`). Fix any compilation or execution errors.
3. Read the comments in front of function `display_DB` and implement it. One of the overloaded functions `put` (declared in `movies.h`) can be useful here. Compile and test the program (press `F9`). Fix all errors.
4. Carefully read the comments before function `write_to_file` and implement it. One of the overloaded functions `put` (declared in `movies.h`) can be useful here. Compile and test the program (press `F9`). Fix all errors. Check whether the text file `movies_sorted.txt` was created and has the right content.

Congratulations! Exercise 1 is now ready.

Exercise 2

This is your last lab exercise in this course. LiU aims at you become a professional and independent programmer. Therefore, we are not giving you detailed instructions of how to proceed in this exercise. Obviously, you should follow a stepwise approach as in the previous exercises.

Problem description

Customers can rent movies and you are now requested to write a program that keeps track of the movies rented out. Assume that the shop has at most 4 copies of each movie, though this limit can be changed in the future.

For each rental, the program keeps information about the personal number (ID) of the customer and which movie she has rented. Personal numbers are positive integers. Your program should start by loading a database of movies from the file `movies.txt`, as described in exercise 1. The database should be able to store up to 100 different movies. For simplicity, assume that every time you start the program the same file `movies.txt` is read as if it is the first day of the shop.

The program displays a user menu with the following options:

1. Display information about all movies owned by the shop, i.e. name, number of copies, type, and how many copies are rented out. The displayed information should be sorted alphabetically by movie name.
2. Display the personal number of all customers who are renting a given movie. If the given movie does not exist in the database then an error message should be displayed.
3. Rent a movie given the movie name and the customer personal number. If the movie does not exist, or there are no copies available for renting, then display an error message.
4. Return a movie to the shop given the movie name and the customer personal number. If the movie does not exist in the database, or the personal number does not correspond to any of the customers who are currently renting the movie, then display an error message.
5. Exit the program.

Before exiting, your program must write all data on the movies (movie name, type, number of copies, numbers of current rentals, and ID of who is renting the movie) to a text file named `movies_sorted.txt`. The movies should be sorted by alphabetical order of the movie's name.

A small example is shown below. You should first read the left column and then right column on the same page.

In this exercise, you are allowed to add new functions to the file `movies.h`, if needed. Note that the idea is that all functions in this header file represent a certain functionality related to one movie (like to read one movie from a file).

Note that the quality of your solution depends on how your program is structured. For instance, small functions performing simple tasks and a simple `main` increase the quality of the code. Code readability and how easy it is to modify the code to accommodate simple changes in the described problem are other criteria for evaluation of your solution.

```
*****************************
1. Display all movies
2. Display rentals of a movie
3. Rent a movie
4. Return a movie
5. Exit
*****************************
Option ? 1

Artur       2 comedy  rented: 0
Matrix      3 action  rented: 0
Zombies     4 horror  rented: 0
Fight_Club  4 drama   rented: 0
Seven       1 drama   rented: 0
Up          1 comedy  rented: 0
```

```
*****************************
1. Display all movies
2. Display rentals of a movie
3. Rent a movie
4. Return a movie
5. Exit
*****************************
Option ? 3
Movie name? Up
Personal number? 1
```

```
*****************************
1. Display all movies
2. Display rentals of a movie
3. Rent a movie
4. Return a movie
5. Exit
*****************************
Option ? 3
Movie name? Up
Personal number? 2

No copies available!!
```

```
*****************************
1. Display all movies
2. Display rentals of a movie
3. Rent a movie
4. Return a movie
5. Exit
*****************************
Option ? 3
Movie name? Artur
Personal number? 5
```

```
*****************************
1. Display all movies
2. Display rentals of a movie
3. Rent a movie
4. Return a movie
5. Exit
*****************************
Option ? 1

Artur       2 comedy  rented: 1
Matrix      3 action  rented: 1
Zombies     4 horror  rented: 0
Fight_Club  4 drama   rented: 0
Seven       1 drama   rented: 0
Up          1 comedy  rented: 1
```

```
*****************************
1. Display all movies
2. Display rentals of a movie
3. Rent a movie
4. Return a movie
5. Exit
*****************************
Option ? 3
Movie name? X_man
Personal number? 2

Non existing movie!!
```

```
*****************************
1. Display all movies
2. Display rentals of a movie
3. Rent a movie
4. Return a movie
5. Exit
*****************************
Option ? 3
Movie name? Artur
Personal number? 10
```

```
*****************************
1. Display all movies
2. Display rentals of a movie
3. Rent a movie
4. Return a movie
5. Exit
*****************************
Option ? 2
Movie name: Artur

Customer: 5
Customer: 10
```

```
******************************
1. Display all movies
2. Display rentals of a movie
3. Rent a movie
4. Return a movie
5. Exit
******************************
Option? 4
Movie name? Up
Personal number? 1
```

```
******************************
1. Display all movies
2. Display rentals of a movie
3. Rent a movie
4. Return a movie
5. Exit
******************************
Option ? 1

Artur      2 comedy  rented: 2
Matrix     3 action  rented: 1
Zombies    4 horror  rented: 0
Fight_Club 3 drama   rented: 0
Seven      1 drama   rented: 0
Up         1 comedy  rented: 0
```

```
******************************
1. Display all movies
2. Display rentals of a movie
3. Rent a movie
4. Return a movie
5. Exit
******************************
Option? 4
Movie name? Artur
Personal number? 15

Renting not found!!
```

```
******************************
1. Display all movies
2. Display rentals of a movie
3. Rent a movie
4. Return a movie
5. Exit
******************************
Option ? 9

Wrong option!!
```

```
******************************
1. Display all movies
2. Display rentals of a movie
3. Rent a movie
4. Return a movie
5. Exit
******************************
Option ? 5

Goodbye!
```

Appendix

To create a project in Code::Blocks which contains all necessary files follow the steps below.

1. Make sure you have downloaded the files `movie.h`, `movie.cpp`, and `movies_database.cpp` and placed them in `TND012\Labs\Lab6`.
2. Start Code::Blocks.
3. Select `File` → `New` → `Project`→ `Console application`
4. Follow the wizard (choose `C++` option, give a name to the project, and place it in the folder `TND012\Labs\Lab6`).
5. After the wizard has terminated, click on `Sources`, on the left pane. Then, you should be able to see a file named `main.cpp`.
6. Right-click on `main.cpp` → `Remove file from project`
7. Right click on the project name on the left pane → `Add files…` → select the files `movies_database.cpp, movies.h` and `movies.cpp` → a new window opens → click `OK`
8. You can now see all the `.cpp` and `.h` files of the project under the project name, on the left side pane.

You should also have a new folder created by `Code::Blocks` named with the project name. Any text files read by the program must be place inside this folder.