APT Lab 1 Exercises

Prof. Dr. Michael Lipp <Michael.Lipp@h-da.de>

Version 1.3.1, 08.11.2022

Preface

These are your first exercises for the APT Lab. As outlined in the "Organization" section of the Moodle course for this lab, you can achieve up to three points for each exercise, resulting in a total of 27 points for the complete lab. You need 40% to pass, i.e. 11 points.

Talents differ. Maybe programming isn't your favorite subject and you have to spend a great deal of time to complete the exercises. In this case keep in mind that not everybody has to pass this lab with the highest score. If it takes you too much time to complete all exercises, try to complete two thirds of each exercise (or two out of three) and you will still achieve 18 points in total and easily pass the lab.

Of course, the exercises will gradually become more difficult as your knowledge increases during the course, so play it safe and don't be sloppy with these first exercises.

Make use of the open labs! As the rules state, "mandatory exercises on these dates is restricted to questions regarding the understanding of the description of the exercises. There will be neither hints regarding the solution nor a review of your solution for a mandatory exercise before that exercise has been submitted"; well it has to be like that, else you'd eventually all end up with the supervisor's idea of the best solution.

But, of course, if there is e.g. a compilation problem that you cannot solve, you will receive help. And if you haven't fully understood a concept from the lecture, the open lab provides an opportunity to get some additional explanation, usually using an example that is different from the one used in the exercises.

During the review labs, there will be time for a short review only. The main purpose is to evaluate your points, not to give you detailed feedback regarding your work. So don't start to discuss things or ask questions regarding how to improve your programming during the review lab. But, by all means, come to the next open lab after the review and discuss your solution and obtain a more detailed feedback what to do better in the exercises to come.

Rules and hints

General rules

- Including `<windows.h>` or `<conio.h>` results in grading with 0 points for the complete submission. These files contain Windows-specific function declarations that make your code less portable and are never required to solve the exercises in this lab.
- The maximum line length is 80 characters. This also applies to comments. There is an Eclipse setting that allows you to display a vertical bar after column 80. Violating this rule results in 0 points for the submission.
- Local variables must not be declared earlier than required. Typically, they are declared (and initialized) when there initial value is known. Declaring all (or most) local variables at the beginning of a block without necessity results in 0 points for the submission.

Rules for this exercise

This exercise consists of three parts and you will therefore create and submit three projects (exported in a single zip-file, simply check all projects in the export dialog).

Nevertheless, the rule from the section "Organization" in the Moodle course which states that compilation errors result in 0 points applies to the complete exercise. Even if only a single project has compilation errors, the complete exercise will be graded with 0 points.

Exercise 1.1: Board for Reversi

In the pre-course participants implemented a program that can serve as the basis of a board game. (In case you didn't participate, please find the original task description in the appendix.) In this exercise you are to implement a board for playing Reversi (https://en.wikipedia.org/wiki/Reversi). (There are several browser based implementations online, e.g. https://cardgames.io/reversi/, that can give you an idea of the game.)

Write a class `ReversiBoard` that represents the state of the board, i.e. which fields are empty, occupied with a piece with player 1's color up or occupied with player 2's color up. Use an enumeration to represent the fields' states.

The board class must provide a method for querying any field's state and a method for setting a field's state, i.e. putting a new piece on the board. This latter method must check that the move is valid (the piece may be put at the given location) and update the state of the board ("turn over" other pieces as required by the rules).

Write a second class `ReversiConsoleView` that prints the board on the console. The class receives a pointer to an instance of `ReversiBoard` as parameter of its constructor. In its `print()` method, it uses the pointer to obtain the states of the fields and creates the appropriate output.

In your `main` function, implement a small loop that queries the user where to put the next piece, invokes the method to put the piece on the board and prints the board's new state.

Provide hand-written (or hand-drawn) UML class diagrams for your classes.

Don't forget to provide proper doxygen style documentation as described in the "Organization" section of the Moodle course for this lab. In order to verify your documentation, you must generate a "web-site" from your source code.

Have a look at the uploaded source code from the second lecture in Moodle (lecture course). It includes a file `SimpleVector.doxyfile`. Copy this file into your project directory and rename it (e.g. to `Reversi.doxyfile`). Now right-click on your project and select the menu item "`@ Build Documentation`". You will see some output from the build process in the console window. When building has finished, there will be a new folder in your project called `html`. Within this folder, you'll find a file called `index.html`. Open it in your browser and you'll see the generated documentation.

Remember that this documentation is what another programmer reads when he wants to use the classes that you have developed. Therefore the documentation must be comprehensive. Imagine that you wanted to make use of your classes and all you had is this documentation (no source code). Verify that your documentation really includes all required descriptions and hints.

If you have problems generating the documentation go to the open labs and ask for help. Remember that the generated documentation must be presented during the review for all tasks from this exercise and all exercises to come.

Exercise 2.1: Modulo Counter

In the C-Precourse, you had the task of implementing a single digit and a multi digit modulo n counter (again, the task description can be found in the appendix). Now that you know about OOP, you can provide a much better solution, of course.

Implement the classes `ModuloNDigit` and `ModuloNCounter` that provide the same functionality as the `struct`s and their associated functions from the pre-course task (make sure that you have understood the difference between a digit and a number). Define constructors and destructors as substitutes of the functions with the corresponding purpose.

Additionally, provide the functionality to increment the values of instances of the two counter types by overloading the prefix and postfix increment operators, thus making it possible to write `counter++` and `++counter`. Observe the proper semantics of the two operators.

Execute the same test as in the pre-course task.

Provide UML class diagrams and documentation as required in the previous exercise.

Exercise 3.1: Heat Controller

Remember the heat(ing) controller used as an example of an object in the lecture? As mentioned, it is a "complex" object, because it can be composed from several components (component objects).

Decompose the heat controller, i.e. identify the components that it is built from. Use a decomposition by functional components. "PCB" or "Casing" are physical components and not the components that we are looking for (but for sure there is e.g. a functional component temperature sensor).

For each component (it should be easy to find at least five, seven are required for full points), define a class and provide an exhaustive (doxygen) documentation about of the component’s purpose. Identify the state information associated with the component and model it as data members of the classes. Add the capabilities, i.e. the methods that the classes provide (and provide documentation, of course).

Provide a hand-written UML class diagram of all classes including their relationships. Write a small "story" about what happens when the user increases the desired temperature by 5 degrees (which methods are invoked by which component).

Appendix A: Exercises from the Pre-course

Board game

Write a small program which serves as a starting point for a board game:

- The board is a char array with a size of x columns and y rows
- The size of the board is entered by the user
- Once the board is created, the user can enter characters at any position
- The board can be printed on the screen

Hints:

- C does not provide 2d char arrays. The 2d array is represented by an x*y byte memory.
- Write 3 functions clearBoard, printBoard and setBoard, taking a pointer to the board and additional parameters as needed.
- Test the program after every function

Example output:

```
The starting point of a board game
Please enter the size (x,y) of the game:
3 5
Allocated board of dimension x = 3, y = 5 at adress 672b08
. . .
. . .
. . .
. . .
. . .
Please enter a position (x,y) and a stone character (0 to end):
2 4 x
. . .
. . .
. . .
. . .
. . x
Please enter a position (x,y) and a stone character (0 to end):
1 3 o
. . .
. . .
. . .
. o .
. . x
Please enter a position (x,y) and a stone character (0 to end):
```

Single Modulo n Counter

Write a small program which uses a data structure to store the properties of a modulo n counter, which counts from 0 to n-1, just beginning at 0 again after having reached n-1 (overflow). Declare a data structure for counters with integer members for:

- the maximum count value (n)
- the current count value

Define a new type for the data structure using typedef.

Write functions which

- initializes a counter by setting the current count value to 0 and the maximum count value to an arbitrary value.
- counts, that is increase the current count value of a passed counter by 1 or reset it to 0, in case of the value becomes greater than n-1 (overflow). It returns 1 if an overflow had occurred otherwise 0.
- optionally prints the name of the counter (modulo n counter, n should be replaced by the current value of the maximum count value) or the current count value (without newline!). Use an enumeration type to pass the option.

Test the code with a modulo 10 counter, print its name, count three times until the overflow while printing the actual count value after each counting step.

Sample call:	Example output:

```
counter_t c1;
init_counterSD(&c1,10);
print_counterSD(&c1, COUNTER_NAME);
int i;
for(i=0; i < 31; i++) {
    print_counterSD(&c1, COUNTER_VALUE);
    count_SD(&c1);
    printf("\n");
}
```
	```
	1
	2
	3
	4
	5
	6
	7
	8
	9
	0
	1
	...
	```

Multi Digit Counter

The program from the previous exercise has to be expanded in order to handle not only counters with one digit but with m digits. For that, declare a new structure for a multiple digit counter, consisting of:

- a pointer for the address of an array of modulo n counters;
- the number of digits m (which is equal to the number of modulo n counters in the array).

Write additional functions which

- **initializes a multiple digit counter** by creating an array of m counters and storing the number of digits into the structure member. Additionally, the function initializes the counters of the array by setting their current count values to 0 and their maximum count value to one of the following values: 2 (binary), 8 (octal), 10 (decimal) or 16 (hexadecimal). The function returns 0 in case the maximum count value is not allowed or the number of digits is 0, otherwise 1.
  Hints: You have to pass the address of a multiple digit structure variable. Why?
  You have to allocate dynamic memory! Why?
  Of course, each counter of the array must have the same maximum count value.
- **increases the actual count value of a multiple digit counter** by 1 or resets it to 0 in case of an overflow.
- **optionally prints the name of the counter** (m digit binary/octal/decimal/hexadecimal counter, m and the type should be replaced by the current values) or the current count value of the m digit counter (with one space after the digits!). Use the enumeration type to pass the option.
  Note: Try to format the output into 8 columns if n equals to 8, 10 columns if n equals to 10 and 16 columns if n equals to 2 or 16. Consider that hexadecimal digits have values between 0 and 9 and 'A' and 'F'.
- **de-initializes a multiple digit counter** by releasing the memory of its array of counters and setting its members to NULL and 0 respectively.

Whenever it makes sense, call the functions of the modulo n counters from the previous exercise.

Test the code. The user enters the number of digits (m) and the maximum count value of the counter (n). The program, creates and initializes a multiple digit counter, prints its name and counts until the overflow occurs while printing the current count value after each counting step. If the user enters an invalid maximum count value, the program stops. Don't forget to release the memory!

```
Please enter the parameters of your counter:
number of digits: 4
type (2/8/10/16): 2

4 digit binary counter
0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
0000 0001

Please enter the parameters of your counter:
number of digits: 2
type (2/8/10/16): 16

2 digit hexadecimal counter
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
...
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
...
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
00 01

Please enter the parameters of your counter:
number of digits: 2
type (2/8/10/16): 8

2 digit octal counter
00 01 02 03 04 05 06 07
10 11 12 13 14 15 16 17
20 21 22 23 24 25 26 27
...
70 71 72 73 74 75 76 77
00 01

Please enter the parameters of your counter:
number of digits: 2
type (2/8/10/16): 10

2 digit decimal counter
00 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19
```

```
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99
00 01

Please enter the parameters of your counter:
number of digits: 0
type (2/8/10/16): 0

End of the test!
```