Högskolan Kristianstad
291 88 Kristianstad
044-20 30 00
www.hkr.se

DT555B Programming in C

Lab 4 Instruction

Objectives and Assessment

This final lab provides you with the practice on file input and output, arrays and pointers.

You only need to complete one task among the given options in this lab.

Task 1 Game of Life (need to download lab4-src.zip file from the course page)
Task 2 Computer Aided Instruction
Task 3 Advent of Code

Each task has extra requirement for grade 5.

Implementation

This lab is mandatory and individual. Discussion with your classmates are promoted. But the copy-and-paste is not allowed.

- You should find time yourself and develop algorithm(s) to solve problem(s), implement your solution, test your solution, and write a lab report.
- You need to upload the report in due date if you want your report being graded in time.
- AI tools can be used during development. Then you need to provide the conversation with AI tools in the report appendix to allow the instructor to evaluate your skills on algorithms. Keep in mind that if AI tools produce the main algorithms, it will be failed.

Task 1 Game of Life

Introduction to Game of Life

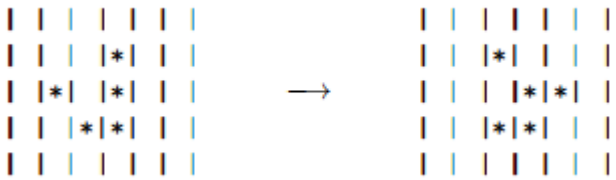
The Game of Life, invented by John Conway in 1970, is an example of a zero-player "game" known as a cellular automaton. The game consists of a two-dimensional world extending infinitely in all directions, divided into "cells." Each cell is either "dead" or "alive" at a given "generation." The game consists of a set of rules that describe how the cells evolve from generation to generation.

These rules calculate the state of a cell in the next generation as a function of the states of its neighboring cells in the current generation. In a two-dimensional world, a cell's neighbors are those 8 cells vertically, horizontally, or diagonally adjacent to that cell. Conway's set of rules are summarized as follows:

1. Any live cell with fewer than two live neighbors dies, as if caused by under-population.
2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by overcrowding.
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

In this lab task, we will be implementing Conway's Game of Life, with the minor restriction that our two-dimensional world is **finite**. The neighbors of a cell on the **edge (borders)** that are beyond the edge are assumed dead. You can read more about Conway's Game of Life on Wikipedia at http://en.wikipedia.org/wiki/Conway's_Game_of_Life.

Example (the so-called "glider"):



where * = alive, space = dead.

There are 3 sub-tasks. Sub-tasks 1-1 and 1-2 are required for grade 3, while all 3 sub-tasks are required for grade 5.

Task 1-1 Code Reading

In practice, you may work in a team so that you collaborate with your team members to develop the application; you may also take over some application and further develop the application. So this task will provide you with this practice.

The ability to read and understand a program is an important skill for the software developers. Code reading means the process to read the program source code and understand its operation. Code reading is relevant to program maintenance: a programmer must gain sufficient understanding of the code before he/she could design modifications to extend, adapt or correct the code. Also, code reading (of good, great programs) is an important way to improve the programming skills.

A general guide on finding out what a function is doing?

1. Guess by the function name

People usually name a function meaningfully.

2. Read the comment at the beginning of the function
   It is a good practice to provide the comment on a function: the input parameters, purpose and return value
3. Examine how the function is used
   It is about how the function is called and used
4. Read the code in the function body (implementation)
   Find the details inside a function (for possible improvement or re-use it)
5. Find external documentation
   Many library functions are provided with external (online) documentation.

One design on the game of life application is given in this lab, and the skeleton of the code is available on the course page (in the folder lab4). You need to download these files to work with this lab (lab4-src.zip):

1) lifegame.h
it is the header file, and contains some symbolic constants and prototypes of the functions in the application.

2) lifegame.c
it contains the declaration of the data structure, and some basic functions

3) lab4-task1.c
Task 1 Option 1 program. In Task 1, you work on this file (you need lifegame.c too. But you do not need to change anything there at the stage)

4) lab4-task2.c (grade 5)
Task 2 Option 1 program. In Task 2, you work on this file and lifegame.c

In this task, you do not need to read lab4-task2.c file. You focus on the solution given in lab4-task1.c and lifegame.c. The solution is not complete and you will implement the missing part in this task.

If you are given a solution in the form of programs, where to start, how to study the solution? You need to use pencil-and-paper approach. From a *.c file, write down the file name and list the functions declared in the file. From a *.h file, write down the file name and list which constants are defined in the file, which function prototypes are declared in the file. When this inventory work is done, it is time to read the C code. You start from the main( ) function, since the program starts to execute the first statement in the main( ) and ends with the last statement in the main( ). From the main( ), you find out the first level of the sub-problems and solutions. When you encounter a function calling, you go to read that function, and learn how the function solves the sub-problem, you may find that it contains callings to some other functions. Then it means the sub-problem is further divided into several sub- sub-problems. During this reading and analysis process, you need to draw the relationship among the functions on the paper. When you have completed the reading and analysis, you have the function listings and relationship diagram. Then you can formulate a top-design diagram for the solution.

The following questions should be answered in the report:

Question 1: How the world is represented and defined in the solution?

Question 2: Draw the top-down design diagram to show how the solution (lab4-task1.c) is constructed. Remember **that top-down design diagram is not a flowchart, it is a graph showing how the problem is divided into smaller sub-problems.**

Question 3: Critical reflection on this design of the solution. What are good and what could be done better?

Task 1-2 Implementing the Evolution

After you have completed Task 1-1, you can move on to implement the missing code for the solution in lab4-task1.c. You find the comments "TO DO:" in file lab4-task1.c.

If you use CodeBlocks IDE for the program development, you need to create a C **console application project**, and add the lab4-task1.c and lifegame.c to the source file folder in the project, *.h to the header file folder. If you have problems with this, please contact me or others for help/support. (note: you can read more about creation of a C project at http://www.dummies.com/programming/c/how-to-create-a-new-codeblocks-project-in-c/)

In the report, the following questions should also be answered:

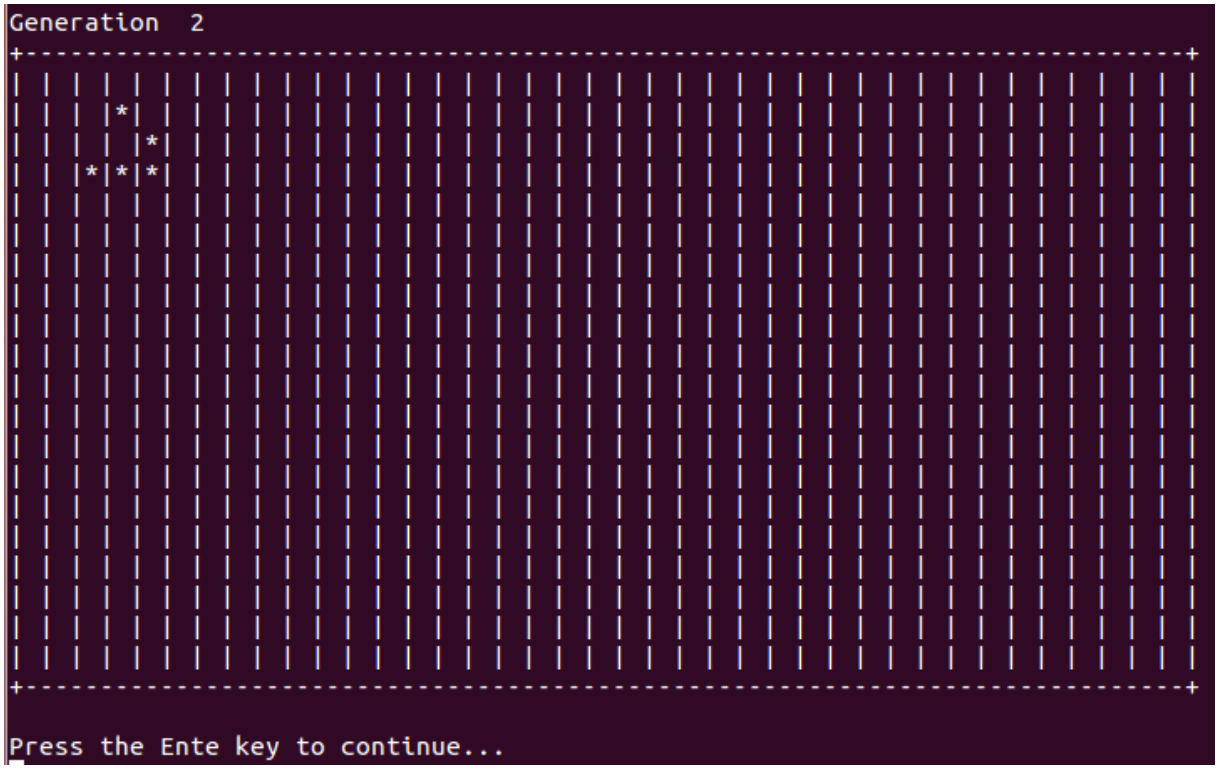
Question 4: In lifegame.c, you can find that the variables *world* and *nextstates* as well as some functions are declared to be **static**. What does the specifier "**static**" mean? Why it is used? *Read 4.6* in the English textbook (page 83) for the explanation, or 6.7.4 Lagrinsklassen static.

Question 5: Why the 2-dimenational array nextstates[][] is required?

Question 6: Read lifegame.h. What **symbolic constants** are defined? Why do we define these symbolic constants?

Test your solution to see if it works correctly. Followings are first 2 generations of the world:

Check the screenshots from my implementation:


Generation 2
Press the Ente key to continue...

Task 1-3 The World in a File (grade 5)

In lab4-task1.c, the initial state of the world was **hard-coded** into lifegame.c and the final state of the world was output to the console. In this Task, you will modify the code so that the initial state of the world can be read from a file, and the final state is saved to a file.

You need to implement 2 functions for input and output in lifegame.c: **initialize_world_from_file**(filename) and **save_world_to_file**(filename). You need also to fill in the missing code in main( ) of lab4-task2.c (the other function you implemented in Task 1-2 can be copied to this file).

a) Design the input.
In practice, most interesting patterns of the world have limited number of live cells. So it is more efficient to use a pair of numbers *row* and *col* to indicate the cell at which it is alive. For example, the glider pattern hard-coded in lifegame.c can be represented by the following sequence of numbers: 1 2 3 1 3 2 3 3 2 3. It means that cells at locations [1, 2], [3, 1], [3, 2], [3, 3] and [2, 3] are alive, while all others are dead.
You should implement the function **initialize_world_from_file( )** that reads the locations and sets the cells at these locations to be alive.
More interesting patterns can be found on the http://en.wikipedia.org/wiki/Conway's_Game_of_Life page.

b) Design the output
At the end of evolution, you should save the state of the world to a text file. You only need to save locations of live cells to the file.

c) Design the main function.
You work on file lab4-task2.c. Copy the for-loop and the functions you implemented in la4-task1.c to this file.
Then, you complete the main ( ). If the input file is not found, the program should default to initializing the world state to the hard-coded "glider" pattern.
Save the final output to the file "world.txt."

Task 2 Computer Aided Instruction Program

For those who want to extend the CAI program developed in earlier labs with more features, you can choose this task. There are two options for the task: grade 3 and grade 5 tasks.

Grade 3 Task: Simple CAI Program

This task continues the work specified in lab 3 Grade 3 Task Option 2 A simple CAI program. You need to add the following new features:

1) Save the test result in a text file
   Each time when a user completes a test, the test result and the username should be saved to a history file. Remember that you should only use one file.

2) Display the test history
   Add one more menu choice name "display test history" in the main menu. If the user chooses this choice, your program should print out a list of the test results: results and usernames. The output should be well-formatted.

3) Display the best result
   Add one menu choice named "best result" in the main menu. If the user chooses to display best result, your program should display the best result.

Grade 5 Task: CAI Program

This task continues the work specified in lab 3 Grade 5 Task Option 2. You need to implement the following new features:

1) Save the test result in a text file
   Each time when a user completes a test, the test result, test type (1 for additions, 2 for subtractions, 3 for mixed questions), the username and test **date** (year, month, day, hour, min, sec) should be saved to a history file. To get the current date, you can use C library functions **time( )** and **localtime( )**. You find them on Internet or in the appendix of the textbooks. One such a page is https://stackoverflow.com/questions/1442116/how-to-get-the-date-and-time-values-in-a-c-program.

2) Display the test history
   Add one more menu choice name "display my test history" in the main menu. If the user chooses this choice, your program should print out this user's test history (exclude other users history): results, test types, dates. The output should be well formatted.

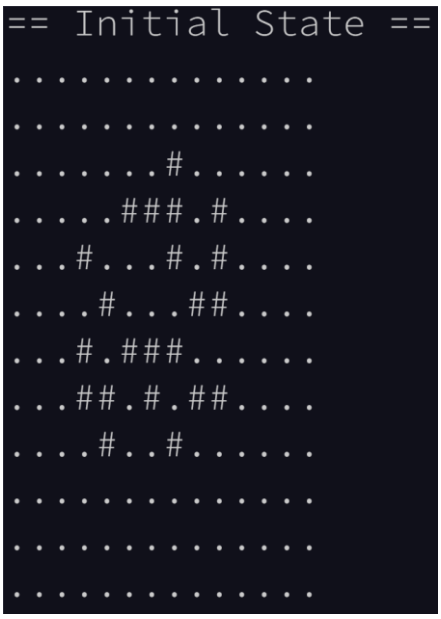
3) Display the top 5 results
   Add one menu choice named "display top 5 test results" in the main menu. If the user chooses to display top 5, your program should list the top 5 test results together with the question types, the usernames and test dates.

Task 3 Advent of Code (Grade 5)

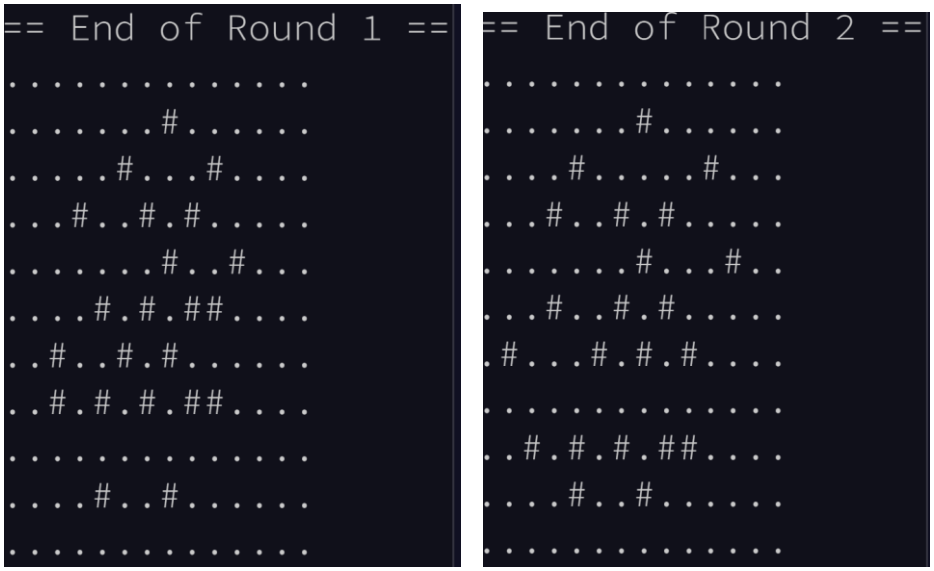
Each year, the Advent of Code is launched on December 1. For 25 days, a task consisting of 2 parts is presented for each day. The tasks are usually quite interesting and challenging (part 2 in each task could be more difficult). If you have time and can complete all tasks, you will find that you would improve your programming skills a lot.

So, if you are interested and follow the Advent of Code, you can contact me to choose one task for grade 5 task in lab 4.

In year 2022, Day 23 task is interesting and like the Game of Life as described in the first option of this instruction. It can be a grade 5 task for lab4. Only Part 1 is required for this Day 23 task. On the course page, under lab4, you can find the instruction and input data files for this task. If you register on the advent of code page, you can get your own input to the task. Extra requirements for this task is, you should read the initial state from the file using file input function, and should print out the pattern after each round (generation in Game of Life). In this way, you can follow the development and check whether your program works correctly. For example,



After the movements and at the end of the first 2 rounds:



Good luck!