VG101 Lab Manual

Lab 7

Instructor: Dr. Yifei ZHU
TA: Hangrui CAO
TA: Haoxuan SHAN
TA: Qinhang WU
TA: Muchen XU

Table of Contents

- Joint Online Judge
- MISC: Makefile
- C structure
- Lab Design: Arknights (II)

Workflow

Content	Approx. Time
Joint Online Judge(JOJ)	10 mins
Makefile	20 mins
Warmup: Basic Syntax Exercise	30 mins
Break	10 mins
Lab Design: Arknights (Part II)	60 mins
Break	10 mins
Practical Exercise	50 mins

Joint Online Judge(JOJ)

From this week, we'll start to use JOJ for grading your homework and exams. Onling judge(OJ) is a platform for testing your program. You can get immediate response from the OJ.

How to submit

1. Go to website "https://joj.sjtu.edu.cn/d/vg101_summer_2020/"
2. Enter the homework to handle
3. Claim the homework and you'll see the detailed questions to that homework.
4. Enter each question, read the description (if any) and write your code on your computer.
5. Compress your `.c` or `.cpp` code into `.zip` `.rar` or `.tar` document. Submit it on JOJ.
6. Wait for a while, you can then see the feedback.

Score on JOJ

The score on JOJ isn't your final submission score. Usually, it is a feedback of how your program does with test cases.

Need to mention, right after your submission, the "Accepted" status only means that your program pass the compile and pass those pretest cases. Even if you pass all the pretest cases, you still may fail to pass some test cases.

Some problems

JOJ applies Linux system, which is similar with WLS we mentioned before. In general , a program works on your computer can work on JOJ, however, in some case, it can't.

System difference

In Windows usually works in Linux. For example, in a Windows text file, '\r\n' may represent "new line" while in Linux, '\n' is the correct way.

More strict compiling command

Also, JOJ has a more strict requirement in compiling. It compiles with the option `-Wall -Wextra -Werror` which means, it opens all warnings and turns all warning into errors. The complete compile command on JOJ, in general, has the form `gcc -O2 -Wall -Wextra -Werror -pedantic -Wno-unused-result -std=c11 -o out test.cpp -lm` (to compile a file test.cpp and output a executable program out).

Plus, JOJ usually requires higher on your code. Problems like "Out of range address", "misusage of library function", which may not happen in your computer, may happen on JOJ. Hence, you need to make sure what you are writing.

Compare your result with the standard answer

JOJ do offer access to the standard answer to some pretest cases and you can also find your answer to those cases. Need to mention that, the answer showed on JOJ is limited by length. For some long output, you're not able to find the complete answer.

Start your trial now

Starting from this lab, please try to submit your program on JOJ if it's available. If there's any problems, feel free to go to the office hour.

Makefile (Compare CMake)

A makefile stores the information which is to be given to the compiler. It's useful when compiling multiple documents. The compiler will follow the makefile's instruction to compile the files.

Each part of a makefile has the general form (Notice the tab before command is necessary):

```
Target:Dependency
    Command
```

An examples from VE280 slides. This example generate the executable file `run_add`. It's composed by `run_add.o` and `add.o`. These two `.o` files are compiled by more detailed `.cpp` files. The compiler will start from the end of the dependency link and finish the whole process.

```
all:run_add

run_add:run_add.o add.o
    gcc -o run_add run_add.o add.o

run_add.o:run_add.c
    gcc -c run_add.c

add.o:add.c
    gcc -c add.c

clean:
    rm -f run_add *.o
```

Have a try if you would like to. It can benefit a lot if you're choosing C++ as the language to use in your project.

Lab Design: Arknights (Part II)

In Lab 6, we've implemented a C-style tower defense game. For Lab 7, we need include a bit more features in that game. We've provided you with a set of starter files so as to make your life easier.

For specification of the game, please refer to Lab 6 Manual.

Header files

In lab 7, we separate the previous `Arknights.c` into two source files and a header file: `Arknights.c`, `arkMap.c` and `arkMap.h`, correspondingly. In order to compile the game correctly, you need the following **gcc** command: `gcc Arknights.c arkMap.c -o arknights`. Note that you only need to include source files.

For clion user, you may add `add_executable(arknights Arknights.c arkMap.c)` to your CMakeLists.

For VScode user, you may add the following lines in **tasks.json**:

```
// previous lines omitted
"args": [
    "-g",
    "${fileDirname}\\Arknights.c",
    "${fileDirname}\\arkMap.c",
    "-o",
    "${fileDirname}\\arknights.exe",
    "-std=c11"
],
// subsequent lines omitted
```

Structure

In order to organize the data better, we define the following structures:

```
typedef struct _Enemy{
    double health; // current health
    int x; // x coordinate
```

```
    int y; // y coordinate
}Enemy;

typedef struct _PRTS{
    int operatorNum; // number of operators
    int heatpumpNum; // number of heat pumps
    int enemyNum; // number of enemies
    Enemy *enemy;
    int invX;
    int invY;
    int debugStatus;
}PRTS;
```

In this section, your mission is to use the structure defined and the code skeleton provided to implement the function `mapDebugger`.

You may also try yourself to implement a new structure `Point` to organize the x and y coordinate of a certain point on the map (*optional*).

DFS

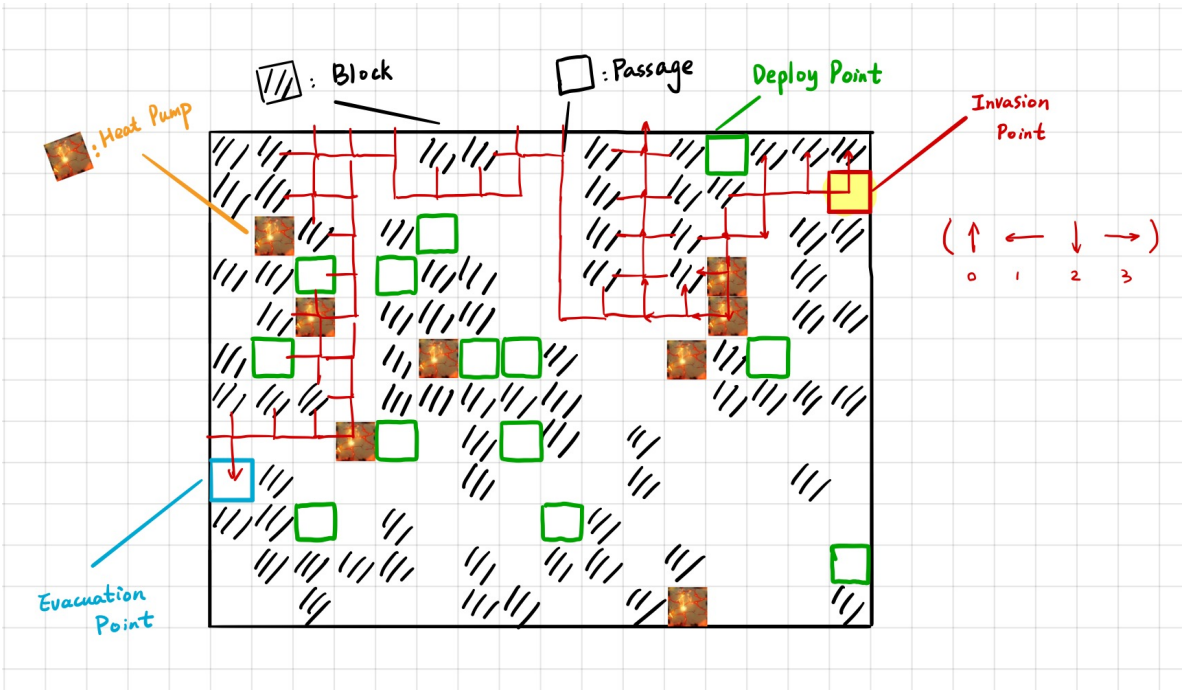
Now that we've implement the function `mapGenerator` to generate a random map and the function `mapDebugger` to test the map manually, we need to find a way to test the map automatically.

In this section, your mission is to design a function `mapSearcher` that takes the generated map as input and print the **minimum** step for the enemy to move from the Invasion Point to the Evacuation Point.

Hint: you may consider using DFS (Depth First Search) algorithm.

The Depth First Search (**DFS**) is an algorithm frequently used to solve maze-related problem, realized by recursion. Recursion can be used here for the following 3 reasons:

- The target is clearly defined, that we finished searching all the cells that can be reached.
- We are uncertain about how many steps are necessary to reach the target, which makes the problem hard to settle by simple loops.
- Each step of the process is generally the same. For this problem, we are just repeating the following steps.
    - Check the four directions of the current cell. If it is neither unreachable nor reached before, take a step to the new cell.
    - Check whether there's treasure in the new cell.
    - Mark the new cell as reached.



Think about the problem: what to do if we need to calculate a routine for the enemy to get to the evacuation point with maximal health points? (*optional*)

Grading Rubric

Criteria	Weight	Available Time	Due Time	Entry
Attendance	30%	**4:00pm**, June.28	**11:59am**, June.29 (**noon**)	Canvas Assignment
In-lab quiz	70%	**9:00pm**, June.28	**11:59pm**, June.30	Canvas Quiz
Arknights bonus	30%	**9:00pm**, June.28	**11:59pm**, June.30	Canvas Assignment

- For the attendance score, you need to submit your code for **exercises** in the **worksheet** on Canvas. We won't judge the correctness of your code. You'll earn full credits as long as you've tried these exercises.
- The Arknights bonus will be counted towards your final lab score. You need to submit the code as well as the output of the game when calling `arknights.exe --gen --search` (you're free to specify the output by yourself) so as to earn the full bonus.

Reference

[1] VE280 Slides, 2020SU, 03-Developing-Programs