VG101 Lab Manual

Lab 6

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

Table of Contents

Workflow

Content	Approx. Time
Lab Workshop: Debugging Skills	20 mins
Warmup: Basic Syntax Exercise	30 mins
Break	10 mins
Lab Design: Arknights (Part I)	40 mins
Break	10 mins
Practical Exercise	70 mins

Debugging Skills

Clion is a powerful tool, with functions of debugging executable C/C++ with GDB on all platforms and with the bundled LLDB on macOS and Linux. There is also an LLDB-based debugger for MSVC toolchain on windows. In the demo, we use custom GDB to debug.

Static Observation

Used to examine some simple bugs. In Clion, you can set warnings by

`set(CMAKE_C_FLAGS "-Wall -Wextra -Werror -pedantic -Wno-unused-result -g3")`

Adding this line to cmakelist and it will open all warnings and consider all warnings as errors.

```
scanf("%d",x);
```

```
for (int i=n;i>0;i++)
    ...
```

```
for (int i=0;i<n;i++)
    for(int j=0;j<n;i++)
        ...
```

```
if (a[i]=min) ...
```

Output Intermediate

Use `printf` to output the intermediate variable to check the correctness of functions. For example, in following example, by using `printf`, check whether function `gcd` works or not.

```
scanf("%d",&cur);
for (int i=1;i<n;i++)
{
    scanf("%d",&a[i]);
    int d=gcd(cur,a[i]);
    printf("%d %d\n",cur,a[i]);
    cur=cur/d*a[i];
}
```

Step Through

Use step into to see the inside code of one function, use step over to move to the next line of the caller.

```
int gcd(int x,int y)
{
    z=x%y;
    while(z!=0)
    {
        y=z;
        x=y;
        z=x%y;
    }
    return x;
}
```

Breakpoint

Breakpoints are special markers that suspend program execution at a specific point. This lets you examine the program state and behavior. Breakpoints can be simple (for example, suspending the program on reaching some line of code) or involve more complex logic (checking against additional conditions, writing log messages, and so on).

Watch

If you want to keep track of some variable or the result of a more complex expression, set up a watch for this variable or expression. This is useful when you need to add something that is not regularly displayed on the list of variables, or to pin some instance variable thus eliminating the need to expand the tree after each step.

Step over (F8)

Steps over the current line and takes you to the next line even if the current line includes function calls. The calls are skipped, and you move straight to the next line of the caller.

Step into (F7)

Steps inside the code of a called function.

Step out (Shift+F8)

Steps out of the current function and takes you to the code the caller.

Run to cursor (Alt+F9)

Continues the execution until the position of the cursor is reached. Just place the cursor at the line where you want the program to pause.

Lab Design: Arknights (Part I)

In this section, we'll try to design and implement a simplified tower defense game in C style.

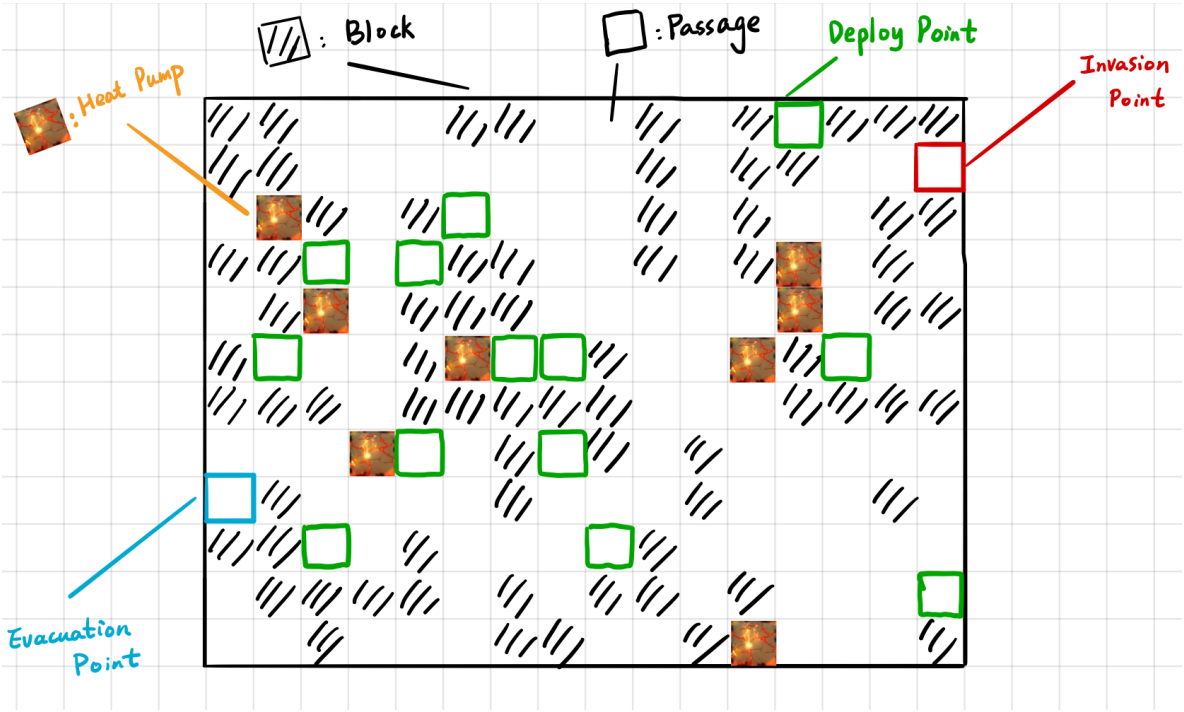
Basic Settings

For simplicity and consistency, we will define the general setting, rules and other specifications here.

A valid Arknights map consists of the following elements:

- **Invasion Point**: the cell on the map where enemies get in. For simplicity, we assume that the invasion point is always on the border of the map. For Lab 6, you may only consider the case where one enemy exists.
- **Evacuation Point**: the cell on the map where enemies target at. If any enemy gets in the evacuation point, you get lost.
- **Deploy Point**: the cell on the map where you can deploy your operators to fight against the enemies.
- **Passage**: the cell where enemies can walk through freely.
- **Heat Pump**: the cell where enemies will lose health points continuously.
- **Block**: the cell where enemies cannot walk on and your operators cannot be deployed.

The figure below shows a valid 14*18 Arknights map:



Note that the generated map is enclosed by a layer of blocks.

For simplicity, we assume that before the game starts, each deploy point on the map is occupied by an operator that can shot the enemies freely within the range of a 3*3 block centered at the corresponding deploy point.

We've provided you with a lab starter file so that you may concentrate on the implementation of functions rather than the structure of the whole project.

Program Argument

The design of such a tower defense game is complicated that involves various functions. In order to let the users play with your game easily, we need the assistance of program arguments.

To accept command line arguments, use the `argc` and `argv` as arguments to the main function. A sample code is listed as follow:

```
1 #include <stdio.h>
```

```

int main(int argc, char *argv[]) {
// argc (argument count): number of arguments
// argv (argument vector): array of strings, each holding an argument
// arguments are separated by spaces
// first argument is the name of the executable
// (prefixed by "./" if called in this style)

// demo: print all the arguments
    for (int i = 0; i < argc; ++i) {
        printf("%s\n", argv[i]);
    }
    return 0;
}
```

In this lab, you should process three arguments in total:

- `--help`, print a list of all valid arguments
- `--gen`, call the function `mapGenerator` and print the map using ASCII characters
- `--debug`, call the function `mapDebugger`. Note that if `--debug` is included without `--gen`, you should print an error message and end the program.
- for other arguments (or that there is no arguments), print a list of all valid arguments, just as `--help` does.

Map Generator

In this part, you will need to randomly generate an Arknights map with certain parameters specified in the input.

We divide this task into three parts. Firstly, generate a map filled with *Passage* surrounded by a layer of *Block*. The following sample shows a 5*5 initialized map (where 0 denotes *Block* and 1 denotes *Passage*).

```
00000
01110
01110
01110
00000
```

Secondly, fill the inner part of the map with randomly generated *Passage* and *Block* using the following algorithm:

```
#define UP 0
#define LF 1
#define DW 2
#define RT 3

//...

void plainMapGen(int x, int y, int dir){
    // EFFECTS: randomly generate an Arknights map with blocks and passages recursively
    // MODIFIES: arkMap

    // x, y denotes the coordinate of the current location;
```

```
    // dir denotes the direction of the extension of blocks
    int rn[4][3] = {
        {x-2,y,LF},
        {x+2,y,RT},
        {x,y+2,UP},
        {x,y-2,DW}
    };
    randomShuffle(rn); // TO-DO: shuffle rn[0],rn[1],rn[2] and rn[3]

    for(int cn=0;cn<4;cn++) {
        if(inBounds(rn[cn][0],rn[cn][1])/*TO-DO: judge whether the current node is out of border*/ && !arkMap[rn[cn][1]][rn[cn][0]]) {
            if(rn[cn][2]==LF)
                arkMap[rn[cn][1]][rn[cn][0]+1] = 1;
            else if(rn[cn][2]==RT)
                arkMap[rn[cn][1]][rn[cn][0]-1] = 1;
            else if(rn[cn][2]==UP)
                arkMap[rn[cn][1]-1][rn[cn][0]] = 1;
            else if(rn[cn][2]==DW)
                arkMap[rn[cn][1]+1][rn[cn][0]] = 1;

            arkMap[rn[cn][1]][rn[cn][0]] = 1;
            plainMapGen(rn[cn][0],rn[cn][1],rn[cn][2]);
        }
    }
}
```

You should implement function `randomShuffle` and `inBounds` by yourself.

Thirdly:

- replace `p` *Passage* with `HeatPump`
- replace `q` *Block* with *Deploy Point*
- `p` and `q` will be given in the starter file.
- replace one *Block* on the border with *Invasion Point*
- replace one *Passage* with *Evacuation Point*

For detailed specifications and hints, see the starter file.

In order to display the map in the command line, we use the following ASCII characters to represent the special cells in the map: `I` for Invasion Point, `E` for Evacuation Point, `o` for Deploy Point, `X` for Heat Pump and `#` for Block. The Passage is represented by a blank character.

Note that you're free to specify anything by yourself.

Map Debugger

In this part, you will need to act as soldiers from the *Reunion Movement* to test the map that you've generated with keyboard prompts.

For detailed requirements, see the starter file.

In the next lab, we will take advantage of some implemented functions here to further improve the game.

Appendix

Negative Example

This is an example of calculating the least common multiple(lcm) using Euclidean algorithm. (`gcd` refers to the greatest common divisor) For the `lcd` and `gcd`, the relationship is: $gcd(a,b) * lcm(a,b) = a * b$.

```c
#include <stdio.h>
int a[10];
int gcd(int x, int y)
{
    int z = x % y;
    while (z != 0)
    {
        y = z;
        x = y;
        z = x % y;
    }
    return x;
}
int main()
{
    int n;
    scanf("%d", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    int cur = a[0];
    for (int i = 1; i < n; i++)
    {
        int d = gcd(cur, a[i]);
        // printf("gcd(%d,%d)=%d\n", cur, a[i], d);
        cur = cur / d * a[i];
    }
    printf("%d\n", cur);
    return 0;
}
//input: 6 1 2 3 4 5 6
//output: 60
```

Grading Rubric

Criteria	Weight	Available Time	Due Time	Entry
Attendance	30%	**4:00pm**, June.19	**11:59am**, June.20	Canvas Assignment
In-lab quiz	70%	**9:00pm**, June.19	**11:59pm**, June.21	Canvas Quiz

- 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.

Reference

[1] Step Through, Jetbrains.

[2] Breakpoints, Jetbrains

[3] Watches, Jetbrains