Lab5 Report

Reporter: Chen Jiatong

Student ID: 3200105258

Lab Name: Skiing

Lab Task: My job is to write a program to help Professor Patt to read the map where he is skiing and tell the longest distance he can slide.

Lab Requirements:

1. You are required to write in **LC-3 assembly language**.
2. Your program should start at x3000, which means the first instruction of your program is located in position x3000.
3. Your program **must** use recursive ways to solve this problem.

Analysis:

1. Since the program is asked to use recursive ways to solve this problem, so we need to attach a lot of importance to the recursive functions. In order to search the longest way, we can define a function named MaxLength. We can test whether it's allowed to ski to left, right, up and down neighbor hill in order. If it's allowed, we add the length and update the starting point. Then call MaxLength again. If all the directions are illegal or have been tested, we store the max length for this hill, return to last hill and test next direction. After all the hills have been tested, we can get the max length.

I write the function in C to show what I mean:

```
int MaxLength(int i, int j)
{
    int s, max;
    max=0;
    if(LegitimacyTest(Right))
    [
        s=MaxLength(i,j+1);
        max=s>max?s:max;
    }

    if(LegitimacyTest(Left))
    {
        s=MaxLength(i,j-1);
        max=s>max?s:max;
    }

    if(LegitimacyTest(Up))
    {
```

```
            s=MaxLength(i-1,j);
            max=s>max?s:max;
        }
        if((LegitimacyTest(Down))
        {
            s=MaxLength(i+1,j);
            max=s>max?s:max;
        }

            return (max+1);
}
```

2. However, it's not good enough to stop here. We notice that the local optimal solution is consistent with the global optimal solution. So, we can use Greedy Algorithm to optimize our program. We can use an array to store the local optimal solution for every point. When we need the same point the second time, all we need to do is access the array and get the data we need. In that case, we can avoid repeated calculations since every point is computed only once.

Codes:

There are several functions used in my program. They are so easy that I won't show all the codes. However, I will show what they do for me.

MUL: Compute R_3*R_4 and store the result in R_0.
OffsetFigure: Compute the index of (R_1, R_2) in the array MAP.
LegitimacyTest: Test whether it's legal to go through the direction to the neighbor hill.
MemIn: Clear a part of memory.

Also, there is an important part of memory which stores data we need. I'll explain their meanings.

StartLine	
StartRow	Mark the point we start searching.
Return	Store the return value of MaxLength.
LineNumber	
RowNumber	Store the size of the map.
Data	The address of data segment.
DataStack	The address of stack segment.
HaveAccessed	The array which stores max length for every point.

Then I think it's time to show the most important codes.

```
Main:       LD R0, Data
            LDR R1, R0, #-2
            ST R1, LineNumber
```

```
                LDR R1, R0, #-1
                ST R1, RowNumber
                LD R6, DataStack
                JSR MemIn

                AND R1, R1, #0
                AND R2, R2, #0

                ADD R6, R6, #1    ;max
                STR R1, R6, #0
                BRnzp LOOP2

LOOP1:          LD R3, StartLine
                ADD R3, R3, #1
                LD R4, LineNumber
                NOT R4, R4
                ADD R4, R4, #2
                ADD R5, R3, R4
                BRp ExitSearch
                ST R3, StartLine
                AND R3, R3, #0
                ST R3, StartRow

LOOP2:          LD R1, StartLine
                LD R2, StartRow
                JSR MaxLength
                LD R0, Return
                LDR R3, R6, #0
                NOT R3, R3
                ADD R4, R0, R3
                BRn Adjust
                STR R0, R6, #0

Adjust:         LD R3, StartRow
                ADD R3, R3, #1
                LD R4, RowNumber
                NOT R4, R4
                ADD R4, R4, #2
                ADD R5, R3, R4
                BRp LOOP1
                ST R3, StartRow
                BRnzp LOOP2

ExitSearch:     LDR R2, R6, #0
```

```
                ADD R6, R6, #-1
                HALT

MaxLength:      ADD R6, R6, #1
                STR R7, R6, #0
                ADD R6, R6, #1    ;max
                AND R0, R0, #0
                STR R0, R6, #0
                ADD R6, R6, #1    ;s

                ADD R6, R6, #1    ;OldLocation

                JSR OffsetFigure
                STR R0, R6, #0
                LEA R3, HaveAccessed
                ADD R3, R3, R0
                LDR R0, R3, #0
                BRp NormalExit

                ADD R2, R2, #1
                JSR LegitimacyTest
                ADD R0, R0, #0
                BRp Next1
                ADD R6, R6, #2
                STR R1, R6, #-1
                STR R2, R6, #0
                JSR MaxLength
                LDR R2, R6, #0
                LDR R1, R6, #-1
                ADD R6, R6, #-2
                LD R0, Return
                LDR R5, R6, #-2
                NOT R5, R5
                ADD R3, R0, R5
                BRn Next1
                STR R0, R6, #-2
;Test Right. Other directions are omitted in report.

Exit:           LDR R0, R6, #-2
                ADD R0, R0, #1
                LEA R5, HaveAccessed
                LDR R4, R6, #0
                ADD R5, R5, R4
                STR R0, R5, #0
```

```
NormalExit:     ST R0, Return
                LDR R7, R6, #-3
                ADD R6, R6, #-4
                RET
```

Improvement according to the TA's question

1. My program also prints out the final result. Since it’s not required, these codes are not mentioned in my report.
2. It’s a good idea to use Greedy Algorithm in this Lab.
3. It’s better to follow the standards of calling functions in LC-3. My way to deal with the return value is not suggested.