Southeast University, Bangladesh

School of Science and Engineering

Department of Computer Science and Engineering

CSE162: Programming Language I Lab Section: 7

Fall 2022

Lab Report

=: Submitted To :=	=: Submitted By :=
Tashreef Muhammad[TMD]	**Name :** Md. Zubayer Ahmad Shibly
Lecturer, Dept. of CSE,	**SEU ID :** 2022100000052
Southeast University, Bangladesh	**Batch :** 61

Jan 30, 2022

Contents

1 Experiment 1

1.1 Topics Covered

Basic Data types, variables, Basic I/O, Conditional Statements, Nested Conditions

1.2 Problem Statement

You should write a code that will do the following tasks

- Ask the user to provide a number which will be the threshold of the passing number.
- Ask the user to provide another integer number which will be the threshold of the excellence number.
- Now, you must validate whether the excellence number is greater than or equal to the passing number or not. If it is not the case, then print that the provided values are invalid and stop code.
- If the values are correct then ask the user to enter a number. If the number is a failing number, then print it is a failing number. If the number is a passing number, print that it is a passing number. If the number is an excellence number, print that it is an excellence number.

You can take help from Algorithm 1 and Figure 1 to understand how you should solve the problem more clearly

Algorithm 1 Probable Algorithm for Solving Experiment 01

```
Print → Enter the threshold value for passing number:
p ← User Input
Print → Enter the threshold value for excellence number:
e ← User Input
if e ≥ p then
    Print → Provide a number:
    n ← User Input
    if n < p then
        Print → It is a failing number
    else
        if n ≥ p then
            Print → It is a passing number
        end if
        if n ≥ e then
            Print → It is an excellent number
        end if
    end if
else
    Print → Input is invalid
end if
```

1.3 Solution Approach

This code is an example of conditional statements and nested conditions where the user will enter two integer number. Those numbers are Passing number and Excellence number. At first a condition is applied, which will check whether the Excellence number is greater than or equal to the Passing number or not. If the condition is not followed, the code will stop while printing "Input is Invalid". If the condition is followed, it will proof the values are valid. After that, the user will enter a number. Then some nested conditions will help to find, whether the Number is Failing number or Passing number or both Passing and Excellence number. If the number is less than the Passing number, "It is a failing number" will be printed. If the

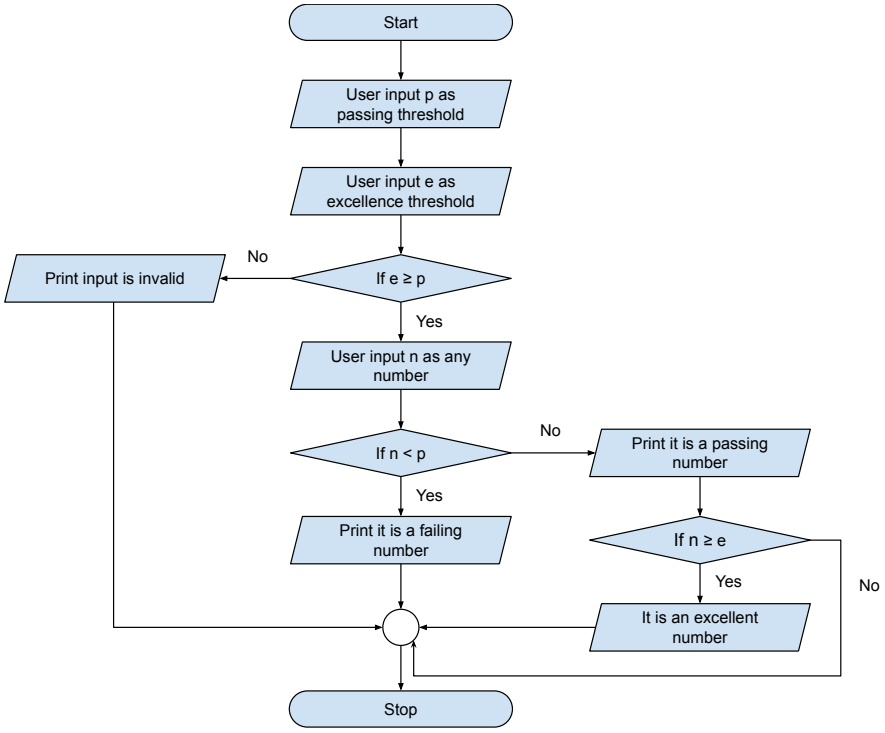

Figure 1: Flowchart of the required code

number is greater than or equal to the Passing number, “It is a passing number” will be printed. If it is a Passing number, there will be another condition. Which will check, if it is also an Excellence number or not. If the number is greater than or equal to the Excellence number, “It is an excellent number” will also be printed. If not, the code will end there.

1.4 Solution Code

```
#include<stdio.h>
int main(){
    int p,e;
    printf("Enter the threshold value of the passing number:");
    scanf("%d",&p);
    printf("Enter the threshold value of the excellence number:");
    scanf("%d",&e);
     if(e>=p){
         int n;
         printf("Enter a number:");
         scanf("%d",&n);
         if(n<p)
              {
              printf("It is a failing number");
              }
         else {
         if(n>=p)
              {
                      printf("It is a passing number\n");
              }
         if(n>=e)
              {
                      printf("It is an excellent number");
              }
              }
           }
         else
              {
             printf("Input is invalid");
              }
return 0;
}
```

Listing 1: C example

2 Experiment 2

2.1 Topics Covered

Loops, Nested Loops

2.2 Problem Statement

You have to write a code that is going to print a specific pattern. You should use loops to print this pattern. Your code should comply with the following instructions

- Ask the user to provide an integer number.
- From the provided integer number, print the required pattern of asterisk (*)

You should refer to sample cases [1] for better understanding the pattern. The text in red is the user input, and the text in blue is the lines printed by your code.

Sample I/O 1

Enter an Integer Number: 6

```
            *
          * *
        * * *
      * * * *
    * * * * *
* * * * * *
* * * * *
* * * *
* * *
* *
*
```

Sample I/O 2

Enter an Integer Number: 5

```
          *
        * *
      * * *
    * * * *
* * * * *
* * * *
* * *
* *
*
```

[1] Sample cases mean only a few cases to explain the workflow. You may be given any integer as input and should print the appropriate pattern. Do not aim at only complying with the sample cases.

Sample I/O 3

Enter an Integer Number: 2

```
   *
 * *
*
```

2.3 Solution Approach

This C program employs loops and nested loops to print a pattern. The user must provide the application with an integer value of n. The pattern's size will be determined by this integer. At first, it will print Mirrored Right Triangle Star Pattern. Then it will print Inverted Right Triangle Star Pattern. Mirrored Right Triangle Star Pattern: It will be created by the program using a for loop, and this for loop will have two nested for loops. The outer for loop (row) runs from 1 to n in increments of 1. Starting at n-1 and moving down to row is the first inner for loop (col) or the first nested loop (col). Before printing the stars, the first inner loop is employed to make the spacing. The first inner loop (col) will reduce the amount of space created by the outer loop by one for each iteration. The stars of the mirrored right triangle pattern are made using the next nested for loops or second inner loop (col). The value of the outer loop (row), which runs from 1 to n, is 1. Starting at 1, the second inner loop (col) moves up to row. The second inner loop will print one extra star after each iteration of the outer loop. Inverted Right Triangle Star Pattern: It will be created by the program using an additional outer for loop. The outer for loop (row) runs from 1 to n in increments of 1. Starting at n-1 and moving down to row is the inner for loop (col). Before the next line is printed, the stars are made using the inner loop. The inner loop will produce one fewer star for every outer loop iteration. The computer program returns 0 and instructs the user to stop when the loops are complete.

2.4 Solution Code

```c
#include<stdio.h>
int main(){
int n, row, col;
printf("Enter an integer value of n: ");
scanf("%d", &n);
for(row=1; row<=n; row++){
for(col=1; col<=n-row; col++){
        printf(" ");
       }
    for(col=1; col<=row; col++){
        printf("*");
    }
    printf("\n");
}
for(row=n-1; row>=1; row--){
    for(col=1; col<=row; col++){
        printf("*");
    }
    printf("\n");
}
return 0;
}
```

Listing 2: C example

3 Experiment 3

3.1 Topics Covered

Arrays (1D, Multidimensional)

3.2 Problem Statement

Write a code to develop a 2D grid of integers. Each of the boxes in the grid contains an integer. You have to identify the cell whose sum of adjacent cells is maximum. Alternatively you can print another 2D grid that contains the sum of adjacent cells for each existent cell. Adjacent cells are explained in the later part of this question. You should take input from the user, an integer n. Then construct a nxn 2D array/matrix. Take input from the user about the different elements of the matrix. Then print two integers, i and j, which will be the index of the cell whose sum of adjacent cells is maximum.

Adjacent Cells

Adjacent Cells are those cells in a table that are connected to each other. For example, in Figure 2, the Blue Cells are adjacent cells of the Red Cell.

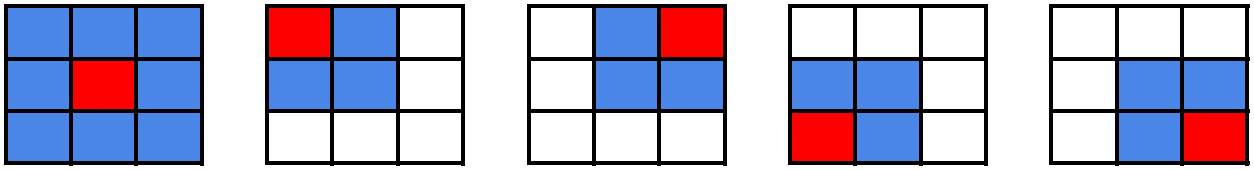

Figure 2: A Figure Showing Adjacent Cells

3.3 Solution Approach

This program is a 2D grid with N*N size, where the user inputs the size and elements of the grid. It then prints the grid and the sum of adjacent cells of each cell in the grid. It also finds the cell with the maximum sum of adjacent cells, and prints the sum and the cell's coordinates. The program first prompts the user to enter a positive integer for the size of the 2D grid, and then uses nested for loops to take input for the elements of the 2D grid. It then prints the 2D grid using another nested for loop. In the next step, the program uses nested for loops to calculate the sum of the adjacent cells for each cell in the grid. The program checks the position of the cell (i, j) in the grid and uses if-else statements to determine the sum of the cells adjacent to it. The program also keeps track of the maximum sum of adjacent cells and its corresponding cell coordinates (max_sum, max_i, max_j). Finally, the program prints the maximum sum of adjacent cells and the cell's coordinates (i, j) with the maximum sum.

3.4 Solution Code

```
#include <stdio.h>

int main()
{
    int n,i,j,sum=0,max_sum = 0,max_i, max_j;
    printf("Enter a positive integer for the size of the 2D grid: ");
    scanf("%d", &n);
    int a[n][n];
    printf("Enter positive integer elements of the 2D grid:\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
```

```
        {
            scanf("%d",&a[i][j]);
        }
    }
    printf("The 2D grid is:\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            printf("%d ",a[i][j]);
        }
        printf("\n");
    }
    printf("The 2D grid with the sum of adjacent cells is:\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            if(i==0)
            {
                if(j==0)
                {
                    sum=a[i][j+1]+a[i+1][j]+a[i+1][j+1];
                }
                else if(j==n-1)
                {
                    sum=a[i][j-1]+a[i+1][j-1]+a[i+1][j];
                }
                else
                {
                    sum=a[i][j-1]+a[i][j+1]+a[i+1][j-1]+a[i+1][j]+a[i+1][j+1];
                }
            }
            else if(i==n-1)
            {
                if(j==0)
                {
                    sum=a[i-1][j]+a[i-1][j+1]+a[i][j+1];
                }
                else if(j==n-1)
                {
                    sum=a[i-1][j-1]+a[i-1][j]+a[i][j-1];
                }
                else
                {
                    sum=a[i-1][j-1]+a[i-1][j]+a[i-1][j+1]+a[i][j-1]+a[i][j+1];
                }
            }
            else
            {
                if(j==0)
                {
                    sum=a[i-1][j]+a[i-1][j+1]+a[i][j+1]+a[i+1][j]+a[i+1][j+1];
                }
                else if(j==n-1)
                {
                    sum=a[i-1][j-1]+a[i-1][j]+a[i][j-1]+a[i+1][j-1]+a[i+1][j];
                }
                else
                {
                    sum=a[i-1][j-1]+a[i-1][j]+a[i-1][j+1]+a[i][j-1]+a[i][j+1]+a[i+1][j-1]+a[
i+1][j]+a[i+1][j+1];
                }
            }
            if(sum>max_sum)
            {
                max_sum=sum;
                max_i=i;
```

```
                max_j=j;
            }
            printf("%d ",sum);
        }
        printf("\n");
    }
    printf("The maximum sum of adjacent cells is %d and the cell with the maximum sum is (%d
    ,%d).\n",max_sum,max_i, max_j);
    return 0;
}
```

Listing 3: C example

4 Experiment 4

4.1 Topics Covered

Multifunction Programming, Recursive Function

4.2 Problem Statement

Write a recursive function to calculate the following series of numbers:
$1 \times 3 + 3 \times 5 + 5 \times 7 + 7 \times 9 + 9 \times 11 + \ldots + (2k-1) \times (2k+1)$
You will be given the value of $(2k-1)$ only and you must calculate the sum using recursive functions.

4.3 Solution Approach

The code is a recursive function which calculates the sum of odd numbers from 1 to the given number. The user is prompted to enter an odd number, and the program uses a recursive function "calSum" to calculate the sum. Recursive function "calSum" takes an integer as input and returns an integer. The name of the parameter is oddN. The function has a base case of "if (oddN==1) return 3", which means that if the input is 1, the function will return 3 (the sum of the first odd number, 1). Otherwise, the function calls itself with the input decremented by 2, and adds the result to the current odd number multiplied by the next odd number. The main function then prints the final result.

4.4 Solution Code

```c
#include<stdio.h>
int calSum(int oddN);
int main(){
    int n;
    printf("Enter only odd number: ");
    scanf("%d",&n);
    printf("%d",calSum(n));
    return 0;
}
int calSum(int oddN){
    if(oddN==1)
    return 3;
    return calSum(oddN-2)+(oddN*(oddN+2));
}
```

Listing 4: C example