

CS 1004 Object Oriented Programming

Assignment 2

Deadline: Sunday 5th March, 2023

Instructions:

- Make sure that you read and understand each and every instruction. If you have any questions or comments you are encouraged to discuss your problems with your colleagues (and instructors) on google classroom.
- The student is solely responsible for checking the final .cpp files for issues like corrupt files, viruses in the file, or mistakenly exe sent. If we cannot download the file from Google Classroom, it will lead to zero marks in the assignment.
- The displayed output should be well-mannered and well presented. Use appropriate comments and indentation in your source code.
- If there is a syntax error in the code, zero marks will be awarded in that part of the assignment.
- **Keep a backup of your work always that will be helpful in preventing any mishap and avoid last hour submissions.**

Submit a single '.zip' file for your assignment and each problem solution must be provided in a separate CPP file. For instance, you must name the file containing solution of first file as 'q1.cpp' and second as 'q2.cpp' and so on.

Deadline: The deadline to submit the assignment is **Sunday, March 5th, 2023 (11:59 pm)**. No late submission will be accepted. Correct and timely submission of the assignment is the responsibility of every student; hence no relaxation will be given to anyone.

Please follow the submission instructions. Failure to submit according to the above format you will be awarded straight zero in this assignment. Submissions other than Google classroom (e.g., email etc.) will not be accepted.

Honor Policy

This assignment is a learning opportunity that will be evaluated based on your ability. Plagiarism cases will be dealt with strictly. If found plagiarized, both the involved parties will be awarded zero marks in this assignment, all the remaining assignments, or even an F grade in the course.

Note: Start early so that you can finish it on time.

Question 1

Write a recursive function that takes 3 integers: A, B, and C.

A: The first number (total number of digits of the integer should not exceed 9)

B: The second number (single digit) that is to be searched from the first integer

C: The third number representing a position of a number .

e.g. SearchDigit(158664,6,1)

The recursive function should return the position of the digit from the left. If there are more than 2 digits found, then the rightmost digits' position is to be returned. In this case 5 would be returned. In the case where the number to be searched is not found or the length of the number exceeds 9, the function should return -1.

Any kind of loops, static and global variables are strictly not allowed.

Question 2

Write a recursive function **Pattern** which print a pattern that takes **two-character** arguments, and two integer arguments **n** and **k**. **n** is the starting number while **k** is the ending limit.

Note: No loops are allowed whatsoever, and you can write a maximum of one helper function apart from the main function.

Example:

Pattern(1, 4, '^', '*') would print the following pattern.

```
1*2^^3***4^^^^3***2^^1*
```

Question 3

Suppose there is an array of 50 numbers. Using the formula: (addition of all the digits of your roll number)/2, you can calculate a number.

E.g. If your roll number is 21i-0865

$$2+1+0+8+6+5=22$$
$$22/2=11$$

Now, you just have to add the numbers from the array that are at this number's multiple indexes. In this case, only the numbers at positions 11, 22, 33, and 44 should be added. There should be two functions: one to calculate your roll number and another recursive function to add the numbers of the array.

Your function prototypes should be as follows:

```
int additionOfRollNumber(string rollnumber);
```

```
int FillArray(int arr[], int index, int calculatedNumber);
```

Question 4

Given an integer n and an integer m, count the total number of digit m appearing in all non-negative integers less than or equal to n.

Function Prototype: int countDigits(int n, int m);

For examples n = 33 and m = 3 return 8

3 -> 1 30 -> 1 33 -> 2

13 -> 1 31 -> 1

23 -> 1 32 -> 1

Note: No loops are allowed whatsoever, and you can write a maximum of one helper function apart from the main function.

Question 5

Calculate the power of a number using bitwise operators. You're not allowed to use the multiplication or division operand.

You can write a maximum of one helper function with two arguments apart from the main function. The parameter extra is for your help. It has a default value assigned to it. You can change the default value of the parameter, however, **DO NOT CHANGE any other part of the function prototype.**

Function Prototype: int power(int base, int exponent, int extra)

Question 6

Write Recursive Functions for Checking if one string is a substring of another string. You can write a maximum of one helper function with two arguments apart from the main function.

Function Prototype: bool isSubStr(char* str1, char* str2);

Question 7

Write Recursive Functions to Perform the following tasks:

- a. Convert integer to Binary.

Hint: With the help question 5 as a helper function you can convert the number into binary

Function Prototype: int toBinary(int num, int extra);

- b. Convert integer to Hexadecimal. (You can use string for this part ONLY)

Function Prototype: string toHexa(int num, int extra);

- c. Convert integer to Octal.

Hint: With the help question 5 as a helper function you can convert the number into Octal

Function Prototype: `int toOctal(int num, int extra);`

The parameter extra is for your help. It has a default value assigned to it. You can change the default value of the parameter, however, **DO NOT CHANGE any other part of the function prototype.**

Question 8

Write a C++ recursive function `PrintPattern()` to print the following pattern using recursion. No loops are allowed whatsoever, and you can write a maximum of one helper functions apart from the main function. For example, calling your function with these arguments `PrintPattern(1,3)` should print the following pattern. Your function prototype must be as follows recursive function.

void PrintPattern(int start, int end);

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

Question 9

Write a C++ recursive function `PrintPattern` to print following pattern using recursion. No loops allowed whatsoever, and you can write maximum two functions apart from main function. For example, calling your function with these argument `PrintPattern(5,'*')` should print following pattern. Your function prototype must be as follows recursive function.

Prototype: `void PrintPattern(int value, char ch);`

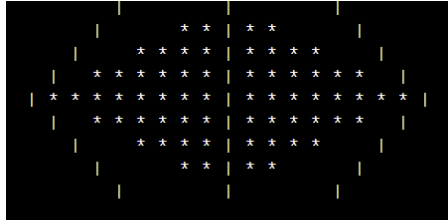
```
      *
     * *
    * *
   * *
  * *
 * * *
```

Question 10

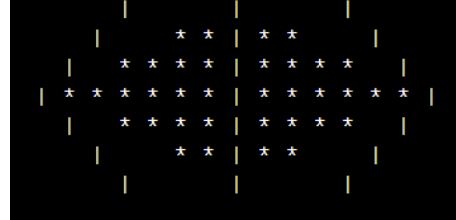
Write a C++ recursive function PrintPattern to print following pattern using recursion. No loops allowed whatsoever, and you can write other helping recursive functions (max 4function). For example, calling your function with these argument PrintPattern (5,1,5) should print following pattern. Your function prototype must be as follows:

Prototype: void PrintPattern(int, int ,int);

PrintPattern (5,1,5)

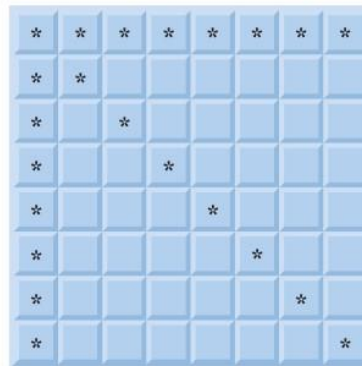


PrintPattern(4,1,4)



Question 11

Solve the Nine Queens problem recursively. The objective is to place nine queens on an empty chessboard (9x9 matrix) so that no queen is “attacking” any other, i.e., no two queens are in the same row, the same column, or along the same diagonal?



Show a solution on the screen and wait for the user to press a key before displaying the next solution. [See the Book Deitel & Deitel’s How to Program Problem 7.26]

Your function prototype must be as follows:

bool nQueens(char **b /*board*/, int n=9/*size of board*/,

int r=0/*current row number*/, int col=1/*current column number*/);

Question 12

Maze Traversal The grid of hashes (#) and dots (.) in the following figure is a two-dimensional array representation of a maze. In the two-dimensional array, the hashes represent the walls of the maze and the dots represent squares in the possible paths through the maze. Moves can be made only to a location in the array that contains a dot. There is a simple algorithm for walking through a maze that guarantees finding the exit (assuming that there is an exit). If there is not an exit, you'll arrive at the starting location again. Place your right hand on the wall to your right and begin walking forward. Never remove your hand from the wall. If the maze turns to the right, you follow the wall to the right. As long as you do not remove your hand from the wall, eventually you'll arrive at the exit of the maze. There may be a shorter path than the one you've taken, but you are guaranteed to get out of the maze if you follow the algorithm.

```

# # # # # # # # # # # #
# . . . # . . . . . #
# . # . # . # # # . #
# # # . # . . . . # . #
# . . . . # # # . # . .
# # # # . # . # . # . #
# . . # . # . # . # . #
# # . # . # . # . # . #
# . . . . . . . # . #
# # # # # # . # # # . #
Exit  [X] . . . . . # . . . #
# # # # # # # # # # # #

```

Write recursive function `mazeTraverse` to walk through the maze. The function should receive arguments that include a 12-by-12-character array representing the maze and the starting location of the maze. As `mazeTraverse` attempts to locate the exit from the maze, it should place the character `X` in each square in the path. The function should display the maze after each move, so the user can watch as the maze is solved.



Here is the final solution.

```

# # # # # # # # # # # #
# . . . # X X X X X X #
# . # . # X # # # # X #
# # # . # X X X X # X #
# . . . . # # # X # X X
# # # # . # . # X # X #
# . . # . # . # X # X #
# # . # . # . # X # X #
# . . . . . X X X # X #
# # # # # # X # # # X #
X X X X X X X # X X X #
# # # # # # # # # # # #

```

Your Code should solve this maze as well

```
Exit 
# # # # # # # # # # # # #
# . . . # . . . . . #
. . # . # . # # # . #
# # # . # . . . . # . #
# . . . . # # # . # . .
# # # # . # . # . # . #
# . . # . # . # . # . #
# # . # . # . # . # . #
# . . . . . . . # . #
# # # # # # . # # # . #
# . . . . . # . . . #
# # # # # # # # # # # #
Entry 
```

Your function prototype must be as follows:

```
bool mazeSolver( char **, int nrows, int ncols, int srow/*starting row */, int scol /*starting
column*/);
```

Question 13

Write a C++ program to find out the substrings which starts and ends with same character.

Input : S = "abcab"

Output : 7

There are 15 substrings of "abcab" a, ab, abc, abca, abcab, b, bc, bca bcab, c, ca, cab, a, ab, b

Out of the above substrings, there are 7 substrings : a, abca, b, bcab, c, a and b.

Input : S = "aba"

Output : 4

The substrings are a, b, a, and aba

Function Prototype:

```
int countSubstrs(string str, int starting, int index, int size);
```

Question 14

Write a recursive method named `permut` that accepts two integers `n` and `r` as parameters and returns the number of unique permutations of `r` items from a group of `n` items. For given values of `n` and `r`, this value $P(n, r)$ can be computed as follows:

$$P(n, r) = \frac{n!}{(n - r)!}.$$

For example, `permut(7, 4)` should return 840. It may be helpful to note that `permut(6, 3)` returns 120, or $840 / 7$.

Your function prototype must be as follows:

`long permute(int n, int r);`

Attention:

- Maximum of 2 helper function allowed unless stated otherwise.
- Helper functions must also be recursive.
- All functions must contain no loops and use only recursion.
- You cannot use **global, static and pass by reference variables** in this assignment.
- Do not use any **string or math library** unless specified for that question or part ONLY.
- Do not use any built-in function. Zero marks will be awarded in the entire assignment if it is found.
- We will be running your code against our test cases, and a test case failure or a segmentation fault/incorrect result or even syntax error will result in zero marks.
- Your code must be GENERIC.
- **Do not edit Function Prototypes** as that will result in a direct zero.

***Good Luck ***