NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCES ISLAMABAD CAMPUS

CS-1004 Object Oriented Programming Spring-2024
ASSIGNMENT-01
Section (A, B, C, D, E, F and G)
Deadline: 22nd February 2024 03:00 PM

Instructions:

1) **10 Marks for correct Submission. Correct submission means submit only one cpp file named "Submission.cpp" that must be runnable (No errors) with testcases provided to you. These marks are only applicable if you have done 30% of assignment. Don't remove any existing function from Submission.cpp.**
2) Do not use any String or math libraries (such as cmath, cstring, string etc) and also do not use built-in function (such as strlen, strcmp etc). **Caution: zero marks** will be awarded.
3) Your code must be generic. Do not edit Function Prototypes. **zero marks** will be awarded.
4) Test Cases are provided for each question. Your code will be evaluated with similar test cases. If the required output is generated, you will be awarded full marks. Failing to generate the correct output will result in zero marks. Total Marks: 200.
5) For **PrintPattern** questions, the **output** should be properly displayed and well presented. There will be **no gtests** for **PrintPattern** questions. **Static and global variables are not allowed for recursive functions.**
6) **Plagiarism** of any kind (copying from others, copying from the internet, etc) is not allowed. If found plagiarized, you will be awarded zero marks in the **whole assignment category**.
7) **Test cases:** Test cases (in gtest) will be shared with you on Google Classroom**. 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.**
8) **Submission Guidelines:** Dear students, we will be using **auto-grading tools (gtest),** so failure to submit according to the below format would result in zero marks in the relevant evaluation instrument.
   a) Copy all your main and helping functions in submission.cpp given to you. Please don't include the main function while submitting the file. And don't remove test cases (in testcases.cpp) or function prototypes (in Submission.cpp). Submission.cpp file must contain your name, student-id, and assignment # on the top of the file in the comments.
   b) Move you submission.cpp in one folder. The folder must contain only submission.cpp file (no binaries, no exe files etc.,). If we unable to download your submission due to any reason you will be awarded zero mark.
   c) Rename the folder as ROLL-NUM_SECTION (e.g. 23i-0001_A) and compress the folder as a zip file. (e.g. 23i-0001_A.zip). Only zip file will be acceptable.

Note: Follow the given instruction to the letter, failing to do so will result in a zero.

Pointers

Q1: Base Converter [Marks:20]

Make a Generic code converter for all bases using the knowledge you acquired form your Digital Logic Design class. Remember the that the base can be any so handle except negative.

For reference you may use this link: https://www.youtube.com/watch?v=FFDMzbrEXaE

Function Prototype:

char* BaseConverter (float number, int base);

Q2: MS Word [Marks:100]

You are given a 3D-pointer array representing a book in MS Word. The book has multiple pages, and each page contains multiple lines. You are required to implement different operations of MS word such as cut, copy, paste. You also need to implement the insert and delete function of MS Word for line, page and text. You need to efficiently allocate and deallocate the memory.

Constraint you must follow

- A line must have a **maximum** of 40 characters (Space is included in characters).

- A page must have a **maximum** of 10 lines.

- In case of adding content to a page or a line, the whole word should be written at the next available page/line.

Function Prototype:

void makeBook (char* text, char***& book, int& totalPages)

void cut (char***& book, int& totalPages, int FromPage, int ToPage, int FromLine, int ToLine, int FromWord, int ToWord, char*& clipboard)

void copy (char***& book, int& totalPages, int FromPage, int ToPage, int FromLine, int ToLine, int FromWord, int ToWord, char*& clipboard)

void paste (char***& book, int& totalPages, char* clipboard, int atPage, int atLine, int afterWord)

void Delete (char***& book, int& totalPages, int page) *// For page deletion* void Delete (char***& book, int& totalPages, int page, int line) *//For line deletion*

void DeleteText (char***& book, int& totalPages, char* text) *//For text deletion*

void insert (char***& book, int& totalPages, int position) *// insert empty page after position*

void insert (char***& book, int& totalPages, int page, int position) *// insert empty line on page at position*

void insertText (char***& book, int& totalPages, int page, int line, int word, char* text)

// Insert text after the word.

void removeDoubleSpaces (char***& book, int& totalPages)

void concatenateBooks (char***& book1, int totalPages1, char***& book2, int totalPages2, char***& newBook, int& newPages)

char* toString (char***& book, int totalPages) *//return book with format given below*

toString Return:

```
---------- Page 1 ----------
In a tiny village nestled between
rolling hills and babbling brooks, there
lived a peculiar cat named Whiskers.
Whiskers wasn't your ordinary feline; he
had a knack for solving mysteries. One
day, the village's prized possession,
the Giant Cheese Wheel, went missing.
Whiskers, with his magnifying glass in
paw and detective hat perched atop his
head, embarked on a thrilling quest. His
---------- Page 2 ----------
journey took him through meadows and
```

over bridges until he stumbled upon a
suspicious trail of cheesy paw prints
leading to Farmer Brown's barn. Inside
the barn, Whiskers discovered a group of
mischievous mice having a cheese feast.
Among them was a mouse named Munch, the
mastermind behind the cheesy caper.
However, instead of scolding them,
Whiskers proposed a deal: they could
---------- Page 3 ----------
have a cheese party every month as long
as they promised not to pilfer the Giant
Cheese Wheel.
---------- Page 4 ----------
From that day forward, the
village enjoyed monthly cheese
celebrations, and Whiskers became a
hero. As for Munch and his mouse
friends, they learned that sometimes,
it's better to negotiate than to nibble.
And so, peace reigned in the village,
thanks to the clever detective work of
Whiskers, the cheese-loving cat.

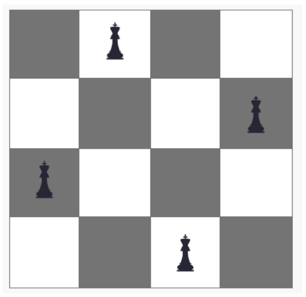
Recursive Functions

Q3: Royal Harmony [Marks:25]

Once upon a time, a King had multiple queens. However, they often engaged in heated arguments and constant quarrels. The King, wanting to restore peace in his kingdom, decided to find a way to place the queens in a palace where they could not see each other. Each queen had the ability to observe in up, down, left, right, and diagonally.

As the wise advisor to the King, you were tasked with arranging the n queens on nxn chessboard in such a way that no queen could threaten another. The challenge was to position them strategically, ensuring that no queen had a direct line of sight to any other queen, thus preventing further conflicts among them.

This is a solution for the 4 Queen problem. The expected output is in the form of a matrix that has '1's for the blocks where queens are placed and the empty spaces are represented by '0'. There are multiple solutions to this question according to the approach (row-wise/ column-wise).



Function Prototype:

bool royalHarmony (int**& board, int row, int column, int queens);

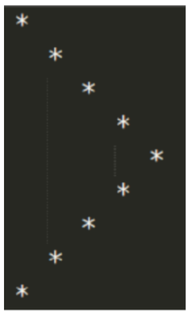
Note: Definition of this function should be recursive. Straight zero marks if loops are used in function definition. Your solution will be tested on input 5 or greater than 5.

Q4: Patterns [Marks:25]

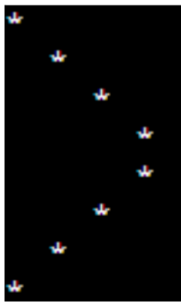
Write a C++ recursive functions to print the following pattern using recursion. No loops allowed whatsoever, and you can write other helping (recursive) functions. All functions i.e. Main and helping functions must be recursive.

1. void RecursivePattern1(int start, int end);

   Pattern for RecursivePattern1(1,10). Pattern for RecursivePattern1(1,9).



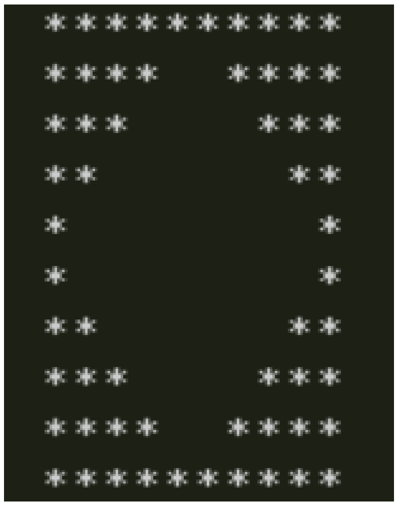
```
*
 *
  *
   *
    *
   *
  *
 *
*
```



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

2. void RecursivePattern2(int n1, int n2); *// Hollow Diamond*

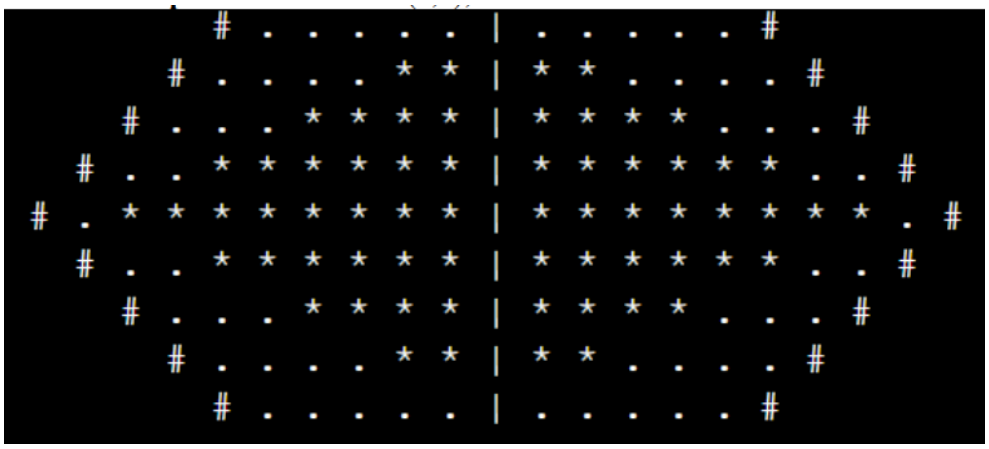
   Pattern for RecursivePattern2(5,5);



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

3. void RecursivePattern3(int, int);

   Pattern for RecursivePattern3(5,5);



```
        # . . . . . | . . . . . #
      # . . . . * * | * * . . . . #
    # . . . * * * * | * * * * . . . #
  # . . * * * * * * | * * * * * * . . #
# . * * * * * * * * | * * * * * * * * . #
  # . . * * * * * * | * * * * * * . . #
    # . . . * * * * | * * * * . . . #
      # . . . . * * | * * . . . . #
        # . . . . . | . . . . . #
```

Struct

Q5: CodeSaver Bank [Marks:20]

In the lively town of Codeville, a bunch of friends, including Alex, it was time to save money. Each friend wanted their own special place to track their savings at the local bank, "**CodeSaver Bank**." They thought of creating a cool "**SavingAccount**" structure to hold everyone's account details. Now, they need your help to make this happen! Can you assist Alex and his friends in creating the "SavingAccount" structure with all the important information for each user? Join them on this coding adventure, and let's make savings fun for everyone!

struct SavingAccount *//create a struct SavingAccount with following data members*

char* name
float annualInterestRate
double savingBalance
char* accountNum *// the account numbers will be from "SA00" to "SA99". Each account must have a unique account number that has not been assigned to other customer before.*

The following two will be passed as arguments to functions mentioned below

1. **SavingAccount *savers[100]** – *an array of 100 SavingAccount pointers.*

2. **int accountsOpen** – *an integer to store the current accounts open and can also be used as an index for the customer's array.*

Implement the following functions:

void OpenCustomerAccount (SavingAccount * savers [], int & accountsOpen, char* name, float interestRate, double balance) *// a function to create a new account and assign it an account number.*

float calculateMonthlyInterest (SavingAccount * saver) *// that calculates the monthly interest by multiplying the balance by annualInterestRate divided by 12*

void modifyInterestRate(SavingAccount * saver, float newValue)

int SearchCustomer (SavingAccount * savers[], int accountsOpen, char* accountNum) *// a function that searches for an account using an account number. If the customer is found it returns the array index otherwise return -1*

bool UpdateAccountBalance (SavingAccount * savers[], int accountsOpen, char *accountNumVal, double balanceVal) *// a function that updates a customer's balance*