NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCES ISLAMABAD CAMPUS
Programming Fundamentals (CS1002)| FALL2022

Due Date: 26th November 2022 on GCR

Assignment 5 – Functions, Arrays, Two-Dimensional Arrays
[100 marks]

Submission: For each question, make a separate file. i.e., Q1.cpp, Q2.cpp, ..., Q5.cpp. Combine all your work (all files) in one .zip file. Use proper naming convention for your submission file. Name the .zip file as SECTION_REG# _02.zip (For example, A_22i0412_02.zip). Submit .zip file on Combined Google Classroom within the deadline. Failure to submit according to the above format would result in deduction of 10% marks.

Plagiarism: Plagiarism cases will be dealt strictly. If found plagiarized, both the involved parties will be awarded zero marks in this assignment, all of the remaining assignments, or even an F grade in the course. Copying from the internet is the easiest way to get caught! Try to learn by doing assignments yourself.

Deadline: The deadline to submit the assignment is **26th November 2022** at **11:59 PM**. Late submission with marks deduction will be accepted according to the following criteria
Time Deduction

- Submission within next 24 hours 50%
- Submission after 24hrs but within 48 hours 75%
- Submissions received after 48 hours 100%

Note: Correct and timely submission of the assignment is the responsibility of every student; hence no relaxation will be given to anyone.
Comments: Comment your code properly. Write your name and roll number (as a block comment) at the beginning of the solution to each problem.
Tip: For timely completion of the assignment, start as early as possible.

NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCES ISLAMABAD CAMPUS
Programming Fundamentals (CS1002)| FALL2022

Question # 1 Matrix Multiplication [15 Marks]

Write a C++ program that takes two matrices of order Row-1 x Column -1 and Row-2 x Column-2 respectively. Then, the program multiplies these two matrices (if possible) and displays the results on the screen.

1. Create a *verify_order* function that returns true if multiplication is possible and false if multiplication is not possible.
2. To store your matrixes, since we have not covered dynamic memory allocations and flexible arrays are not available for all versions of compilers, you should use matrixes of 10 X 10. Note that only part of the matrix is used, rest of the cells are not used. For example, for a 2 x 2 matrix, only 2 rows and 2 columns contain valid values. Rest of the array cells should not be used.
3. The elements of the two matrices (values) should be read from the user.

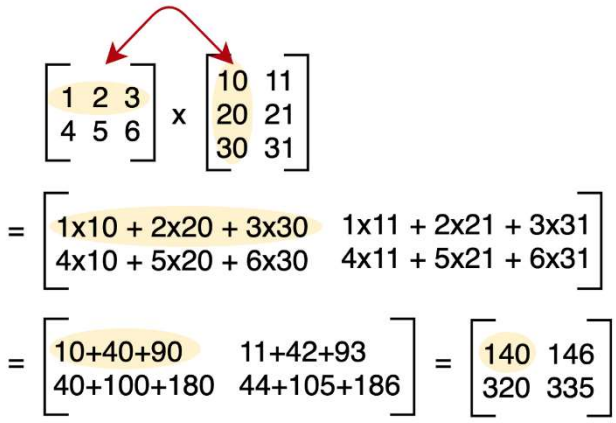

Figure 1 Matrix Multiplication Example

Question # 2 String manipulation [20 Marks]

Handling text is an important feature of any programming languages and many modern programming languages provide special support for handling text like "string" in C++. The datatype string provides many useful functions for manipulating text, such as pattern matching in strings, replacing part of string with another string, and extracting a substring.

However, these features were not always available and often the programmers had to resort of writing their own functions to get the job done. For this assignment, the students shall create their own set of methods for processing strings. To complete the assignment, the students should:

1. Create a menu in C++ (from main) that asks the user to enter a sentence. For ease, the sentence entered by the user can be stored in a *string* variable (we will use the string datatype from C++ for this).
2. Create a function string *substring (int start, int end, string text)* that creates a sub-string from the given text, starting from the *start* till *end.*

Example : if the function is called from main as follows

```
string text = "this is a sample text" ;
string new_string = substring(0, 3);
// new_string should contain "this"
```

For extracting the sub-string, you are not allowed to use pre-implemented string functions. Instead iterate all elements of text and create the substring yourself.

3. Create a function int *find (string pattern, string text)* that searches *text* for occurrence of *pattern* and returns the index where the pattern is found.

   Example : if the function is called from main as follows

   ```
   string text = "this is a sample text" ;
   int index = find ("is", "this is a sample text");
   // index should contain the starting location of "is"
   ```

   For extracting the sub-string, you are not allowed to use pre-implemented string functions. Instead iterate all elements of text and create the substring yourself.
4. Create a function string *replace (string pattern, string text, int start, int end)* that replaces the characters in the *text* with characters from *pattern*.
5. Create a function *string invert (string text)* , that returns the text in reverse order (the last character becomes the first and so on..)
6. Replacing the pattern can result in increasing the length of the string *text.*
7. Values such as start, end cannot be negative. Similarly, pattern being searched cannot be empty. Create a call necessary error_checking functions where needed.

Question # 3 Battleship Game [20 Marks]

We have now sufficient information to start creating simple 2-D games. For this question, we will implement a simple Battleship game using a 2-D Array. The basic grid of the game consists of a 20 X 20 matrix. The program should display the initial grid to the user and ask them to place their battleship on the board.

For example, in the shown example, the user has placed 1 battleship of length 5 at location [0][2]. The orientation of the battleship is horizontal ('H').

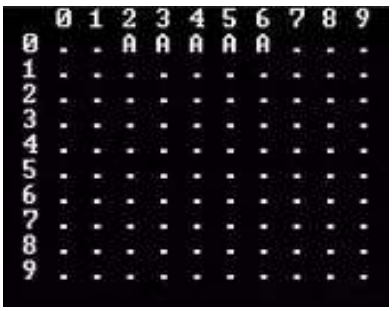

Figure 2 Placement of Battleships

NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCES ISLAMABAD CAMPUS
Programming Fundamentals (CS1002)| FALL2022

For this program,

1. Create a grid of 20 X 20 and ask the user to place ***n*** number of battleships, with battleship of some fixed length (length should be defined as constant and may be set to different values before compiling and executing). In figure 2, only 1 battleship of size 5 is placed. The user also specifies the orientation of the battleship. In figure 2, the orientation is horizontal. The possible values for orientation is horizontal or vertical ('h' or 'v').
2. The empty spaces on the grid should be indicated by some symbol, for example '.' Or '0'. This should be defined as constant and changing it at one location should be enough to change the display. Figure 3 shows the grid with battleship placement hidden. Also note that the grid locations are shown by 0 here instead of '.' As in the previous example.

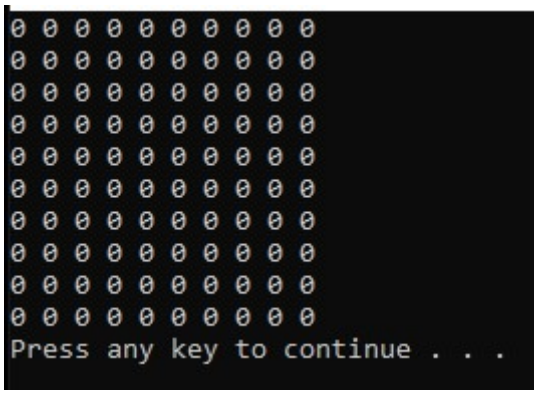

Figure 3 Battleships are hidden in the battle round

3. After placing each battleship, show the grid to the user with the placed battleship (you can use any character to indicate a battleship).
4. When all the battleships are places, clear the screen of all inputs and start the 'battle round'. In the battle round, a second user now sits on the computer and the grid is shown to the user, but the battleships are hidden (i.e, the user cannot see the battleships). User is given 5 Missiles that he/she can fire at the grids.
5. User fires a missile by giving a coordinate. If any part of the battleship is present on that coordinate, then the battleship becomes visible, and the symbols are changed to "H" (indicating the battleship is hit).
6. Score is calculated based on how many battleships user can hit by firing missiles at "guessed" coordinates.
7. The program should display the scorecard and the grid containing the remaining battleships, the destroyed battleships.

NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCES ISLAMABAD CAMPUS
Programming Fundamentals (CS1002)| FALL2022

Question # 4 Cipher Implementation – Substitution Cipher [20 Marks]

The Baconian cipher is a substitution cipher in which each letter is replaced by a sequence of 5 characters. Each letter is assigned to a string of five binary digits. These could be the letters 'A' and 'B', the numbers 0 and 1 or whatever else the programmer may desire. We are going to implement the cipher so that all 26 English language alphabets have an equivalent code defined.

For example,

A can be replaced by 00000

B can be replaced by 00001

…..

Z can be replaced by 11001

Using the cipher the text "AB" will be coded as "`0000000001`".

Hint : Use string to store the input, rather than an int.

You can use an array to store the encodings. Do not hard code the encodings in any condition structure such as switch or if. Use an Array to store the mapping.

1. Write a utility function, *string look(char ch),* that takes a char as input and returns the encoded string for that char. Write a similar function to support decoding.
2. Write a function *string cipher(string text)* which takes as input a piece of text as parameter and returns an encrypted message.
3. Write a function *string de_cipher(string msg)* that takes an encrypted message as input and returns the un-encrypted message.
4. Write a main to demonstrate your program's working.
5. Please note that user will provide the text to encrypt in the form of a sentence. Your program will need to do the necessary processing, particularly when decrypting the message (5 chars in the encrypted string correspond to one char in un-encrypted string).

Question # 5 Find the Cheese in Maze [25 Marks]

For this assignment, you will create a simple Maze game, where the rat tries to find the cheese. The maze is characterized by outer boundary wall and a set of inner walls/obstacles. Inside the maze there are open paths, on which rat can travel in its search of the cheese.

The structure of the maze can be represented using a two-dimensional Array. An example of such as array is shown:

```
int         maze[ROWS][COLUMNS]={
                {42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42},
                {42, 80, 0, 0, -1, 0, 0, 0, -1, 0, -1, 0, 0, 0, 0, 42},
                {42, -1, -1, 0, -1, -1, -1, 0, -1, 0, -1, 0, -1, -1, -1, 42},
                {42, -1, -1, 0, -1, -1, -1, 0, -1, 0, -1, 0, -1, -1, -1, 42},
                {42, 0, 0, 0, 0, 0, -1, 0, 0, 0, -1, 0, 0, 0, 0, 42},
                {42, 0, -1, -1, -1, 0, -1, 0, -1, 0, -1, 0, -1, -1, 0, 42},
                {42, 90, -1, -1, -1, 0, -1, 0, -1, 0, -1, 0, -1, -1, 0, 42},
                {42, 0, -1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, -1, -1, 0, 42},
                {42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42},
                        };
```

in the above code example, the maze is initialized statically. The following variables can be used to define the structure of the array.

define ROWS 9 // you can also use *const int ROWS = 9;* as global variable.
define COLUMNS 16 // you can also use *const int COLUMNS = 16;* as global variable.
and
int mouseshape=80;
int cheeseshape=90;
int wallshape = -1;
int boundry = 42;

The number '80' represents the mouse (instead of showing a mouse, we show number '80').
This can of course be easily replaced by any other number or character. Cheese is represented by '90'. It can be easily represented by any other number/character/shape. The interior walls are represented by '-1' while outer boundary walls are represented by 42. Please note that the different configurations of the maze can be generated simply by varying the size of the array, changing the location of the inner walls/obstacles and the location of the cheese.

Task 1. Create a program that encodes and initializes the above defined maze.

Task 2. Implement a function runMouse( ) that implements an algorithm for traversing the maze.

The body of the function should move the mouse through the maze and try to find the Cheese. Of course, mouse can only move through open spaces and cannot go through outer boundary walls or inner walls. A few algorithms can be implemented for mouse movements.

(i) Random: Each turn mouse may move randomly in one direction (keeping in view the location of walls /obstacles). Such an algorithm has a low chance of success but is simplest to implement.

You should implement a function runMouse(…) that implements the algorithm to control mouse movement.

Bonus Task

A better way to get to cheese is through *backtracking*. For an iterative implementation, you can use an array to track previous mouse location. One simple algorithm is to record the mouse movements in an array. Each turn simply finds an empty location and move there. Keep moving until either cheese is found, or there are no empty locations (you have arrived at a dead end). When at a dead end, move back in the backtrack array and find an unexplored path. An unexplored path is the one that does not have obstacles on all sides. Keep repeating until the cheese is found.

int pathOfMouse[100][2];

The array can store up to 100 moves, storing row and column number (identifying the mouse location). This is used for backtracking.

Note: *runMouse(..)* functions terminate when cheese is found or 1000 iterations/recursive calls have been executed without finding cheese.

Sample Run

Results from first three iterations/recursive calls is shown using the simple find the zero (with backtracking) algorithm.

```
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
42  0  0  0 -1  0  0  0 -1  0 -1  0  0  0  0 42
42 -1 -1  0 -1 -1 -1  0 -1  0 -1  0 -1 -1 -1 42
42 -1 -1  0 -1 -1 -1  0 -1  0 -1  0 -1 -1 -1 42
42  0  0  0  0  0 -1  0  0  0 -1  0  0  0  0 42
42  0 -1 -1 -1  0 -1  0 -1  0 -1  0 -1 -1  0 42
42  0 -1 -1 -1  0 -1  0 -1  0 -1  0 -1 -1  0 42
42 90 -1  0  0  0  0  0 -1  0  0  0 -1 -1 80 42
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
mouseCounter 0
zero found
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
42  0  0  0 -1  0  0  0 -1  0 -1  0  0  0  0 42
42 -1 -1  0 -1 -1 -1  0 -1  0 -1  0 -1 -1 -1 42
42 -1 -1  0 -1 -1 -1  0 -1  0 -1  0 -1 -1 -1 42
42  0  0  0  0  0 -1  0  0  0 -1  0  0  0  0 42
42  0 -1 -1 -1  0 -1  0 -1  0 -1  0 -1 -1  0 42
42  0 -1 -1 -1  0 -1  0 -1  0 -1  0 -1 -1 80 42
42 90 -1  0  0  0  0  0 -1  0  0  0 -1 -1  1 42
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42

mouseCounter 1
zero found
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
42  0  0  0 -1  0  0  0 -1  0 -1  0  0  0  0 42
42 -1 -1  0 -1 -1 -1  0 -1  0 -1  0 -1 -1 -1 42
42 -1 -1  0 -1 -1 -1  0 -1  0 -1  0 -1 -1 -1 42
42  0  0  0  0  0 -1  0  0  0 -1  0  0  0  0 42
42  0 -1 -1 -1  0 -1  0 -1  0 -1  0 -1 -1 80 42
42  0 -1 -1 -1  0 -1  0 -1  0 -1  0 -1 -1  1 42
42 90 -1  0  0  0  0  0 -1  0  0  0 -1 -1  1 42
```