NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCES
ISLAMABAD CAMPUS

CS-1004 Object Oriented Programming
Assignment No. 1

Section (A, B, C and D)

Submission Information

Submission Deadline: **February 24th, 2023 at 10:00 AM**. You are supposed to submit your assignment on GOOGLE CLASSROOM (CLASSROOM TAB not lab). Only ".ZIP" files are acceptable. Other formats should be directly given ZERO. Correct and timely submission of the assignment is the responsibility of every student, hence no relaxation will be given to anyone. Late Submission policy will be applied as described in course outline.

Tips: For timely completion of the assignment, start as early as possible.

Plagiarism: Plagiarism is not allowed. If found plagiarized, you will be awarded zero marks in the assignment (copying from the internet is the easiest way to get caught).

Instructions:

Dear students, we will be using auto-grading tools, so failure to submit according to the format below would result in zero marks in the relevant evaluation instrument.

I. For each question in your assignment, make a separate cpp file e.g. for question 1, make ROLL-NUM_SECTION_Q#.cpp (22i-0001_A_Q1.cpp) and so on. Each file that you submit must contain your name, student-id, and assignment # on top of the file in comments.
II. Combine all your work in one folder. The folder must contain only .cpp files (no binaries, no exe files etc.).
III. Run and test your program on a lab machine before submission.
IV. Rename the folder as ROLL-NUM_SECTION (e.g. 22i-0001_A) and compress the folder as a zip file. (e.g. 22i-0001_A.zip). Do not submit .rar file.
V. Submit the .zip file on Google Classroom within the specified deadline.
VI. Submission other than Google classroom (e.g. email etc.) will not be accepted.
VII. The student is solely responsible to check the final zip files for issues like corrupt file, virus in the file, mistakenly exe sent. If we cannot download the file from Google classroom due to any reason it will lead to zero marks in the assignment.
VIII. Displayed output should be well mannered and well presented. Use appropriate comments and indentation in your source code.
IX. Total Marks: 100.

Learning Objective

The AIM of this assignment is to have hands-on Pointers, dynamic memory allocation and multidimensional arrays.

Warning:

- If there is a syntax error in the code, zero marks will be awarded in that part of the assignment.
- Your code must be generic and handle all errors and exceptions.

Task 1: Reward distance detection in 4-dimensional matrix

In this task you will design a game in which the player has to guess the location of rewards in a 4-dimensional matrix, and earns a score based on the number of correct guesses.

The matrix represents a 3D space where each point is represented by x, y and z coordinates, and the fourth dimension is time, so essentially you can think of the matrix as representing the state of a 3-dimensional space at different points of time. Each index of the matrix represents a particular location in the 3D space at a particular point in time. You will consider only 3 time points, past, present and future. The fourth dimension is therefore fixed at 3. For the other three dimensions you should take user input, but they must always be the same value, for example, the length, breadth and depth of your matrix may all be 5. In this case, the user will input 5 and you will generate a 5 X 5 X 5 X 3 matrix.

You will initialize the matrix such that it has 60% empty locations, while 40% of the locations contain a reward. The rewards must be placed randomly in the matrix. A location of the matrix containing a reward means that a reward exists in those particular coordinates only at the particular time represented by the time index, i.e. either in the past, the present or the future.

You should implement a menu with 3 items linked to the following functions that you should implement:

1. Initialize: calls an init(int dimension) function that takes as the dimension as a parameter and initializes the 4D array with empty locations and rewards.
2. Print matrix: Calls a print_reward_locations function that prints the indices at which the rewards have been placed in a particular run of the game, as follows:
   Reward_1: 3,4,4, PRESENT
   Reward_2: 2,3,1, PAST
3. Play: Calls a play_game function that begins the game.

When the user selects the *Play* option, the game should proceed as follows:

- Ask the user to select a starting location (in the form of a 4-dimensional index where the first 3 dimensions represent coordinates x, y, and z of a 3D space). Thus, the user can choose to start in the past, present or future and move across time and space as the game proceeds.
- Calculate the distances* of all the rewards from the user and display how far each the rewards is from where the user is, as follows:
  Reward_1: Distance is 2
  Reward_2: Distance is 13
- Now ask the user to move to another location where he thinks a reward might be.
- When the user enters another 4-dimensional index, move the user to that location – now if the user collides with a reward (distance to reward is zero AND the time value matches), he gets one point. If the reward is at the same coordinates but at a different point of time, e.g. the user is in the present but the reward is at those coordinates in the past, the user gets half a point. Otherwise the user gets zero points.
- Now again display the distances of rewards from the user's new location and repeat the process above, i.e. allow the user another move. After each move, the game

should display the score for that move as well as the combined score from all previous moves.
- In this way, the user is allowed to move N times, where N is input by the user at the start of the game (i.e. the start of the play_game function).
- Calculate the total score of the user.

*distances should be calculated as follows. If the user is on index P1(x0,y0,z0,t0), and the reward R1 is on index P2(x1,y1,z1,t1), the distance from the user is:

$$\boldsymbol{d(P1, P2) = \sqrt{(x1 - x0)^2 + (y1 - y0)^2 + (z1 - z0)^2}}$$

Note that all array manipulation must be done through pointer notation. Use of the subscript operator for indexing is not allowed. All arrays must be dynamic.

Task 2: Making your own (limited) String library

The String library in C++ contains many useful functions for string manipulation, such as copy, concatenate or compare two strings. Your task is to use char pointers to implement from scratch the following functions of the string library:

int toupper (char* ch);
Return the uppercase of ch

int tolower (char* ch);
Return the lowercase of ch

int isspace (const char* ch);
Return 1 if ch is a white space (blank ' ', carriage return '\r',
newline '\n', tab '\t', form feed '\f', vertical tab '\v') and 0 otherwise

char * strncpy (char * dest, const char * src, size_t n)
Copy at most n characters from src into dest, return dest

int strcmp (const char * cstr1, const char * cstr2)
Compare cstr1 and cstr2. Return 0 if cstr1 is equal to cstr1,
less than zero (usually -1) if cstr1 is less than cstr2,
more than zero (usually 1) if cstr1 is more than cstr2.

char * strncat (char * dest, const char * src, size_t n)
Append at most n characters from src into dest, return src.

char * strstr (char * cstr1, char * cstr2);
Return a pointer to the first occurrence of cstr2 in cstr1

char * strtok (char * cstr, const char * delim)
Tokenize cstr with delim as the delimiters

Note that all array manipulation must be done through pointer notation. Use of the subscript operator for indexing is not allowed. All arrays must be dynamic.