# National University of Computer and Emerging Sciences, Lahore Campus

| | | | | |
|---|---|---|---|---|
| | **Course:** | **Computer Programming** | **Course Code:** | **CL-1004** |
| | **Program:** | **BS (Computer Science)** | **Semester:** | **Spring 2024** |
| | **Duration:** | **120 Minutes** | **Total Marks:** | **30** |
| | **Paper Date:** | **19-March-24** | **Weight** | **30%** |
| | **Section:** | **B** | **Page(s):** | **4** |
| | **Exam:** | **Midterm** | **Reg. No.** | |

**Instruction/Notes:**

1. Understanding the question paper is also part of the exam, so do not ask any clarification.
2. Make sure to switch off your mobile phones before the Exam starts.
3. No USB's are allowed. Please see that the area in your threshold is clean. You will be charged for any material which can be classified as 'helping in the paper' found near you.
4. Talking/Discussion is not allowed. It is your responsibility to protect your code and save it from being copied. If you don't protect it all matching codes are considered copy/cheating cases.
5. You are not allowed to use internet for any purpose.
6. For each question you have to submit a single cpp file e.g. l221234-q1.cpp. Don't zip or compress your solution. Your solution will not be marked if it doesn't adhere to the submission instructions

**Submission Path:**

**\\cactus1\Xeon\Spring 2024\Muhammad Hasan\OOP\BSE 2B**

---

**Q. No. 1:** **[15]**

**Implementing Tic Tac Toe using C++ Classes and 2D Pointers**

Tic-tac-toe is played by two players A and B on a **3 x 3** grid. The rules of Tic-Tac-Toe are:

1. Players take turns placing characters into empty squares ' '
2. The first player A always places 'X' characters, while the second player B always places 'O' characters.
3. 'X' and 'O' characters are always placed into empty squares, never on filled ones.
4. The game ends when there are three of the same (non-empty) character filling any row, column, or diagonal.
5. The game also ends if all squares are non-empty.

Your implementation should allow two players to take turns placing their marks (X and O) on a 3x3 grid. The game should detect when a player has won or when there is a draw.

Your task involves the following steps:

Define a class named TicTacToe that represents the game. Implement necessary member functions and variables within the class to facilitate gameplay. Utilize a 2D pointer to represent the game board. Implement functions to display the current state of the board, allow players to make moves, and check for a win or draw condition. Ensure proper initialization and cleanup of memory.

Implementation should follow these guidelines:

- Use appropriate access specifiers for class members.
- Use dynamic memory allocation for the game board.
- Detect when a player has won horizontally, vertically, or diagonally.
- Provide appropriate messages to players indicating the game state (win, draw, or continue).
- Ensure that memory allocated dynamically is deallocated properly to avoid memory leaks.

Your code should include comments to explain the purpose of each function and any complex logic. Additionally, provide a simple demonstration of the game's functionality by simulating a few moves and displaying the game board after each move.

| | | |
|---|---|---|
| X | | |
| | X | |
| O | O | X |

Player **A** wins

| | | |
|---|---|---|
| X | X | O |
| X | O | |
| O | | |

Player **B** wins

| | | |
|---|---|---|
| X | X | O |
| O | O | X |
| X | O | X |

**Draw**

**Q. No. 2:** **[15]**

The native int type in C++ cannot store large integers of more than 10 decimal digits. This can be troublesome for applications, which require much larger range of integers. In this question you have to implement a HugeInt class that behaves like the int type, but can handle integers up to 50 decimal digits. The HugeInt class should have an integer pointer as its private data member and another integer to store the maximum number of digits in the HugeInt.

You will use dynamic array to store digits of various Huge Integers, for 50 decimals digits you will need (5 integers) not 50, so allocate memory accordingly. Note: the integers can be both positive and negative numbers.

Functions:

- The HugeInt class must have the following constructors and a destructor:

o HugeInt(); // default constructor, set the value to be 0

o HugeInt( int x ); // set the value to be x

o HugeInt( HugeInt & ); //Copy Constructor

o ~ HugeInt();

- Assignment Operator (=) To assign one HugeInt number to other one.

- Arithmetic Operations

1. + operator Add two HugeInt numbers.

2. * operator Multiply two HugeInt numbers.

3. - operator Subtract two HugeInt numbers

- Relational operators for comparing HugeInt numbers for.

1. Equality (= =)

2. Inequality (! =)

3. Less than (< ) operator

4. Less than or Equal to <= operator

5. Greater than > operator

6. Greater than and Equal to >= operator

- Input function to take input from user

- A print function that prints a HugeInt on screen

Following are two HugeInt numbers that can appear as input:

1. 15000609000000008999990

2. 99999999999999999999999999999999999999999999

- Implement a user interactive calculator which can perform all the above mentioned operations for integers up to 50 decimal digit