University of Central Punjab
(Incorporated by Ordinance No. XXIV of 2002 promulgated by Government of the Punjab)
FACULTY OF INFORMATION TECHNOLOGY

| | |
|---|---|
| **Course** | Object Oriented Programming |
| **Semester** | Fall 2025 |
| **Assignment #** | 01 |
| **Deadline** | 29th Oct 2025 EOD (11:59 PM) |
| **Submission forum** | University portal |
| **What to submit?** | A single compressed file containing all header & CPP files. |

**Plagiarism Policy:** Copying even single line of code from other sources (internet, AI bots, peers) will be considered as plagiarism, and consequences for plagiarism are clearly declared in course outline document.

**Topics covered in this assignment:** Class, Object, dot (.) operator in C++, Access specifiers (private, public), Setter functions, Getter functions, Constructors (default, parameterized, copy, delegating), Constructor overloading, Member/constructor initializer list, Arrow (->) operator, this pointer, const data member in class, Default arguments with constructors, Passing objects by value to a function, Returning object by value from a function, Separate interface and implementation (working with multiple files), Scope resolution operator(::), Deep VS Shallow Copy

---

## TASK # 01 — The Date Class

Design and implement a class Date to represent and manipulate date information using the principles of object-oriented programming.

**Class Definition:**

```
class Date {
public:
    Date();                         // Default constructor: initialize attributes with default values
    Date(int d, int m, int y);      // Parameterized constructor: if invalid, set to default or system date
    Date(const Date& other);        // Copy constructor
    bool inputDate();               // Input date from user; return true if valid, false otherwise
    bool inputCompleteDate(int d, int m, int y); // Assign and validate date directly
    bool copyDate(const Date& other);           // Copy date from another Date object
    void retrieveDate(int& d, int& m, int& y); // Retrieve current date values via references
    void showDate();          // Display date in a readable format (e.g., DD/MM/YYYY)
```

```
    bool isEqual(const Date& other);// Compare two dates for equality
    bool isLeapYear();              // Return true if year is leap year
    // Additional Functions
    void setDay(int d);             // Setter for day with validation
    void setMonth(int m);           // Setter for month with validation
    void setYear(int y);            // Setter for year with validation
    int getDay();             // Getter for day
    int getMonth();           // Getter for month
    int getYear();            // Getter for year
    void incrementDay();            // Add one day to the date
    void decrementDay();            // Subtract one day from the date

private:
    bool validateDate();            // Validate current date values
    int day;
    int month;
    int year;
};
```

## TASK # 02 — The Matrix Class

Design and implement a class Matrix to represent and manipulate matrices using the principles of object-oriented programming.

**Objective:**
To understand dynamic memory allocation using pointers and apply object-oriented principles to represent and manipulate a 2D integer matrix.

**Restrictions:**

- Each matrix element must be stored dynamically using pointers.
- Ensure deep copy in the copy constructor — every element must be individually copied.
- Use bounds checking in setElement() and getElement() functions to avoid invalid access.
- Return false from operations (addMatrix, subtractMatrix, multiplyMatrix) if the matrix dimensions are incompatible.
- Try to avoid using any built-in array or STL container.

**Class Definition:**

```
class Matrix {
public:
    Matrix();                           // Default constructor: initialize an empty (0x0) matrix
    Matrix(int rows, int cols);         // Parameterized constructor: dynamically allocate memory
    Matrix(const Matrix& other);        // Copy constructor: deep copy of matrix

    bool inputMatrix();                 // Input all matrix elements from user (cin)
    void setElement(int r, int c, int value);// Set value at specific row and column (with bounds check)
    int getElement(int r, int c);       // Return value at specific row and column
    void displayMatrix();               // Display all elements in matrix form
    // ----- Matrix Operations -----
    bool addMatrix(const Matrix& other);    // Add another matrix (same size), store result in current matrix
    bool subtractMatrix(const Matrix& other);// Subtract another matrix (same size)
    bool multiplyMatrix(const Matrix& other);// Multiply with another matrix (matrix multiplication rules)
    void transpose();                   // Transpose current matrix
    bool isEqual(const Matrix& other);      // Compare if two matrices are identical
    // ----- Utility and Accessors -----
    int getRows();                      // Return total number of rows
    int getCols();                      // Return total number of columns
    void fillRandom();                  // Fill matrix with random integer values (e.g., 1–9)
    void clear();                       // Release memory and reset to empty matrix
private:
    const int rows;                     // Number of rows
    const int cols;                     //  Number of columns
    int** data;                         // Pointer to a dynamically allocated 2D array

    bool isValidIndex(int r, int c);        // Helper to check valid row/column index
    void allocateMatrix();              // Allocate memory dynamically for given rows and cols
    void copyFrom(const Matrix& other);     // Copy data from another matrix
};
```

# TASK # 03 — The Name Class

Design and implement a class Name to represent and manipulate name of a person using the principles of object-oriented programming.

**Motive:**
This task checks your understanding of working with character pointers and arrays, shallow vs deep copy, dynamic memory allocation and deallocation, memory leakage and dangling pointer issues.

You must implement the class, test all its functions at least once in the main() program, and verify that your implementation avoids shallow copy and memory leaks.

**Class Definition:**

```
class Name {
private:
   char* firstName;              // dynamically allocated C-string for first name
   char* lastName;               // dynamically allocated C-string for last name
public:
   // ----- Constructors
   Name(char* = nullptr, char* = nullptr); // Parameterized constructor with default values
   Name(const Name& other);                    // Copy constructor (must perform deep copy)
   // ----- Custom Copy -----
   void copyName(const Name& other);              // Copy content of one Name to another
   // ----- String Manipulation Methods -----
   void camelCase();                           // Capitalize first letter of each name

   void toLower();                            // Convert both names to lowercase

   void toUpper();                            // Convert both names to uppercase

   int nameLength();                         // Return total length of first + last names (excluding space)

   void swapNames();                             // Swap firstName and lastName values

   char* fullName();                         // Return concatenated full name (with space in between)

   // ----- Validation and Display -----
```

```
    bool isValidName();                     // Return true if both names contain only alphabets

    void display();                         // Display full name neatly on console

    // ----- Setters and Getters -----

    void setFirstName(const char* fname);       // Assign first name

    void setLastName(const char* lname);        // Assign last name

    const char* getFirstName();                 // Return pointer to first name

    const char* getLastName();                  // Return pointer to last name
};
```

## TASK # 04 — The MyString Class

**Motive:**

This task helps you understand how the standard std::string class in C++ works internally by manually implementing your own simplified version, MyString. Through this exercise, you will:

- Reinforce dynamic memory handling with character arrays
- Understand string manipulation algorithms (copy, concat, compare, search)
- Avoid memory leaks and dangling pointers
- Implement fundamental C-string operations (strlen, strcpy, strcmp, etc.) manually

**Class Definition:**

```
class MyString {

private:
    const size_t MAX_LEN;       // Maximum length size
    char* const data;   // Dynamically allocated character array
    size_t len;      // Length of the string (excluding null terminator)
public:
    // ----- Constructors
    MyString();                 // Default constructor
    MyString(const char* str);        // Construct from C-string
    MyString(const MyString& other);   // Copy constructor (deep copy)
```

```
    // ----- Member Functions -----
    int length();                       // 1. Returns length of string (like strlen)
    void copy(const MyString& source);          // 2. Copy content (like strcpy)
    void copy_n(const MyString& source, int n);   // 3. Copy first n chars (like strncpy)
    void concat(const MyString& other);         // 4. Concatenate another string (like strcat)
    int compare(const MyString& other);     // 5. Compare two strings (like strcmp)
    int find_char(char ch);                 // 6. Find first occurrence of char (like strchr)
    int find_substr(const MyString& substr); // 7. Find substring (like strstr)
    void print();                       // Display string contents
};
```