Question no 1:

Important Note:

- **Define your class in "Date.h" file, Implement all the functions of Date class in "Date.cpp" and Test your class in "Driver.cpp"**

Exercise 1:

- Create a class **Date** having the following private data members:
  Int Day
  Int Month
  Int Year
- Create an object of Date "***date1***" and run your program

Exercise 2 [Default Constructor]:

- Write a default Constructor of Date that initializes the object to 1st January 1926 and prints "*Default Constructor Called*" in start
- Now run your program and test what does date1 prints?

Exercise 3 [Print Function]:

- Implement a function **Print** in Date class which prints a date in following format:
  dd/mm/yyyy (e.g. 1/1/1926 for date1)
- Print object date1 in your main function and run the program.
- What does it print and how can we initialize the data of date1 at the time of creation?

Exercise 4 [Overloaded Constructor with Default Argument]:

- Write an overloaded Constructor of Date class that initializes the date object to date, month and year provided as parameter and prints "Overloaded Function Called"
- Now create another object ***independanceDay*** in main that is 14/08/2018
- Print ***independanceDay*** by calling Print function of Date class and run your program

Exercise 5 [Destructor]:

- Write Destructor of Date class that prints "Destructor called"
- Run your program and test it

Exercise 6 [Input Function]:

- Write a function **Input** in your Date class that takes input from user to populate a Date object
- Call "*date1.Input()*" and "*date1.Print()*" in your driver program and test it

Exercise 7 [Setters]:

- Create an object ***xmasDay*** using default constructor
- Print xmasDay and see what it prints
- Write Setters i.e. SetDay, SetMonth and SetYear in your class
- Now set xmasDay to 25/12/2018 using Setters in main

Exercise 8 [Getters]:

- Write Getters i.e. GetDay, GetMonth and GetYear in your date class

- Now print xmasDay using Getters in your Driver program

Exercise 9 [Passing object by value]:

- Write a function **int Compare(Date)** that compares two dates, returns 1 if left hand side object is greater than right hand side object, -1 if lhs is smaller and 0 otherwise
- Test your function

Exercise 10 [Return object by value]:

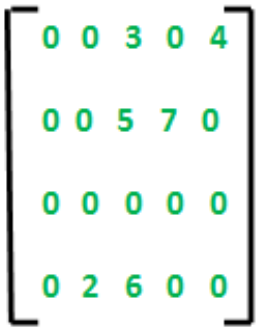
- Write a function **Date IncrementMonth()** that returns a newly created Date object with one month next to the current date object. For example, if date1 = 2/01/2016 *date1.IncrementDate()* will return 2/02/2016 without changing date1
- Print both the date1 and newly created date in your driver program to test the result

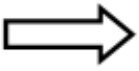
Question Number: 2

You are required to implement a C++ class for sparse matrices and overload operators to support common matrix operations, such as addition, subtraction, multiplication. Your implementation should focus on efficient storage and manipulation of large sparse matrices using CSR format.

Background:

Sparse matrices are matrices where most elements are zero. In many applications, such as finite element analysis, computational fluid dynamics, and graph theory, sparse matrices are prevalent due to their ability to efficiently represent large, sparsely populated data sets. To optimize storage and computational efficiency, sparse matrices can be stored in compressed formats such as Compressed Sparse Row (CSR) or Compressed Sparse Column (CSC) formats. In this assignment, you will implement operator overloading for sparse matrices using CSR formats.





Row	0	0	1	1	3	3
Column	2	4	2	3	1	2
Value	3	4	5	7	2	6

Class Skeleton:

class SparseMatrix {

```
private:
    int rows; // Number of rows in the matrix
    int cols; // Number of columns in the matrix
    int ** sparseMatrix; //Compressed Matrix

public:
    // Constructors
    SparseMatrix();
    SparseMatrix(const std::string& filename);
// Copy constructor
    SparseMatrix(const SparseMatrix& other);

    // Destructor
    ~SparseMatrix();

    // Assignment operator
    SparseMatrix& operator=(const SparseMatrix& other);

    // Overloaded operators
    SparseMatrix operator+(const SparseMatrix& other) const;
    SparseMatrix operator-(const SparseMatrix& other) const;
    SparseMatrix operator*(const SparseMatrix& other) const;

    bool operator==(const SparseMatrix& other) const;
    bool operator<=(const SparseMatrix& other) const;
    bool operator>=(const SparseMatrix& other) const;

// Overload [][] operator
    int* operator[](size_t index);
```

```
    // Const version for accessing elements
    const int* operator[](size_t index) const;

    // Other member functions
    void transpose();
    // Other member functions for basic matrix operations

    // Helper functions
    // Function prototypes for helper functions

    // Overloaded input and output operators
    friend std::istream& operator>>(std::istream& input, SparseMatrix& matrix);
    friend std::ostream& operator<<(std::ostream& output, const SparseMatrix& matrix);
};
```

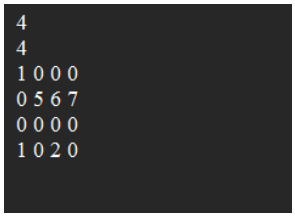
Note. You must do all the operations on the compressed form of the Matrix.

1) SparseMatrix();

Intailze the matrix to null and row and col to zero

2) SparseMatrix(const std::string& filename);

File format:



First two lines for the size of rows columns respectively and then the array. First load the Matrix in a Array then put its compressed form in the data member: sparseMatrix.

3) SparseMatrix(const SparseMatrix& other);

Copy constructor that creates a new sparse matrix object by copying the contents of another sparse matrix object (other).

4) ~SparseMatrix();

Destructor that is called when a sparse matrix object goes out of scope or is explicitly deleted. It deallocates any dynamically allocated memory associated with the object.

5) SparseMatrix& operator=(const SparseMatrix& other);

Assignment operator overload that assigns the contents of one sparse matrix (other) to another sparse matrix object. Returns a reference to the assigned object.

6) SparseMatrix operator+(const SparseMatrix& other) const;

Overload of the addition operator (**+**) to perform element-wise addition of two sparse matrices.

Note. That this overload does not change the value of the object that called the function. It only returns a new Sparse Matrix after the operation.

Example:

Sparse Matrix A

Row	1	1	3	4	4
Column	2	4	3	1	2
Value	10	12	5	15	12

Sparse Matrix B

Row	1	2	3	4	4
Column	3	4	3	1	2
Value	8	23	9	20	25

Resultant Sparse Matrix:

Row	1	1	1	2	3	4	4
Column	2	3	4	4	3	1	2
Value	10	8	12	23	14	35	37

7) SparseMatrix operator-(const SparseMatrix& other) const;

Overload of the addition operator (**-**) to perform element-wise addition of two sparse matrices.

Note. That this overload does not change the value of the object that called the function. It only returns a new Sparse Matrix after the operation.

8) SparseMatrix operator*(const SparseMatrix& other) const;

Overload of the addition operator (*) to perform element-wise addition of two sparse matrices.

Note. That this overload does not change the value of the object that called the function. It only returns a new Sparse Matrix after the operation.

Example:

Sparse Matrix A

Row	1	1	3	4	4
Column	2	4	3	1	2
Value	10	12	5	15	12

Sparse Matrix B

Row	1	2	3	4	4
Column	3	4	3	1	2
Value	8	23	9	20	25

Resultant Sparse Matrix:

Row	1	1	1	3	4	4
Column	1	2	4	3	3	4
Value	240	300	230	45	120	276

9) bool operator==(const SparseMatrix& other) const:

Overload of the equality operator (**==**) to compare two sparse matrices for equality.

10) bool operator<(const SparseMatrix& other) const:

Overload of the less than or equal to operator (**<=**) to compare two sparse matrices. One sparse matrix is less than the other if the sum of all its values is less than the other.

11) bool operator>(const SparseMatrix& other) const:

Overload of the greater than or equal to operator (**>=**) to compare two sparse matrices. One sparse matrix is greater than the other if the sum of all its values is greater than the other.

12) void transpose():

A member function that transposes the sparse matrix, exchanging rows and columns.

Example:

Sparse Matrix A

Row	1	1	3	4	4
Column	2	4	3	1	2

Value	10	12	5	15	12

Transpose of Sparse Matrix A

Row	1	2	2	3	4
Column	4	1	4	3	1
Value	15	10	12	5	12

13) friend std::istream& operator>>(std::istream& input, SparseMatrix& matrix):

Overload of the input operator (**>>**) to allow the user to input data into a sparse matrix object from a stream. First take the number of rows and columns of the original matrix then input the data of the original matrix then put its compressed form in the data member: sparseMatrix.

14) friend std::ostream& operator<<(std::ostream& output, const SparseMatrix& matrix):

Overload of the output operator (**<<**) to allow the user to output the contents of a sparse matrix in the following format:

Row	0	0	1	1	3	3
Column	2	4	2	3	1	2
Value	3	4	5	7	2	6

15) int* operator[](int index) and const int* operator[](int index) const;

Overload the subscript operator such as by using sparsematrix[0][0] we get the element at 0 0 position of the compressed matrix. Hint: see the prototypes.