Objective:

- Understanding and implementing weak/strong Aggregation and dealing dynamic memory allocation for objects.
- Reusability through composition/aggregation.
- Extension/Modification in ADT without changing the existing interface or code.
- Focusing on Array of objects.

Task-1: *Manipulating Square Matrices*

As you know, that we implement Matrix class few weeks before in a quiz/lab. Suppose that the user wants to manipulate square matrices only and for this purpose he want to mention matrix order in single variable like N-Order matrix instead of separately giving rows and columns. For this purpose, what we can do? Well, considering the knowledge of OOP that we have explored so far, we can write a wrapper class (lets name it "SMatrix") and reuse the methods defined in Matrix class.
While implementing 'SMatrix', the idea is to give all the features of matrices without changing a bit in 'Matrix' class but reusing the functions defined in Matrix class instead of reinventing the wheel.
The basics has also been discussed in class/lecture, so I hope that you will be able to accomplish this solution.

```
class Matrix
{
    int rows,cols;
    double ** data;
public:
    Matrix();
    Matrix(int,int);
    Matrix(const Matrix &);
    ~Matrix();
    double & at( int, int);
    const double & at( int, int) const;
    int getRows() const;
    int getColums() const;
    void display() const;
    Matrix Transpose() const;
    Matrix add(const Matrix &) const;
    Matrix multiply(const Matrix &) const;
    Matrix reSize(int);
    //many other functions
};
```

Task-2: *Scheduler*

We need to design an application like the one you normally use in your mobiles about keeping record of tasks to be done for any particular date and time.
For this purpose, we need following classes:

Class Task: will be responsible for recording the message/task to be done in particular date and time. So, for this purpose the following members are needed for class Task:

```
class Task
{
    Date taskDate;
    Time taskTime;
    CString taskMsg;
};
```

You should be quite familiar with the class 'Date', class 'Time', and class 'CString'. So, we can say that a Task is composed of Date, Time, and CString objects (strong-aggregation/composition).

Class Scheduler: But as you know that a scheduler will record/save many tasks (more than one Task objects) in it. So we need another class, which will be responsible for keeping record of all tasks recorded/saved by user. Class 'Scheduler' will be responsible for this purpose, which is as follows:

```
class Scheduler
{
    Task * taskList;
    int noOfTasks;
    int capacity;
};
```

- taskList: will point to an array of objects of type Task.
- noOfTasks: keep record of the number of tasks saved by user in Scheduler object.
- capacity: size of the array pointed by 'taskList'

Class SchedulerApp: all the interface related things will come in it.

Class CString: Same old CString ☺.

Class Date and Time: Nothing new for you guys, but I have also provided few functions implementation of Date class.

You should decide yourself about placing copy constructor, and destructor in any class given below.

Also implement the function given in class 'Scheduler' and class 'SchedulerApp'

```
class CString
{
// same as given in previous practice file
};
class Time
{
// same as given in previous practice file
};
class Date
{
    static const int daysInMonth[ 13 ];
    int day;
    int month;
    int year;
    bool isLeapYear(int y) const;
    bool isValidDate(int d, int m, int y) const;

public:
    Date():day(14), month(10), year(2019)
    { }
    Date( int d, int m, int y):day(14), month(10), year(2019)
    {
        setDate(d,m,y);
    }
    void setDate( int d, int m, int y);
    void setDay(int d);
    void setMonth(int m);
    void setYear(int y);
    int getDay() const;
    int getMonth() const;
    int getYear() const;
    void printFormat1()const
    {
        cout.fill('0');
        cout<<setw(2)<<day<<"/"<<setw(2)<<month<<"/"<<setw(4)<<year;
    }
    void printFormat2()const;
    void printFormat3()const;
    void incDay(int=1);
    void incMonth(int=1);
    void incYear(int=1);
    CString getDateInFormat1() const
    {
        CString date;
        date.reSize(11);//OR CString date("00/00/0000");
        date.concatEqual(to_string(day).c_str());
        //you can't use  to_string function ☺
        date.concatEqual("/");
        date.concatEqual(to_string(month).c_str());
        date.concatEqual("/");
        date.concatEqual(to_string(year).c_str());
        return date;
    }
    CString getDateInFormat2()const;
    CString getDateInFormat3()const;
};

const int Date::daysInMonth[ 13 ] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

```
class Task
{
    Date taskDate;
    Time taskTime;
    CString taskMsg;
public:
    Task()
    {}
    Task(const Date & d, const Time & t, const CString & m):
taskDate(d),taskTime(t),taskMsg(m)
    {
    }
    void setTask(const Date & d, const Time & t, const CString & m)
    {
        taskDate = d;//no issue with shallow copy
        taskTime = t;//no issue with shallow copy
        taskMsg = m; // shallow copy will create issues
    }
    void updateDate(const Date & nd)
    {
        taskDate = nd;
    }
    void updateTime(const Time & nt)
    {
        taskTime = nt;
    }
    void updateMessage(const CString & m)
    {
        taskMsg = m;
    }
    Date getDate()
    { return taskDate; }
    Time getTime()
    { return taskTime; }
    CString getMessage()
    { return taskMsg; }
};
class Schedular
{
    Task * taskList;
    int noOfTasks;
    int capacity;
public:
    Schedular()
    {
        capacity=5;
        noOfTasks=0;
        taskList = new Task[capacity];
    }
    void addTask(const Task & t);
    void displayTask(const Date & d=Date(0,0,0))
    {
        //show todays tasks by default or according to the date received
    }
    void displayTodaysTasks();
    void reSize( int ); //resizes the array whenever it gets full.
};
class SchedulerApp
{
public:
    static void startApp()
    {
        //all the interface related things will come here
    }
};
```

Note:

- I have explicitly left couple of bugs in few places, which you got to identify/rectify yourself. And this statement may itself be a cattywampus ☺.
- To save space I have given definitions within class body. You got to follow all the coding conventions that we have discussed so far.