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: *Set using Array*

You must be quite familiar with the class 'Array' implemented below. The other class i.e. 'Set' is also not new to you. Although the purpose of class 'Set' is same like the previous version but its data members are changed and 'Set' is implemented using class 'Array' object. I have defined a few function in Set class, you are required to implement rest of the function given in Set class. Remember: You must not change anything private/public in class Array. Wisely decide about the fate of destructor and copy constructor of class 'Set'

```
class Array
{
    int *data;
    int  capacity;
    int isValidIndex( int index ) const
    {
        return index>=0 && index<capacity;
    }
public:
    ~Array()
    {
        if (data)
            delete [] data;
        data=0;
        capacity=0;
    }
    int & getSet(int index)
    {
        if (isValidIndex(index))
            return data[index];
        exit(0);
    }
    const int & getSet(int index) const
    {
        if (isValidIndex(index))
            return data[index];
        exit(0);
    }
    int getCapacity() const
    {
        return capacity;
    }
    void reSize ( int newCap )
    {
        if (newCap<=0)
        {
            this->~Array();
            return;
        }
        int * ptr = new int[newCap];
        memcpy(ptr, data,
(newCap<capacity?newCap:capacity)*sizeof(int));

        this->~Array();

        capacity =
newCap<capacity?newCap:capacity;
        data=ptr;
    }
    Array ( const Array & ref)
    {
        if (ref.data==0)
        {
            data=0;
            capacity=0;
            return;
        }
        capacity=ref.capacity;
        data = new int[capacity];
        memcpy(data, ref.data,
capacity*sizeof(int));
    }
Array(int argCount=5, ...)
    {
        if (argCount<=0)
        {
            capacity=0;
            data=0;
            return;
        }
        capacity = argCount;
        data = new int[capacity];
        va_list vl;
        va_start(vl, argCount);
        for ( int i=0; i<capacity; i++)
            data[i] = va_arg(vl, int);
    }
};
class Set
{
public:
    Array data;
    int noOfElements;
public:
    Set( int cap = 5 ):data(cap)
    {
        noOfElements=0;
    }
    void insert (int element);
    void remove (int element);
    void print() const;
    int getCardinality() const;
    int isMember ( int val ) const;
    int isSubSet ( const Set & s2 ) const;
    void reSize ( int newcapacity );
    Set( const Set & ref);
    Set calcUnion ( const Set & s2 ) const;
    Set calcIntersection ( const Set & s2 ) const;
    Set calcDifference ( const Set & s2 ) const;
    Set calcSymmetricDifference ( const Set & s2 )
const;
};
```

Task-2: *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.

This basic has also been discussed in class/lecture, so I hope that you will be able to accomplish this solution.

```
class Matrix
{
    int rows,cols;
    int ** data;
public:
    Matrix();
    Matrix(int,int);
    Matrix(const Matrix &);
    ~Matrix();
    int & at( int, int);
    const int & 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, int);
    //many other functions
};
```

```
class SMatrix
{
    Matrix mat;
public:
    SMatrix()
    {}
    SMatrix(int n):mat(n,n)
    {}
    int & at( int r, int c)
    {
        return mat.at(r,c);
    }
    int getMatrixOrder()
    {
        return mat.getRows();//OR return
mat.getCols();
    }
};
```

Task-3: *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 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: Some basic functions are given but should add other functions as well that we added in the standard CString class with const version.

Class Date and Time: Nothing new for you guys, but I have also given their basic implementation as well.

You must 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
{
public:
    char * data;
    int size;

    int getLength(const char * s) const
    {
        int i=0;
        while(s[i++]!='\0');
        return i-1;
    }
public:
    int getLength() const
    {
        return getLength(data);
    }
    CString()
    {
        data=0;
        size=0;
    }
    CString ( const char * s)
    {
        size = getLength(s)+1;
        data = new char[size];
        memcpy(data, s, size);
    }
    CString(const CString & ref)
    {
        size = ref.size;
        data = new char[size];
        memcpy(data, ref.data, size);
    }
```

```
    ~CString()
    {
        if (data)
        {
            delete[] data;
            data=0;
            size=0;
        }
    }
    void display() const
    {
        if (data)
            cout<<data;
    }
    char & at(int i)
    {
        if (i>=0 && i<size)
            return data[i];
        exit(0);
    }
    const char & at(int i) const
    {
        if (i>=0 && i<size)
            return data[i];
        exit(0);
    }
    void reSize(int ns)
    {
        if (ns<=0)
        {
            this->~CString();
            return;
        }
        char * temp = new char[ns];
        int i=0;
        while(i<ns && i<size)
        {
            temp[i] = data[i];
            i++;
        }
        this->~CString();
        size=ns;
        data=temp;
    }
    void concatEqual (const CString & ref )
    {
        int len1 = getLength();
        int len2 = ref.getLength();
        char * temp = new char[len1+len2+1];
        int i=0;
        memcpy(temp, data, len1+1);
        memcpy(&temp[len1], ref.data, len2+1);
        this->~CString();
        data = temp;
        size = len1+len2+1;
    }
};
class Date
{
    static const int daysInMonth[ 13 ];
    int day;
    int month;
    int year;
    bool isLeapYear(int y)
    {
        return (y%4==0 && y%100!=0) || y%400==0;
    }
    bool isValidDate(int d, int m, int y)
    {
        if (!(y>=1900 && y<=2100))
            return false;
        if (!(m>=1 && m<=12))
            return false;
        if (!( (d>=1 && d<=daysInMonth[m]) ||
            (m==2 && d<=29 &&isLeapYear(y))))
            return false;
        return true;
    }

public:
    Date():Date(1,12,2015)
    {
    }
    Date( int d, int m, int y)
    {
        if (isValidDate(d,m, y))
        {
            day=d;
            month=m;
            year=y;
        }
        else
        {
            day=1;
            month=12;
            year=2015;
        }
    }
    void setDate( int d, int m, int y)
    {
        if (isValidDate(d, m, y))
        {
            day=d;
            month=m;
            year=y;
        }
    }
    void setDay(int d)
    {
        if (isValidDate(d,month, year))
        {
            day=d;
        }
    }
    void setMonth(int m)
    {
        if (isValidDate(day,m, year))
        {
            month=m;
        }
    }
    void setYear(int y)
    {
        if (isValidDate(day,month, y))
        {
            year=y;
        }
    }
    int getDay() const
    {
        return day;
    }
    int getMonth() const
    {
        return month;
    }
    int getYear() const
    {
        return year;
    }
    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());
        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 Time
{
private:
    int hour;
    int minutes;
    int seconds;
public:
    Time():Time(0,0,0)
    {}
    Time ( int h, int m, int s )
    {
        hour = (h>=0 && h<=23)?h:0;
        minutes = (m>=0 && m<=59)?m:0;
        seconds = (s>=0 && s<=59)?s:0;
    }
    void setHours(int h)
    {
        hour = (h>=0 && h<=23)?h:hour;
    }
    void setMinutes( int m)
    {
        minutes = (m>=0 && m<=59)?m:minutes;
    }
    void setSeconds( int s )
    {
        seconds = (s>=0 && s<=59)?s:seconds;
    }
    int getHour() const
    {
        return hour;
    }
    int getMinutes() const
    {
        return minutes;
    }
    int getSeconds() const
    {
        return seconds;
    }
};
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 removeTask(const Date & d=Date(0,0,0))
    {
        //show todays tasks by default or
according to the date received
    }
    void displayTodaysTasks()
    {
    }
};
class SchedulerApp
{
public:
    static void startApp()
    {
        //all the interface related things will
come here
    }

};
```