Objective:

- It will help you understand the idiom "Use const wherever possible" and "Principle of least privilege".
- Constant members and constant objects.

Task-1:

Implement your own String class. It should not be difficult for you, as you have implemented most of the functions as global functions in your PF class. The idiom "Use const wherever possible" is also applied on the member functions.

```
class CString
{
    char * data;
    int size;
public:
```

`CString ();`	
`CString (const char c);`	
`CString(const char * const);`	
`~CString ();`	
`void input();`	Takes input from console in calling object.
`char & at( const int index);`	*Index:* Receives the index for string. *Return Value:* reference of array location represented by index
`const char & at( const int index) const;`	
`int isEmpty( ) const;`	Tells whether string is empty or not *Return Value*: return 0 if string empty otherwise nonzero value.
`int getLenght()const;`	
`void display()const;`	
`int find( const char * const substr, const int start=0 )const;`	
`int find( const char ch, const int start=0 ) const;`	
`int insert( const int index, const char * const substr);`	
`int insert( const int index, const char ch);`	
`int remove( const int index, const int count=1);`	
`int remove ( const char ch );`	
`void replace( const char newC );`	
`int replace( const char old, const char newC );`	
`int replace( const char * const  old, const char * const newC );`	
`void trim();`	
`void trimLeft();`	
`void trimRight();`	
`void makeUpper();`	
`void makeLower();`	
`void reverse();`	
`void reSize(const int add);`	
`void concatEqual( const CString & s2 );`	
`void concatEqual( const char * const s2 );`	
`int isEqual(const CString & s2 )const  ;`	
`int isEqual(const char * const s2 )  ;`	
`CString ( const CString & );`	

`CString left( int count ) ;`	*Count:* The number of characters to extract from calling object from left side *Return Value:* A CString object that contains a copy of the specified range of characters
`CString right( int count ) ;`	
`CString concat( const CString& s2 ) const ;`	
`CString concat( const char * const s2 ) const;`	
`CString tokenzie( const char * const delim ) ;`	Returns a Cstring object which contains the substring by extracting it from the calling object CString string depending upon the delimiter characters passed. See the following Sample Run to further understand the functionality:

```
void main()
{
      CString str(" This, --a sample string. nothing");
      CString token = str.tokeniz (“,.-”);

      // After the execution of above statement:
      // the token contains “ This”
      // the contents of ‘str’ becomes “ --a sample string. nothing”

      // So if we call it again

      token = str.tokenize (“,.-”);

      // then ‘token’ contains “ ”
      // ‘str’ contents becomes “-a sample string. nothing”

      //and calling again

      token = str.tokenize (“,.-”);

      // then token contains “”
      // str contents becomes “a sample string. nothing”

      //and calling again

      token = str.tokenize (“,.-”);

      // then token conatins “a sample string”
      // str contents becomes “ nothing”
}
```

};

Task-2: Something to learn different

In addition to applying "use const wherever possible" idiom, I have also added a quite different kind of constructor in the Array class, which uses variable number of arguments feature. You are directed to Google (one of the link is: http://www.cprogramming.com/tutorial/c/lesson17.html) this feature and then explore the following code.

```
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));
    }
    //Google variable number of arguments
    //Then come and discuss with me
    //you may explore the following link for this purpose
    //http://www.cprogramming.com/tutorial/c/lesson17.html

    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);
    }
};
void display(const Array & ref)
{
    for ( int i=0; i<ref.getCapacity(); i++)
        cout<<ref.getSet(i)<<" ";
    cout<<endl;
}
int main()
{
    Array a(3,1,2,4);//first argument is array size : rest are values
    a.getSet(1)=90;
    display(a);
    Array b;
    Array c(10);
    const Array d(5,10,20,30,40,50);
    //d.getSet(1)=110;//not allowed as d is constant
    display(d);
}
```