Objective:

- Issue related to Object transition by value.
- Copy Constructor
- Something to learn different (Variable number of arguments) ☺

Task-1:

Add/Update/Modify the following in your CString class.
Note: You are not allowed to use any library functions related to strings.

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

`CString ( const CString & );`	
`int find(CString subStr, int start=0 ) ;`	Find the substring in the calling CString object. By default, search starts from 0 index. Return the count of occurrences found in calling object.
`void insert(int index, CString subStr);`	Insert the substring at given index in calling object.
`int replace(CString old, CString newSubStr );`	Find all the occurrences of old substring and replace it with new substring. Return the count of occurrences found in calling object.
`int compare(CString s2 );`	Compare the calling and receive object string. It should behave just like strcmp
`void shrink();`	Resize/shrink the array equal to the length of string pointed by data.

```
};
```

It's not the final version, we shall keep updating this library over the next couple of weeks.☺

Task-2: Something to learn different

I have added a quite different kind of constructor in the Array class, which uses variable number of arguments feature. You are directed to Google (http://www.cprogramming.com/tutorial/c/lesson17.html) this feature and then explore the code given below.

```
class Array
{
    int * data;
    int  capacity;
    int isValidIndex( int index ) const
    {
        return index>=0 && index<capacity;
    }
public:
    ~Array()
    {
        if (data)
            delete [] data;
        data=nullptr;
        capacity=0;
    }
    int & getSet(int index)
    {
        if (isValidIndex(index))
```

```
    //Variable Number of Arguments
    Array(int argCount=0, ...)
    {
        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);
va_end ( vl );
    }
};
void display(const Array & ref)
```

```
            return data[index];
        exit(0);
    }
    int getCapacity()
    {
        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;
        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));
    }
```

```
{
    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);
    return 0;
}
```