```
            capacity=0;
            return;
        }
        capacity=ref.capacity;
        data = new int[capacity];
        memcpy(data, ref.data, capacity*sizeof(int));
    }
    //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)
{
    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;
}
```

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;
        data = ptr;
    }
    Array ( const Array & ref)
    {
        if (ref.data==0)
        {
            data=0;
```

	string
void trim();	Removes all the white space characters on both left and right sides of string
void makeUpper();	Change all the alphabets to uppercase
void makeLower();	Change all the alphabets to lowercase
void reverse();	It reverses the string stored in the calling object
void reSize(int);	You know what to do.
int compare(CString s2) const;	Compare the calling and receive object string and behave just like strcmp
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) ;	
int toInteger() const;	
float toFloat() const;	
CString concat(CString s2) const ;	It returns the concatenated result of received and calling object without changing calling object.
void concatEqual(CString s2);	It concatenates the received object string with calling object.
CString tokenzie(CString delim) ;	Returns a CString object which contains the substring by extracting it from the calling object CString depending upon the delimiter characters passed. See the following Sample Run to further understand the functionality:

```
int 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"
      token = str.tokenize (",.-");

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

      token = str.tokenize (",.-");

      // then token conatins "a sample string"
      // str contents becomes " nothing"
      return 0;
}
};
```

Objective:

- Issue related to Object transition by value.
- It will help you understand the idiom "Use const wherever possible" and "Principle of least privilege".
- Constant members/objects and something to learn different (Variable number of arguments) ☺

Task-1:

An updated version of *CString*, which will provide basic functionalities related to strings.
Note: You are not allowed to use any library functions related to strings.

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

`CString ();`	Initializes data and size to 0.
`CString (const char c);`	Initializes data with char c
`CString(const char *);`	Initializes the data with received string by allocating memory on heap.
`CString ( const CString & );`	
`~CString ();`	You know what to do.
`void input();`	Takes input from console in calling object.
`void shrink();`	Resize/shrink the array equal to the length of string pointed by data.
`char & at(int index);`	*Index:* Receives the index for string. *Return Value:* reference of array location represented by index
`const char & at( const int index) const;`	
`bool isEmpty( ) const;`	Tells whether string is empty or not *Return Value*: return true if string empty otherwise true.
`int getLength()const;`	Returns length of the string
`void display()const;`	Prints the string on console
`int find(CString subStr, int start=0 ) const;`	Find the first occurrence of substring in the calling CString object. By default, search starts from 0 index. If found then return the starting position of subStr found otherwise return -1.
`void insert(int index, CString subStr);`	Insert the substring at given index in calling object.
`void remove(int index, int count=1);`	Remove the characters (how many? Given in count) starting from index
`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.
`void trimLeft();`	Removes all the white space characters on the left of string
`void trimRight();`	Removes all the white space characters on the right of