## Objective:

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

## 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.<br>*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<br>*Return Value*: return true if string empty otherwise false. |
| `int getLength() const;` | Returns length of the string |
| `void display() const;` | Prints the string on console |
| `int find(const 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, const 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(const CString & old, const 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 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(const 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<br>*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(const CString & s2 ) const  ;` | It returns the concatenated result of received and calling object without changing calling object. |
| `void concatEqual(const CString & s2 );` | It concatenates the received object string with calling object. |
| `CString tokenzie(const CString & delim )  ;` | Returns a CString object which contains the substring by extracting it from the calling object CString depending upon the delimiter characters passed.<br>See the following Sample Run to further understand the functionality: |

```
int main()
{
    CString str(" This, --a sample string.
nothing");
    CString token;
    cout<<"String = ";str.display();cout<<"\n";

    while(!str==false)
    {
        token = str.tokenize(".-,");
        cout<<"Token =
";token.display();cout<<"\n";
    }
    cout<<endl;
    return 0;
}
```

**Console Output**

```
String =  This, --a sample string. nothing
Token =  This
Token =
Token =
Token = a sample string
Token =  nothing
```

**Note:**
The code given in main function will produce runtime error because of shallow copying of CString objects. i.e. token = str.tokenize (",.-");
The learning that we have done so far, we are able to copy such objects only at the time of declaration. We haven't been able to find its solution yet and still avoid such syntax with our objects. So, you got to think some other way/syntax of assigning CString objects until we rectify this issue on our CString class.

};