C++ Programming – Lecture 13

Templates

- Templates promote source-code level reuse, whereas Inheritance and Containership promote object-code level reuse
- It is possible to create template functions as well as template classes
- Once the template function / class is ready we can use them with any standard or user-defined type
- Syntax of defining and calling a template function :

```
// template function definition
template < class T >     // or template < typename T >
void printArray ( T[ ] arr )
{
    ..
}

// call to template function
int intarr[ ] = { 10, -2, 37, 42, 15 } ;
printArray ( intarr ) ;
```

- Template function can receive multiple types as in

```
template < class T, class S, class Z >
void printTypes ( T a, S b, Z c )
{
    ..
}
```

- Syntax for using and defining a template class :

```
// using template class
stack  < int >  s1 ;
s1.push ( 10 ) ;

// defining template class
template < class  T >
class Stack
{
    ..
}
```

Standard Template Library

- To store, retrieve and manipulate multiple numbers / strings arrays can be used
- Arrays suffer from 2 limitations :
    - They have no mechanism to maintain data in different ways like key -value pairs, ordered sets, etc.
    - Arrays have no means to access data in FIFO, LIFO, Sorted order, etc.
- Instead of arrays we should use ready-made library called Standard Template Library (STL)
- Advantages of using STL
    - Very efficient, time tested, written by experts
    - Readymade classes for most data structures, so we can concentrate on program rather than building data structures
    - It is possible to extend the classes to suit our needs
    - Three key components of STL :

      1) Containers - Store data
      2) Iterators - Traverse container elements
      3) Algorithms - Perform multiple opns on container elements

    - Container types :

      Sequence - vector, deque, list
      Associative - set, multiset, map, multimap
      Container Adapters - stack, queue, priority_queue
      Others - bitsets, valarrays

    - Iterator types :

      Input iterator
      Output iterator
      Forward iterator
      Bidirectional iterator
      Random access iterator

    - Different containers support different types of iterators
    - Const iterators can modify elements of a container, non-const iterators cannot
    - Algorithms - Template functions that perform common operations like insertion, deletion, searching, sorting and comparing elements or entire containers.