Hope you already understand the concept of C++ Template which we already have discussed in one of the chapters. The C++ STL StandardTemplateLibrary is a powerful set of C++ template classes to provides general-purpose templatized classes and functions that implement many popular and commonly used algorithms and data structures like vectors, lists, queues, and stacks.

At the core of the C++ Standard Template Library are following three well-structured components:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Containers</td>
<td>Containers are used to manage collections of objects of a certain kind. There are several different types of containers like deque, list, vector, map etc.</td>
</tr>
<tr>
<td>Algorithms</td>
<td>Algorithms act on containers. They provide the means by which you will perform initialization, sorting, searching, and transforming of the contents of containers.</td>
</tr>
<tr>
<td>Iterators</td>
<td>Iterators are used to step through the elements of collections of objects. These collections may be containers or subsets of containers.</td>
</tr>
</tbody>
</table>

We will discuss about all the three C++ STL components in next chapter while discussing C++ Standard Library. For now, keep in mind that all the three components have a rich set of pre-defined functions which help us in doing complicated tasks in very easy fashion.

Let us take the following program demonstrates the vector container aC++ StandardTemplate which is similar to an array with an exception that it automatically handles its own storage requirements in case it grows:

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    // create a vector to store int
    vector<int> vec;
    int i;

    // display the original size of vec
    cout << "vector size = " << vec.size() << endl;

    // push 5 values into the vector
    for(i = 0; i < 5; i++){
        vec.push_back(i);
    }

    // display extended size of vec
    cout << "extended vector size = " << vec.size() << endl;

    // access 5 values from the vector
    for(i = 0; i < 5; i++){
        cout << "value of vec [" << i << "] = " << vec[i] << endl;
    }

    // use iterator to access the values
    vector<int>::iterator v = vec.begin();
    while( v != vec.end() ) {
        cout << "value of v = " << *v << endl;
        v++;
    }
}````
When the above code is compiled and executed, it produces the following result:

```cpp
} return 0;
```

Here are following points to be noted related to various functions we used in the above example:

- The `push_back` member function inserts value at the end of the vector, expanding its size as needed.
- The `size` function displays the size of the vector.
- The function `begin` returns an iterator to the start of the vector.
- The function `end` returns an iterator to the end of the vector.