Thư viện Template chuẩn (STL) trong C++

Hy vọng bạn đã hiểu khái niệm về Template trong C++ đã được chúng tôi trình bày trong chương về Template. Standard Tamplate Library (STL) trong C++ là một tập hợp các lớp Template mạnh mẽ trong C++ để cung cấp các lớp và các hàm được tạo theo khuân mẫu cho mục đích lập trình tổng quát, mà triển khai nhiều thuật toán và cấu trúc dữ liệu được sử dụng phổ biến và thông dụng như vector, list, queue và stack.

Có ba thành phần mang tính cấu trúc mạnh mẽ của Standard Tamplate Library (STL) trong C++ là:

Thành phần	Miêu tả
Containers	Containers được sử dụng để quản lý các tập hợp đối tượng của một kiểu cụ thể. Có một số kiểu Containers khác nhau như list, vector, map, …
Algorithms	Algorithms hoạt động trên containers. Chúng cung cấp các phương thức mà theo đó bạn sẽ thực hiện việc khởi tạo, sắp xếp, tìm kiếm nội dung của Containers
Iterators	Iterators được sử dụng để duyệt qua các phần tử trong tập hợp các đối tượng. Những tập hợp này có thể là Containers hoặc Subset của Containers

Chúng ta sẽ đề cập về 3 thành phần của STL trong C++ trong chương tiếp theo khi bàn luận về Thư viện chuẩn C++ (C++ Standard Library). Bây giờ, bạn ghi nhớ rằng, 3 thành phần này có một tập hợp các hàm được định nghĩa trước, mà giúp chúng ta trong việc thực hiện các tác vụ phức tạp trở nên đơn giản hơn.

Xét ví dụ sau minh họa vector containers (là một Template chuẩn trong C++), mà giống như một mảng với một exception mà tự động xử lý các nhu cầu về storage của riêng nó khi nó cần:

```
#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++;
   }

   return 0;
}
```

Khi code trên được biên dịch và thực thi, nó cho kết quả sau:

```
vector size = 0
extended vector size = 5
value of vec [0] = 0
value of vec [1] = 1
value of vec [2] = 2
value of vec [3] = 3
value of vec [4] = 4
value of v = 0
value of v = 1
value of v = 2
value of v = 3
value of v = 4
```

Dưới đây là các điểm quan trọng cần ghi nhớ liên quan tới các hàm đa dạng đã được sử dụng trong ví dụ trên:

- Hàm thành viên push_back() chèn giá trị tại phần cuối của vector, mở rộng kích cỡ của nó khi cần.
- Hàm size() hiển thị kích cỡ của vector.
- Hàm begin() trả về một iterator tới phần đầu của vector.
- Hàm end() trả về một interator tới phần cuối của vecor.