Tutorial Link https://course.testpad.chitkarauniversity.edu.in/tutorials/C++ Standard Template Library (STL): Usefuls/5b5d3d10aaf783423c812354

TUTORIAL

C++ Standard Template Library (STL): Usefuls

Topics

Algorithm

C++ provides a set of generic algorithm for STD in header `<algorithm>`, which includes:

- Searching: `find()`, `count()`.
- Sorting: `sort()`, `partial_sort()`, `merge()`.
- Generation, mutation and deletion:
- Numeric and relational:

The algorithms operate on elements of STL container only indirectly through the iterator.

The generic algorithms are non-member functions that are applicable to all STL containers. It accepts a pair of iterators, denoted as `first` and `last`, that mark the range of operation as `[first,last)` (including first, but excluding last). For example,

```
sort(aVector.begin(), aVector.end());  // Sort the entire vector
sort(aVector.begin(), aVector.begin + aVector.size()/2);  // Sort
```

```
first half
```

Let's begin with some examples.

C++

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

// Use iterator to print the entire vector
void print(vector<int> & v) {
   for (vector<int>::iterator iter = v.begin(); iter != v.end(); ++iter) {
      cout << *iter << " ";   // dereference
   }
   cout << endl;
}

int main() {
   const int SIZE = 10;
   int array[SIZE] = {11, 55, 44, 33, 88, 99, 11, 22, 66, 77};
   vector<int> v(array, array + SIZE);
   print(v);

   // Sort
   sort(v.begin(), v.end());  // entire container [begin(),end())
   print(v);

   // Reverse
   reverse(v.begin(), v.begin() + v.size()/2);  // First half
   print(v);

   // Random Shuffle
   random_shuffle(v.begin() + 1, v.end() - 1);  // exclude first and last elements
   print(v);
```

```
31
32    // Search
33    int searchFor = 55;
34    vector<int>::iterator found = find(v.begin(), v.end(),
      searchFor);
35    if (found != v.end()) {
36       cout << "Found" << endl;
37    }
38    return 0;
39 }
```

- Most of the algorithm functions takes at least two iterators: `first` and `last`, to specify the range `[first,last)` of operation. They could have additional parameters.
- All STL containers provides members functions `begin()` and `end()`, which return the begin and pass-the-end elements of the container, respectively.
- To apply sort, the elements shall have overloaded the `'<'` operator, which is used for comparing the order of the elements.

for_each()

The `for_each()` applies a function to each element of the given range.

```
template <class InputIterator, class Function>
Function for-each (InputIterator first, InputIterator last,
Function f);
```

C++

```
1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4 using namespace std;
5
6 void square(int & n) { n *= n; }
7 void print(int & n) { cout << n << " "; }
8
```

```
int main() {
  vector<int> v;
  v.push_back(11);
  v.push_back(3);
  v.push_back(4);
  v.push_back(22);

  // Invoke the given function (print, square)
  // for each element in the range
  for_each(v.begin(), v.end(), print);
  for_each(v.begin() + 1, v.begin() + 3, square);
  for_each(v.begin(), v.end(), print);
  return 0;
}
```

STL and the string class

The `string` class is not part of the STL, but has implemented many STL features. `string` can be treated as a STL container of `char`. It defines member functions `begin()`, `end()`, `rbegin()`, `rend()` which returns an iterator for forward and reverse transversal. Most of the algorithms (such as `transform()`, `sort()`) are applicable to `string`, operating on individual characters.

C++

```
#include <iostream>
#include <string>
#include <algorithm>
#include <iterator>
#include <cctype>
using namespace std;

int main() {
  string s1("codequotient");
  cout << s1 << endl;   // "codequotient"
  string s2("codequotient");
  sort(s2.begin(), s2.end());
```

```
  cout << s2 << endl;   // "cdeeinooqttu"

  transform(s1.begin(), s1.end(), s1.begin(), ::toupper);
  cout << s1 << endl;   // "CODEQUOTIENT"
  transform(s1.begin(), s1.end(), s1.begin(),
bind1st(plus<char>(), 'a'-'A'));
  cout << s1 << endl;   // "codequotient"
  return 0;
}
```

vector, valarray and array

C++ has 3 array template classes: `vector`, `valarray`, `array` (C++11). `vector` and `array` are STL; while `valarray` is not.

vector

`vector` is certainly the most commonly used STL container. `vector` is dynamically allocated, with support for `push_back()` and `insert()`.

array

The `array` class is a wrapper for the fixed-sized built-in array with the STL container interfaces. It is designed as a substitute for the built-in array type, for applications where dynamic resizable `vector` is not needed (so as to reduce the overhead of dynamic array). `array` does not support `push_back()` and `insert()`, as it cannot be resized.

valarray

`valarray` is designed for numeric computation. It is variable-size but does not supports STL operations such as `push_back()` or `insert`, but provides a simple interface for many mathematical operations. For example, the arithmetic operators (such as +, -, *, /, %) and mathematical functions (such as `pow`, `sqrt`, `exp`, `log`, `sin`, `cos`, etc.) are overloaded to operate on the entire valarray (instead of individual element). For example,

C++

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

void print(const valarray<int> & va)
{
  for (int i = 0; i < va.size(); ++i)
  {
    cout << va[i] << " ";
  }
  cout << endl;
}

int main() {
  const int SIZE = 5;
  int a1[SIZE] = {1, 6, 5, 4, 3};
  int a2[SIZE] = {18, 10, 11, 14, 25};
  valarray<int> va1(a1, SIZE);
  valarray<int> va2(a2, SIZE);
  valarray<int> va3(SIZE);  // all 0
  print(va1);
  print(va2);
  print(va3);

  va3 = va1 + va2;   // + operates on all elements
  print(va3);

  va3 = pow(va2, va1);  // pow() operates on elements
  print(va3);

  cout << "sum is " << va1.sum() << endl;
  cout << "max is " << va1.max() << endl;
  cout << "min is " << va1.min() << endl;
  return 0;
}
```