**CPP - STL**

--------------

important C++ STL functions

-------------------------------------

1. Containers

------------------

bool is\_inContainer = container.find(x) != container.end()

2. Algorithms

-----------------

std::sort( container.begin(),container.end(), <Comparator>)

- no need comparatoer for basic types.

- need comparator for custom classes, structs, you can provide operator() for comparator.

std::reverse( container.begin(), container.end())

3. Strings

-------------

substring :

substring(start,end);

substring(stat) // lenght is end

append :

+ operator

4. Streams

--------------

p = "2000"

int i = stoi(p)

Good Online Tutorials

----------------------------

http://www.dreamincode.net/forums/topic/95826-stringstream-tutorial/

http://www.devarticles.com/c/a/Cplusplus/Using-Stringstreams-in-Cplusplus/

http://www.cprogramming.com/tutorial/c++-iostreams.html

stringstream

read input as a file.

convert to and from int,double,char etc.. to string.

istringstream

string myStream = "45";

istringstream buffer(myString);

int value;

buffer >> value; // value = 45

ostringstream

ss.str();

converting base

stringstream ss;

ss << "f";

ss << std::hex;

int f ;

ss >> f;

converting int to string

int Number = 123; // number to be converted to a string string Result; // string which will contain the result ostringstream convert; // stream used for the conversion convert << Number; // insert the textual representation of 'Number' in the characters in the stream Result = convert.str(); // set 'Result' to the contents of the stream

**STL set operations**

A summary with code samples of the set operations that come with STL: union, intersection, symmetrical difference and set difference.

For consistency, the two sets of integer vectors used in each example are the same and are:

1. vals1 = { 1, 2, 4 }
2. vals2 = { 1, 2, 5, 7 };

**Unions**

The union of two sets is formed by the elements that are present in either one of the sets, or in both.

1. #include <iostream>
2. #include <algorithm>
3. #include <iterator>
4. #include <string>
5. #include <vector>
7. **typedef** std::vector<**int**>::iterator iter;
9. **template**<**typename** T, **typename** InputIterator>
10. **void** Print(std::ostream& ostr,
11. InputIterator itbegin,
12. InputIterator itend,
13. **const** std::string& delimiter)
14. {
15. std::copy(itbegin,
16. itend,
17. std::ostream\_iterator<T>(ostr, delimiter.c\_str()));
18. }
20. **int** main()
21. {
22. // Initialise sample data sets and vectors
23. **int** vals1[] = { 1, 2, 4 };
24. **int** vals2[] = { 1, 2, 5, 7 };
25. **int** size1 = **sizeof**( vals1 ) / **sizeof** ( vals1[ 0 ] );
26. **int** size2 = **sizeof**( vals2 ) / **sizeof** ( vals2[ 0 ] );
28. std::vector<**int**> v1( vals1, vals1 + size1 );
29. std::vector<**int**> v2( vals2, vals2 + size2 );
30. std::vector<**int**> v( size1 + size2 );
31. std::vector<**int**>::iterator it\_union;
32. std::vector<**int**>::iterator it;
34. // Sort the vectors (essential)
35. std::sort( v1.begin(), v1.end() );
36. std::sort( v2.begin(), v2.end() );
38. // Find the union of the two vector sets: elements present in either
39. // of the sets or both
40. it\_union = set\_union ( v1.begin(), v1.end(), v2.begin(), v2.end(), v.begin());
41. **int** union\_size = **int**( it\_union - v.begin() );
42. std::cout << "Union size = " << union\_size << std::endl;
44. // Print the union
45. Print<**int**, iter>( std::cout, v.begin(), it\_union, " " );
47. **return** 0;
48. }

Output:

1. Union size = 5
2. 1 2 4 5 7

**Intersections**

Intersections consist of elements present in both sets at the same time.

1. #include <iostream>
2. #include <algorithm>
3. #include <iterator>
4. #include <string>
5. #include <vector>
7. **typedef** std::vector<**int**>::iterator iter;
9. **template**<**typename** T, **typename** InputIterator>
10. **void** Print(std::ostream& ostr,
11. InputIterator itbegin,
12. InputIterator itend,
13. **const** std::string& delimiter)
14. {
15. std::copy(itbegin,
16. itend,
17. std::ostream\_iterator<T>(ostr, delimiter.c\_str()));
18. }
20. **int** main()
21. {
22. // Initialise sample data sets and vectors
23. **int** vals1[] = { 1, 2, 4 };
24. **int** vals2[] = { 1, 2, 5, 7 };
25. **int** size1 = **sizeof**( vals1 ) / **sizeof** ( vals1[ 0 ] );
26. **int** size2 = **sizeof**( vals2 ) / **sizeof** ( vals2[ 0 ] );
28. std::vector<**int**> v1( vals1, vals1 + size1 );
29. std::vector<**int**> v2( vals2, vals2 + size2 );
30. std::vector<**int**> v( size1 + size2 );
31. std::vector<**int**>::iterator it\_intersect;
32. std::vector<**int**>::iterator it;
34. // Sort the vectors (essential)
35. std::sort( v1.begin(), v1.end() );
36. std::sort( v2.begin(), v2.end() );
38. // Find the intersection of the two vector sets: elements present in both sets
39. it\_intersect = set\_intersection( v1.begin(), v1.end(), v2.begin(), v2.end(), v.begin());
40. **int** intersect\_size = **int**( it\_intersect - v.begin() );
41. std::cout << "Intersection size = " << intersect\_size << std::endl;
43. // Print the intersection
44. Print<**int**, iter>( std::cout, v.begin(), it\_intersect, " " );
46. **return** 0;
47. }

Output:

1. Intersection size = 2
2. 1 2

**Symmetric Differences**

The set of elements that are present in one of the sets, but not in the other.

1. #include <iostream>
2. #include <algorithm>
3. #include <iterator>
4. #include <string>
5. #include <vector>
7. **typedef** std::vector<**int**>::iterator iter;
9. **template**<**typename** T, **typename** InputIterator>
10. **void** Print(std::ostream& ostr,
11. InputIterator itbegin,
12. InputIterator itend,
13. **const** std::string& delimiter)
14. {
15. std::copy(itbegin,
16. itend,
17. std::ostream\_iterator<T>(ostr, delimiter.c\_str()));
18. }
20. **int** main()
21. {
22. // Initialise sample data sets and vectors
23. **int** vals1[] = { 1, 2, 4 };
24. **int** vals2[] = { 1, 2, 5, 7 };
25. **int** size1 = **sizeof**( vals1 ) / **sizeof** ( vals1[ 0 ] );
26. **int** size2 = **sizeof**( vals2 ) / **sizeof** ( vals2[ 0 ] );
28. std::vector<**int**> v1( vals1, vals1 + size1 );
29. std::vector<**int**> v2( vals2, vals2 + size2 );
30. std::vector<**int**> v( size1 + size2 );
31. std::vector<**int**>::iterator it\_symm\_diff;
32. std::vector<**int**>::iterator it;
34. // Sort the vectors (essential)
35. std::sort( v1.begin(), v1.end() );
36. std::sort( v2.begin(), v2.end() );
38. // The symmetric difference of two sets is formed by the elements that are present in one
39. // of the sets, but not in the other.
40. it\_symm\_diff = set\_symmetric\_difference( v1.begin(), v1.end(), v2.begin(), v2.end(), v.begin());
41. **int** symm\_diff\_size = **int**( it\_symm\_diff - v.begin() );
42. std::cout << "Symmetric difference size = " << symm\_diff\_size << std::endl;
44. // Print the symmetric difference
45. Print<**int**, iter>( std::cout, v.begin(), it\_symm\_diff, " " );
47. **return** 0;
48. }

Output:

1. Symmetric difference size = 3
2. 4 5 7

**Set Differences**

The elements that present in the first set, but not in the second one.

1. #include <iostream>
2. #include <algorithm>
3. #include <iterator>
4. #include <string>
5. #include <vector>
7. **typedef** std::vector<**int**>::iterator iter;
9. **template**<**typename** T, **typename** InputIterator>
10. **void** Print(std::ostream& ostr,
11. InputIterator itbegin,
12. InputIterator itend,
13. **const** std::string& delimiter)
14. {
15. std::copy(itbegin,
16. itend,
17. std::ostream\_iterator<T>(ostr, delimiter.c\_str()));
18. }
20. **int** main()
21. {
22. // Initialise sample data sets and vectors
23. **int** vals1[] = { 1, 2, 4 };
24. **int** vals2[] = { 1, 2, 5, 7 };
25. **int** size1 = **sizeof**( vals1 ) / **sizeof** ( vals1[ 0 ] );
26. **int** size2 = **sizeof**( vals2 ) / **sizeof** ( vals2[ 0 ] );
28. std::vector<**int**> v1( vals1, vals1 + size1 );
29. std::vector<**int**> v2( vals2, vals2 + size2 );
30. std::vector<**int**> v( size1 + size2 );
31. std::vector<**int**>::iterator it\_set\_diff;
32. std::vector<**int**>::iterator it;
34. // Sort the vectors (essential)
35. std::sort( v1.begin(), v1.end() );
36. std::sort( v2.begin(), v2.end() );
38. // The set difference of two sets is formed by the elements that are present in the first
39. // sets, but not in the second.
40. it\_set\_diff = set\_difference( v1.begin(), v1.end(), v2.begin(), v2.end(), v.begin());
41. **int** set\_diff\_size = **int**( it\_set\_diff - v.begin() );
42. std::cout << "Set difference size = " << set\_diff\_size << std::endl;
44. // Print the set difference
45. Print<**int**, iter>( std::cout, v.begin(), it\_set\_diff, " " );
47. **return** 0;
48. }

Output:

1. Set difference size = 1
2. 4

**Practical example: using union / set intersection to calculate the Jaccard Index**

The [Jaccard index](http://en.wikipedia.org/wiki/Jaccard_index) is a means of measuring the similarity and/or diversity of sample sets:

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
4. #include <iterator>
5. #include <string>

8. **class** Token
9. {
10. **private**:
11. std::string str;
13. **public**:
14. Token() {}
15. Token( std::string val ) : str( val ) {}
16. std::string Word() **const**
17. { **return** str; }
18. };
20. **struct** Ascending
21. {
22. **bool** operator() ( Token& start, Token& end )
23. {
24. **return** start.Word() < end.Word();
25. }
26. };

29. **int** main()
30. {
31. std::string str1[] = { "The", "crazy", "cat", "sat", "on",  "the", "mat" };
32. std::string str2[] = { "The", "cat",   "sat", "on",  "the", "red", "mat" };
34. std::vector<Token>::iterator tok\_intersect, tok\_union;
36. **int** size1 = **sizeof**( str1 ) / **sizeof**( str1[ 0 ] );
37. **int** size2 = **sizeof**( str2 ) / **sizeof**( str2[ 0 ] );
38. std::vector<Token> tokens1( str1, str1 + size1 );
39. std::vector<Token> tokens2( str2, str2 + size2 );
41. std::vector<Token> tokens( size1 + size2 );
43. std::sort( tokens1.begin(), tokens1.end(), Ascending() );
44. std::sort( tokens2.begin(), tokens2.end(), Ascending() );
46. tok\_intersect = std::set\_intersection( tokens1.begin(),
47. tokens1.end(),
48. tokens2.begin(),
49. tokens2.end(),
50. tokens.begin(),
51. Ascending() );
53. **int** intersect\_size = **int**( tok\_intersect - tokens.begin() );
55. tok\_union = std::set\_union ( tokens1.begin(),
56. tokens1.end(),
57. tokens2.begin(),
58. tokens2.end(),
59. tokens.begin(),
60. Ascending() );
62. **int** union\_size = **int**( tok\_union - tokens.begin() );
64. **double** JaccardIndex = (**double**) intersect\_size / (**double**) union\_size;
66. **return** 0;
67. }

In this example, the size of the union set is 8, since there are 8 words in either one of the sets, or in both: "The", "crazy", "cat", "sat", "on", "the", "red", "mat". The intersection size is 6 since there are 6 words present in both sets at the same time: "The", "cat", "sat", "on", "the", "mat". Thus the Jaccard Index is calculated as 6 / 8 = 0.75.