Contents

[01 Learn STL - Introduction of Templates – Not there 2](#_Toc392267744)

[02 Introduction of STL 1 – Overview 2](#_Toc392267745)

[03 Introduction of STL 2 - Sequence Containers 3](#_Toc392267746)

[04 Introduction of STL 3 - Associative Containers 6](#_Toc392267747)

[05 Introduction of STL 4 - Unordered Containers 8](#_Toc392267748)

[06 Introduction of STL 5 - Iterators and Algorithms 10](#_Toc392267749)

[07 Introduction of STL 6 - Functors 16](#_Toc392267750)

[08 STL Algorithms 1 - Non-modifying Algorithms 20](#_Toc392267751)

[09 STL Algorithms 2 - Modifying Algorithms 23](#_Toc392267752)

[10 STL Algorithms 3 – Sorting 29](#_Toc392267753)

[11 STL Algorithms 4 - Sorted Data n Numeric Algorithms 32](#_Toc392267754)

[12 C++ String 1 - Constructor and Size 35](#_Toc392267755)

[13 C++ String 2 - Accessing String Characters 37](#_Toc392267756)

[14 C++ String 3 - Member Function Algorithms 39](#_Toc392267757)

[15 C++ String 4 - Non-member Functions 40](#_Toc392267758)

[16 C++ String 5 - String and Algorithms 41](#_Toc392267759)

[17 C++ Stream 1 – Introductory 43](#_Toc392267760)

[18 C++ Stream 2 - File Stream and Error Handling 44](#_Toc392267761)

[19 C++ Stream 3 - Formatted and Unformatted IO 45](#_Toc392267762)

[20 C++ Stream 4 – Manipulators 47](#_Toc392267763)

[21 C++ Stream 5 - Stream Buffer 48](#_Toc392267764)

[22 C++ Stream 6 - String Stream 49](#_Toc392267765)

[23 C++ Stream 7 - Enable Streaming for Your Own Class 50](#_Toc392267766)

# 01 Learn STL - Introduction of Templates – Not there

# 02 Introduction of STL 1 – Overview

int main(void)

{

vector<int> vec;

vec.push\_back(4);

vec.push\_back(5);

vec.push\_back(2);

vec.push\_back(-9);

vector<int>::iterator itr1 = vec.begin();

vector<int>::iterator itr2 = vec.end();

// iterator is like a pointer

// We can assign value to them, compare them like pointer, increment them

// and we can dereference them.

for (vector<int>::iterator itr = itr1; itr != itr2; itr++)

{

cout << \*itr << " ";

}

// vec.begin() vec.end()

//   +                         +

//   |                         |

//   |                         |

//   v                         v

// +----+----+----+----+----+

// |    |    |    |    |    |

// +----+----+----+----+----+

// cout << \*vec.end() << endl; // BOOM!!!

sort(itr1, itr2);

cout << endl;

for (vector<int>::iterator itr = itr1; itr != itr2; itr++)

{

cout << \*itr << " ";

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Reasons to use C++ standard library:

1. Code reuse, no need to re-invent the wheel.

2. Efficiency.

3. Accurate, less buggy.

4. Terse, readable code, reduced control flow.

5. standardization, guaranteed availability.

6. A role model of writing library.

7. Good knowledge of data structures and algorithm.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

return 0;

}

# 03 Introduction of STL 2 - Sequence Containers

#include <iostream>

#include <vector>

#include <deque>

#include <list>

#include <forward\_list>

#include <array>

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Containers

1. Sequence containers (array and linked list)

- STL provides vector, deque, list, forward list and array

2. Associative containers (binary tree) : always sorted

- set, multiset

- map, multimap

3. Unordered containers (hash tables)

- Unordered set/multiset

- Unordered map/multimap

#include <vector>

#include <deque>

#include <list>

#include <forward\_list>

#include <array>

#include <set> // Set and multiset

#include <map> // Map and multimap

#include <unordered\_set> // Unordered set/multiset

#include <unordered\_map> // Unordered map/multimap

#include <iterator>

#include <algorithm>

#include <numeric> // Some numeric algorithm

#include <functional> // For functors

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(void)

{

#pragma region Vector

{

cout << "##################### VECTOR #####################" << endl;

vector<int> aVec; // aVec.size() == 0

aVec.push\_back(4);

aVec.push\_back(1);

aVec.push\_back(6);

aVec.push\_back(7); // aVec: {4, 1, 6, 7}; aVec.size() == 4

//cout << aVec[8] << endl;// This was of accessing doesn't do "range check"

//cout << aVec.at(8) << endl;// throws range\_error exception of out of range

for (int i = 0; i < aVec.size(); i++)

cout << aVec[i] << " ";

cout << endl;

for (vector<int>::iterator itr = aVec.begin(); itr != aVec.end(); itr++)

cout << \*itr << " ";

cout << endl;

for (auto it : aVec) // In C++11

cout << it << " ";

cout << endl;

{ // Common member functions provided by all containers

if (aVec.empty()) { cout << "Not possible\n"; } // aVec.size() == 4

cout << aVec.size() << endl;

vector<int> bVec(aVec); // Copy constructor

aVec.clear(); // Removes all items in aVec and aVec.size() == 0

bVec.swap(aVec); // Interchange the elements

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Properties of vector

1. Fast insert/remove at the end : O(1)

2. Slow insert/remove at the beginning or in the middle : O(n)

3. Slow search : O(n)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma endregion Vector

#pragma region Deque

// Deque can grow both at the end and at the beginning

//  +-----+-----+-----+-----+-----+-----+-----+

// <----      |     |     |     |     |     |      ---->

//  +-----+-----+-----+-----+-----+-----+-----+

cout << "##################### DEQUE #####################" << endl;

deque<int> aDeque{ 4, 6, 7 };

aDeque.push\_back(2); // { 4, 6, 7, 2 };

aDeque.push\_front(5); // { 5, 4, 6, 7, 2 };

for (deque<int>::iterator itr = aDeque.begin(); itr != aDeque.end(); itr++)

cout << \*itr << " ";

cout << endl;

// Deque has similar interface with vector - Random Access

cout << aDeque[2] << endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Properties of deque

1. Fast insert/remove at the beginning and at the end : O(1)

2. Slow insert/remove in the middle : O(n)

3. Slow search : O(n)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma endregion Deque

#pragma region List

cout << "##################### LIST #####################" << endl;

// List is doubly liked list!!!

// +--> +----+-----------> +----+--->

//  |  5 |             |  7 |

// <----+----+ <-----------+----+ <---+

list<int> aList{ 5, 2, 9 };

aList.push\_back(6); // { 5, 2, 9, 6 };

aList.push\_front(8); // { 8, 5, 2, 9, 6 };

for (list<int>::iterator itr = aList.begin(); itr != aList.end(); itr++)

cout << \*itr << " ";

cout << endl;

list<int>::iterator itr = find(aList.begin(), aList.end(), 2);

// Will insert the item in front of the iterator

aList.insert(itr, 100); // { 8, 5, 100, 2, 9, 6 };

for (list<int>::iterator itr = aList.begin(); itr != aList.end(); itr++)

cout << \*itr << " ";

cout << endl;

itr++;

aList.erase(itr); // { 8, 5, 100, 2, 6 };

for (list<int>::iterator itr = aList.begin(); itr != aList.end(); itr++)

cout << \*itr << " ";

cout << endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Properties of list

1. Fast insert/remove at any place : O(1)

2. No random access, no [] operator.

3. Slow search : O(n)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma endregion List

#pragma region Forward List

cout << "##################### FORWARD LIST #####################" << endl;

// +-----+-----> +-----+-----> +-----+-----> +-----+

// |     |       |     |       |     |       |     |

// +-----+       +-----+       +-----+       +-----+

forward\_list<int> aForwardList{ 1, 2, 3, 4 };

aForwardList.push\_front(6); // { 6, 1, 2, 3, 4 };

for (forward\_list<int>::iterator itr = aForwardList.begin(); itr != aForwardList.end(); itr++)

cout << \*itr << " ";

cout << endl;

#pragma endregion Forward List

#pragma region Array

cout << "##################### ARRAY #####################" << endl;

array<int, 5> aArray = { 1, 2, 3, 4, 5 };

for (array<int, 5>::iterator itr = aArray.begin(); itr != aArray.end(); itr++)

cout << \*itr << " ";

cout << endl;

#pragma endregion Array

return 0;

}

# 04 Introduction of STL 3 - Associative Containers

#include <iostream>

#include <set>

#include <map>

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Associative Container

- The elements of associative container are always sorted.

- They are implemented using binary tree.

- Elements are sorted using operator <

- No push\_back() or push\_front(), cos these operation doesn't make

any sense.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(void)

{

#pragma region Set

// Set/Multiset: value of the elements can't be modified!!!

// Set doesn't allow duplicate items!!!

cout << "##################### SET #####################" << endl;

set<int> aSet;

// Insert take log time

aSet.insert(3); // aSet: { 3 }

aSet.insert(10); // aSet: { 3, 10 }

aSet.insert(-4); // aSet: { -4, 3, 10 }

for (set<int>::iterator itr = aSet.begin(); itr != aSet.end(); itr++)

cout << \*itr << " ";

cout << endl;

// Search takes log time

set<int>::iterator itr = aSet.find(10); // O(log(n)), itr points to 10

// Insertion

pair<set<int>::iterator, bool> ret;

ret = aSet.insert(10);

if (false == ret.second)

{

itr = ret.first; // Now the itr points to 10

cout << "Set already contains this element!!!" << endl;

}

// Note that insert will not insert at the end, but will be inserted at

// proper position. By providing itr to insert function, we are providing

// hint to the compiler.

aSet.insert(itr, 9); // aSet: { -4, 3, 9, 10 }

for (set<int>::iterator itr = aSet.begin(); itr != aSet.end(); itr++)

cout << \*itr << " ";

cout << endl;

#pragma endregion Set

#pragma region MultiSet

cout << "##################### MULTI SET #####################" << endl;

// Allows duplicate items!!!

multiset<int> aMultiSet;

aMultiSet.insert(10); // aMultiSet: { 10 }

aMultiSet.insert(11); // aMultiSet: { 10, 11}

aMultiSet.insert(1); // aMultiSet: { 1, 10, 11 }

aMultiSet.insert(12); // aMultiSet: { 1, 10, 11, 12 }

aMultiSet.insert(10); // aMultiSet: { 1, 10, 10, 11, 12 }

for (multiset<int>::iterator itr = aMultiSet.begin(); itr != aMultiSet.end(); itr++)

cout << \*itr << " ";

cout << endl;

multiset<int>::iterator itr1 = aMultiSet.find(10);

// \*itr1 = 10; // can't be modified.

for (; itr1 != aMultiSet.end(); itr1++)

cout << \*itr1 << " ";

cout << endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Properties of vector

1. Fast search: O(log(n)).

2. Traversing is slow (compared to vector & deque).

3. No random access, no [] operator.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma endregion MultiSet

#pragma region Map

cout << "##################### MAP #####################" << endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Associative containers are always sorted. Sometimes,

people want to sort based on keys. So we can use

Map or Multimap!!!

Keys can't be modified!!!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Map doesn't allow duplicate items!!!

map<char, int> aMap;

aMap.insert(pair<char, int>('a', 100));

aMap.insert(make\_pair('z', 200));

for (map<char, int>::iterator itr = aMap.begin(); itr != aMap.end(); itr++)

cout << (\*itr).first << " => " << (\*itr).second << endl;

cout << endl;

for (auto itr = aMap.begin(); itr != aMap.end(); itr++)

cout << (\*itr).first << " => " << (\*itr).second << endl;

cout << endl;

auto it = aMap.begin();

aMap.insert(it, pair<char, int>('b', 400));

for (auto itr = aMap.begin(); itr != aMap.end(); itr++)

cout << (\*itr).first << " => " << (\*itr).second << endl;

cout << endl;

#pragma endregion Map

#pragma region Multimap

cout << "##################### MULTI MAP #####################" << endl;

multimap<char, int> aMultiMap;

aMultiMap.insert(pair<char, int>('n', 100));

aMultiMap.insert(make\_pair('i', 200));

aMultiMap.insert(pair<char, int>('s', 100));

aMultiMap.insert(pair<char, int>('s', 500));

for (auto itr = aMultiMap.begin(); itr != aMultiMap.end(); itr++)

cout << (\*itr).first << " => " << (\*itr).second << endl;

cout << endl;

#pragma endregion Multimap

return 0;

}

# 05 Introduction of STL 4 - Unordered Containers

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Unordered Containers (C++11)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(void)

{

#pragma region UnorderedSet

cout << "##################### Unordered Set #####################" << endl;

// Doesn't allow duplicate items!!!

// 2 items with red are there, but only one item will be stored!!!

unordered\_set<string> aUnorderedSet = { "red", "blue", "green", "red" };

for (auto it = aUnorderedSet.begin(); it != aUnorderedSet.end(); it++)

cout << \*it << " ";

cout << endl;

auto itr = aUnorderedSet.find("green"); // Takes O(1)

if (itr != aUnorderedSet.end()) // Important check

cout << \*itr << endl;

aUnorderedSet.insert("yellow"); // Takes O(1)

vector<string> aVec = { "purple", "pink" };

aUnorderedSet.insert(aVec.begin(), aVec.end());

// Hash table specific API's

// Ratio of total number of elements to total number of buckets

cout << "Load Factor = " << aUnorderedSet.load\_factor() << endl;

string x = "red";

// In which bucket the element is stored?

cout << x << " is in bucket #" << aUnorderedSet.bucket(x) << endl;

cout << "Total buckets = " << aUnorderedSet.bucket\_count() << endl;

#pragma endregion UnorderedSet

#pragma region UnorderedMultiset

cout << "##################### Unordered Multiset #####################" << endl;

// Unordered set that allows duplicate items!!!

unordered\_multiset<string> multiSet = { "red", "blue", "green", "red" };

for (auto itr1 = multiSet.begin(); itr1 != multiSet.end(); itr1++)

cout << \*itr1 << " ";

cout << endl;

#pragma endregion UnorderedMultiset

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Properties of Unordered Containers

1. Fastest search/insert at any place: O(1)

Associative containers takes O(log(n))

vector, deque takes O(n)

list takes O(1) to insert and O(n) to search

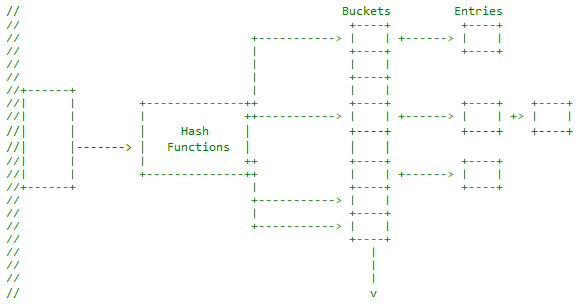
2. Unordered set/multiset: element value cannot be changed.

Unordered map/multimap: element key cannot be changed.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

return 0;

}



# 06 Introduction of STL 5 - Iterators and Algorithms

#include <iostream>

#include <vector>

#include <list>

#include <forward\_list>

#include <set>

#include <algorithm>

#include <iterator>

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Iterators

- There are 5 categories of iterators

1. Random Access Iterators: vector, deque, array

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool isOdd(int i) { return (i % 2); }

int main(void)

{

#pragma region No 1 Random Access Iterator

vector<int> aVec;

aVec.push\_back(10);

aVec.push\_back(1);

aVec.push\_back(12);

aVec.push\_back(14);

aVec.push\_back(92);

aVec.push\_back(44);

aVec.push\_back(36);

aVec.push\_back(25);

// We can access the elements of the container randomly!!!

// vector, deque, array provides random access iterator

auto itr = aVec.begin();

auto itr2 = aVec.end();

// With random access iterators, we can compare 2 iterators

//if (itr == itr2)

// Adding value to iterator

itr += 5; // Advance itr by 5

itr -= 4;

++itr; // faster than itr++

--itr;

#pragma endregion No 1 Random Access Iterator

#pragma region No 2 Bidirectional Iterator

// list, set/multiset, map/multimap provides Bidirectional Iterator

// With bidirectional iterator, we can increment

// and decrement the iterator but we cannot add or subtract value

// to/from the iterator.

// We can't compare 2 iterators!!!

list<int> aList;

aList.push\_front(12);

aList.push\_back(14);

aList.push\_back(92);

aList.push\_back(44);

aList.push\_back(36);

aList.push\_back(25);

auto aListItr = aList.begin();

aListItr++;

aListItr--;

#pragma endregion No 2 Bidirectional Iterator

#pragma region No 3 Forward Iterator

// Can only be incremented cannot be decremented

forward\_list<int> fList;

fList.push\_front(10);

fList.push\_front(14);

fList.push\_front(92);

fList.push\_front(44);

fList.push\_front(36);

fList.push\_front(25);

auto fListItr = fList.begin();

fListItr++;

//fListItr--; // Error!!!

#pragma endregion No 3 Forward Iterator

// Unordered containers provide at least Forward iterator.

// But they have the option to provide Bidirectional iterator!!!

#pragma region No 4 Input Iterator

// Read and process values while iterating forward

// We can read and dereference input iterator but we can't write to it!!!

/\* Both input and output iterator can only move forward \*/

#pragma endregion No 4 Input Iterator

#pragma region No 5 Output Iterator

// Output values while iterating forward

// We can write the values but we can't write to it!!!

// Output iterator can only move forward

#pragma endregion No 5 Output Iterator

#pragma region Const Iterator

// Every container has a iterator and a const\_iterator

{

cout << "######################## CONST\_ITERATOR ######################" << endl;

set<int>::iterator itr;

set<int>::const\_iterator citr, citr1; // Read only access to container elements

set<int> aSetP{ 2, 3, 4, 5, 6, 7, 8, 9, 12, 34, 56, 78, 99 };

for (citr = aSetP.begin(); citr != aSetP.end(); citr++)

cout << \*citr << " ";

cout << endl;

for\_each(aSetP.begin(), aSetP.end(), [] (int const& i) { cout << i << " "; });

cout << endl;

// Some functions

//citr = aSetP.begin();

//citr1 = aSetP.begin();

//advance(citr, 5); // Move citr forward 5 spots. -> citr += 5;

//distance(citr, citr1); // Measure the distance between citr and citr1

}

#pragma endregion Const Iterator

#pragma region Iterator Adapter

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Iterator Adapter (Predefined Iterator)

- A special, more powerful iterator

1. Insert iterator.

2. Stream iterator.

3. Reverse iterator.

4. Move iterator.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma region Insert Iterator

{

vector<int> vec1 = { 4, 5 };

vector<int> vec2 = { 12, 13, 14, 15 };

vector<int>::iterator it = find(vec2.begin(), vec2.end(), 14);

insert\_iterator<vector<int>> i\_itr(vec2, it);

copy(vec1.begin(), vec1.end(), i\_itr); //{ 12, 13, 4, 5, 14, 15 }

for (auto itr3 = vec2.begin(); itr3 != vec2.end(); itr3++)

cout << \*itr3 << " ";

cout << endl;

// There are other insert iterators

// 1. back\_insert\_iterator - inserts at the back

// 2. front\_insert\_iterator - inserts at the end

}

#pragma endregion Insert Iterator

#pragma region Stream Iterator

// Is to iterate through the data of stream

{

cout << "########################### STREAM ITERATOR ###########################" << endl;

vector<int> vec2 = { 12, 13, 14, 15 };

// Copies the content of vec2 to output stream.

copy(vec2.begin(), vec2.end(), ostream\_iterator<int>(cout, " "));

cout << endl;

}

#pragma endregion Stream Iterator

#pragma region Reverse Iterator

// To traverse the container in the reverse order.

{

cout << "########################### REVERSE ITERATOR ###########################" << endl;

vector<int> vec2 = { 12, 13, 14, 15 };

reverse\_iterator<vector<int>::iterator> ritr;

// rbegin() and rend() are the functions provided by STL

for (ritr = vec2.rbegin(); ritr != vec2.rend(); ritr++)

cout << \*ritr << " ";

cout << endl;

}

#pragma endregion Reverse Iterator

#pragma endregion Iterator Adapter

#pragma region Algorithms

{

cout << "######################### ALGORITHMS #########################" << endl;

vector<int> vec2 = { 12, 66, 88, 7, 16, 18, -9, 33, 77 };

auto itr = min\_element(vec2.begin(), vec2.end());

cout << "Minimum in the vector is: " << \*itr << endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

- Algorithm functions needs to know the where the data is to work

on and that data is often time is the range of data represented

by pair of iterators

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Note 1:

{

cout << "########################### Note 1 ###########################" << endl;

// Algorithms always process range in a half-open way. This means

// it includes first item, but not include last item.

// Example:

// vec2 = { 12, 66, 88, 7, 16, 18, -9, 33, 77 };

for (auto it = vec2.begin(); it != vec2.end(); it++)

cout << \*it << " ";

cout << endl;

auto itr1 = min\_element(vec2.begin(), vec2.end()); // itr1 points to -9

// vec2.begin() points to 12 which is included

// itr1 points to -9 which is not included

sort(vec2.begin(), itr1); // [begin, end)

// vec2 = { 7, 12, 16, 18, 66, 88, -9, 33, 77 }

for (auto it = vec2.begin(); it != vec2.end(); it++)

cout << \*it << " ";

cout << endl;

reverse(itr1, vec2.end()); // vec2 = { 7, 12, 16, 18, 66, 88, 77, 33, -9 }

for (auto it = vec2.begin(); it != vec2.end(); it++)

cout << \*it << " ";

cout << endl;

}

// Note 2:

{

cout << "########################### Note 2 ###########################" << endl;

// Sometimes algorithms needs not only one range of data, it needs

// 2 range of data

vector<int> vec = { 12, 66, 88, 7, 16, 18, -9, 33, 77 };

vector<int> vec2(10);

cout << "Original vector is: ";

for (auto it = vec.begin(); it != vec.end(); it++)

cout << \*it << " ";

cout << endl;

auto itr = find(vec.begin(), vec.end(), 7);

copy(itr, vec.end(), // This is the source

vec2.begin()); // Destination

for (auto it = vec2.begin(); it != vec2.end(); it++)

cout << \*it << " ";

cout << endl;

}

}

{ // Algorithm with function

cout << "########################### Note 3 ###########################" << endl;

vector<int> vec = { 12, 66, 88, 7, 16, 18, -9, 33, 77 };

cout << "Original vector is: ";

for (auto it = vec.begin(); it != vec.end(); it++)

cout << \*it << " ";

cout << endl;

auto fit = find\_if(vec.begin(), vec.end(), isOdd);

cout << "First Odd Element:  " << \*fit << endl;

}

{ // Algorithms can even with C++ native array

cout << "########################### Note 4 ###########################" << endl;

int myArray[10] = { 1, 2, 3, 4, 50, 9, 8, 7, 6 };

for (int i : myArray) cout << i << " ";

cout << endl;

sort(myArray, myArray + (sizeof(myArray) / sizeof(int)));

for (int i : myArray) cout << i << " ";

cout << endl;

}

#pragma endregion Algorithms

return 0;

}

# 07 Introduction of STL 6 - Functors

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

#include <functional>

#include <set>

#include <deque>

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Functors

What is the benefit of defining a a function? Why don't we use

regular function?

Benefits of functors:

1. Smart function that can do more than a regular function. For example

the class object can declare a data member which can remember the state.

2. Functor can have its own type.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma region Functor Intruducton

class X

{

public:

// Overloading operator ()

// For functor, we have to put the type before the keyword operator

void operator()(string str)

{

cout << "Calling functor X with parameter: " << str << endl;

}

// This is type conversion function!!!

// For type conversion function, we have to put the type after the

// keyword operator

operator string () const { return "X"; }

};

int main(void)

{

X foo; // Instance of X

//

foo("Hi there");

return 0;

}

#pragma endregion Functor Intruducton

#pragma region Problem

// add2 function is used to add 2 to each element in the vector.

// What if we want to add 3 instead of 2?

void add2(int i) { cout << i + 2 << " "; }

int main(void)

{

vector<int> vec{ 1, 3, 5, 6, 8, 4, 7 };

cout << "Original vector" <<endl;

for (int i : vec)cout << i << " ";

cout << endl;

for\_each(vec.begin(), vec.end(), add2);

return 0;

}

#pragma endregion Problem

#pragma region Solution 1 - Not Applicable

// Now we can change the val to any value we like. But template variable

// is resolved at compile time.

template<int val>

void addVal(int i) { cout << i + val << " "; }

int main(void)

{

vector<int> vec{ 1, 3, 5, 6, 8, 4, 7 };

cout << "Original vector" << endl;

for (int i : vec)cout << i << " ";

cout << endl;

for\_each(vec.begin(), vec.end(), addVal<2>);

// Since the template is resolved at compile time, below code won't compile.

//int x = 2;

//for\_each(vec.begin(), vec.end(), addVal<x>);

return 0;

}

#pragma endregion Solution 1 - Not Applicable

#pragma region Best Solution - Functors

class AddValue

{

int value\_;

public:

AddValue(int j) :value\_(j) { }

void operator()(int i)

{

cout << i + value\_ << endl;

}

};

int main(void)

{

vector<int> vec{ 1, 3, 5, 6, 8, 4, 7 };

cout << "Original vector" << endl;

for (int i : vec)cout << i << " ";

cout << endl;

int x = 2;

for\_each(vec.begin(), vec.end(), AddValue(x));

return 0;

}

#pragma endregion Best Solution - Functors

#pragma region Built-in Functors

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

STL provides built-in functors...

less, greater, greater\_equal, less\_equal, not\_equal\_to

logical\_and, logical\_not, logical\_or

multiplies, minus, plus, divides, modulus, negate

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(void)

{

int x = multiplies<int>()(3, 4);

cout << "Value of X: " << x << endl;

if (not\_equal\_to<int>() (x, 10))

cout << "X Not equal to 10" << endl;

return 0;

}

#pragma endregion Built-in Functors

#pragma region Parameter Binding

int main(void)

{

set<int> mySet{ 2, 3, 4, 5 };

vector<int> vec;

// We want to multiply each element of set by 10 and

// store the result in vector.

transform(mySet.begin(), mySet.end(), // Source

back\_inserter(vec), // Destination

// transform takes functor. But multiplies takes 2 parameters (multiplies<int>()(3, 4);)

// bind function can be used to bind and placeholders::\_1 is the place holder

// for the first argument for multiplies functor and will be substituted

// for each element in mySet.

bind(multiplies<int>(), placeholders::\_1, 10) // Functor

);

for (int i : vec) // vec: { 20, 30, 40, 50 };

cout << i << " ";

cout << endl;

return 0;

}

#pragma endregion Parameter Binding

#pragma region Function to Functor

double Pow(double x, double y) { return pow(x, y); }

int main(void)

{

set<int> mySet{ 3, 1, 25, 7, 12 };

deque<int> d;

// We want to raise every element of mySet to power of 2

// and save the result in deque d.

// function is used to convert regular function Pow into

// functor f!!!

auto f = function<double(double, double)>(Pow); // in C++11

transform(mySet.begin(), mySet.end(),

back\_inserter(d),

bind(f, placeholders::\_1, 2)

);

for (int i : d) // d: { 1, 9, 49, 144, 625 };

cout << i << " ";

cout << endl;

return 0;

}

#pragma endregion Function to Functor

#pragma region Need for Functor in STL

//bool lsb\_less(int x, int y) { return (x % 10) < (y % 10); }

class Lsb\_less

{

public:

bool operator()(int x, int y)

{

return (x % 10) < (y % 10);

}

};

int main(void)

{

set<int> mySet{ 3, 1, 25, 7, 12 }; // mySet is automatically sorted { 1, 3, 7, 12, 25 }

cout << "mySet is: ";

for (int i : mySet)

cout << i << " ";

cout << endl;

set<int, less<int>> mySet1 = { 3, 1, 25, 7, 12 };

cout << "mySet1 is: ";

for (int i : mySet1)

cout << i << " ";

cout << endl;

set<int, Lsb\_less> leastSigByteSet = { 3, 1, 25, 7, 12 };

cout << "leastSigByteSet is: ";

for (int i : leastSigByteSet)

cout << i << " ";

cout << endl;

return 0;

}

#pragma endregion Need for Functor in STL

#pragma region Predicate

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Predicate

A function or a functor that

1. Returns a boolean or something that can be converted to bool.

2. Doesn't modify the data.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class NeedCopy

{

public:

bool operator()(int x)

{

return (x > 20) || (x < 5);

}

};

int main(void)

{

set<int> mySet{ 3, 1, 25, 7, 12 };

deque<int> d;

cout << "My set is: ";

for (int i : mySet) { cout << i << " "; }

cout << endl;

transform(mySet.begin(), mySet.end(), // Source

back\_inserter(d), // Destination

NeedCopy()

);

cout << "My deque is: ";

for (int i : d) { cout << i << " "; }

cout << endl;

return 0;

}

#pragma endregion Predicate

# 08 STL Algorithms 1 - Non-modifying Algorithms

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

STL Algorithm Walk-through

Non-modifying Algorithms

count, min and max, compare, linear search, attribute

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(void)

{

vector<int> vec = { 9, 60, 90, 8, 45, 87, 90, 69, 69, 55, 7 };

vector<int> vec2 = { 9, 60, 70, 8, 45, 87 };

vector<int> vec3 = { 9, 60, 8, 90, 87, 45, 69, 90, 69, 7, 55 };

vector<int>::iterator itr, itr2;

pair<vector<int>::iterator, vector<int>::iterator> pairOfItrs;

#pragma region Count

cout << "################ COUNT ################" << endl;

//Algo Data Operation

int count69 = count(vec.begin(), vec.end(), 69);

cout << "Number of elements with data 69 : " << count69 << endl;

//Algo Data

int countGt10 = count\_if(vec.begin(), vec.end(),

// Operation

[](int x){return x > 10; }

);

cout << "Number of elements > 10: " << countGt10 << endl;

#pragma endregion Count

#pragma region MinNMax

cout << "################ MINnMAX ################" << endl;

// Returns the first Max value

itr = max\_element(vec.begin(), vec.end());

cout << "Max element in the vector is: " << \*itr << endl;

itr = max\_element(vec.begin(), vec.end(),

// Own version of comparison function

// Below function will return the first element whose last digit is biggest

[](int x, int y){ return ((x % 10) < (y % 10)); }

);

cout << "Max element whose last digit is Max: " << \*itr << endl;

itr = min\_element(vec.begin(), vec.end());

cout << "Min element in the vector is: " << \*itr << endl;

pairOfItrs = minmax\_element(vec.begin(), vec.end());

cout << "First smallest element: " << \*pairOfItrs.first << endl;

cout << "Last biggest element: " << \*pairOfItrs.second << endl;

pairOfItrs = minmax\_element(vec.begin(), vec.end(),

[](int x, int y) { return ((x % 10) < (y % 10)); }

);

cout << "First smallest element: " << \*pairOfItrs.first << endl;

cout << "Last biggest element: " << \*pairOfItrs.second << endl;

#pragma endregion MinNMax

#pragma region Linear Searching

cout << "################ LINEAR SEARCH ################" << endl;

// Use linear search when the data is not sorted.

// Use binary search if the data is already sorted.

// Linear search returns the first item which matches the condition

itr = find(vec.begin(), vec.end(), 55);

cout << "Element is :" << \*itr << endl;

itr = find\_if(vec.begin(), vec.end(),

[](int x){ return x > 80; }

);

cout << "Element > 80 :" << \*itr << endl;

itr = find\_if\_not(vec.begin(), vec.end(),

[](int x){ return x > 80; }

);

cout << "Element < 80 :" << \*itr << endl;

itr = search\_n(vec.begin(), vec.end(), 2, 69);

cout << "Consecutive 2 items of 69 :" << \*itr << " and " << \*itr++ << endl;

// Search sub range

vector<int> sub{ 45, 87, 90 };

itr = search(vec.begin(), vec.end(), sub.begin(), sub.end());

cout << "Sub vector is : ";

for (auto i = itr; i != vec.end(); i++)

cout << \*i << " ";

cout << endl;

// Search any of

vector<int> items = { 87, 69 };

itr = find\_first\_of(vec.begin(), vec.end(),

items.begin(), items.end());

cout << "Search Any Of { 87, 69 }: " << \*itr << endl;

#pragma endregion Linear Searching

#pragma region Compare Range

cout << "################ COMPARE RANGE ################" << endl;

if (!equal(vec.begin(), vec.end(), vec2.begin()))

cout << "vec and vec2 are not same!!!" << endl;

if (is\_permutation(vec.begin(), vec.end(), vec3.begin()))

cout << "vec and vec3 have same items, but in different order!!!" << endl;

pairOfItrs = mismatch(vec.begin(), vec.end(), vec3.begin());

// returns the pair of iterators for first mismatch

cout << "Mismatch in vec:" << \*pairOfItrs.first << endl;

cout << "Mismatch in vec3:" << \*pairOfItrs.second << endl;

if (lexicographical\_compare(vec.begin(), vec.end(), vec3.begin(), vec3.end()))

cout << "vec < vec 3" << endl;

else

cout << "vec3 < vec " << endl;

// All of the comparing algorithms have generalized from of the same name.

#pragma endregion Compare Range

#pragma region Check Attributes

cout << "################ CHECK ATTRIBUTES ################" << endl;

if (is\_sorted(vec.begin(), vec.end()))

cout << "vec is sorted" << endl;

else

cout << "vec is not sorted" << endl;

// Finds the first place where the data is no longer sorted

itr = is\_sorted\_until(vec.begin(), vec.end());

cout << "From here vec is unsorted :";

for (auto i = itr; i != vec.end(); i++)

cout << \*i << " ";

cout << endl;

if (all\_of(vec.begin(), vec.end(), [](int x){ return x > 6; }))

cout << "All the vec items are > 6 !!!" << endl;

if (any\_of(vec.begin(), vec.end(), [](int x){ return x == 90; }))

cout << "One of the item in vec is equal to 90 !!!" << endl;

if (none\_of(vec.begin(), vec.end(), [](int x){ return x < 0; }))

cout << "One of the item in vec is < 0 !!!" << endl;

#pragma endregion Check Attributes

return 0;

}

# 09 STL Algorithms 2 - Modifying Algorithms

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Value Changing Algorithms

- copy, move, transform, swap, fill, replace, remove.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(void)

{

#pragma region Copy

{

vector<int> vec = { 9, 60, 70, 8, 45, 87, 90 }; // 7 items

vector<int> vec2 = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }; // 11 items

vector<int>::iterator itr, itr2;

pair<vector<int>::iterator, vector<int>::iterator> pair\_of\_itr;

cout << "################### COPY ###################" << endl;

cout << "vec2 before copy: ";

for (int& i : vec2) cout << i << " ";

cout << endl;

// Copies everything from vec to vec2.

// vec2 should be at least as big as vec. otherwise it is undefined behavior.

copy(vec.begin(), vec.end(),// Source

vec2.begin()); // Destination

cout << "vec2 after copy: ";

for (int& i : vec2) cout << i << " ";

cout << endl;

// copy\_if, copies everything that satisfies the condition

copy\_if(vec.begin(), vec.end(),// Source

vec2.begin(), // Destination

[](int x) { return x > 80; });// Condition

cout << "vec2 after copy\_if: ";

for (int& i : vec2) cout << i << " ";

cout << endl;

// copy\_n, copies n items from vec to vec2

copy\_n(vec.begin(), // Source

4, // Number of items

vec2.begin()); // Destination

cout << "vec2 after copy\_n: ";

for (int& i : vec2) cout << i << " ";

cout << endl;

// copy\_backward, copies vec to the end of vec2.

copy\_backward(vec.begin(), vec.end(),// Source

vec2.end());// Destination, here destination range starts with evc2.end()

cout << "vec2 after copy\_backward: ";

for (int& i : vec2) cout << i << " ";

cout << endl;

}

#pragma endregion Copy

cout << endl;

#pragma region Move

{

vector<string> sVec = { "apple", "orange", "pear", "grape" };// 4 items

vector<string> sVec2 = { "", "", "", "", "", "" };// 6 items

vector<string> sVec3 = { "A", "B", "C", "D" };// 4 items

cout << "################### MOVE ###################" << endl;

// move, moves the items from vec to vec2.

// If move semantics are defined for element type, elements are moved else

// they are copied with copy constructor, just like copy.

move(sVec.begin(), sVec.end(),// Source

sVec2.begin()); // Destination

cout << "sVec2 after move: ";

for (string& i : sVec2) cout << i << " ";

cout << endl;

// move\_backward, moves vec3 to end of vec2.

move\_backward(sVec3.begin(), sVec3.end(),// Source

sVec2.end());// Destination

cout << "sVec2 after move\_backward: ";

for (string& i : sVec2) cout << i << " ";

cout << endl;

}

#pragma endregion Move

cout << endl;

#pragma region Transform

{

vector<int> vec = { 9, 60, 70, 8, 45, 87, 90 }; // 7 items

vector<int> vec2 = { 9, 60, 70, 8, 45, 87, 90 };// 7 items

vector<int> vec3 = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }; // 11 items

cout << "################### TRANSFORM ###################" << endl;

// transforms the result and store it in another place.

// Below, transform will decrement all the items of vec by 1

// and stores the result in vec3.

transform(vec.begin(), vec.end(),// Source

vec3.begin(),// Destination

[](int x){ return x - 1; });// Operation

cout << "vec3 after transform: ";

for (int& i : vec3) cout << i << " ";

cout << endl;

transform(vec.begin(), vec.end(),// Source #1

vec2.begin(),// Source #2

vec3.begin(),// Destination

[](int x, int y){ return x + y; }// Operation

);

cout << "vec3 after transform: ";

for (int& i : vec3) cout << i << " ";

cout << endl;

}

#pragma endregion Transform

cout << endl;

#pragma region Swap Ranges

{

vector<int> vec = { 9, 60, 70, 8, 45, 87, 90 }; // 7 items

vector<int> vec2 = { 9, 60, 70, 8, 45, 87, 90 };// 7 items

vector<int> vec3 = { 0, 0, 0, 0, 0, 0, 0 }; // 7 items

cout << "################### TRANSFORM ###################" << endl;

cout << "vec3 before swap\_ranges: ";

for (int& i : vec3) cout << i << " ";

cout << endl;

swap\_ranges(vec.begin(), vec.end(), vec3.begin());

cout << "vec3 after swap\_ranges: ";

for (int& i : vec3) cout << i << " ";

cout << endl;

}

#pragma endregion Swap Ranges

cout << endl;

#pragma region Fill

{

vector<int> vec3 = { 0, 0, 0, 0, 0, 0, 0 }; // 7 items

cout << "################### FILL ###################" << endl;

cout << "vec3 before fill: ";

for (int& i : vec3) cout << i << " ";

cout << endl;

// Fills the vec3 with 9

fill(vec3.begin(), vec3.end(), 9);

cout << "vec3 after fill: ";

for (int& i : vec3) cout << i << " ";

cout << endl;

// fill\_n, will fill n items with certain value

fill\_n(vec3.begin(), 3, 0);

cout << "vec3 after fill\_n: ";

for (int& i : vec3) cout << i << " ";

cout << endl;

// generate takes a number generation function!!!

generate(vec3.begin(), vec3.end(), rand);

cout << "vec3 after generate: ";

for (int& i : vec3) cout << i << " ";

cout << endl;

generate\_n(vec3.begin(), 3, rand);

cout << "vec3 after generate\_n: ";

for (int& i : vec3) cout << i << " ";

cout << endl;

}

#pragma endregion Fill

cout << endl;

#pragma region Replace

{

vector<int> vec = { 9, 9, 70, 8, 9, 87, 90 };// 7 items

vector<int> vec2 = { 9, 60, 70, 8, 45, 87, 90 };// 7 items

vector<int> vec3 = { 0, 0, 0, 0, 0, 0, 0 }; // 7 items

cout << "################### REPLACE ###################" << endl;

cout << "vec before replace: ";

for (int& i : vec) cout << i << " ";

cout << endl;

replace(vec.begin(), vec.end(), // Data range

9,// Old value

100);// New value

cout << "vec after replace: ";

for (int& i : vec) cout << i << " ";

cout << endl;

replace\_if(vec.begin(), vec.end(),// Data range

[](int x){ return x > 80; },// Value condition

999);// New value

cout << "vec after replace\_if: ";

for (int& i : vec) cout << i << " ";

cout << endl;

replace\_copy(vec.begin(), vec.end(),// Source

vec3.begin(),// Destination

999,// Value condition

0); // New value

cout << "vec3 after replace\_copy: ";

for (int& i : vec3) cout << i << " ";

cout << endl;

}

#pragma endregion Replace

cout << endl;

#pragma region Remove

{

vector<int> vec = { 9, 9, 70, 8, 9, 87, 90 };// 7 items

vector<int> vec2 = { 9, 60, 70, 8, 45, 87, 90 };// 7 items

vector<int> vec3 = { 0, 0, 0, 0, 0, 0, 0 }; // 7 items

cout << "################### REMOVE ###################" << endl;

cout << "vec before remove: ";

for (int i : vec) cout << i << " ";

cout << endl;

// removes all 70!!!

remove(vec.begin(), vec.end(), 70);

cout << "vec after remove: ";

for (int i : vec) cout << i << " ";

cout << endl;

// We have, remove\_if with condition

// and remove\_copy, to copy to other container after removing

// unique, removes consecutive equal elements.

unique(vec.begin(), vec.end());

cout << "vec after unique: ";

for (int i : vec) cout << i << " ";

cout << endl;

// We have unique\_copy to remove consecutive elements and then

// copy to other container.

}

#pragma endregion Remove

cout << endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Order Changing Algorithms

- reverse, rotate, permute, shuffle

// They change the order of elements in container, but not necessarily

the elements themselves.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma region Reverse

{

vector<int> vec = { 9, 9, 70, 8, 9, 87, 90 };// 7 items

vector<int> vec2 = { 9, 60, 70, 8, 45, 87, 90 };// 7 items

vector<int> vec3 = { 0, 0, 0, 0, 0, 0, 0 }; // 7 items

cout << "################### REVERSE ###################" << endl;

cout << "vec before reverse: ";

for (int i : vec) cout << i << " ";

cout << endl;

reverse(vec.begin(), vec.end());

cout << "vec after reverse: ";

for (int i : vec) cout << i << " ";

cout << endl;

reverse\_copy(vec.begin(), vec.end(), vec3.begin());

cout << "vec3 after reverse\_copy: ";

for (int i : vec3) cout << i << " ";

cout << endl;

}

#pragma endregion Reverse

cout << endl;

#pragma region Rotate

{

vector<int> vec = { 9, 9, 70, 8, 9, 87, 90 };// 7 items

vector<int> vec2 = { 9, 60, 70, 8, 45, 87, 90 };// 7 items

vector<int> vec3 = { 0, 0, 0, 0, 0, 0, 0 };// 7 items

cout << "################### ROTATE ###################" << endl;

cout << "vec before rotate: ";

for (int i : vec) cout << i << " ";

cout << endl;

rotate(vec.begin(), vec.begin()+3, vec.end());

cout << "vec after rotate: ";

for (int i : vec) cout << i << " ";

cout << endl;

rotate\_copy(vec.begin(), vec.begin()+3, vec.end(), vec3.begin());

cout << "vec3 after rotate\_copy: ";

for (int i : vec3) cout << i << " ";

cout << endl;

}

#pragma endregion Rotate

cout << endl;

#pragma region Permute

{

vector<int> vec = { 9, 9, 70, 8, 9, 87, 90 };// 7 items

cout << "################### PERMUTE ###################" << endl;

cout << "vec before next\_permutation: ";

for (int i : vec) cout << i << " ";

cout << endl;

next\_permutation(vec.begin(), vec.end());

cout << "vec after next\_permutation: ";

for (int i : vec) cout << i << " ";

cout << endl;

cout << "vec before prev\_permutation: ";

for (int i : vec) cout << i << " ";

cout << endl;

prev\_permutation(vec.begin(), vec.end());

cout << "vec after prev\_permutation: ";

for (int i : vec) cout << i << " ";

cout << endl;

}

#pragma endregion Permute

cout << endl;

#pragma region Shuffel

{

vector<int> vec = { 9, 9, 70, 8, 9, 87, 90 };// 7 items

cout << "################### SHUFFEL ###################" << endl;

// Rearrange the data randomly

cout << "vec before random\_shuffle: ";

for (int i : vec) cout << i << " ";

cout << endl;

random\_shuffle(vec.begin(), vec.end());

cout << "vec after random\_shuffle: ";

for (int i : vec) cout << i << " ";

cout << endl;

}

#pragma endregion Shuffel

cout << endl;

return 0;

}

# 10 STL Algorithms 3 – Sorting

#include <iostream>

#include <algorithm>

#include <vector>

#include <string>

#include <functional>

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Sorting

- Sorting algorithms requires random access iterators

- vector, deque, container array and native array

- List and unordered containers can't be sorted.

- Associative container doesn't need sort, because they are already

sorted.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool leastSignificantDigit(int x, int y) { return ((x % 10) < (y % 10)); }

bool lessThan10(int x) { return (x < 10); }

int main(void)

{

#pragma region Sort

{

vector<int> vec = { 9, 1, 10, 2, 45, 3, 90, 4, 9, 5, 8 };// 11 item;

cout << "################### SORT ###################" << endl;

cout << "vec before sort: ";

for (int& i : vec) cout << i << " ";

cout << endl;

sort(vec.begin(), vec.end());// Default comparison function if operator <

cout << "vec after sort: ";

for (int& i : vec) cout << i << " ";

cout << endl;

// If we want to sort according to the last digit, then

// we can declare a function to do that...

sort(vec.begin(), vec.end(), leastSignificantDigit);

cout << "vec after leastSignificantDigit: ";

for (int& i : vec) cout << i << " ";

cout << endl;

}

#pragma endregion Sort

cout << endl;

#pragma region Partial Sort

{

vector<int> vec = { 9, 60, 70, 8, 45, 87, 90, 69, 69, 55, 7 };

cout << "################### PARTIAL SORT ###################" << endl;

cout << "vec before partial\_sort: ";

for (int& i : vec) cout << i << " ";

cout << endl;

// Find top 5 students according to their test scores

// greater<int>() - greater<int> is a functor, we need to default construct it

partial\_sort(vec.begin(), vec.begin() + 5, vec.end(), greater<int>());

cout << "vec after partial\_sort: ";

for (int& i : vec) cout << i << " ";

cout << endl;

partial\_sort(vec.begin(), vec.begin() + 5, vec.end());

cout << "vec after partial\_sort :Overloaded : ";

for (int& i : vec) cout << i << " ";

cout << endl;

}

#pragma endregion Partial Sort

cout << endl;

#pragma region nTh Element

{

vector<int> vec = { 9, 60, 70, 8, 45, 87, 90, 69, 69, 55, 7 };

cout << "################### nth ELEMENT ###################" << endl;

cout << "vec before nth\_element: ";

for (int& i : vec) cout << i << " ";

cout << endl;

// Find top 5 students according to their test scores, but we don't care

// their order.

// greater<int>() - greater<int> is a functor, we need to default construct it

nth\_element(vec.begin(), vec.begin() + 5, vec.end(), greater<int>());

cout << "vec after nth\_element: ";

for (int& i : vec) cout << i << " ";

cout << endl;

}

#pragma endregion nTh Element

cout << endl;

#pragma region Partition

{

vector<int> vec = { 9, 60, 70, 8, 45, 87, 90, 69, 69, 55, 7 };

vector<int> vec1 = { 9, 60, 70, 8, 45, 87, 90, 69, 69, 55, 7 };

cout << "################### PARTITION ###################" << endl;

cout << "vec before partition: ";

for (int i : vec) cout << i << " ";

cout << endl;

// We want to move the students whose score is less

// than 10 to the front

partition(vec.begin(), vec.end(), lessThan10);

cout << "vec after partition: ";

for (int i : vec) cout << i << " ";

cout << endl;

cout << "vec1 before stable\_partition: ";

for (int i : vec1) cout << i << " ";

cout << endl;

stable\_partition(vec1.begin(), vec1.end(), lessThan10);

cout << "vec1 after stable\_partition: ";

for (int i : vec1) cout << i << " ";

cout << endl;

}

#pragma endregion Partition

cout << endl;

#pragma region Heap Algorithm

//HEAP ALGORITHMS

{

vector<int> vec = { 9, 1, 10, 2, 45, 3, 90, 4, 9, 5, 8 };

cout << "################### HEAP ###################" << endl;

cout << "vec before make\_heap: ";

for (int i : vec) cout << i << " ";

cout << endl;

make\_heap(vec.begin(), vec.end());

cout << "vec after make\_heap: ";

for (int i : vec) cout << i << " ";

cout << endl;

// Remove largest element from the heap.

// pop\_heap,

// 1. Swap vec[0] with last item vec[size-1]

// 2. Heapify [vec.begin(), vec.end()-1)

pop\_heap(vec.begin(), vec.end());

cout << "vec after pop\_heap: ";

for (int i : vec) cout << i << " ";

cout << endl;

vec.pop\_back();

cout << "vec after vec.pop\_back(): ";

for (int i : vec) cout << i << " ";

cout << endl;

// Add new element

vec.push\_back(100);

push\_heap(vec.begin(), vec.end());

cout << "vec after vec.push\_back(100): ";

for (int i : vec) cout << i << " ";

cout << endl;

// Heap Sorting

vector<int> vec1 = { 9, 1, 10, 2, 45, 3, 90, 4, 9, 5, 8 };

cout << "vec1 before sort\_heap: ";

for (int i : vec1) cout << i << " ";

cout << endl;

make\_heap(vec1.begin(), vec1.end());

sort\_heap(vec1.begin(), vec1.end());

cout << "vec1 after sort\_heap: ";

for (int i : vec1) cout << i << " ";

cout << endl;

}

#pragma endregion Heap Algorithm

cout << endl;

return 0;

}

# 11 STL Algorithms 4 - Sorted Data n Numeric Algorithms

#include <iostream>

#include <algorithm>

#include <vector>

#include <string>

#include <numeric> // Numeric Algorithms

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Sorted Data Algorithms and Numeric Algorithms

 - Algorithms that require date being presorted.

 - Binary search, merge, set operations.

 - If we apply these algorithm on unsorted data, result is undefined.

 - Generalized form also exists for these algorithms.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(void)

{

/\*\*\*\*\*\*\*\*\*\* Sorted Data Algorithms \*\*\*\*\*\*\*\*\*\*/

#pragma region Binary Search

{

cout << "################### Binary Search ###################" << endl;

vector<int> vec = { 8, 9, 9, 9, 45, 87, 90 };

// Check if 9 is present in the vec

bool found = binary\_search(vec.begin(), vec.end(), 9);

if (found) cout << "9 is present in vec!!!" << endl;

// If we want to search multiple item, then we should

// use include()

vector<int> searchThese = { 9, 45, 87 }; // Must be Sorted!!!

found = includes(vec.begin(), vec.end(), // Range #1

searchThese.begin(), searchThese.end()); // Range #2

if (found)

cout << "Items in the searchThese vector are found in vec" << endl;

// lower\_bound, finds the first position where 66 can be

// inserted and still keep sorting. It returns iterator.

vector<int>::iterator itr = lower\_bound(vec.begin(), vec.end(), 66);

// upper\_bound, finds the last position where 9 could be

// inserted and still keep the sorting

itr = upper\_bound(vec.begin(), vec.end(), 9);

// equal\_range, returns both upper bound and lower bound.

pair<vector<int>::iterator, vector<int>::iterator> pairOfItrs;

pairOfItrs = equal\_range(vec.begin(), vec.end(), 9);

}

#pragma endregion Binary Search

cout << endl;

#pragma region Merge

{

cout << "################### Merge ###################" << endl;

vector<int> vec = { 8, 9, 9, 10 };

vector<int> vec2 = { 7, 9, 10 };

vector<int> vec\_out(10);

// merge(), merges two range of sorted data into one big range

// of sorted data. All the duplicates are kept.

merge(vec.begin(),vec.end(), // Range #1

vec2.begin(), vec2.end(), // Range #2

vec\_out.begin()); // Output

cout << "After merge(): ";

for (const int& i : vec\_out) cout << i << " ";

cout << endl;

// inplace\_merge(), takes one range of data as input. But that data

// has two range of sorted data. Example:

// |-Sorted-|  |---Sorted---|

vector<int> inplaceMerge = { 1, 2, 3, 4, 1, 2, 3, 4, 5 };

cout << "Before inplace\_merge(): ";

for (const int& i : inplaceMerge) cout << i << " ";

cout << endl;

inplace\_merge(inplaceMerge.begin(), inplaceMerge.begin() + 4, inplaceMerge.end());

cout << "After inplace\_merge(): ";

for (const int& i : inplaceMerge) cout << i << " ";

cout << endl;

}

#pragma endregion Merge

cout << endl;

#pragma region Set Operations

{

cout << "################### Set Operations ###################" << endl;

vector<int> vec = { 8, 9, 9, 10 }; // Sorted

vector<int> vec2 = { 7, 9, 10 }; // Sorted

vector<int> vec\_out(10);

// set\_union(), takes two range of sorted data and does union on them

// and stores the result in another vector.

// If item X is there in both vec and vec2, only one X will be in

// output vector!!!

set\_union(vec.begin(), vec.end(), // Input Range #1

vec2.begin(), vec2.end(), // Input Range #2

vec\_out.begin()); // Output

cout << "After set\_union(): ";

for (const int& i : vec\_out) cout << i << " ";

cout << endl;

vector<int> setInter(10);

// Intersection of 2 vectors

set\_intersection(vec.begin(), vec.end(),// Input Range #1

vec2.begin(), vec2.end(), // Input Range #2

setInter.begin()); // Output

cout << "After set\_intersection(): ";

for (const int& i : setInter) cout << i << " ";

cout << endl;

// set\_difference, items in vec but not in vec2

// here 9 and 10 are in vec and also in vec2 but 8 and 9 are in

// vec but not in vec2.

vector<int> setDiff(10);

set\_difference(vec.begin(), vec.end(),// Input Range #1

vec2.begin(), vec2.end(), // Input Range #2

setDiff.begin()); // Output

cout << "After set\_difference(): ";

for (const int& i : setDiff) cout << i << " ";

cout << endl;

// set\_symmetric\_difference, items which are not present in

// each other vectors!!!

vector<int> setSymDiff(10);

set\_symmetric\_difference(vec.begin(), vec.end(),// Input Range #1

vec2.begin(), vec2.end()// Input Range #2

setSymDiff.begin());// Output

cout << "After set\_symmetric\_difference(): ";

for (const int& i : setSymDiff) cout << i << " ";

cout << endl;

}

#pragma endregion Set Operations

cout << endl;

/\*\*\*\*\*\*\*\*\*\* Numeric Algorithms \*\*\*\*\*\*\*\*\*\*/

// Accumulate, inner product, partial sum, adjacent difference

#pragma region Accumulate

{

cout << "################### Accumulate ###################" << endl;

vector<int> vec = { 8, 9, 9, 10 };

// 10 + vec[0] + vec[1] + vec[2] + ...

int x = accumulate(vec.begin(), vec.end(), 10);

cout << "Accumulated value + 10 : " << x << endl;

// 10 \* vec[0] \* vec[1] \* vec[2] \* ...

int y = accumulate(vec.begin(), vec.end(), 10, multiplies<int>());

cout << "Accumulated value \* 10 with multiplies: " << y << endl;

}

#pragma endregion Accumulate

cout << endl;

#pragma region Inner Product

{

cout << "################### Inner Product ###################" << endl;

vector<int> vec = { 9, 60, 70, 8, 45, 87, 90 };

// 10 + vec[0]\*vec[4] + vec[2]\*vec[5] + vec[3]\*vec[6] + ...

int x = inner\_product(vec.begin(), vec.begin()+3,// Range #1

vec.end()-3, // Range #2

10); // Initial Value

cout << "inner\_product value + 10 : " << x << endl;

}

#pragma endregion Inner Product

cout << endl;

#pragma region Partial Sum

{

cout << "################### Partial Sum ###################" << endl;

vector<int> vec = { 9, 60, 70, 8, 45, 87, 90 };

vector<int> vec2(10);

cout << "Vector vec:";

for (const int& i : vec) cout << i << " ";

cout << endl;

partial\_sum(vec.begin(), vec.end(),

vec2.begin());

for (const int& i : vec2) cout << i << " ";

cout << endl;

}

#pragma endregion Partial Sum

cout << endl;

#pragma region Adjacent Difference

{

cout << "################### Adjacent Difference ###################" << endl;

vector<int> vec = { 9, 60, 70, 8, 45, 87, 90 };

vector<int> vec2(10);

cout << "Vector vec:";

for (const int& i : vec) cout << i << " ";

cout << endl;

adjacent\_difference(vec.begin(), vec.end(),

vec2.begin());

for (const int& i : vec2) cout << i << " ";

cout << endl;

}

#pragma endregion Adjacent Difference

cout << endl;

return 0;

}

# 12 C++ String 1 - Constructor and Size

#include <iostream>

#include <string>

using namespace std;

int main(void)

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Constructors \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma region Constructor

{

// String can be constructed from C string as shown below

string s1("Hello");

cout << "s1: " << s1 << endl;

// Here number specifies the size.

string s2("Hello", 3);

cout << "s2: " << s2 << endl;

// Copy constructor

// Here, the number is the starting position of s1 from

// where we want to construct s3

string s3(s1, 2);

cout << "s3: " << s3 << endl;

// s4 is constructed form s2 starting from 2nd position

// and has the size 2

string s4(s1, 2, 2);

cout << "s4: " << s4 << endl;

// s5 contains 5 chars of 'a'

string s5(5, 'a');

cout << "s5: " << s5 << endl;

// Also can be constructed from initializer list

string s6({ 'a', 'b', 'c' });

cout << "s6: " << s6 << endl;

}

#pragma endregion Constructor

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Size \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma region Size

{

string s1 = "Goodbye";

// size() and length() are synonymous functions

cout << "Size of s1: " << s1.size() << endl;

cout << "Size of s1: " << s1.length() << endl;

// capacity(), returns the storage space currently

// allocated to s1

cout << "Storage space allocated to s1 :" << s1.capacity() << endl;

//capacity can be changed using reserve()

s1.reserve(100);

cout << "Storage space allocated to s1 :" << s1.capacity() << endl;

// reserve() function doesn't change the content of s1. It doesn't

// change the size of s1. It only works on the capacity!!!

s1.reserve(5);

cout << "Storage space allocated to s1 :" << s1.capacity() << endl;

cout << "s1 is: " << s1 << endl;

// Shrinks the capacity to hold the content

s1.shrink\_to\_fit();

cout << "shrink\_to\_fit: Size of s1 :" << s1.size() << endl;

cout << "s1 is: " << s1 << endl;

// To reduce the size of s1

s1.resize(9);

cout << "resize(9): Size of s1 :" << s1.size() << endl;

s1.resize(9,'x');

cout << "s1 is: " << s1 << endl;

// This will print Goo

s1.resize(3);

cout << "resize(3): s1 is: " << s1 << endl;

}

#pragma endregion Size

return 0;

}

# 13 C++ String 2 - Accessing String Characters

#include <iostream>

#include <string>

using namespace std;

int main(void)

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Single Element Access \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma region Single Element Access

{

cout << "############### Single Element Access ###############" << endl;

string s1 = "Goodbye";

// s[2] will return the reference to the 2nd element of s1

cout << "2nd Char is :" << s1[2] << endl;

s1[2] = 'x';

cout << "Now s1 is: " << s1 << endl; // Goxdbye

s1.at(2) = 'y';

cout << "Now s1 is: " << s1 << endl; // Goydbye

//s1[20] = 'a'; //Undefined behavior

//s1.at(20) will throw exception of out\_of\_range

// s1.at() throws exception across all the random access

// containers if we try to access out side the bound.

{

// string provides below APIs to be consistent with other

// containers

// Like vector, string doesn't have push\_front() and pop\_front()

cout << "front(): First char is: " << s1.front() << endl;

cout << "back(): Last char is: " << s1.back() << endl;

s1.push\_back('z');

cout << "push\_back('z'): Now s1 is: " << s1 << endl;

s1.pop\_back();

cout << "pop\_back(): Now s1 is: " << s1 << endl;

string::iterator begSting = s1.begin();

string::iterator endString = s1.end();

for (string::iterator it = begSting; it != endString; it++)

cout << \*it << " ";

cout << endl;

string s3(s1.begin(), s1.begin() + 3);

cout << "s3 constructed using iterators: " << s3 << endl;

}

}

#pragma endregion Single Element Access

cout << endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Ranged Access \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma region Ranged Access

// assign, append, insert and replace

{

cout << "############### Ranged Access ###############" << endl;

string s2 = "Dragon Land";

string s1;

s1.assign(s2); // s1 = s2;

cout << "s1.assign: s1 is :" << s1 << endl;

// Starting from 2nd position and size is 4

// When the first parameter is string, 2nd parameter is starting position

s1.assign(s2, 2, 4);

cout << "s1.assign: s1 is :" << s1 << endl;

s1.assign("Wisdom");

cout << "s1.assign: s1 is :" << s1 << endl;

// When the 1st parameter is C-string, 2nd parameter is size

s1.assign("Wisdom", 3);

cout << "s1.assign: s1 is :" << s1 << endl;

//s1.assign(s1, 3); //ERROR!!!

s1.assign(3, 'x');

cout << "s1.assign: s1 is :" << s1 << endl;

s1.assign({ 'a', 'b', 'c' });

cout << "s1.assign: s1 is :" << s1 << endl;

// append, insert and replace

s1.append(" def");

cout << "s1.append: s1 is :" << s1 << endl;

// insert 4 chars of mountain into 2nd position of s1

s1.insert(2, "mountain", 4);

cout << "s1.insert: s1 is :" << s1 << endl;

s1.replace(2, 5, s2, 3, 3);

cout << "s1.replace: s1 is :" << s1 << endl;

// Pattern is 1 number is position

// 2nd number is size

s1.erase(1, 4);

cout << "s1.erase: s1 is :" << s1 << endl;

s1 = s2.substr(2, 4);

cout << "s1.substr: s1 is :" << s1 << endl;

s1 = "abc";

s1.c\_str(); // This will return C-String "abc\0"

s1.data(); // This will return C-String "abc\0"

s1.swap(s2);

cout << "s1.swap: s1 is :" << s1 << endl;

cout << "s1.swap: s2 is :" << s2 << endl;

}

#pragma endregion Ranged Access

cout << endl;

return 0;

}

# 14 C++ String 3 - Member Function Algorithms

#include <iostream>

#include <string>

using namespace std;

int main(void)

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Member Function Algorithms \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// copy(), find() and compare()

{

string s1 = "abcdefg";

char buf[20];

// Will copy 3 chars from s1 to buf and the returned value is the

// number of chars that are indeed copied.

size\_t len = s1.copy(buf, 3);

buf[len] = '\0';

cout << "buf is: "<< buf << endl;

cout << "Number of chars copied = " << len << endl;

// here copy is the only function with exception where the 1st

// number is the size and the 2nd number is the position

len = s1.copy(buf, 4, 2);

buf[len] = '\0';

cout << "buf is: " << buf << endl;

cout << "Number of chars copied = " << len << endl;

s1 = "If a job is worth doing, it's worth doing well";

size\_t found = s1.find("doing");

cout << "Position of 'doing' in s1 is :" << found << endl;

// This find will start searching from 20th position onwards

found = s1.find("doing", 20);

cout << "Position of 'doing' in s1 is :" << found << endl;

found = s1.find("doing well", 0);

cout << "Position of 'doing' in s1 is :" << found << endl;

found = s1.find("doing well", 0, 5);

cout << "Position of 'doing' in s1 is :" << found << endl;

found = s1.find\_first\_of("doing");

cout << "One of the chars from 'doing' appeared at: " << found << endl;

found = s1.find\_first\_of("doing", 10);

cout << "One of the chars from 'doing' appeared at: " << found << endl;

found = s1.find\_first\_of("doing", 10, 1);

cout << "One of the chars from 'doing' appeared at: " << found << endl;

found = s1.find\_last\_of("doing");

cout << "find\_last\_of: " << found << endl;

found = s1.find\_first\_not\_of("doing");

cout << "find\_first\_not\_of: " << found << endl;

found = s1.find\_last\_not\_of("doing");

cout << "find\_last\_not\_of: " << found << endl;

// Compare

// Returns +ve if (s1 > s2)

// -ve if (s1 < s2)

// 0 if (s1 == s2)

string s2(s1);

if (s1 == s2)

cout << "Both s1 and s2 are equal!!!" << endl;

s2 = "zzz";

if (s1 < s2)

cout << "s1 is less than s2" << endl;

s2 = "aaa";

if (s1 > s2)

cout << "s1 is greater than s2" << endl;

// Start the comparison at position 3 and compare 2 chars

cout << "s1.compare(3, 2, s2): " << s1.compare(3, 2, s2) << endl;

string ss = s1 + s2;

cout << "ss = s1 + s2; : " << ss << endl;

}

cout << endl;

return 0;

}

# 15 C++ String 4 - Non-member Functions

#include <iostream>

#include <string>

using namespace std;

int main(void)

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Non-member Function \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

string s1("Hello world");

cout << s1 << endl;

//cout << "Enter a string, press ENTER key to terminate: ";

//cin >> s1; // Reads until carriage return, which is the default delimiter

//cout << "Entered string is : " << s1 << endl;

//cout << "Enter a string, press ENTER key to terminate: ";

//getline(cin, s1); // Reads until carriage return, which is the default delimiter

//cout << "Entered string is : " << s1 << endl;

//cout << "Enter a string, press ; to terminate: ";

//getline(cin, s1, ';');

//cout << "Entered string is : " << s1 << endl;

s1 = to\_string(8);

cout << "to\_string(8): " << s1 << endl;

s1 = to\_string(2.3e7);

cout << "to\_string(2.3e7): " << s1 << endl;

s1 = to\_string(0XA4);

cout << "to\_string(0XA4): " << s1 << endl;

s1 = to\_string(034);

cout << "to\_string(034): " << s1 << endl;

// String into a number

s1 = "190";

int i = stoi(s1);

cout << "Int i = " << i << endl;

s1 = "190 monkeys";

size\_t position;

// i will be 190

// position will be the location number where the stoi() has stopped parsing

i = stoi(s1, &position);

cout << "Position where stoi() has stopped: " << position << endl;

s1 = "a monkey";

//i = stoi(s1, &position); // This will throw invalid\_argument

/\*16 for hexa decimal number\*/

i = stoi(s1, &position, 16);

cout << "i = stoi(s1, &position, 16): " << i << endl;

// We also have

// stol - string to long

// stod - string to double

// stof - string to float

return 0;

}

# 16 C++ String 5 - String and Algorithms

#include <iostream>

#include <string>

#include <algorithm>

using namespace std;

int main(void)

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* String and Algorithm \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// We can apply STL algorithms to string.

string s1 = "Variety is the spice of life";

int num = count(s1.begin(), s1.end(), 'e');

cout << "Number of e's in '" << s1 << "' is " << num << endl;

// All the letter of abcde...

num = count\_if(s1.begin(), s1.end(),

[](char ch) { return ((ch <= 'e') && (ch >= 'a')); });

cout << "Number of a's + b's + c's + d's + e's :" << num << endl;

s1 = "Goodness is better than beauty";

// Search consecutive 2 s

string::iterator itr = search\_n(s1.begin(), s1.begin() + 20,

2, 's');

s1.erase(itr, itr + 5);

cout << "s1.erase(itr, itr + 5) :" << s1 << endl;

s1.insert(itr, 3, 'z');

cout << "s1.insert(itr, 3, 'z') :" << s1 << endl;

s1.replace(itr, itr + 3, 3, 'y'); // Replace the substring

// Member function replace

cout << "s1.replace(itr, itr + 3, 3, 'y') :" << s1 << endl;

string s2 = s1;

is\_permutation(s1.begin(), s1.end(), s2.begin());

// Algorithm function replace

replace(s1.begin(), s1.end(), 'e', ' '); // Replace characters

cout << "s1.replace(s1.begin(), s1.end(), 'e', ' ') :" << s1 << endl;

transform(s1.begin(),s1.end(),

s2.begin(),

[](char ch) {

if (ch < 'n') return 'a';

else return 'z';

});

cout << "After transform s2 is: " << s2 << endl;

s1 = "abcdefg";

rotate(s1.begin(), s1.begin() + 3, s1.end());

cout << "After rotate s1: " << s1 << endl;

u16string s3; // string char16\_t

u32string s4; // string char32\_t

wstring s5; // string wchar\_t (wide character)

return 0;

}

# 17 C++ Stream 1 – Introductory

#include <iostream>

#include <fstream>

#include <algorithm>

#include <bitset>

#include <complex>

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Stream

 - Stream is a C++ input/output library.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(void)

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

- cout: is a global object of ostream which lives in std namespace

- ostream is typedef basic\_ostream<char> ostream

- << : is the member function of ostream

Signature of :

ostream&& ostream::operator<< (string v);

- endl; assume '\n' + flush operation

What is stream?

- Serial IO interface to external devices (file, stdin/stdout, network, etc...)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

cout << "Hello" << endl;

// What do you mean by serial in the stream definition?

string s("hello");

s[3] = 't'; // This is random access!!!

//cout[3] = 't'; // ERROR!!!

{

// This will open a file to write. If the file doesn't exists, it will

// create a new file!!!.

ofstream of("MyLog.txt");

// << is overloaded for all the fundamental data types

of << "Experience is the mother of wisdom" << endl;

of << 224 << endl;

// << is also overloaded for some of STL data types

of << bitset<8>(14) << endl;

of << complex<int>(2, 3) << endl;

} // Here the file is closed. RAII

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Every IO operation

1. Formatting the data

2. Communication the data with external devices

stream class provides a common API's to formate the data. - Low coupling

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

return 0;

}

# 18 C++ Stream 2 - File Stream and Error Handling

#include <iostream>

#include <fstream>

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

File Stream and Error Handling

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(void)

{

{

// This will open file for writing also this will clear

// the content of the file!!!.

//ofstream of("MyLog.txt");

// Move the output pointer to the end of the file

ofstream of("MyLog.txt",ofstream::app);

of << "Honesty is the best policy" << endl;

}

{

ofstream of("MyLog.txt",ofstream::in | ofstream::out );

// Moves the output pointer 10 chars after begin

of.seekp(10, ios::beg);

of << "12345"; // Overwrite 5 chars

// Moves the output pointer 5 chars before end

of.seekp(-5, ios::end);

of << "Nothing ventured, nothing gained" << endl;

// Moves the output pointer 5 chars before current position

of.seekp(-5, ios::cur);

of << "Hello" << endl;

}

{

ifstream inf("MyLog.txt");

int i;

// This will read 1 word, parse it as integer and save it into i.

// So below operation will fail since the first data is character!!!

inf >> i;

/\* For stream, error status is indicated with 4 bits

Error Status:

1. goodbit

2. badbit.

3. failbit.

4. eofbit.

\*/

if (inf.good()) // goodbit ==1

cout << "Everything is OK" << endl;

if (inf.bad()) // badbit == 1

cout << "Non-recoverable error has happened" << endl;

if (inf.fail()) // failbit == 1 and badbit == 1

cout << "Failed stream operation" << endl;// Recoverable

if (inf.eof()) // eofbit == 1

cout << "End of file" << endl;

// This will clear all the error status

// Sets the goodbit to 1 and sets everything to 0

inf.clear();

// Sets a new value to the error flag

// sets badbit to 1, and sets everything to 0

inf.clear(ios::badbit);

// This will read the current status flag

inf.rdstate();

// This will clear only the failbit and everything else

// stay the same

inf.clear(inf.rdstate() & ~ios::failbit);

if (inf) // Equivalent to : if(!inf.fail())

cout << "Read successfully" << endl;

if (inf>>i)

cout << "Read successfully" << endl;

/\* Handle errors with exceptions \*/

// Setting the exception mask. When badbit or failbit are set to 1,

// an exception is thrown!!! When eofbit is set to 1, no exception is

// thrown

//inf.clear();

inf.exceptions(ios::badbit | ios::failbit);

inf.exceptions(ios::goodbit); // No exception

}

return 0;

}

# 19 C++ Stream 3 - Formatted and Unformatted IO

#include <iostream>

#include <fstream>

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Formatted and Unformatted IO

- Main purpose of stream class is to format the data

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(void)

{

// Formatting

cout << 34 << endl; // Writes 34 to stdout

cout.setf(ios::oct, ios::basefield);

cout << 34 << endl;

cout.setf(ios::showbase);

cout << 34 << endl;

cout.setf(ios::hex, ios::basefield);

cout << 34 << endl;

cout.unsetf(ios::showbase);

cout << 34 << endl;

cout.setf(ios::dec, ios::basefield);

cout << 34 << endl;

// We can also change the way the output is shown

cout.width(10);

cout << 26 << endl;

cout.setf(ios::left, ios::adjustfield);

cout << 26 << endl;

// Floating point value

cout.setf(ios::scientific, ios::floatfield);

cout << 340.1 << endl;

cout.setf(ios::fixed, ios::floatfield);

cout << 340.1 << endl;

cout.precision(3);

cout << 340.1 << endl;

// Input formatting

int i;

cin.setf(ios::hex, ios::basefield);

//cin >> i;

ios::fmtflags f = cout.flags();

cout.flags(ios::oct | ios::showbase);

// Unformatted IO

// Input

ifstream inf("MyLog.txt");

char buf[80];

// Reads up to 80 chars and save in buf

inf.get(buf, 80);

// Reads up to 80 chars or until \n

inf.getline(buf, 80);

// read 20 chars

inf.read(buf, 20);

// Ignore the next 3 chars in the stream

inf.ignore(3);

// Returns the char on top of stream

inf.peek();

// Returns a char back to the stream

inf.unget();

inf.get();

// Returns the number of chars read by last unformatted read

inf.gcount();

// Output

ofstream of("MyLog.txt");

of.put('c');

of.write(buf, 6);

of.flush();

return 0;

}

# 20 C++ Stream 4 – Manipulators

#include <iostream>

#include <iomanip>

using namespace std;

ostream& endl(ostream& sm) // Manipulator!!!

{

sm.put('\n');

sm.flush();

return sm;

}

ostream& ostream::operator<<(ostream& (\*func)(ostream&))

{

return (\*func)(\*this);

}

int main()

{

// endl is new line + flush

cout << "Hello" << endl;

// But what is endl? is it a object? or Built-in data type?

// or is it function?

// endl is a function!!! endl takes ostream reference and returns

// ostream reference

// Then what is << operator?

// Now << is taking function(endl) as parameter

cout << ends; // Insert a '\0'

cout << flush;

//cin >> ws; // Read and discard white spaces!!!

// It will print 8 chars and 99 is aligned to left and fill with underscore

cout << setw(8) << left << setfill('\_') << 99 << endl;

cout << hex << showbase << 14 << endl;

return 0;

}

# 21 C++ Stream 5 - Stream Buffer

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Stream Buffer

- Typically IO operations has 2 steps...

1. Formatting the data. -> It is the responsibility of STREAM class

2. Communicating data to external devices. -> Done by STREAM buffer!!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(void)

{

// If we do cout << 34, It is the responsibility of cout to format

// the data appropriately and then delegate the job of communicating

// the data to external device to its internal module, which is STRAM

// buffer!!!

cout << 34;

//cout.rdbuf() -> This will give the pointer to internal module of STRAM buffer.

streambuf\* pbuf = cout.rdbuf();

// This will create another ostream which has the same STREAM buffer

// as the cout!!! In other words, myCOUT has the same transferring

// channel as that of cout!!!

ostream myCOUT(pbuf);

myCOUT << 34 << endl; // This will print 34 to standard out!!!

// Changing the format of myCOUT

myCOUT.setf(ios::showpos); // Show the +ve sign of a number

myCOUT.width(20); // Width = 20, alignment = right

myCOUT << 12 << endl; //           +12

cout << 12 << endl; //12

// Store the STREAM buffer in a temp variable

streambuf\* originalBuf = cout.rdbuf();

ofstream of("MyLog.txt");

// of.rdbuf() -> will give the pointer to buffer

// Below we are assigning other stream buffer to cout!!!

cout.rdbuf(of.rdbuf()); // Now cout will write to file!!!

cout << "Hello" << endl; // Will write Hello to file

// Restore the original stream buffer of cout

cout.rdbuf(originalBuf);

cout << "Goodbye!!!" << endl; // Will write Goodbye to screen

/\* STREAM buffer iterator \*/

istreambuf\_iterator<char> i(cin);

ostreambuf\_iterator<char> o(cout);

while (\*i != 'x')

{

\*o = \*i;

o++;

i++;

}

copy(istreambuf\_iterator<char>(cin), istreambuf\_iterator<char>(), ostreambuf\_iterator<char>(cout));

return 0;

}

# 22 C++ Stream 6 - String Stream

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

String Stream

- It doesn't do IO operation.

- It read/write operation on string.

- It treas the string as file. But Why????

The main purpose of this is to reuse the formatting functions of

stream to process the string. CODE REUSE.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(void)

{

stringstream ss; // This doesn't do IO operation

ss << 89 << " Hex: " << hex << 89 << " Oct: " << oct << 89;

cout << ss.str() << endl;

int a, b, c;

string s1;

// Formatted input. Works token by token.

// Tokens are space, tabs, newlines.

ss >> hex >> a;

cout << a << endl;

ss >> s1;

cout << s1 << endl;

ss >> dec >> b;

cout << b << endl;

// Ignore the next 6 chars in the stream.

ss.ignore(6);

ss >> oct >> c;

cout << c << endl;

// ostringstream -- formatted output

// istringstream -- formatted input

return 0;

}

# 23 C++ Stream 7 - Enable Streaming for Your Own Class

#include <iostream>

#include <string>

#include <ostream>

#include <istream>

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enable Streaming for Your Own Class

- Creating stream enabled class

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct Dog

{

int age\_;

string name\_;

};

ostream& operator<<(ostream& sm, const Dog& d)

{

sm << "My name is " << d.name\_ << " and my age is " << d.age\_ << endl;

return sm;

}

istream& operator>>(istream& sm, Dog& d)

{

sm >> d.age\_;

sm >> d.name\_;

return sm;

}

int main(void)

{

Dog d{ 2, "Bob" };

cout << d;

cin >> d;

cout << d;

return 0;

}