Standard Template Library

Last updated 5/1/19

STL

- Motivation
  - The Standard Template Library contains a number of useful templates
    - Data Structures
    - Algorithms

STL

- Data Structures

  - Containers
    - Class that organizes information
    - Sequence – organizes data in a sequence
    - Associative – organizes data via “keys”
    - Currently 3 : vector, deque, list

  - Iterators
    - Abstraction of the concept of a pointer
    - Used to access data in the container
    - Associated with a specific container

STL

- Algorithms

  - Function Templates
    - Perform various function on containers

    - binary_search
    - count
    - find
    - …

STL

- Iterator usage

```
/*
 * vector_no_iter.cpp
 *
 *  Created on: May 1, 2019
 *      Author: johnsontimoj
 */
// show vector without using iterator

#include <vector>

#include <iostream>
using namespace std;

int main(void){
  int cnt;

  // create vector
  vector<int> my_vector;

  // load vector
  for(cnt=0; cnt<10; cnt++)
    my_vector.push_back(cnt);

  // print vector
  cout << "Vector size is: " << my_vector.size() << endl;
  for(cnt=0; cnt<10; cnt++)
    cout << my_vector[cnt] << " ";
  cout << endl;

  return 0;
}
```

```
Vector size is: 10
0 1 2 3 4 5 6 7 8 9
```

```
/*
 * vector_iter.cpp
 *
 *  Created on: May 1, 2019
 *      Author: johnsontimoj
 */
// show vector using iterator

#include <vector>

#include <iostream>
using namespace std;

int main(void){
  int cnt;

  // create vector
  vector<int> my_vector;

  // create iterator
  vector<int>::iterator itr;

  // load vector
  for(cnt=0; cnt<10; cnt++)
    my_vector.push_back(cnt);

  // print vector
  cout << "Vector size is: " << my_vector.size() << endl;
  for(itr = my_vector.begin(); itr < my_vector.end(); itr++)
    cout << *itr << " ";
  cout << endl;

  // print vector backwards
  cout << "Vector size is: " << my_vector.size() << endl;
  for(itr = my_vector.end()-1; itr >= my_vector.begin(); itr--)
    cout << *itr << " ";
  cout << endl;

  return 0;
}
```

begin() and end() functions return iterators to the first and (1 past the last) - element of the vector (container)

since itr is like a pointer - must dereference it's value to get the data

```
Vector size is: 10
0 1 2 3 4 5 6 7 8 9
Vector size is: 10
9 8 7 6 5 4 3 2 1 0
```

STL

- Vector member functions

Member functions
(constructor)
Construct vector (public member function)
(destructor)
Vector destructor (public member function)
operator=
Assign content (public member function)

Iterators:
begin
Return iterator to beginning (public member function)

end
Return iterator to end (public member function)

rbegin
Return reverse iterator to reverse beginning (public member function)

rend
Return reverse iterator to reverse end (public member function)

cbegin
Return const_iterator to beginning (public member function)

cend
Return const_iterator to end (public member function)

crbegin
Return const_reverse_iterator to reverse beginning (public member function)

crend
Return const_reverse_iterator to reverse end (public member function)

Capacity:
size
Return size (public member function)

max_size
Return maximum size (public member function)

resize
Change size (public member function)

capacity
Return size of allocated storage capacity (public member function)

empty
Test whether vector is empty (public member function)

reserve
Request a change in capacity (public member function)

shrink_to_fit
Shrink to fit (public member function)

Element access:
operator[]
Access element (public member function)

at
Access element (public member function)

front
Access first element (public member function)

back
Access last element (public member function)

data
Access data (public member function)

Modifiers:
assign
Assign vector content (public member function)

push_back
Add element at the end (public member function)

pop_back
Delete last element (public member function)

insert
Insert elements (public member function)

erase
Erase elements (public member function)

swap
Swap content (public member function)

clear
Clear content (public member function)

emplace
Construct and insert element (public member function)

emplace_back
Construct and insert element at the end (public member function)

Allocator:
get_allocator
Get allocator (public member function)

Non-member function overloads

relational operators
Relational operators for vector (function template)

swap
Exchange contents of vectors (function template)

STL

- Algorithms

Non-modifying sequence operations:
all_of
Test condition on all elements in range (function template)

any_of
Test if any element in range fulfills condition (function template)

none_of
Test if no elements fulfill condition (function template)

for_each
Apply function to range (function template)

find
Find value in range (function template)

find_if
Find element in range (function template)

find_if_not
Find element in range (negative condition) (function template)

find_end
Find last subsequence in range (function template)

find_first_of
Find element from set in range (function template)

adjacent_find
Find equal adjacent elements in range (function template)

count
Count appearances of value in range (function template)

count_if
Return number of elements in range satisfying condition (function template)

mismatch
Return first position where two ranges differ (function template)

equal
Test whether the elements in two ranges are equal (function template)

is_permutation
Test whether range is permutation of another (function template)

search
Search range for subsequence (function template)

search_n
Search range for elements (function template)

Modifying sequence operations:
copy
Copy range of elements (function template)

copy_n
Copy elements (function template)

copy_if
Copy certain elements of range (function template)

copy_backward
Copy range of elements backward (function template)

move
Move range of elements (function template)

move_backward
Move range of elements backward (function template)

swap
Exchange values of two objects (function template)

swap_ranges
Exchange values of two ranges (function template)

iter_swap
Exchange values of objects pointed to by two iterators (function template)

transform
Transform range (function template)

replace
Replace value in range (function template)

replace_if
Replace values in range (function template)

replace_copy
Copy range replacing value (function template)

replace_copy_if
Copy range replacing value (function template)

fill
Fill range with value (function template)

fill_n
Fill sequence with value (function template)

generate
Generate values for range with function (function template)

generate_n
Generate values for sequence with function (function template)

remove
Remove value from range (function template)

remove_if
Remove elements from range (function template)

remove_copy
Copy range removing value (function template)

remove_copy_if
Copy range removing values (function template)

STL

- Algorithms

unique
Remove consecutive duplicates in range (function template)

unique_copy
Copy range removing duplicates (function template)

reverse
Reverse range (function template)

reverse_copy
Copy range reversed (function template)

rotate
Rotate left the elements in range (function template)

rotate_copy
Copy range rotated left (function template)

random_shuffle
Randomly rearrange elements in range (function template)

shuffle
Randomly rearrange elements in range using generator (function template)

Partitions:
is_partitioned
Test whether range is partitioned (function template)

partition
Partition range in two (function template)

stable_partition
Partition range in two - stable ordering (function template)

partition_copy
Partition range into two (function template)

partition_point
Get partition point (function template)

Sorting:
sort
Sort elements in range (function template)

stable_sort
Sort elements preserving order of equivalents (function template)

partial_sort
Partially sort elements in range (function template)

partial_sort_copy
Copy and partially sort range (function template)

is_sorted
Check whether range is sorted (function template)

is_sorted_until
Find first unsorted element in range (function template)

nth_element
Sort element in range (function template)

Binary search (operating on partitioned/sorted ranges):
lower_bound
Return iterator to lower bound (function template)

upper_bound
Return iterator to upper bound (function template)

equal_range
Get subrange of equal elements (function template)

binary_search
Test if value exists in sorted sequence (function template)

Merge (operating on sorted ranges):
merge
Merge sorted ranges (function template)

inplace_merge
Merge consecutive sorted ranges (function template)

includes
Test whether sorted range includes another sorted range (function template)

set_union
Union of two sorted ranges (function template)

set_intersection
Intersection of two sorted ranges (function template)

set_difference
Difference of two sorted ranges (function template)

set_symmetric_difference
Symmetric difference of two sorted ranges (function template)

Heap:
push_heap
Push element into heap range (function template)

pop_heap
Pop element from heap range (function template)

make_heap
Make heap from range (function template)

sort_heap
Sort elements of heap (function template)

is_heap
Test if range is heap (function template)

is_heap_until
Find first element not in heap order (function template)

Min/max:
min
Return the smallest (function template)

max
Return the largest (function template)

minmax
Return smallest and largest elements (function template)

min_element
Return smallest element in range (function template)

max_element
Return largest element in range (function template)

minmax_element
Return smallest and largest elements in range (function template)

Other:
lexicographical_compare
Lexicographical less-than comparison (function template)

next_permutation
Transform range to next permutation (function template)

prev_permutation
Transform range to previous permutation (function template)

STL

- Algorit

```
Vector size is: 10
0 1 2 3 4 5 6 7 8 9
Vector size is: 10
8 1 9 2 0 5 7 3 4 6
Vector size is: 10
0 1 2 3 4 5 6 7 8 9
Found it
Not there
Vector size is: 10
6 4 9 7 3 0 1 8 5 2
The first 7 is at location: 3
```

```cpp
/*
 * algorithms.cpp
 *
 *  Created on: May 1, 2019
 *      Author: johnsontimoj
 */

#include <vector>
#include <algorithm>

#include <iostream>
using namespace std;

int main(void){
  int cnt;

  // create vector
  vector<int> my_vector;

  // create iterator
  vector<int>::iterator itr;

  // load vector
  for(cnt=0; cnt<10; cnt++)
  my_vector.push_back(cnt);

  // print vector
  cout << "Vector size is: " << my_vector.size() << endl;
  for(itr = my_vector.begin(); itr < my_vector.end(); itr++)
    cout << *itr << " ";
  cout << endl;

  // shuffle the vector
  random_shuffle(my_vector.begin(), my_vector.end());
```

```cpp
  // print vector
  cout << "Vector size is: " << my_vector.size() << endl;
  for(itr = my_vector.begin(); itr < my_vector.end(); itr++)
    cout << *itr << " ";
  cout << endl;

  // sort vector
  sort(my_vector.begin(), my_vector.end());

  // print vector
  cout << "Vector size is: " << my_vector.size() << endl;
  for(itr = my_vector.begin(); itr < my_vector.end(); itr++)
    cout << *itr << " ";
  cout << endl;

  // search for an element
  if(binary_search(my_vector.begin(), my_vector.end(), 4))
    cout << "Found it" << endl;
  else
    cout << "Not there" << endl;

  // search for an element
  if(binary_search(my_vector.begin()+5, my_vector.end(), 4))
    cout << "Found it" << endl;
  else
    cout << "Not there" << endl;

  // shuffle the vector
  random_shuffle(my_vector.begin(), my_vector.end());

  // print vector
  cout << "Vector size is: " << my_vector.size() << endl;
  for(itr = my_vector.begin(); itr < my_vector.end(); itr++)
    cout << *itr << " ";
  cout << endl;

  // find 7
  itr = find(my_vector.begin(), my_vector.end(), 7);
  cout << "The first 7 is at location: " << (itr - my_vector.begin()) << end

return 0;
}
```

STL

- STL – list
  - Doubly linked list

```cpp
/*
 * list_example.cpp
 *
 *  Created on: May 1, 2019
 *      Author: johnsontimoj
 */
// Create a list of Boxes
#include <list>
#include "box.h"

#include <iostream>
using namespace std;

void print_list(const list<Box> & the_list);

int main(void){
  // create list
  list<Box> mybox_list;

  // create iterator
  list<Box>::iterator itr;

  //create some boxes
  for(int i=0; i<10; i++)
    mybox_list.push_back(Box(i, i, i, i));

  // display box volumes
  // Note - no < operator for list iterator
  for(itr=mybox_list.begin(); itr!=mybox_list.end(); itr++)
    cout << (*itr).calcVolume() << " ";
  cout << endl;

  // reverse the list
  mybox_list.reverse();

  // display box volumes
  print_list(mybox_list);
```

```cpp
  // Insert a box
  Box new_box(11, 2.5, 2.5, 2.5);
  itr = mybox_list.begin();
  for(int i=0; i<5; i++)    // note: no + for list iterator
    itr++;
  mybox_list.insert(itr, new_box);

  // display box volumes
  print_list(mybox_list);

  // remove a box
  itr = mybox_list.begin();
  for(int i=0; i<3; i++)    // note: no + for list iterator
    itr++;
  mybox_list.erase(itr);

  // display box volumes
  print_list(mybox_list);

  return 0;
}

void print_list(const list<Box> & the_list){
  // display box volumes
  // Note - no < operator for list iterator
  // Note - when passing in a const - iterator must also be const
  list<Box>::const_iterator itr;
  for(itr=the_list.begin(); itr!=the_list.end(); itr++)
    cout << (*itr).calcVolume() << " ";
  cout << endl;

  return;
}
```

```
0 1 8 27 64 125 216 343 512 729
729 512 343 216 125 64 27 8 1 0
729 512 343 216 125 15.625 64 27 8 1 0
729 512 343 125 15.625 64 27 8 1 0
```

STL

- STL – unordered Map
  - Associative container
    - Contains key/data pairs

```cpp
/*
 * umap_example.cpp
 *
 *  Created on: May 1, 2019
 *      Author: johnsontimoj
 */
/////////////////////////////
//
// Unordered Map example
//
// student name / account bal
//
///////////////////////////////
#include <unordered_map>
#include <cstring>

#include <iostream>
using namespace std;

void print_map(const unordered_map<string, double> & the_map);

int main(void){
  // create map
  unordered_map<string, double> mymap;

  // create iterator
  unordered_map<string, double>::iterator itr;

  //create some entries
  mymap["joe"] = 125.30;
  mymap["sue"] = 215.20;
  mymap["al"] = 12.34;
  mymap["samantha"] = 3.55;
  mymap["j"] = 99.99;
```

```cpp
  // display the map
  // Note - no < operator for list iterator
  for(itr=mymap.begin(); itr!=mymap.end(); itr++)
    cout << itr->first << " : " << itr->second << "\t";
  cout << endl;

  // insert an entry
  mymap.insert(make_pair("jenna", 0.00));
  // print map
  print_map(mymap);

  // search
  string key = "al";
  if(mymap.find(key) == mymap.end())
    cout << key << " not found" << endl;
  else
    cout << key << " has a balance of $" << mymap[key] << endl;

  key = "susan";
  if(mymap.find(key) == mymap.end())
    cout << key << " was not found" << endl;
  else
    cout << key << " has a balance of $" << mymap[key] << endl;

  return 0;
}

void print_map(const unordered_map<string, double> & the_map){
  // display map values
  // Note - when passing in a const - iterator must also be const
  unordered_map<string, double>::const_iterator itr;
  for(auto i : the_map)
    cout << i.first << " : " << i.second << "\t";
  cout << endl;

  return;
}
```

```
j : 99.99    joe : 125.3    sue : 215.2    al : 12.34    samantha : 3.55
jenna : 0    j : 99.99j    oe : 125.3    sue : 215.2    al : 12.34    samantha : 3.55
al has a balance of $12.34
susan not found
```