The Standard Template Library

Kafi Rahman

Assistant Professor @CS
Truman State University

Algorithms for Set Operations

```
vector<int> foo{100, 200, 300};
vector<int> bar{1, 2, 3, 4, 5};

// when inserting in a sequence container, insertion
point advances
// because each std::insert_iterator::operator= updates
the target iterator
copy(foo.begin(), foo.end(), inserter(bar, bar.begin()));

//bar.insert(bar.begin(), foo.begin(), foo.end());

for (int n : bar) // display all the values from bar
    cout << n << ' ';
```

- inserter function uses the range of values from foo.begin() to foo.end() and adds them in the bar variable (starting from the begin() position)

Algorithms for Set Operations

STL Function Template	Description
set_union	Finds the union of two sets, which is a set that contains all the elements of both sets, excluding duplicates.
set_intersection	Finds the intersection of two sets, which is a set that contains only the elements that are found in both sets.
set_difference	Finds the difference of two sets, which is the set of elements that appear in one set, but not the other.
set_symmetric_difference	Finds the symmetric difference of two sets, which is the set of elements that appear in one set, but not both.
set_includes	Determines whether one set includes another.

- The STL provides function templates for basic mathematical set operations.

Algorithms for Set Operations: set_union

```
set<int> oneSet = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
set<int> twoSet = {5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};

set<int> resultSet;

// set_union: all the elements from both sets (no duplicates)
set_union(oneSet.begin(), oneSet.end(),             //first set
          twoSet.begin(), twoSet.end(),             //second set
          inserter(resultSet, resultSet.begin())); // result set

for (auto val : resultSet) // displaying all the values
    cout << val << " ";
```

- Output: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Algorithms for Set Operations: set_intersection

```
set<int> oneSet = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
set<int> twoSet = {5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};

set<int> resultSet;

// set_intersection: elements that are in both sets
set_intersection(oneSet.begin(), oneSet.end(),          //first set
          twoSet.begin(), twoSet.end(),                 //second set
          inserter(resultSet, resultSet.begin()));      //result set

for (auto val : resultSet) // displaying all the values
    cout << val << " ";
```

- Output: 5 6 7 8 9 10

Algorithms for Set Operations: set_symmetric_difference

```
set<int> oneSet = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
set<int> twoSet = {5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};

set<int> resultSet;

// set_symmetric_difference: appear in either set but not both
set_symmetric_difference(oneSet.begin(), oneSet.end(), //first set
          twoSet.begin(), twoSet.end(),                //second set
          inserter(resultSet, resultSet.begin()));     //result set

for (auto val : resultSet) // displaying all the values
    cout << val << " ";
```

- Output: 1 2 3 4 11 12 13 14 15

Algorithms for Set Operations: set_difference

```
set<int> oneSet = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
set<int> twoSet = {5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};

set<int> resultSet;

// set_difference: appear in first set but not the second
set_difference(oneSet.begin(), oneSet.end(),          //first set
          twoSet.begin(), twoSet.end(),               //second set
          inserter(resultSet, resultSet.begin()));    //result set

for (auto val : resultSet) // displaying all the values
    cout << val << " ";
```

- Output: 1 2 3 4

Algorithms for Set Operations: set_includes

```
set<int> oneSet = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
set<int> twoSet = {5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};

set<int> resultSet;

// set_includes: first set contains all the elements
// of the second set
// we do not have this function.
// But, how can we determine this?
```

- Output: first set includes all the elements of the second set

Algorithms for Set Operations: set_includes

```
set<int> oneSet = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
//set<int> twoSet = {5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
set<int> twoSet = {1, 2, 3, 4, 5, 6, 7, 8, 9, 11};

set<int> result_set;

if(oneSet.size()!=twoSet.size())
{
      cout<<"They are not equal" << endl;
}
else
{
      set_symmetric_difference(oneSet.begin(), oneSet.end(),     //first set
            twoSet.begin(), twoSet.end(),                        //second set
            inserter(result_set, result_set.begin()));

      if(result_set.size()==0)
            cout<<"One set contains all the elements of the other set" << endl;
      else
            cout<<"They are not equal" << endl;
}
```

- Output: first set includes all the elements of the second set

17.7

Something that we have never seen before.
It is out of this world! Prepare for the ride.

17.7

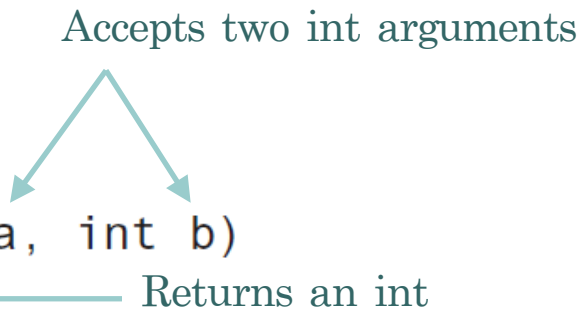
Introduction to Function Objects and Lambda Expressions

Function Objects

- A function object is an object that acts like a function.
  - It can be called
  - It can accept arguments
  - It can return a value
- Function objects are also known as functors

Function Objects

```
1   #ifndef SUM_H
2   #define SUM_H
3
4   class Sum
5   {
6   public:
7       int operator()(int a, int b)
8       { return a + b; }
9   };
10  #endif
```



- To create a function object, you write a class that overloads the () operator.

Function Objects

Program 17-34

```
 1  #include <iostream>
 2  #include "Sum.h"
 3  using namespace std;
 4
 5  int main()
 6  {
 7     // Local variables
 8     int x = 10;
 9     int y = 2;
10     int z = 0;
11
12     // Create a Sum object.
13     Sum sum;
14
15     // Call the sum function object.
16     z = sum(x, y);
17
18     // Display the result.
19     cout << z << endl;
20
21     return 0;
22  }
```

Program Output

```
12
```

Anonymous Function Objects

```
1   #ifndef IS_EVEN_H
2   #define IS_EVEN_H
3
4   class IsEven
5   {
6   public:
7       bool operator()(int x)
8       { return x % 2 == 0; }
9   };
10  #endif
```

- Function objects can be called at the point of their creation, without being given a name. Consider this class:

Anonymous Function Objects

Program 17-36

```
 1  #include <iostream>
 2  #include <vector>
 3  #include <algorithm>
 4  #include "IsEven.h"
 5  using namespace std;
 6
 7  int main()
 8  {
 9     // Create a vector of ints.
10     vector<int> v = { 1, 2, 3, 4, 5, 6, 7, 8 };
11
12     // Get the number of elements that even.
13     int evenNums = count_if(v.begin(), v.end(), IsEven());
14
15     // Display the results.
16     cout << "The vector contains " << evenNums << " even numbers.\n";
17     return 0;
18  }
```

An IsEven object is created here, but not given a name. It is anonymous.

Program Output

```
The vector contains 4 even numbers.
```

Predicate Terminology

- A function or function object that returns a Boolean value is called a predicate.
- A predicate that takes only one argument is called a unary predicate.
- A predicate that takes two arguments is called a binary predicate.
- This terminology is used in much of the available C++ documentation and literature.

Lambda Expressions

- A lambda expression is a compact way of creating a function object without having to write a class declaration.
- It is an expression that contains only the logic of the object's operator() member function.
- When the compiler encounters a lambda expression, it automatically generates a function object in memory, using the code that you provide in the lambda expression for the operator() member function.

Lambda Expressions

- General format:

```
[](parameter list)
{ //function body
}
```

- The [] is known as the lambda introducer. It marks the beginning of a lambda expression.
- parameter list is a list of parameter declarations for the function object's operator() member function.
- function body is the code that should be the body of the object's operator() member function.

Lambda Expressions

- Example: a lambda expression for a function object that computes the sum of two integers:

```
[](int a, int b) { return x + y; }
```

Lambda Expressions

- Example: a lambda expression for a function object that determines whether an integer is a even:

```
[](int x) { return (x % 2) == 0; }
```

Lambda Expressions

- Example: a lambda expression for a function object that takes an integer as input and prints the square of that integer:

```
[](int a) { cout << a * a << " "; }
```

Lambda Expressions

- When you call a lambda expression, you write a list of arguments, enclosed in parentheses, right after the expression.
- For example, the following code snippet displays 7, which is the sum of the variables x and y:

```
int x = 2;
int y = 5;
cout << [](int a, int b) {return a + b;}(x, y)
<< endl;
```

Lambda Expressions

- The following code segment counts the even numbers in a vector:

```
// Create a vector of ints.
vector<int> v = { 1, 2, 3, 4, 5, 6, 7, 8 };

// Get the number of elements that are even.
int evenNums = count_if(v.begin(), v.end(), []
(int x) {return x % 2 == 0;});

// Display the results.
cout << "The vector contains " << evenNums << "
even numbers.\n";
```

Lambda Expressions

- Because lambda expressions generate function objects, you can assign a lambda expression to a variable and then call it through the variable's name:

```
auto sum = [](int a, int b) {return a + b;};
int x = 2;
int y = 5;
int z = sum(x, y);
```

Lambda Expressions

Program 17-37

```
 1  #include <iostream>
 2  #include <vector>
 3  #include <algorithm>
 4  using namespace std;
 5
 6  int main()
 7  {
 8     // Create a vector of ints.
 9     vector<int> v = { 1, 2, 3, 4, 5, 6, 7, 8 };
10
11     // Use a lambda expression to create a function object.
12     auto isEven = [](int x) { return x % 2 == 0; };
13
14     // Get the number of elements that even.
15     int evenNums = count_if(v.begin(), v.end(), isEven);
16
17     // Display the results.
18     cout << "The vector contains " << evenNums << " even numbers.\n";
19     return 0;
20  }
```

Program Output

```
The vector contains 4 even numbers.
```

Lambda Expressions

```
class Student
{ private:
      int st_id;
      string name;
  public:
      Student(int id=0, string name="")
      {
            this->st_id = id;
            this->name = name;
      }

      int get_id() { return  st_id; }
      string get_name() { return name;}
};
```

```
int main()
{

auto is_even = [](Student one)
  { return one.get_id() %2 == 0; };

vector<Student> my_students {
Student(1, "Hengyi"), Student(2, "Murtuza"),
Student(3, "Saba")};

int count = count_if(my_students.begin(),
my_students.end(), is_even);

cout<< count << endl;

return 0;
}
```

Functional Classes in the STL

Table 17-15 STL Function Object Classes

Functional Class	Description
`less<T>`	`less<T>()(T a, T b)` is true if and only if `a < b`
`less_equal<T>`	`less_equal()(T a, T b)` is true if an only if `a <= b`
`greater<T>`	`greater<T>()(T a, T b)` is true if and only if `a > b`
`greater_equal<T>`	`greater_equal<T>()(T a, T b)` is true if and only if `a >= b`

- The STL library defines a number of classes that you can instantiate to create function objects in your program.
- To use these classes, you must #include the <functional> header file.
- Table 17-15 in your textbook lists a few of the functional classes:

Chapter 17

- Sections
  - 17.1, 17.2, 17.3, 17.4, 17.5, 17.6, 17.7
- Checkpoint exercise
  - 17.1, 17.2, 17.3, 17.4, 17.5, 17.6, 17.7, 17.8, 17.9, 17.10, 17.11, 17.12, 17.13, 17.14, 17.15, 17.16, 17.17, 17.20, 17.21, 17.22, 17.23, 17.24, 17.25, 17.26, 17.27, 17.28, 17.29, 17.30, 17.31, 17.32, 17.33, 17.34, 17.36, 17.37, 17.38, 17.39, 17.41, 17.42, 17.43, 17.44, 17.45, 17.46, 17.49
- Short answer
  - 1, 2, 4, 7, 13, 14, 15,
- Fill-in-the Blank
  - Try all of them
- Algorithm workbench
  - 54, 58, 60, 61, 62, 64, 65, 66, 67
- True/False
  - Try all of them
- Find the errors
  - Try all of them
- Programming challenges
  - 1, 2, 3, 4 (when decrypting use another map), 5 (use set and apply set functions), 6 (use map), 8

Please review this chapter

Thank you