

CHAPTER 16 – STANDARD TEMPLATE LIBRARY

Code using several STL elements: Listing 16.19 - usealgo.cpp

```
//usealgo.cpp -- using several STL elements
#include <iostream>
#include <string>
#include <vector>
#include <set>
#include <map>
#include <iterator>
#include <algorithm>
#include <cctype>
using namespace std;

char toLower(char ch) {
    return tolower(ch);
}

string & ToLower(string & st);

void display(const string & s);

int main()
{
    vector<string> words;
    cout << "Enter words (enter quit to quit):\n";
    string input;

    while (cin >> input && input != "quit")
        words.push_back(input);

    cout << "You entered the following words:\n";
    for_each(words.begin(), words.end(), display);
    cout << endl;

    // place words in set, converting to lowercase
    set<string> wordset;

    transform(words.begin(), words.end(),
        insert_iterator<set<string> > (wordset, wordset.begin()),
        ToLower);
```

```
    cout << "\nAlphabetic list of words:\n";
    for_each(wordset.begin(), wordset.end(), display);
    cout << endl;

    // place word and frequency in map
    map<string, int> wordmap;
    set<string>::iterator si;

    for (si = wordset.begin(); si != wordset.end(); si++)
        wordmap[*si] = count(words.begin(), words.end(), *si);

    // display map contents
    cout << "\nWord frequency:\n";
    for (si = wordset.begin(); si != wordset.end(); si++)
        cout << *si << ": " << wordmap[*si] << endl;

    // cin.get();
    // cin.get();
    return 0;
}

string & ToLower(string & st)
{
    transform(st.begin(), st.end(), st.begin(), toLower);
    return st;
}

void display(const string & s)
{
    cout << s << " ";
}

Enter words (enter quit to quit):
The dog saw the cat and thought the cat fat
The cat thought the cat perfect
quit
You entered the following words:
The dog saw the cat and thought the cat fat The cat thought the cat perfect

Alphabetic list of words:
and cat dog fat perfect saw the thought

Word frequency:
and: 1
cat: 4
dog: 1
fat: 1
perfect: 1
saw: 1
the: 5
thought: 2
```

Vector:

Listing 16.7 – vect1.cpp

```
// vect1.cpp -- introducing the vector template
#include <iostream>
#include <string>
#include <vector>

const int NUM = 5;
int main()
{
    using std::vector;
    using std::string;
    using std::cin;
    using std::cout;
    using std::endl;

    vector<int> ratings(NUM);
    vector<string> titles(NUM);
    cout << "You will do exactly as told. You will enter\n"
         << NUM << " book titles and your ratings (0-10).\n";
    int i;
    for (i = 0; i < NUM; i++)
    {
        cout << "Enter title #" << i + 1 << ": ";
        getline(cin,titles[i]);
        cout << "Enter your rating (0-10): ";
        cin >> ratings[i];
        cin.get();
    }
    cout << "Thank you. You entered the following:\n"
         << "Rating\tBook\n";
    for (i = 0; i < NUM; i++)
    {
        cout << ratings[i] << "\t" << titles[i] << endl;
    }
    // cin.get();

    return 0;
}
```

Listing 16.8 – vect2.cpp

```
// vect2.cpp -- methods and iterators
#include <iostream>
#include <string>
#include <vector>

struct Review {
    std::string title;
    int rating;
};

bool FillReview(Review & rr);

void ShowReview(const Review & rr);

int main()
{
    using std::cout;
    using std::vector;
    vector<Review> books;
    Review temp;

    while (FillReview(temp))
        books.push_back(temp);
    int num = books.size();

    if (num > 0)
    {
        cout << "Thank you. You entered the following:\n"
            << "Rating\tBook\n";
        for (int i = 0; i < num; i++)
            ShowReview(books[i]);
        cout << "Reprising:\n"
            << "Rating\tBook\n";
        vector<Review>::iterator pr;
        for (pr = books.begin(); pr != books.end(); pr++)
            ShowReview(*pr);
        vector<Review> oldlist(books);    // copy constructor used
        if (num > 3)
        {
            // remove 2 items
            books.erase(books.begin() + 1, books.begin() + 3);
            cout << "After erasure:\n";
            for (pr = books.begin(); pr != books.end(); pr++)
                ShowReview(*pr);
        }
    }
}
```

```
        // insert 1 item
        books.insert(books.begin(), oldlist.begin() + 1,
                     oldlist.begin() + 2);
        cout << "After insertion:\n";
        for (pr = books.begin(); pr != books.end(); pr++)
            ShowReview(*pr);
    }
    books.swap(oldlist);
    cout << "Swapping oldlist with books:\n";
    for (pr = books.begin(); pr != books.end(); pr++)
        ShowReview(*pr);
}
else
    cout << "Nothing entered, nothing gained.\n";
// std::cin.get();
return 0;
}

bool FillReview(Review & rr)
{
    std::cout << "Enter book title (quit to quit): ";
    std::getline(std::cin, rr.title);
    if (rr.title == "quit")
        return false;
    std::cout << "Enter book rating: ";
    std::cin >> rr.rating;
    if (!std::cin)
        return false;
    // get rid of rest of input line
    while (std::cin.get() != '\n')
        continue;
    return true;
}

void ShowReview(const Review & rr)
{
    std::cout << rr.rating << "\t" << rr.title << std::endl;
}
```

Iterators:

Listing 16.10 – copyit.cpp

```
// copyit.cpp -- copy() and iterators
#include <iostream>
#include <iterator>
#include <vector>

int main()
{
    using namespace std;

    int casts[10] = {6, 7, 2, 9, 4, 11, 8, 7, 10, 5};
    vector<int> dice(10);

    // copy from array to vector
    copy(casts, casts + 10, dice.begin());
    cout << "Let the dice be cast!\n";

    // create an ostream iterator
    ostream_iterator<int, char> out_iter(cout, " ");

    // copy from vector to output
    copy(dice.begin(), dice.end(), out_iter);
    cout << endl;
    cout << "Implicit use of reverse iterator.\n";
    copy(dice.rbegin(), dice.rend(), out_iter);
    cout << endl;
    cout << "Explicit use of reverse iterator.\n";

    // vector<int>::reverse_iterator ri; // use if auto doesn't work
    for (auto ri = dice.rbegin(); ri != dice.rend(); ++ri)
        cout << *ri << ' ';
    cout << endl;
    // cin.get();
    return 0;
}
```

Listing 16.11 – inserts.cpp

```
// inserts.cpp -- copy() and insert iterators
#include <iostream>
#include <string>
#include <iterator>
#include <vector>
#include <algorithm>

void output(const std::string & s) {
    std::cout << s << " ";
}

int main()
{
    using namespace std;
    string s1[4] = {"fine", "fish", "fashion", "fate"};
    string s2[2] = {"busy", "bats"};
    string s3[2] = {"silly", "singers"};
    vector<string> words(4);

    copy(s1, s1 + 4, words.begin());

    for_each(words.begin(), words.end(), output);
    cout << endl;

    // construct anonymous back_insert_iterator object
    copy(s2, s2 + 2, back_insert_iterator<vector<string> >(words));
    for_each(words.begin(), words.end(), output);
    cout << endl;

    // construct anonymous insert_iterator object
    copy(s3, s3 + 2, insert_iterator<vector<string> >(words, words.begin()));
    for_each(words.begin(), words.end(), output);
    cout << endl;

    // cin.get();
    return 0;
}
```

List: Listing 16.12 – list.cpp

```
// list.cpp -- using a list
#include <iostream>
#include <list>
#include <iterator>
#include <algorithm>

void outint(int n) { std::cout << n << " ";}
int main()
{
    using namespace std;
    list<int> one(5, 2); // list of 5 2s
    int stuff[5] = {1,2,4,8, 6};
    list<int> two;
    two.insert(two.begin(),stuff, stuff + 5 );
    int more[6] = {6, 4, 2, 4, 6, 5};
    list<int> three(two);
    three.insert(three.end(), more, more + 6);
    cout << "List one: ";
    for_each(one.begin(),one.end(), outint);
    cout << endl << "List two: ";
    for_each(two.begin(), two.end(), outint);
    cout << endl << "List three: ";
    for_each(three.begin(), three.end(), outint);
    three.remove(2);
    cout << endl << "List three minus 2s: ";
    for_each(three.begin(), three.end(), outint);
    three.splice(three.begin(), one);
    cout << endl << "List three after splice: ";
    for_each(three.begin(), three.end(), outint);
    cout << endl << "List one: ";
    for_each(one.begin(), one.end(), outint);
    three.unique();
    cout << endl << "List three after unique: ";
    for_each(three.begin(), three.end(), outint);
    three.sort();
    three.unique();
    cout << endl << "List three after sort & unique: ";
    for_each(three.begin(), three.end(), outint);
    two.sort();
    three.merge(two);
    cout << endl << "Sorted two merged into three: ";
    for_each(three.begin(), three.end(), outint);
    cout << endl;
    return 0;
}
```


Associative: Listing 16.13 – setops.cpp

```
#include <iostream>
#include <string>
#include <set>
#include <algorithm>
#include <iterator>
int main()
{
    using namespace std;
    const int N = 6;
    string s1[N] = {"buffoon", "thinkers", "for", "heavy", "can", "for"};
    string s2[N] = {"metal", "any", "food", "elegant", "deliver", "for"};
    set<string> A(s1, s1 + N);
    set<string> B(s2, s2 + N);
    ostream_iterator<string, char> out(cout, " ");
    cout << "Set A: ";
    copy(A.begin(), A.end(), out);
    cout << endl;
    cout << "Set B: ";
    copy(B.begin(), B.end(), out);
    cout << endl;
    cout << "Union of A and B:\n";
    set_union(A.begin(), A.end(), B.begin(), B.end(), out);
    cout << endl;
    cout << "Intersection of A and B:\n";
    set_intersection(A.begin(), A.end(), B.begin(), B.end(), out);
    cout << endl;
    cout << "Difference of A and B:\n";
    set_difference(A.begin(), A.end(), B.begin(), B.end(), out);
    cout << endl;
    set<string> C;
    cout << "Set C:\n";
    set_union(A.begin(), A.end(), B.begin(), B.end(),
        insert_iterator<set<string>>(C, C.begin()));
    copy(C.begin(), C.end(), out);
    cout << endl;
    string s3("grungy");
    C.insert(s3);
    cout << "Set C after insertion:\n";
    copy(C.begin(), C.end(), out);
    cout << endl;
    cout << "Showing a range:\n";
    copy(C.lower_bound("ghost"), C.upper_bound("spook"), out);
    cout << endl;
    return 0;
}
```

Listing 16.14 – multimap.cpp

```
// multimap.cpp -- use a multimap
#include <iostream>
#include <string>
#include <map>
#include <algorithm>

typedef int KeyType;
typedef std::pair<const KeyType, std::string> Pair;
typedef std::multimap<KeyType, std::string> MapCode;

int main()
{
    using namespace std;
    MapCode codes;

    codes.insert(Pair(415, "San Francisco"));
    codes.insert(Pair(510, "Oakland"));
    codes.insert(Pair(718, "Brooklyn"));
    codes.insert(Pair(718, "Staten Island"));
    codes.insert(Pair(415, "San Rafael"));
    codes.insert(Pair(510, "Berkeley"));

    cout << "Number of cities with area code 415: "
         << codes.count(415) << endl;
    cout << "Number of cities with area code 718: "
         << codes.count(718) << endl;
    cout << "Number of cities with area code 510: "
         << codes.count(510) << endl;
    cout << "Area Code  City\n";
    MapCode::iterator it;
    for (it = codes.begin(); it != codes.end(); ++it)
        cout << "    " << (*it).first << "    "
             << (*it).second << endl;

    pair<MapCode::iterator, MapCode::iterator>
        auto range
        = codes.equal_range(718);
    cout << "Cities with area code 718:\n";
    for (it = range.first; it != range.second; ++it)
        cout << (*it).second << endl;
    // cin.get();
    return 0;
}
```

Smart Pointers:

Listing 16.5 – smrtptrs.cpp

```
// smrtptrs.cpp -- using three kinds of smart pointers
#include <iostream>
#include <string>
#include <memory>

class Report
{
private:
    std::string str;
public:
    Report(const std::string s) : str(s) { std::cout << "Object created!\n"; }
    ~Report() { std::cout << "Object deleted!\n"; }
    void comment() const { std::cout << str << "\n"; }
};

int main()
{
    {
        std::auto_ptr<Report> ps (new Report("using auto_ptr"));
        ps->comment(); // use -> to invoke a member function
    }

    {
        std::shared_ptr<Report> ps (new Report("using shared_ptr"));
        ps->comment();
    }

    {
        std::unique_ptr<Report> ps (new Report("using unique_ptr"));
        ps->comment();
    }

    // std::cin.get();
    return 0;
}
```

Listing 16.6 – fowl.cpp

```
// fowl.cpp -- auto_ptr a poor choice
#include <iostream>
#include <string>
#include <memory>

int main()
{
    using namespace std;

    auto_ptr<string> films[5] =
    {
        auto_ptr<string> (new string("Fowl Balls")),
        auto_ptr<string> (new string("Duck Walks")),
        auto_ptr<string> (new string("Chicken Runs")),
        auto_ptr<string> (new string("Turkey Errors")),
        auto_ptr<string> (new string("Goose Eggs"))
    };

    auto_ptr<string> pwin;
    pwin = films[2]; // films[2] loses ownership

    cout << "The nominees for best avian baseball film are\n";
    for (int i = 0; i < 5; i++)
        cout << *films[i] << endl;

    cout << "The winner is " << *pwin << "!\n";
    // cin.get();
    return 0;
}
```