The Standard Template Library Tutorial

184.437 Wahlfachpraktikum (10.0)

Johannes Weidl

Information Systems Institute
Distributed Systems Department
Technical University Vienna

Advisor	Dipl. Ing. Georg Trausmuth
Professor	DI Dr. Mehdi Jazayeri

Sunday, 26. November 1995

Exercise Solution Part

Table of contents

Exercise 4.1.1:

```
#define __MINMAX_DEFINED   // use STL's generic min and max templates

#include "vector.h"        // include STL vector implementation

#include <iostream.h>

void main (void)
{

  vector<int> v(5);              // define a vector of int and
                                 // reserve memory for five elements

  for (int i = 0; i < 5; i++)

        v[i] = 2*i;                 // store arbitrary values into v[0] to v[4]

  cout << "Five values stored in a vector are written to cout:" << endl;

  for (i = 0; i < 5; i++)

        cout << v[i] << "  ";       // print values to cout

  cout << endl;

  // of course you can also use iterators

  // define an iterator to the first vector element
  vector<int>::iterator first = v.begin();
  // define an iterator past the last vector element
  vector<int>::iterator last  = v.end();

  cout << "Now the output loop works with iterators:" << endl;

  while (first != last)

        cout << *first++ << "  ";    // first the iterator is dereferenced,
                                    // then it is incremented
}
```

Exercise 4.1.2:

```
#define __MINMAX_DEFINED   // use STL's generic min and max templates

#include "list.h"          // include STL-list implementation

#include <iostream.h>

void main (void)
{

  list<int> bit_seq;             // define a list of int
  int input = 0;                 // value read from cin
  int count_1 = 0;               // counter for number of 1's

  cout << "Insert values 0 and 1, another value to stop input..." << endl;

  while (cin >> input) {

        if (!(input == 0 || input == 1)) break;
        bit_seq.push_back (input);             // list member function push_back
  }

  // create a new list for bit_stuffing
```

```
  list<int> bit_stuffed_seq (bit_seq);

  // define loop iterators
  list<int>::iterator first = bit_stuffed_seq.begin();
  list<int>::iterator last  = bit_stuffed_seq.end();

  // bit stuff loop

  while (first != last) {

                  if (*first == 0)
                           count_1 = 0;       // reset 1's-counter
                  else if (*first == 1)
                           count_1++;         // increment number of
                                             //  consecutive 1's

                  first++;                    // go to the next list element

                  if (count_1 == 5) {
                              // insert a 0 after the fifth consecutive 1

                    bit_stuffed_seq.insert (first, 0);
                    count_1 = 0;                          // reset counter
                  }
  }

}
```

Exercise 4.1.3:

```
#define __MINMAX_DEFINED  // use STL's generic min and max templates

#include "list.h"         // include STL-list implementation

#include <iostream.h>

void main (void)
{

  list<int> bit_seq;            // define a list of int
  int input = 0;                // value read from cin
  int count_1 = 0;              // counter for number of 1's

  cout << "Insert values 0 and 1, another value to stop input..." << endl;

  while (cin >> input) {

        if (!(input == 0 || input == 1)) break;
        bit_seq.push_back (input);             // list member function push_back
  }

  // output loop

  cout << "Original bit sequence:" << endl;

  // define an iterator to the first list element
  list<int>::iterator first = bit_seq.begin();

  // define an iterator past(!) the last list element
  list<int>::iterator last  = bit_seq.end();

  while (first != last)

               cout << *first++;       // dereference iterator to get value
                                       // then increment iterator
  cout << endl;

  // create a new list for bit_stuffing
  list<int> bit_stuffed_seq (bit_seq);

  // define loop iterators
  first = bit_stuffed_seq.begin();
```

```
  last  = bit_stuffed_seq.end();

  // bit stuff loop

  while (first != last) {

                  if (*first == 0)
                              count_1 = 0;       // reset 1's-counter
                  else if (*first == 1)
                              count_1++;         // increment number of
                                               //  consecutive 1's

                  first++;                      // go to the next list element

                  if (count_1 == 5) {
                                  // insert a 0 after the fifth consecutive 1

                    bit_stuffed_seq.insert (first, 0);
                    count_1 = 0;                          // reset counter
                  }
  }

  // output loop

  cout << "Bit-stuffed bit sequence:" << endl;

  first = bit_stuffed_seq.begin();
  last  = bit_stuffed_seq.end();

  while (first != last)

              cout << *first++;       // dereference iterator to get value
                                      // then increment iterator
  cout << endl;
}
```

Exercises 4.1.4 and 4.1.5:

```
#define __MINMAX_DEFINED  // use STL's generic min and max templates

#include "list.h"         // include STL-list implementation

#include <iostream.h>

// Since the output loop is often used, we can form a template function
// which does the job.

template <class InputIterator>
void copy_to_cout (InputIterator first, InputIterator last) {

  while (first != last) cout << *first++;

  cout << endl;
}

// The template class "InputIterator" gets a meaning when you study
// chapter 4.2 in iterators.

void main (void)
{

  list<int> bit_seq;             // define a list of int
  int input = 0;                 // value read from cin
  int count_1 = 0;               // counter for number of 1's

  cout << "Insert values 0 and 1, another value to stop input..." << endl;

  while (cin >> input) {

        if (!(input == 0 || input == 1)) break;
        bit_seq.push_back (input);             // list member function push_back
  }
```

```
// output loop

cout << "Original bit sequence:" << endl;

copy_to_cout (bit_seq.begin(), bit_seq.end());

// create a new list for bit_stuffing
list<int> bit_stuffed_seq (bit_seq);

// define loop iterators
list<int>::iterator first = bit_stuffed_seq.begin();
list<int>::iterator last  = bit_stuffed_seq.end();

// bit stuff loop

while (first != last) {

                  if (*first == 0)
                                 count_1 = 0;       // reset 1's-counter
                  else if (*first == 1)
                                 count_1++;         // increment number of
                                                   //  consecutive 1's

                  first++;                          // go to the next list element

                  if (count_1 == 5) {
                                      // insert a 0 after the fifth consecutive 1

                    bit_stuffed_seq.insert (first, 0);
                    count_1 = 0;                          // reset counter
                  }
}

// output loop

cout << "Bit-stuffed bit sequence:" << endl;

copy_to_cout (bit_stuffed_seq.begin(), bit_stuffed_seq.end());

double rel_exp;               // relative expansion (in percent)

rel_exp = (double) bit_stuffed_seq.size() / bit_seq.size();
rel_exp = (rel_exp - 1) * 100;

cout.precision (4);
cout << "Relative expansion: " << rel_exp << "%" << endl;
cout << "Absolute expansion: " << (bit_stuffed_seq.size()-bit_seq.size());
cout << " bit" << endl;

// bit unstuff loop

first = bit_stuffed_seq.begin();
last  = bit_stuffed_seq.end();
list<int>::iterator erase_it;    // used because the erase-iterator
                                 // gets invalid
count_1 = 0;

while (first != last) {

           if (*first == 0) count_1 = 0; else count_1++;

           if (count_1 == 5) {

                  erase_it = first;
                  if (*(++erase_it) != 0) {        // error in input bit sequence
                    cout << "not a valid bit-stuffed sequence!" << endl;
                    exit(0);
                  }
                  bit_stuffed_seq.erase (erase_it);       // erase zero
                  count_1 = 0;
           }

           ++first;
```

```
  }

  cout << "After bit-unstuffing: ";

  // check for success (using operator==)

  bit_stuffed_seq == bit_seq ?  cout << "sequences are equal" :  \
                                cout << "sequences are not equal";
  cout << endl;
}
```

Exercise 4.2.1:

Only the important code pieces are presented:

```
#include <fstream.h>

// read original bit sequence from file bit_seq
ifstream in_file("bit_seq");

copy (istream_iterator<int, ptrdiff_t> (in_file),
      istream_iterator<int, ptrdiff_t> (), back_inserter (bit_seq) );

// store bit stuffed sequence in file bit_stff
ofstream out_file("bit_stff");

copy (bit_stuffed_seq.begin(), bit_stuffed_seq.end(),
      ostream_iterator<int> (out_file, " ") );
```

Exercise 4.3.1:

```
#define __MINMAX_DEFINED   // use STL's generic min and max templates

#include "vector.h"       // include STL vector implementation
#include "algo.h"         // include STL algorithm implementations

#include <iostream.h>

void main (void)
{

  vector<int> a;
  vector<int> b;

  for (int i = 0; i < 4; i++) {a.push_back (i*2); b.push_back ((i+1)*3); }

  vector<int> c(4); // allocate memory for 4 int values!!

  // use the algo "transform" and the function object "plus"
  // transfrom takes the elements of vectors a and b, adds them using
  // plus and writes the results to c
  transform  (a.begin(), a.end(), b.begin(), c.begin(), plus<int>() );

  copy (c.begin(), c.end(), ostream_iterator<int> (cout, " ") );
}
```

Exercise 4.3.2:

```
#define __MINMAX_DEFINED   // use STL's generic min and max templates

#include "vector.h"       // include STL vector implementation
#include "algo.h"         // include STL algorithm implementations

#include <iostream.h>

template <class value_type>
class gen {
      public:
            gen() : start(0), step(1) {}
```

```
            gen (value_type sta, value_type ste) : start(sta), step(ste) {}
            value_type operator() (void)
              { value_type tmp = start; start += step; return tmp; }
      private:
            value_type   start, step;
};


void main (void)
{

  vector<int> v(10);

  generate (v.begin(), v.end(),
            gen<vector<int>::value_type> (1, 2) );

  copy (v.begin(), v.end(), ostream_iterator<int> (cout, " ") );

}
```