CSC2040 Data Structures, Algorithms and Programming Languages

Practical 5

Wednesday 8th November 2017

In this practical, we practise the creation of C++ template functions and classes. The template is one of the C++'s most sophisticated and high-powered features. Using templates, it is possible to create generic functions and classes. In a generic function or class, the type of data upon which the function or class operates is specified as a parameter. Thus, you can use one function or class with several different types of data without having to explicitly recode specific versions for each data type.

Program 1

Write a template function `find_max` that takes a generic array and its length as arguments, and returns the index of the element in the array that has maximum value. As outlined in your notes the template function should be defined in a header file. Then you create a test program, `TestTemplateFunction.cpp`, to examine the correctness of your code by calling your template function with an `int` array and a `double` array, each containing an appropriate number of random numbers. Note that your template function must include appropriate checks for the validity of the array that is passed to the function. If you use **new** to allocate the test arrays, you must **delete** them after use.

Program 2

As you know, in C++, it is possible to overrun (or underrun) an array boundary at run time without generating a runtime error message. The following shows a class that implements an `int` type *safe array* that provides runtime boundary checking by overloading the [ ] operator. Study the class and the attached test program carefully:

1. The class uses a constructor to allocate the memory `data` for the array and a destructor to free the memory after the array object is destroyed;
2. The class overloads the operator [ ] for accessing the individual elements of the array with boundary checking;
3. The operator function returns the address (i.e. the reference) of the element that is being accessed.

The class definition

```cpp
class s_array {
public:
    s_array(int size);
    ~s_array();

    int &operator[](int i);
private:
    int size;
    int* data;
};
```

```cpp
s_array::s_array(int size)
    : size(size)
{
    if (size > 0) data = new int[size];
    else {
        cout << "invalid array size" << size << endl;
        exit(1);
    }
}
s_array::~s_array()
{
    delete[] data;
}
int &s_array::operator[](int i)
{
    if (i < 0 || i >= size) {
        cout << "index " << i << " is out-of-bounds." << endl;
        exit(1);
    }
    return data[i];
}
```

A test program

```
int main()
{
    s_array array(10);  // create a 10-element int array

    array[5] = 5;       // in-bound accesses
    cout << array[5] << endl;

    array[-1] = 2;      // out-bound accesses
    cout << array[15];

    return 0;
}
```

The above class, however, only allows you to create safe arrays for integers. In this practice, you are asked to implement a template class, to create a generic safe-array type that can be used for creating safe arrays of any data type. You can implement this template class by following the above example.

Then you design a test program, `TestTemplateClass.cpp`, to test your code by creating safe arrays for at least three data types: `int`, `double`, and `char`. You should test your code thoroughly by accessing the arrays you create within bounds and out-of-bounds.

Program 3

Part of the C++ Standard Library, the Standard Template Library (STL) contains many useful container classes and algorithms. As you might imagine, these various parts of the library are written using templates and so are generic in type. The containers found in the STL are lists, maps, queues, sets, stacks, and vectors. The algorithms include sequence operations, sorts, searches, merges, heap operations, and min/max operations. In this program, we ask you to practise the use of the set class, in header **<set>.** Create a test program, `TestSet.cpp`, and type in the following code:

```
#include <iostream>
#include <set>
#include <algorithm>
using namespace std;

int main() {
  set<int> iset;
  iset.insert(5);
  iset.insert(9);
  iset.insert(1);
  iset.insert(8);
  iset.insert(3);

  cout << "iset contains:";
  set<int>::iterator it;
  for(it=iset.begin(); it != iset.end(); it++)
        cout << " " << *it;
  cout << endl;

  int searchFor;
  cin >> searchFor;
  if(binary_search(iset.begin(), iset.end(), searchFor))
        cout << "Found " << searchFor << endl;
  else
        cout << "Did not find " << searchFor << endl;

  return 0;
}
```

In this program, you create an integer set and insert several integers into it. You then create an iterator corresponding to the set at line 15. An iterator is basically a pointer that provides a view of the set. (Most of the other containers also provide iterators.) By using this iterator, you display all the elements in the set and print out `iset contains: 1 3 5 8 9`. Try inserting 3 into the set again and see what is printed out. What do you notice? Note that the set automatically sorts its own items. Finally, you ask the user for an integer, search for that integer in the set, and display the result.

Practical Test 2

Practical Test 2 will take place during the practical class on Wednesday 15 November. You will be asked to develop C++ programs to solve a given set of problems, and then to use the Assignment Tool in Queens Online to upload your answers. The test is open book, but with restrictions. See the Course Information for the “Format for practical tests and final examination”. Pen and paper can be used for working out / rough work.