Northwestern University
Department of Electrical Engineering and Computer Science

Programming Assignment 5

EECS 231 Advanced Programming
Fall Quarter 2006

Due : Thursday 11/30/06 at 11:59pm

There are 4 pages to this handout

Goals of this assignment

- Design and implement a simple structure from scratch using class templates
- Integrate a simple user-defined structure into the STL

Part 1

Code a class template for a "standard" doubly linked list. You must not use the STL provided list. You must provide the following methods:

```
DList();             // default constructor
DList(const DList&); // copy constructor
~DList();            // destructor

DList& operator=(const DList&);  // overloaded assignment operator

T& front();  // returns a reference to the first element of the list
T& back();   // returns a reference to the last element of the list

void insert(iterator& pos, const T & val);  // insert val at position pos
void erase(iterator& pos);                  // remove element at position pos

iterator begin();           // return an iterator to the first element
iterator end();             // return an iterator just past the last element

iterator search(const T&);  // search for the given data. Return an iterator
                            // to that element if found, end() otherwise

void push_front(const T&);  // insert at the beginning of the list
void push_back(const T&);   // insert at the end of the list
void pop_front();           // remove the first element
void pop_back();            // remove the last element
```

We want to be able to apply STL algorithms on our list. To that end, we need to implement iterators for `DList`, and in particular bidirectional iterators. The iterator class for our list will inherit from the STL std::iterator as shown below. The operations that need to be implemented (see 16.ppt, p.5) are

`->, *, ++, --, +=, -=, +, -, ==, >, <, >=, <=. !=}`

A partial class definition follows:

```
template <typename T>
class DList {
   public:
      class iterator : public std::iterator<std::bidirectional_iterator_tag, T> {
           private:
                Node *current; // the node where the iterator object points
           public:
                iterator(Node *n=NULL) : current(n) { } ;  // default constructor

                T& operator*() const   // dereferencing operator
                {
                    return current->data;  // 'data' is the value held in that node
                }
      };
};
```

The abstract view of the linked list is as described in class. Internally, it is convenient to implement it as a circular list with a "buffer"/"dummy" node between the last and first element. Keeping track of that node instead of the head of the list makes it easy to quickly access the head and the tail. `.end()` will return an iterator to that buffer node. The contents of this node are undefined. If the user tries to access them, then the behavior of the program will be undefined.

A list node should of course contain data of type T, a pointer to the next node and a pointer to the previous node in the list. The `Node` class should be defined within the list class in accordance with the rules of encapsulation and information hiding:

```
template <typename T>
class DList {
   private:
        class Node {
            ...
        };
};
```

What to submit for this part

- A file `dlist.h` containing the `DList` class template (definition and implementation)
- A file `part1.cpp` containing a `main()` function that tests *all* the methods you have defined as well as the use of the STL algorithms `find()` and `replace_if()`.

Part 2

You have been asked to provide a self-organizing list that uses the Move-To-Front heuristic. Self-organizing lists reorder their elements, typically after every access operation, based on some heuristic, in order to improve average access time. The Move-To-Front heuristic involves moving the target of a search operation to the beginning of the list of so that it is found faster next time.

How would you modify the list you wrote for Part 1? Can your self-organizing list inherit from `DList`? Why or why not? Write a header file `part2.h` that contains a comment section explaining your class design and a code section containing only the definition of the self-organizing list class with comments explaining the function of the methods you have defined. No implementation is needed.

What to submit for this part

- The file `part2.h` as described above.

Part 3

You have been asked to provide a self-organizing list that uses the Frequency Count heuristic. This keeps the elements of the list ordered by the number of times each element is the target of a search.

How would you modify the list you wrote for Part 1? Can your self-organizing list inherit from `DList`? Why or why not? Write a header file `part3.h` that contains a comment section explaining your class design and a code section containing the definition of the self-organizing list class with comments explaining the function of the methods you have defined. You should only include the implementation of the search method and any other methods that may be affected by the use of this heuristic.

What to submit for this part

- The file `part3.h` as described above.

Part 4

You are given the following program:

```
#include<iostream>
#include<vector>
using std::vector;
#include<iostream>
using std::cout;
using std::endl;

int main () {
    vector<int> V(3,10);   // make a vector containing three 10s
    vector<int>::iterator it=V.begin();
    V.insert(V.end(), 5);  // add a 5 at the end
    while (it!=V.end()) {
        cout << *it << endl;  // print the contents. We expect 10 10 10 5
        it ++;
    }

    return 0;
}
```

Compile and run this code. Answer the following questions:

- What was the output of this program?
- It is clearly not what the programmer expected. Why is that?
- Would you expect your `DList` to behave in the same way? Why or why not?

What to submit for this part

- The file `part4.txt` containing the answers to the questions posed above.

How to submit your assignment

Tar and email your files to `ido715@ece.northwestern.edu`.