

Laboratory Exercise 11 (Templated Lists)

Topics: Template Linked Lists

Iterators

Overloaded ++, --, and * operators

Goals: Upon successful completion of this lab you should be able to:

1. Define an iterator for a template container class
2. Overload the increment (++) and decrement (--) operators for an iterator.
3. Overload the de-referencing operator (*) for an iterator.
4. Use the Iterator for a Linked List class in an application program

Related text sections:

Chapter 16

Chapter 17

Laboratory Exercise 11 Instructions

1. Copy your LList2.tmp file from Lab 10 to a new Lab 11 directory.
2. Modify the node type to be a struct.. You will no longer need the “public:” key word since all of the elements of a struct are public by default.
3. Add an iterator class and begin/end and rbegin/rend functions to your LList2 class description:

```
template <class LT> class LList2;

template <class LT> ostream & operator << (ostream & outs, const LList2<LT> & L);

template <class LT>
class LList2
{
    private:
        typedef struct LNode
        {
            LNode ();
            LT data;
            LNode * next;
            LNode * prev;
        };

    public:
        class Iterator
        {
            public:
                Iterator ();
                Iterator (LNode * NP);
                const LT operator * () const;
                Iterator operator ++ ();
                Iterator operator ++ (int);
                Iterator operator -- ();
                Iterator operator -- (int);
                bool operator == (const Iterator & other) const;
                bool operator != (const Iterator & other) const;
            private:
                LNode * current;
        };

        LList2 ();
        LList2 (const LList2 & other);
        ~LList2 ();
        LList2 & operator = (const LList2 & other);
        bool operator == (const LList2 & other);
        int Size () const;
        friend ostream & operator << >> (ostream & outs, const LList2<LT> & L);
        bool InsertFirst (const LT & value);
        bool InsertLast (const LT & value);
        bool DeleteFirst ();
        bool DeleteLast ();
        void Forward (void function (const LT & param));
        void Backward (void function (const LT & param));
        Iterator begin () const;
        Iterator rbegin () const;
        Iterator end () const;
        Iterator rend () const;
};
```

```

private:
    LNode * first;
    LNode * last;
    int size;
};

```

4. Add the following Iterator function implementations to your LList2.tmp file:

```

template <class LT>
LList2<LT>::Iterator::Iterator ()
{
    current = NULL;
}

template <class LT>
LList2<LT>::Iterator::Iterator (LNode * NP)
{
    current = NP;
}

template <class LT>
const LT LList2<LT>::Iterator::operator * () const
{
    return current->data;
}

template <class LT>
typename LList2<LT>::Iterator LList2<LT>::Iterator::operator ++ ()
{
    current = current->next;
    return *this;
}

template <class LT>
typename LList2<LT>::Iterator LList2<LT>::Iterator::operator ++ (int)
{
    Iterator temp = *this;
    current = current->next;
    return temp;
}

template <class LT>
typename LList2<LT>::Iterator LList2<LT>::Iterator::operator -- ()
{
    current = current->prev;
    return *this;
}

template <class LT>
typename LList2<LT>::Iterator LList2<LT>::Iterator::operator -- (int)
{
    Iterator temp = *this;
    current = current->prev;
    return temp;
}

template <class LT>
bool LList2<LT>::Iterator::operator == (const Iterator & other) const
{
    return current == other.current;
}

```

```

template <class LT>
bool LList2<LT>::Iterator::operator != (const Iterator & other) const
{
    return current != other.current;
}

```

5. Add the following LList2 function implementations to your LList2.tmp file:

```

template <class LT>
typename LList2<LT>::Iterator LList2<LT>::begin () const
{
    Iterator temp (first);
    return temp;
}

template <class LT>
typename LList2<LT>::Iterator LList2<LT>::rbegin () const
{
    Iterator temp (last);
    return temp;
}

template <class LT>
typename LList2<LT>::Iterator LList2<LT>::end () const
{
    Iterator temp;
    return temp;
}

template <class LT>
typename LList2<LT>::Iterator LList2<LT>::rend () const
{
    Iterator temp;
    return temp;
}

```

6. Enter the following application program in a new file called application.cpp. **DO NOT COMPILE THIS PROGRAM!**

```

#include <iostream>
#include "LList2.tmp"

using namespace std;

#define MAX 8

int main ()
{
    LList2 <int> L1;
    for (int i = 0; i < MAX; i++)
        if (i % 2)
            L1.InsertFirst(i);
        else
            L1.InsertLast(i);
    cout << "L1 in order: ";
    cout << L1 << endl;
    cout << "Testing begin, end, post++ and *\n";
}

```

```

    for (LList2<int>::Iterator itr = L1.begin (); itr != L1.end(); itr++)
        cout << "Value: " << *itr << endl;
    return 0;
}

```

7. **Before** compiling and executing the program, predict the output of the program:

8. Compile the program using the following command:

```
g++ -w application.cpp -o lab11a
```

Note: the `-w` option will suppress all warnings.

9. Execute the program. How accurate was your prediction?

10. Add code to the program to test the `rbegin`, `rend`, and `post--` functions:

```

    cout << "Testing rbegin, rend, post-- and *\n";
    for (LList2<int>::Iterator itr = L1.rbegin(); itr != L1.rend(); itr--)
        cout << "Value: " << *itr << endl;

```

Before compiling and executing the program, predict the output of this segment of the program:

Compile and execute the program; how accurate was your prediction?

11. Add code to the program to test the `pre++` function:

```

    cout << "Testing pre++\n";
    cout << "Should write second item in list L1\n";
    List2<int>::Iterator itr = L1.begin();
    cout << *++itr << endl;

```

Before compiling and executing the program, predict the output of this segment of the program:

Compile and execute the program; how accurate was your prediction?

12. Add code to the program to test the `pre--` function:

```

    cout << "Testing pre--\n";
    cout << "Should write second to last item in list L1\n";
    itr = L1.rbegin();
    cout << *--itr << endl;

```

Before compiling and executing the program, predict the output of this segment of the program:

Compile and execute the program; how accurate was your prediction?

13. Copy the files `card.h` and `card.cpp` from the `Lab11` subdirectory in the `cs215pickup` directory. These files contain a class description and implementation for a set of playing cards. Carefully review the class description and its implementation. (Listings of `card.h` and `card.cpp` start on the next page.)
14. Copy the file `cards.in` from `cs215pickup`. This file contains a list of cards in the card class format.

```
AC AD AH AS
2C 2D 2H 2S
3C 3D 3H 3S
4C 4D 4H 4S
```

15. Create a new application that includes `card.h` and your `LList2.tmp`, reads the cards in the input file, stores them in a linked list, and writes out the list of cards. A sample makefile for this program is illustrated below:

```
lab11 : Lab11app.o card.o
      g++ -o lab11 Lab11app.o card.o -g

Lab11app.o : Lab11app.cpp LList2.tmp card.h
      g++ -c Lab11app.cpp -g -w

card.o : card.cpp card.h
      g++ -c card.cpp -g

clean :
      rm -f core.* *.o lab11
```

16. Add an additional segment of code to your program that creates two new linked lists, each holding half of the deck, and writes out the two new lists of cards.
17. Add an additional segment of code to your program that shuffles the two half lists, starting with the first card in the second list, into a new linked list and writes out the shuffled list.
18. For the input file you copied from the pickup folder, the output should be:

```
The cards: AC AD AH AS 2C 2D 2H 2S 3C 3D 3H 3S 4C 4D 4H 4S
```

```
First half: AC AD AH AS 2C 2D 2H 2S
```

```
Second half: 3C 3D 3H 3S 4C 4D 4H 4S
```

```
Shuffled cards: 3C AC 3D AD 3H AH 3S AS 4C 2C 4D 2D 4H 2H 4S 2S
```

18. When you are convinced that your card shuffling program is working correctly, copy your well documented `.cpp` file to the dropbox as `yourlastnameL11.cpp`.

card.h

```
#ifndef CARD_H
#define CARD_H
#include <iostream>
using namespace std;

class card
{
public:
    card ();
    card (const card & other);
    ~card ();
    card & operator = (const card & other);
    bool operator == (const card & other) const;
    bool operator != (const card & other) const;
    bool operator < (const card & other) const;
    bool operator > (const card & other) const;
    bool operator <= (const card & other) const;
    bool operator >= (const card & other) const;
    friend istream & operator >> (istream & ins, card & C);
    friend ostream & operator << (ostream & outs, const card & C);
    char rank;
    char suit;
private:
    void evaluate ();
    int sortval;
};
```

card.cpp

```
#include "card.h"

card::card ()
{
    rank = suit = '?';
    sortval = 0;
}

card::card (const card & other)
{
    rank = other.rank;
    suit = other.suit;
    sortval = other.sortval;
}

card::~~card ()
{
}

card & card::operator = (const card & other)
{
    if (this == &other)
        return * this;
    rank = other.rank;
    suit = other.suit;
    sortval = other.sortval;
    return * this;
}

bool card::operator == (const card & other) const
{
    return sortval == other.sortval;
}
```

```

bool card::operator != (const card & other) const
{
    return sortval != other.sortval;
}

bool card::operator < (const card & other) const
{
    return sortval < other.sortval;
}

bool card::operator > (const card & other) const
{
    return sortval > other.sortval;
}

bool card::operator <= (const card & other) const
{
    return sortval <= other.sortval;
}

bool card::operator >= (const card & other) const
{
    return sortval >= other.sortval;
}

istream & operator >> (istream & ins, card & C)
{
    card T;
    ins >> T.rank;
    ins >> T.suit;
    T.evaluate();
    if (T.sortval)
        C = T;
    else
        ins.setstate(ios::failbit);
    return ins;
}

ostream & operator << (ostream & outs, const card & C)
{
    outs << C.rank << C.suit;
    return outs;
}

void card::evaluate ()
{
    rank = toupper(rank);
    suit = toupper(suit);
    sortval = 0;
    char r[] = {'A','2','3','4','5','6','7','8','9','T','J','Q','K'};
    char s[] = {'C','D','H','S'};
    int i;
    for (i = 0; i < 13 && rank != r[i]; i++);
        sortval = (i+1) * 100;
    for (i = 0; i < 4 && suit != s[i]; i++);
        sortval += i+1;
    if (sortval / 100 < 1 || sortval / 100 > 13 ||
        sortval % 100 < 1 || sortval % 100 > 4)
        sortval = 0;
}

```