

## Laboratory Exercise 10 (Doubly Linked Lists)

Topics: Doubly Linked Lists

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

1. Define a doubly linked list class
2. Define a corresponding node class
3. Insert items into the beginning or end of a doubly linked list
4. Remove items from the beginning or end of a doubly linked list
5. Write an application program that uses a doubly linked list class
6. Write traversal functions using passed functions for a doubly linked list.

Related text sections:

Chapter 16, 17

## Laboratory Exercise 10 Instructions

1. Create a new directory (folder) called Lab10.
2. Copy your LList.tmp file from Lab 9 to a new file called LList2.tmp in your new Lab10 directory. Modify the class description as indicated below. Modify the function implementations as described in lecture (the modified default constructors and the traversal functions are illustrated below).

```
#ifndef LLIST2_TMP
#define LLIST2_TMP

#include <iostream>
using namespace std;

template <class LT> class LList2;

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

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

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

template <class LT>
LList2<LT>::LNode::LNode ()
{
    next = NULL;
    prev = NULL;
}

template <class LT>
LList2<LT>::LList2 ()
{
```

```

    first = NULL;
    last = NULL;
    size = 0;
}
        . . . More modified functions here . . .

template <class LT>
void LList2<LT>::Forward (void function (const LT & param))
{
    for (LNode * n = first; n; n = n->next)
        function (n->data);
}

template <class LT>
void LList2<LT>::Backward (void function (const LT & param))
{
    for (LNode * n = last; n; n = n->prev)
        function (n->data);
}

#endif

```

3. Modify the InsertFirst and InsertLast functions to update the previous links and the pointer to the last node as needed.
4. Modify the DeleteFirst and DeleteLast functions to update the previous links and the pointer to the last node as needed.
5. Create an application program to test the functions of your doubly linked list template class using a standard C++ type for the instantiation.
6. Two new traversal functions have been added to this class: Forward and Backward. These functions are designed to traverse the list and apply a function to the data in each node of the list. To use these functions you will need to create a function in your application program to pass to the traversal. This function will need to have one argument of the same type as your list instantiation. For example, if you are using an integer list:

```
LList2 <int> L;
```

then, the following function could be used when traversing the list:

```

void PrintValue (const int & value)
{
    cout << "The value in the list is " << value << endl;
}

```

To call the Forward traversal, include the command

```
L.Forward (PrintValue);
```

in your application program. As the Forward method progresses through the nodes of L (following the next links, it will call the function PrintValue for each node. The value in data will be passed to the function.

7. Compile and test your new application program.

8. Copy the files Fraction.h, Fraction.cpp, Lab10app.cpp, makefile, and Lab10.in from ~tiawatts/cs215pickup/Lab10. (Lab10app.cpp is listed below; listings of Lab10.in, makefile, Fraction.h, and Fraction.cpp start on the next page.)

#### Lab10app.cpp

```
#include <iostream>
#include <fstream>
#include "LList2.tmp"
#include "Fraction.h"

using namespace std;

// Add a global variable for holding the sum of the fractions here

// Add function prototype here

int main ()
{
    ifstream input ("Lab10.in");
    fraction one;
    LList2 <fraction> FL;

    while (input >> one)
        FL.InsertLast (one);
    cout << "The fractions are: ";
    cout << FL << endl;

// Add code to find the sum of the fractions in the list FL here

// Add code to print the sum here

    return 0;
}

// Add function implementation here
```

9. Add code (where indicated) to find and print the sum of the fractions read from the input file into the list FL.
- You will need to declare a global variable of type fraction to hold the sum of the values.
  - Your function header will need one parameter: a constant reference to an object of type fraction. Your function's return type should be void.
  - Your function should add the fraction passed to it to the global sum.
  - To find the sum of the fractions, you will need to use one of the traversal functions (Forward or Backward) from LList2. You will pass the name of your function to the traversal function.
  - Add code (where indicated) to print the calculated sum stored in your global variable.
10. Compile and execute your program. The output should be:

```
The fractions are: 3 1/2 4 2/3 0 1/2 1 3/4
The total is 10 5/12
```

11. When you are convinced that your LList2 template class is working correctly, copy your LList2.tmp file to the dropbox as yourlastnameL10.tmp.
12. When you are convinced that your program is working correctly, copy your well documented Lab10app.cpp file to the dropbox as yourlastnameL10.cpp.

#### Lab10.in

```
3 1/2
4 2/3
0 1/2
1 3/4
```

#### makefile

```
lab10 : Lab10app.o Fraction.o
        g++ -o lab10 Lab10app.o Fraction.o -g

Lab10app.o : Lab10app.cpp LList2.tmp Fraction.h
        g++ -c Lab10app.cpp -g

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

clean :
        rm -f core.* *.o lab10
```

#### Fraction.h

```
#ifndef FRACTION_H
#define FRACTION_H

#include <iostream>
using namespace std;

class fraction
{
public:
    fraction ();
    fraction (int w);
    fraction (int n, unsigned d);
    fraction (int w, unsigned n, unsigned d);
    ~fraction ();

    fraction operator = (const fraction & other);
    fraction operator + (const fraction & other) const;

    friend istream & operator >> (istream & ins, fraction & f);
    friend ostream & operator << (ostream & outs, const fraction & f);

private:
    void reduce ();

    int whole;
    int numerator;
    unsigned denominator;
};

#endif
```

## Fraction.cpp

```
#include "Fraction.h"

fraction::fraction ()
{
    whole = 0;
    numerator = 0;
    denominator = 1;
}

fraction::fraction (int w)
{
    whole = w;
    numerator = 0;
    denominator = 1;
}

fraction::fraction (int n, unsigned d)
{
    whole = 0;
    numerator = n;
    denominator = d;
    reduce();
}

fraction::fraction (int w, unsigned n, unsigned d)
{
    whole = w;
    numerator = n;
    denominator = d;
    this->reduce();
}

fraction::~fraction ()
{
}

fraction fraction::operator = (const fraction & other)
{
    this->whole = other.whole;
    this->numerator = other.numerator;
    this->denominator = other.denominator;
    this->reduce();

    return *this;
}

fraction fraction::operator + (const fraction & other) const
{
    fraction sum;

    sum.whole = this->whole + other.whole;
    sum.numerator = this->numerator * other.denominator
        + this->denominator * other.numerator;
    sum.denominator = this->denominator * other.denominator;
    sum.reduce();

    return sum;
}
```

```

istream & operator >> (istream & ins, fraction & f)
{
    // Note: trivial version of >>

    char temp;

    ins >> f.whole >> f.numerator >> temp >> f.denominator;

    return ins;
}

ostream & operator << (ostream & outs, const fraction & f)
{
    // Note: trivial version of <<

    outs << f.whole << ' ' << f.numerator << '/' << f.denominator;

    return outs;
}

void fraction::reduce ()
{
    int f = 2;

    denominator = denominator == 0 ? 1 : denominator;
    whole += numerator / denominator;
    numerator %= denominator;
    while (f <= numerator)
        if ((numerator % f == 0) && (denominator % f == 0))
        {
            numerator /= f;
            denominator /= f;
        }
        else
            f++;
}

```

