

## Laboratory Exercise 12 (Exception Handling)

Topics: Overloaded [] operator  
Exception Handling

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

1. Overload the indexing ([]) operator for a template container class.
2. Use the C++ try...throw...catch syntax
3. Incorporate exception handling in a template container class
4. Write a program that incorporates exception handling

Related text sections:  
Chapter 18

## Laboratory Exercise 12 Instructions

1. Copy your Llist2.tmp file from Lab 11 to a new Lab 12 directory.
2. Add the following prototype and draft implementation to your LList2 template class:

```
LT & operator [] (const int & index) const;

template <class LT>
LT & LList2<LT>::operator [] (const int & index) const
{
    LList2<LT>::LNode * n = first;
    return n->data;
}
```

*Note that the function operator[] returns a reference type. This causes it to be an l-value, allowing you to use subscripted expressions on either side of assignment operators.*

The complete description of the LList2 template is at the end of this Laboratory Exercise.

3. Enter the following program in a file called application.cpp.

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

using namespace std;

#define MAX 10

int main ()
{
    LList2 <int> L1, L2;
    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 the [] operator\n";
    cout << "First to last:\n";
    for (int i = 0; i < L1.Size(); i++)
        cout << "L1[" << i << "] is " << L1[i] << endl;
    L1[3] = 7;
    cout << "Last to first:\n";
    for (int i = L1.Size()-1; i >= 0; i--)
        cout << "L1[" << i << "] is " << L1[i] << endl;
    return 0;
}
```

4. Compile and execute the program. Are the results what you expected from the code in the application program?
5. Modify the [] operator function to traverse through the list until it reaches the entry corresponding to the position in the index parameter. The operator should then return the data item in the selected entry.
6. Test your program again. The correct output should be:

```

First to last:
L1[0] is 9
L1[1] is 7
L1[2] is 5
L1[3] is 3
L1[4] is 1
L1[5] is 0
L1[6] is 2
L1[7] is 4
L1[8] is 6
L1[9] is 8
Last to first:
L1[9] is 8
L1[8] is 6
L1[7] is 4
L1[6] is 2
L1[5] is 0
L1[4] is 1
L1[3] is 7
L1[2] is 5
L1[1] is 7
L1[0] is 9

```

7. If you did not get the correct output, review the code in your [] operator and modify it.
8. Currently the dereferencing operator in your template can only be used on the Right Hand Side (rhs) of an assignment statement. Edit the dereferencing operator (\*) for the iterator in LList2.tmp to allow it to be used on the Left Hand Side (lhs) of an assignment statement:

The prototype should be:

```
LT & operator * () const;
```

The implementation should be:

```

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

```

9. Add statements to application.cpp to test the dereferencing operator as both a lhs and as a rhs operator.
10. Copy the C++ files in ~tiawatts/cs215pickup/Lab12 to your Lab 12 directory.



18. Are you now getting the expected output? If not, continue to debug the .cpp file until the program executes correctly.
19. Compile the file Lab12c.cpp.
20. **Before running the program** carefully read the source code and predict the output if the value calculated for “num” is 11 and the value calculated for “val” is 5.

21. Execute the program. Executing this program should result in a Segmentation Fault. Using gdb, determine where the program crashed and modify the .cpp file to eliminate the crash. Describe the problem:

22. Are you now getting the expected output? If not, continue to debug the .cpp file until the program executes correctly.

23. Modify the \*, ++, --, and [] operators for the Iterator class to catch exceptions using C++ try-throw-catch structures. Each of these operators should pass an appropriate error message to the catch and should then exit from the program. Each exit command should have a unique numeric identifier argument. For example:

```
template <class LT>
LT & LList2<LT>::Iterator::operator * () const
{
    try
    {
        if (current == NULL)
            throw ("Cannot dereference a NULL pointer");
        return current->data;
    }
    catch (const char * message)
    {
        cerr << message << endl;
        exit (1);
    }
}
```

24. When you are convinced that your Linked List template is working correctly, copy your well documented .tmp file to the dropbox as yourlastnameL12.tmp.

```

template <class LT> class LList2;

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

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

public:
    typedef class Iterator
    {
    public:
        Iterator ();
        Iterator (LNode * NP);
        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;
    LT & operator [] (const int & index) const;
private:
    LNode * first;
    LNode * last;
    int size;
};

```