

Laboratory Exercise 8 (Dynamic memory allocation)

Topics: Dynamic Array Lists
Singly Linked Lists

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

1. Maintain a list class in a dynamic array
2. Insert items into the beginning or end of a list
3. Remove items from the beginning or end of a list
4. Resize a list
5. Write an application program that uses a dynamic array list class
6. Initialize a singly linked list class (of integers)
7. Traverse a singly linked list
8. Insert a node at the front of a singly linked list
9. Insert a node at the end of a singly linked list
10. Remove a node from the front of a singly linked list
11. Remove a node from the end of a singly linked list
12. Write an application program that uses a singly linked list class

Related text sections:

Chapter 10
Chapter 17

Laboratory Exercise 8 Instructions

Part 1.

1. Logon to the linux system. Create a subdirectory called Lab8.1
2. Copy the 4 files from ~tiawatts/cs215pickup/Lab8.1 to your new subdirectory.
3. List the files you copied. You should see makefile, AList.h, AList.cpp, and application.cpp,.

makefile:

```
lab8 : application.o AList.o
      g++ -o lab8 application.o AList.o -g

application.o : application.cpp AList.h
      g++ -c application.cpp -g

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

clean :
      rm -f core.* *.o lab8
```

AList.h

```
#ifndef ALIST_H
#define ALIST_H

#define MIN_SIZE 10
#define GROW_FACTOR 1.25
#define SHRINK_LIMIT 0.5
#define SHRINK_FACTOR 0.8

#include <iostream>
using namespace std;

class AList
{
public:
    AList ();
    AList (const AList & other);
    ~AList ();
    AList & operator = (const AList & other);
    bool AList::operator == (const AList & other);
    int Capacity () const;
    int Used () const;
    friend ostream & operator << (ostream & outs, const AList & L);
    bool Insert (const int & value);
    bool Delete (const int & value);
private:
    int * data;
    int capacity;
    int used;
};
#endif
```

AList.cpp

```
#include "AList.h"

AList::AList ()
{
    data = new int [MIN_SIZE];
    capacity = MIN_SIZE;
    used = 0;
}

AList::AList (const AList & other)
{
    data = new int [other.capacity];
    for (int i = 0; i < other.used; i++)
        data[i] = other.data[i];
    capacity = other.capacity;
    used = other.used;
}

AList::~AList ()
{
    delete [] data;
}

AList & AList::operator = (const AList & other)
{
    if (this == &other)
        return * this;
    delete [] data;
    data = new int [other.capacity];
    for (int i = 0; i < other.used; i++)
        data[i] = other.data[i];
    capacity = other.capacity;
    used = other.used;
    return * this;
}

bool AList::operator == (const AList & other)
{
    if (used != other.used)
        return false;
    for (int i = 0; i < used; i++)
        if (data[i] != other.data[i])
            return false;
    return true;
}

int AList::Capacity () const
{
    return capacity;
}

int AList::Used () const
```

```

    {
        return used;
    }

ostream & operator << (ostream & outs, const AList & L)
{
    if (L.used)
        outs << L.data[0];
    for (int i = 1; i < L.used; i++)
        outs << ' ' << L.data[i];
    return outs;
}

bool AList::Insert (const int & value)
{ // This is a stubb; you will be completing this function.

    return false;
}

bool AList::Delete (const int & value)
{ // This is a stubb; you will be completing this function.
    return false;
}

```

4. You will need to complete the implementation of the Insert function. This function should:
 - a. Test to see if there is enough room in the data array to hold a new value.
 - b. If there is not enough room, the data array will need to be resized. The new capacity of the array should be the product of the current capacity and the GROWTH_FACTOR.
 - c. If the data array is being resized, if the system cannot allocate enough space, false should be returned.
 - d. Locate the position where the new value should be stored to maintain ascending order.
 - e. Move the array entries following the position back one space each so that there is a space for the new value.
 - f. Insert the new value in the array in the desired position.
 - g. Increment the used count.
 - h. Return true if the insertion is successful.

5. You will need to complete the implementation of the Delete function. This function should:
 - a. Search for the input value in the data array.
 - b. If the value is not in the data array, return false.
 - c. If the value is in the list, remove it from the list by moving the data items following the value. If there is more than one occurrence of the value in the list, just remove the first one.
 - d. Decrement the used counter.
 - e. If less than SHRINK_LIMIT of the capacity is currently in use, the data array will need to be resized. The new capacity of the array should be the maximum of MIN_SIZE or the product of the current capacity and the SHRINK_FACTOR.
 - f. If the data array is being resized, if the system cannot allocate enough space, false should be returned.
 - g. Return true if the deletion is successful.

application.cpp

```
// Lab 8 Part 1 application program
// Author: Dr. Watts
// Data: Spring 2009
// Description: This program is designed to test the AList class.

#include <iostream>
#include <fstream>
#include "AList.h"

using namespace std;

int main (int argc, char * argv[])
{
    if (argc < 2)
    {
        cerr << "Usage: " << argv[0] << " <file-name>\n";
        exit (1);
    }
    ifstream ins (argv[1]);
    if (ins.fail ())
    {
        cerr << "No file " << argv[1] << " found\n";
        exit (2);
    }

    AList AL;
    char action;
    int value;

    while (ins >> action)
        switch (action)
        {
            case 'C' : cout << AL.Capacity() << endl;
                       break;
            case 'D' : ins >> value;
                       if (AL.Delete (value))
                           cout << value << " deleted from list\n";
                       else
                           cout << value << " not deleted from list\n";
                       break;
            case 'I' : ins >> value;
                       if (AL.Insert (value))
                           cout << value << " inserted in list\n";
                       else
                           cout << value << " not inserted in list\n";
                       break;
            case 'U' : cout << AL.Used() << endl;
                       break;
            case 'W' : cout << AL << endl;
                       break;
            default : cout << action << " is not a valid option\n";
        }
    return 0;
}
```

6. The application program provides a framework for testing this class. You will need to create an input file to drive the testing process. A simple input file might simply place a value in the list, check the list's capacity and its used count and print the list:


```
I 10
C
U
W
```
7. By adding new commands to the input file, test the accuracy of your Insert and Delete functions.
8. Have all of the functions in the class been tested? Add a cout statement at the beginning of each function to verify that it is actually being called. If any of the functions are not being called, add additional actions to the application program.
9. When you are convinced that your class is working correctly, remove the cout statements from your AList function implementations and copy your AList.cpp file to the dropbox as yourlastnameL8-1.cpp.

Part 2.

1. Create a subdirectory called Lab8.2
2. Copy the 2 files from ~tiawatts/cs215pickup/Lab8.2 to your new subdirectory.
3. List the files you copied. You should see makefile and application.cpp.

makefile

```
lab8 : application.o LList.o
      g++ -o lab8 application.o LList.o -g

application.o : application.cpp LList.h
      g++ -c application.cpp -g

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

clean :
      rm -f core.* *.o lab8
```

4. You will be using a text editor to create the files LList.h and LList.cpp.

5. Enter the following class description into a file called LList.h:

```
#ifndef LLIST_H
#define LLIST_H

#include <iostream>
using namespace std;

class LList
{
    private:
        class LNode
        {
            public:
                LNode ();
                int data;
                LNode * next;
        };

    public:
        LList ();
        LList (const LList & other);
        ~LList ();
        LList & operator = (const LList & other);
        bool operator == (const LList & other);
        int Size () const;
        friend ostream & operator << (ostream & outs, const LList & L);
        bool InsertFirst (const int & value);
        bool InsertLast (const int & value);
        bool DeleteFirst ();
        bool DeleteLast ();
    private:
        LNode * first;
        int size;
};

#endif
```

6. Enter the following class description into a file called LList.cpp:

```
#include "LList.h"

LList::LNode::LNode ()
{
    data = 0;
    next = NULL;
}

LList::LList ()
{
    first = NULL;
    size = 0;
}
```

```

LList::LList (const LList & other)
{
    first = NULL;
    size = 0;
    for (LNode * n = other.first; n != NULL; n = n->next)
        InsertLast (n->data);
}

LList::~LList ()
{
    while (first)
        DeleteFirst();
}

LList & LList::operator = (const LList & other)
{
    if (this == &other)
        return * this;
    while (first)
        DeleteFirst();
    for (LNode * n = other.first; n != NULL; n = n->next)
        InsertLast (n->data);
    return * this;
}

bool LList::operator == (const LList & other)
{
    if (size != other.size)
        return false;
    LNode * n = first;
    LNode * m = other.first;
    while (n != NULL)
    {
        if (n->data != m->data)
            return false;
        n = n->next;
        m = m->next;
    }
    return true;
}

int LList::Size () const
{
    return size;
}

ostream & operator << (ostream & outs, const LList & L)
{
    if (L.first == NULL)
        return outs;
    outs << L.first->data;
    for (LList::LNode * n = L.first->next; n != NULL; n = n->next)
        outs << ' ' << n->data;
    return outs;
}

```

```

bool LList::InsertFirst (const int & value)
{ // This is a stubb; you will be completing this function.
  return false;
}

bool LList::InsertLast (const int & value)
{ // This is a stubb; you will be completing this function.
  return false;
}

bool LList::DeleteFirst ()
{ // This is a stubb; you will be completing this function.
  return false;
}

bool LList::DeleteLast ()
{ // This is a stubb; you will be completing this function.
  return false;
}

```

6. Enter the following data into a file called test.in:

```

I 5
J 10
I 30
J 40
I 8
W
S
J -3
I 17
J 24
I 35
W
S
D
E
D
E
D
E
D
W
S

```

7. After carefully reading the application program:

```
// Lab 8 Part 2 application program
// Author: Dr. Watts
// Data: Spring 2009
// Description: This program is designed to test the LList class.

#include <iostream>
#include <fstream>
#include "LList.h"

using namespace std;

int main (int argc, char * argv[])
{
    if (argc < 2)
    {
        cerr << "Usage: " << argv[0] << " <file-name>\n";
        exit (1);
    }
    ifstream ins (argv[1]);
    if (ins.fail ())
    {
        cerr << "No file " << argv[1] << " found\n";
        exit (2);
    }
    LList LL;
    char action;
    int value;

    while (ins >> action)
        switch (action)
        {
            case 'D' : if (LL.DeleteFirst ())
                        cout << "First value deleted from list\n";
                    else
                        cout << "First value not deleted from list\n";
                    break;
            case 'E' : if (LL.DeleteLast ())
                        cout << "Last value deleted from list\n";
                    else
                        cout << "Last value not deleted from list\n";
                    break;
            case 'I' : ins >> value;
                    if (LL.InsertFirst (value))
                        cout << value << " inserted at front of list\n";
                    else
                        cout << value << " not inserted at front of list\n";
                    break;
            case 'J' : ins >> value;
                    if (LL.InsertLast (value))
                        cout << value << " inserted at end of list\n";
                    else
                        cout << value << " not inserted at end of list\n";
                    break;
            case 'S' : cout << LL.Size() << endl;
                    break;
            case 'W' : cout << LL << endl;
                    break;
            default : cout << action << " is not a valid option\n";
        }
    return 0;
}
```

7a. Predict the output of the program for test.in after the class has been completed:

7b. Predict the output of the program for test.in with the class in its current state:

8. Compile (make) and execute the the program using test.in as the input file:

```
lab8 test.in
```

were your predictions for 7b correct?

9. Implement the InsertFirst function. This function should:
 - a. Allocate space for a new node.
 - b. Return false if space cannot be allocated for the node.
 - c. Set the data of the new node to the input value
 - d. Set the next of the new node to the current first.
 - e. Set the first of the list to the new node.
 - f. Increment the size of the list.
 - g. Return true.
10. Compile and test the program to verify that your InsertFirst function is working correctly.
11. Implement the DeleteFirst function: This function should:
 - a. Return false if the list is empty (first == NULL OR size == 0).
 - b. Save the first pointer in a temporary variable.
 - c. Change first to the current first's next.
 - d. Delete the node that was first (its address is saved in your temporary variable.)
 - e. Decrement the size of the list.
 - f. Return true.
12. Compile and test the program to verify that your DeleteFirst function is working correctly.
13. Implement the InsertLast function. This function should:
 - a. If the list is empty, call InsertFirst and return its result.
 - b. Allocate space for a new node.
 - c. Return false if space cannot be allocated for the node.
 - d. Set the data of the new node to the input value
 - e. Set the next of the new node to NULL.
 - f. Traverse to the last node in the current list; save the pointer to the last node in a temporary variable.
 - g. Set the next of the last node to the new node.
 - h. Increment the size of the list.
 - i. Return true.
14. Compile and test the program to verify that your InsertLast function is working correctly.
15. Implement the DeleteLast function: This function should:
 - a. Return false if the list is empty (first == NULL OR size == 0).
 - b. If there is only one node in the list (size == 1), call DeleteFirst and return its result
 - c. Traverse to the 2nd to last node in the list..
 - d. Save the pointer to the last node in a temporary variable.
 - e. Change the next of the 2nd to last item to NULL..
 - f. Delete the node that was last (its address is saved in your temporary variable.)
 - g. Decrement the size of the list.
 - h. Return true.
16. Compile and test the program to verify that your DeleteLast function is working correctly.
17. When you are convinced that your LList class is working correctly, copy your LList.cpp file to the dropbox as yourlastnameL8-2.cpp.