Lab6
Linked List

Sheharyar Khan
CS-116

## Source Code

```
#include <iostream>
#include <typeinfo>
#include <iomanip>
#include <string>
#include <ctime>

using namespace std;

/*
 Purpose- Print out the programmer's information such as name,
 * class information, and date/time when the program is run

 @author Ron Sha
 @version 1.0 1/1/2016
 @param name- the name of the programmer
 @param courseInfo- the name of the course
 @return- none
*/

void printMeFirst(string name, string courseInfo)
{
     cout << "Program written by: " << name << endl;
     cout << "Course Info: " << courseInfo << endl;
     time_t now = time(0);
     char* dt = ctime(&now);
     cout << "Date: " << dt << endl;
}


template< typename T > class List; // forward declaration

template< typename T  >
class ListNode
{

public:
  friend class List< T >; // make List a friend
  ListNode( const T & ); // constructor
  T getData() const; // return the data in the node

  // set nextPtr to nPtr
  void setNextPtr( ListNode *nPtr )
  {
     nextPtr = nPtr;
  } // end function setNextPtr

  // return nextPtr
  ListNode *getNextPtr() const
  {
     return nextPtr;
  } // end function getNextPtr

private:
  T data; // data
  int key; // used for key for the list
  ListNode *nextPtr; // next node in the list
}; // end class ListNode

// constructor
template< typename T >
ListNode< T >::ListNode( const T &info )
{
  data = info;
```

```
  nextPtr = NULL;
} // end constructor

// return a copy of the data in the node
template< typename T >
T ListNode< T >::getData() const
{
  return data;
} // end function getData


template< typename T  >
class List
{
public:
  List(); // default constructor
  List( const List< T > & ); // copy constructor
  ~List(); // destructor

  void insertAtFront( const T & );
  void insertAtBack( const T & );
  bool removeFromFront( );
  bool removeFromBack( );
  bool isEmpty() const;
  void print() const;
  void printPtrFunc(   );
  T * getInfo(int myKey);
     // return nextPtr
  ListNode< T >  *getFirstPtr() const
  {
     return firstPtr;
  } // end function getNextPtr

protected:
  ListNode< T > *firstPtr; // pointer to first node
  ListNode< T > *lastPtr; // pointer to last node

  // Utility function to allocate a new node
  ListNode< T > *getNewNode( const T &);
}; // end class template List

// default constructor
template< typename T >
List< T >::List()
{
  firstPtr = lastPtr = NULL;
} // end constructor

// copy constructor
template< typename T >
List< T >::List( const List<T> &copy )
{
  firstPtr = lastPtr = NULL; // initialize pointers

  ListNode< T > *currentPtr = copy.firstPtr;

  // insert into the list
  while ( currentPtr != NULL )
  {
     insertAtBack( currentPtr->data );
     currentPtr = currentPtr->nextPtr;
  } // end while
} // end List copy constructor

// destructor
template< typename T >
```

```
List< T >::~List()
{
  if ( !isEmpty() ) // List is not empty
  {
//      cout << "Destroying nodes ...\n";

     ListNode< T > *currentPtr = firstPtr;
     ListNode< T > *tempPtr;

     while ( currentPtr != NULL ) // delete remaining nodes
     {
        tempPtr = currentPtr;
//         cout << tempPtr->data << ' ';
        currentPtr = currentPtr->nextPtr;
        delete tempPtr;
     } // end while
  } // end if

//   cout << "\nAll nodes destroyed\n\n";
} // end destructor

// Insert a node at the front of the list
template< typename T >
void List< T >::insertAtFront( const T &value)
{
  ListNode<T> *newPtr = getNewNode( value);

  if ( isEmpty() ) // List is empty
     firstPtr = lastPtr = newPtr;
  else // List is not empty
  {
         /*
          * write code to implement insert at front
          * 1. the new node needs to point to the first node
          * 2. first node needs to point to the new node
          * */

     newPtr->nextPtr = firstPtr;
     firstPtr = newPtr;


  } // end else
} // end function insertAtFront

// Insert a node at the back of the list
template< typename T >
void List< T >::insertAtBack( const T &value)
{
  ListNode< T > *newPtr = getNewNode( value);

  if ( isEmpty() ) // List is empty
     firstPtr = lastPtr = newPtr;
  else // List is not empty
  {

          /*
          * write code to implement insert at back
          * 1. next pointer of the last node points to the new node
      * 2. last node needs to point to the new node*/


/* ??????
     lastPtr ?????
     lastPtr = ??????
*/
     lastPtr->nextPtr=newPtr;//setting the next pointer of the last pointer to the new node
```

```
      lastPtr=newPtr;//setting the last pointer equal to new pointer
   } // end else
} // end function insertAtBack

// Delete a node from the front of the list
template< typename T >
bool List< T >::removeFromFront(  )
{
   if ( isEmpty() ) // List is empty
      return false; // delete unsuccessful
   else
   {
      ListNode< T > *tempPtr = firstPtr;

      /*
       * check to see if first pointer is the same as last point
       * if it is the same, that means there is only one node, so
       * set both pointer (first and last) to NULL
       * if it is not the same, that means there is more than one node
       * in the linked list.  So, make the the first node pointer points
       * to the the next node of the first node
       *
       * */
      if ( firstPtr == lastPtr )
         firstPtr = lastPtr = NULL;
      else
         firstPtr = firstPtr->nextPtr;
      delete tempPtr;
      return true; // delete successful
   } // end else
} // end function removeFromFront

// delete a node from the back of the list
template< typename T >
bool List< T >::removeFromBack(  )
{
   if ( isEmpty() )
      return false; // delete unsuccessful
   else
   {
      ListNode< T > *tempPtr = lastPtr;

      if ( firstPtr == lastPtr )
         firstPtr = lastPtr = NULL;
      else
      {
         ListNode< T > *currentPtr = firstPtr;

        /*
         * while next node of currentPtr is NOT the same as lastPtr node
         *  - then change currentPtr to next node of currentPtr
         *
         while ( ????? )
            ???????

         once you found the node before the lastPtr, then you need to
         assign lastPtr to the currentPtr and set next node of
         currentPtr to NULL
         ?????
         ????

         */
         while(currentPtr->nextPtr!=lastPtr){//while next node of currentPtr is NOT the same as
lastPtr node
            currentPtr=currentPtr->nextPtr;//then change currentPtr to next node of currentPtr
         }
```

```
        lastPtr=currentPtr;//you need to assign lastPtr to the currentPtr
        currentPtr->nextPtr=NULL;//and set next node of currentPtr to NULL

     } // end else

     delete tempPtr;
     return true; // delete successful
  } // end else
} // end function removeFromBack

// Is the List empty?
template< typename T >
bool List< T >::isEmpty() const
{
  // return true if the List is empty otherwise return false
  // ????????
  if(firstPtr==0 && lastPtr==0){//returns true if the first and last pointer equal 0/NULL
     return true;
  }
  else{//otherwise returns false
     return false;
  }

} // end function isEmpty

// Return a pointer to a newly allocated node
template< typename T >
ListNode< T > *List< T >::getNewNode(
  const T &value)
{
  ListNode< T > *ptr = new ListNode< T >( value );
  return ptr;
} // end function getNewNode

// Display the contents of the List
template< typename T >
void List< T >::print() const
{
  if ( isEmpty() ) // empty list
  {
     cout << "The list is empty\n\n";
     return;
  } // end if

  ListNode< T > *currentPtr = firstPtr;

  cout << "The list is: ";

  while ( currentPtr != NULL ) // display elements in list
  {
      int i;
      string s;
      double d;
      char c;
      /*
       * check to make the data type is the same
       * */
      if (typeid(currentPtr->data).name() == typeid(i).name() ||
          typeid(currentPtr->data).name() == typeid(d).name() ||
          typeid(currentPtr->data).name() == typeid(s).name() ||
          typeid(currentPtr->data).name() == typeid(c).name())
      {
       // data value is a simple data type and can be printed
       cout << currentPtr->data << ' ';
      }
      else {
```

```
        cout <<"Can't print - Not a simple data type (int, string, char, double)\n";
      }
     currentPtr = currentPtr->nextPtr;
  } // end while

  cout << "\n\n";
} // end function print


class Wine
{
public:
  Wine();
  Wine(string name, int vintage, int score, double price, string type);
  void setInfo(string name, int vintage, int score,
      double price, string type);
  void setPrice(double price);
  string getName() const;
  int getPrice() const;
  void printInfo();

private:
  string name;
  int vintage;
  int score;
  double price;
  string type;
};

Wine::Wine()
{
   price = 0;
}

Wine::Wine(string name, int vintage, int score, double price, string type)
{
   this->name = name;
   this->vintage = vintage;
   this->score = score;
   this->price = price;
   this->type = type;
}

void Wine::setInfo(string name, int vintage, int score, double price, string type)
{
   this->name = name;
   this->vintage = vintage;
   this->score = score;
   this->price = price;
   this->type = type;
}

void Wine::setPrice(double price)
{
   this->price = price;
}

string Wine::getName() const
{
   return name;
}

int Wine::getPrice() const
{
   return price;
}
```

```
void Wine::printInfo()
{
   cout <<" Wine: " << name << " / " << type;
   cout <<" / $" << price << " / Score: " << score << "/ Year: "
            << vintage << endl;
}



class Person
{
public:
  Person();
  Person(string pname, int page);
  void set_name(string n) {name = n;};
  void set_age(int a) {age = a;};
  void set_info(string n, int a) {name = n; age=a;};
  string get_name() const;
  int get_age() const;
  void printInfo() { cout <<"Name: "<<name;
                     cout << "\tAge: "<<age<<endl; };
private:
  string name;
  int age; /* 0 if unknown */
};

Person::Person()
{
}

Person::Person(string pname, int page)
{
  name = pname;
  age = page;
}

string Person::get_name() const
{
  return name;
}

int Person::get_age() const
{
  return age;
}


template< typename T >
void printNoteInfo (  List< T > & nodeList)
{
  T *wp;
  ListNode< T > *currentPtr;

  currentPtr =  nodeList.getFirstPtr(); // point to 1st node

//  cout << "The node list is: \n";
  //print out all the info in linked list
  while ( currentPtr != NULL ) // display elements in list
  {
     wp = (T *) currentPtr; //convert to correct data type
     wp->printInfo(); // calling print info function
     currentPtr = currentPtr->getNextPtr();  // move to next node
  } // end while
}
```

```
/*
Test of generic linked list implementation to handle all
data types including class data types using class template

Also handles using a pointer to call a print function
*/


int main()
{
  List< int > list1; // storage for first list
  Person p;
  List< Person > personList;

  Wine w;
  List< Wine > wineList;

  // ??????  change "Ron Sha" to your name
  printMeFirst("Sheharyar Khan", "CS-116 2018FA");

  p.set_info("Sherry", 20);
  /*
  cout << p.get_name() << " is " <<
     p.get_age() << " years old.\n";
  */
  personList.insertAtFront( p);
  p.set_info("Sha", 30);
  personList.insertAtBack( p );

  /*
   * the printPersonInfo is customized print function only for
   * Person data type, but it can't print any other data type
   * */

 // printPersonInfo(personList);
  /*
   * Using template to print the linkedlist so it works will all
   * data type
   * */
  cout << "Using template to print Person linked list \n" ;

  printNoteInfo (personList);


  w.setInfo("Prisoner", 2014, 92, 44.99, "Red");
  wineList.insertAtBack(w);
  w.setInfo("Vermentino", 2014, 85, 27, "White");
  wineList.insertAtFront(w);
  w.setInfo("Stags Chardonnay Carneros", 2013, 89, 45, "White");
  wineList.insertAtBack(w);
  w.setInfo("Castello Reserve Cabernet", 2011, 92, 92, "Red");
  wineList.insertAtBack(w);
  w.setInfo("Harlan Estate Bordeaux", 2011, 97, 850, "Red");
  wineList.insertAtFront(w);
  w.setInfo("Futo Bordeaux Red", 2009, 97, 324.99, "Red");
  wineList.insertAtBack(w);

  // printWineInfo (wineList);

  /*
   * Using template to print the linkedlist so it works will all
   * data type
   * */
  cout << "Using template to print W lineinked list \n" ;

  printNoteInfo (wineList);
```

```
  // assign intger into first list, from 1 to 3
  for (int i=0; i <3;  i++)
     list1.insertAtFront( i );

  // call function print to print the list
  list1.print();

  // assign from 3 to 5 into second list
  for (int i=3; i<5;   i++ )
     list1.insertAtBack( i);

  list1.print(); //print generic data type
  list1.removeFromBack();
  cout <<"After remove last node \n";
  list1.print();

} // end main
```

**Purpose**

The purpose of this lab was to practice and understand Linked List. We were supposed to read the entire code and clarify our understanding of the code. I didn't really code much in this lab. I wrote like 12 lines of code (in blue). Most of the code was provided to us. And the comments explain it really well.

**Logic**

The *InsertAtBack* function checks if the list is empty. If it is it sets the first node and the last node equal to the new node. If it isn't then it iterates through the list until it gets to the last node then makes it point to the new node and make the new node the the last node.

The *RemoveFromBack* function checks if the list is empty. If it is then it say the list is empty. If it isn't then it iterates through the list all the way to the second last pointer. Makes it point to NULL and sets it equal to the last node. Then deletes the last node.

The *IsEmpty* function returns true when the first and last node are equal to NULL meaning there are no nodes in the file. Otherwise it returns false.

**Test Case**

```
Program written by: Sheharyar Khan
Course Info: CS-116 2018FA
Date: Mon Dec 03 19:32:54 2018

Using template to print Person linked list
Name: Ron       Age: 20
Name: Sha       Age: 30
Using template to print W lineinked list
 Wine: Harlan Estate Bordeaux / Red / $850 / Score: 97/ Year: 2011
 Wine: Vermentino / White / $27 / Score: 85/ Year: 2014
 Wine: Prisoner / Red / $44.99 / Score: 92/ Year: 2014
 Wine: Stags Chardonnay Carneros / White / $45 / Score: 89/ Year: 2013
 Wine: Castello Reserve Cabernet / Red / $92 / Score: 92/ Year: 2011
 Wine: Futo Bordeaux Red / Red / $324.99 / Score: 97/ Year: 2009
The list is: 2 1 0

The list is: 2 1 0 3 4

After remove last node
The list is: 2 1 0 3
```