Project-3: Santa's Lists
F15 CSCI 232 - Data Structures and Algorithms

Phillip J. Curtiss, Assistant Professor
Computer Science Department, Montana Tech
Museum Building, Room 105

October 14, 2015

Project-3: Due 2015-10-26 by midnight

Purpose: Santa is getting ready for the upcoming Holiday Season. He has been spending all year long creating lists of those who are naughty and those who are nice. These lists can be long and difficult to manage. He would desperately like for a program where he could load these list data items, in such a way that they are sorted by country and postal code so he can then print out the list in this ordered fashion on the Big Day. The output should also include the list of gifts for each nice girl and boy, and the single gift of coal for all the naughty children.

You are provided Axioms for the ADT List along with pre- and postconditions. You must provide a linked list implementation of the ADT List that conforms with the specifications (the Axioms and pre- and postconditions) and provides the behavior of the ADT List and functions as described below.

Objectives:

- Using UML Diagrams to Specify ADT
- Using existing C++ class methods
- Create methods that implement the ADT List
- Manipulate Lists of Complex Data Types
- Manipulate Linked List Structures
- Work with Pointers in C++
- Pass objects by reference
- Use C++ source file standards and doxygen to generate documentation
- Modify and add features to the provided main program

Obtaining the Project Files: You will complete the project using your own user account on the department's linux system `katie.mtech.edu`. You should ssh into your account, and execute the command `mkdir -p csci232/proj3`. This will make the directory `proj3` inside the directory `csci232`, and also create the directory `csci232` if it does not yet exist - which it should from our previous lab session. You should then change the current working directory to proj3 by executing the command `cd csci232/proj3`. You can test that you are in the correct current working directory by executing the command `pwd` which should print something like `/home/students/<username>/csci232/proj3`, where `<username>` is replaced with the username associated with your account. Lastly, execute the command `tar -xzvf ~curtiss/csci232/proj3.tgz` and this will expand the project files from the archive into your current working directory. You may consult `man tar` to learn about the archive utility and its command line options.

Table 1: UML Specification of ADT List

```
+isEmpty(): boolean
+getLength(): integer
+insert(newPosition: integer, newEntry: ItemType): boolean
+remove(position: integer): boolean
+clear(): void
+getEntry(position: integer): ItemType
+setEntry(position: integer, newEntry: ItemType): void
+swap(positionA: integer, positionB: interger): boolean
+reverse(): void
+replace(position: integer, newEntry: ItemType): boolean
+getPosition(entry: ItemType): integer
+contains(entry: ItemType): boolean
+loadList(string: filename): boolean
+displayList(): void
```

Project Function: Your program will be provided different test files to read where each line of the file contains the following items in the following structure:

Token-0: List name 'naughty' 'nice'

Token-1: 2-character country code

Token-2: Integer-based postal code

Token-3: Last Name of Girl or Boy

Token-4: First Name of Girl or Boy

Token-5...n: List of Single Word Strings representing gift(s)

An extract of such a file:

```
nice us 59701 coe doug chemistry_set hawaiian_shirt baseball_cap
naughty ru 129 potin vladamir coal
```

You will need to instantiate multiple lists, each with a structure where each item in the list should contain the following:

- Country Code
- Postal Code
- Last Name
- First Name
- A List - of gift(s)

You need to add and remove items from the list in a way that ensures the items in the list for a given set of country codes are clustered in a contiguous group, and the items within this group are ordered by their postal code. When the list is displayed, the output should be a report consisting of multiple rows of the form:

Country:	us	Postal Code:	59701	Last Name:	coe	First Name:	Doug
Gifts:	chemistry_set hawaiian_shirt baseball_cap ...						

Table 2: Axioms for List ADT

Axiom 1. *(**new** List()).isEmpty() = **true***

Axiom 2. *(**new** List()).getLength() = 0*

Axiom 3. *aList.getLength() = (aList.insert(i, x)).getLength() − 1*

Axiom 4. *aList.getLength() = (aList.remove(i)).getLength() + 1*

Axiom 5. *(aList.insert(i, item)).isEmpty() = **false***

Axiom 6. *(**new** List()).remove(i) = **false***

Axiom 7. *(aList.insert(i, x)).remove(i) = aList*

Axiom 8. *(**new** List()).getEntry(i) = error*

Axiom 9. *(aList.insert(i, x)).getEntry(i) = x*

Axiom 10. *aList.getEntry(i) = (aList.insert(i, x)).getEntry(i + 1)*

Axiom 11. *aList.getEntry(i + 1) = (aList.remove(i)).getEntry(i)*

Axiom 12. *(**new** List()).setEntry(i, x) = error*

Axiom 13. *(aList.setEntry(i, x)).getEntry(i) = x*

The list of Axioms for the List ADT should not be violated in your implementation and you should definitely take care to make use of exception handling throughout your implementation to assist your program in recovering gracefully from errors.

You are required to use Javadoc-style comments so doxygen can be used to create html or pdf documentation from your code. Examples of complaint comments are provided throughout the supplied source files. For further doxygen documentation see the site `http://doxygen.org`.

Building the Project: The project includes a `Makefile` you may use to generate the object and executable files from source code files. Similarly, some of the source code files can be generated from the UML diagram that is also included. Consult the `Makefile` and understand the rules included and the dependencies and the rule sets that are used to generate the executable program. Use caution when updating the `Makefile` to ensure rule sets make sense.

Helpful Reminders: Study and pay close attention to the provided class(es) and methods. Understand their return types and use them in the code you author to provide robust code that can handle exceptions to inputs and boundary conditions. Look at all the code provided. Read the codes's comments and implement what is required where indicated. The feedback provided to the user may be poor in certain areas of the main driver program. Fix this and provide appropriate feedback to the end-user to inform them of what the program is doing. Make sure the user confirms actions that result in modifications to the contents of their bag. Be cognizant of the *best practices we discussed in lecture and abide by good coding style - all of which will be factored into the assessment and grade for this project.* Be sure to review the UML diagram and the `Makefile` and understand how files are being generated and their dependencies.

Submission of Project: You have been provided a `Makefile` for this project that will help you not only build your project, but also submit the project in the correct format for assessment and grading. Toward the bottom of the provided `Makefile` you should see lines that look like:

```
# Rule to submit programming assignments to graders
```

```
# Make sure you modify the $(subj) $(msg) above and the list of attachment
# files in the following rule - each file needs to be preceeded with an
# -a flag as shown
subj    = "CSCI232 DSA - Proj3"
msg     = "Please review and grade my Project -3 Submission"
submit: listing-A1.cpp listing-A2.cpp
        $(tar) $(USER)-proj3.tgz $?
        echo $(msg) | $(mail) -s $(subj) -a $(USER)-proj3.tar.gz $(addr)
```

Make sure you update the dependencies on the `submit:` line to ensure all the required files (source files) are included in the archive that gets created and then attached to your email for submission. You do not need to print out any of your program files - submitting them via email will date and time stamp them and we shall know they come from your account. If you submit multiple versions, we will use the latest version up to when the project is due.

Extra Credit: Create a UML diagram that represents the relationship between classes and interfaces. You should use the UML tools in the `dia` graphing program. Once you generate the files, either (1) rename the existing files and replace them with the output of the `dia2code -t cpp` files, or (2) moify the source files to use the output of the `dia2code -t cpp` command - don't forget to update the `Makefile` as appropriate.

Questions: If you have any questions, please do not hesitate to get in contact with either Phil Curtiss (`pjcurtiss@mtech.edu`) or Ross Moon (`rmoon@mtech.edu`) at your convenicne, or stop by during office hours, and/or avail yourself of the time in the MUS lab when Ross is available.

Project File Manifest:

List Interface

```
/** Interface for the ADT list
 * @file ListInterface.h
 */
#ifndef _LIST_INTERFACE
#define _LIST_INTERFACE

#include "Node.h"

template < class ItemType > class ListInterface
{
public:
/** Sees whether this list is empty.
@return True if the list is empty; otherwise returns false. */
  virtual bool isEmpty() const = 0;

/** Gets the current number of entries in this list.
@return The integer number of entries currently in the list. */
  virtual int getLength() const = 0;

/** Inserts an entry into this list at a given position.
@pre None.
@post If 1 <= position <= getLength() + 1 and the insertion is
successful, newEntry is at the given position in the list,
other entries are renumbered accordingly, and the returned
value is true.
@param newPosition The list position at which to insert newEntry.
@param newEntry The entry to insert into the list.
@return True if insertion is successful, or false if not. */
  virtual bool insert(int newPosition, const ItemType & newEntry) = 0;

/** Removes the entry at a given position from this list.
@pre None.
@post If 1 <= position <= getLength() and the removal is successful,
the entry at the given position in the list is removed, other
items are renumbered accordingly, and the returned value is true.
@param position The list position of the entry to remove.
@return True if removal is successful, or false if not. */
  virtual bool remove(int position) = 0;

/** Removes all entries from this list.
@post List contains no entries and the count of items is 0. */
  virtual void clear() = 0;

/** Gets the entry at the given position in this list.
@pre 1 <= position <= getLength().
@post The desired entry has been returned.
@param position The list position of the desired entry.
@return The entry at the given position. */
  virtual Node<ItemType>* getEntry(int position) const = 0;
```

```
/** Replaces the entry at the given position in this list.
@pre 1 <= position <= getLength().
@post The entry at the given position is newEntry.
@param position The list position of the entry to replace.
@param newEntry The replacement entry. */
  virtual void setEntry(int position, const ItemType & newEntry) = 0;

/** Returns the position within the List of the entry provided or -1 if
    not found in the List
@pre 1 <= position <= getLength()
@post none
@param entry to search for in the List
@return the position within the List if entry is found, or -1 otherwise */
  virtual int getPosition(const ItemType & entry) = 0;

/** Returns a boolean value indicating whether the entry provided is in the List
@pre none
@post none
@param entry to search for in the List
@ return boolean value indicating whether the entry is in the List */
  virtual bool contains(const ItemType & entry) = 0;

/** loads the List with entries from the file referenced by the filename
    provided
@pre none
@post List has entries from the file referenced by the filename
@param string referencing the filename
@return boolean indicating if the operation wass successful */
  virtual bool loadList(const string filename) = 0;

/** displays a tabular representaiton of the List
@pre none
@post none
@param none
@return void */
  virtual void displayList() = 0;
};  // end ListInterface
#endif
```

List Main Driver

```
/** @file ListDriver.cpp
 * List Main Driver - use to generate Santa List
 *
 * @author  Phil Curtiss
 * @version 1.0
 * @date 10/14/15
 ******************************************************/

//------------------------
// Using statements
//------------------------
using namespace std;
```

```
//-------------------------
// C++ includes
//-------------------------
#include <iostream>
#include <fstream>
#include <string>

//-------------------------
// Application includes
//-------------------------
#include "ListInterface.h"
#include "ListADT.h"

/** List Driver - use to help Santa with his Lists
 *******************************************************/
int main(int argc, char *argv[])
{
  // Instantiate a ListADT object
  ListInterface<string> *naughtyList = new ListADT<string>();
  ListInterface<string> *niceList = new ListADT<string>();

  // Check to make sure we are passed one or more filenames to
  // process for Santa - insert into our ListADT() in such a way
  // that the entries are ordered and grouped.
  if (argc < 3){
        cout << "Usage " << "foo" << ": <naughty filename> <nice filename>" << endl;
        return(1);
  }

  // We now have commmand line arguments to process, so let's
  // get to it and populate santa's lists
  if (naughtyList->loadList(argv[1]) == false)
        cout << "Problem reading entries from naughty list: " << argv[1] << endl;

  if (niceList->loadList(argv[2]) == false)
        cout << "Problem reading entries from nice list: " << argv[2] << endl;

  // If the list has some items present, then let's display this
  // list for Santa
  cout << "Your sorted nice list of boys and girls to visit:" << endl;
  if (niceList->isEmpty() == false)
      niceList->displayList();
  else
        cout << "Couldn't find any nice little borys and girls. ;-(" << endl;

  // If the list has some items present, then let's display this
  // list for Santa
  cout << "Your sorted naughty list of boys and girls to visit:" << endl;
  if (naughtyList->isEmpty() == false)
    naughtyList->displayList();
  else
        cout << "Couldn't find any naughty little boys and girls. :-)" << endl;
```

```
  return 0;              //Exit program
}//main()
```

Node Header

```
/** @file Node.h */
#ifndef _NODE
#define _NODE
template < class ItemType>
class Node
{
    private :
            ItemType item; // A data item
            Node<ItemType>* next; // Pointer to next node
    public :
            Node();
            Node( const ItemType& anItem);
            Node( const ItemType& anItem, Node<ItemType>* nextNodePtr);
            void setItem( const ItemType& anItem);
            void setNext(Node<ItemType>* nextNodePtr);
            ItemType getItem() const ;
            Node<ItemType>* getNext() const ;
}; // end Node
#include "Node.cpp"
#endif
```

Node Implementation

```
/** @file Node.cpp */
#ifndef _NODEIMP
#define _NODEIMP

#include "Node.h"
#include <cstddef>

template < class ItemType >
Node < ItemType >::Node ():next (nullptr)
{
}   // end default constructor

template < class ItemType >
Node < ItemType >::Node (const ItemType & anItem): item (anItem), next (nullptr)
{
}   // end constructor

template < class ItemType >
Node < ItemType >::Node (const ItemType & anItem,
                         Node < ItemType > *nextNodePtr): item (anItem),
                                                          next (nextNodePtr)
{
}   // end constructor

template < class ItemType >
```

```
void Node < ItemType >::setItem (const ItemType & anItem)
{
  item = anItem;
}   // end setItem

template < class ItemType >
void Node < ItemType >::setNext (Node < ItemType > *nextNodePtr)
{
  next = nextNodePtr;
}   // end setNext

template < class ItemType >
ItemType Node < ItemType >::getItem () const
{
  return item;
}   // end getItem

template < class ItemType >
Node < ItemType > *Node < ItemType >::getNext () const
{
  return next;
}   // end getNext
#endif
```

ListADT Header

```
/**
 * ListADT: Linked List Implementation
 */
#ifndef _LISTADT
#define _LISTADT

#include "ListInterface.h"
#include "Node.h"

template<class ItemType>
class ListADT: public ListInterface<ItemType>
{
private:
        // Whatever private we need

public:
        // Must have at least these based on Interface
        ListADT();

        ~ ListADT();

        bool isEmpty() const;

        int getLength() const;

        bool insert(int newPosition, const ItemType& newEntry);

        bool remove(int position);
```

```
        void clear();

        Node<ItemType>* getEntry(int position) const;

        void setEntry(int position, const ItemType& newEntry);

        int getPosition(const ItemType& entry);

        bool contains(const ItemType& entry);

        bool loadList(const string filename);

        void displayList();
};

#include "ListADT.cpp"
#endif
```

ListADT Implementation

```
/**
 * Implementation
 */
#ifndef _LISTADTIMP
#define _LISTADTIMP

#include "ListInterface.h"
#include "ListADT.h"
#include "Node.h"

template<class ItemType>
ListADT<ItemType>::ListADT() {}

template<class ItemType>
bool ListADT<ItemType>::isEmpty() const { return true; }

template<class ItemType>
int ListADT<ItemType>::getLength() const { return 1;}

template<class ItemType>
bool ListADT<ItemType>::insert(int newPosition, const ItemType& newEntry) { return 1; }

template<class ItemType>
Node<ItemType>* ListADT<ItemType>::getEntry(int position) const {Node<ItemType>* nPtr = nu

template<class ItemType>
void ListADT<ItemType>::setEntry(int position, const ItemType& newEntry) {}

template<class ItemType>
int ListADT<ItemType>::getPosition(const ItemType& entry) { return 1; }

template<class ItemType>
bool ListADT<ItemType>::contains(const ItemType& entry) { return true; }
```

```
template<class ItemType>
bool ListADT<ItemType>::loadList(const string filename) { return true; }

#endif
```

Makefile

```
#
# Makefile for Generating C++ executables
#
# F15 CSCI 232 - Data Structures and Algorithms
# Phillip J. Curtiss, Associate Professor
# Computer Science Department, Montana Tech
# Museum Buildings, Room 105
#
# Project-3: Santa's Lists
# Date Assigned: 2015-10-14
# Date Due: 2015-10-26 by Midnight

# Define Macros related to printing and submitting programs
a2ps    = a2ps -T 2
mail    = mail
addr    = pcurtiss@mtech.edu rmoon@mtech.edu
tar     = tar -cvzf

# Define Macros to help generate the program file required
DIA     = dia2code
C++     = g++ -std=c++11
CFLAGS  = -Wall -Werror
LD      = g++
LDFLAGS =
LIBS    =
OBJS    = ListDriver.o ListADT.o Node.o
EXEC    = proj3

# Provide dependency lists here, one on each line - don't forget to make sure
# if you have source files depending (or generated by) UML diagrams to include them as wel
.SUFFIXES:              .dia
all:                    $(EXEC)
${EXEC}:                ListDriver.o
ListDriver.o:   ListInterface.h ListDriver.cpp
ListADT.o:              ListADT.cpp ListInterface.h Node.h Node.cpp
Node.o:                 Node.h Node.cpp

################################################################################
# Rules Used to Generate executable from object - DO NOT EDIT
$(EXEC):        $(OBJS)
        $(LD) $(LDFLAGS) -o $@ $? $(LIBS)

# Rule to generate object code from cpp source files - DO NOT EDIT
.cpp.o:
        $(C++) $(CFLAGS) -c $<
```

```
# Rule to generate header and source files from Dia UML Diagram - DO NOT EDIT
.dia.h:
	$(DIA) -t cpp $<

.dia.cpp:
	$(DIA) -t cpp $<
##############################################################################

# Rule to Clean up (i.e. delete) all of the object and execuable
# code files to force make to rebuild a clearn executable only
# from the source files
clean:
	rm -f $(OBJS) $(EXEC)

# Rule to submit programming assignments to graders
# Make sure you modify the $(subj) $(msg) above and the list of attachment
# files in the following rule - each file needs to be preceeded with an
# -a flag as shown
subj    = "CSCI232 DSA - Proj3"
msg     = "Please review and grade my Project-3 Submission"
submit: _PLACE_YOUR_SOURCE_CODE_FILES_HERE_
	$(tar) $(USER)-proj3.tgz $?
	echo $(msg) | $(mail) -s $(subj) -a $(USER)-proj3.tgz $(addr)

print:  _PLACE_YOUR_SOURCE_CODE_FILES_HERE_
	$(a2ps) $?
```