CSCE 221 Cover Page
Program Assignment # 2

First Name: Rong Last Name: Xu UIN: 928009312

User Name: Abby-xu E-mail address: rongx0915@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more in the Aggie Honor System Office http://aggiehonor.tamu.edu/

Type of sources					
People					
Web pages (provide URL)					
Printed material					
Other Sources					

I certify that I have listed all the sources that I used to develop the solutions/code to the submitted work.

"*On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.*"

Your Name (signature) Rong Xu Date 2020/10/06 The Pro-

gramming Assignment Report Instructions
CSCE 221

1. The description of an assignment problem.
   (a) This lab including two parts:
      i. Part one involves implementing a simple Double Linked List and a Double Linked List template ADT.
      ii. Part two involves writing a simple library manage system of the Double Linked list template and write a report.

2. The description of data structures and algorithms used to solve the problem.
   (a) Provide definitions of data structures by using Abstract Data Types (ADTs)
      i. We used double Linked List data structure in this assignment. For each node is the list, we use a struct for defining it.
      ii. For the library manage system, we use a class called record to put the basic infomation of each book including title, author, ISBN, year and edtion. And we use the Double Linked List template for storing the records, and a vector for storing the linked list. In the vector, we have 26 elements(the double linked list) which are in the Alphabet order.
   (b) Write about the ADTs implementation in C++.
      i. We used double Linked List data structure in this assignment. During the implementation of the double linked list, for each list node, it is connected to a front node and a back node by using two pointers previous and next. A friend class called DDListNode is also implemented, which has three private members, hence T obj, prev pointer and next pointer. Thwe double linked list class is a list collecting the all DLListNodes.
      ii. For the Record class, we have five string private members called title, author, ISBN, year and edtion. and the get/set functions for these five memebers. We also have to overload the operators “>>” “<<” and “==”.
      iii. For the Library class, we have store the infomation of each book, then each records will be as each node for a double linked list, then we have a vector for storing the linked lists.
   (c) Describe algorithms used to solve the problem.
      i. Double Linked List:
         A. insert_before/after
         B. remove_before/after
         C. insert_last/first
         D. remove_last/first
         E. copy/move constructor
         F. copy/move operator
         G. output operator
      ii. Library:
         A. import_database
         B. export_database
         C. print_database
         D. add/remove_record
   (d) Analyze the algorithms according to assignment requirements.
      i. Double Linked List:
         A. insert_before/after O(1): insert a node before/after a node of the double linked list. Create a new node; point teh next to where the header pointing to and then let teh header pointer pointing to it.
         B. remove_before/after O(1): delete a node before/after a node of the double linked list.

C. insert_last/first O(1): insert a node at the beginning/ending of the double linked list. Create a new node; point teh next to where the header pointing to and then let teh header pointer pointing to it.

D. remove_last/first O(1): delete a node at the beginning/ ending of the double linked list.

E. copy/move constructor O(n): this function will copy every element in another list

F. copy/move assignment O(n): the function will clear the linked lisit first(O(n)). Then copy the whole list from the other list (O(n)). O(n) + O(n) = O(n)

G. output operator O(n): this function will print out every nodes content in the linked list.

ii. Library:

A. import_database O(n): this function will read the records from a txt file then store them into the database. the number returned is the number of records installed, which is as same as n.

B. export/print_database O(n): this function will export/print the database out. so basically, it will go over the whole vector (26 elements) and the 26 linked list. and export/print each record out.

C. add_record: this function is going to add the record into the database. //best case: when the title of the record is start "A" and in the data base the linked list "A", there is no record in the list. ==> O(1).//worst case: when the title of the record is start "Z" and in the database the linked list "Z", there have the n records. ==> 26n for search and n for insert_last function ==> 26n + n = O(n) //average case: O(n)

D. remove record: this function is going to remove one of the record in teh database. //best case: O(1) //worst case / average case: O(n)

3. A C++ organization and implementation of the problem solution

   (a) Provide a list and description of classes or interfaces used by a program such as classes used to implement the data structures or exceptions.

```
<typename T>
struct DLListNode {
T obj;
DLListNode<T> *prev, *next;
DLListNode(T e = T(), DLListNode *p = nullptr, DLListNode *n = nullptr);
};

// doubly linked list class
template <typename T>
class DLList {
private:
DLListNode<T> header, trailer;
public:
DLList();// default constructor
DLList(const DLList<T>& dll); // copy constructor
DLList(DLList<T>&& dll); // move constructor
~DLList(); // destructor
void clear();
DLList<T>& operator=(const DLList<T>& dll); // copy assignment operator
DLList<T>& operator=(DLList<T>&& dll); // move assignment operator
// return the pointer to the first node
DLListNode<T> *first_node() const { return header.next; }
// return the pointer to the trailer
const DLListNode<T> *after_last_node() const { return &trailer; }
// return if the list is empty
bool is_empty() const { return header.next == &trailer; }
T first() const; // return the first object
T last() const; // return the last object
void insert_first(T obj); // insert to the first node
T remove_first(); // remove the first node
void insert_last(T obj); // insert to the last node
T remove_last(); // remove the last node
void insert_after(DLListNode<T> &p, T obj);
void insert_before(DLListNode<T> &p, T obj);
T remove_after(DLListNode<T> &p);
T remove_before(DLListNode<T> &p);
};

class Record {
private:
//member variables
std::string title, author, ISBN, year, edition;
public:
    void set_title(std::string a);
std::string get_title()const;
        void set_author(std::string a);
std::string get_author()const;
        void set_ISBN(std::string a);
std::string get_ISBN()const;
        void set_year(std::string a);
std::string get_year()const;
        void set_edition(std::string a);
std::string get_edition()const;
};

class Library {
public:
//Searches for a title in the database and returns vector of matching records
std::vector<Record> search(std::string title);

//Imports records from a file.  Does not import duplicates.
// Return the number of records added to the database
int import_database(std::string filename);

//Exports the current database to a file
//Return the number of records exported
int export_database(std::string filename);

void print_database();

//add record to database, avoid complete duplicates
bool add_record(Record book);

//Deletes a record from the database
void remove_record(Record book);
```

```
//Prompts user for yes or no and returns choice Y or N
char prompt_yes_no();

//Given a vector of menu options returns index of choice
int prompt_menu(std::vector<std::string>);

//Prompts user for a new record
Record prompt_record();

//Prompt for a valid title
std::string prompt_title();

// Prompt for a valid string
std::string prompt_string(std::string prompt);
private:
std::vector<DLList<Record>> book_db = vector<DLList<Record>>(26); };
```

(b) Include in the report the class declarations from a header file (.h) and their implementation from a source file (.cpp).

i. Templated Double Linked List

```
#include "TemplatedDLList.h"
template <typename T>
DLList::DLList() : header(T()), trailer(T()) {
header.next = &trailer;
trailer.prev = &header;
}
DLList::DLList(const DLList<T>& dll) {
header.next = &trailer;
trailer.prev = &header;
for (DLListNode* n = dll.first_node(); n != dll.after_last_node(); n = n -> next)
this -> insert_last(n -> obj);
}
DLList::DLList(DLList<T>&& dll) {
if (dll.is_empty()) {
header.next = &trailer;
trailer.prev = &header;
} else {
header.next = dll.first_node();
trailer.prev = dll.trailer.prev;
dll.header.next = &dll.trailer;
dll.trailer.prev = &dll.header;
header.next -> prev = &header;
trailer.prev -> next = &trailer;
} }
DLList::~DLList() { clear(); }
DLList<T>& DLList::operator=(const DLList<T>& dll) {
if(this == &dll)
return *this;
header.next = &trailer;
trailer.prev = &header;
DLListNode<T> *temp_node = dll.header.next;
while(temp_node != dll.trailer.prev){
this -> insert_last(temp_node->obj);
temp_node = temp_node ->next;
}
this -> insert_last(temp_node->obj);
return *this; }
DLList<T>& DLList::operator=(DLList<T>&& dll) {
if (this != &dll) {
if (dll.is_empty()) {
header.next = &trailer;
trailer.prev = &header;
} else {
header.next = dll.first_node();
trailer.prev = dll.trailer.prev;
dll.header.next = &dll.trailer;
dll.trailer.prev = &dll.header;
header.next -> prev = &header;
trailer.prev -> next = &trailer;
}}
void DLList::clear() {
DLListNode<T> *nextNode = header.next;
while (nextNode != &trailer) {
nextNode = nextNode->next;
delete header.next;
header.next = nextNode;
}
```

```
header.next = &trailer;
trailer.prev = &header; }
T DLList::first() const { return header.next -> obj; }
T DLList::last() const { return trailer.prev -> obj; }
void DLList::insert_first(T obj) {
DLListNode* n = new DLListNode(obj);
DLListNode* temp = header.next;
header.next = n;
n -> prev = &header;
n -> next = temp;
}
T DLList::remove_first() {
if (is_empty())
throw("The␣LinkedList␣is␣empty...");
DLListNode* temp = header.next;
T data = temp -> obj;
header.next = temp -> next;
temp -> next -> prev = &header;
delete temp;
return data;
}
void DLList::insert_last(T obj){
DLListNode* n = new DLListNode(obj, trailer.prev, &trailer);
trailer.prev -> next = n;
trailer.prev = n; }
T DLList::remove_last() {
if (is_empty())
throw("The␣LinkedList␣is␣empty...");
DLListNode* temp = trailer.prev;
T data = temp -> obj;
trailer.prev = temp -> prev;
temp -> prev -> next = &trailer;
delete temp;
return data; }
void DLList::insert_after(DLListNode<T> &p, T obj) {
DLListNode<T>* n = new DLListNode<T>(obj, &p, p.next);
p.next -> prev = n;
p.next = n;
}
void DLList::insert_before(DLListNode<T> &p, T obj) {
DLListNode<T>* n = new DLListNode<T>(obj, p.prev, &p);
p.prev -> next = n;
p.prev = n;
}
T DLList::remove_after(DLListNode<T> &p) {
if (p.next == &trailer || is_empty())
throw("The␣LinkedList␣is␣empty...");
DLListNode<T>* temp = p.next;
T data = temp -> obj;
temp -> prev -> next = temp -> next;
temp -> next -> prev = temp -> prev;
delete temp;
return data;
}
T DLList::remove_before(DLListNode<T> &p) {
if (is_empty())
throw("The␣LinkedList␣is␣empty...");
DLListNode<T>* temp = p.prev;
T data = temp -> obj;
p.prev = temp -> prev;
temp -> prev -> next = &p;
delete temp;
return data;
}
ostream& operator<<(ostream& out, const DLList& dll){
DLListNode<T> *temp = dll.first_node();
while(temp != dll.after_last_node()){
out << temp -> obj << ",␣";
temp = temp -> next;    }
return out;
}
```

ii. Record

```
void Record::set_title(std::string a) {this -> title = a;}
std::string Record::get_title()const {return title;}

void Record::set_author(std::string a) {this -> author = a;}
std::string Record::get_author()const {return author;}

void Record::set_ISBN(std::string a) {this -> ISBN = a;}
```

```
std::string Record::get_ISBN()const {return ISBN;}

void Record::set_year(std::string a) {this -> year = a;}
std::string Record::get_year()const {return year;}

void Record::set_edition(std::string a) {this -> edition = a;}
std::string Record::get_edition()const {return edition;}

// Stream operators std::istream& operator>>(std::istream& is, Record& rec) {
std::string Blank, Title, Author, ISBN, Year, Edition;
getline(is, Blank, '\n');
getline(is,Title, '\n'); rec.set_title(Title);
getline(is,Author, '\n'); rec.set_author(Author);
getline(is,ISBN, '\n'); rec.set_ISBN(ISBN);
getline(is,Year, '\n'); rec.set_year(Year);
getline(is,Edition, '\n'); rec.set_edition(Edition);
return is;
}
std::ostream& operator<<(std::ostream& os, Record& rec) {
os << rec.get_title() << std::endl;
os << rec.get_author() << std::endl;
os << rec.get_ISBN() << std::endl;
os << rec.get_year() << std::endl;
os << rec.get_edition() << std::endl;
return os;
}
// Comparison operators
bool operator==(const Record& r1, const Record& r2) {
return (r1.get_title() == r2.get_title() &&
r1.get_author() == r2.get_author() &&
r1.get_ISBN() == r2.get_ISBN() &&
r1.get_year() == r2.get_year() &&
r1.get_edition() == r2.get_edition());
}
```

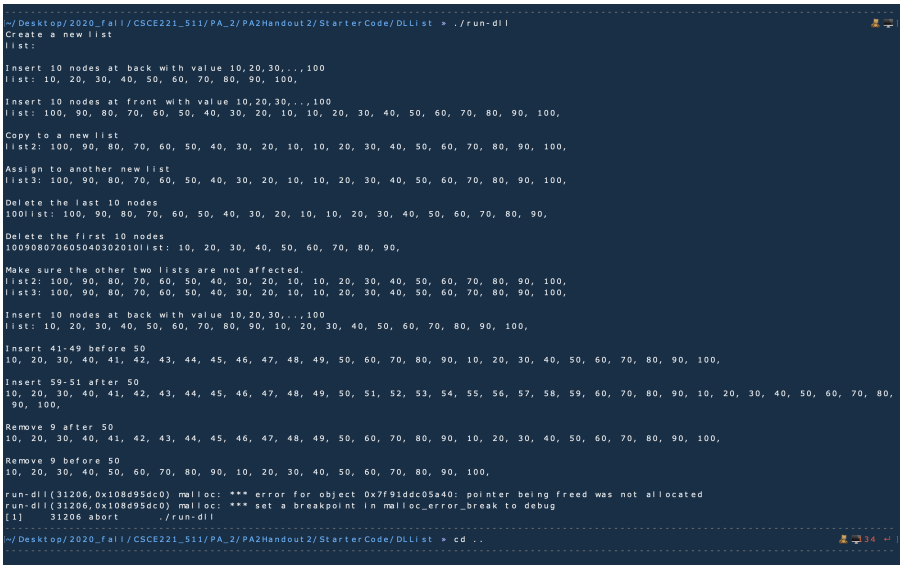
iii. Library

```
#include "Library.h"
#include "TemplatedDLList.h"
#include <fstream>
#include <string>
using namespace std;
//Searches for a title in the database and returns vector of matching records std::vector<Record>
Library::search(std::string title) {    vector<Record> temp;
int index = title[0] - 'A';
if(index < 0 || index > 25)
return temp;
else if(book_db.at(index).is_empty())
return temp;
else {
DLListNode<Record>* n = book_db.at(index).first_node();
while (n != book_db.at(index).after_last_node()) {
if(title == n -> obj.get_title())
temp.push_back(n -> obj);
n = n -> next;
}       }
return temp;
}
//Imports records from a file.  Does not import duplicates.
// Return the number of records added to the database int Library::import_database(std::string filename) {
string title, author, ISBN, year, edition;
int num = 0;
std::ifstream inFS(filename);
Record temp;
while(inFS >> temp) {
if (add_record(temp)) {
int index = temp.get_title()[0] - 'A';
book_db.at(index).insert_last(temp);
num += 1;
} }
inFS.close();
return num;
}
//Exports the current database to a file
//Return the number of records exported int Library::export_database(std::string filename) {
int num = 0;
std::ofstream outFS (filename);
for (int i = 0; i < 26; i++) {
DLListNode<Record>* n = book_db.at(i).first_node();
while (n != book_db.at(i).after_last_node()) {
outFS << n -> obj;
```

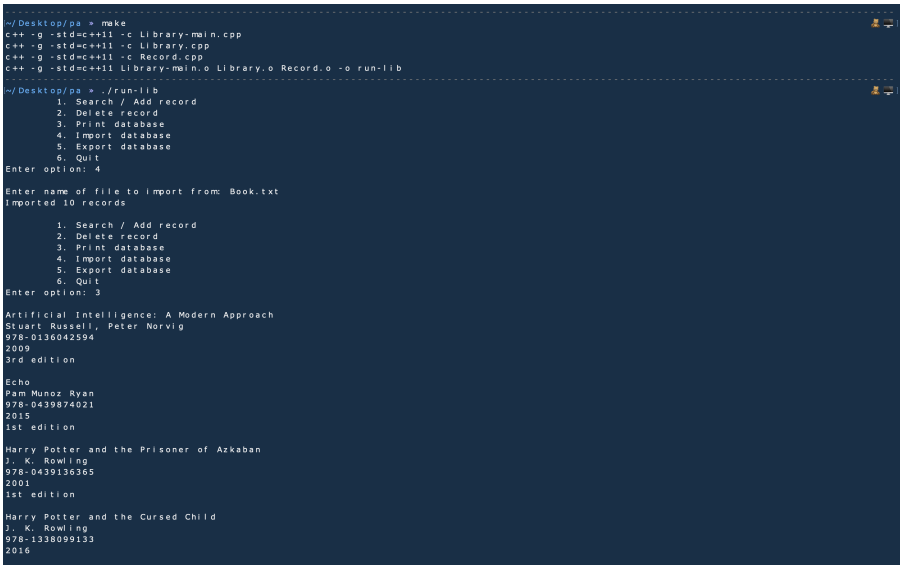
```
num += 1;
n = n -> next;
}}
outFS.close();
return num;
}
void Library::print_database() {
for (int i = 0; i < 26; i++) {
DLListNode<Record>* header_node = book_db.at(i).first_node();
while(header_node != book_db.at(i).after_last_node()){
cout << header_node -> obj << endl;
header_node = header_node -> next;
} } }
//add record to database, avoid complete duplicates bool Library::add_record(Record book) {
vector<Record> search_result;
search_result = search(book.get_title());
int index = book.get_title()[0] - 'A';
if(search_result.size() == 0 && index < 25 && index > 0) {
book_db.at(index).insert_last(book);
return true;
}
return false;
}
//Deletes a record from the database
void Library::remove_record(Record book) {
int index = book.get_title()[0] - 'A';
DLListNode<Record>* n = book_db.at(index).first_node();
while(n != book_db.at(index).after_last_node() -> prev) {
if (book.get_title() == n -> obj.get_title()) {
n = n -> next;
book_db[index].remove_before(*n);
break;
}
n = n -> next;
} }
//Prompts user for yes or no and returns choice Y or N char Library::prompt_yes_no() {
cout << "Please␣Enter␣Y/N" << endl;
char if_prompt;
cin >> if_prompt;
return if_prompt; }

//Given a vector of menu options returns index of choice int Library::prompt_menu(std::vector<std::string> a) {
cout << "Please␣select␣an␣option" << endl;
for (int i = 0; i < a.size(); i++)
cout << i + 1 << ".␣" << a.at(i) << endl;
int b;
cin >> b;
return b - 1;
}
//Prompts user for a new record Record Library::prompt_record() {       Record new_record;
cin.ignore();
cout << "\n␣Please␣enter␣the␣author:" << endl;
std::string author; getline(cin,author);
new_record.set_author(author);
cout << "\n␣Please␣enter␣the␣ISBN:" << endl;
std::string ISBN; getline(cin,ISBN); new_record.set_ISBN(ISBN);
cout << "\n␣Please␣enter␣the␣year:" << endl;
std::string year; getline(cin,year); new_record.set_year(year);
cout << "\n␣Please␣enter␣the␣edition:" << endl;
std::string edition; getline(cin,edition);
new_record.set_edition(edition);
return new_record;  }
//Prompt for a valid title std::string Library::prompt_title() {
std::string prompt_title;
getline(cin, prompt_title);
return prompt_title;
}
// Prompt for a valid string std::string Library::prompt_string(std::string prompt) {
cout << prompt << endl;
std::string usr_input;
getline(cin, usr_input);
return usr_input; }
```

(c) Provide features of the C++ programming paradigms like Inheritance or Polymorphism in case of object oriented programming, or Templates in the case of generic programming used in your implementati

i.



ii.



4. A user guide description how to navigate your program with the instructions how to:

   (a) compile the program: specify the directory and file names, etc.

```
run-lib: Library-main.o Library.o Record.o
c++ -g -std=c++11 Library-main.o Library.o Record.o -o run-lib

Library.o: Library.cpp Library.h Record.h TemplatedDLList.h
c++ -g -std=c++11 -c Library.cpp

Library-main.o: Library-main.cpp Library.h Record.h
c++ -g -std=c++11 -c Library-main.cpp

Record.o: Record.cpp Record.h
c++ -g -std=c++11 -c Record.cpp

DLList.o: DLList.cpp DLList.h
c++ -g -std=c++11 -c DLList.cpp

DLList-main.o: DLList-main.cpp DLList.h
c++ -g -std=c++11 -c DLList-main.cpp
```

(b) run the program: specify the name of an executable file.

```
» cd ./PA_2/PA2Handout2/StarterCode/DLList
» make clean
» make
» ./run-dll

» cd ./PA_2/PA2Handout2/StarterCode/Record
» make clean
» make
» ./record

» cd ./PA_2/PA2Handout2/StarterCode/Library
» make clean
» make
» ./
```

5. Specifications and description of input and output formats and files

   (a) The type of files: keyboard, text files, etc (if applicable).

      i. Input file: .txt file

   (b) A file input format: when a program requires a sequence of input items, specify the number of items per line or a line termination. Provide a sample of a required input format.

      i. Input file: input file should be an .txt file which have the format like this:

```
Harry Potter And The Chamber Of Secrets
J. K. Rowling
978-0439064873
2000
1st edition

H is for Hawk
Helen Macdonald
978-0802123411
2015
1st edition

Harry Potter and the Cursed Child
J. K. Rowling
978-1338099133
2016
1st edition
```

      The format is required, the first line is the title of the book; second line is the author of the book; the third line is the ISBN; the forth line is the publish year; and the last line is the number of edition.

      ii. The first letter of title should use upper case.

      iii. The search keywork is case sensitive.

   (c) Discuss possible cases when your program could crash because of incorrect input (a wrong file name, strings instead of a number, or such cases when the program expects 10 items to read and it finds only 9.)

      i. none, if the program cannot open the input file, it will throw an exception and exit.

      ii. no special case.

      iii. The first elementt of the title of a book cannot be number.

6. Provide types of exceptions and their purpose in your program.

   (a) logical exceptions (such as deletion of an item from an empty container, etc.).

      i. No logical error has been found in testing

   (b) runtime exception (such as division by 0, etc.)

      i. cannot open the .txt file that need to be imported.

7. Test your program for correctness using valid, invalid, and random inputs (e.g., insertion of an item at the beginning, at the end, or at a random place into a sorted vector). Include evidence of your testing, such as an output file or screen shots with an input and the corresponding outpu

   (a) Invalid inputs were tested. The title with same string was tested.

   (b) Wrong input file name teasted.