CSCI 2720: Data Structures

Spring 2021

Project 1
UnsortedList ADT

Due: Friday, Feb 5, 2021 @11:59:59 PM

In this project you will implement the array representation of the Unsorted List as a Generic ADT.

Objectives:

- Practice the array-based implementation of the List ADT.
- Practice implementing generic data types as template.
- Practice implementing and using dynamically allocated arrays.
- Practice using generic data types at the application level, with both primitive data types and user defined data types.

Project Requirements:

Use C++ template mechanism and define a generic **UnsortedList** ADT. Implement the **UnsortedList** ADT using a dynamically allocated array.

I am attaching the UnsortedList.h file, the driver program, and the test files to this assignment. You need to implement the Unsorted list functions (UnsortedList.cpp) and complete the code of the driver program.

ADT Unsorted List Specification:

Private data members:

1. **info:** A dynamically allocated array. Array elements are of type ItemType (The generic data type).
2. **length:** integer; stores the actual number of items in the list. Last occupied array index is length-1;
3. **Max_Items:** integer. The size of the list. Default value = 50

The UnsortedList ADT should implement the following public functions (defined in the attached file UnsortedList.h). Create and save the implementation in a file called UnsortedList.cpp.

1. **Constructors**
   Function: Initializes the list.
   Precondition: None.
   Postcondition: List is initialized.

Implement the following constructors:

1.1 A no argument constructor to initialize **Max_Items** to 50.
1.2 A one argument constructor with a formal parameter **size**, that initializes **Max_Items** to **size**.
Both constructors dynamically allocate the array **info** of size **Max_Items.**

2. **bool isEmpty() const;**
   Function: Determines if the list is empty.
   Precondition: List is initialized.
   Postcondition: The function returns true if the list is empty and false otherwise.

3. **bool isFull() const;**
   Function: Determines if the list is full.
   Precondition: List is initialized.
   Postcondition: The function returns true if the list is full and false otherwise.

4. **int getLength() const;**
   Function: Returns number of elements in the list.
   Precondition: List is initialized.
   Postcondition: Function value equals number of list elements. **This function should run in constant time: O(1)**

5. **void putItem (ItemType newItem);**
   Function: Adds a new element to list. This function should not allow duplicate keys, and must check that there is space for the new item.
   Precondition: List is initialized.
   Postconditions: 1) newItem is in the list, if list is not full and newItem's key was not in the list.
   2) The FullList exception is thrown if the list is full.
   3) The DuplicateItem exception is thrown if newItem's keys was found in the list.
   This function should call the function findIndex (see function number 10 below)

6. **void deleteItem (ItemType item);**
   Function: Removes an element from the list.
   Precondition: List is initialized.
   Postconditions: item is not in the list. The ItemNotFound exception is thrown if item was not found in the list.
   This function should call the function findIndex (see function number 10 below)
   This function should have a linear time T(n)= N+ 1 = O(N)
   i.e. O(N) to search + O(1) to delete the item logically.

7. **ItemType getAt(int index) const;**
   Function: Returns the element at the specified position (index) in this list.
   Precondition: List is initialized.
   Postcondition: The function returns the element at the specified position in this list,
   or throws the OutOfBound exception if the index not valid.
   (index < 0 || index >= length of the list).

8. **void makeEmpty();**
   Function: Reinitializes the list to empty state.
   Precondition: None.
   Postcondition: List is empty.

9. **~UnsortedList();** // class destructor
   Function: Reinitializes the list to empty state. Deallocates all dynamically allocated data members.
   Precondition: None.
   Postcondition: List is empty.

10. int findIndex (ItemType item); // helper function (private function)

Function: Searches the list for **item** and returns **item's** index if the item was found. The function should return -1 if **item** was not in the list.

This function should be used by your putItem and deleteItem functions.

1) Implement a simple Student class

Implement a simple Student class with two data members: studentID (int) and studentName (String). Implement a constructor, setter, and getter functions for class Student.

You may need to overload **operators**. For example, you *may* need to overload the operator == if your findIndex function compares two Student objects in the form:
if (item==info(i)), where info is an array of student objects. The operator == should return true if the compared student objects have the same studentID and false otherwise.

2) The UnsortedList driver program:

I am attaching the file UnsortedListDriver.cpp. This program will test your generic UnsortedList ADT. The attached file is ready to test the a list of integers, you need to complete the other test function, `testSudentList()`.

To test the generic data type, we will define two lists of different datatypes; each is defined in the corresponding test function.

UnsortedList<int> integerList(5); // list of integers of size 5 defined in `testIntegerList()`
UnsortedList<Student> studentList(10); // Students list of size 10. Defined in `testSudentList()`.

The attached driver UnsortedListDriver.cpp will start by reading an integer that represents the user choice of the type of list elements from the user. **This input is from standard input (cin>>), not from the input file**.

Enter Elements Type
1 for integer
2 for student

The test driver calls one of the test functions based on the elements type. The program will continue reading commands and data from the input file **inFile** until the command "Quit" is read.

Commands are: (IsEmpty, IsFull, MakeEmpty, GetLength, GetAt, PutItem, DeleteItem, PrintList and Quit). Based on the command, the program will call the corresponding UnsortedList function and writes output to an output file outFile.txt.

We will test your program with our command files (text files, see next section). So, make sure you are using the same commands and be aware of case sensitivity of the commands. (e.g. IsEmpty not isEmpty).

3) Text files:

We will test your program with 2 text files, each contains testing commands for a specific data type. Your test driver should carefully associate the inFile with the corresponding commands file. Our command file for integer lists is called **intcommands.txt**. Our commands

file to test a list of student objects is called **studcommands.txt**. I am attaching two sample test files to this project.

A sample execution is at the end of this project description, see **Sample test execution (1) and (2).**

Make sure that the program will give the correct results and will not stop due to any run-time error.

I would recommend that you test your program gradually and extensively. Test one function then integrate another function, in other words, test with a file that contains one command then add another command, and so on… then test using the attached test file. To avoid an infinite loop and thus exceeding your server quota, make sure to include the Quit command at the last line of our input fie.

Use the attached UnsortedList.h file. The driver works based on the exception classes defined at the top of the UnsortedList.h file.

What to Submit?

Name your project directory **project1**. Submit your project1 directory to **csci-2720c** on *odin* (use the following command: *submit project1 csci-2720c*).

If you are off campus, you must use UGA VPN (remote.uga.edu) before you can ssh into *odin.cs.uga.edu* using your MyID and password.

Your project1 directory should contain the following files:

1. UnsortedList.h
2. UnsortedList.cpp
3. UnsortedListDriver.cpp
4. Student.h and Student.cpp
5. README file to tell us how to compile and execute your program.
6. The makefile (read the syllabus for details)

Basic grading criteria:

1. If the project passed all test cases defined in our commands files 100%
2. If the project did not compile on *odin* 0%
3. If the project compiled but did not run 0% - 30%
   30% is given if all required files were submitted and program code completely satisfies all functional requirements.
4. If the project runs with wrong output for some test cases, test case rubrics will apply

See course syllabus for late submission evaluation

Sample test execution (1)

The left column is the command read from the command file, the right column is the expected output of the command written to outFile.txt; a blank line indicates that nothing will be written on the output file (so your output file should not contain blank lines and would look more compact than the example shown below).

Command file(intcommands.txt)	**Expected output: outFile.txt**
IsEmpty	list is empty
IsFull	list is not full
PutItem 10	
PrintList	10
PutItem 2	
PrintList	10 2
DeleteItem 7	item is not in the list
DeleteItem 10	
PrintList	2
DeleteItem 2	
PrintList	
IsEmpty	list is empty
PutItem 7	
PutItem 90	
PutItem 5	
PutItem 100	
PutItem 3	
PrintList	7 90 5 100 3
DeleteItem 90	
PrintList	7 3 5 100
GetAt 0	Item at index 0 is: 7
GetAt 3	Item at index 3 is: 100
GetAt 5	index is out of range
PutItem 6	
PutItem 20	List is full, insertion failed
IsFull	List is full
MakeEmpty	
PrintList	
Quit	

Screen

```
Enter Elements Type
1 for int
2 for Student
1
```

Sample test execution (2)

Command file(studcommands.txt)	**Expected output: outFile.txt**
IsEmpty	list is empty
IsFull	list is not full
PutItem 81012 Eman	
PrintList	81012 Eman
PutItem 81001 John	
IsEmpty	list is not empty
PrintList	81012 Eman 81001 John
DeleteItem 810111	item is not in the list
IsFull	List is not full
GetAt 1	Item at index 1 is: 81001 John
GetLength	2
MakeEmpty	
IsEmpty	List is empty
GetLength	0
Exit	Unrecognized command
Quit	

Screen

```
Enter Elements Type
1 for int
2 for Student
2
```