**The ADT Sorted List**

1. **Design Document**

**Introduction**

ADT sorted list is an ordered ascending or descending catalog of a mathematical model for a certain classes of data structures that have similar behavior, or for certain data types with similar semantics. This program takes names from the user and orders them from least to greatest string; In addition to that, the user can modify and remove the inserted data’s.

**Data structure**

The program uses a template class that can adapt to the users need. The array, ‘items’, holds each string read from the input command prompt. The class, sorted, sets the maximum allowed value of the array length, and the array items is set to accept 10 data’s from the user set by a static constant integer variable ‘DEFAULT\_BAG\_SIZE’. The integer variables, ‘itemCount’ and ‘maxItem’, keeps track of the array size and lets the user know if the array is full or have a memory space.

**Functions**

The Program has one class and an interface class. In ‘sorted’ class, I have used 9 functions including the constructor. To figure out if the array is empty, I used a Boolean type function called ‘sortedIsEmpty()’. This function checks and returns true if there is something saved in the array. For checking the maximum length of the array size and current length of the array, I have used ‘sortedMaxLength()’ and ‘sortedGetLength()’ respectively. I have also used ‘listSortedArray()’ for listing all the data’s stored in the array ‘items’. ‘sortedInsert()’ is a Boolean function that accepts the user value and inserts the value in the array by comparing the values with already saved data’s inside arrays and shifting them to the right if they are higher than the new one. ‘sortedRemove()’ is a Boolean function that returns true if there is a value proclaimed by the user and successfully removed it. It compares the entered value with the stored value starting from the highest value and shifting it to the left until the index value of the removed value. ‘sortedRetrive()’ function accepts 2 parameters; one is for the index value and second one is for the data Item to be saved at. This Boolean functions uses ‘sortedGetLength()’ function to check whether the index value is within the boundaries of limits. The final function I used is an integer function that returns the index of the required or prompted value by the user. It would return -1 if it hasn’t found a value, and other values for found values.

The program also uses toupper() from the standard libraries for making sure that all the user inputs would be converted to capital letters for ease of comparison.

**The main program**

The main program starts with introducing what the program is intended to be used. Latter, it creates an object using the class ‘sorted’ with the name ‘sortedData.’ The program later introduces the main menu where the user can select choices for use. These menus includes:

1. Inserting strings: this can be achieved with entering I or i. Later, I used ‘switch-case’ statement to call ‘sortedinsert ().’
2. Removing strings: this can be achieved by selecting R or r. Later, I used ‘switch-case’ statement to call ‘sortedRemove ().’
3. Finding strings: this can be achieved by entering L or l. Later, I used ‘switch-case’ statement to call ‘sortedLengthLength ().’
4. Retrieving Data: this can be achieved by entering G or g. Later, I used ‘switch-case’ statement to call ‘sortedRetrive().’
5. Getting index value: this can be achieved by entering P or p. Later, I used ‘switch-case’ statement to call ‘locatePosition().’
6. Show all contents: this can be achieved by entering S or s. Later, I used ‘switch-case’ statement to call ‘listSortedArray().’
7. Quit the program: this can be achieved by entering Q or q. Later, I used ‘switch-case’ statement to display ‘Good bye’ message and used while loop to stop repeating the loop.
8. **Code list**

**Assign2\_Sorted.cpp: main source**

#include <iostream>

#include <string>

#include "sortedList.h"

#include "sorted.cpp"

using namespace std;

int main()

{

cout << "\tThis program implements ADT sorted list\n";

sorted<string> sortedData;

string tempData;

char select = ' ';

cout << "Menu:\n"

<< "\tEnter I or i for inserting a name\n"

<< "\tEnter R or r for removing a name\n"

<< "\tEnter L or l to find the number of saved arrays\n"

<< "\tEnter G or g to retrieve a data\n"

<< "\tEnter P or p to get the position of a value\n"

<< "\tEnter S or s to show all the contents\n"

<< "\tEnter Q or q to quit from the program\n\n";

do{

cout << "Enter your choice: ";

cin >> select;

switch (toupper(select))

{

case 'I':

{

cout << "Enter a name: ";

cin >> tempData;

if (sortedData.sortedInsert(tempData))

cout << "Your value has been successfully added\n";

else

cout << "Memeory is full, Try deleting a value\n";

}

break;

case 'R':

{

cout << "Enter the name you want to delete: ";

cin >> tempData;

if (sortedData.sortedRemove(tempData))

cout << tempData << " has been successfully deleted\n";

else

cout << "Couldnt find " << tempData << ". Try again\n";

}

break;

case 'L':

{

cout << "You have used " << sortedData.sortedGetLength() << " memory spaces of the total " << sortedData.sortedMaxLength() << " memory spaces" << endl;

}

break;

case 'G':

{

int dataV;

cout << "Which index value would you like to retrive: ";

cin >> dataV;

sortedData.sortedRetrive(dataV, tempData);

cout << "The vaue you are looking for at "<<dataV <<" is " << tempData << endl;

}

break;

case 'P':

{

cout << "Which value do you want to get the value of: ";

cin >> tempData;

if (sortedData.locatePosition(tempData)>-1)

cout << "\n The value you were looking for is at " << sortedData.locatePosition(tempData) << " position" << endl;

else

cout << "Couldnt find " << tempData << ".Try again\n";

}

break;

case 'S':

cout << "These are the values you entered in order from least to greatest\n";

sortedData.listSortedArray();

break;

case 'Q':

cout << "\nGood bye";

break;

default:

cout << "Invalid input,Try again\n";

break;

}

cin.get();

} while (toupper(select) != 'Q');

cin.get();

return 0;

}

**SortedListInterface.h: header file**

#ifndef SORTEDINTERFACE

#define SORTEDINTERFACE

using namespace std;

template<class ListItemType>

class sortedInterface

{

public:

/\*\* Sees whether this array is empty.

@return True if the array is empty, or false if not. \*/

virtual bool sortedIsEmpty() const = 0;

/\*\* gets the max lenfth of the array

@return the max length of the sorted array. \*/

virtual int sortedMaxLength() const = 0;

/\*\* Gets the current number of entries in this array.

@return The integer number of entries currently in the array. \*/

virtual int sortedGetLength() const = 0;

/\*\* Adds a new entry to this sorted array.

@post If successful, newEntry is stored in the array and

the count of items in the bag has increased by 1.

@param newItem The object to be added as a new entry.

@return True if addition was successful, or false if not. \*/

virtual bool sortedInsert(const ListItemType& newItem) = 0;

/\*\* Removes one occurrence of a given entry from this sorted array,

if possible.

@post If successful, anItem has been removed from the sorted array

and the count of items in the sorted array has decreased by 1.

@param anItem The entry to be removed.

@return True if removal was successful, or false if not. \*/

virtual bool sortedRemove(const ListItemType& anItem) = 0;

/\*\* gets the position anItem belongs

@post If successful, anItem has been removed from the sorted array

and the count of items in the sorted array has decreased by 1.

@param anItem The entry to be located.

@return Returns the position where anItem belongs or exists in this sorted list. The item anItem and the list are unchanged \*/

virtual int locatePosition(const ListItemType& anItem) const = 0;

/\*\* takes the index and saves the value found to dataItem.

@param index The index to be found

@param dataItem If index is 1 ≤ index ≤ sortedGetLength(), dataItem to the item

@return True if retrival was successful, or false if not. \*/

virtual bool sortedRetrive(int index, ListItemType& dataItem) const = 0;

/\* outputs all the datas saved in the arrray \*/

virtual void listSortedArray() const = 0;

};

#endif

**sortedList.h: header file**

#ifndef \_SORTED

#define \_SORTED

#include "stdafx.h"

#include <string>

#include "sortedListInterface.h"

using namespace std;

template<class ListItemType>

class sorted : public sortedInterface<ListItemType>

{

public:

sorted();

bool sortedIsEmpty() const;

int sortedMaxLength() const;

int sortedGetLength() const;

void listSortedArray() const;

bool sortedInsert(const ListItemType& newItem);

bool sortedRemove(const ListItemType& anItem);

int locatePosition(const ListItemType& anItem) const;

bool sortedRetrive(int index, ListItemType& dataItem) const; // nothing should be left unchanged

private:

static const int DEFAULT\_BAG\_SIZE = 10;

ListItemType items[DEFAULT\_BAG\_SIZE]; // array of list items

int itemCount; // current count of list items

int maxItems; // max capacity of the list

};

#endif

**sorted.cpp: source file**

#include <string>

#include "sortedList.h"

template<class ListItemType>

sorted<ListItemType>::sorted() : itemCount(0), maxItems(DEFAULT\_BAG\_SIZE)

{} // end default constructor

template<class ListItemType>

bool sorted<ListItemType>::sortedIsEmpty() const

{

return itemCount == 0;

} // end isEmpty

template<class ListItemType>

int sorted<ListItemType>::sortedMaxLength() const

{

return maxItems;

} // end isEmpty

template<class ListItemType>

int sorted<ListItemType>::sortedGetLength() const

{

return itemCount;

} // end getCurrentSize

template<class ListItemType>

void sorted<ListItemType>::listSortedArray() const

{

for (int i = 0; i < sortedGetLength(); i++)

cout << items[i] << " ";

if (sortedGetLength() == 0)

cout << "Empty list";

cout << endl;

}

template<class ListItemType>

bool sorted<ListItemType>::sortedInsert(const ListItemType& newItem)

{

bool hasRoomToAdd = (itemCount < maxItems);

if (hasRoomToAdd)

{

int count = itemCount;

// compare with the last value and shift based on that

while (count >= 0)

{

if (count != 0)

{

if (items[count - 1] > newItem)//if (strcmp(items[count - 1], newItem)==-1)//

{

items[count] = items[count - 1];

count--;

}

else

{

items[count] = newItem;

break;

}

}

else

{

items[count] = newItem;

break;

}

}

itemCount++;

}

return hasRoomToAdd;

}

template<class ListItemType>

bool sorted<ListItemType>::sortedRemove(const ListItemType& anItem)

{

//ListItemType temp;

int curPos = locatePosition(anItem);

bool canRemove = false;

int count = curPos;// item count value since we increament at last

if (curPos > -1)

{

ListItemType temp;//= items[count - 1];

while (count < itemCount - 1)

{

//temp = items[count - 1];

items[count] = items[count + 1];

//items[count - 1] = ite;

count++;

}

itemCount--;

canRemove = true;

}

return canRemove;

}

template<class ListItemType>

bool sorted<ListItemType>::sortedRetrive(int index, ListItemType& dataItem) const

{

bool success = index >= 1 && index < sortedGetLength();

if (success)

dataItem = items[index];

return success;

}

template<class ListItemType>

int sorted<ListItemType>::locatePosition(const ListItemType& anItem) const

{

bool found = false;

int result = -1;

int searchIndex = 0;

// if the bag is empty, itemCount is zero, so loop is skipped

while (!found && (searchIndex < itemCount))

{

if (items[searchIndex] == anItem)

{

found = true;

result = searchIndex;

}

else

{

searchIndex++;

} // end if

} // end while

return result;

} // end getIndexOf

1. **User Document**

This program has an easy to use procedure, but it has certain constraints that must be followed for successful usage. This program is used to compare strings; reading the instruction could be very much of helpful.

Follow this instructions to run the programs:

* To compile the program, simply enter: g++ mainSorted.cpp
* To run the program, type in: a.out in the command window
* To quit the program Q or q

1. **Test Data Plan**

|  |  |  |
| --- | --- | --- |
| Valid input Values |  |  |
|  | Input: chars I, i, R, r, L, l, G, g, P, p, S, s, Q, q for main menu selection. | Does respective choice |
|  | Input: any strings for inserting strings to the array | It will insert strings to the array if not full, else shows successfully inserted. |
|  | Input: any strings for removing values | It will remove the strings entered, if found, else shows value not found. |
|  | Input: strings for getting positions and retrieving data. | Values will be displayed if found, else displays not found. |
| Boundary values |  |  |
|  | Max array size | 10 |
|  |  |  |
| Invalid input values |  |  |
|  |  |  |
|  | Input: other characters in place of given choices for the main menu. | Error Message: ‘Invalid input, Try again.’ |

1. **Summary**

This program uses two header and two source files for computing and completing the task. The program asks the user for varieties of functionalities including inserting, removing and finding a value from the array.

Sorted inserting is done by comparing the values with the inserted data and shifting stored arrays values to the right; while sorted removing a value from an array, is done by first finding the index number and shifting the arrays to the left until the requested removal index number.

The program could be extended in several ways that would make it more useful for large projects by utilizing permanent storing systems. This can be achieved using a text file or similar data extensions by utilizing libraries like ‘ifstream’ and ‘ofstream.’ Besides utilizing a permanent storage system, we can also use GUI to make it easier and simpler for users.