CPE 212 - Fundamentals of Software Engineering

Unsorted Lists

Outline

- Defining Lists
- List variations
- Sorting
- Smart Insert

List

- List
  - A variable-length, linear collection of like components (homogeneous)
  - A sequence of zero or more components
- List Length
  - The number of components currently stored in the list

List Operations

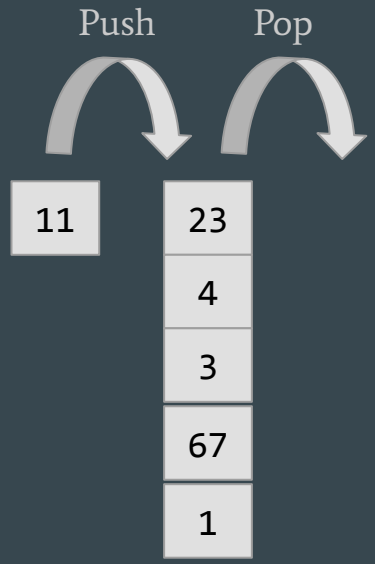
- Create
- Add an item to the list
- Remove an item from the list
- Print the list
- Search the list for a particular item
- Sort the list
- Purge duplicates
- Any others?

List Variations

- Unsorted List

- Sorted List

- Stack



- Queue

List Implementations

Sequential Implementation

- Array-based implementation
- Components stored in contiguous array cells
- Order of components determined by the order in which the elements are stored

Linked Implementation

- Components stored in nodes that may not be consecutive in memory
- Each node typically contains the component data along with a “link” which indicates the location of the next node

Unsorted List Example

```
//*******  List.h Standard Header Information Here **********

#ifndef LIST_CLASS_H_

 #define LIST_CLASS_H_

 const int MAX_LENGTH = 100;       // Maximum number of list items

 typedef int ItemType;     // Data type of each item in list

 class List            // Unordered list of items
 {
   private:
      int  length;         // Actual number of items in list
      ItemType  data[MAX_LENGTH];  // List of unsorted data items

   public:
      List();          // Default constructor creates empty list
      bool IsEmpty() const;        // Returns TRUE if empty, FALSE otherwise
      bool IsFull() const;     // Returns TRUE if full, FALSE otherwise
      int  Length() const;     // Returns length of list
      void Insert(ItemType item);  // Adds item to end of list assuming list is not full
      void Delete(ItemType item);  // Removes first item occurrence from list if not empty
      bool IsPresent(ItemType item);   // Returns TRUE if item in list, otherwise FALSE
 };

#endif  // End of LIST_CLASS_H_
```

Unsorted List Example

```
//*******  List.cpp Standard Header Information Here *********
#include “List.h”

List::List()            // Default constructor creates empty list
{
 length = 0;
} // End List::List()

bool List::IsEmpty() const      // Returns TRUE if empty, FALSE otherwise
{
 return (length == 0);
} // End List::IsEmpty()

bool List::IsFull() const       // Returns TRUE if full, FALSE otherwise
{
 return (length == MAX_LENGTH);
} // End List::IsFull()

int List::Length() const        // Returns length of list
{
 return length;
} // End List::Length()
```

Unsorted List Example

```
//*******  List.cpp continued from previous slide **********

void List::Insert(ItemType item)    // Adds item to end of list assuming list is not full
{
 data[length] = item;
 length++;
} // End List::Insert(…)


void List::Delete(ItemType item)    // Removes first occurrence from list if not empty
{
 int index = 0;

 while ((index < length) && (item != data[index])) // Locate item index first
 {
   index++;
 }

 if (index < length)
 {
   data[index] = data[length-1];   // Overwrite item with last item in the list
   length--;
 }
} // End List::Delete(…)
```

Unsorted List Example

```
//*******  List.cpp continued from previous slide **********


bool List::IsPresent(ItemType item)  // Returns TRUE if item in list,
otherwise FALSE
{
 int index = 0;

 while ((index < length) && (item != data[index])) // Locate item index first
 {
   index++;
 }

 return (index < length);  // index == length => item not found.  Index <
length => item found

} // End List::IsPresent(…)
```

Unsorted List Example main()

```
//*******  SomeClient.cpp continued from previous slide **********
#include <iostream>
#include <fstream>
#include "List.h"
using namespace std;

int main()
{
 List  temps;       // List of temperature values
 ifstream  dataFile;    // Input file stream
 int someTemp;      // One data value

 dataFile.open("tempsdata.txt");

 if (!dataFile)
 {
   cout << "Unable to open data filetempsdata.txt" << endl;
   return 1;
 }

 dataFile >> someTemp; // Priming read
 while ( dataFile && !temps.IsFull() ) // While not EOF or FULL list
 {
   temps.Insert(someTemp);
   dataFile >> someTemp;   // Try to read another temp value from file
 }
 cout << endl << "A total of " <<temps.Length() << " temperatures were input." << endl;
  // Do something useful with your list of temps here…

 return 0;
} // End main()
```

Iterator Example

- Methods that allow a structure such as a List to be processed item by item

```
//*******  List.h Standard Header Information Here **********

#ifndef LIST_CLASS_H_

 #define LIST_CLASS_H_

 const int MAX_LENGTH = 100;       // Maximum number of list items

 typedef int ItemType;     // Data type of each item in list

 class List            // Unordered list of items
 {
   private:
      int  length;         // Actual number of items in list
      int  currentPos;     // Current list position <<============== ADDED
      ItemType  data[MAX_LENGTH];  // List of unsorted data items

   public:
      List();          // Default constructor creates empty list
      bool IsEmpty() const;        // Returns TRUE if empty, FALSE otherwise
      bool IsFull() const;     // Returns TRUE if full, FALSE otherwise
      int  Length() const;     // Returns length of list
      void Insert(ItemType item);  // Adds item to end of list assuming list is not full
      void Delete(ItemType item);  // Removes first item occurrence from list if not empty
      bool IsPresent(ItemType item);   // Returns TRUE if item in list, otherwise FALSE
      void Reset();        // Initializes iteration to position 0 <<===== ADDED
      ItemType GetNextItem();      // Returns value of current item <<===== ADDED
               // and advances currentPos indicator
 };

#endif  // End of LIST_CLASS_H_
```

Iterator Example

```
void List::Reset()       // Initializes iteration to position 0
{
 currentPos = 0;
} // End List::Reset()

ItemType List::GetNextItem()     // Returns value of current item & advances
currentPos
{
 ItemType item;          // Value of current item

 item = data[currentPos];

 if (currentPos == (length-1))  // If at end of list, wrap back to beginning
 {
   currentPos = 0;
 }
 else              // Else advance currentPos to next item
 {
   currentPos++;
 }

 return item;
} // End List::GetNextItem()
```

Iterator Example main()

```
// Sample use of an Iterator in a client program
// #includes and using statement here

int main()       // Initializes iteration
{
 List temps;
 int numItems;

 // Input values into temps here

 // Print every item in List temps
 temps.Reset();
 numItems = temps.Length();
 for (int count = 0; count < numItems; count++)
 {
   cout << temps.GetNextItem() << endl;
 }

 // Something useful here

 return 0;
} // End main()
```

Sorted Lists

Option 1

- Add a sort list method

Option 2

- Modify the Insert method to insert each item into the list in the correct sorted position

Question
Which is the better option?

List Sort Example

```
//*******  List.h Standard Header Information Here **********

#ifndef LIST_CLASS_H_

 #define LIST_CLASS_H_

 const int MAX_LENGTH = 100;       // Maximum number of list items

 typedef int ItemType;     // Data type of each item in list

 class List            // Unordered list of items
 {
   private:
      int  length;         // Actual number of items in list
      int  currentPos;     // Current list position
      ItemType  data[MAX_LENGTH];  // List of unsorted data items

   public:
      List();          // Default constructor creates empty list
      bool IsEmpty() const;        // Returns TRUE if empty, FALSE otherwise
      bool IsFull() const;     // Returns TRUE if full, FALSE otherwise
      int  Length() const;     // Returns length of list
      void Insert(ItemType item);  // Adds item to end of list assuming list is not full
      void Delete(ItemType item);  // Removes first item occurrence from list if not empty
      bool IsPresent(ItemType item);   // Returns TRUE if item in list, otherwise FALSE
      void Reset();        // Initializes iteration
      ItemType GetNextItem();      // Returns value of current item
      void AscendingSort();        // Sorts list items smallest to largest <<===== ADDED
 };

#endif  // End of LIST_CLASS_H_
```

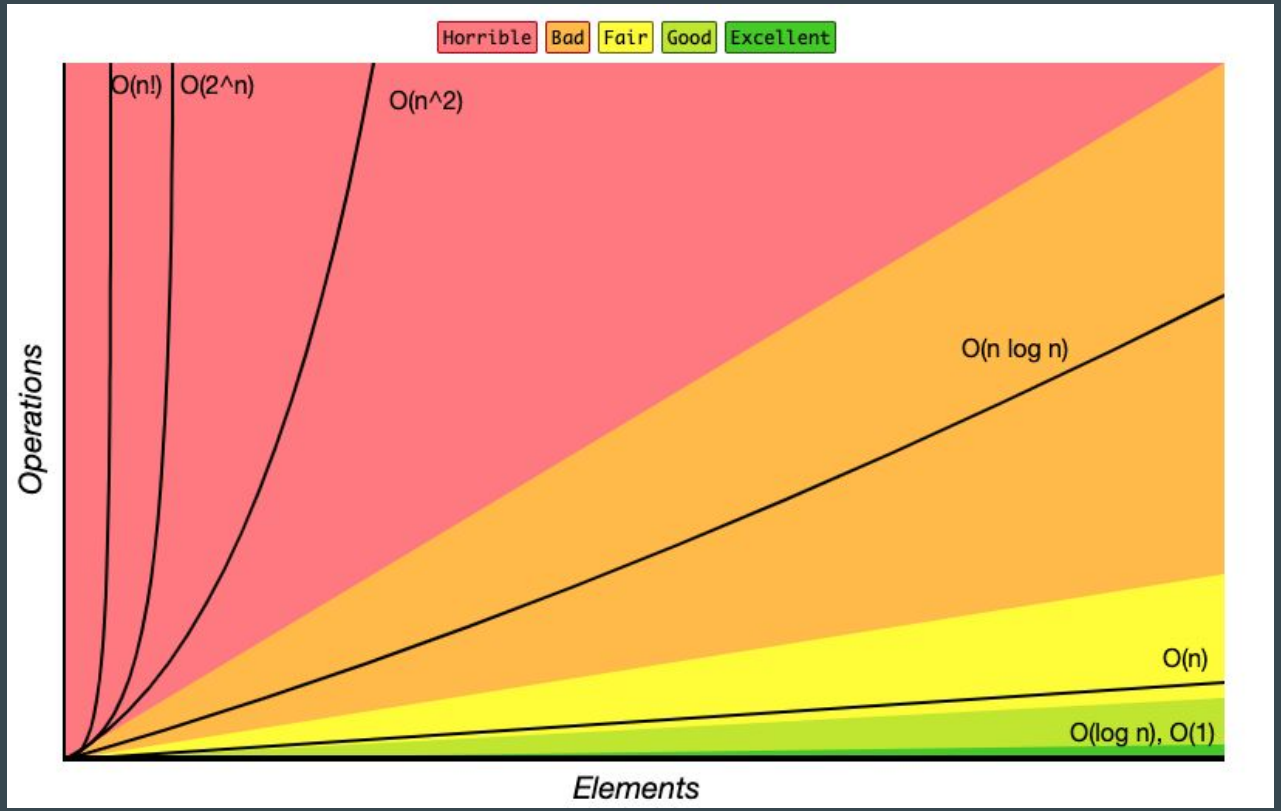
List Sort Example AscendingSort()

- Using the selection sort algorithm
- How efficient is this algorithm?

```
void List::AscendingSort()  // Sorts list items smallest to largest using Selection Sort
{
 ItemType temp;
 int passCount;
 int searchIndx;
 int minIndx;

 for(passCount = 0; passCount < (length-1); passCount++)
 {
   minIndx = passCount;

   for(searchIndx = passCount+1; searchIndx < length; searchIndx++)
   {
     if (data[searchIndx] < data[minIndx])     // Smaller value found, update minIndx
     {
       minIndx = searchIndx;
     }
   }

   temp = data[minIndx];       // Move smallest found on this pass into position
   data[minIndx] = data[passCount];
   data[passCount] = temp;
 }
} // End List::AscendingSort()
```

Big-O Complexity Chart

- Selection Sort is O(n^2)



Sorted List Example

```
//*******  SortedList.h Standard Header Information Here **********

#ifndef SORTED_LIST_CLASS_H_

 #define SORTED_LIST_CLASS_H_

 const int MAX_LENGTH = 100;       // Maximum number of list items

 typedef int ItemType;     // Data type of each item in list

 class SortedList          // Ordered (sorted) list of items
 {
   private:
      int  length;         // Actual number of items in list
      int  currentPos;     // Current list position
      ItemType  data[MAX_LENGTH];  // List of unsorted data items

   public:
      SortedList();        // Default constructor creates empty list
      bool IsEmpty() const;        // Returns TRUE if empty, FALSE otherwise
      bool IsFull() const;     // Returns TRUE if full, FALSE otherwise
      int  Length() const;     // Returns length of list
      void Insert(ItemType item);  // Adds item to list assuming list is not full <=== Modify
      void Delete(ItemType item);  // Removes first item occurrence from list if not empty
      bool IsPresent(ItemType item);   // Returns TRUE if item in list, otherwise FALSE
      void Reset();        // Initializes iteration to position 0
      ItemType GetNextItem();      // Returns value of current item
 };

#endif  // End of SORTED_LIST_CLASS_H_
```

Sorted List Example Insert()

- There is no Sort method since all of the inserts are accounting for order

```
void SortedList::Insert(ItemType item)      // Inserts element into sorted
array
{
 int index;

 index = length-1;
 // While looking for correct place to put the
 // new item, bump each item on the end down
 // one position to make room for new item
 while ((index >= 0) && (item < data[index]))
 {
   data[index+1] = data[index];
   index--;
 }
 data[index+1] = item;
 length++;
} // End SortedList::Insert()
```

Linked List

Same basic operations as other lists

- Create
- Insert
- Delete
- IsFull
- IsEmpty
- Implementation of these methods is different

Array-based implementation of Linked Lists

- Each element of the array is a node
- Each node contains a data item and a link to the next item in the list
- Adjacent items in the list may not be adjacent within the array
- Need to know where in the list begins within the array

Array-Based Linked List Example

Head = 2

```
struct NodeType
{
  ItemType component;
  int nextItem;
};
```

	component	nextItem
node[0]	58	-1
node[1]		
node[2]	4	5
node[3]		
node[4]	46	0
node[5]	16	7
node[6]		
node[7]	39	4

Link List Example

```
//*******  LList.h Standard Header Information Here **********

#ifndef LLIST_CLASS_H_
 #define LLIST_CLASS_H_

 const int MAX_LENGTH = 100;       // Maximum number of list items
 typedef int ItemType;     // Data type of each item in list
  struct NodeType           // Node record with data field and next item field
 {
   ItemType value;
   int nextItem;
 };
 class LList           // Unordered Linked List of items
 {
   private:
      int  length;         // Actual number of items in list <<==== ADDED
      int  head;           // Index of first node in list <<==== ADDED
      NodeType  node[MAX_LENGTH];  // List of unsorted data items

   public:
      LList();         // Default constructor creates empty list
      bool IsEmpty() const;        // Returns TRUE if empty, FALSE otherwise
      bool IsFull() const;     // Returns TRUE if full, FALSE otherwise
      int  Length() const;     // Returns length of list
      void Insert(ItemType item);  // Adds item to end of list assuming list is not full
      void Delete(ItemType item);  // Removes first item occurrence from list if not empty
      bool IsPresent(ItemType item);   // Returns TRUE if item in list, otherwise FALSE
 };

#endif  // End of LLIST_CLASS_H_
```

Link List Example

```
//*******  LList.cpp Standard Header Information Here **********

#include “LList.h”

LList::LList()          // Default constructor creates empty list
{
 length = 0;
} // End LList::List()



bool LList::IsEmpty() const     // Returns TRUE if empty, FALSE otherwise
{
 return (length == 0);
} // End LList::IsEmpty()



bool LList::IsFull() const      // Returns TRUE if full, FALSE otherwise
{
 return (length == MAX_LENGTH);
} // End LList::IsFull()



int LList::Length() const       // Returns length of list
{
 return length;
} // End LList::Length()
```

Link List Example

```
//*******  LList.cpp continued from previous slide **********


bool LList::IsPresent(ItemType item)  // Returns TRUE if item in list,
otherwise FALSE
{
 int index = head;    // Start looking at the beginning of the linked list

 while ((index != -1) && (item != data[index])) // Locate item index first
 {
   index = data[index].nextItem; // Follow the link to the next item in list
 }

 return (index != -1);  // index == length => item not found.  Index == -1 =>
item found

} // End LList::IsPresent(…)
```

Link List Example

```
//*******  LList.cpp continued from previous slide **********

void LList::Insert(ItemType item)   // Adds item to end of list assuming list is not full
{

// ???
// Must now worry about memory management since items are no longer stored in consecutive
// memory cells
//
// If list is empty, must also set nextItem link of new item to -1 to terminate the list
// properly.
 } // End LList::Insert(…)


void List::Delete(ItemType item)    // Removes first occurrence from list if not empty
{

// ???
// Must now worry about memory management since items are no longer stored in consecutive
// memory cells.

} // End LList::Delete(…)
```