Announcements

• Programming Assignment 1:
  – Due Friday April 16, 2010

• CMPSC 24 HelpDesk:
  – Sunday through Thursday: 7PM-9PM
  – Send mail to cs24@cs
  – One of the TAs will be on-call to answer queries
  – Intended primarily for debugging help

• HelpDesk versus Office Hours:
  – discuss
Example linkedlist.h

```cpp
#include <iostream>  // compiler directives
using namespace std;  // compiler directives

Class linkedlist
{
    private:
        struct node
        {
            int data;
            node *link;
        } *list;
    public:
        linkedlist();  // Constructor
        void append(int num);  // append a node with "num"
        void delete(int num);  // delete the node containing "num"
        bool find(int num);  // true if "num" exists in the list
        ~linkedlist();  // Destructor (to ensure no memory leak)
};

Skeleton of linkedlist.cpp

linkedlist::linkedlist()
{
    pNilList;
}

void linkedlist::append(int num)
{
    // the code we developed goes in here
}

void linkedlist::delete(int num)
{
    // the code we developed goes in here
}

bool linkedlist::find(int num)
{
    // the code we developed goes in here
}

Linkedlist::~linkedlist()
{
    // ask the students what should happen here?
}
Usage of linkedlist Class

```c++
int main()
{
    linkedlist myList;
    myList.append(35);
    myList.append(45);
    myList.append(37);
    if (myList.find(36)) cout << "36 on the list???" << endl;
    else cout << "36 not on the list!!!" << endl;

    // you should get the idea...
    // would it be nice to have myList.display() ??? Left as an exercise!!!
}
```

Recap

- Abstract Data Types
  - Specification (What?)
  - Implementation (How?)

LIST ADT
Sorted and Unsorted List ADTs

**UNSORTED LIST**
Elements are placed into the list in no particular order.

**SORTED LIST**
List elements are in a sorted order—either numerically or alphabetically by the elements themselves, or by a component of the element (called a KEY member).

Name some possible keys:

---

ADT Unsorted List

- **Transformers**
  - MakeEmpty
  - InsertItem
  - DeleteItem

- **Observers**
  - IsFull
  - GetLength
  - RetrieveItem

- **Iterators**
  - ResetList
  - GetNextItem

---

ADT Unsorted List

Common vocabulary:
- **location** accesses a particular element
- **Node(location)** is all data of element
- **Info(location)** is the user’s data at location
- **Next(location)** is the node following Node(location)

Two implementations:

---
### Specification
```c++
#include "ItemType.h"

class UnsortedType // declares a class data type
{
public:
    // 8 public member functions
    UnsortedType();
    void MakeEmpty();
    bool IsFull() const; // returns length of list
    int GetLength() const; // returns length of list
    void RetrieveItem(ItemType& item, bool& found);
    void InsertItem(ItemType item);
    void DeleteItem(ItemType item);
    void ResetList();
    void GetNextItem(ItemType& item);
}
```

Public declarations are the same for both implementations, only private data changes.

### Generic Data Type
- A type for which the operations are defined but the types of the items being manipulated are not defined
- How can we make the items on the list generic?
- List items are of class ItemType, which has a ComparedTo function that returns (LESS, GREATER, EQUAL)

### Array-Based Implementation
Private data members for array-based implementation
```c++
private
    int length;
    ItemType info[MAX_ITEMS];
    int currentPos;
};
```

Where does MAX_ITEMS come from?
Array-Based Implementation

Notice the difference between the array and the list stored in the array.

Constructor

A special member function of a class that is implicitly invoked when a class object is defined.

What should the constructor do?

```cpp
UnsortedType::UnsortedType()
{
    length = 0;
}
```

Checking for full and empty lists

- What is a full list? An empty list?

```cpp
bool UnsortedType::IsFull()
{
    return (length == MAX_ITEMS);
}

bool UnsortedType::IsEmpty()
{
    return (length == 0);
}
```
If the list is unsorted, where should the next element go?

insert("Doe");

That was easy! Can you code it?

Insert

length 4
info[0] Maxwell
[1] Bradley
[2] Asad
[3] Doe
[MAX_ITEMS-1]
RetrieveItem

How would you go about finding an item in the list?
Cycle through the list looking for the item

What are the two ending cases?
The item is found
The item is not in the list

How do we compare items?
We use function ComparedTo in class ItemType

Pseudocode for RetrieveItem

Initialize location to position of first item
Set found to false
Set moreToSearch to (have not examined Info(last))
while moreToSearch AND NOT found
  if (item.ComparedTo(Info(location))) == EQUAL
    { Set found to true
      Set item to Info(location)
    }
  else
    { Set location to Next(location)
      Set moreToSearch to (have not examined Info(last))
    }

Replace bold general statements with array-based code

C++ code for RetrieveItem

```cpp
void UnsortedType::RetrieveItem(ItemType item, bool& found)
// Pre: Key member(s) of item is initialized.
// Post: If found, item’s key matches an element’s key in the
//       list and a copy of that element has been stored in item;
//       otherwise, item is unchanged.
{ bool moreToSearch;
  int location = 0;
  found = false;
  moreToSearch = (location < length);
  while (moreToSearch && !found){
    if (item.ComparedTo(Info[location])) == EQUAL{
      found = true;
      item = Info[location];
    }
    else{
      location++;
      moreToSearch = (location < length);
    }
  }
}
```
C++ code for RetrieveItem

```cpp
void UnsortedType::RetrieveItem(ItemType& item, bool& found)
// Pre: Key member(s) of item is initialized.
// Post: If found, item's key matches an element's key in the
//       list and a copy of that element has been stored in item;
//       otherwise, item is unchanged.
{
    bool moreToSearch;
    int location = 0;
    found = false;
    moreToSearch = (location < length);
    while (moreToSearch && !found)
    {
        if (item.ComparedTo(info[location]) == EQUAL)
        {
            found = true;
            item = info[location];
        }
        else
        { location++;
            moreToSearch = (location < length);
        }
    }
    Loop invariant:
    (location <= length)
    and
    moreToSearch == (location < length)
    and
    (!found → (ItemType.key not in Info[0..location-1]))
    and
    found → ItemType.key == Info[location].key
```}

Delete

How do you delete an item?

First you find the item

Yes, but how do you delete it?

Move those below it up one slot, or

Replace it with another item

What other item?

How about the item at info[length-1]?

C++ code for delete

```cpp
void UnsortedType::DeleteItem ( ItemType item )
// Pre: Item's key has been initialized.
// Post: No element in the list has a key that matches item's.
{
    int location = 0;
    while (item.ComparedTo (info[location]) != EQUAL)
    { location++;
        // move last element into position where item was located
        info[location] = info[length - 1 ];
        length--;
    }
    Why don't we have to check for end of list?
```
C++ code for delete

```cpp
void UnsortedType::DeleteItem ( ItemType  item )
// Pre: item's key has been initialized.
// Post: No element in the list has a key that matches item's.
{
    int location = 0 ;
    while (item.ComparedTo (info[location]) != EQUAL)
        location++;
    // move last element into position where item was located
    info [location] = info [length - 1] ;
    length-- ;
}
```

Why don't we have to check for end of list?

Loop invariant:
(0 < length) and (item.key in info[location..length-1])

PrintList

```cpp
void PrintList(ofstream & dataFile, UnsortedType list)
// Pre:  list has been initialized.
// dataFile is open for writing.
// Post: Each component in list has been written.
// dataFile is still open.
{
    int length;
    ItemType item;
    list.ResetList();
    length = list.GetLength();
    for (int counter = 1; counter <= length; counter++)
    {
        list.GetNextItem(item);
        item.Print(dataFile);
    }
}
```

How do ResetList and GetNextItem work?

ResetList and GetNextItem

```cpp
void UnsortedType::ResetList ( )
// Pre:  List has been initialized.
// Post: Current position is prior to first element.
{
    currentPos = -1 ;
}

void UnsortedType::GetNextItem ( ItemType&  item )
// Pre:  List has been initialized. Current position is defined.
//       Element at current position is not last in list.
// Post: Current position is updated to next position.
//       item is a copy of element at current position.
{
    currentPos++ ;
    item = info [currentPos] ;
}
```
Class `ItemType`

// SPECIFICATION FILE itemtype.h

const int MAX_ITEM = 5;
enum RelationType { LESS, EQUAL, GREATER };
class ItemType // declares class data type
{
public:
    // 3 public member functions
    RelationType ComparedTo(ItemType otherItem) const;
    void Print() const;
    void Initialize(int number);

private:
    // one private data member
    int value; // could be any type
};

Class `ItemType`

// IMPLEMENTATION FILE (itemtype.cpp)

#include "itemtype.h"
#include <iostream>

RelationType ComparedTo(ItemType otherItem) const
{
    if (value < otherItem.value)
        return LESS;
    else if (value > otherItem.value)
        return GREATER;
    else return EQUAL;
}

void Print() const
{
    using namespace std;
    cout << value << endl;
}

void Initialize(int number)
{
    value = number;

How would this class change if the items on the list were strings rather than integers?

UML Diagram