Chapter 3

ADT Unsorted List

Outline

- Abstract Data Type
- Unsorted List
- Array-based Implementation
- Linked Implementation
- Comparison

Different Views of Data

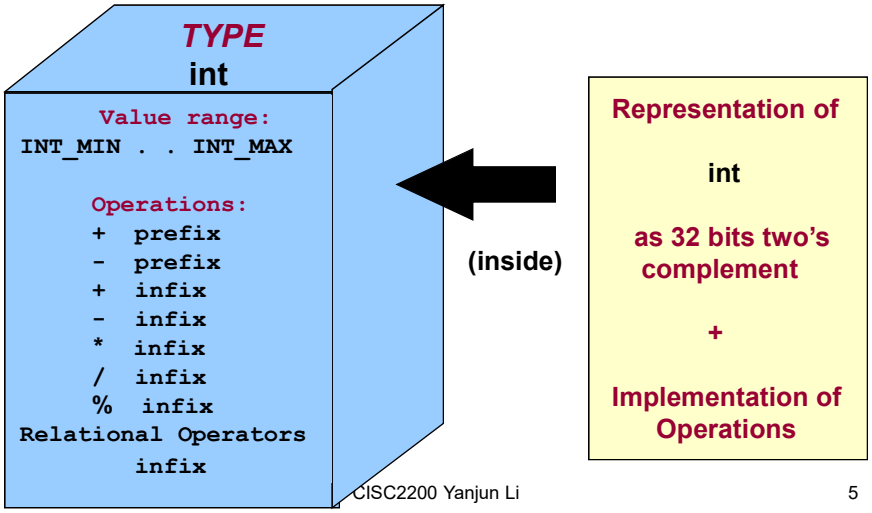
- **Data**
  - The representation of information in a manner suitable for communication or analysis by humans or machines
- Data are the **nouns** of the programming world:
  - The objects that are manipulated
  - The information that is processed

Different Views of Data

- **Data Abstraction**
- Separation of a data type's logical properties from its implementation

LOGICAL PROPERTIES	IMPLEMENTATION
What are the possible values?	**How can this be done in C++?**
What operations will be needed?	**How can data types be used?**

Different Views of Data



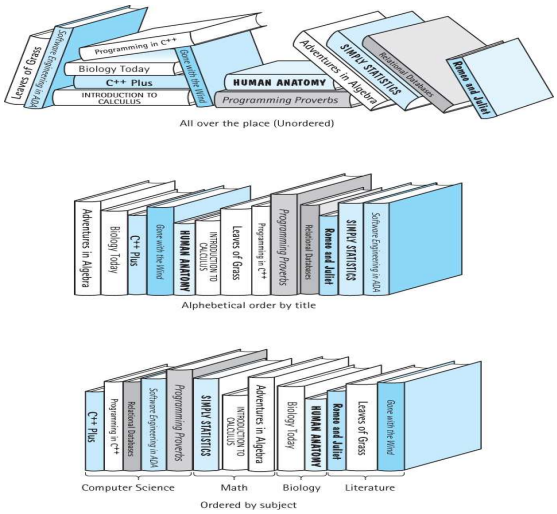
Different Views of Data

- **Abstract Data Type**
  - is a specification of
    - a set of data sharing the same features
    - the set of operations that can be performed on the data
  - Implementation independent.
- A ***Data Structure*** is an concrete implementation of an ***Abstract Data Type.***

Different Views of Data

- **Application (or user) level** modeling real-life data in a specific context
- **Logical (or ADT) level** abstract view of the domain and operations
- **Implementation level** specific representation of the structure to hold the data items, and the coding for operations

Different Views of Data



What is the application view?

The logical view?

The implementation view?

Different Views of Data

- **Application (or user) level** Library of Congress, or Baltimore County Public Library
- **Logical (or ADT) level** domain is a collection of books; operations include: check book out, check book in, pay fine, reserve a book
- **Implementation level** representation of the structure to hold the "books" and the coding for operations

Different Views of Data

- ***Application (or user) level:*** **modeling real-life data in a specific context.**
- ***Logical (or ADT) level:*** **abstract view of the domain and operations.** **WHAT**
- ***Implementation level:*** **specific representation of the structure to hold the data items, and the coding for operations.** **HOW**

ADT Operators (operations)

- **Constructors**
  - Operation that creates new instances of an ADT; usually a language feature
- **Transformers (mutators)**
  - Operations that change the state of one or more data values in an ADT
- **Observers**
  - Operations that allow us to observe the state of the ADT
- **Iterators**
  - Operations that allow us to access each member of a data structure sequentially

Lists

- **Linear relationship**
  - Each element except the first has a unique predecessor, and each element except the last has a unique successor
- **Length**
  - The number of items in a list; the length can vary over time
- **Unsorted list**
  - A list in which data items are placed in no particular order; the only relationships between data elements are the list predecessor and successor relationships

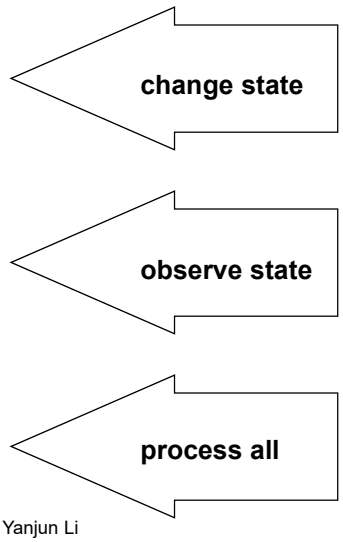
Lists

- **Sorted list**
  - A list that is sorted by the key; there is a semantic relationship among the keys of the items in the list
- **Key**
  - The attributes that are used to determine the logical order of the list

Name some possible keys

ADT Unsorted List

- **Transformers**
  - **MakeEmpty**
  - **InsertItem**
  - **DeleteItem**
- **Observers**
  - **IsFull**
  - **GetLength**
  - **RetrieveItem**
- **Iterators**
  - **ResetList**
  - **GetNextItem**



ADT Unsorted List

- Client is responsible for error checking.
  - Precondition of each function enforces it.
  - Provide clients the tools with which to check for the conditions.
    - Observers

ADT Unsorted List

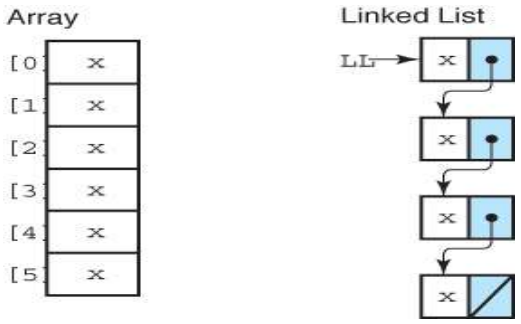
- **Generic data type**
  - A type for which the operations are defined but the types of the items being manipulated are not defined
  - *How can we make the items on the list generic?*
  - List items are of class `ItemType`, which has a `ComparedTo` function that returns `LESS`, `GREATER`, `EQUAL`
    - `ComparedTo` function compares the keys of items.

Class ItemType

```
//  SPECIFICATION FILE ItemType.h
#ifndef ITEMTYPE_H
#define ITEMTYPE_H

const int  MAX_ITEM = 5;
enum  RelationType { LESS, EQUAL, GREATER };

class  ItemType          // declares class data type
{
public :                 // 4 public member functions
  ItemType(){}
  RelationType ComparedTo( ItemType ) const{..}
  void Print( )  const{..}
  void Initialize( int value ){…}

private :                // 1 private data member
   int value;            // could be any type
} ;
#endif
```


ADT Unsorted List

- **Implementation Level**



Two implementations

```
// SPECIFICATION FILE              ( UnsortedType.h )
#include "ItemType.h"

class  UnsortedType        // declares a class data type
{
public :

  UnsortedType();

  void MakeEmpty( );         // 8 public member functions
  bool IsFull( ) const;
  bool IsEmpty( ) const;
  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 either implementation; only private data changes

What is ItemType?

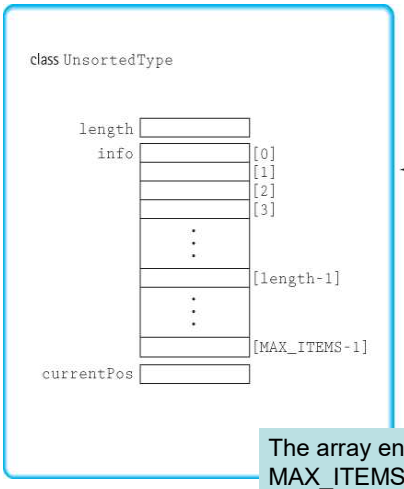
Array-Based Implementation

Private data members for array-based implementation

```
private :
   int length;
   ItemType info[MAX_ITEMS];
   int currentPos;
};
```

Where does MAX_ITEMS come from?

Array-Based Implementation



Logical List items stored in an array

Array-based implementation

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

The array ends at the slot with index MAX_ITEMS – 1 ; the list ends in the slot with index length - 1

ADT Unsorted List

- **Constructor**
  - A special member function of a class that is implicitly invoked when a class object is defined

```
UnsortedType();
```

- *What should the constructor do?*

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

Array-Based Implementation

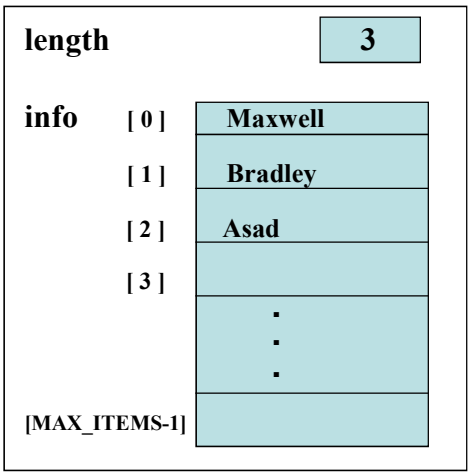
- *What is a full list? An empty list?*

```
//pre: List has been initialized
//post: function value = (list is full)
bool UnsortedType::IsFull() const
{
  return (length == MAX_ITEM);
}

//pre: List has been initialized
//post: function value = (list is empty)
bool UnsortedType:: IsEmpty() const
{
  return (length == 0);
}
```


Member Function

```
//pre: List has been initialized
//post:
int UnsortedType::GetLength()
  const
{
   return length ;
}
```

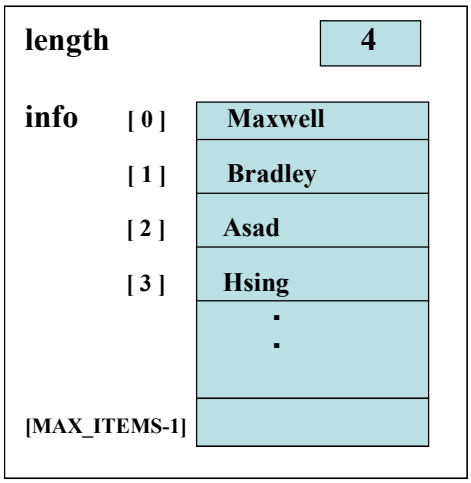
Array-Based Implementation



insert("Hsing");

If the list is unsorted, where should the next element go ?

Array-Based Implementation



That was easy! Can you code it ?

Array-Based Implementation

```
//pre: List has been initialized. List is not
//     full, item is not in list.
// Post: item is in the list.

void UnsortedType::InsertItem(ItemType item)
{
  info[length] = item;
  length++;
}
```


Array-Based Implementation

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 in the list - found.
- The item is not in the list – not found.

How do we compare items?

We use function **`ComparedTo`** in class **`ItemType`**

`(compare keys of items)`

Array-Based Implementation

```
// 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.
void UnsortedType::RetrieveItem(ItemType& item, bool& found)
{
      found  =  false;
      for ( int i=0; i<length; i++)
     {
            if (item.ComparedTo(info[i]) == EQUAL)
            {
                  item = info[i];
                  found = true;
                  break;
            }
     }
}
```


Array-Based Implementation

How do you <u>delete</u> an item?

First you find the item

Yes, but <u>how</u> do you delete it?

1. Move those below it up on slot Or
2. Replace it with another item

** Tips: This is an unsorted list.*

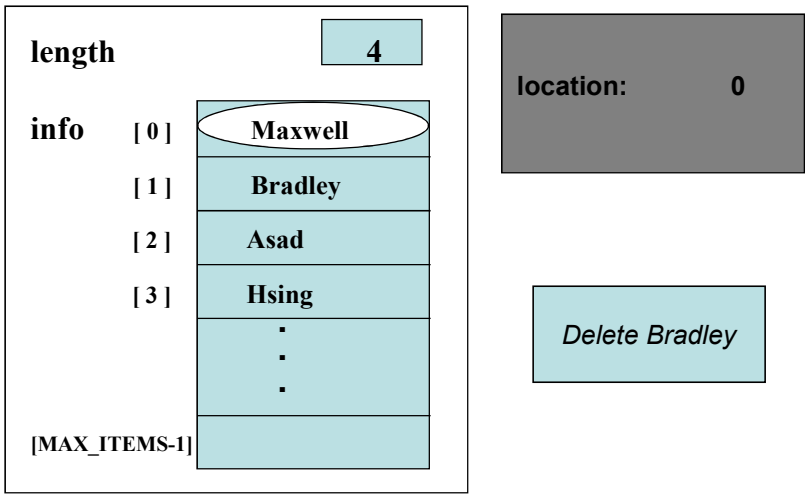
What <u>other</u> item?

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

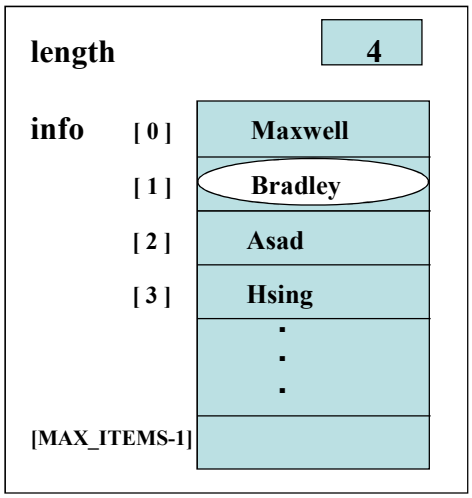
Array-Based Implementation

```
// Pre: item's key has been inititalized.
//      One and only one element in the list has a key that
//       matches item's.
// Post: No element in the list has a key that matches item's.
void UnsortedType::DeleteItem ( ItemType  item )
{
  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-- ; //the length of the list is decreased
}
```


Array-Based Implementation



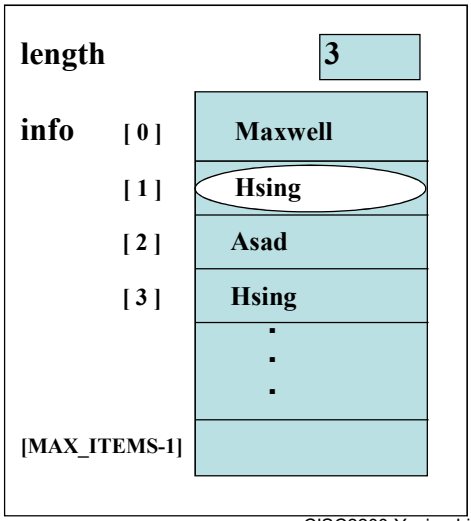
Array-Based Implementation



location: 1

Key Bradley has been matched

Array-Based Implementation



location: 1

Copy of last list element is in the position where the key Bradley was before; length has been decremented

Array-Based Implementation

- *What will happen if the matching element is the last element ?*
- *What will happen if there is only one element in the list ?*
- *Do we worry about the empty List ?*
- *What if the elements are pointers and they points to dynamically allocated memory ?*

Array-Based Implementation

```
// Pre:  N/A
// Post: the list is empty
bool UnsortedType::MakeEmpty()
{
    length = 0;
}
```

- We do not have to do anything to the array that holds the list items to make a list empty. *What if the items are pointers?*

Array-Based Implementation

```
// Pre:  List has been inititalized.
// Post: Current position is prior to first element.
void UnsortedType::ResetList ( )
{
  currentPos  =  -1 ;
}
// 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.
void UnsortedType::GetNextItem(ItemType&  item)
{
  currentPos++  ;
  item = info [currentPos] ;
}
```

Iterators

Array-Based Implementation

- **Application Level**
- How to use the List implemented?
  - PrintList(UnsortedType&)
  - CreateFromFile(ifstream &, UnsortedType&)
- Users do not need to know how the list is implemented.

Testing Array-Based Implementation

```
// Pre:  list has been initialized.
//
// Post: Each component in list has been written.
//
void PrintList(UnsortedType & list)
{
  int length;
  ItemType item;

  list.ResetList( );
  length = list.GetLength( );
  for (int counter = 1; counter <= length; counter++)
  {
    list.GetNextItem(item);
    item.Print();
  }
}
```


Test Plan

- Testing Strategy
  - Combination of black-box and clear-box.
  - Black-box : test precondition & postcondition.
  - Clear-box : test the code inside the functions, try path testing.
- Values for testing
  - End cases.

Data Encapsulation

- Information internal to the implementation of the Unsorted List ADT.
  - Private data members
    - The array `info[ ]`
    - The index of the iterator `currentPos`
  - Local variable `location`, which contains the array index of the list item being processed.

Reference

- Reproduced from C++ Plus Data Structures, 4th edition by Nell Dale.
- **Reproduced by permission of Jones and Bartlett Publishers International.**