

CSE1303 Part A
Data Structures and Algorithms
Summer Semester 2003

Lecture A5 – Basic Data Structures – Continued (Lists)

Kymberly Fergusson

Basic Data Types

- ✓ Stack
 - ✓ Last-In, First-Out (LIFO)
 - ✓ initialize, push, pop, status
- ✓ Queue
 - ✓ First-In, First-Out (FIFO)
 - ✓ initialize, append, serve, status
- ⇨ List

2

Overview

- Programming Styles.
- Abstract Data Types (ADT).
- Lists.
- Operations.
- Implementations of Lists.

3

Programming Styles

- Procedural Programming
Decide which procedures you want: use the best algorithms you can find.
- Modular Programming
Decide which modules you want: partition the program so that data is hidden in modules.
- Data Abstraction
Decide which types you want: provide a full set of operations for each type.
- Object-Oriented Programming
Decide which classes you want: provide a full set of operations for each class; make commonality explicit by using inheritance.

4

Abstract Data Type (ADT)

- A Data Structure together with operations defined on it.
- Useful to consider what operations are required before starting implementation.
- Led to the development of object oriented programming.

5

A Stack ADT

A sequence of elements together with these operations:

- Initialize the stack.
- Determine whether the stack is empty.
- Determine whether the stack is full.
- Push an item onto the top of the stack.
- Pop an item off the top of the stack.

6

A Queue ADT

A sequence of elements together with these operations:

- Initialize the queue.
- Determine whether the queue is empty.
- Determine whether the queue is full.
- Find the size of the queue.
- Append an item to the rear of the queue.
- Serve an item at the front of the queue.

7

A List ADT

A sequence of elements together with these operations:

- Initialize the list.
- Determine whether the list is empty.
- Determine whether the list is full.
- Find the size of the list.
- Insert an item anywhere in the list.
- Delete an item anywhere in a list.
- Go to a particular position in a list.

8

Comparison

- Stacks
 - Insert at the top of the stack (push)
 - Delete at the top of the stack (pop)
- Queues
 - Insert at the rear of the queue (append)
 - Delete at the front of the queue (serve)
- Lists
 - Insert at any position in the list.
 - Delete at any position in the list.

9

A Rational Number ADT

Two integers together with these operations:

- Initialize the rational number.
- Get the numerator.
- Get the denominator.
- Simplify a rational number.
- Add two rational numbers.
- Determine whether two rational numbers are equal.
- etc.

10

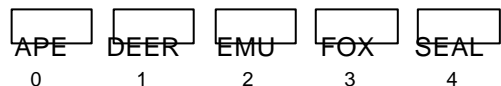
A String ADT

An array of characters together with these operations:

- Initialize the string.
- Copy a string.
- Read in a line of input.
- Concatenate two strings.
- Compare two strings.
- Find a length of a string.
- etc.

11

Lists



- Kept in alphabetical or numerical order
- No spaces between elements
- Simple lists are implemented in arrays

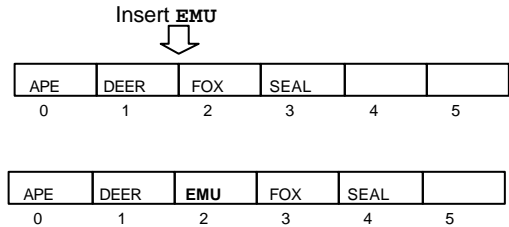
12

List Operations

- Initialize the list.
- Determine whether the list is empty.
- Determine whether the list is full.
- Find the size of the list.
- Insert an item anywhere in the list.
- Delete an item anywhere in a list.
- Go to a particular position in a list.

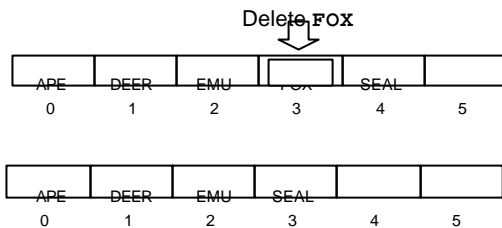
13

Simple List Implementation Insertion



14

Simple List Implementation Deletion



15

```
#ifndef LISTH
#define LISTH

#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>

#define MAXLIST 10

struct ListRec
{
    char list[MAXLIST];
    int count;
};

typedef struct ListRec List;

void Initialise(List *lptr);
void Insert(List *lptr, char item);
void Delete(List *lptr, int pos);
bool IsFull(List *lptr);
bool IsEmpty(List *lptr);
int Find(List *lptr, char item);

/* and any more functions you can think of) */

#endif
```

16

```
#include <stdio.h>
#include "list.h"

void Initialise(List *lptr)
{
    lptr->count = 0;
}

bool isFull(List * lptr)
{
    if(lptr->count == MAXLIST)
        return true;
    else
        return false;
}

bool isEmpty(List *lptr)
{
    if(lptr->count == 0)
        return true;
    else
        return false;
}
```

17

List Insert

- If the list isn't full:
 - step to position to insert item
 - move elements after this position along one position starting from the end of the list, working back to the position.
 - copy item into the position
 - increase count

18

```

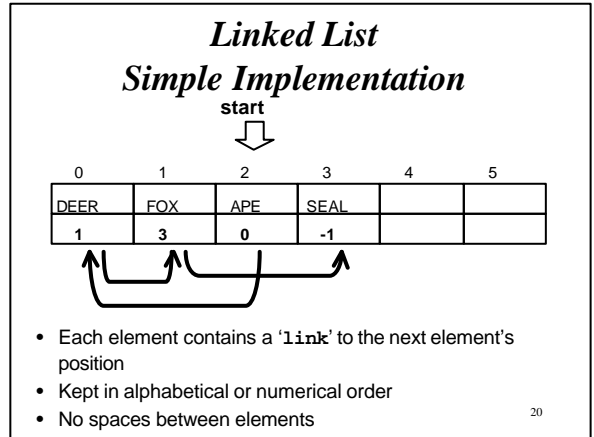
void Insert(List *lptr, char item){
    int pos, current;

    if(IsFull(lptr))
    {
        fprintf(stderr, "List is full.");
        exit(1);
    }

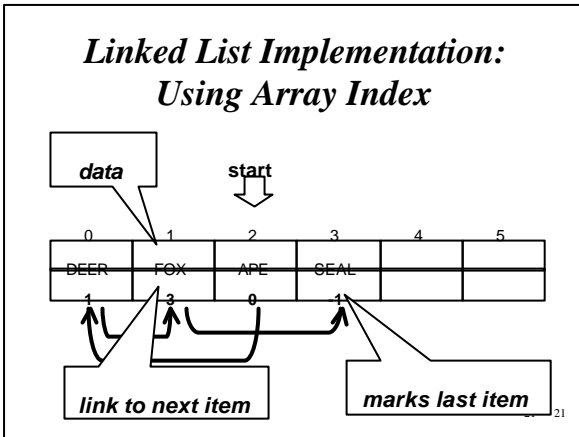
    for(pos = 0; pos < lptr->count; pos++)
    {
        if(lptr->list[pos] > item)
            break;
    }

    current = lptr->count - 1;
    while (current >= pos)
    {
        /* Copy the element into the next position along */
        lptr->list[current+1] = lptr->list[current];
        current--;
    }
    lptr->list[pos] = item;
    lptr->count++;
}
    
```

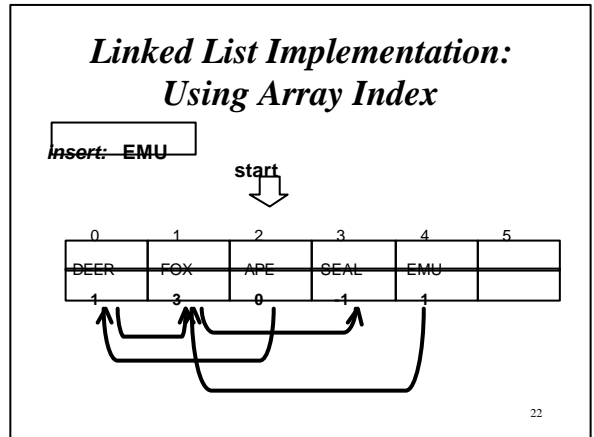
19



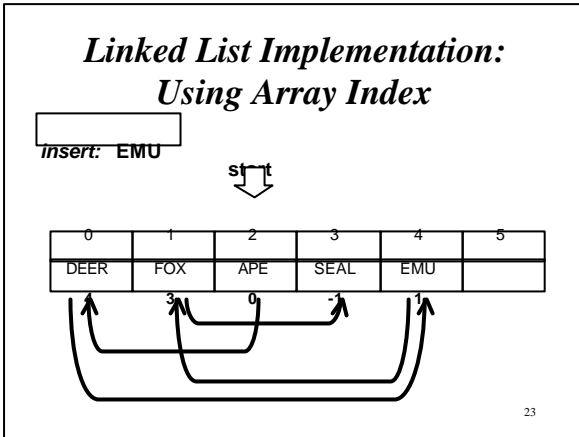
20



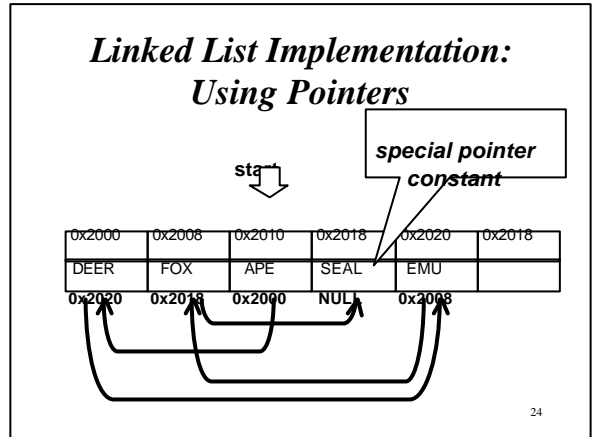
21



22



23



24

```
#ifndef LLISTH
#define LLISTH
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#define MAXLIST 10

typedef struct EntryRec
{
    char ch;
    int next;
} Entry;

typedef struct LinkedListRec
{
    Entry list[MAXLIST];
    int count;
    int start;
} LList;

void Initialise(LList *lptr);
void Insert(LList *lptr, char item);
void Delete(LList *lptr, int pos);
bool IsFull(LList *lptr);
bool IsEmpty(LList *lptr);
int Find(LList *lptr, char item);
/* and any more functions you can think of) */
#endif
```

25

```
#include <stdio.h>
#include "l1ist.h"

void Initialise(LList *lptr)
{
    lptr->count = 0;
    lptr->start = -1;
}

bool isFull(LList *lptr)
{
    if(lptr->count == MAXLIST)
        return true;
    else
        return false;
}

bool isEmpty(LList *lptr)
{
    if(lptr->count == 0)
        return true;
    else
        return false;
}
```

26

Linked List Insert

- If the list isn't full:
 - insert item at next free position.
 - step through list to find the element that comes before item and the one after.
 - change the new item's next value to be the position of the next element.
 - if the new item is not the first element
 - change the previous element's next value to be the position of the new item.
 - if the new item is the first element, change the start of the list to equal the new item's position.
 - increase count.

27

Linked List Implementations

- There are many different types of implementations for linked lists in arrays.
 - Free positions always at the end (count), when deleting need to shuffle items to close the gap
 - Using -2 to indicate a 'free' position – still slow to find a free spot when inserting
 - Using a second linked list – a 'free list' inside the same array – requires a second 'start' indicator. This is the most common implementation.

28

Revision

- List
 - initialize, insert, delete, position, status
 - implementations
- Difference between Stacks, Queues, and Lists.
- ADT.

Revision: Reading

- Kruse 4.5, 4.6, 4.8
- Deitel & Deitel (2e) 12.1, 12.2, 12.4
- Standish 8.1 – 8.4
- Langsam 4.2 – 4.5
- King 17.5

Preparation

Next lecture: *Dynamic Memory*

- Read 4.5 in Kruse et al.
- Read 12.3 in Deitel and Deitel

29