

CSE1303 Part A
Data Structures and Algorithms
Summer Semester 2003

Lecture A7 – Nodes and Linked Structures

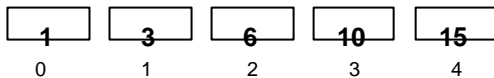
Kymberly Fergusson

Overview of Topic

- Review List Implementations. } Today's Lecture
- Nodes.
- Linked Stacks.
- Linked Queues
- Linked Lists.
- Other List Operations

2

Lists



3

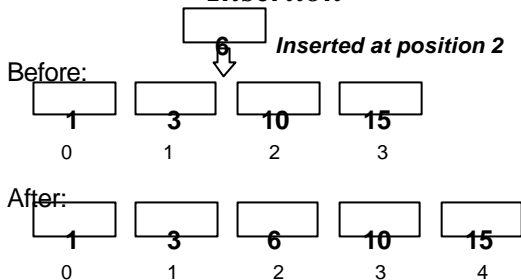
A List ADT

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.

4

Insertion



5

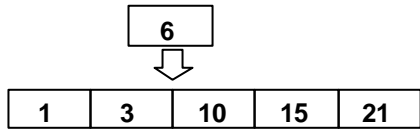
A List ADT

A sequence of elements together with these operations:

- Initialize the list.
- Determine whether the list is empty.
- Find the size of the list.
- Insert an item anywhere in the list. } List needs to be able to expand
- Delete an item anywhere in a list.
- Go to a particular position in a list.

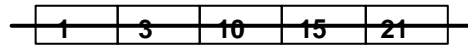
6

Simple List Implementation

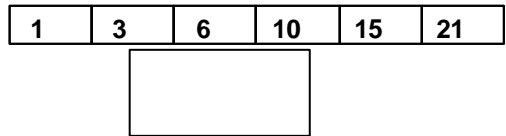


7

Expansion and Insertion



Copy old array, leave space for the new value



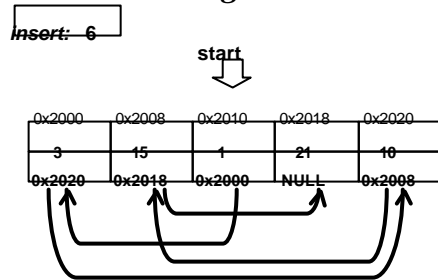
8

Disadvantages

- Lots of memory needs to be allocated.
- Lots of copying needs to be done.

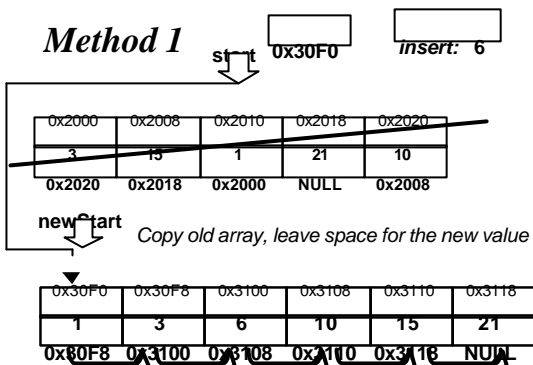
9

Linked List Implementation: Using Pointers



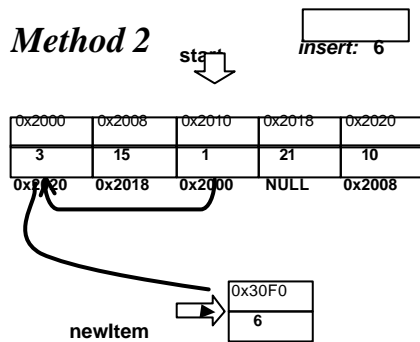
10

Method 1



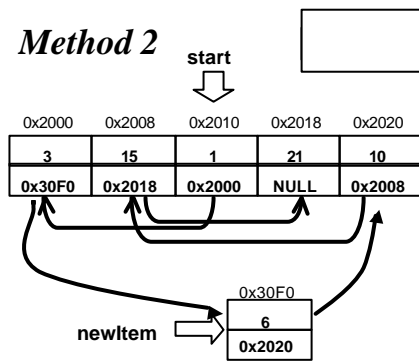
11

Method 2



12

Method 2



13

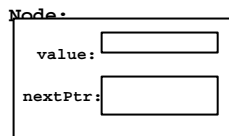
Advantages

- Only a little amount of memory needs to be allocated.
- Only a little amount of copying needs to be done.

14

```
struct NodeRec
{
    float value;
    struct NodeRec* nextPtr;
};

typedef struct NodeRec Node;
```



15

```
#ifndef NODEH
#define NODEH

struct NodeRec
{
    float value;
    struct NodeRec* nextPtr;
};

typedef struct NodeRec Node;

Node* makeNode(float item);

#endif
```

16

Make Node

- To make a new node for an item
 - take enough bytes from the heap
 - remember its address in memory
 - put the item in that location
 - set “next” link to NULL
 - return the node’s address

17

```
Node* makeNode(float item)
{
    Node* newNodePtr = (Node*)malloc(sizeof(Node));

    if (newNodePtr == NULL) {
        fprintf(stderr, "Out of memory");
        exit(1);
    }
    else {
        newNodePtr->value = item;
        newNodePtr->nextPtr = NULL;
    }

    return newNodePtr;
}
```

18

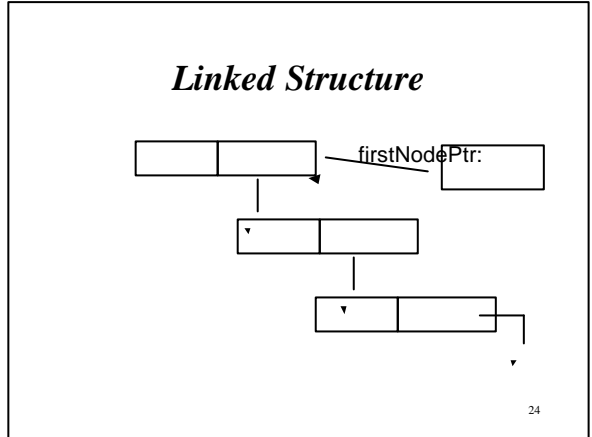
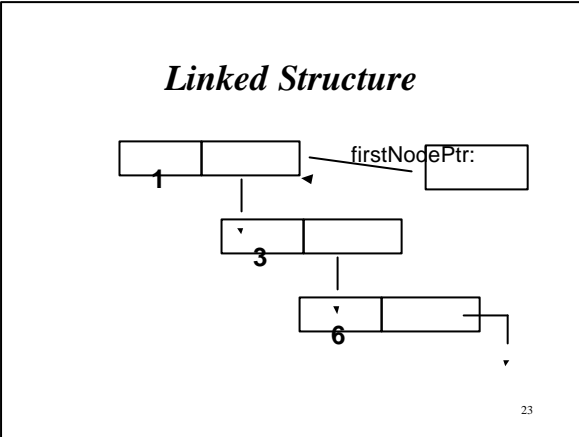
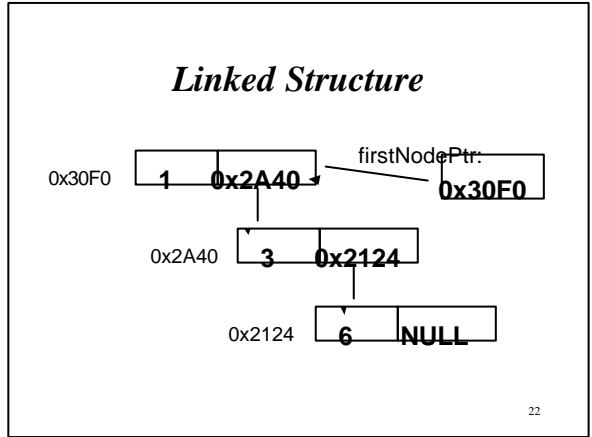
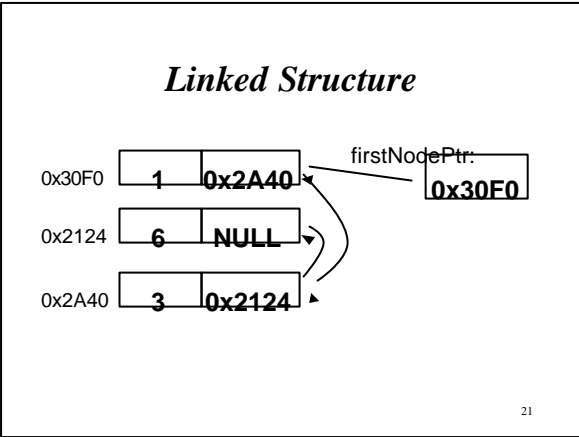
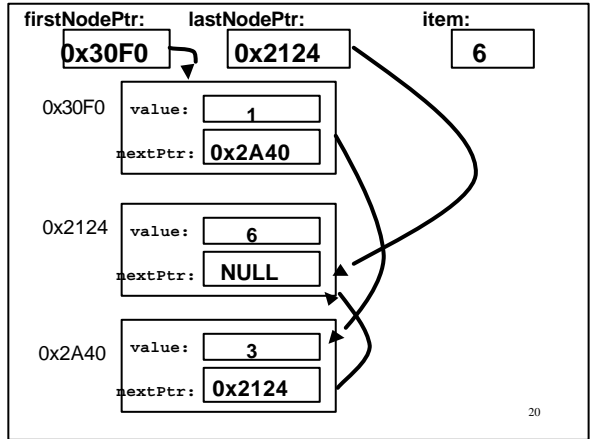
```

#include <stdio.h>
#include <stdlib.h>
#include "node.h"

int main()
{
    float item;
    Node* firstNodePtr = NULL;
    Node* lastNodePtr = NULL;

    while (scanf("%f", &item) != EOF){
        if (firstNodePtr == NULL){
            firstNodePtr = makeNode(item);
            lastNodePtr = firstNodePtr;
        }
        else {
            lastNodePtr->nextPtr = makeNode(item);
            lastNodePtr = lastNodePtr->nextPtr;
        }
    }
}

```



Revision

- Node
- How to make a new Node
- Linked Structures

25

Revision: Reading

- Kruse 4.5

Preparation

Next lecture: Linked Stacks and Queues

- Read 3.1.6 in Kruse et al.

26