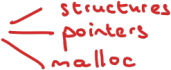


Today: Linked lists 

Computer Programming 143 – Lecture 27

Dynamic Data Structures I

Faculty of Engineering



Motivation and introduction

```
int array[100];
```

```
int *otherArray = malloc(numElements * sizeof(int));
```

12.1 Introduction

Linked lists

- Number of elements stored in a “line”
- Insertion and deletion can happen at any point

Stacks

- Used by compilers and operating systems
- Insertion and deletion only takes place at one point (FILO)

Queues

- Used as waiting queue/buffer
- Data inserted at one end (tail) and removed at the other (head) (FIFO)

Trees

- Used to implement very efficient search and sort algorithms
- Binary trees are the most basic type

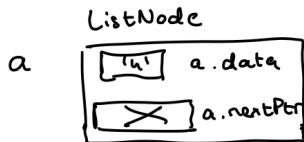
Linked lists

Node. A structure containing:

1. Data
2. Pointer to next element

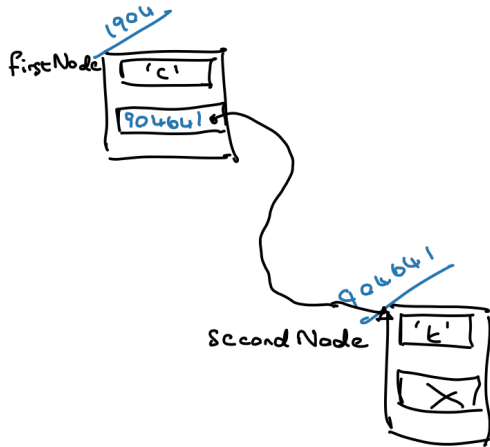
```
struct listNode {  
    char data;  
    struct listNode *nextPtr;  
};  
typedef struct listNode ListNode;
```

```
ListNode a = { 'h' , null};
```



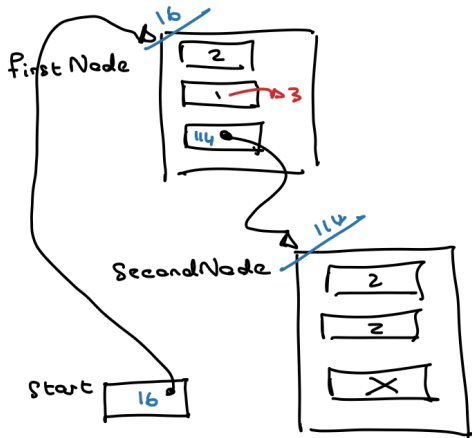
Example 1

```
struct listNode {  
    char data;  
    struct listNode *nextPtr;  
};  
typedef struct listNode ListNode;  
  
int main()  
{  
    ListNode firstNode = { 'c', NULL};  
    ListNode secondNode = { 'k', NULL};  
    firstNode.nextPtr = &secondNode;  
  
    return 0;  
}
```



Example 2

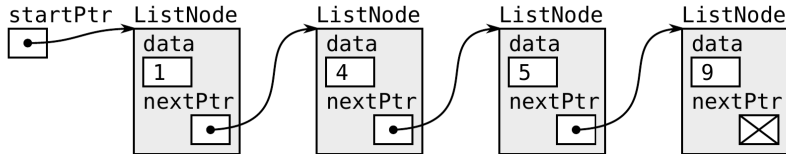
```
struct listNode {  
    int row;  
    int col;  
    struct listNode *nextPtr;  
};  
typedef struct listNode ListNode;  
  
int main()  
{  
    ListNode *start;  
    ListNode firstNode = {2, 1, NULL};  
    start = &firstNode;  
    ListNode secondNode = {2, 2, NULL};  
  
    firstNode.nextPtr = &secondNode;  
    (*start).col = 3;    // start -> col = 3;  
    return 0;  
}
```



12.4 Linked lists

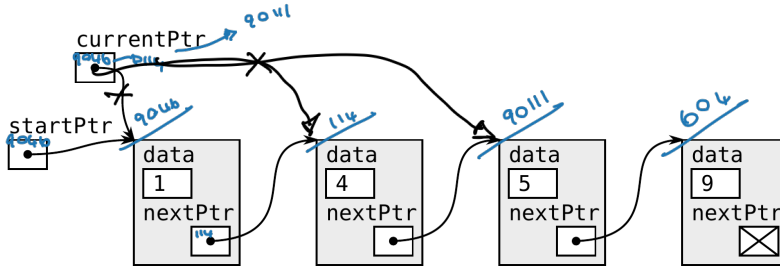
Linked lists

- A linked list is a linear collection of self-referential structures
- Each element that forms part of the list is called a node
- Nodes are connected to one another via pointers called links
- Linked lists are accessed via a pointer to the first element
- Subsequent nodes are accessed via the link pointer member of the current node



12.4 Searching Linked Lists

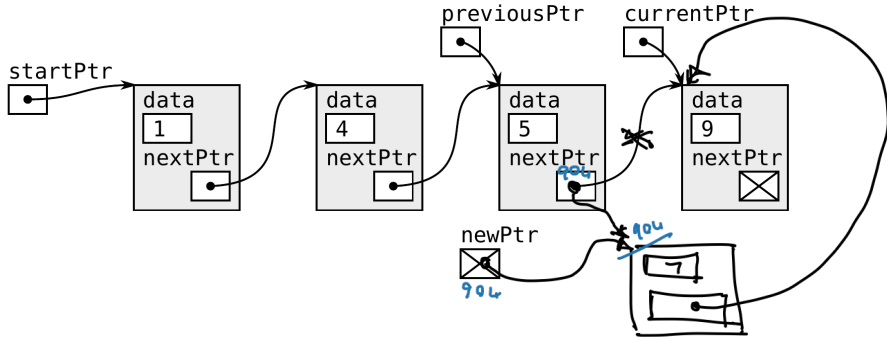
```
value = 5;  
currentPtr = startPtr;  
while ( currentPtr != NULL && value != currentPtr->data ) {  
    currentPtr = currentPtr->nextPtr;  
}
```



12.4 Linked List Insertion

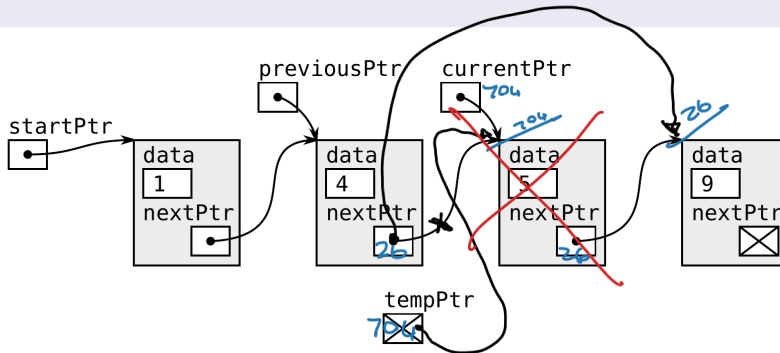
- ✓ `newPtr = malloc( sizeof( ListNode ) );`
- ✓ `newPtr->data = value 7;`
- ✓ `newPtr->nextPtr = NULL;`
- ✓ `previousPtr->nextPtr = newPtr;`
`newPtr->nextPtr = currentPtr;`

*ListNode *currentPtr;*



12.4 Linked List Deletion

```
✓ tempPtr = currentPtr;  
✓ previousPtr->nextPtr = currentPtr->nextPtr;  
  free( tempPtr );
```



12.4 Linked lists

Linked lists vs. arrays

- The information in linked lists can be stored in arrays, but linked lists provide many advantages:
 - Linked lists are applicable when the number of elements in the list at any given time is not known in advance
 - Linked lists are dynamic and its size can change with the requirements; arrays are static
 - Arrays can get full; linked lists can almost always grow to accommodate more data

Homework

Figure 12.1 in Deitel & Deitel