

Today: Stacks and queues

Computer Programming 143 – Lecture 28

Dynamic Data Structures II

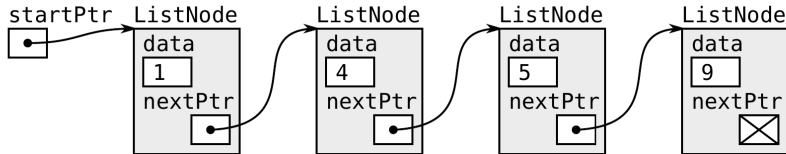
Faculty of Engineering



12.4 Linked lists

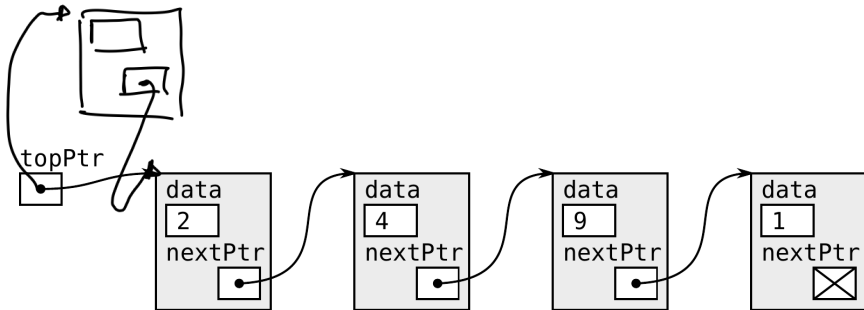
Linked lists

- 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



Stacks

- Can only add to the top (at start)
- Can only remove from top
- Last in first out (LIFO)

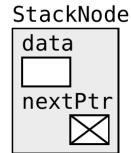


12.5 Stacks

Stacks

- Constrained version of a linked list
 - Is a row/line of self-referential structures
- Elements may only be inserted or deleted from the top (front) of a stack
 - A stack is therefore a **last-in-first-out structure (LIFO)**

```
struct stackNode {  
    int data;  
    struct stackNode *nextPtr;  
};  
typedef struct stackNode StackNode;
```



Use of stacks

- Function calls – store of return address and automatic local variables
- Used by compilers when evaluating expressions

12.5 Insertion in Stacks ("Push")

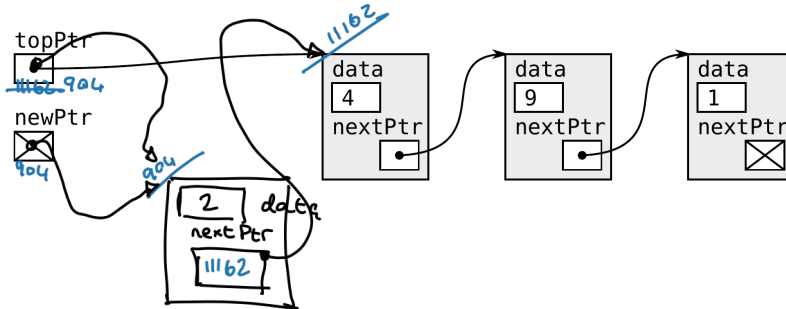
1 `StackNode *newPtr = NULL;`

2 `newPtr = malloc( sizeof( StackNode ) );`

3 `newPtr->data = value;`

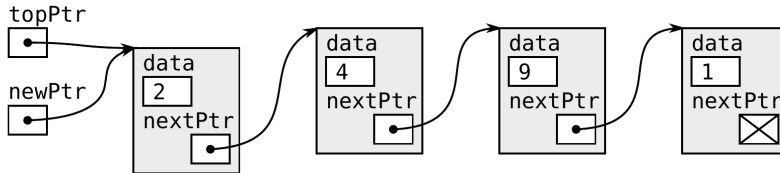
4 `newPtr->nextPtr = topPtr;`

5 `topPtr = newPtr;`



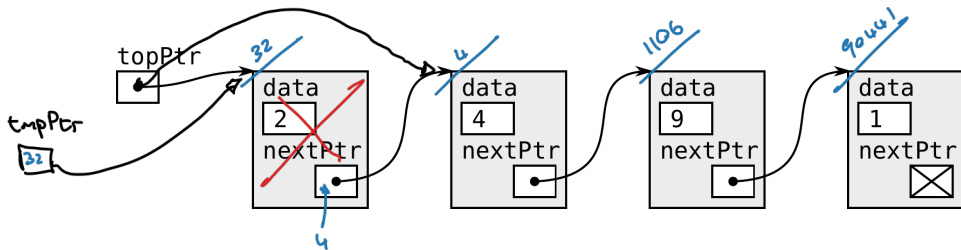
12.5 Insertion in Stacks ("Push")

```
newPtr = malloc( sizeof( StackNode ) );  
newPtr->data = value;  
newPtr->nextPtr = topPtr;  
topPtr = newPtr;
```



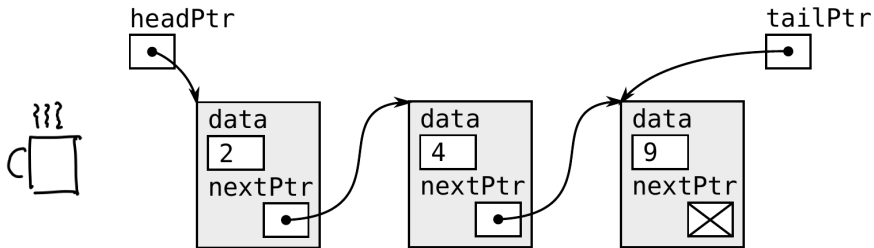
Deletion (popping) in stacks

```
int popValue;  
StackNode * tmpPtr;  
tmpPtr = topPtr;  
popValue = (* tmpPtr).data;    // tmpPtr -> data;  
topPtr = topPtr -> nextPtr;  
free (tmpPtr);
```



Queues

- First in, first out (FIFO)
- I.e., add at back take from front
- E.g. call centre



12.6 Queues

Queues

- Constrained version of a linked list
 - Is a row/line of self-referential structures
- Elements may only be deleted from the front (head) of a stack and inserted at the back (tail)
 - A queue is therefore a **first-in-first-out structure (FIFO)**

```
struct queueNode {  
    int data;  
    struct queueNode *nextPtr;  
};  
typedef struct queueNode QueueNode;
```

QueueNode

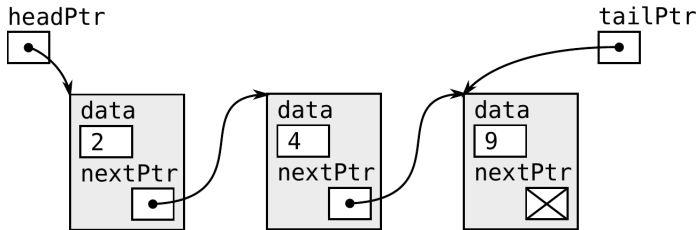


Use of Queues

- Supporting print spooling
- Routing of data packets in computer systems
- Buffers

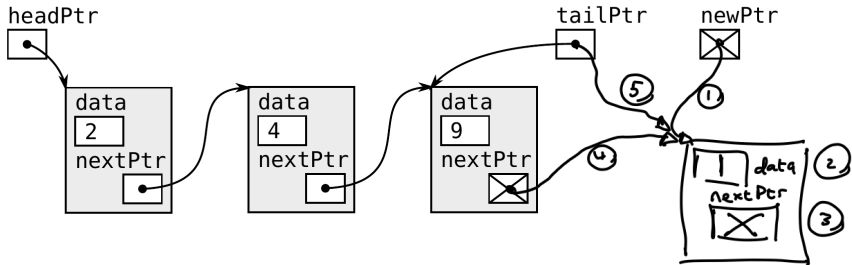
Insertion in queues

Add data 1:



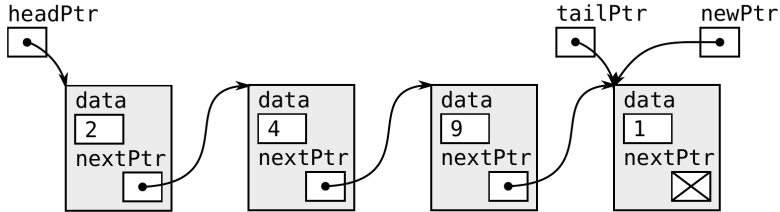
12.6 Insertion in Queues (“Enqueue”)

```
1 newPtr = malloc( sizeof( QueueNode ) );  
2 newPtr->data = value;  
3 newPtr->nextPtr = NULL;  
4 tailPtr->nextPtr = newPtr;  
5 tailPtr = newPtr;
```



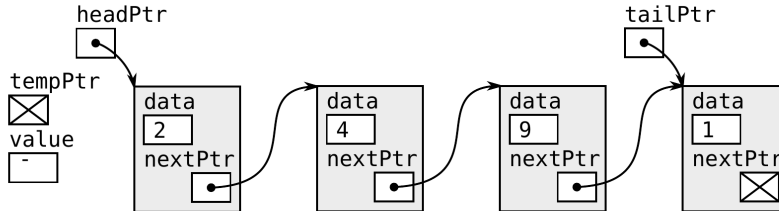
12.6 Insertion in Queues (“Enqueue”)

```
newPtr = malloc( sizeof( QueueNode ) );  
newPtr->data = value;  
newPtr->nextPtr = NULL;  
tailPtr->nextPtr = newPtr;  
tailPtr = newPtr;
```



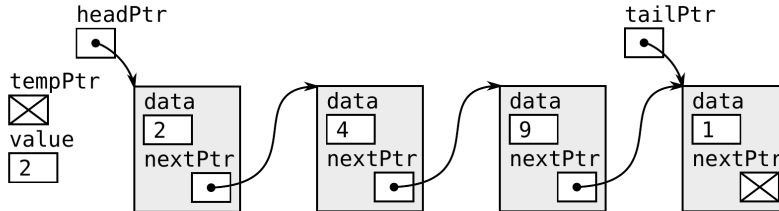
12.6 Deletion in Queues ("Dequeue")

```
value = headPtr->data;  
tempPtr = headPtr;  
headPtr = headPtr->nextPtr;  
free( tempPtr );
```



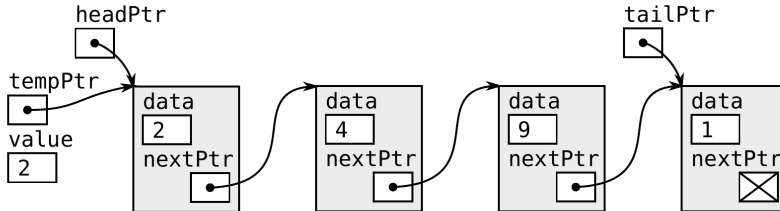
12.6 Deletion in Queues ("Deque")

```
value = headPtr->data;  
tempPtr = headPtr;  
headPtr = headPtr->nextPtr;  
free( tempPtr );
```



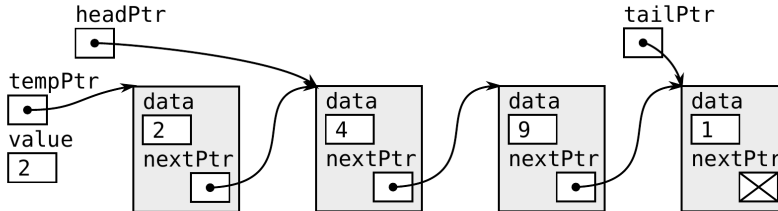
12.6 Deletion in Queues ("Deque")

```
value = headPtr->data;  
tempPtr = headPtr;  
headPtr = headPtr->nextPtr;  
free( tempPtr );
```



12.6 Deletion in Queues ("Deque")

```
value = headPtr->data;  
tempPtr = headPtr;  
headPtr = headPtr->nextPtr;  
free( tempPtr );
```



12.6 Deletion in Queues ("Deque")

```
value = headPtr->data;  
tempPtr = headPtr;  
headPtr = headPtr->nextPtr;  
free( tempPtr );
```

