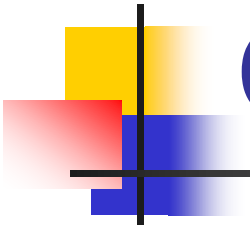
CS 2213
Advanced Programming
Ch 10 – Linear Structures

Turgay Korkmaz

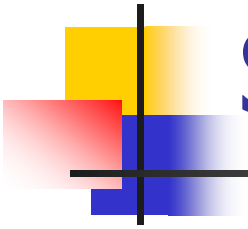
Office: SB 4.01.13
Phone: (210) 458-7346
Fax: (210) 458-4437
e-mail: korkmaz@cs.utsa.edu
web: www.cs.utsa.edu/~korkmaz

Thanks to Eric S. Roberts, the author of our textbook, for providing some slides/figures/programs.



Objectives

- To recognize that stacks can be implemented using linked lists as well as arrays.
- To learn how to implement queues using both arrays and linked lists as the underlying representation.
- To understand and learn about simulations (self-study)



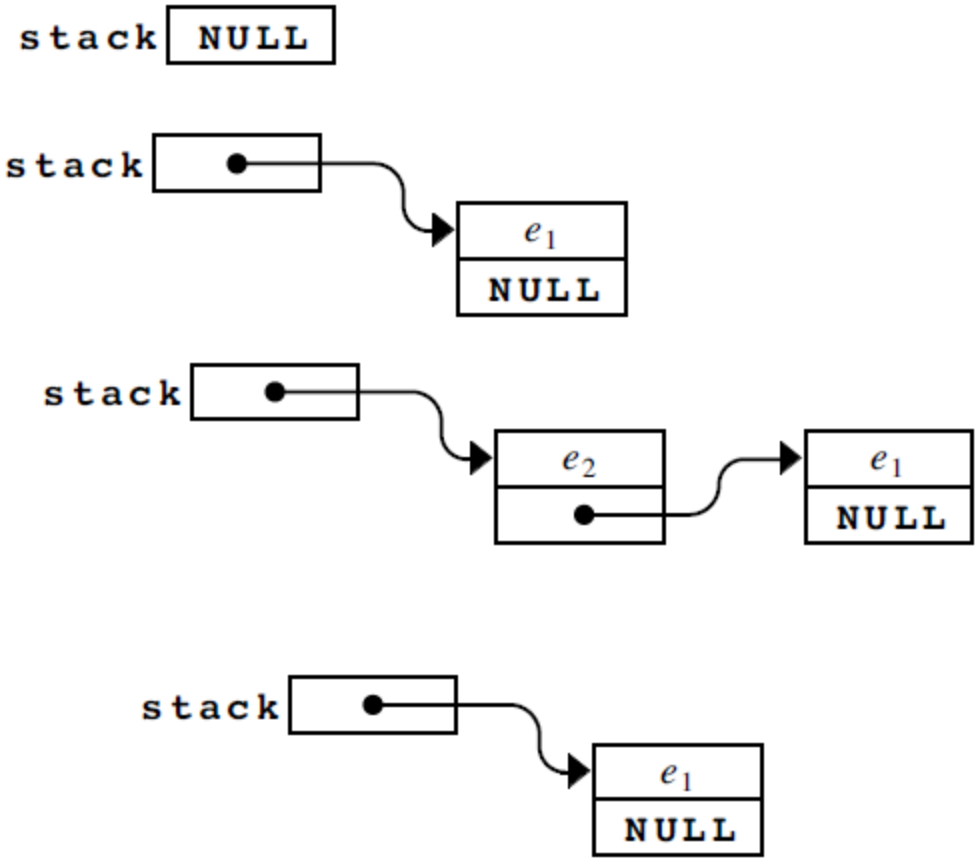
Stack Using Linked List

- Revisit **stack**
  - A storage for a collection of data values (**elements**)
  - Defining behavior of stack is "Last in, First out" (LIFO)
  - Adding a new element to a stack is called **push**
  - Removing the most recent item from a stack is called **pop**
- Re-implement stack.c using **linked list** while keeping the same stack.h interface

Recall stack.h Interface

```
#ifndef _stack_h                /* for comments see stack.h */
#define _stack_h

#include "genlib.h"

typedef double stackElementT;    /* "double" can be replaced by other types */

typedef struct stackCDT *stackADT;

stackADT NewStack(void);
void FreeStack(stackADT stack);
void Push(stackADT stack, stackElementT element);
stackElementT Pop(stackADT stack);
bool StackIsEmpty(stackADT stack);
bool StackIsFull(stackADT stack);
int StackDepth(stackADT stack);
stackElementT GetStackElement(stackADT stack, int index);

#endif
```

The simple strategy to re-implement stack using Linked List



NewStack(){
  return NULL;
}

Push(stack, e1)

Push(stack, e2)

```
cp=New(cellT *);
cp->next = stack;
stack=cp;
```

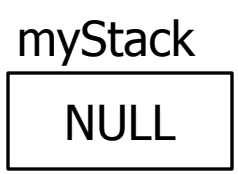
Pop(stack)

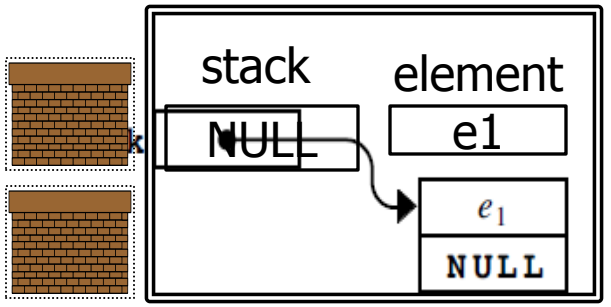
Implementation of stack.c

First provide a concrete representation stackADT

- The simple strategy fails. WHY?

```
typedef struct stackCDT *stackADT; // stack.h
stackADT myStack;
myStack=NewStack();
Push(myStack, e1);
```





- Actually we can make this work if we pass pointer to pointer **, requiring changes in stack.h
- WHAT CAN WE DO TO KEEP stack.h AS IS?

Implementation of stack.c

First provide a concrete representation stackADT

- stack.h will keep the standard abstract def

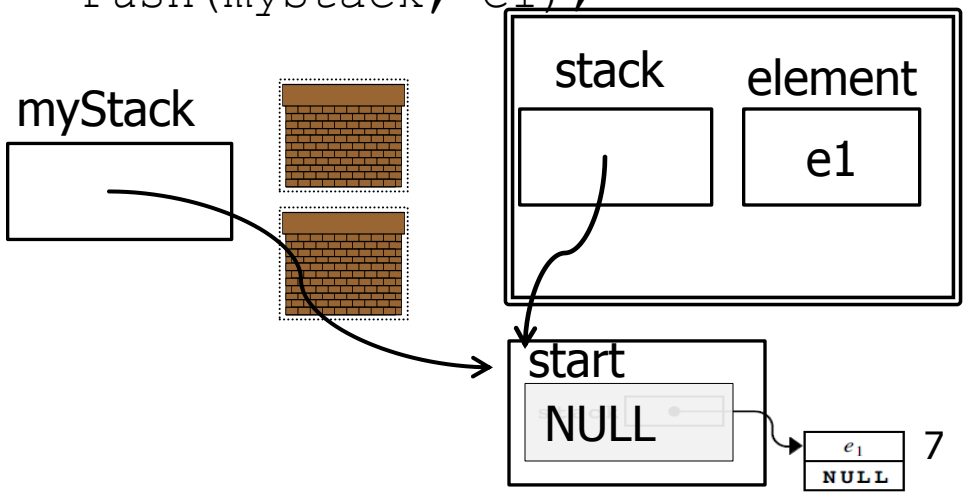
```
typedef struct stackCDT *stackADT;
```

- stack.c then defines `stackCDT` as

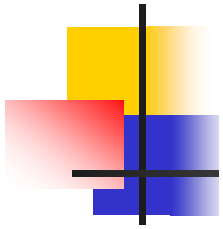
```
struct stackCDT {
   cellT *start;
};
stackADT NewStack()
{
  stackADT stack;
  stack = New(stackADT);
  stack->start = NULL;
  return(stack);
}
```

Then when the client calls

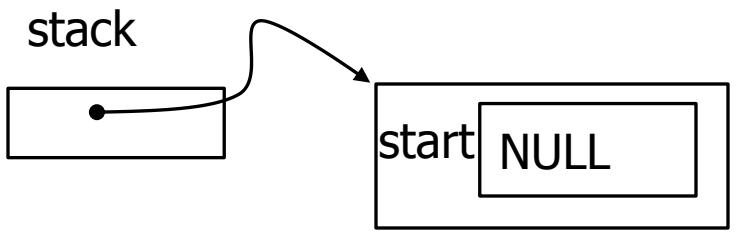
```
myStack=NewStack();
Push(myStack, e1);
```

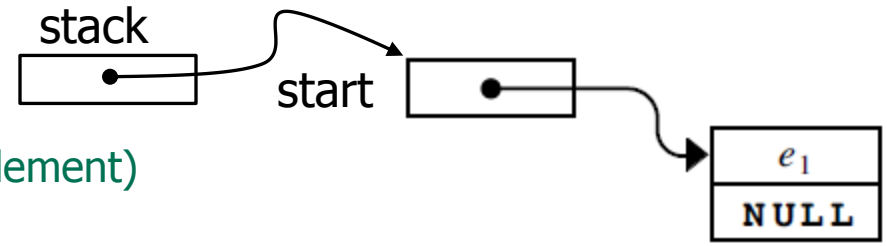


Reimplementing stack.c

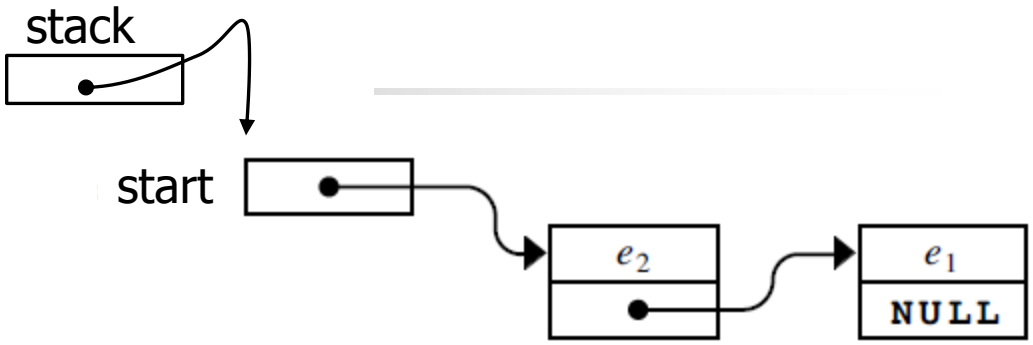


```
#include <stdio.h>
#include "genlib.h"
#include "stack.h"

typedef struct cellT {
   stackElementT element;
   struct cellT *link;
} cellT;

struct stackCDT {
   cellT *start;
};

stackADT NewStack(void)
{
   stackADT stack;

   stack = New(stackADT);
   stack->start = NULL;
   return (stack);
}
```

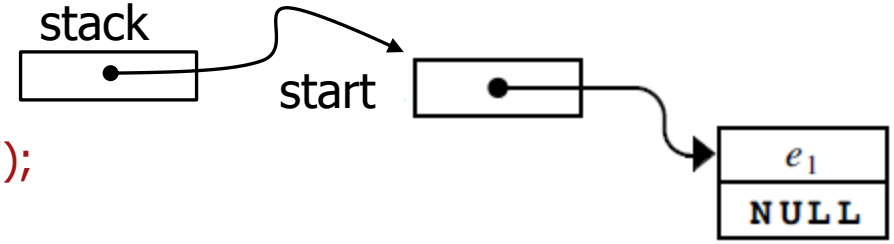





```
void Push(stackADT stack, stackElementT element)
{
    cellT *cp;

    cp = New(cellT *);
    cp->element = element;
    cp->link = stack->start;
    stack->start = cp;
}
```

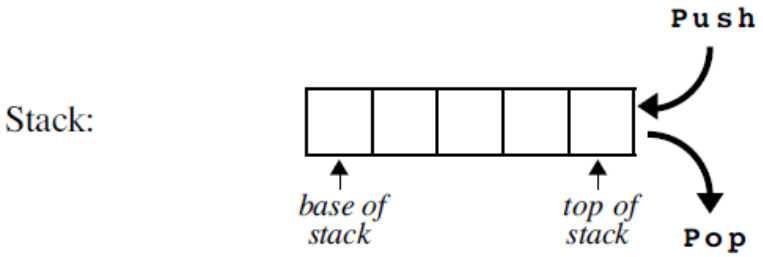


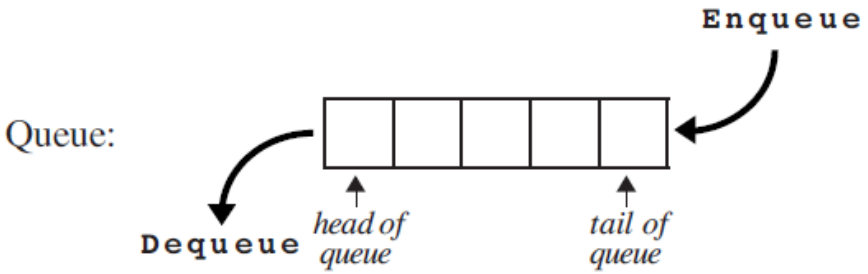

```
stackElementT Pop(stackADT stack)
{
    stackElementT result;
    cellT *cp;

    if (StackIsEmpty(stack))
            Error("Pop of an empty stack");
    cp = stack->start;
    result = cp->element;
    stack->start = cp->link;
    FreeBlock(cp);
    return (result);
}
```



```
void FreeStack(stackADT stack)
{
    cellT *cp, *next;

    cp = stack->start;
    while (cp != NULL) {
        next = cp->link;
        FreeBlock(cp);
        cp = next;
    }
    FreeBlock(stack);
}

bool StackIsEmpty(stackADT stack)
{
    return (stack->start == NULL);
}

bool StackIsFull(stackADT stack)
{
    return (FALSE);
}
```

```
int StackDepth(stackADT stack)
{
    int n;
    cellT *cp;

    n = 0;
    for (cp = stack->start; cp != NULL; cp = cp->link) n++;
    return (n);
}

stackElementT GetStackElement(stackADT stack, int depth)
{
    int i;
    cellT *cp;

    if (depth < 0 || depth >= StackDepth(stack)) {
        Error("Non-existent stack element");
    }
    cp = stack->start;
    for (i = 0; i < depth; i++) cp = cp->link;
    return (cp->element);
}
```





QUEUES

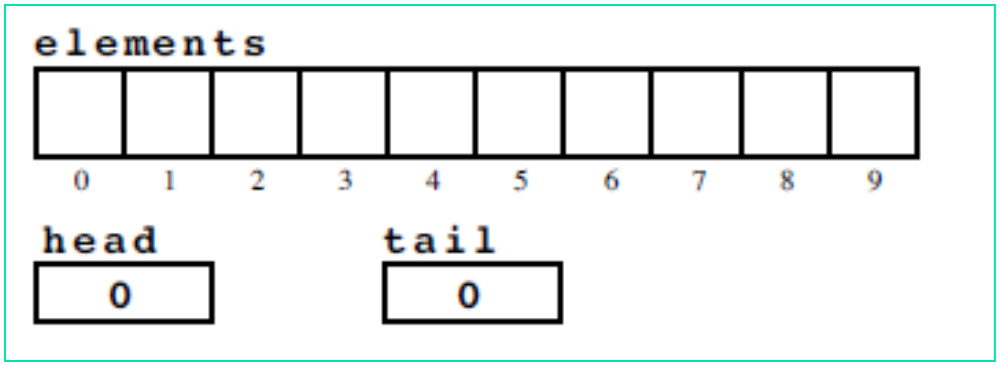
- Very similar to stacks
- The only difference between them is in the order in which elements are processed.
  - A stack uses a last-in/first-out (LIFO) discipline
  - A queue adopts a first-in/first-out (FIFO) model that more closely resembles a waiting line.

queue.h interface

```
#ifndef _queue_h
#define _queue_h
#include "genlib.h"

typedef void *queueElementT;   /* "void *" can be replaced by other types */

typedef struct queueCDT *queueADT;

queueADT NewQueue(void);
void FreeQueue(queueADT queue);

void Enqueue(queueADT queue, queueElementT element);
queueElementT Dequeue(queueADT queue);

bool QueueIsEmpty(queueADT queue);
bool QueueIsFull(queueADT queue);
int QueueLength(queueADT queue);

queueElementT GetQueueElement(queueADT queue, int index);

#endif
```

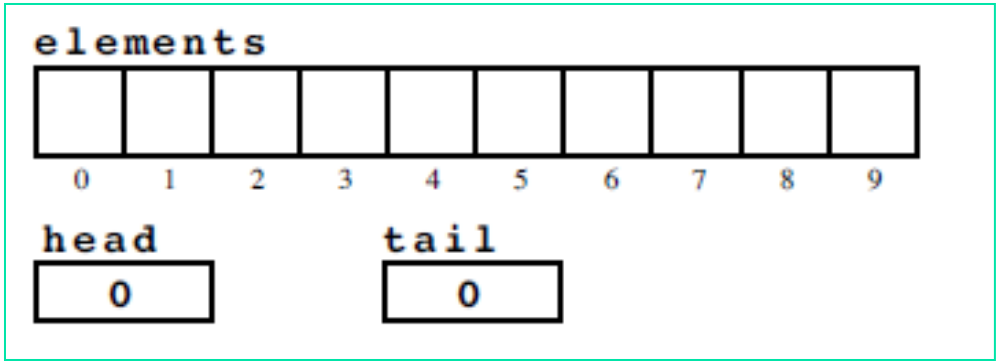
Array Implementation of Queue ADT

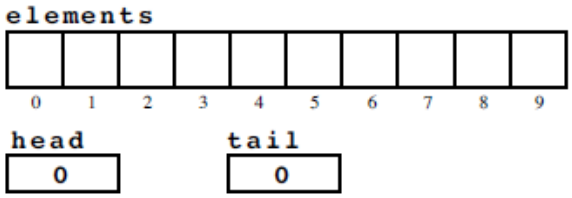
What fields do we need in queueCDT?

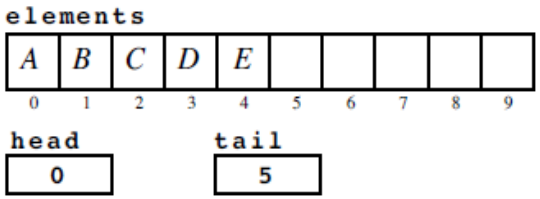


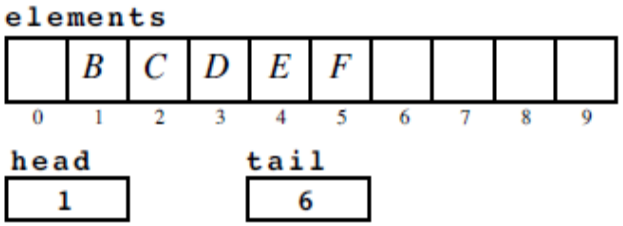
qarray.c

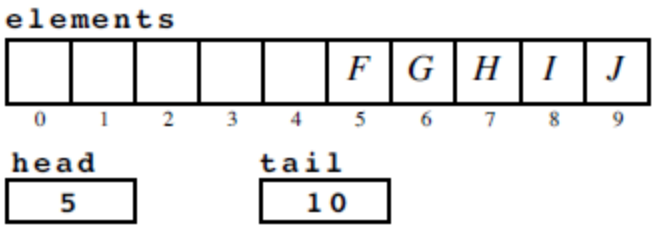
```
#include <stdio.h>
#include "genlib.h"
#include "queue.h"

#define MaxQueueSize   100
#define QueueArraySize (MaxQueueSize + 1)

struct queueCDT {
   queueElementT elements[QueueArraySize];
   int head;
   int tail;
};

queueADT NewQueue(void)
{
   queueADT queue;

   queue = New(queueADT);
   queue->head = queue->tail = 0;
   return (queue);
}
```

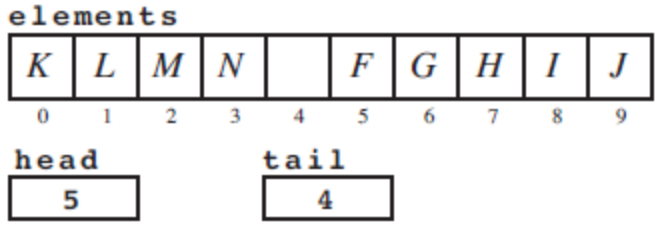


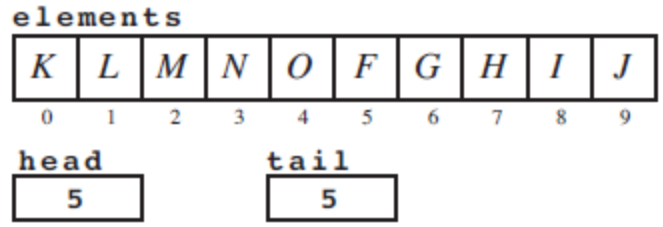












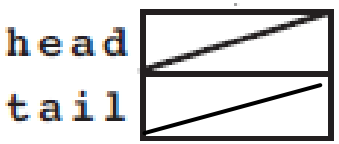

```
void Enqueue(queueADT queue, queueElementT element)
{
  if (QueueIsFull(queue))
        Error("Enqueue: queue is full");
  queue->elements[queue->tail] = element;
  queue->tail = (queue->tail + 1) % QueueArraySize;
}


queueElementT Dequeue(queueADT queue)
{
  queueElementT result;

  if (QueueIsEmpty(queue))
        Error("Dequeue: queue is empty");
  result = queue->elements[queue->head];
  queue->head = (queue->head+1)% QueueArraySize;
  return (result);
}
```

End of qarray.c

```
bool QueueIsEmpty(queueADT queue)
{
   return (queue->head == queue->tail);
}
bool QueueIsFull(queueADT queue)
{
   return ((queue->tail + 1) % QueueArraySize == queue->head);
}
void FreeQueue(queueADT queue)
{
   FreeBlock(queue);
}
int QueueLength(queueADT queue)
{
   return ((QueueArraySize + queue->tail - queue->head) % QueueArraySize);
}
queueElementT GetQueueElement(queueADT queue, int index)
{
   if (index < 0 || index >= QueueLength(queue)) {
      Error("Queue element is out of range");
   }
   return (queue->elements[(queue->head + index) % QueueArraySize]);
}
```

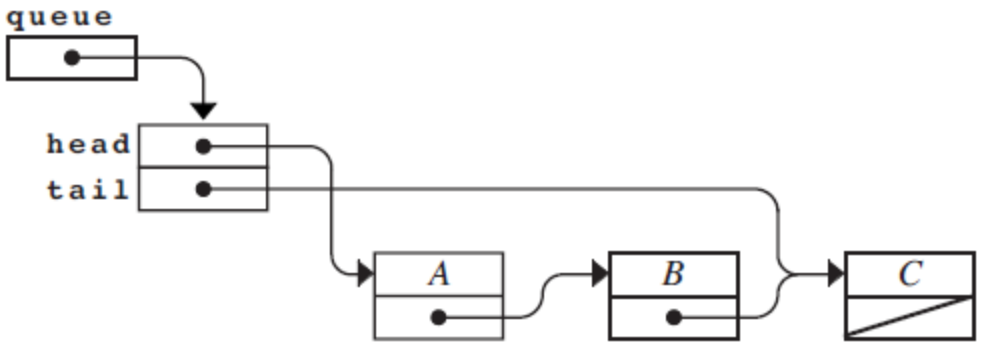
Linked List Implementation of Queue ADT

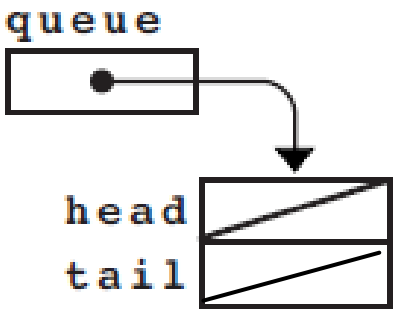
What fields do we need in queueCDT?

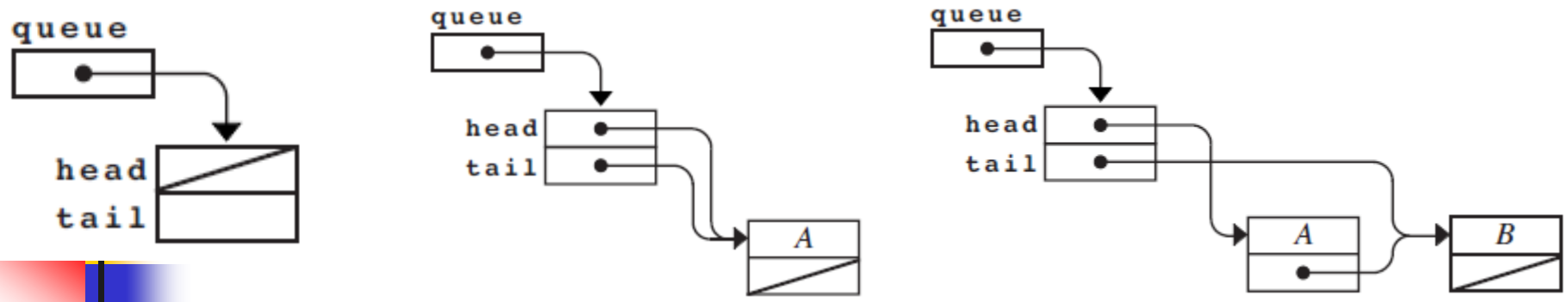


qlist.c

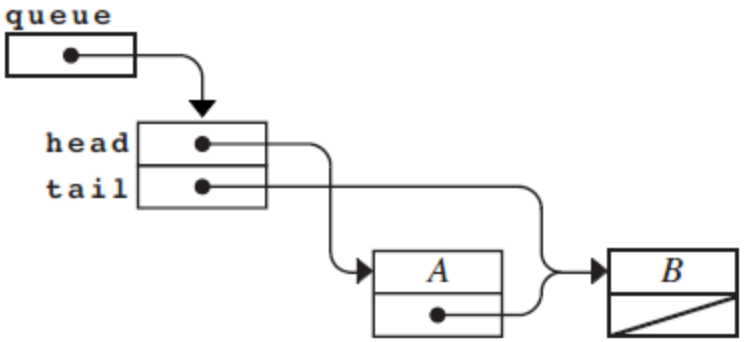
```
#include <stdio.h>
#include "genlib.h"
#include "queue.h"

typedef struct cellT {
    queueElementT value;
    struct cellT *link;
} cellT;

struct queueCDT {
    cellT *head;
    cellT *tail;
};

queueADT NewQueue(void)
{
    queueADT queue;

    queue = New(queueADT);
    queue->head =  queue->tail =  NULL;
    return (queue);
}
```

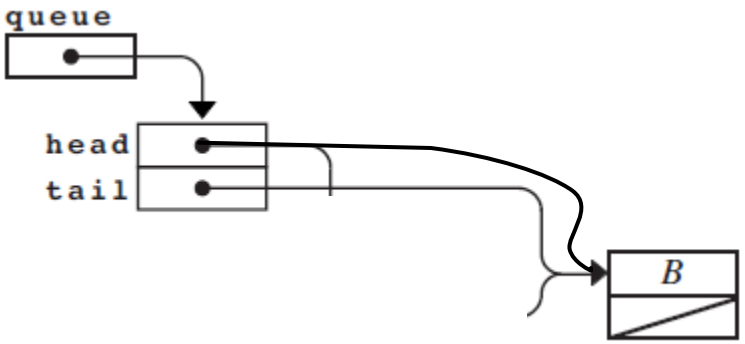







```
void Enqueue(queueADT queue, queueElementT element)
{
    cellT *cp;

    cp = New(cellT *);
    cp->value = element;
    cp->link = NULL;
    if (queue->head == NULL) {
        queue->head = cp;
    } else {
        queue->tail->link = cp;
    }
    queue->tail = cp;
}
```



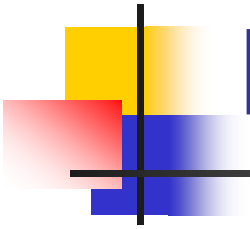



```
queueElementT Dequeue(queueADT queue)
{
    queueElementT result;
    cellT *cp;

    cp = queue->head;
    if (cp == NULL) {
        Error("Dequeue: queue is empty");
    }
    result = cp->value;
    queue->head = cp->link;
    FreeBlock(cp);
    return (result);
}
```

```
bool QueueIsEmpty(queueADT queue)
{
   return (queue->head == NULL);
}

bool QueueIsFull(queueADT queue)
{
   return (FALSE);
}

void FreeQueue(queueADT queue)
{
   cellT *cp, *next;

   cp = queue->head;
   while (cp != NULL) {
      next = cp->link;
      FreeBlock(cp);
      cp = next;
   }
   FreeBlock(queue);
}
```

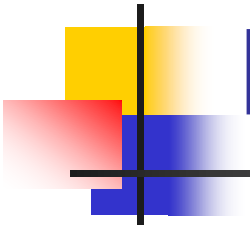
End of qlist.c

```
int QueueLength(queueADT queue)
{
    int n;
    cellT *cp;

    n = 0;
    for (cp = queue->head; cp != NULL; cp = cp->link) n++;
    return (n);
}
queueElementT GetQueueElement(queueADT queue, int index)
{
    int i;
    cellT *cp;

    if (index < 0 || index >= QueueLength(queue)) {
        Error("Queue element is out of range");
    }
    cp = queue->head;
    for (i = 0; i < index; i++) cp = cp->link;
    return (cp->value);
}
```



Exercise: Reverse a queue

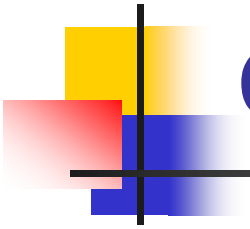
- Can you write a function that will reverse the elements of a queue for array and linked list representations
  - For example if queue has **A B C D,** your function should make the queue **D C B A**
- What will be the complexity?



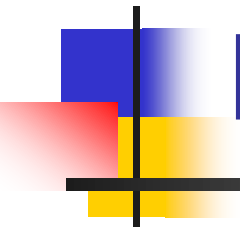
Exercise: Priority queue

- Can you write a function that will insert the elements in a queue in a sorted manner under array and linked list representations
  - For example if queue has **4 8 10 41** and you want to insert **9** your function should insert it after 8 before 10 such that the queue will have **4 8 9 10 41**
- What will be the complexity?

Exercise: merge two sorted queue/list



- Can you write a function that will merge the elements in two sorted queue in a sorted manner under linked list representations
  - For example if queue1 has **4 8 10 41** and queue2 has **5 9 45**, your function should merge them as queue3, which will have **4 5 8 9 10 41 45**
- What will be the complexity?



Application of Queues: Discrete Time Simulations

Self-Study

Section 10.3 form the textbook

Exercise: Queue Test Program

Write a simple simulation of a queue application in which the user can either enter a name that is entered into a queue or the command `serve`, which dequeues and displays the first name. Extend the program to support `quit` and `list` commands.