Chap 4. Linked Lists (2)

Contents

Chapter 1. Basic Concepts

Chapter 2. Arrays And Structures

Chapter 3. Stacks And Queues

Chapter 4. Linked Lists

Chapter 5. Trees (Midterm exam)

Chapter 6. Graphs

Chapter 7. Sorting

Chapter 8. Hashing (Final exam)

Contents

4.1 Singly Linked Lists and Chains

4.2 Representing Chains in C

4.3 Linked Stacks and Queues

4.4 Polynomials

4.5 Additional List Operations

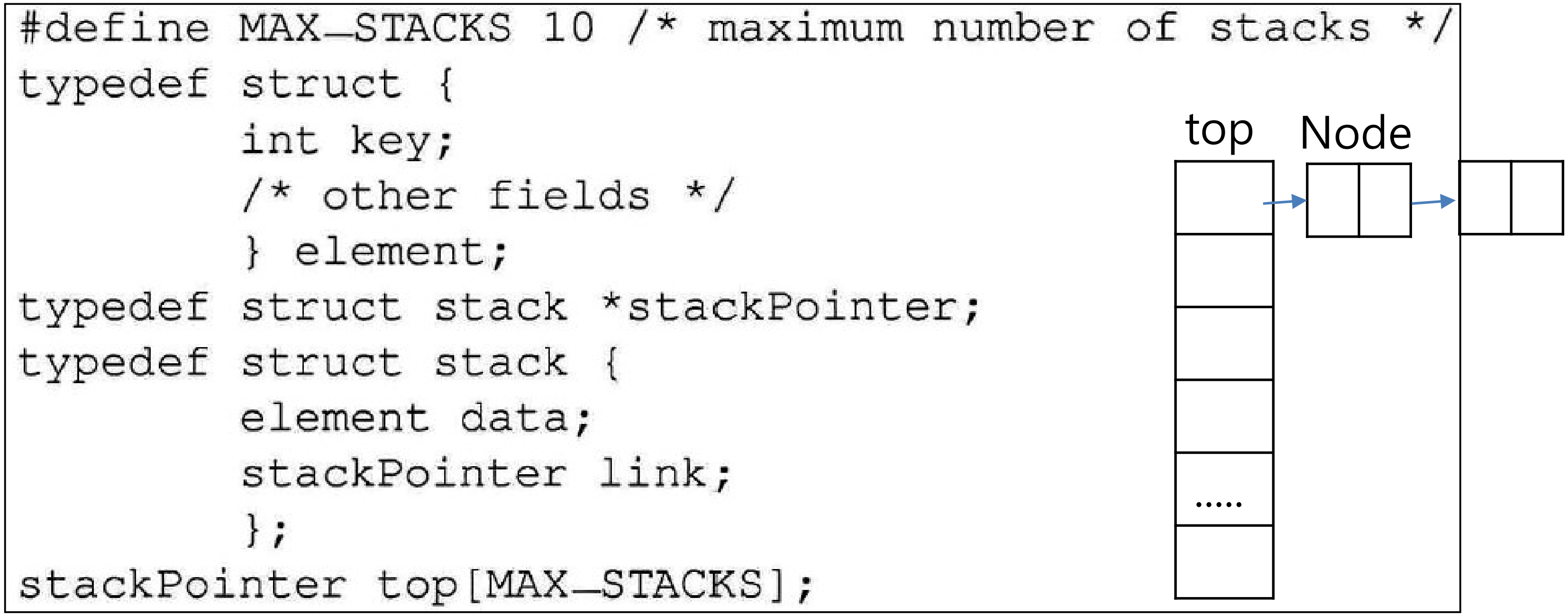
4.6 Equivalence Classes

4.7 Sparse Matrices

4.8 Doubly Linked Lists

4.3 Linked Stacks And Queues

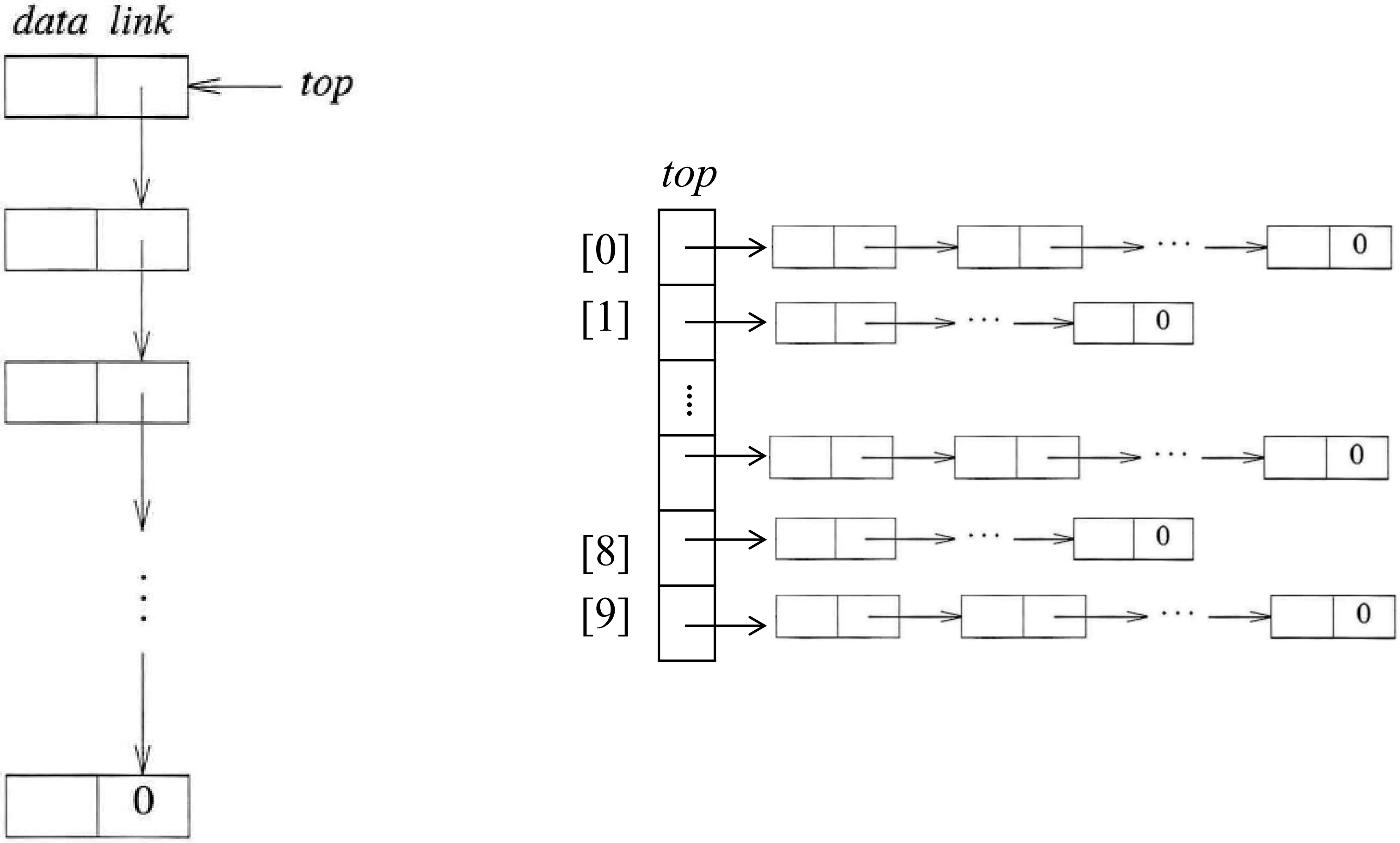
- Representing ***n≤MAX_STACKS* stacks** simultaneously



```
#define MAX_STACKS 10 /* maximum number of stacks */
typedef struct {
        int key;
        /* other fields */
        } element;
typedef struct stack *stackPointer;
typedef struct stack {
        element data;
        stackPointer link;
        };
stackPointer top[MAX_STACKS];
```

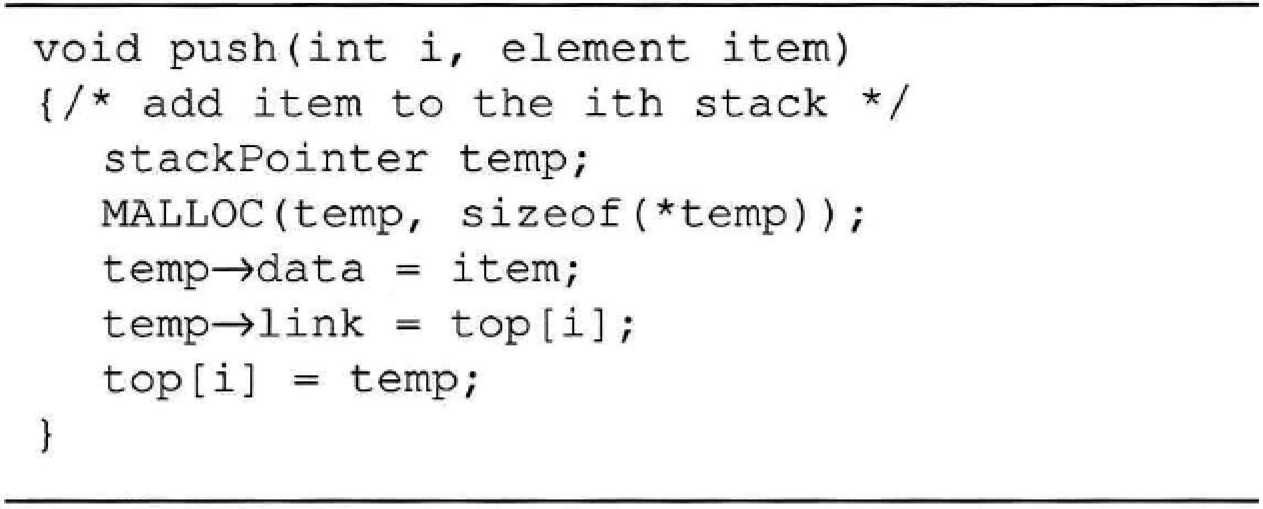
$top[i] = NULL,\ 0 \le i < MAX\_STACKS$ Initial conditions for the stacks

[redacted] iff the *i*th stack is empty Boundary condition for the *i*th stack

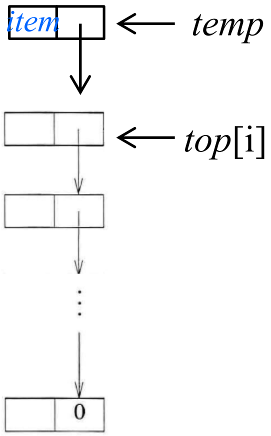


(a) Linked stack

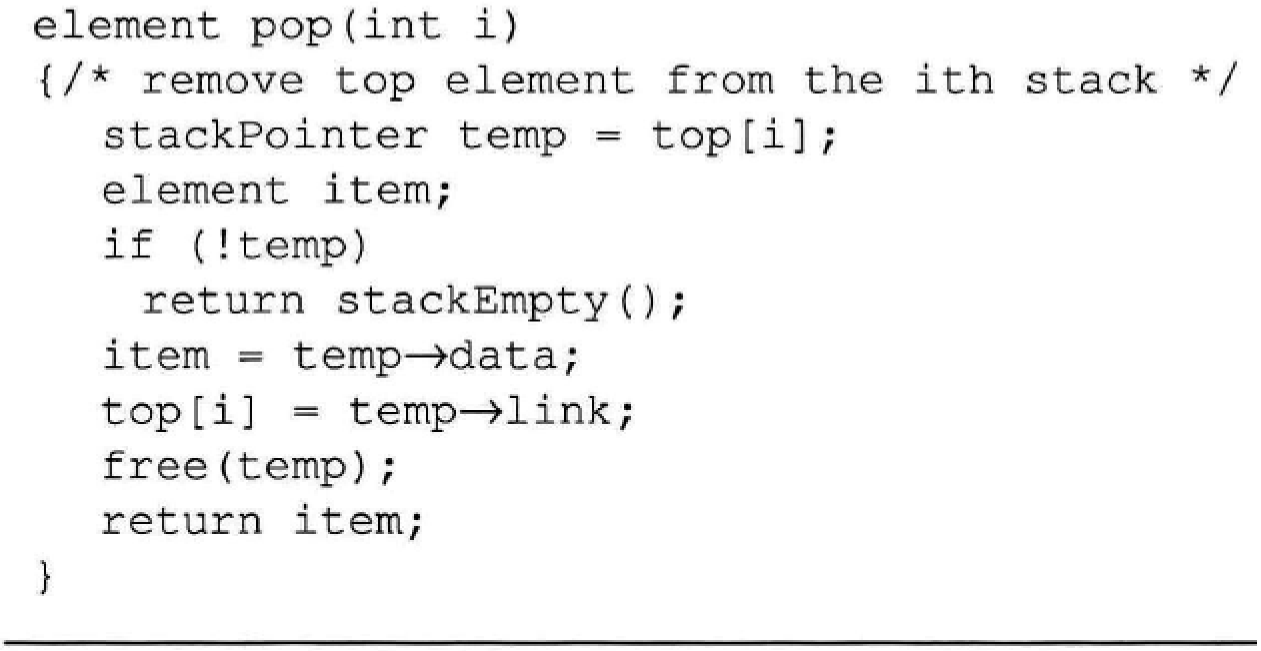
Figure 4.11: Linked stack and queue (1/2)



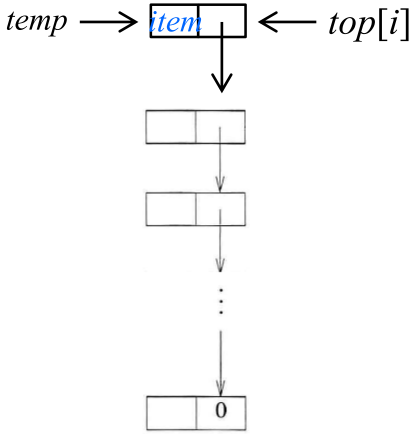
```
void push(int i, element item)
{/* add item to the ith stack */
   stackPointer temp;
   MALLOC(temp, sizeof(*temp));
   temp→data = item;
   temp→link = top[i];
   top[i] = temp;
}
```



Program 4.5: Add to a linked stack *push(i, item)*

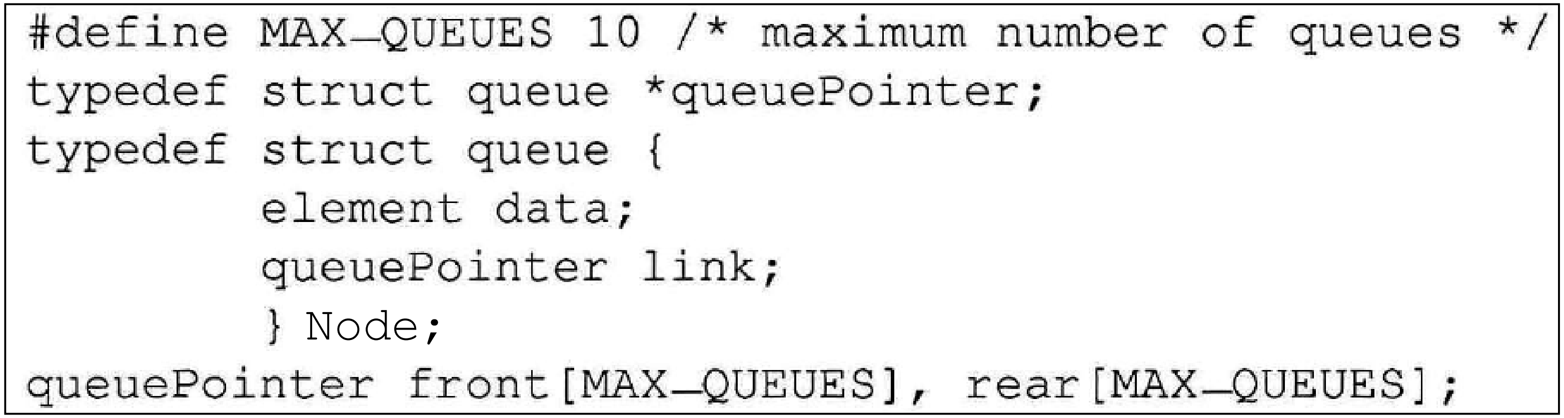


```
element pop(int i)
{/* remove top element from the ith stack */
   stackPointer temp = top[i];
   element item;
   if (!temp)
     return stackEmpty();
   item = temp→data;
   top[i] = temp→link;
   free(temp);
   return item;
}
```



Program 4.6: Delete from a linked stack *item = pop(i)*

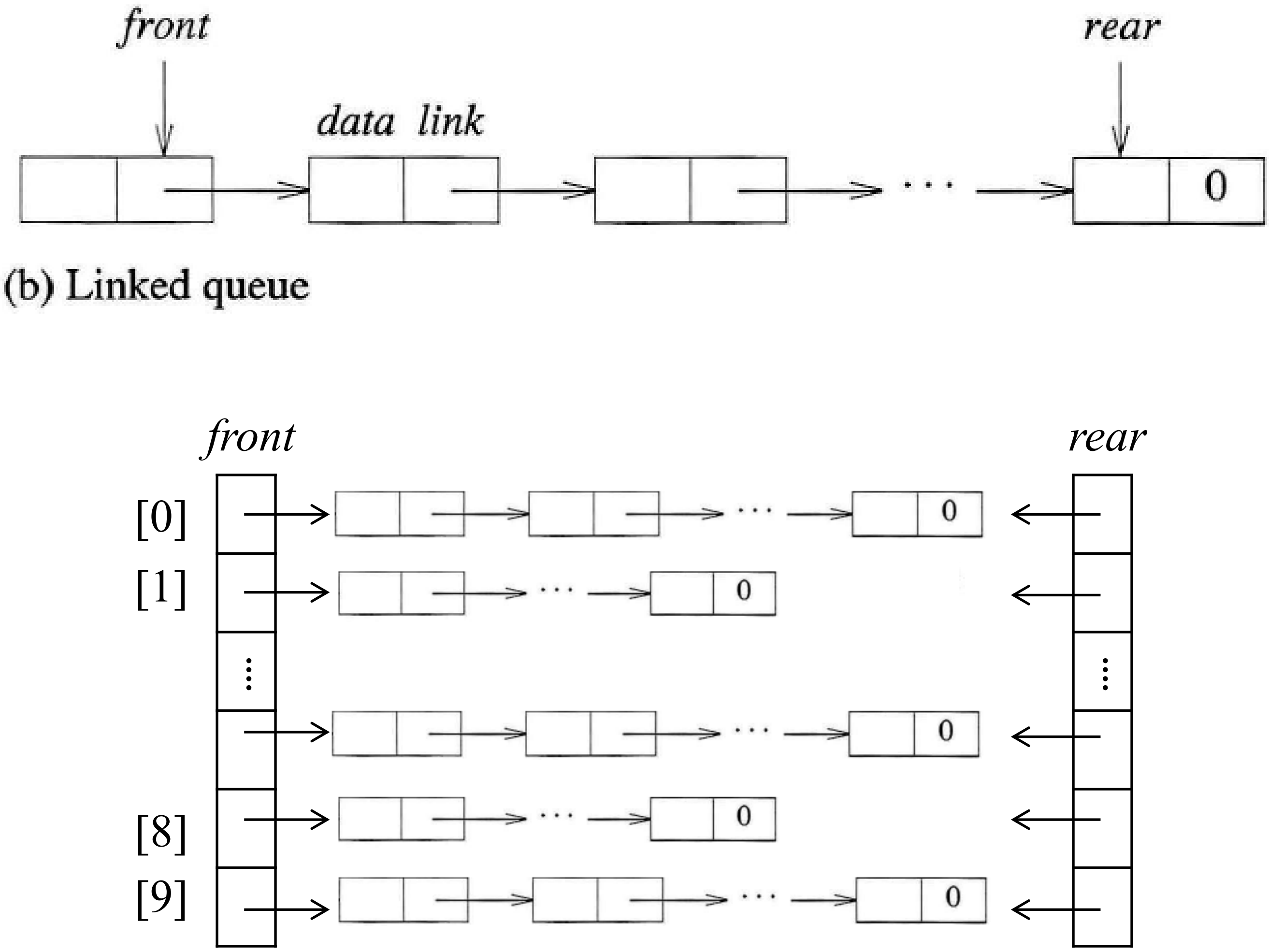
- Representing ***n≤MAX_QUEUES* queues** simultaneously



```
#define MAX_QUEUES 10 /* maximum number of queues */
typedef struct queue *queuePointer;
typedef struct queue {
        element data;
        queuePointer link;
        } Node;
queuePointer front[MAX_QUEUES], rear[MAX_QUEUES];
```

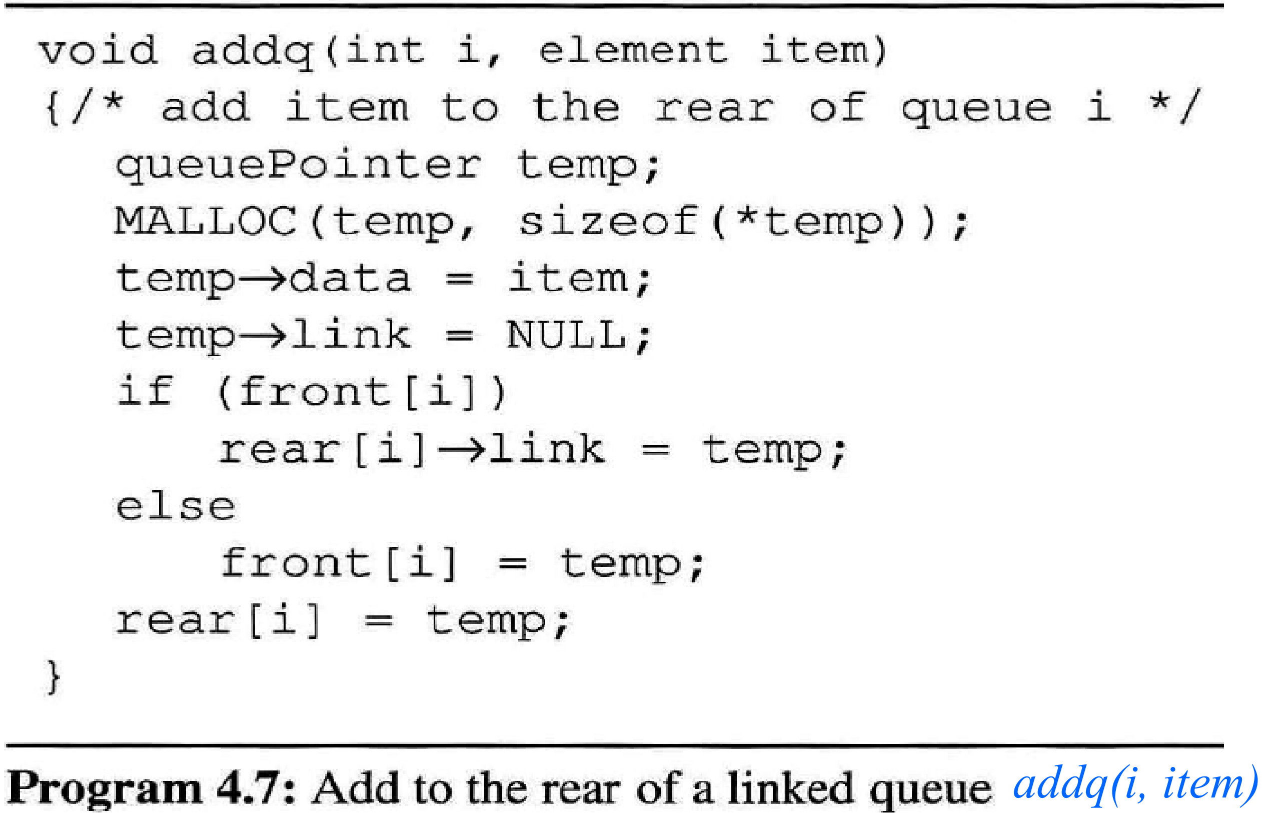
$front[i] = NULL, 0 \leq i < MAX\_QUEUES$ Initial conditions for the queues

$front[i] = \text{NULL iff the } i\text{th queue is empty}$ Boundary condition for the *i*th queue



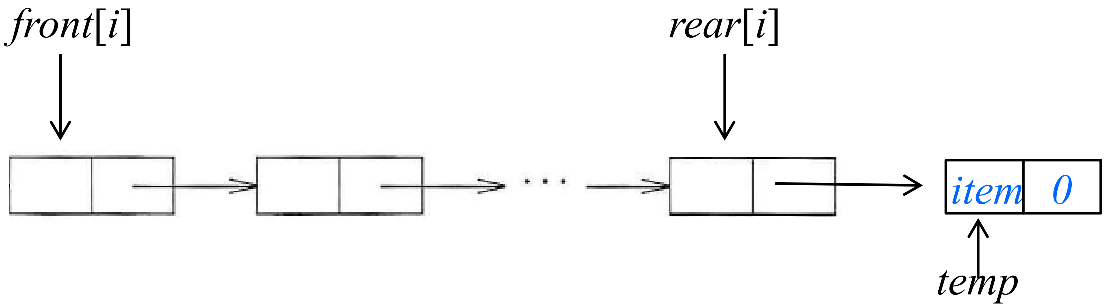
(b) Linked queue

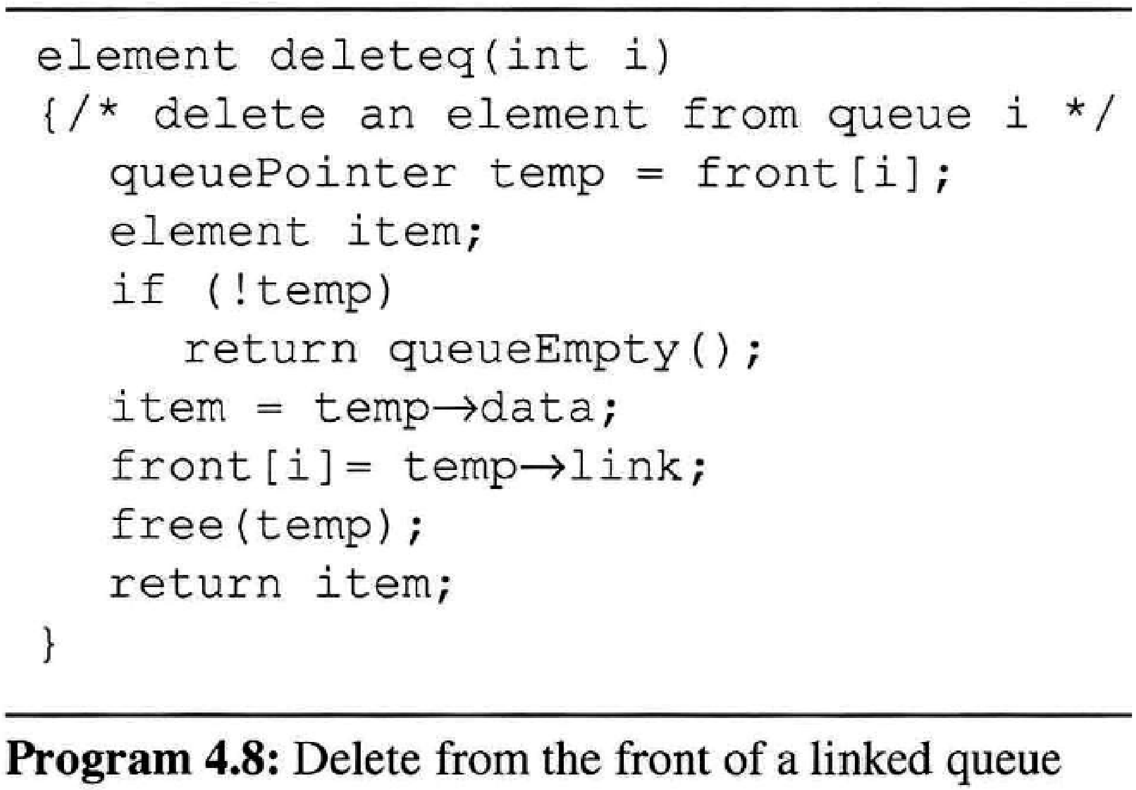
Figure 4.11: Linked stack and queue (2/2)



```
void addq(int i, element item)
{/* add item to the rear of queue i */
   queuePointer temp;
   MALLOC(temp, sizeof(*temp));
   temp→data = item;
   temp→link = NULL;
   if (front[i])
       rear[i]→link = temp;
   else
       front[i] = temp;
   rear[i] = temp;
}
```

Program 4.7: Add to the rear of a linked queue *addq(i, item)*

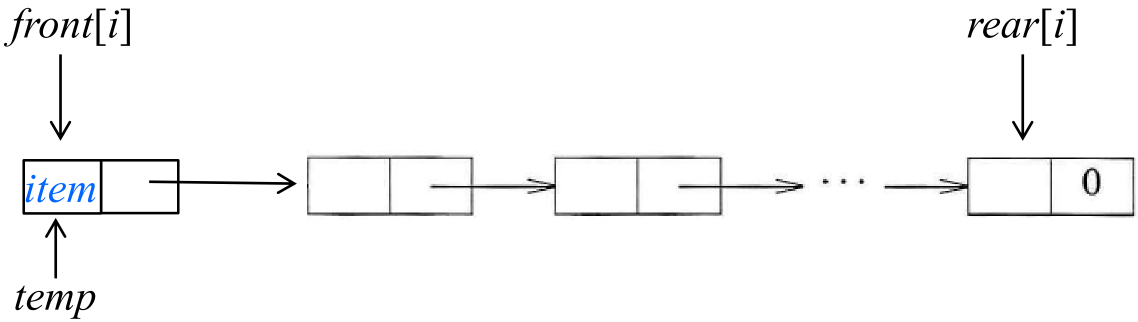




```
element deleteq(int i)
{/* delete an element from queue i */
   queuePointer temp = front[i];
   element item;
   if (!temp)
      return queueEmpty();
   item = temp→data;
   front[i]= temp→link;
   free(temp);
   return item;
}
```

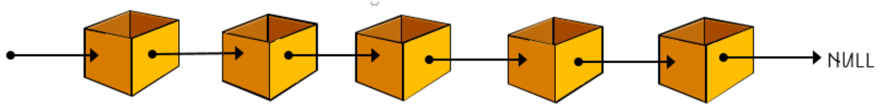
Program 4.8: Delete from the front of a linked queue

item = deleteq(i)

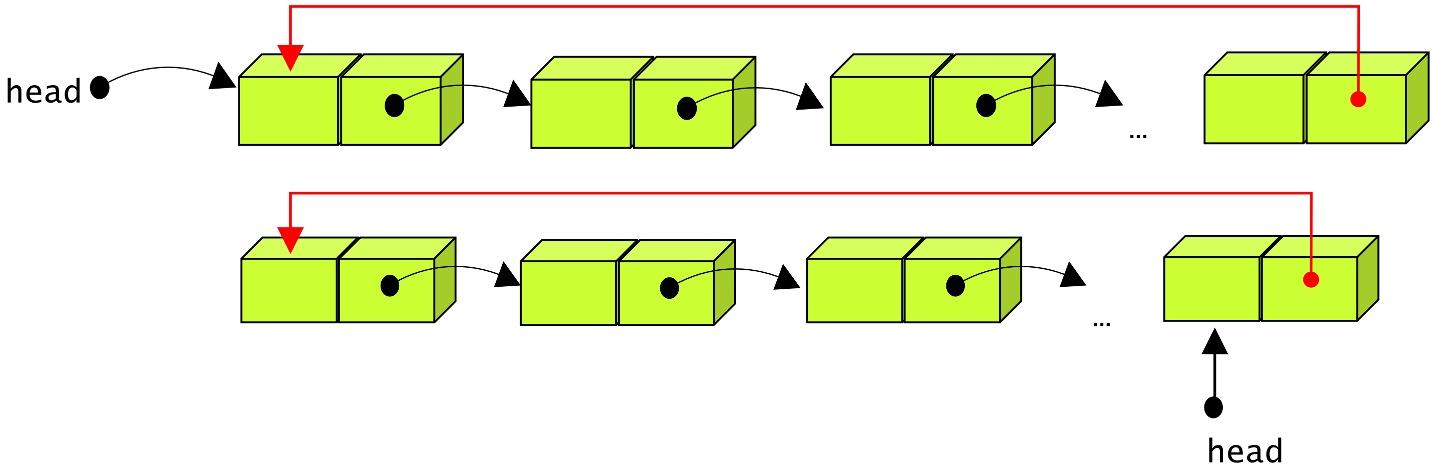


Circular List Representation

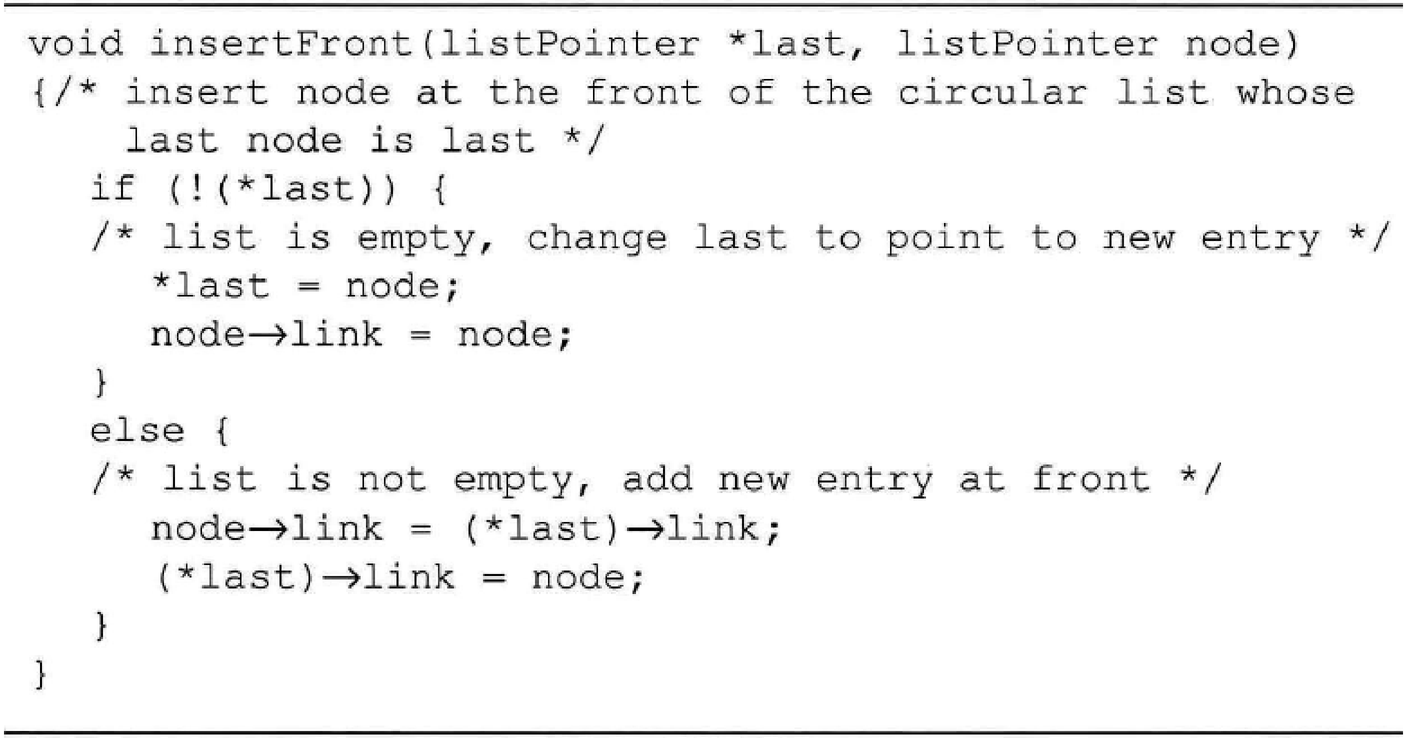
- Chain
  - – A singly linked list in which the last node has a null link



- Circular list
  - – The link filed of the last node points to the first node in the list

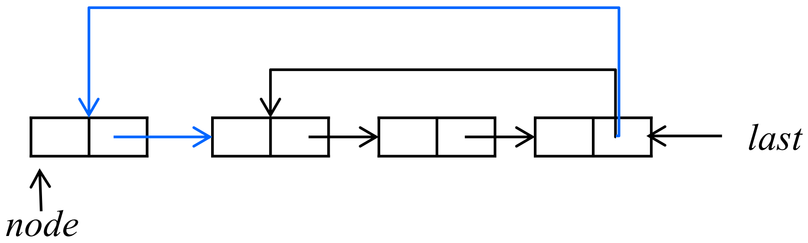


Operations For Circularly Linked Lists



```
void insertFront(listPointer *last, listPointer node)
{/* insert node at the front of the circular list whose
    last node is last */
   if (!(*last)) {
   /* list is empty, change last to point to new entry */
      *last = node;
      node→link = node;
   }
   else {
   /* list is not empty, add new entry at front */
      node→link = (*last)→link;
      (*last)→link = node;
   }
}
```

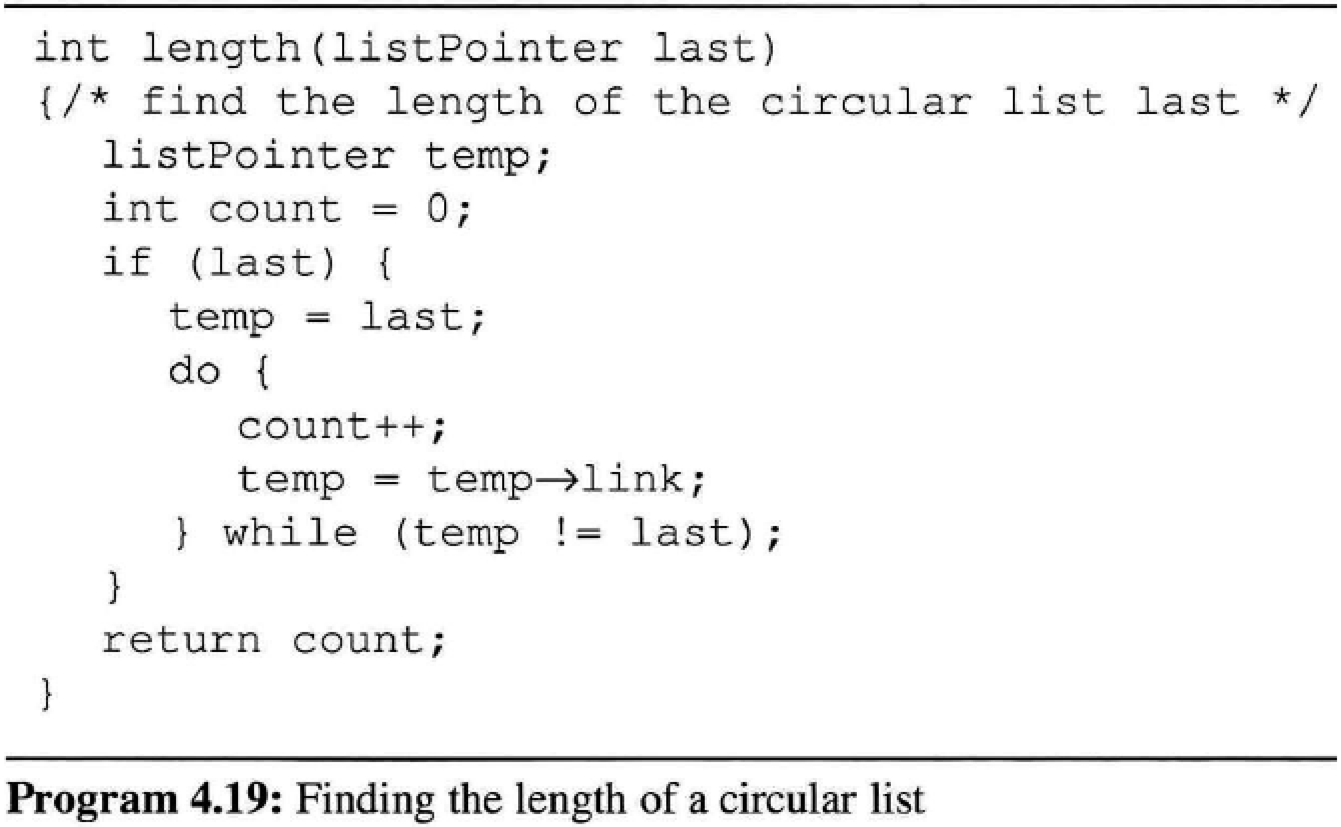
Program 4.18: Inserting at the front of a list



insertFront(&last, node)

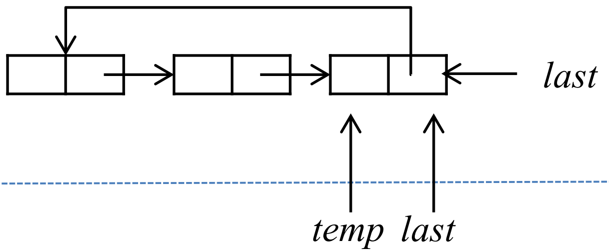
*insertFront(listPointer *last, listPointer node)*

Operations For Circularly Linked Lists



```
int length(listPointer last)
{/* find the length of the circular list last */
   listPointer temp;
   int count = 0;
   if (last) {
      temp = last;
      do {
         count++;
         temp = temp→link;
      } while (temp != last);
   }
   return count;
}
```

Program 4.19: Finding the length of a circular list



length(last)

lengh(listPointer last)

count 0

1