ch3

【1、3】**Array Queue Implementation:** Since all of the work is done in the addq() and deleteq() functions, we only need to print and error message and exit on error or return. Here's the code with the addq() and deleteq() functions.

```
void queue_empty()
{/* check for empty queue is in the deleteq() function */
 fprintf(stderr, "The queue is empty\n");
}

void queue_full()
{/* code to check for a full queue is in the  addq() function */
   fprintf(stderr, "The queue is Full\n");
}

void addq(element item)
{/* add an item to the global queue rear points to the current end of the
queue */
  if (rear == MAX_QUEUE_SIZE-1)
     queue_full();
  else
     queue[++rear] = item;
}

element deleteq()
{/* remove element at the front of the queue */
     if (front == rear)
  queue_empty();
     else
  return queue[++front];
}
```

【2】Similarly all the work for the circular queue is done in the addq() and deleteq() methods. Here's the two methods with the code for printing error message when the queue is empty or full. You can also exit on error, but this makes it difficult to check the correctness of the code to add or delete.

```
void queue_full(){ printf("The queue is Full \n");}
```

```
void queue_empty(){ printf("The queue is empty\n");}
void addq(element item)
{/* add an item to the global queue front and rear mark the two queue ends
*/
     if ((rear+1 == front) || ((rear == MAX_QUEUE_SIZE-1) && !front))
        queue_full();
     else {
        queue[rear] = item;
        rear = (rear+1) % MAX_QUEUE_SIZE;
     }
}
element deleteq()
{/*delete and element from the circular queue */
  int i;
  element temp;
  /* remove front element from the queue and put it in item */
    if (front == rear)   queue_empty();
    else {
       temp = queue[front];
       front = (front+1) % MAX_QUEUE_SIZE;
       return temp;
    }
}
```

【5】The queue starts at the low end of the memory and grows upward. Only a rear pointer is needed because when an item is deleted the queue is shifted downward. The queue is full when the rear and top pointers meet.

The queue operations are:

Adding to the queue

```
void queue_full()
{/* code to check for a full queue is in the
   addq() function */
   fprintf(stderr, "The queue is Full\n");
}

void addq(element item)
{/* add an item to the global queue
     rear points to the current end of the queue */
  if (rear >= top-1)
```

```
        queue_full();
    else
        memory[++rear] = item;
}
```

Deleting from the queue

```
void queue_empty()
{/* check for empty queue is in the deleteq() function */
 fprintf(stderr, "The queue is empty\n");
}

element deleteq()
{/* remove element at the front of the queue */
     int i;
   element item = memory[0];
     if (!rear)
       queue_empty();
     else { /* shift downward */
      for (i = 1; i <= rear; i++)
        memory[i-1] = memory[i];
    rear--;
      return item;
  }
}
```

The stack starts at the high end of memory and grows downward. It is also full when the rear and top pointers meet. The operations are:

Adding to the stack

```
void add(element item)
{/* add an item to the global stack
     top (also global) is the current top of the stack,
     MAX_SIZE is the maximum size */

   if (top <= rear+1)
     StackFull();
   else
      memory[--top] = item;
}
```

Deleting from the Stack

```
void StackEmpty()
{
  printf("The stack is empty.  No item deleted \n");
}

element delete()
{/* remove top element from the stack and put it in item */
  if (top == MAX_SIZE)
    StackEmpty();
  else
    return memory[top++];
}
```

【6】The first stack starts at the low end of the memory and grows upward. The second stack starts at the high end of memory and grows downward. The memory is full when the stack pointers collide. The declarations are:

Stack declarations

```
#define MAX_SIZE 6
typedef struct  {
    int key; } element;

element memory[MAX_SIZE];     /* global queue declaration */
int top [2];
   top[0] = -1;
  top[1] = MAX_SIZE;


CALLS: printf("1. Insert stack 0, 2.Delete stack 0, 3. Insert Stack 1, 4.
Delete Stack 1, 0. Quit: ");
  scanf("%d",&choice);
  while (choice > 0) {
    switch  (choice) {
      case 1: printf("Insert in stack 0: ");
        scanf("%d",&item.key);
        add(0, item);
        break;
      case 2: item = delete(0);
        if (top[0] >= 0 )
         printf ("%d was deleted from the stack 0.\n\n", item.key);
```

```
          break;
        case 3: printf("Enter the number to insert: ");
          scanf("%d",&item.key);
          add(1, item);
          break;
        case 4: item = delete(1);
          if (top[1] < MAX_SIZE)
             printf("%d was deleted from the stack 1 \n\n",item.key);
          break;
       }
```

Adding to a stack

```
void StackFull()
{
   printf("The stack is full.  No item added \n");
}

void add(int topNo, element item)
{/* add an item to the global stack
     top (also global) is the current top of the stack,
     MAX_SIZE is the maximum size */

   if (top[0]+1 >= top[1])
     StackFull();
   else {
      if (!topNo)
       memory[++top[0]] = item;
    else
          memory[--top[1]] = item;
   }
}
```

Deleting from a stack

```
void StackEmpty()
{
  printf("The stack is empty.  No item deleted \n");
}
element delete(int topNo)
{/* remove top element from the stack and put it in item */
  if (!topNo) {
      if (top[0] < 0)
```

```
            StackEmpty();
        return memory[top[0]--];
      }
       else { /*second stack */
        if (top[1] == MAX_SIZE)
         StackEmpty();
        return memory[top[1]++];
      }
     }
```

【7】p137 的 program 3.15

【8】找不到 我自己算的

(1) < { a / [(b - c) + d] } * (e - a) > * c = 8

(2) 看起來不是 postfix 是 prefix ：[(A + B) * C] + { [D * E * (D + E)] + (A * B) } * C = 555