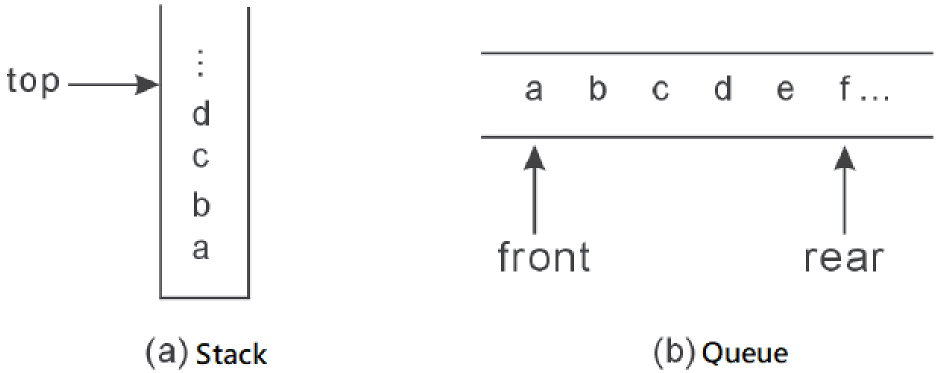
Data Structure

Stack and Queue

23-Apr-23

Basic concepts of Stack and Queue

- Stack and queue are the two most basic themes of data structure.
- A stack is an ordered list, where the insert and delete actions are at the same end, which is usually called the top.
- Since the stack has the property that the elements that go in first will be moved out last. It is also called a Last In First Out (LIFO) list.
- A queue is also called a first-in first-out queue because of its first-in first-out (FIFO) feature.

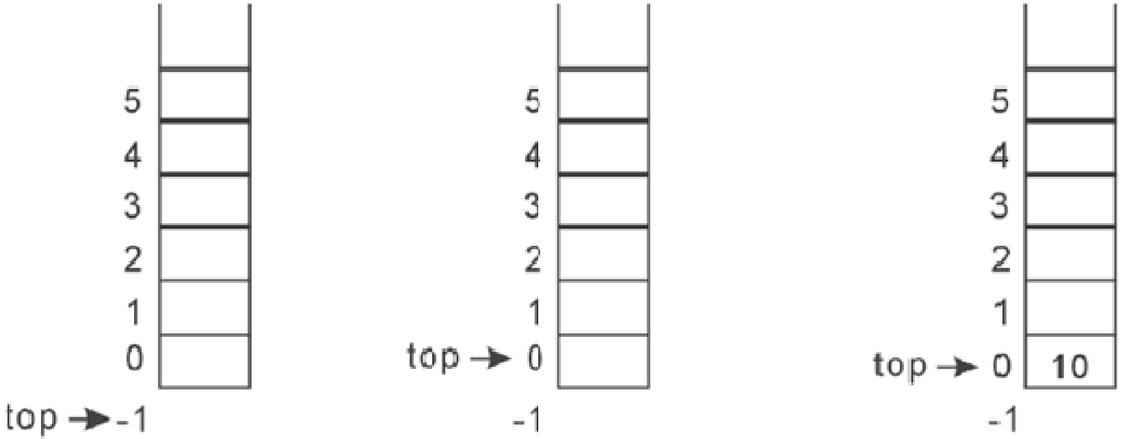


(a) Stack (b) Queue

Stack and Queue

Addition and Deletion of Stack

```
void Stack::push_f(void)
{
  if(top >= MAX-1)   /* When the stack is full, show error message */
    cout << " Stack is full \n";
  else {
    top++;
    cout << " Enter an object into the stack :";
    cin >> a[top];
  }
}
```

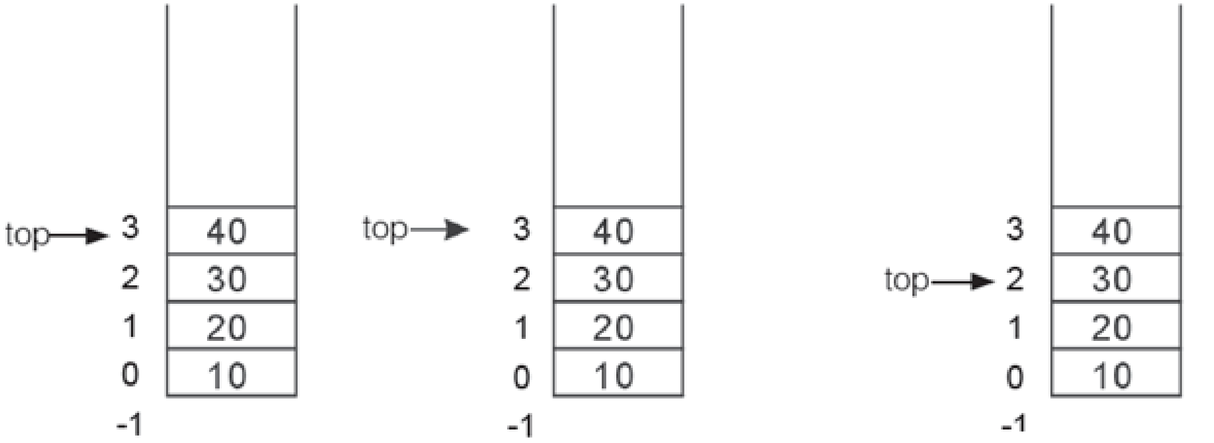


① starting value of top ② Add top by 1 ③ Fill the element (assume 10) with

Add 10 to the stack

Addition and Deletion of Stack

```
void Stack::pop_f(void)
{
  if (top < 0)
     cout << “stack is empty \n";
  else {
     cout << "pop " << a[top] << " from stack \n";
     top--;
}
```



(1) The initial condition of the stack top value is 3

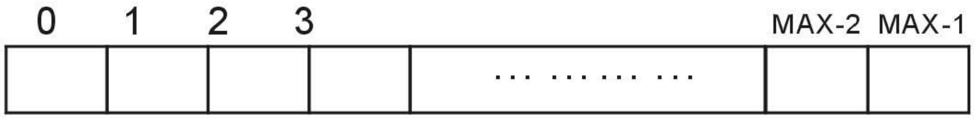
(2) Delete a[3]; i.e. 40

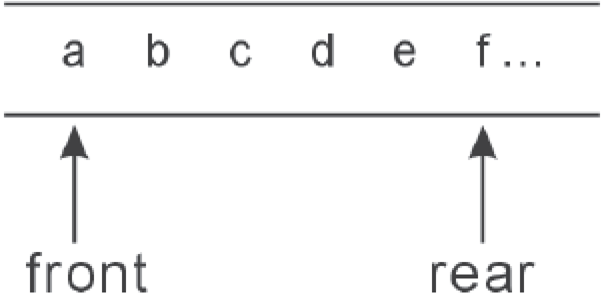
(3) Subtract top by 1

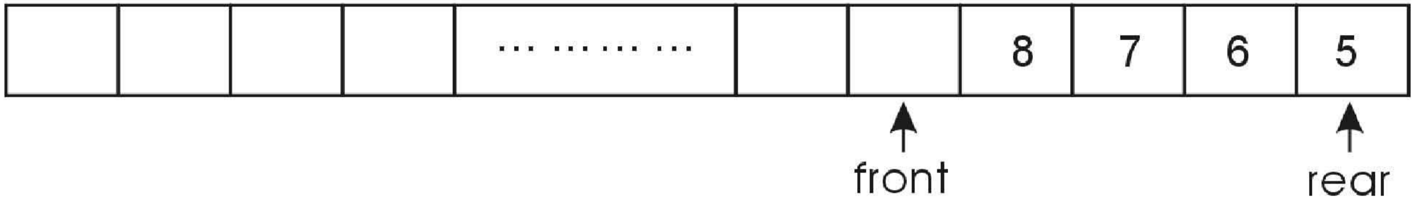
Delete 40 from the stack

Addition and Deletion of Queue

- The operation behavior of the queue is first-in-first-out.
- At the beginning, the front=-1 and the rear=-1 of the queue.
- When adding an element to the queue, the main judgment is whether the rear will exceed the maximum capacity of the array.
- When rear is MAX-1, it means that the array has reached its maximum capacity and no more elements can be added.







Addition and Deletion of Queue

```
void Queue::enqueue_f(void)
{
   if (rear >= MAX-1)
       cout << " Queue is full \n");
   else {
       rear++;
       cout << " Enter an object into the queue : ";
       cin >> a[rear];
   }
}
```

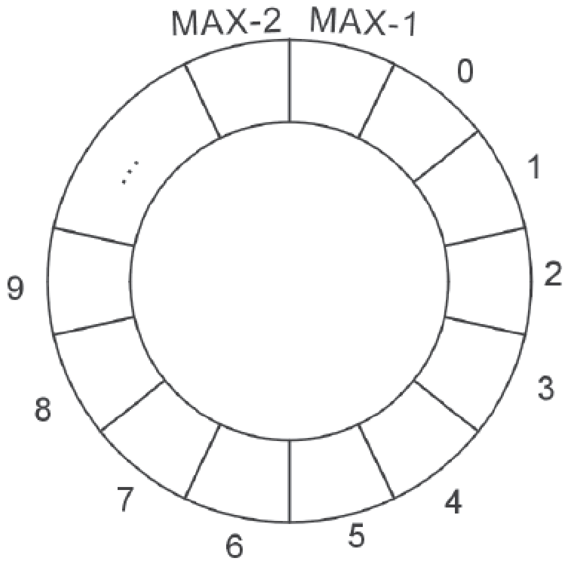
Addition and Deletion of Queue

```
void Queue::dequeue_f(void)
{
  if (front == rear)
     cout << " Queue is empty \n";
  else {
     front++;
     cout << " Delete from queue " << a[front] " \n";
  }
}
```

Circular Queue

- To solve this problem, queues are often represented as a circle queue, CQ(0: MAX-1).
- The initial value of the circular queue is front=rear=MAX-1.
- When there are elements to be added, use the following description.

rear=(rear+1) % MAX;



Circular Queue

```
void Cqueue::encqueue_f(void)
{
  rear=(rear+1) % MAX;
  if (front == rear){
    if (rear == 0 )  /* Return the rear to the correct position */
      rear = MAX-1;
    else
      rear = rear-1 ;
    cout << " Circular queue is full \n";
  }
  else {
    cout << “Enter an object :";
    cin >> cq[rear];
  }
}
```

Circular Queue

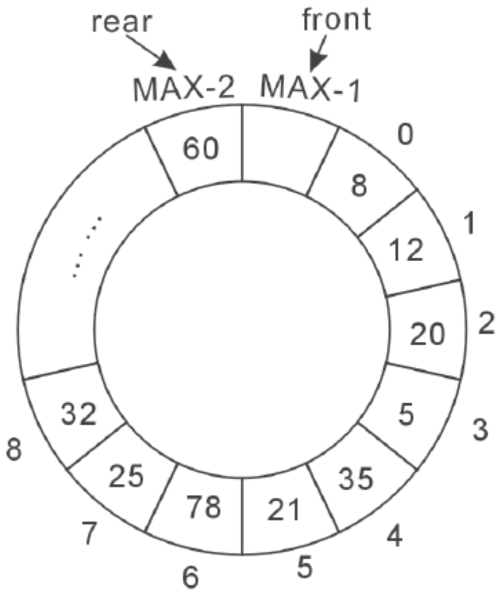
```
void Cqueue::decqueue_f(void)
{
  if ( front == rear)
    cout << " The circular queue is empty \n";
  else {
    front = (front+1) % Max;   /* Move forward in front */
    cout << cq[front] << " Deleted \n";
  }
}
```

Circular Queue

- Among them

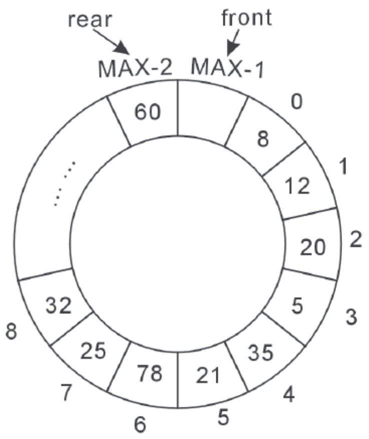
front = (front+1) % MAX ;

- The main purpose is to move the front to the 0 position. Did you notice anything strange?
- Yes, we found that the circular queue wastes a space, as shown in the figure.



Circular Queue

- When the rear is MAX-2 and the front is MAX-1, if an element is added, the program will generate a "full" message.
- If you add it, then when you delete an element, the queue will be empty, which doesn't make sense.



Add

```
void Cqueue::encqueue_f(void)
{
  rear=(rear+1) % MAX;
  if (front == rear){
    if (rear == 0 )  /* Return the rear to the
                               correct position */
      rear = MAX-1;
    else
      rear = rear-1 ;
    cout << " Circular queue is full \n";
  }
  else {
    cout << " Please enter an object :";
    cin >> cq[rear];
  }
}
```

Circular Queue

- Is there a way to make full use of this space? Yes, there is, but it requires an additional variable such as “tag” to assist it.
- In the beginning

front = rear = MAX-1 and tag = 0

Circular Queue

```
void Cqueue::encqueue2_f(void)
{
  if ( front == rear && tag == 1)
      cout << " Circular queue is full \n";
  else {
      rear = (rear+1) % MAX;
      cout << " enter an element :";
      cin >> cq[rear];
      if (front == rear )   /* Determine front is equal to rear */
         tag = 1;       /* If yes, then set the tag to 1 */
  }
}
```

Circular Queue

```
void Cqueue::decqueue2 _f(void)
{
  if (front == rear && tag == 0)
     cout <<  " The circular queue is empty! \n";
  else {
     front = (front+1) % MAX;
     cout << cq[front] << " Deleted \n";
     if (front == rear )
        tag =0;
  }
}
```

Circular Queue

- Comparing encqueue and encqueue2 functions, the main difference is that the latter has more tag variables to judge.
- It will take more time, but also can save a space.
- This is the trade-off between time and space!
- The main difference between add and delete is the tag.
- When the tag is 1, it means the circular queue is full.
- On the contrary, when the tag is 0, it means the circular queue is empty.