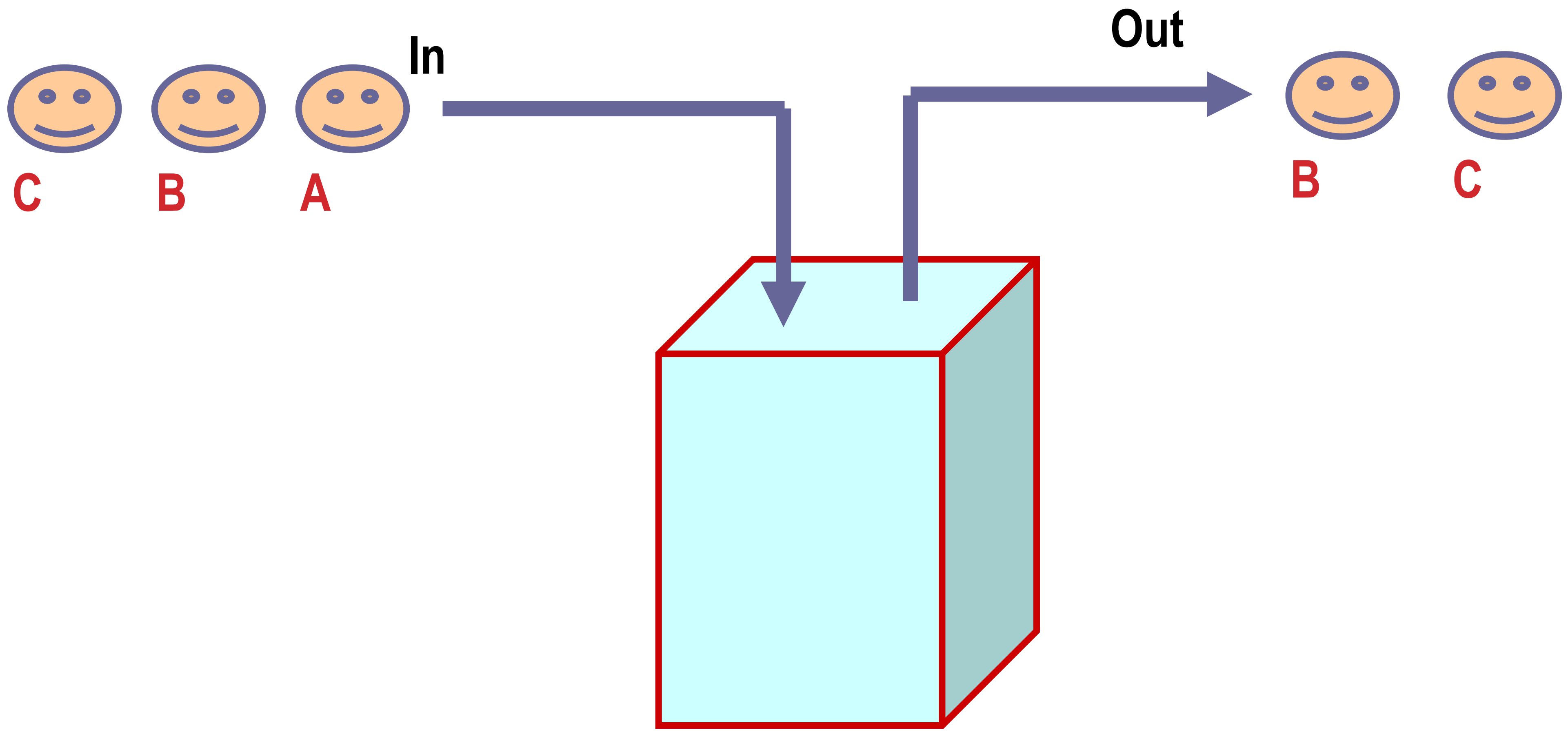
Stacks and Queues

Popular Data Structures

Pallab Dasgupta
Professor,
Dept. of Computer Sc & Engg

Stack is a Last-In-First-Out (LIFO) Data Structure



Stack: *Definition*

```
 #define MAX_STACK_SIZE 100

typedef struct {
                int key;
                /* other fields */
}  element;

typedef struct {
                element list[MAX_STACK_SIZE];
                int top;
 } stack;

 stack z;                                /* Declaration */
 z.top = –1;                             /* Initialization */
```

Stack: *Operations*

```
void push( stack  *s, element item )
{
        if (s-> top >= MAX_STACK_SIZE -1) { stack_full( ); return; }
        (s->top)++;
        s->list[s->top] = item;
}

element pop( stack  *s )
{
        element item;
        if (s->top = –1) return stack_empty( );
        item = s->list[ s->top ];
        (s->top)--;
        return item;
}
```

Application: *Parenthesis Matching*

Given a parenthesized expression, test whether the expression is properly parenthesized.

- Examples:

( ) ({ } [({ } { } ( ))])	is proper
( ) { [ ]	is not proper
({) }	is not proper
) ([ ]	is not proper
([ ]))	is not proper

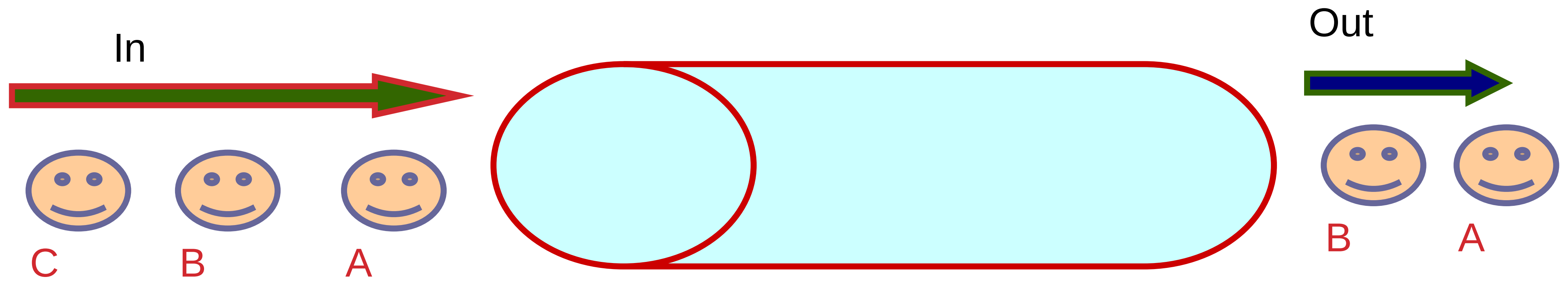
Approach:

- Whenever a left parenthesis is encountered, it is pushed in the stack.
- Whenever a right parenthesis is encountered, pop from stack and check if the parentheses match.
- Works for multiple types of parentheses ( ), { }, [ ]

Parenthesis matching

```
while (not end of string) do
{
        a = get_next_token();
        if (a is ‘(‘ or ‘{‘ or ‘[‘) push (a);

        if (a is ‘)’ or ‘}’ or ‘]’)
        {
                if (is_stack_empty( )) {
                        print (“Not well formed”);
                        exit();
                }
                x = pop();
                if (a and x do not match) {
                        print (“Not well formed”);
                        exit(); }
        }
}
if (not is_stack_empty( )) print (“Not well formed”);
```

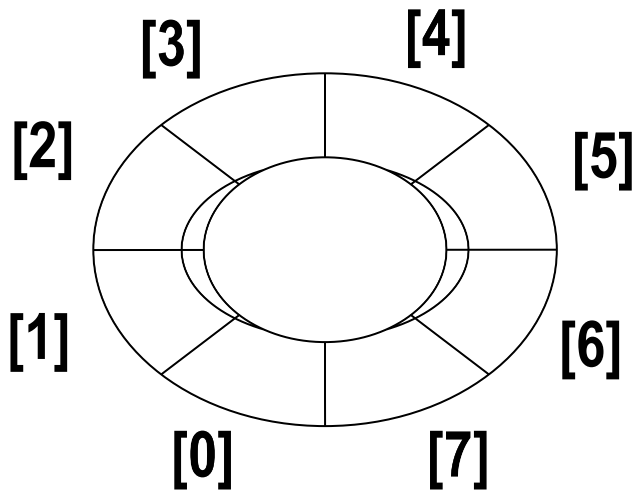
Queue is a First-In-First-Out Data Structure



Queue: *Definition*

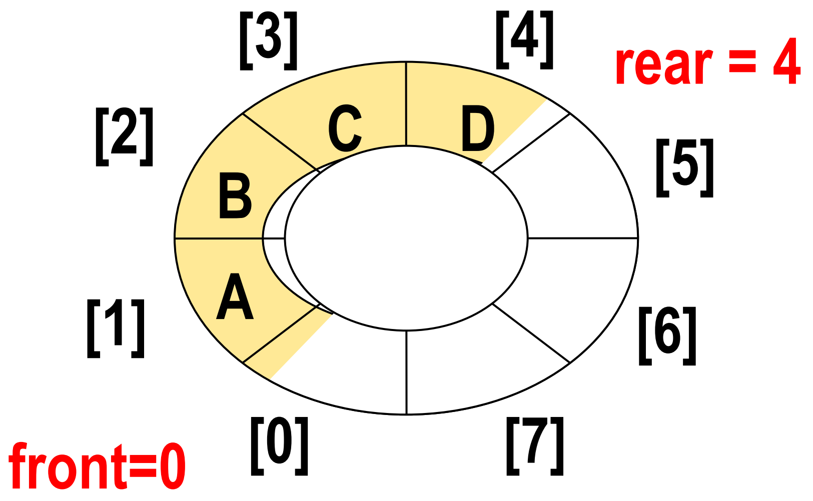
```
 #define MAX_QUEUE_SIZE 100

typedef struct {
            int key;
            /* other fields */
}  element;

typedef struct {
            element list[MAX_QUEUE_SIZE];
            int front;
            int rear;
 } queue;

 queue z;                                    /*Declaration */
 z.front = z.rear = 0;                       /* Initialization */
```

Queue: *Circular Implementation*

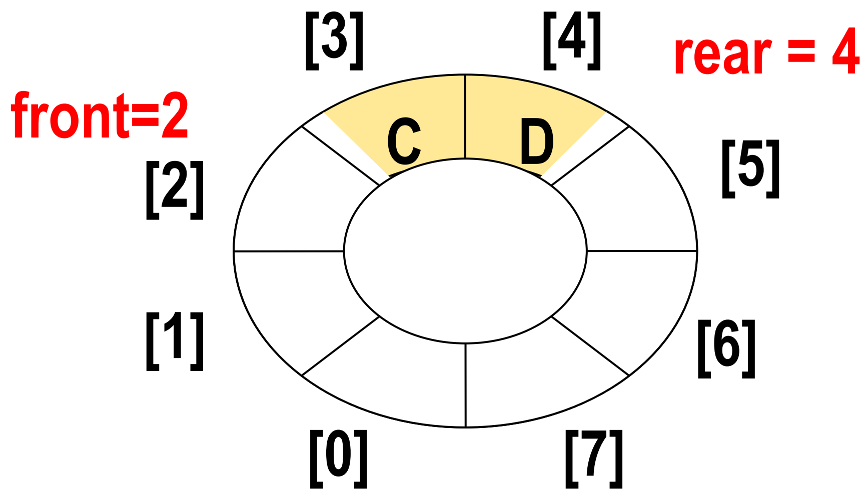


front=0
rear=0

Queue Empty



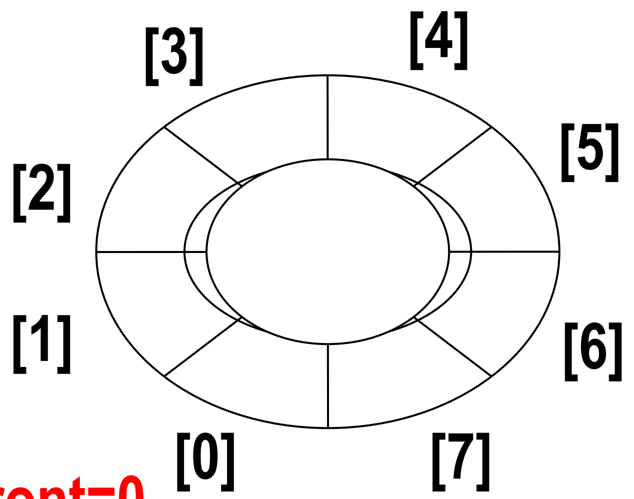
After insertion of A, B, C, D



After deletion of of A, B

Queue: *Circular Implementation*

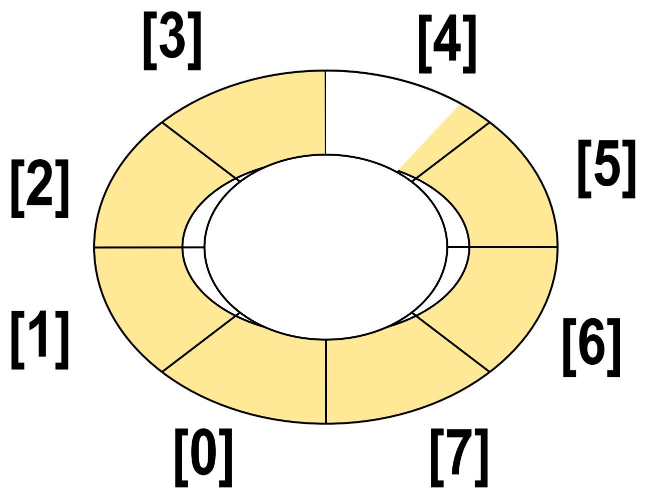
front: *index of queue-head (always empty – why?)*
rear: *index of last element, unless rear = front*



front=0
rear=0

Queue Empty

rear = 3
front=4



Queue Full

Queue Empty Condition: *front == rear*
Queue Full Condition: *front == (rear + 1) % MAX_QUEUE_SIZE*

Queue: *Operations*

```
void addq( queue *q, element item )
{
        q->rear = (q->rear  + 1)% MAX_QUEUE_SIZE;
        if (q->front == q->rear) { queue_full( ); return; }
        q->list[q->rear] = item;
}

element deleteq( queue *q )
{
        element item;
        if (q->front == q->rear) return empty_queue( );
        q-> front = (q-> front + 1)% MAX_QUEUE_SIZE;
        return q->list[q->front] ;
}
```