Stacks and Queues

Fan-Hsun Tseng

Department of Computer Science and Information Engineering

National Cheng Kung University

Outline

- Templates in C++
- The Stack Abstract Data Type
- The Queue Abstract Data Type
- A Mazing Problem
- Evaluation of Expressions

Templates in C++

- Template function in C++ makes it easier to reuse classes and functions.
- A template can be viewed as a variable that can be instantiated to any data type, irrespective of whether this data type is a fundamental C++ type or a user-defined type.

Selection Sort Using Template

```
Template <class T>
void SelectionSort(T *a, int n)
// sort a[0] to a[n-1] into non-decreasing order
{
   for (int i = 0; i < n; i++)
   {
      int j = i;
      // find smallest integer in a[i] to a[n-1]
      for (int k = i+1; k < n; k++)
         if (a[k] < a[j]) { j = k;}
      // interchange
      swap(a[i] = a[j]);
}
```

Selection Sort Using Template (Contd.)

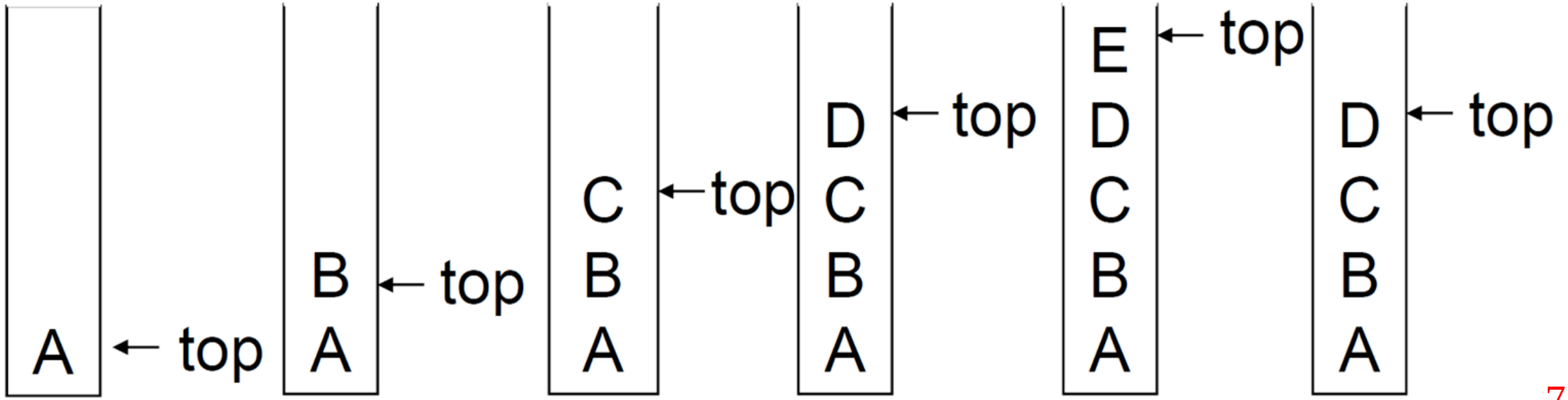
```
float farray[100];
int intarray[250];
……
// Assume that the arrays are initialized at this point
SelectionSort(farray, 100)
SelectionSort(intarray, 250)
```

Outline

- Templates in C++
- The Stack Abstract Data Type
- The Queue Abstract Data Type
- A Mazing Problem
- Evaluation of Expressions

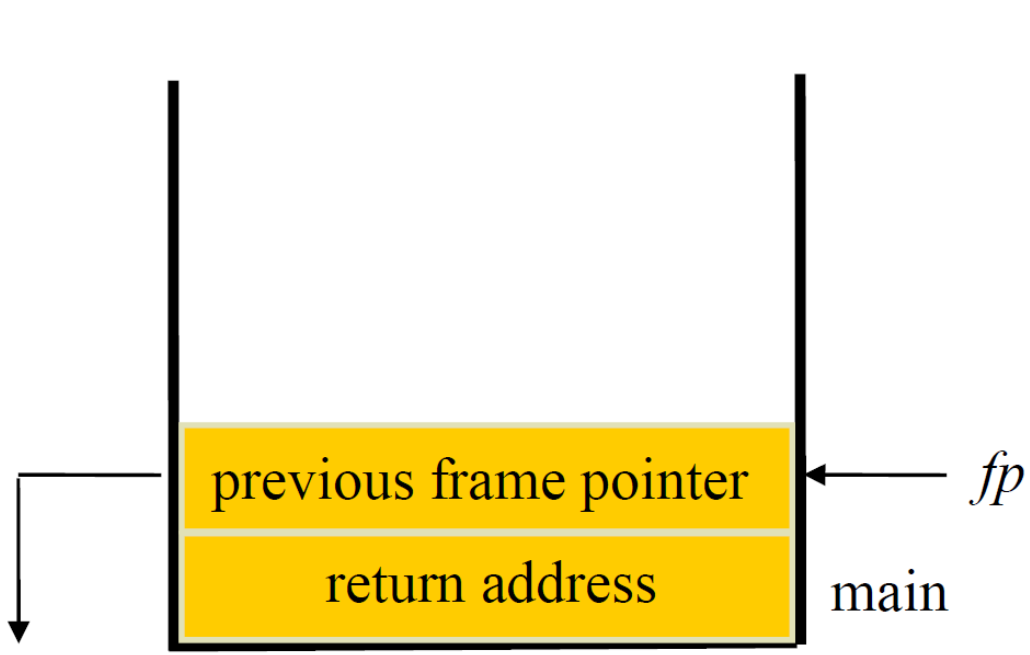
Stack: Last-In-First-Out (LIFO) List

- Push
  - Add an element into a stack
- Pop
  - Get and delete an element from a stack

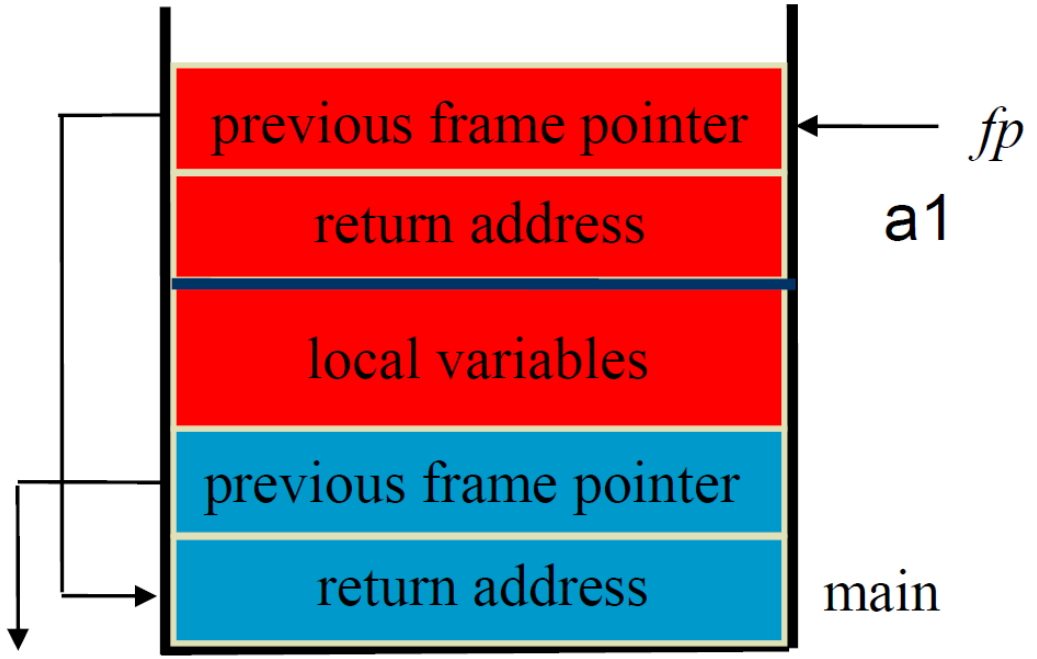


An Application of Stack: Stack Frame of Function Call

- System stack before and after function call



(a)



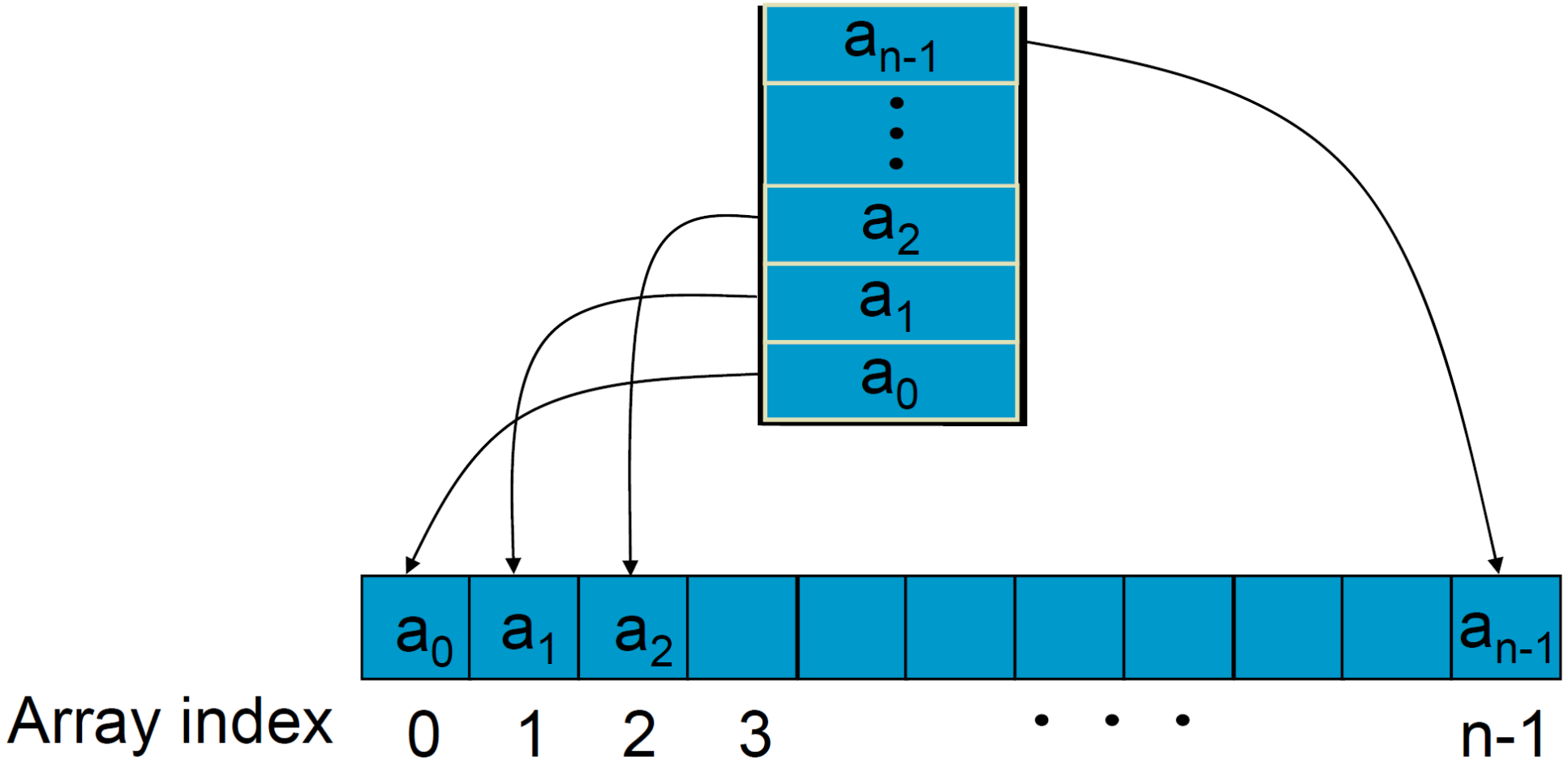
(b)

Abstract Data Type for Stack

```
Template <class KeyType>
class Stack
{  // objects: A finite ordered list with zero or more elements
public:
   Stack (int MaxStackSize = DefaultSize);
   // Create an empty stack whose maximum size is MaxStackSize
   Boolean IsFull();
   // if number of elements in the stack is equal to the maximum size of the stack
   // return TRUE(1) else return FALSE(0)
   void Add(const KeyType& item);
   // if IsFull(), then StackFull(); else insert item into the top of the stack.
   Boolean IsEmpty();
   // if number of elements in the stack is 0, return TRUE(1) else return FALSE(0)
   KeyType* Delete(KeyType& );   // if IsEmpty(), then StackEmpty() and return 0;
   // else remove and return a pointer to the top element of the stack.
};
```

Implementation of Stack by Array

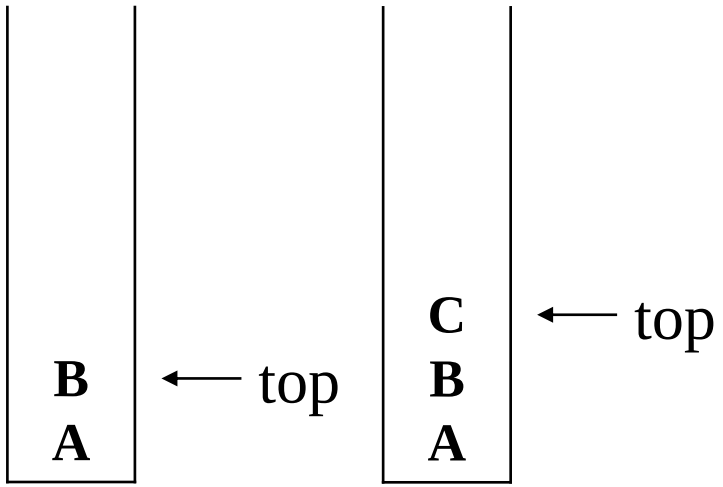
- How to check whether a stack is full or empty?



```
Private:
int top;
KeyType *stack;
int MaxSize;
template<class KeyType>
Stack<KeyType>::Stack(int MaxStackSize):
                        MaxSize(MaxStackSize) {
  stack=new KeyType[MaxSize];
  top=-1;
}
template<class KeyType>
inline Boolean Stack<KeyType>::IsFull() {
if (top==MaxSize-1) return TRUE;
else return FALSE;
}
template<class KeyType>
inline Boolean Stack<KeyType>::IsEmpty() {
if (top==-1) return TRUE;
else return FALSE;
}
```

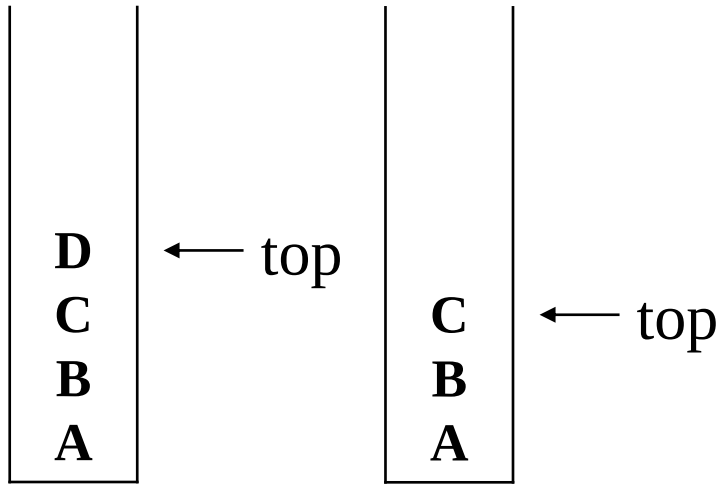
Add An Element to A Stack

```
Template <class KeyType>
void Stack<KeyType>::Add(const KeyType& x)
{
   /* add an item to the global stack */  if (IsFull())
   stack_full( );  else
   stack[++top]=x;
}
```



Delete An Element from A Stack

```
Template <class KeyType>
KeyType* Stack<KeyType>::Delete(KeyType& x)
{
  // return the top element from the stack
  if (IsEmpty())
  {
        stack_empty( );
        /* returns and error key */
        return 0;
  }
  x=stack[top--];
  return &x;
}
```

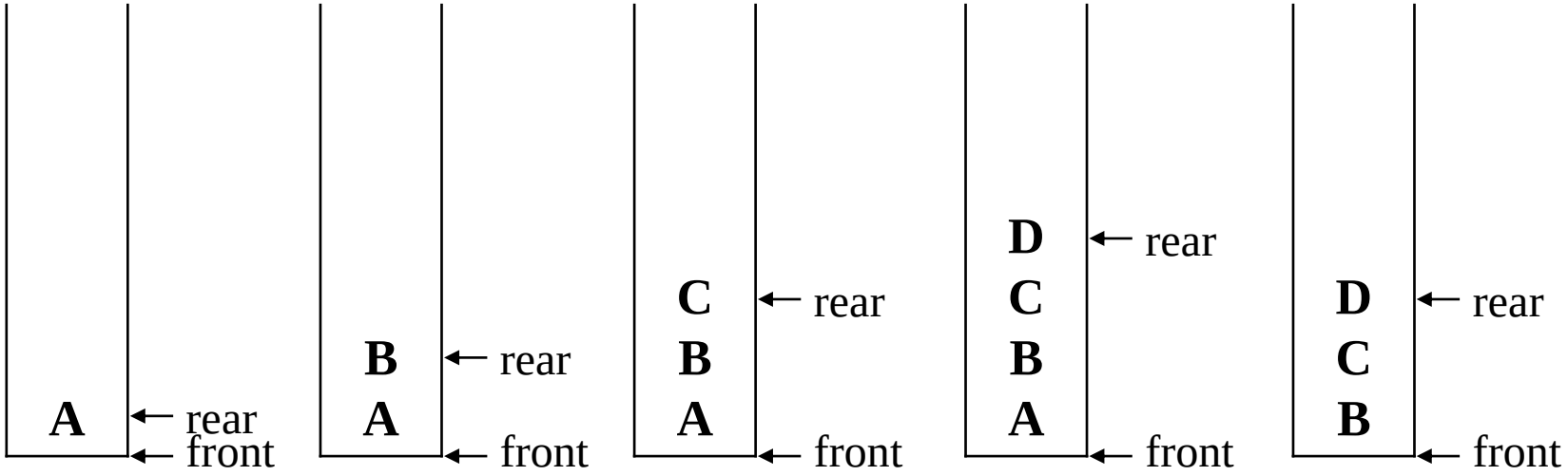


Outline

- Templates in C++
- The Stack Abstract Data Type
- The Queue Abstract Data Type
- A Mazing Problem
- Evaluation of Expressions

Queue: First-In-First-Out (FIFO) List

- Add an element into a queue
- Get and delete an element from a queue
- Variation
  - Priority queue



Application: Job Scheduling

- Insertion and deletion from a sequential queue

front	rear	Q[0]	Q[1]	Q[2]	Q[3]	Comments
-1	-1					Queue is empty
-1	0	J1				J1 is added
-1	1	J1	J2			J2 is added
-1	2	J1	J2	J3		J3 is added
0	2		J2	J3		J1 is deleted
1	2			J3		J2 is deleted

Abstract Data Type of Queue

```
Template <class KeyType>
class Queue {
// objects: A finite ordered list with zero or more elements
  public:
    Queue(int MaxQueueSize = DefaultSize);
    // Create an empty queue whose maximum size is MaxQueueSize
    Boolean IsFull();   /* if number of elements in the queue is equal to the maximum size of
                           the queue, return TRUE(1); otherwise, return FALSE(0) */
    void Add(const KeyType& item);   // if IsFull(), then QueueFull(); else insert item at rear of the queue
    Boolean IsEmpty();   // if number of elements in the queue is equal to 0,
                            return TRUE(1) else return FALSE(0)
    KeyType* Delete(KeyType&);
    // if IsEmpty(), then QueueEmpty() and return 0;
    // else remove the item at the front of the queue and return a pointer to it
};
```

Implementation 1: Using Array

```
Private:
   int front,rear;
   KeyType *queue;
   int MaxSize;
Template<class KeyType>
Queue<KeyType>::Queue(int MaxQueueSize):MaxSize(MaxQueueSize) {
   queue=new KeyType[MaxSize];
   front=rear= -1;
}
template<class KeyType>
inline Boolean Queue<KeyType>::IsFull() {
   if (rear==MaxSize-1) return TRUE;
   else return FALSE;
} template<class KeyType>
inline Boolean Stack<KeyType>::IsEmpty() {
   if (front==rear) return TRUE;
   else return FALSE;
}
```

How to check whether a stack is full or empty?

Add An Element to A Queue

```
Template <class KeyType>
void Queue<KeyType>::Add(const KeyType& x)
{
   /* add an item to the global stack */
   if (IsFull())
      QueueFull( );
   else
      queue[++rear]=x;
}
```

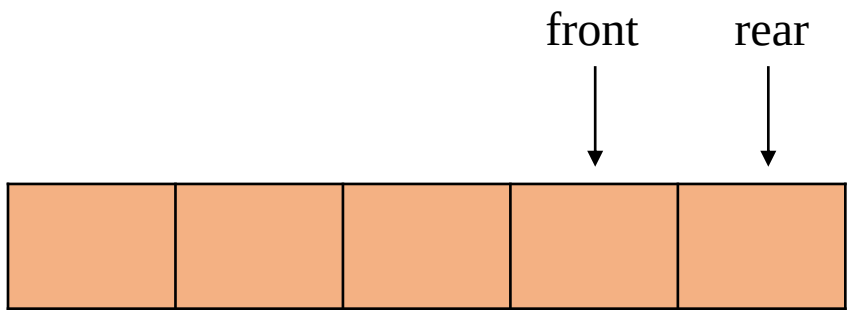
Delete An Element from A Queue

```
Template <class KeyType>
KeyType* Queue<KeyType>::Delete()
{
   // return the top element from the stack
   if (IsEmpty())
   {
      /* returns and error key */
     QueueEmpty( );
     return 0;
   }
   x=queue[++front];
   return x;
}
```

Problem:
There may be available space when QueueFull is true, i.e., data movements are required.

Problem

- As the elements enter and leave the queue, the queue gradually shifts to the right.
  - Eventually the rear index equals MaxSize-1, suggesting that the queue is full even though the underlying array is not full
- Solution:
  - Use a function to move the entire queue to the left so that front=-1
  - It is time-consuming
  - Time complexity=O(MaxSize)

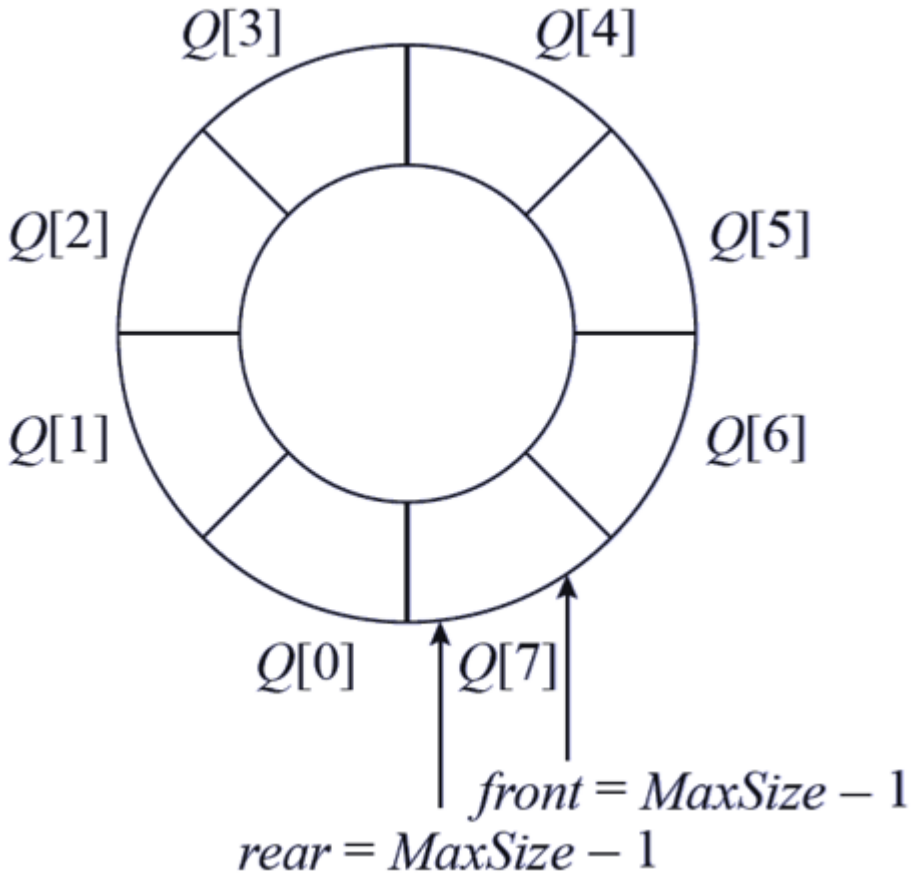


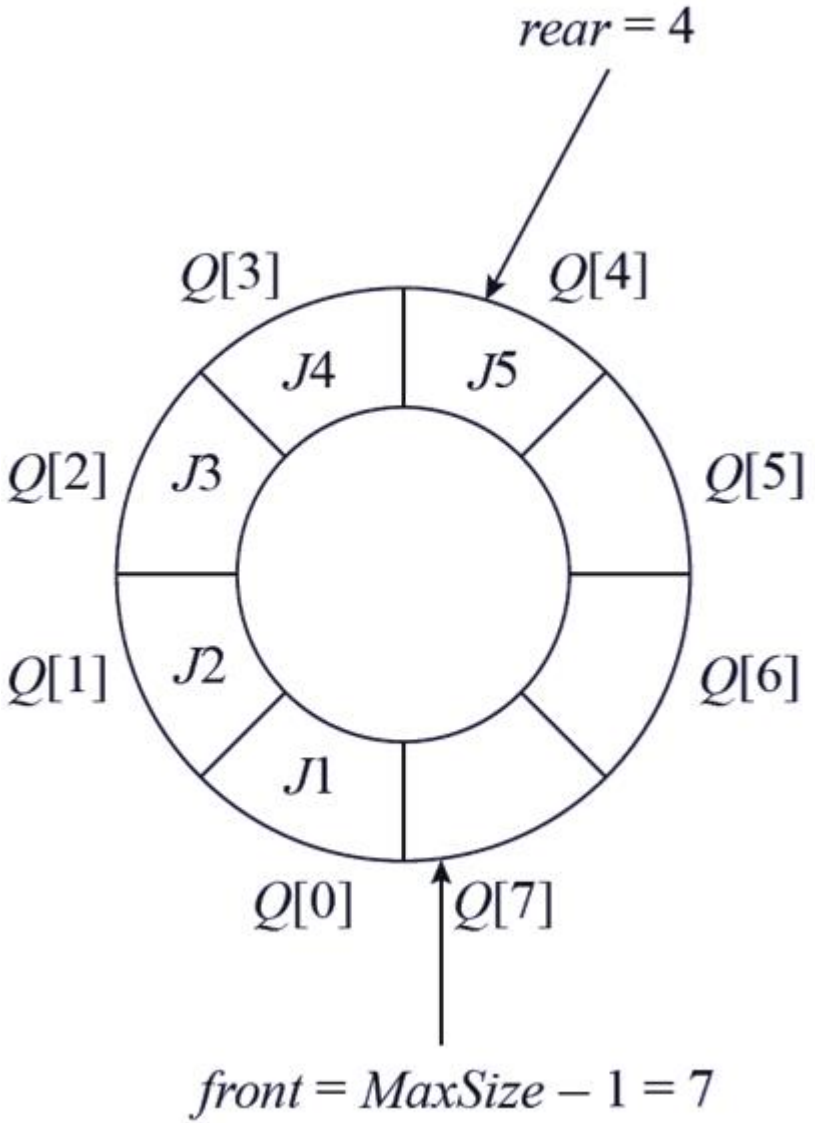
Implementation 2: Regard an Array as a Circular Queue

- Two indices
  - front: point out the start (empty space)
  - rear: current end
- Problem
  - In order to distinguish whether a circular queue is full or empty, one space is left when queue is full

An Example Circular Queue

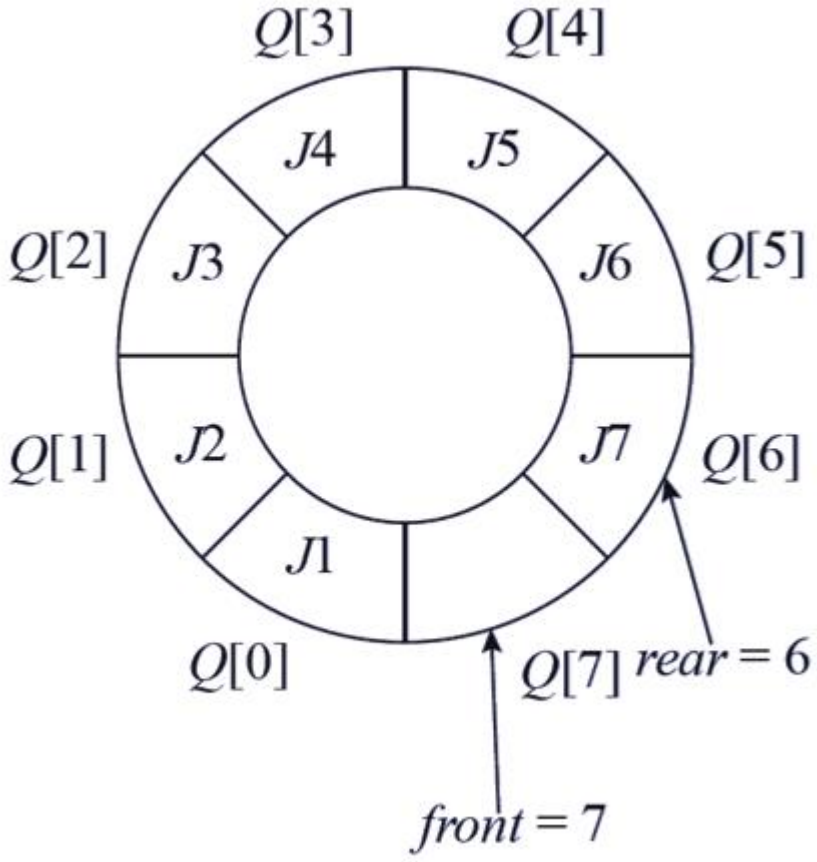
- Add 5 items to an empty circular queue

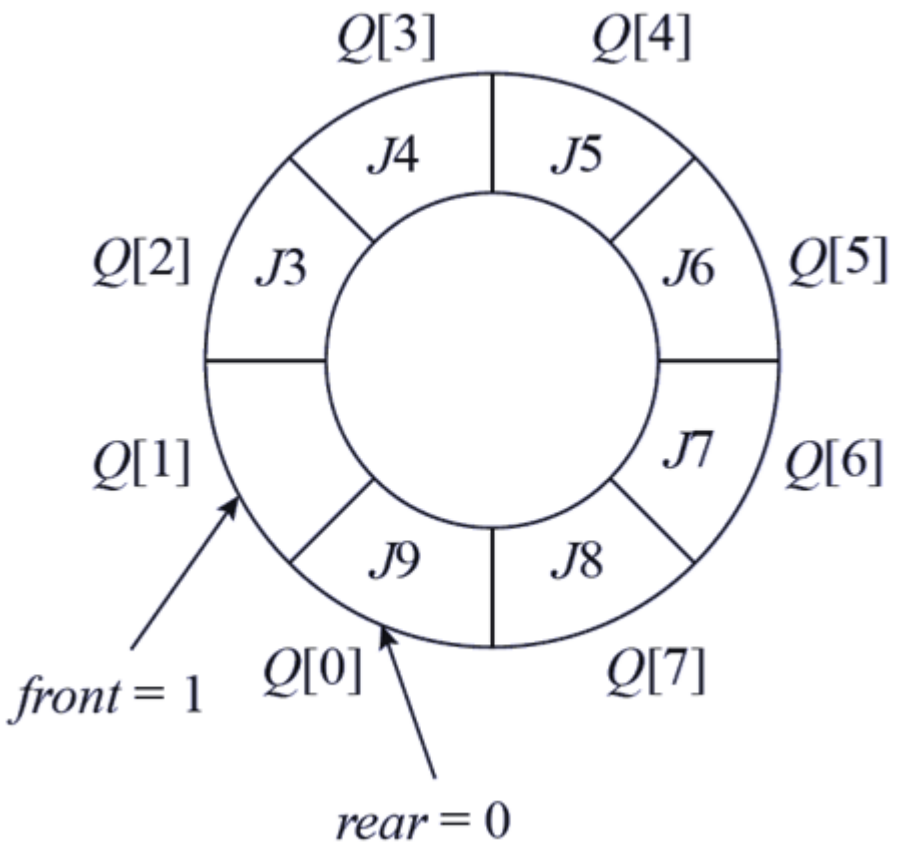




An Example Circular Queue (Contd.)

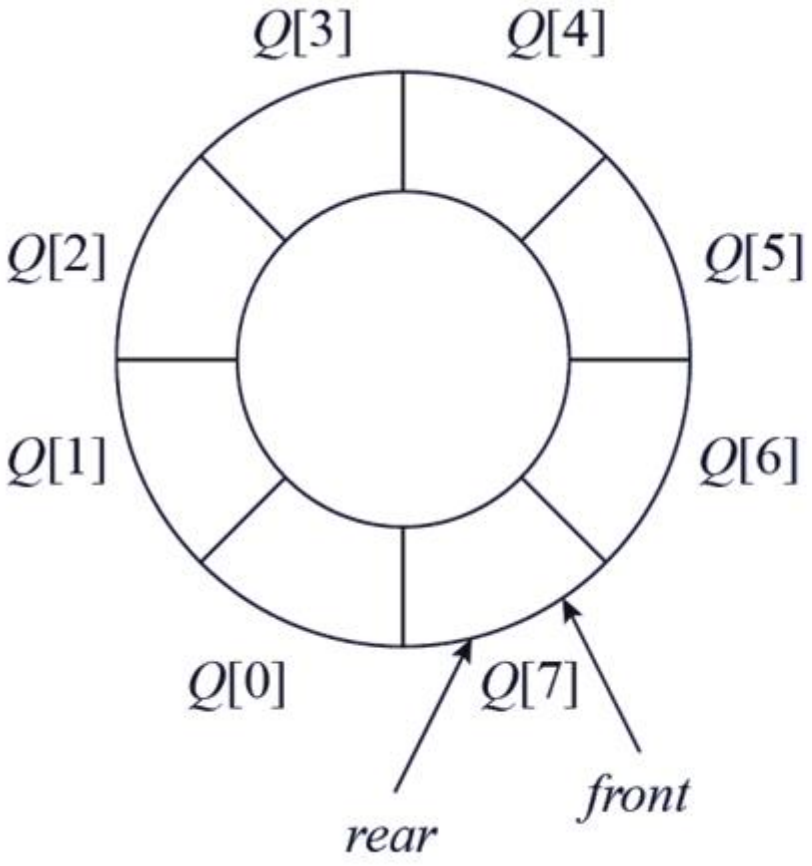
- Add 2 more items as a full circular queue
- Delete J1 and J2, and then add J8 and J9

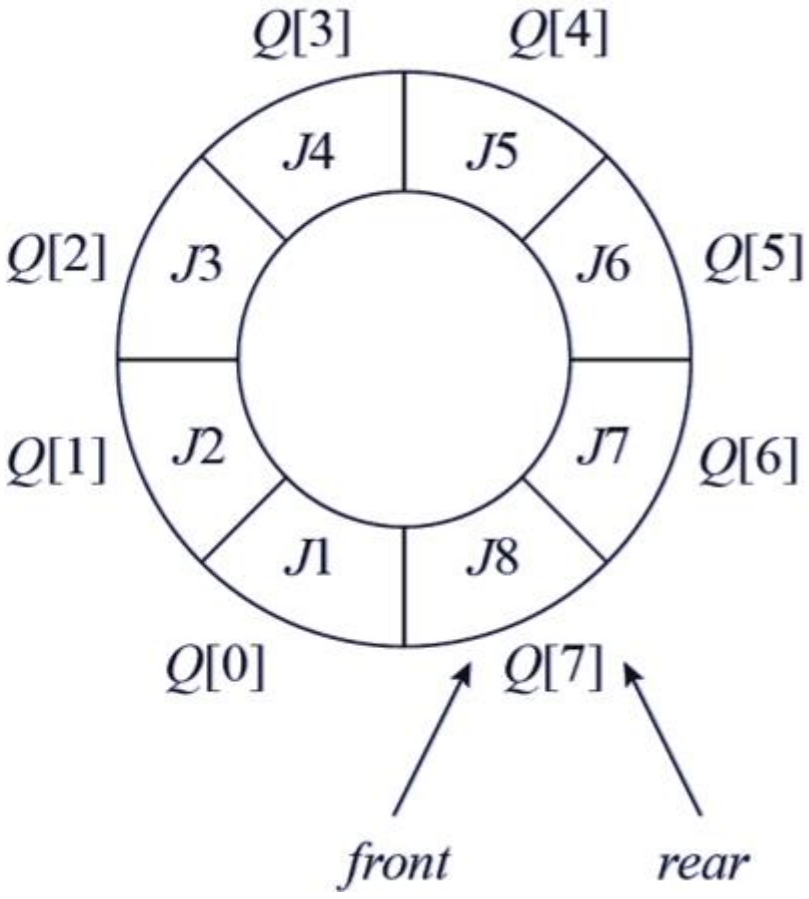




An Example Circular Queue (Contd.)

- How to recognize an empty and a full circular queue?





Add An Element to A Circular Queue

```
Template<class KeyType>
void Queue<KeyType>::Add(const KeyType&)
{
   int newrear = (rear+1)%MaxSize;
   if (front==newrear)
      QueueFull();
   else
      queue[rear=newrear]=x;
}
```

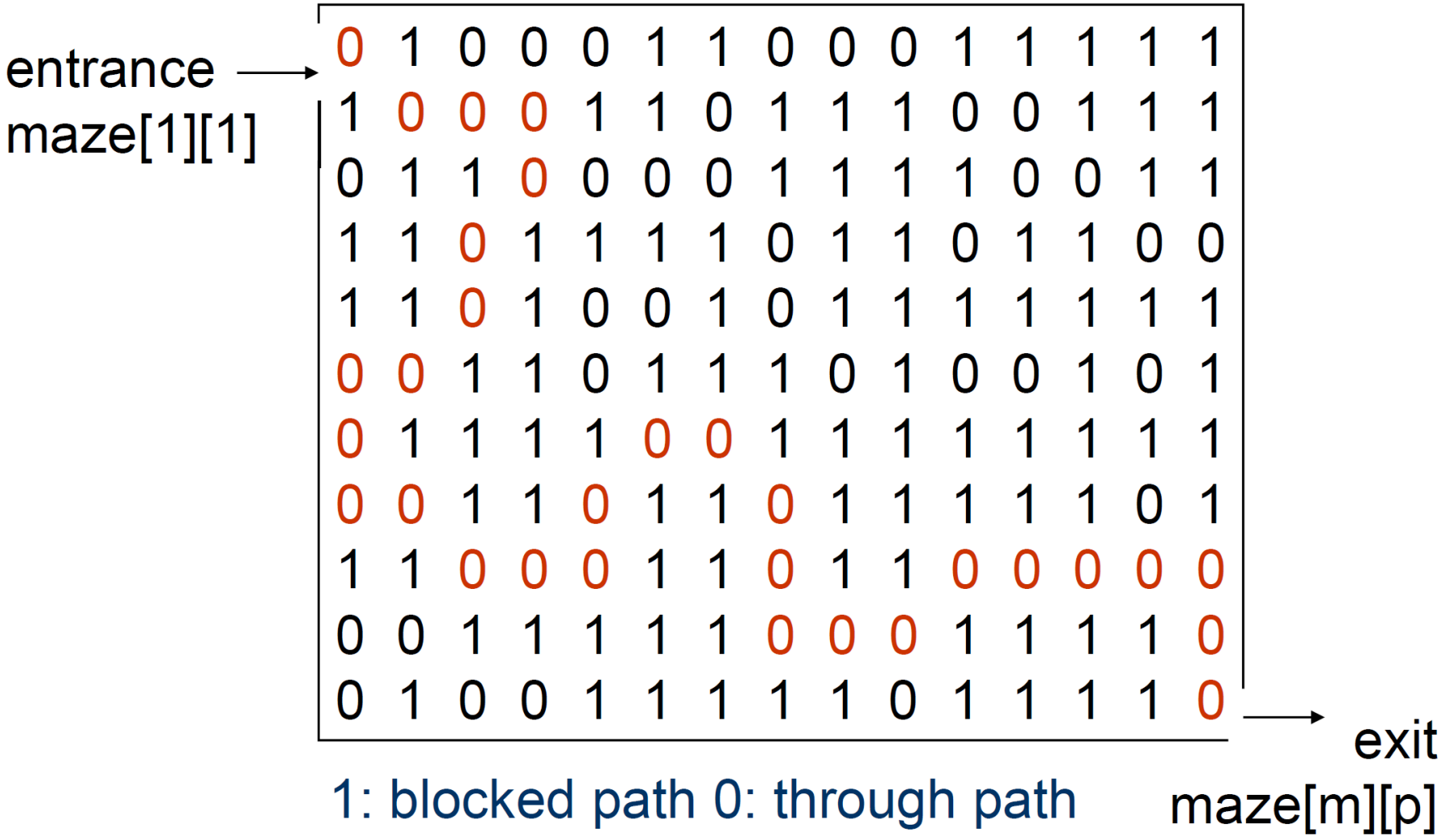
Delete An Element from A Circular Queue

```
Template<class KeyType>
KeyType* Queue<KeyType>::Delete(KeyType& x)
{
  /* remove front element from the queue */  if (front == rear)
  {
  QueueEmpty();
  return 0;
  }
  front = (front+1)%MaxSize;
  x=queue[front];
  return &x;
}
```

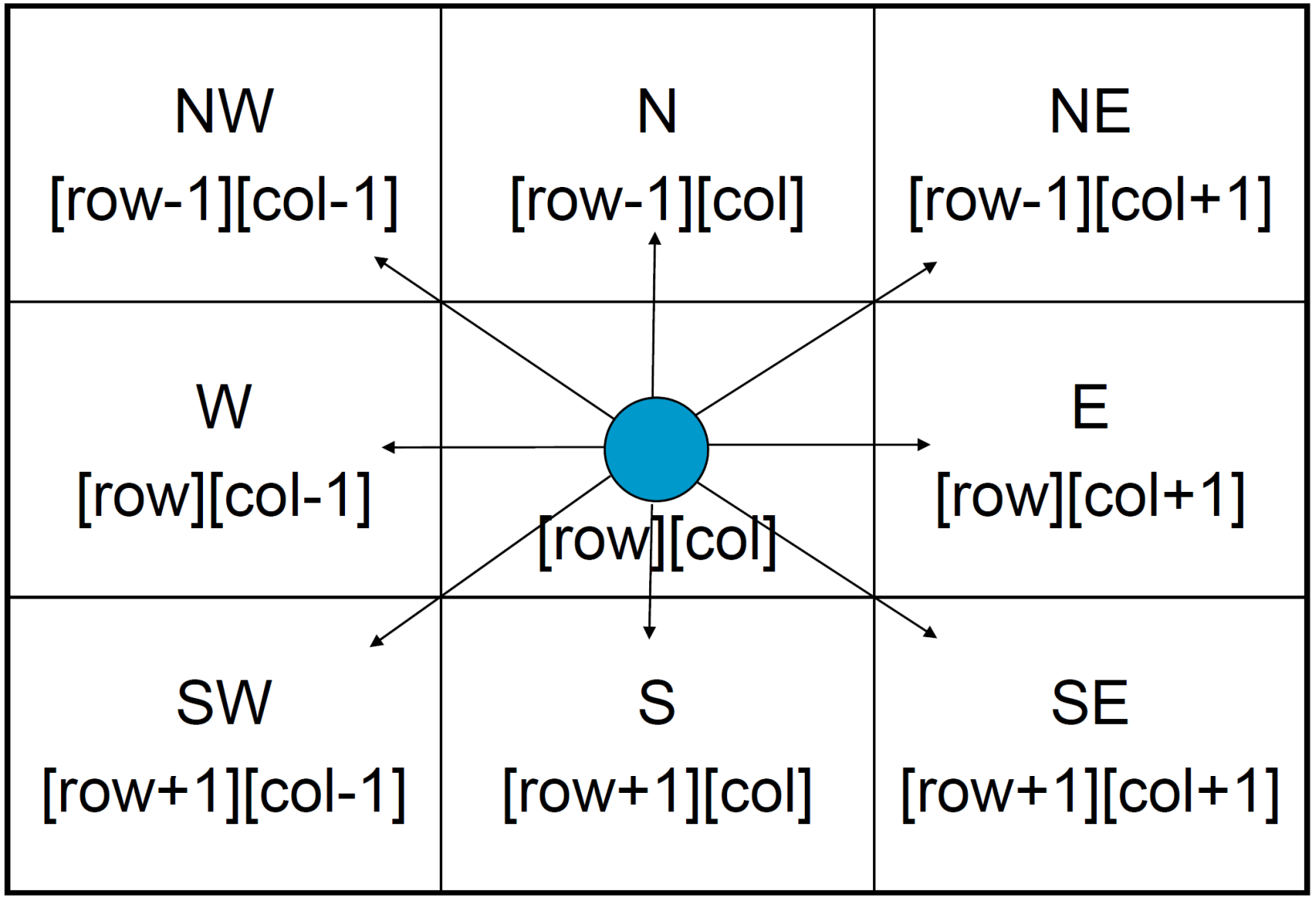
Outline

- Templates in C++
- The Stack Abstract Data Type
- The Queue Abstract Data Type
- A Mazing Problem
- Evaluation of Expressions

A Mazing Problem



A Possible Representation



```
typedef struct {
   int a; /* row */
   int b; /* col */
} offsets;
offsets move[8];
/*array of moves for
each direction*/
```

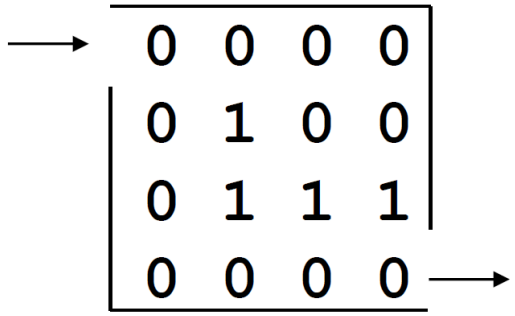
```
next_row = row + move[dir].a;
next_col = col + move[dir].b;
```

Name	Dir	Move[dir].a	Move[dir].b
N	0	-1	0
NE	1	-1	1
E	2	0	1
SE	3	1	1
S	4	1	0
SW	5	1	-1
W	6	0	-1
NW	7	-1	-1

Use a Stack to Keep Pass History

- What is the maximal size of the stack?
  - A maze is represented by a two-dimensional array *maze[m][p]*
  - Since each position is visited at most once, at most *m*×*p* elements can be placed in the

```
typedef struct {
   int x;
   int y;
   int dir;
} item;
item stack[m*p];
```



Stack Position		Action	
-------	(1,1)	Mark (1,1)	Move back
(1,1,2)	(1,2)	Mark (1,2)	(1,4,4) (2,4)
			(1,3,2)
(1,2,2)	(1,3)	Mark (1,3)	(1,2,2)
(1,1,2)			(1,1,2)
(1,3,2)	(1,4)	Mark (1,4)	Move back
(1,2,2)			
(1,1,2)			(1,3,2) (1,4)
			(1,2,2)
(1,4,4)	(2,4)	Mark (2,4)	(1,1,2)
(1,3,2)			
(1,2,2)			...
(1,1,2)			

(2,4,6) (2,3) Mark (2,3)
(1,4,4)
(1,3,2)
(1,2,2)
(1,1,2)

Program 3.15

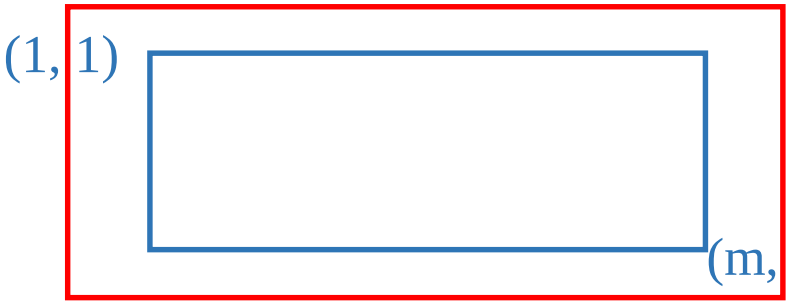
```
initialize stack to the maze entrance coordinates and direction east;
while (stack is not empty)
{
  (i, j, dir) = coordinates and direction deleted from top of stack ;
  while (there are more moves)
  {
     (g, h) = coordinates of next move;
     if ((g == m) && (h == p)) success;
     if ((!maze[g][h]) // legal move
       && (!mark[g][h]) // haven't been here
  before
   {
       mark[g][h] = 1 ;
       dir = next direction to try;
       add (i, j, dir) to top of stack;
       i = g; j = h; dir = north;
   }
  }
}
cout << "not path found" << endl ;
```

Pseudo code

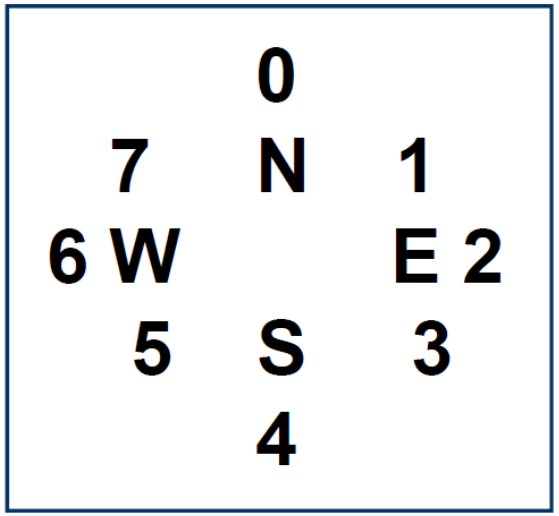
Program as the assignment 1

Program 3.16

```
void path (int m, int p)
/* Output a path (if any) in the maze;
maze[0][i]=maze[m+1][i]= maze[j][0]=maze[j][p+1]=1, 0<=i<= p+1,  0<=j<= m+1. */
{
  //start at (1,1)
  mark[1][1] = 1;
  Stack<items> stack(m*p);
  items temp;
  temp.x = 1 ; temp.y = 1 ; temp.dir = E ; To build borders
  stack.Add(temp) ;
  while (!stack.IsEmpty()) // stack not empty
  {
    temp = *stack.Delete(temp) ; // unstack
    int i = temp.x ; int j = temp.y; int d = temp.dir;
    while (d < 8)  // move forward
    {
      int g = i=move[d].a;
      int h = j + move[d].b;
      if ((g == m) && (h == p))
      { // reached exit
```



(m+2)*(p+2), To build borders



```
          //output path  cout << stack ;
          cout<<i<<" "<<j<<endl; /* last two squares on the path */
          cout << m << " " << p << endl ;  return ;
        }
        if ((!maze[g][h]) && (!mark[g][h])) {
          // new position  mark[g][h] = 1 ;
          temp.x = i ; temp.y = j ; temp.dir = d+1 ;
          stack.Add(temp) ; // stack it
          i = g ; j = h ; d = N ; // move to (g, h)
        }
        else d++ ; // try next direction
      }
    }
    cout << "no path in maze " << endl ;
}
```

Outline

- Templates in C++
- The Stack Abstract Data Type
- The Queue Abstract Data Type
- A Mazing Problem
- Evaluation of Expressions

Evaluation of Expressions

- X = a / b - c + d * e - a * c
- a = 4, b = c = 2, d = e = 3
- Interpretation 1:
  - ((4/2)-2)+(3*3)-(4*2)=0 + 8+9=1
- Interpretation 2:
  - (4/(2-2+3))*(3-4)*2=(4/3)*(-1)*2= -2.66666…
- How to generate the machine instructions corresponding to a given expression?
  - Precedence rule + associative rule

Evaluation of Expressions (Contd.)

- Infix:
  - Each operator comes in-between the operands
  - 2+3
- Postfix
  - Each operator appears after its operands
  - 23+
- Prefix
  - Each operator appears before its operands
  - +23

Evaluation of Expressions (Contd.)

User **Computer**

Infix	Postfix
2+3*4	234*+
a*b+5	ab*5+
(1+2)*7	12+7*
a*b/c	ab*c/
(a/(b-c+d))*(e-a)*c	abc-d+/ea-*c*
a/b-c+d*e-a*c	ab/c-de*ac*-+

- Postfix & prefix: no parentheses, no precedence

Evaluation of Expressions (Contd.)

- Phase 1: Infix to postfix conversion
  - 6/2-3+4*2 → 6 2 / 3 – 4 2 * +
- Phase 2: Postfix expression evaluation
  - 6 2 / 3 – 4 2 * + → 8

Phase 2: Postfix Expression Evaluation

- 6 2 / 3 – 4 2 * +

Token	Stack [0]	[1]	[2]	Top
6	6			0
2	6	2		1
/	3			0
3	3	3		1
–	0			0
4	0	4		1
2	0	4	2	2
*	0	8		1
+	8			0

Program 3.18

```
void eval(expression e)
/* Evaluate the postfix expression e. It is assumed  that the last token ( a token is either an
operator,  operand, or '#') in e is '#'. A function NextToken is used to get the next token
from e. The function  uses the stack stack */
{
   Stack<token> stack ; //initialize stack
   for(token x=NextToken(e);x!='#';x=NextToken(e))
      if(x is an operand) stack.Add(x) // add to stack
         else
        { //operator
          remove the correct number of operands for operator x from stack ;
          perform the operation x and store the result (if any) onto the stack ;
        }
} // end of eval
```

Phase 1: Infix to Postfix Conversion

- Assumptions:
  - operators: +, -, *, /, %
  - operands: single digit integer

Phase 1: Infix to Postfix Conversion (Contd.)

- Intuitive Algorithm

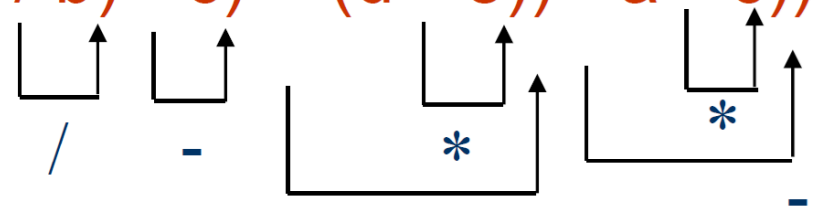
1) Fully parenthesize expression

a / b - c + d * e - a * c -->

((((a / b) - c) + (d * e)) – (a * c))

2) All operators replace their corresponding right parentheses.

((((a / b) - c) + (d * e)) - a * c))



3) Delete all parentheses. +

ab/c-de*+ac*-

- Two passes

The orders of operands in infix and postfix are the same

a + b * c, * > +

Token	Stack [0] [1] [2]	Top	Output
a		-1	a
+	+	0	a
b	+	0	ab
*	+ *	1	ab
c	+ *	1	abc
<eos>		-1	abc*+

The orders of operands in infix and postfix are the same

a * b + c, * > +

Token	Stack [0] [1] [2]	Top	Output
a		-1	a
*	*	0	a
b	*	0	ab
+	+	1	ab*
c	+	1	ab*c
<eos>		-1	ab*c+

$a *_1 (b + c) *_2 d$

Token	Stack [0] [1] [2]	Top	Output
a		-1	a
$*_1$	$*_1$	0	a
(	$*_1$ (	1	a
b	$*_1$ (	1	ab
+	$*_1$ (+	2	ab
c	$*_1$ (+	2	abc
)	$*_1$ match (	0	abc+
$*_2$	$*_2$ $*_1 = *_2$	0	abc+$*_1$
d	$*_2$	0	abc+$*_1$d
<eos>		-1	abc+$*_1$d$*_2$

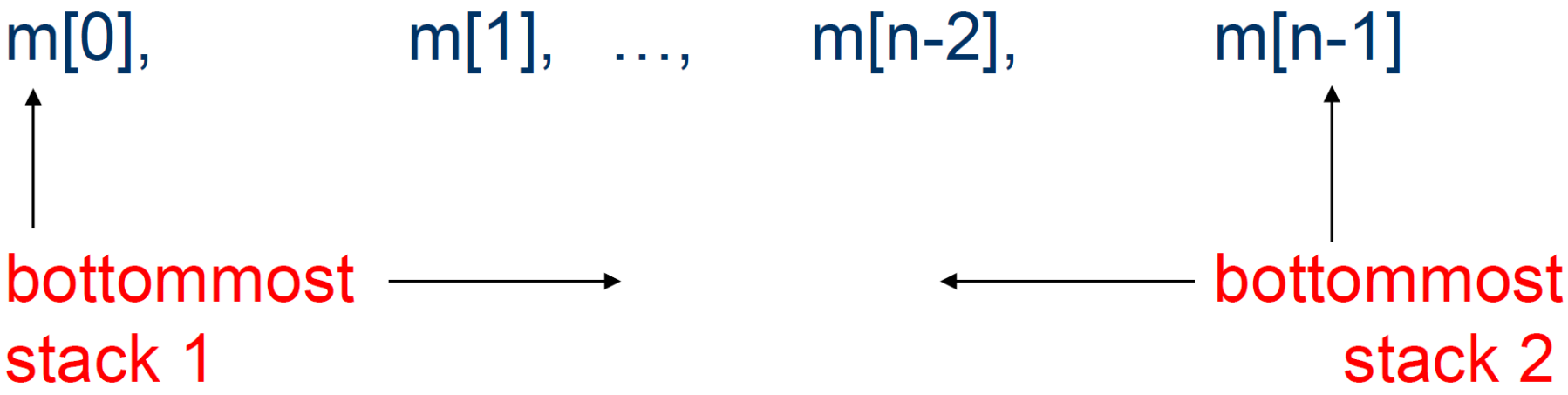
Rules

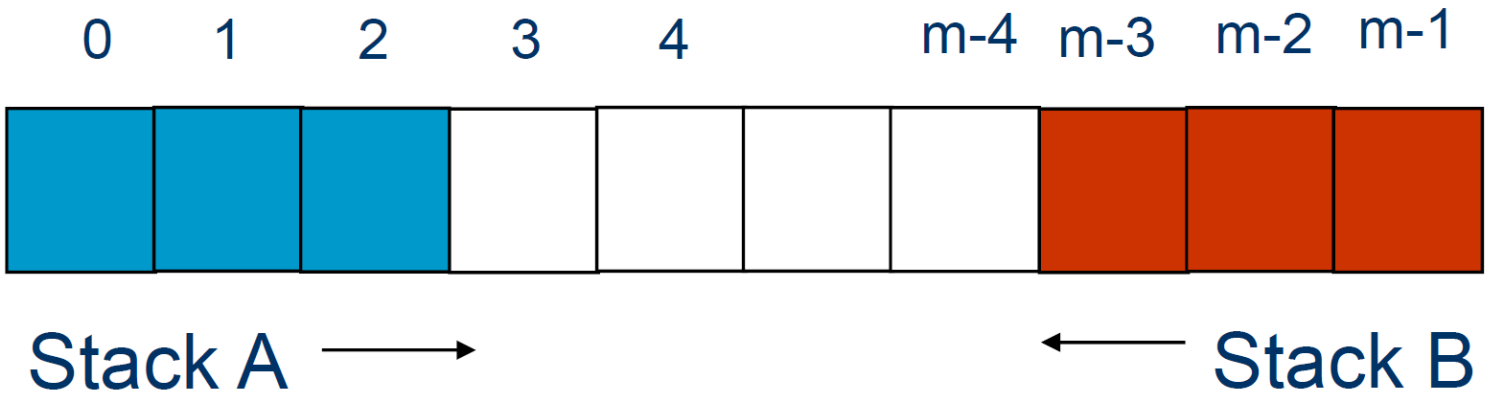
- Operators are taken out of the stack as long as their in-stack precedence is numerically less than or equal to the incoming precedence of the new operator, i.e., isp(y)<=icp(x)
- "(" has lowest in-stack precedence (i.e. 8), and highest incoming precedence (i.e. 0).
  - No operator other than the matching right parenthesis ")" should cause it to get unstacked

Priority	Operator
1	Unary minus, !
2	*,/,%
3	+,-
4	<,<=,>=,>
5	==,!=
6	&&
7	||

Multiple Stacks and Queues

- Two stacks





Multiple Stacks and Queues (Contd.)

- More than two stacks (n)
- Memory is divided into n segments
  - The initial division of these segments may be done in proportion to expected sizes of these stacks if these are known
  - All stacks are empty and divided into roughly equal segments

Multiple Stacks and Queues (Contd.)

- boundary[stack_no]
  - $0 \leq$ stack_no < MAX_STACKS
- top[stack_no]
  - $0 \leq$ stack_no < MAX_STACKS