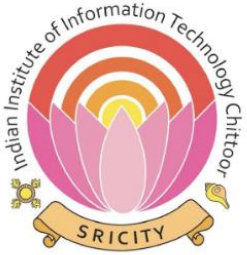


Advanced Data Structures and Algorithms

Dr. Sreeja S R

Assistant Professor

Indian Institute of Information Technology

IIIT Sri City

Stack & Queue

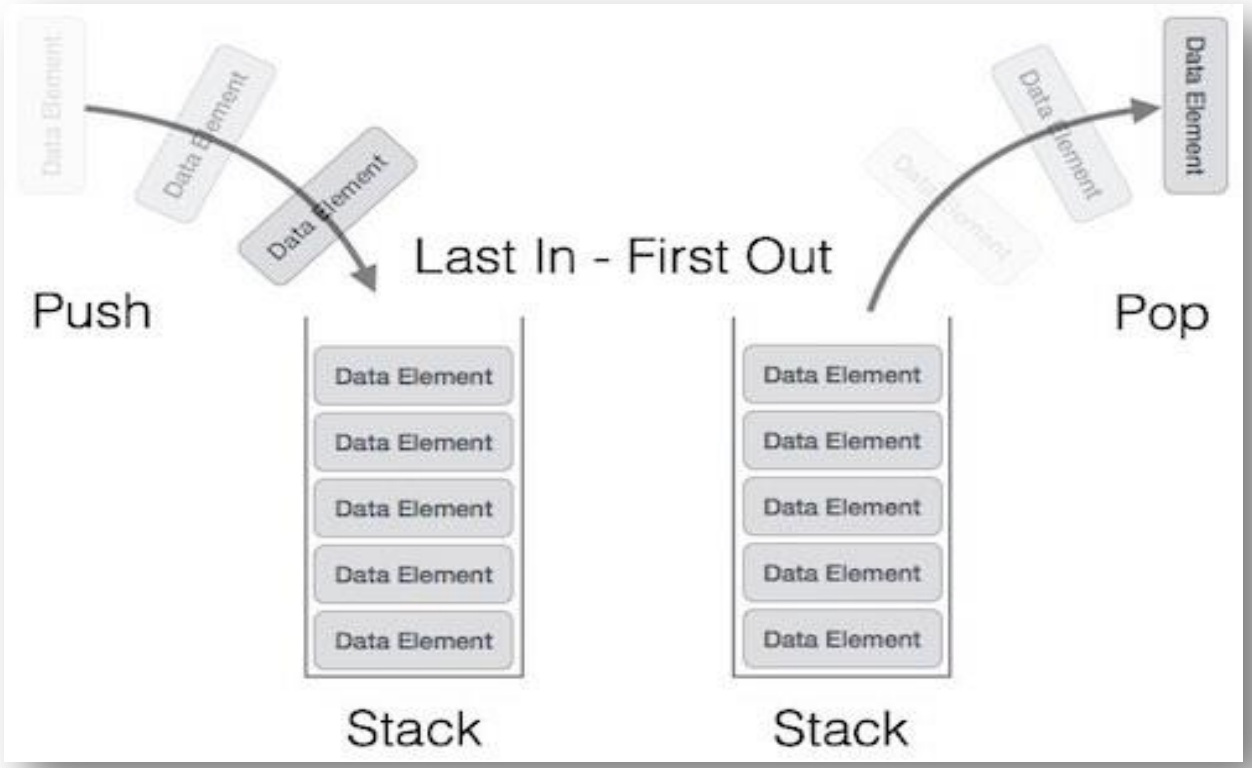
Source: Thanks to Dr Debasis Samanta, IIT Kharagpur

Stack

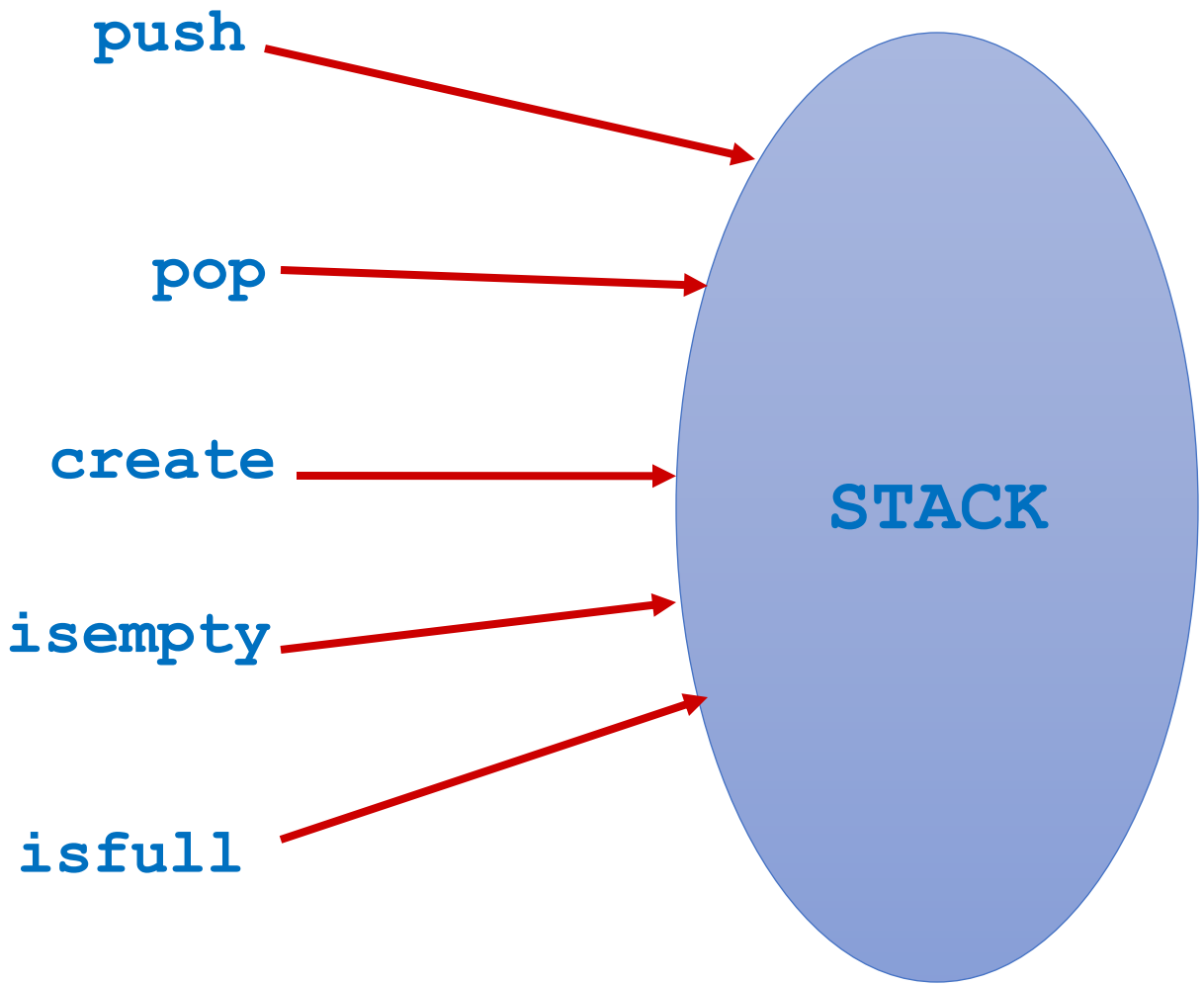
Basic Idea

- A stack is an Abstract Data Type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc.

Stack Representation



- Can be implemented by means of Array, Structure, Pointers and Linked List.
- Stack can either be a fixed size or dynamic.


push
pop
create
isempty
isfull
STACK

STACK: Last-In-First-Out (LIFO)

- `void push (stack *s, int element);`

  /* Insert an element in the stack */
- `int pop (stack *s);`

  /* Remove and return the top element */
- `void create (stack  *s);`

  /* Create a new stack */
- `int isempty (stack *s);`

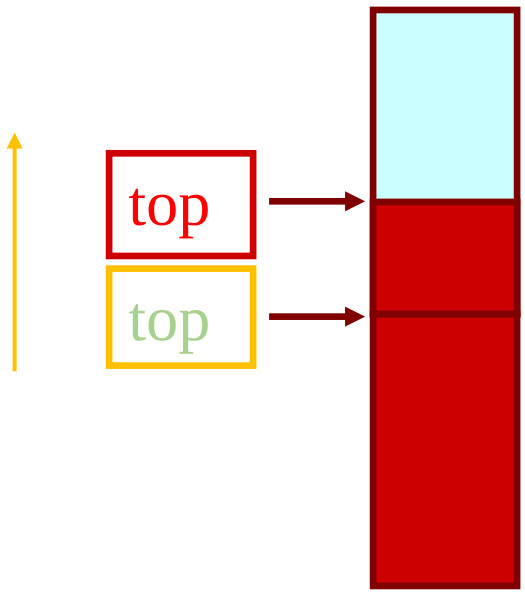
  /* Check if stack is empty */
- `int isfull (stack *s);`

  /* Check if stack is full */

Assumption: stack contains integer elements!

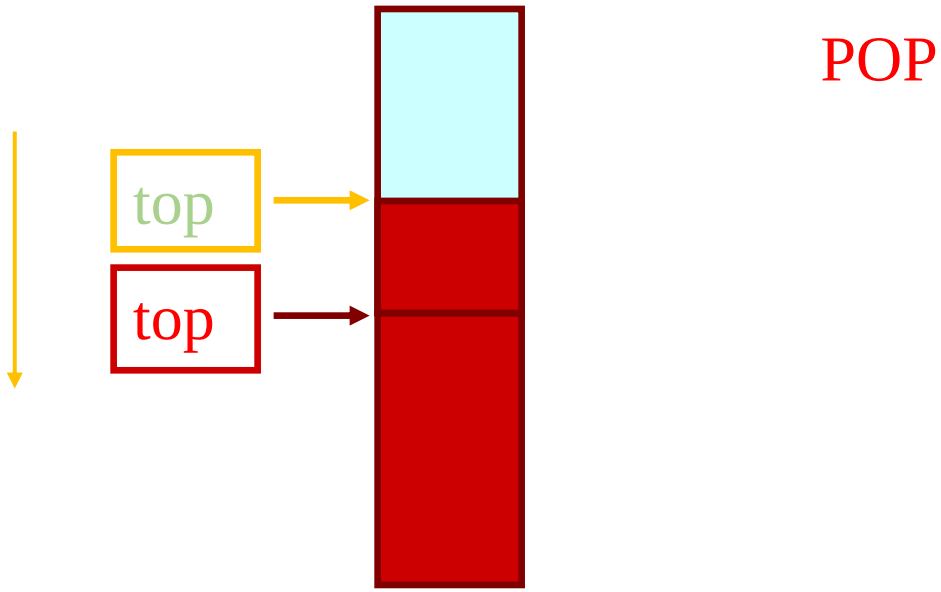
Stack using Array

Push using Stack



PUSH

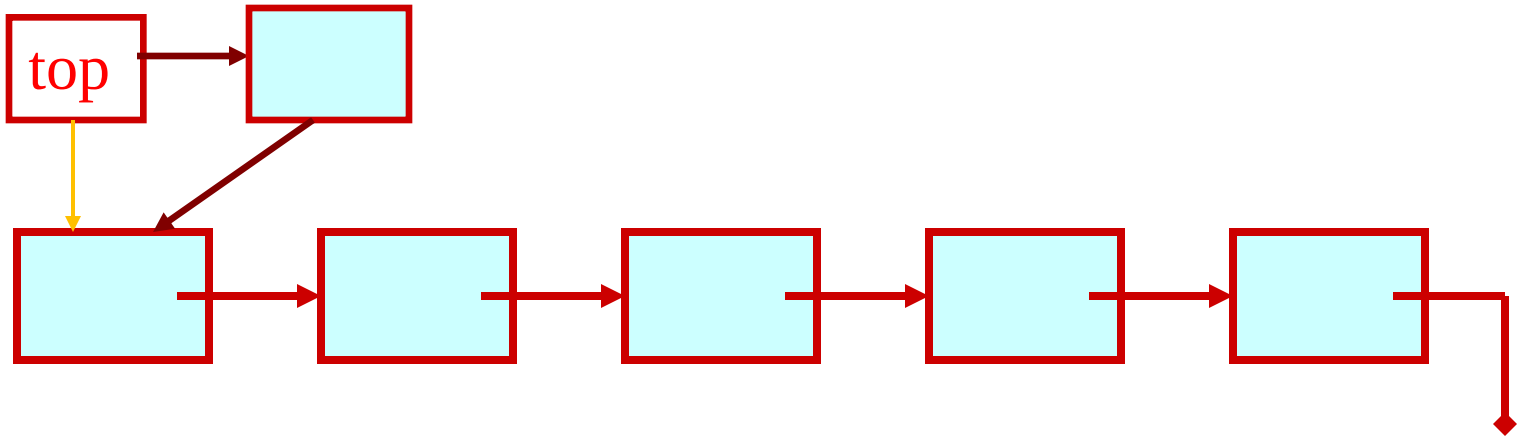
Pop using Stack



Stack using Linked List

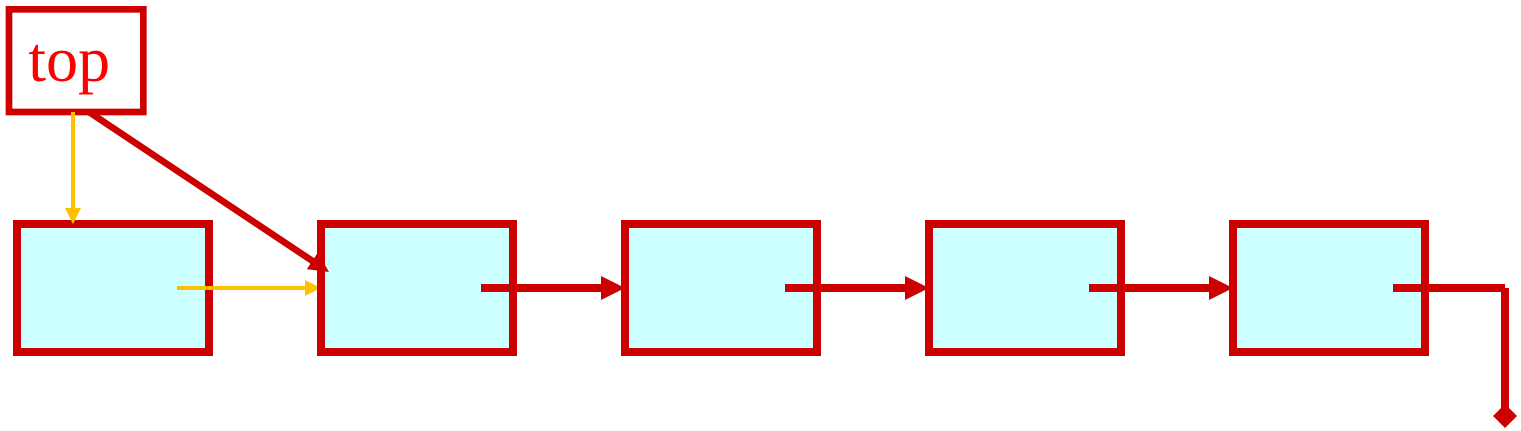
Push using Linked List

PUSH OPERATION



Pop using Linked List

POP OPERATION



Basic Idea

- In the array implementation, we would:
  - Declare an array of fixed size (which determines the maximum size of the stack).
  - Keep a variable which always points to the “top” of the stack.
    - Contains the array index of the “top” element.
- In the linked list implementation, we would:
  - Maintain the stack as a linked list.
  - A pointer variable top points to the start of the list.
  - The first element of the linked list is considered as the stack top.

Declaration

```
#define MAXSIZE 100

struct lifo
{
   int st[MAXSIZE];
   int  top;
};
typedef struct lifo
                  stack;
stack s;
```

ARRAY

```
struct lifo
{
   int value;
   struct lifo *next;
};
typedef struct lifo
                   stack;

stack *top;
```

LINKED LIST

Stack Creation

```
void create (stack *s)
{
   s->top = -1;

   /* s->top points to
      last element
      pushed in;
      initially -1 */
}
```

ARRAY

```
void create (stack **top)
{
   *top = NULL;

   /* top points to NULL,
      indicating empty
      stack              */
}
```

LINKED LIST

Pushing an element into stack

```
void push (stack *s, int element)
  {
   if (s->top == (MAXSIZE-1))
   {
     printf ("\n Stack overflow");
         exit(-1);
     }
     else
     {
         s->top++;
         s->st[s->top] = element;
     }
  }
```

ARRAY

```
void push (stack **top, int element)
{
    stack *new;

    new = (stack *)malloc (sizeof(stack));
    if (new == NULL)
    {
       printf ("\n Stack is full");
       exit(-1);
    }

    new->value = element;
    new->next = *top;
    *top = new;
}
```

LINKED LIST

Popping an element from stack

```
int pop (stack *s)
  {
     if (s->top == -1)
     {
        printf ("\n Stack underflow");
        exit(-1);
     }
     else
     {
        return (s->st[s->top--]);
     }
  }
```

ARRAY

```
int pop (stack **top)
{
   int t;
   stack *p;
   if (*top == NULL)
   {
      printf ("\n Stack is empty");
      exit(-1);
   }
   else
   {
      t = (*top)->value;
      p = *top;
      *top = (*top)->next;
      free (p);
      return t;
   }
}
```

LINKED LIST

Checking for stack empty

```
int isempty (stack *s)
{
   if (s->top == -1)
          return 1;
   else
          return (0);
}
```

ARRAY

```
int isempty (stack *top)
{
   if (top == NULL)
        return (1);
    else
        return (0);
}
```

LINKED LIST

Example: A Stack using an Array

```
#include <stdio.h>
#define MAXSIZE 100

struct lifo
{
   int st[MAXSIZE];
   int  top;
};
typedef struct lifo stack;

main() {
  stack A, B;
  create(&A);
  create(&B);
  push(&A,10);
  push(&A,20);
  push(&A,30);
  push(&B,100);
  push(&B,5);

  printf ("%d %d", pop(&A), pop(&B));

  push (&A, pop(&B));

  if (isempty(&B))
    printf ("\n B is empty");
  return;
}
```

Example: A Stack using Linked List

```
#include <stdio.h>
struct lifo
{
   int value;
   struct lifo *next;
};
typedef struct lifo stack;

main() {
  stack *A, *B;
  create(&A);
  create(&B);
  push(&A,10);
  push(&A,20);
  push(&A,30);
  push(&B,100);
  push(&B,5);

  printf ("%d %d", pop(&A), pop(&B));

  push (&A, pop(&B));

  if (isempty(B))
    printf ("\n B is empty");
  return;
}
```

Applications of Stacks

- Direct applications:
  - Page-visited history in a Web browser
  - Undo sequence in a text editor
  - Chain of method calls in the Java Virtual Machine
  - Validate XML

- Indirect applications:
  - Auxiliary data structure for algorithms
  - Component of other data structures

Infix and Postfix Notations

- Infix: operators placed between operands:

  A+B*C

- Postfix: operands appear before their operators:-

  ABC*+

- There are no precedence rules to learn in postfix notation, and parentheses are never needed

Infix to Postfix

Infix	Postfix
A + B	A B +
A + B * C	A B C * +
(A + B) * C	A B + C *
A + B * C + D	A B C * + D +
(A + B) * (C + D)	A B + C D + *
A * B + C * D	A B * C D * +

A + B * C → (A + (B * C)) → (A + (B C *)) → A B C * +

A + B * C + D → ((A + (B * C)) + D) → ((A + (B C*))+ D) → ((A B C *+) + D) → A B C * + D +

Infix to postfix conversion

- Use a stack for processing operators (push and pop operations).
- Scan the sequence of operators and operands from left to right and perform one of the following:
  - output the operand,
  - push an operator of higher precedence,
  - pop an operator and output, till the stack top contains operator of a lower precedence and push the present operator.

The algorithm steps

1. Print operands as they arrive.
2. If the stack is empty or contains a left parenthesis on top, push the incoming operator onto the stack.
3. If the incoming symbol is a left parenthesis, push it on the stack.
4. If the incoming symbol is a right parenthesis, pop the stack and print the operators until you see a left parenthesis. Discard the pair of parentheses.
5. If the incoming symbol has higher precedence than the top of the stack, push it on the stack.
6. If the incoming symbol has equal precedence with the top of the stack, use association. If the association is left to right, pop and print the top of the stack and then push the incoming operator. If the association is right to left, push the incoming operator.
7. If the incoming symbol has lower precedence than the symbol on the top of the stack, pop the stack and print the top operator. Then test the incoming operator against the new top of stack.
8. At the end of the expression, pop and print all operators on the stack. (No parentheses should remain.)

Infix to Postfix Conversion

Requires operator precedence information

Operands:

Add to postfix expression.

Close parenthesis:

pop stack symbols until an open parenthesis appears.

Operators:

Pop all stack symbols until a symbol of lower precedence appears. Then push the operator.

End of input:

Pop all remaining stack symbols and add to the expression.

Infix to Postfix Rules

Expression:

A * (B + C * D) + E

becomes

A B C D * + * E +

Postfix notation is also called as Reverse Polish Notation (RPN)

	Current symbol	Operator Stack	Postfix string
1	A		A
2	*	*	A
3	(	* (	A
4	B	* (	A B
5	+	* (+	A B
6	C	* (+	A B C
7	*	* (+ *	A B C
8	D	* (+ *	A B C D
9	)	*	A B C D * +
10	+	+	A B C D * + *
11	E	+	A B C D * + * E
12			A B C D * + * E +

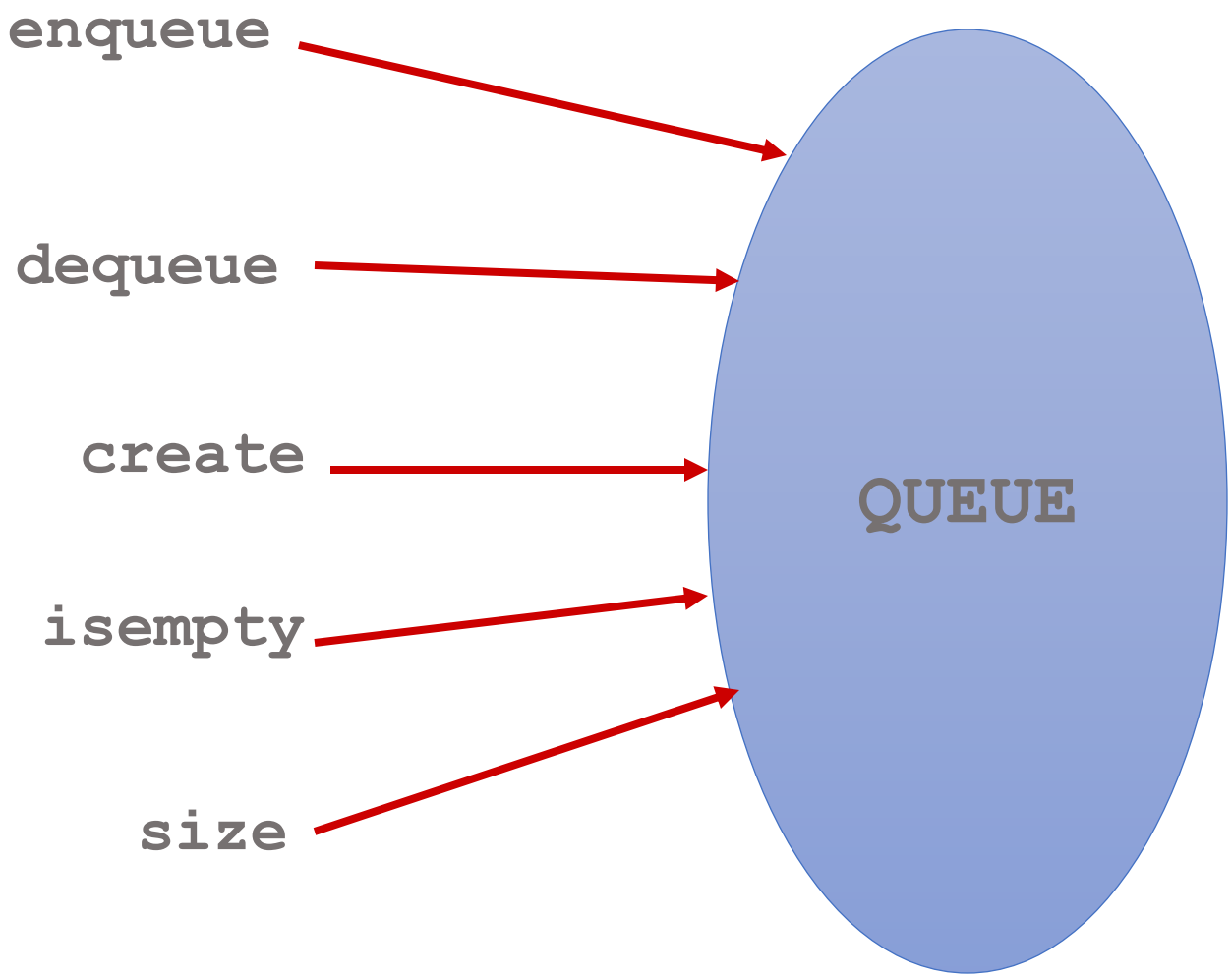
Queue

Basic Idea

- Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue).

Queue Representation

- As in stacks, a queue can also be implemented using Arrays, Linked-lists, Pointers and Structures.


enqueue
dequeue
create
isempty
size
QUEUE

QUEUE: First-In-First-Out (LIFO)

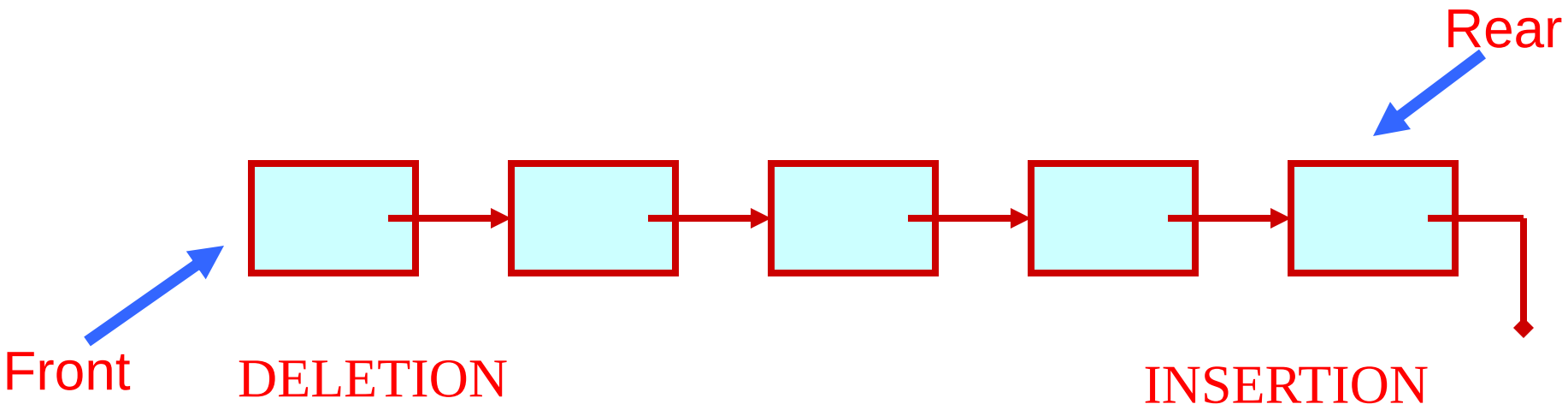
```
void enqueue (queue *q, int element);
                    /* Insert an element in the queue */
int dequeue (queue *q);
                    /* Remove an element from the queue */
queue *create();
                    /* Create a new queue */
int isempty (queue *q);
                    /* Check if queue is empty */
int size (queue *q);
                    /* Return the no. of elements in queue */
```

Assumption: queue contains integer elements!

Queue using Linked List

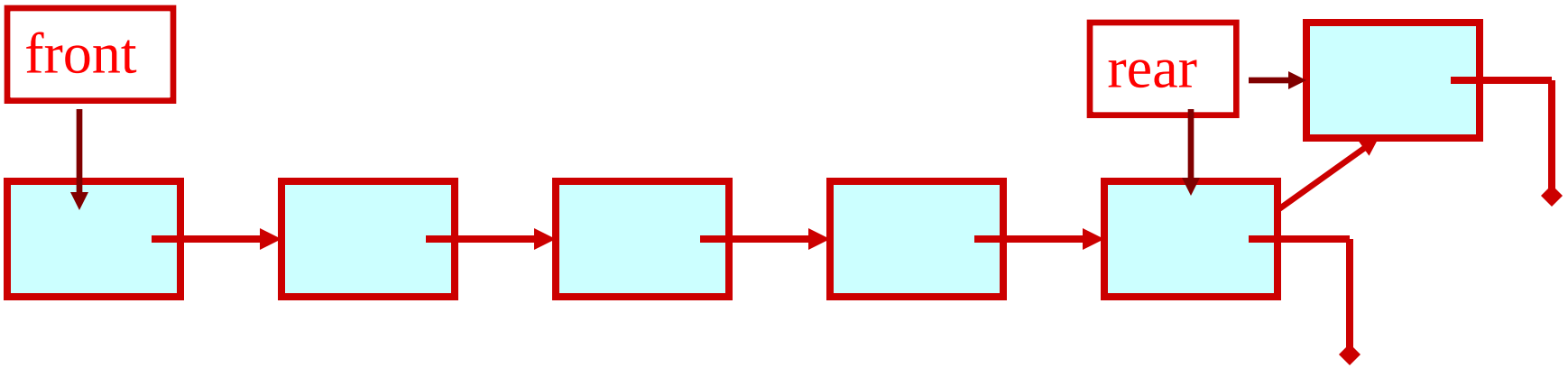
Basic Idea

- Basic idea:
  - Create a linked list to which items would be added to one end and deleted from the other end.
  - Two pointers will be maintained:
    - One pointing to the beginning of the list (point from where elements will be deleted).
    - Another pointing to the end of the list (point where new elements will be inserted).



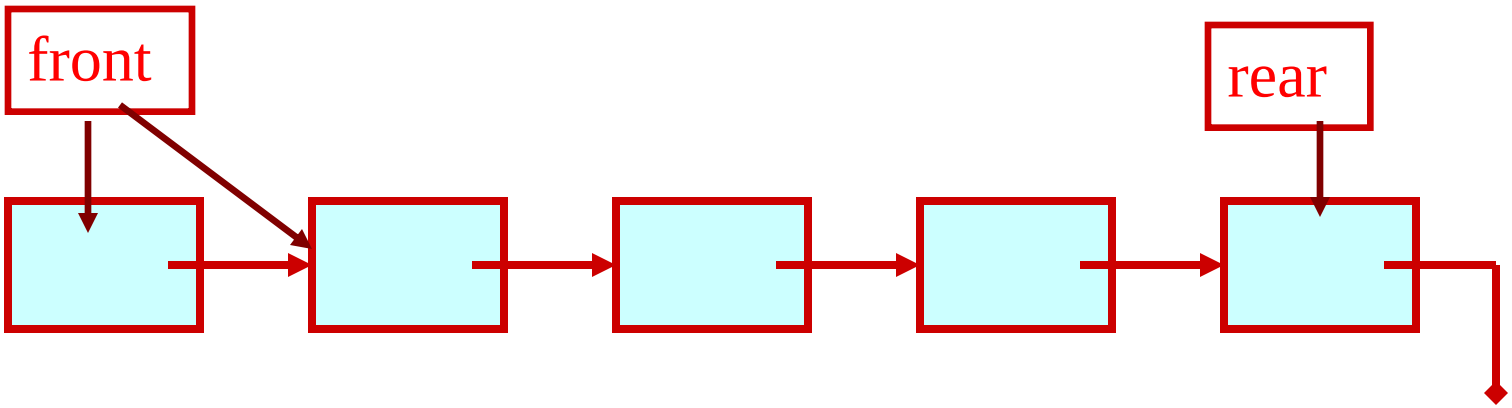
Queue: Linked List Structure

ENQUEUE



Queue: Linked List Structure

DEQUEUE



Example :Queue using Linked List

```
struct qnode
{
  int val;
   struct qnode *next;
};

struct queue
{
   struct qnode *qfront, *qrear;
};
typedef struct queue QUEUE;
```

```
void enqueue (QUEUE *q,int element)
{
   struct qnode *q1;
   q1=(struct qnode *)malloc(sizeof(struct qnode));
   q1->val= element;
   q1->next=q->qfront;
   q->qfront=q1;
}
```

Example :Queue using Linked List

```
int size (queue *q)
{
    queue *q1;
    int count=0;
    q1=q;
    while(q1!=NULL)
    {
      q1=q1->next;
      count++;
    }
return count;
}
```

```
int peek (queue *q)
{
   queue *q1;
   q1=q;
   while(q1->next!=NULL)
      q1=q1->next;
return (q1->val);
}
```

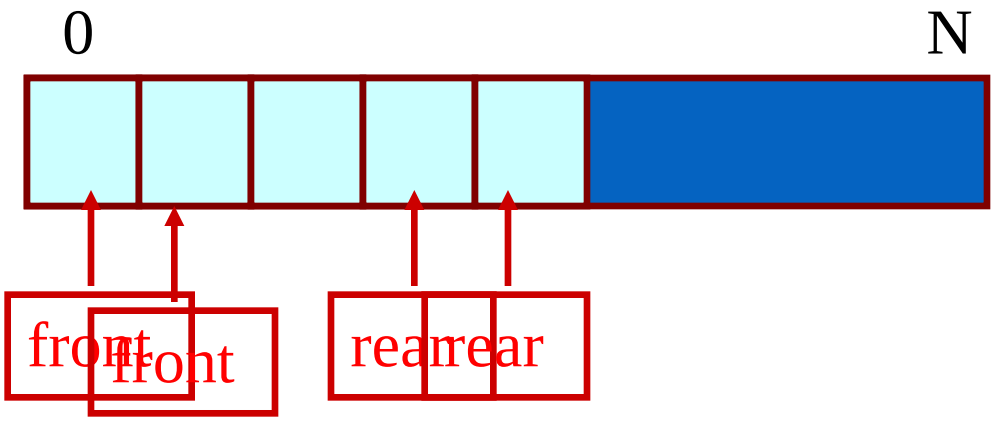
```
int dequeue (queue *q)
{
   int val;
   queue *q1,*prev;
   q1=q;
   while(q1->next!=NULL)
   {
      prev=q1;
      q1=q1->next;
   }
   val=q1->val;
   prev->next=NULL;
   free(q1);
return (val);
}
```

Problem With Array Implementation

- The size of the queue depends on the number and order of enqueue and dequeue.
- It may be situation where memory is available but enqueue is not possible.

ENQUEUE DEQUEUE

Effective queuing storage area of array gets reduced.



Use of circular array indexing

Applications of Queues

- Direct applications:-
  - Waiting lists
  - Access to shared resources (e.g., printer)
  - Multiprogramming

- Indirect applications:-
  - Auxiliary data structure for algorithms
  - Component of other data structures

Any question?