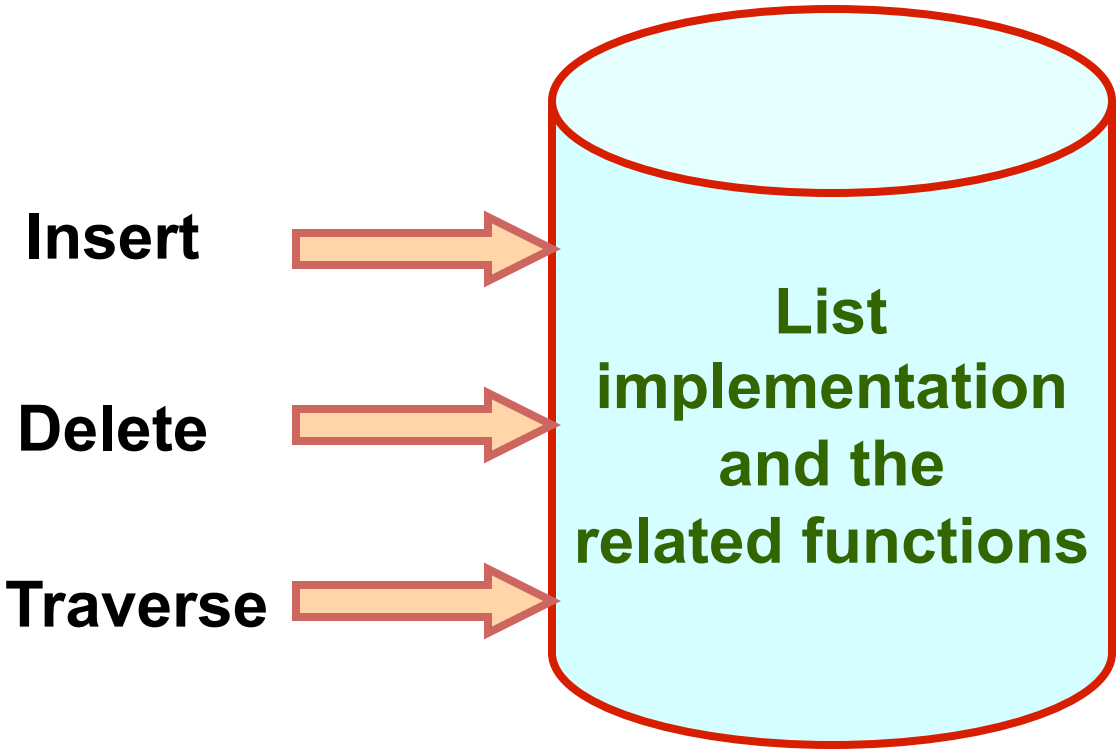
CS11001/CS11002
Programming and Data Structures (PDS) (Theory: 3-0-0)

Teacher: Sourangshu Bhattacharya
sourangshu@gmail.com
http://cse.iitkgp.ac.in/~sourangshu/

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

Conceptual Idea



Abstract Data Types (ADT)

List is an Abstract Data Type

- A class of objects whose logical behavior is defined by a set of values and a set of operations.

- What is an abstract data type (ADT)?
  - It is a data type defined by the user.
  - It is defined by it's behavior (semantics)
  - Typically more complex than simple data types like *int*, *float*, etc.

- Why abstract?
  - Because details of the implementation are hidden.
  - When you do some operation on the list, say insert an element, you just call a function.
  - Details of how the list is implemented or how the insert function is written is no longer required.

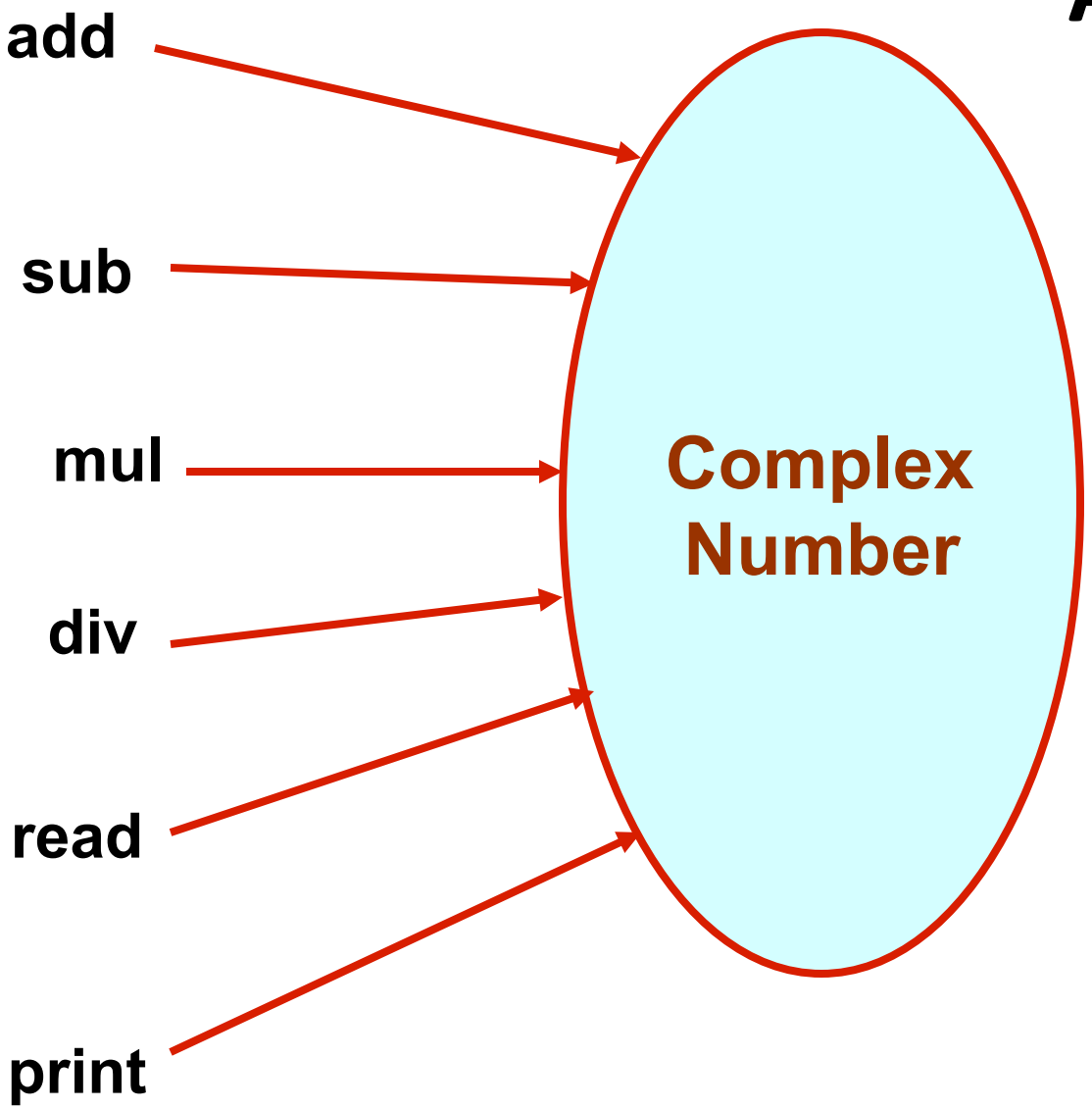
Example 1 :: Complex numbers

```
struct cplx {
              float  re;
              float  im;
            }
typedef struct cplx complex;

complex *add (complex a, complex b);
complex *sub (complex a, complex b);
complex *mul (complex a, complex b);
complex *div (complex a, complex b);
complex *read();
void print (complex a);
```

Structure definition

Function prototypes

ADT



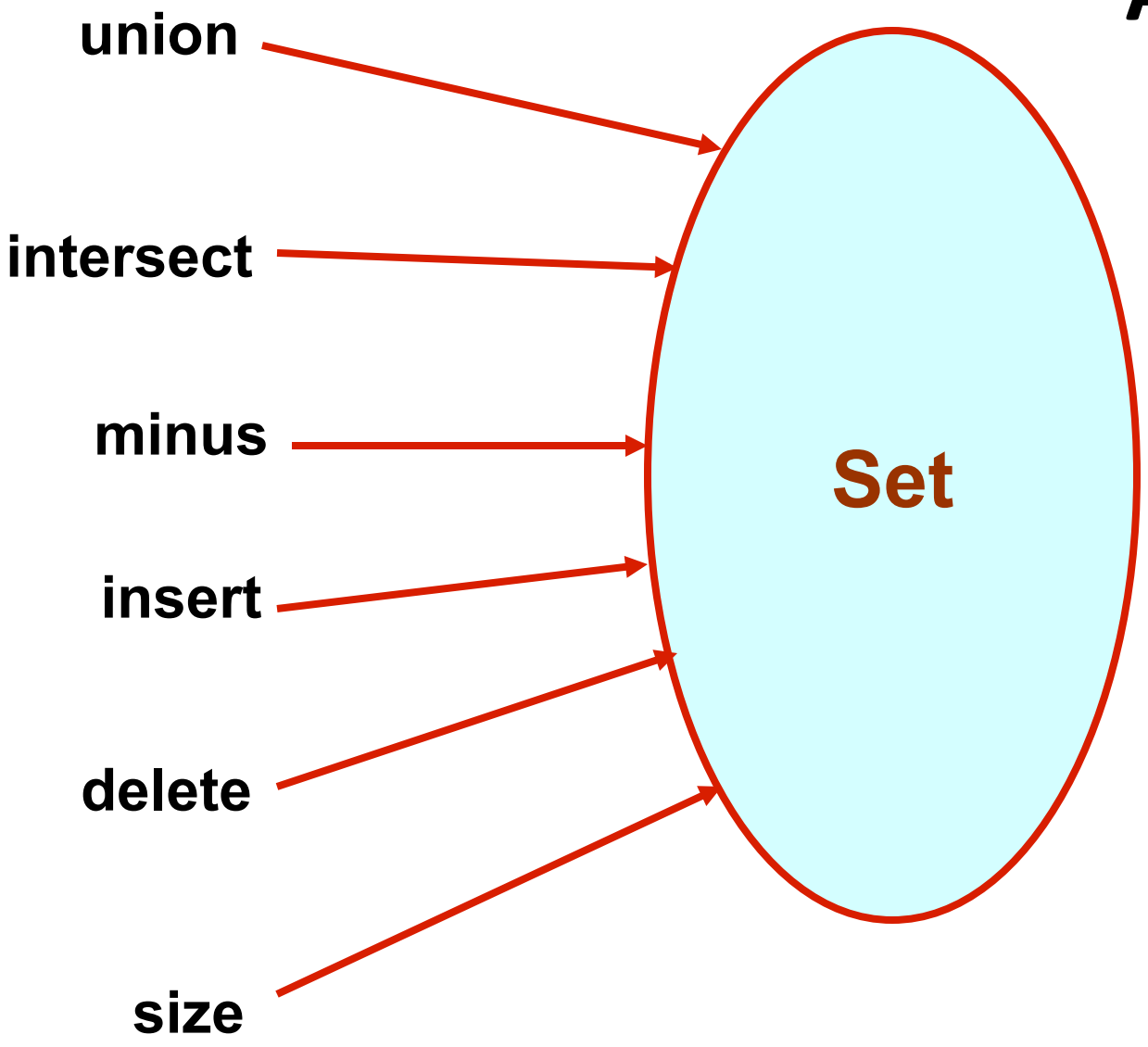
Example 2 :: Set manipulation

```
struct node {
              int element;
              struct node *next;
            }
typedef struct node set;

set *union (set a, set b);

set *intersect (set a, set b);

set *minus (set a, set b);

void insert (set a, int x);

void delete (set a, int x);

int size (set a);
```

Structure definition

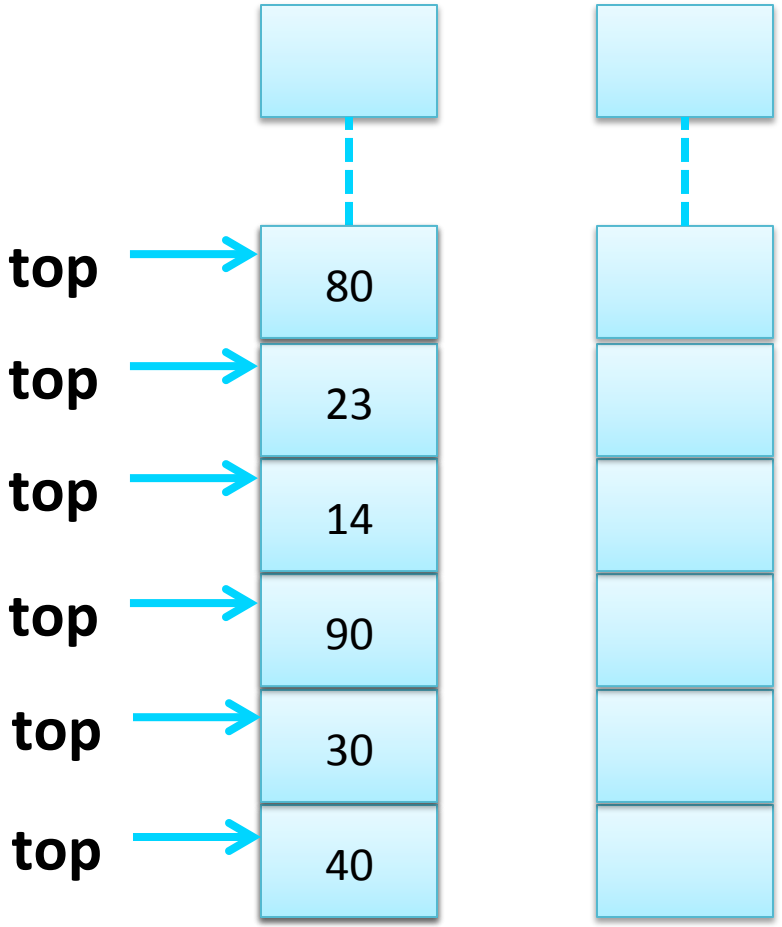
Function prototypes

ADT



STACK: Last-in-first-out (LIFO)

STACK USING ARRAY



PUSH

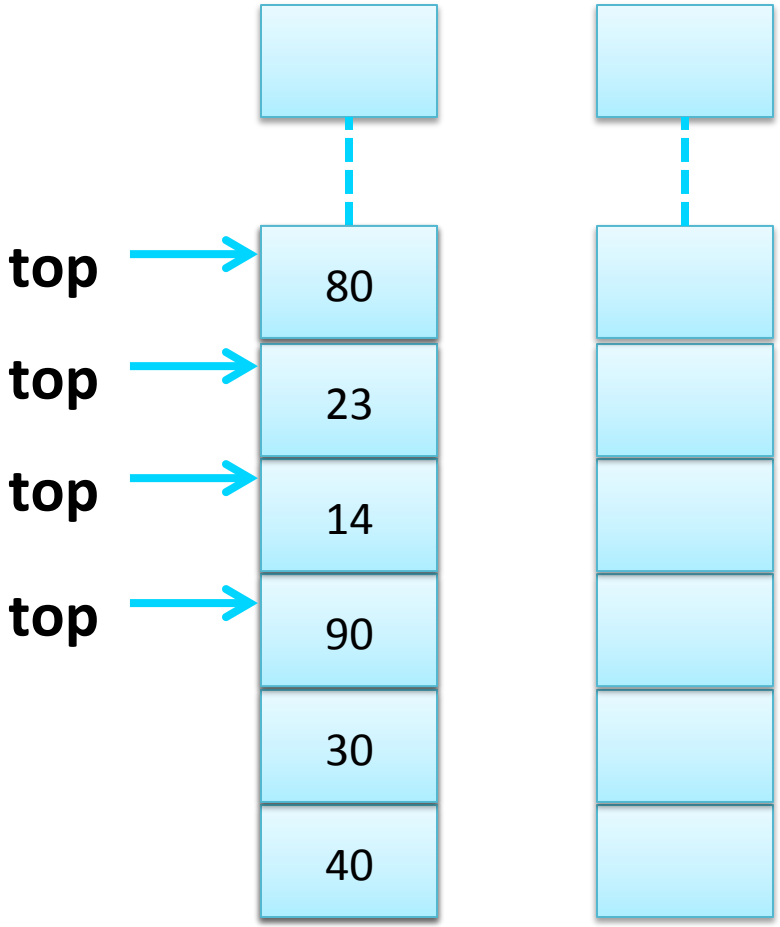
Increment top (array index)

```
#define MAXSIZE 100

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

What do we need?

1. An array to store the elements (of maximum size).
2. An integer variable (act as array index) to indicate the stack top.

STACK USING ARRAY



POP

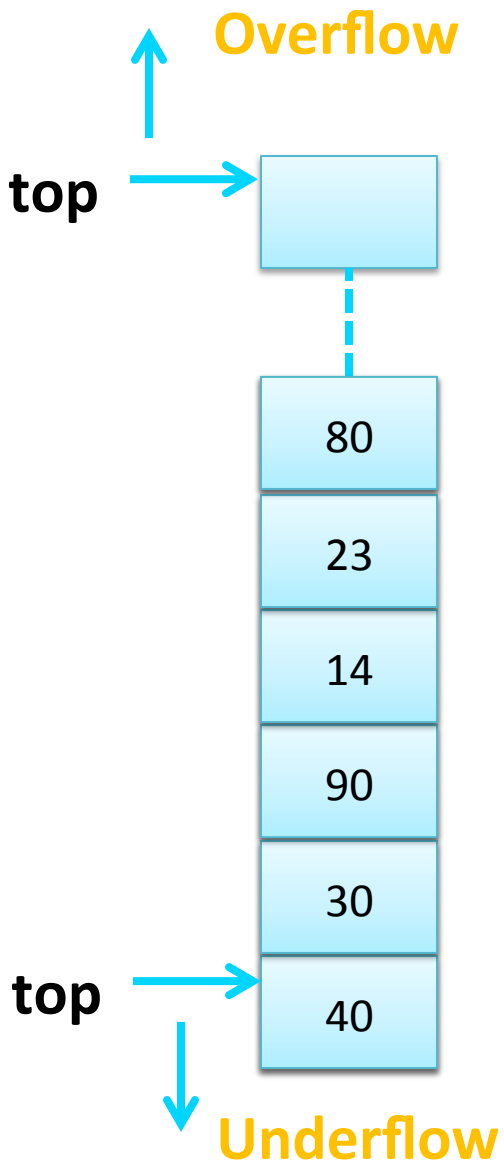
Decrement top (array index)

```
#define MAXSIZE 100

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

STACK using array

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

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

int 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 0;
}
```

STACK: Overflow and Underflow




```
#define MAXSIZE 100

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

Push (increment top) when stack top is at MAXSIZE

Overflow

Pop (decrement top) when stack top is at zero index.

Underflow

STACK: isEmpty() and isFull()

```
#define MAXSIZE 100

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

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

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

STACK: push() and pop()

```
#define MAXSIZE 100

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

```
int push (stack *s, int x)
{
    if(isFull(s))
        return 1;
    else {
        s->top++;
        s->st[s->top]=x;
        return 0;
    }
}
```

```
int pop (stack *s)
{
    if(isEmpty(s))
        return -99999;
    else {
        x=s->top;
        s->top--;
        return x;
    }
}
```

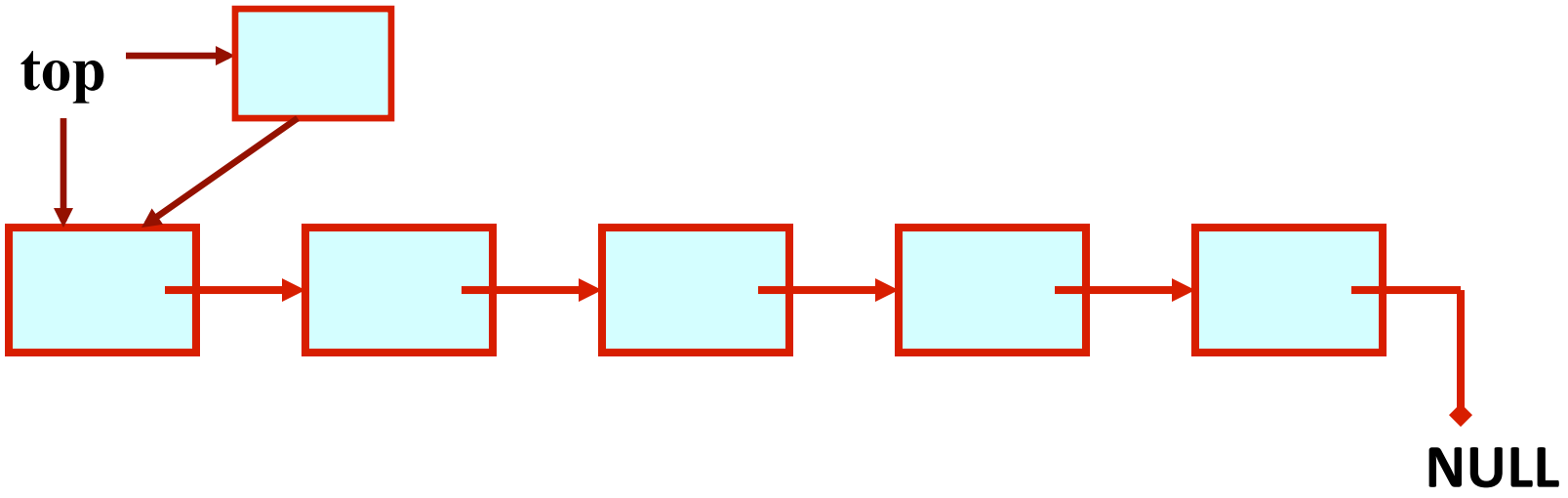
Stack Creation

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

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

Stack: Linked List Structure

PUSH

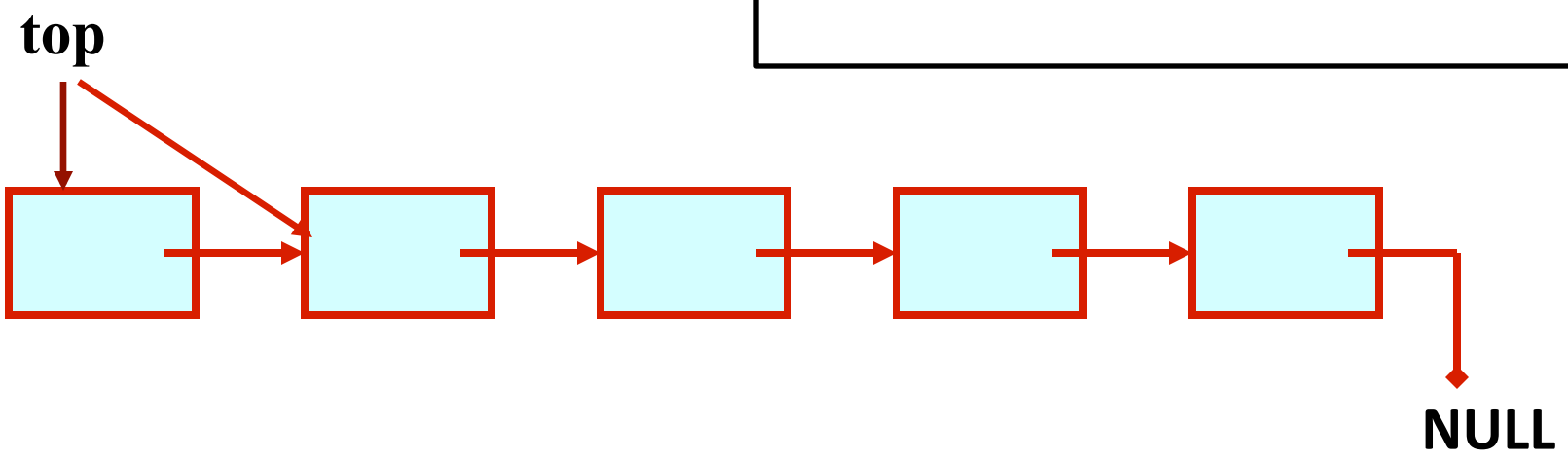
```
struct stack
{
    int value;
    struct stack *next;
};
typedef struct stack STACK;

STACK *top;
```



Stack: Linked List Structure

POP

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

STACK *top;
```



Declaration

```
#define MAXSIZE 100
struct stack
{
   int st[MAXSIZE];
   int  top;
};
typedef struct stack STACK;
STACK s;
```

ARRAY

```
struct stack
{
   int value;
   struct stack *next;
};
typedef struct stack STACK;
STACK *top;
```

LINKED LIST

STACK: push()

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

    new = (stack *) malloc(sizeof(stack));
    if (new == NULL)
    {
       printf ("\n Memory allocation problem.");
       exit(-1);
    }

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

LINKED LIST

STACK: pop()

```
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

STACK: imEmpty()

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

LINKED LIST

isFull() …?

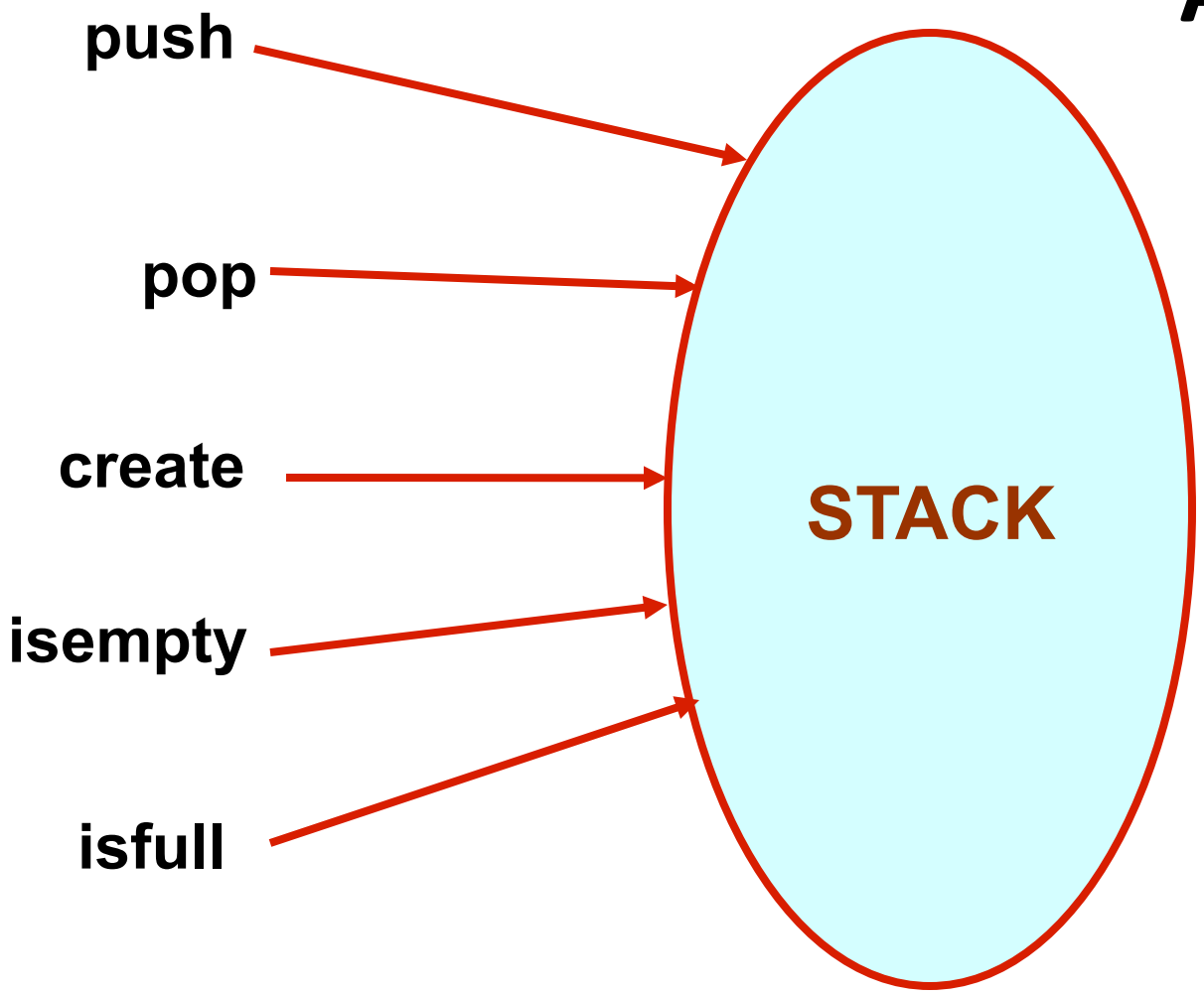
There is underflow. But, there is no overflow (assuming memory is available for dynamic allocation).

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

Assume:: stack contains integer elements

```
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 */
```

ADT



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 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 +

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 +

Infix to Postfix Rules

```
stack s
char  ch, element

while(tokens are available)   {
    ch = read(token);
    if(ch is operand)   {
      print ch ;
    } else  {
      while(priority(ch) <= priority(top most stack))  {
                  element = pop(s);
                  print(element);
            }
      push(s,ch);
     }
}
while(!empty(s))  {
      element = pop(s);
      print(element);
}
```

Homework

Implement Infix to Postfix conversion program in C using stack. You may use array or linked list for your stack.

Evaluating Postfix Expression

1) Create a stack to store operands (or values).

2) Scan the given expression and do following for every scanned element.

   a) If the element is a number, push it into the stack

   b) If the element is a operator, pop operands for the operator from stack. Evaluate the operator and push the result back to the stack

3) When the expression is ended, the number in the stack is the final answer

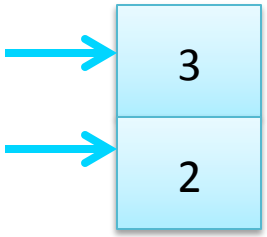
Evaluating Postfix Expression

Infix Expression: 2 * 3 – 4 / 5

Postfix Expression: 2 3 * 4 5 / -

2 3 * 4 5 / -

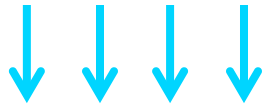
Evaluate Expression



Evaluating Postfix Expression

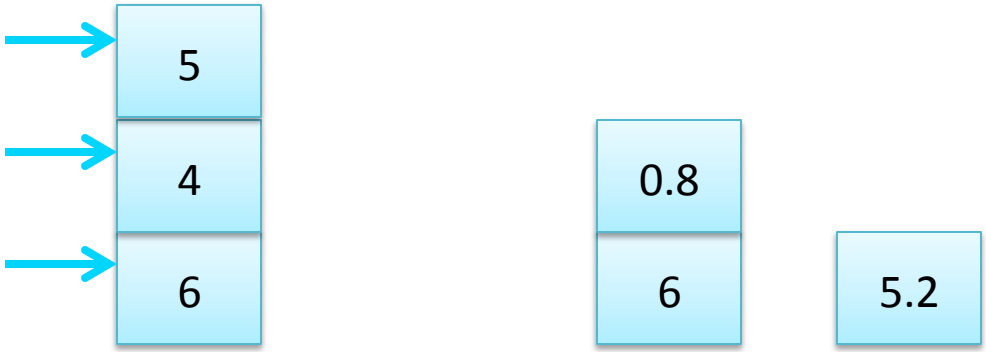
Infix Expression: 2 * 3 – 4 / 5

Postfix Expression: 2 3 * 4 5 / -



2 3 * 4 5 / -

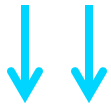
Evaluate Expression



Evaluating Postfix Expression

Infix Expression: 2 * 3 – 4 / 5

Postfix Expression: 2 3 * 4 5 / -



2 3 * 4 5 / -

Evaluate Expression

Evaluated Expression
(Stack top element) = 5.2

Homework

Write a C program to evaluate postfix expression using stack. You may use array or linked list for your stack.

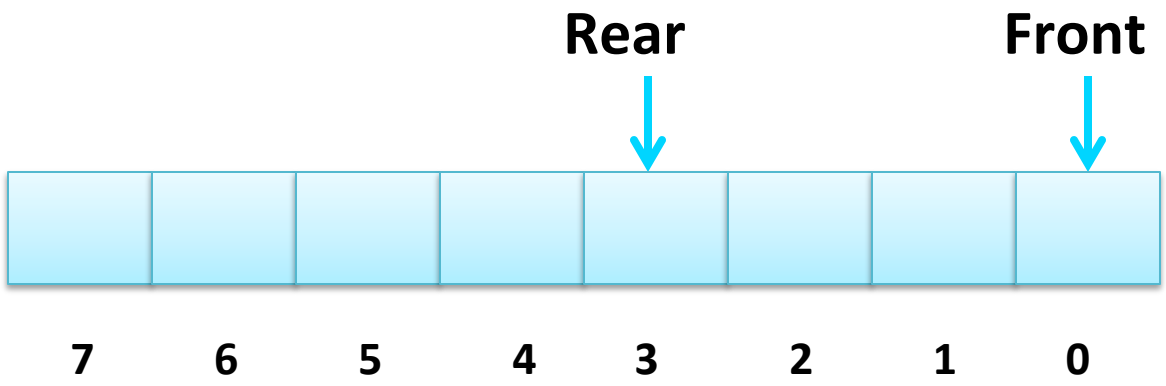
QUEUE: First-in-first-out (FIFO)

QUEUE USING ARRAY

What do we need?

1. An array to store the elements (of maximum size).
2. Two integer variables (act as array index) to indicate front and rear.

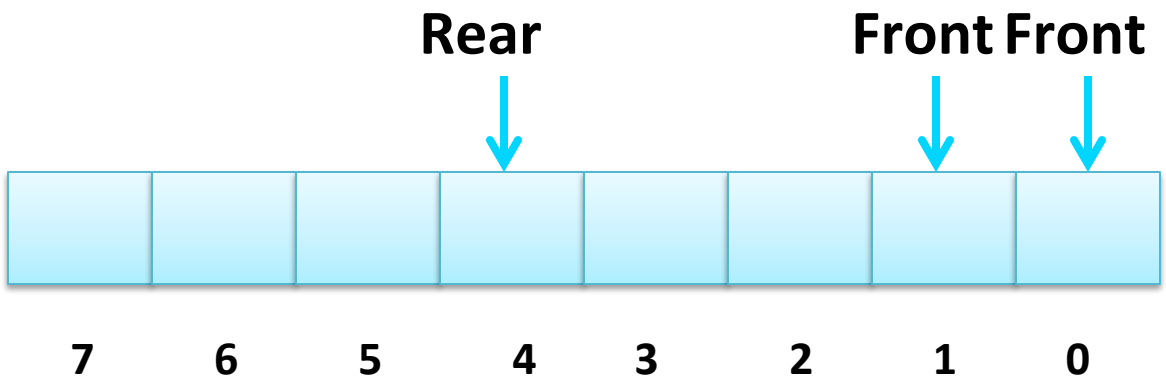
```
#define MAXSIZE 100
struct queue
{
    int que[MAXSIZE];
    int front,rear;
};
typedef struct queue QUEUE;
```



ENQUEUE

Increment front (array index)

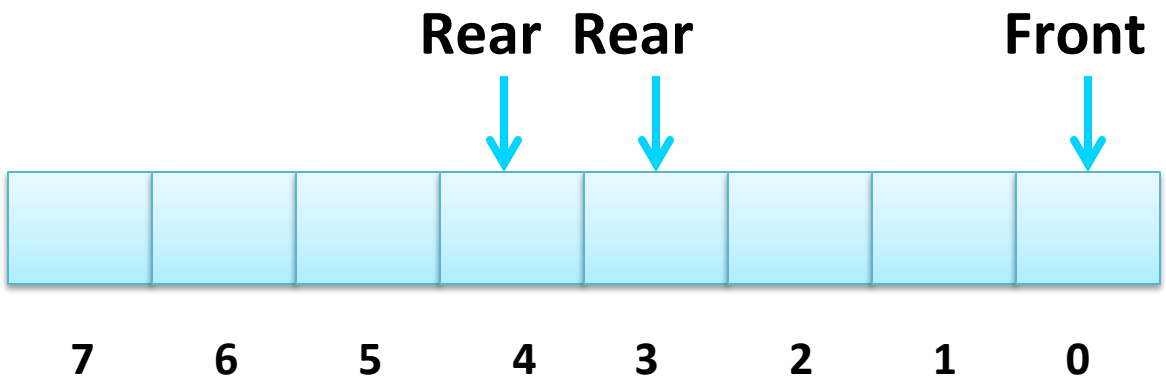
```
#define MAXSIZE 100
struct queue
{
    int que[MAXSIZE];
    int front,rear;
};
typedef struct queue QUEUE;
```



DEQUEUE

Increment rear (array index)

```
#define MAXSIZE 100
struct queue
{
    int que[MAXSIZE];
    int front,rear;
};
typedef struct queue QUEUE;
```

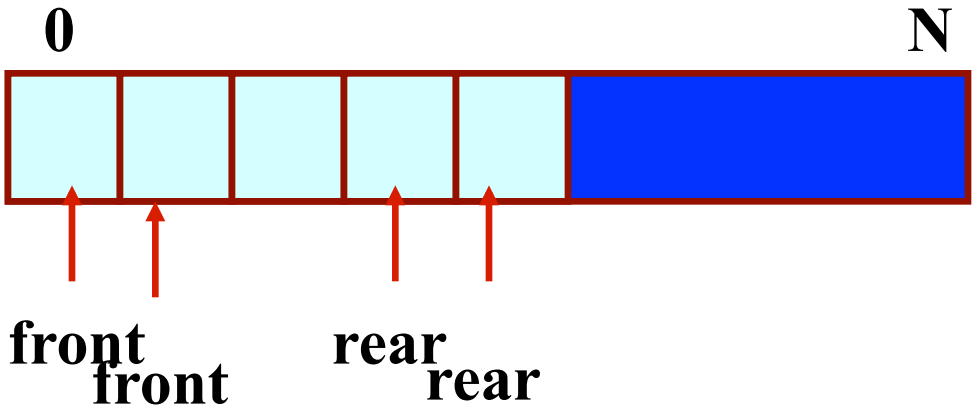


Problem With Array Implementation

- 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**

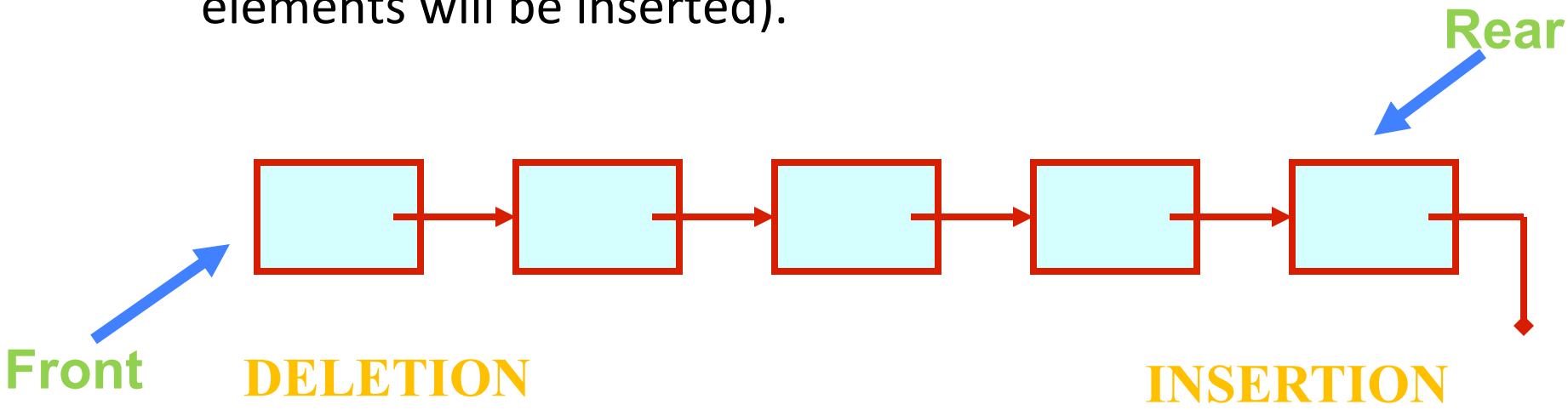
Effective queuing storage area of array gets reduced.



Use of circular array indexing

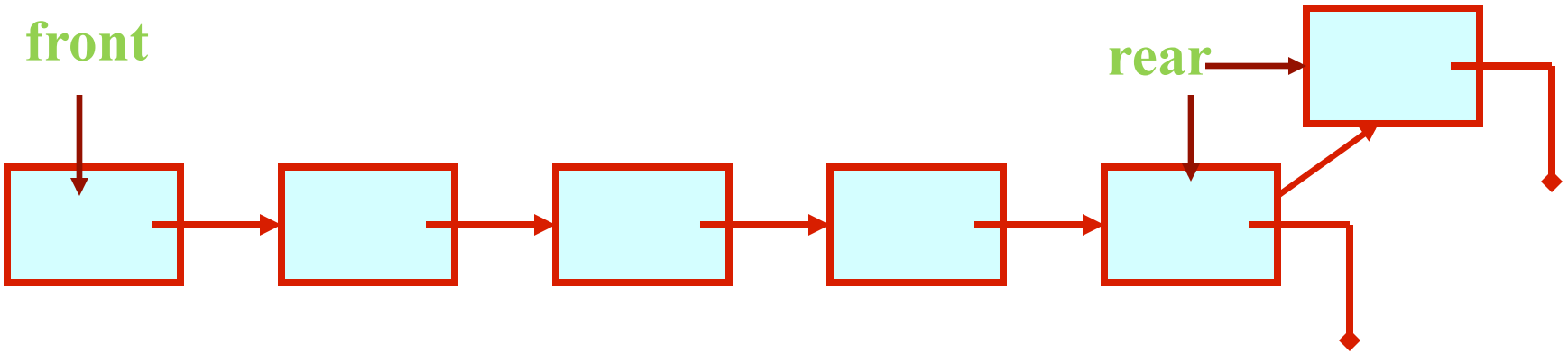
QUEUE USING LINKED LIST

- 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).



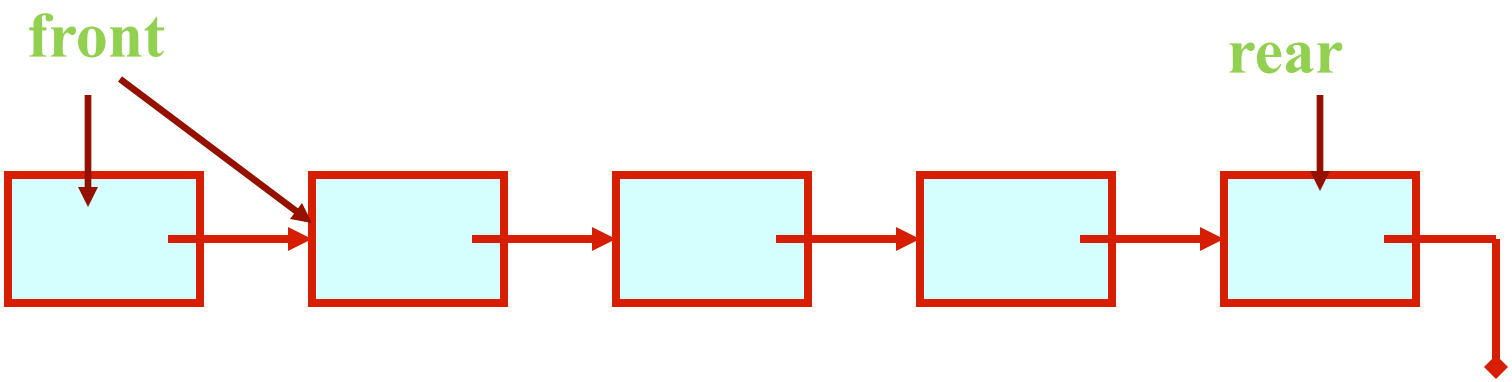
QUEUE: Insertion into a Linked List

ENQUEUE



QUEUE: Deletion from a Linked List

DEQUEUE

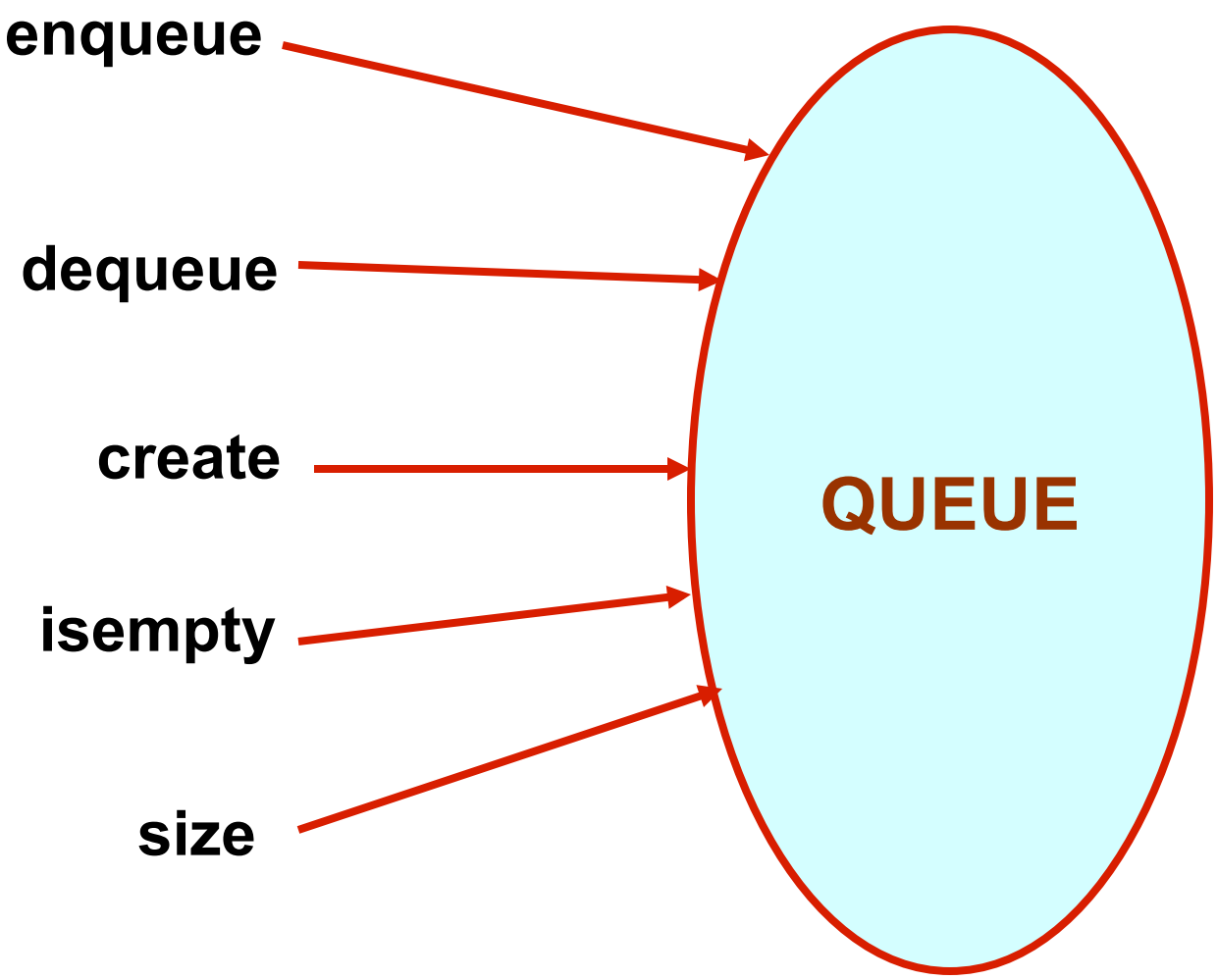


QUEUE:: First-In-First-Out (FIFO)

Assume:: queue contains integer elements

```
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 */
int peek (QUEUE *q);
                    /* dequeue without removing element*/
```

ADT



QUEUE using Linked List

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

struct queue{
    struct qnode *qfront, *qrear;
     };

typedef struct queue QUEUE;
```

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

Assume:: queue contains integer elements

```
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;
}
```

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

Assume:: queue contains integer elements

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

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

Assume:: queue contains integer elements

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

Implement this using QUEUE data structure.

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

Assume:: queue contains integer elements

```
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);
}
```

Implement this using QUEUE data structure.

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