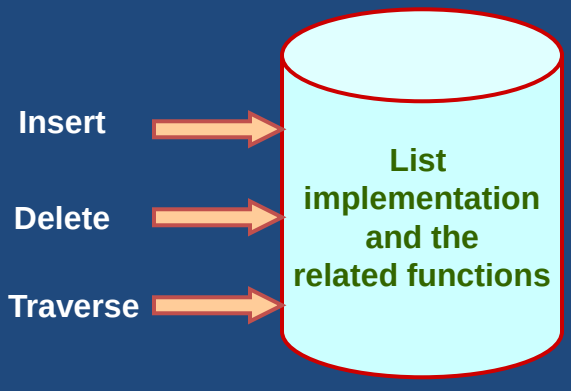
CS11001/CS11002
Programming and Data Structures
(PDS) (Theory: 3-0-0)

Class Teacher: Pralay Mitra

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

Conceptual Idea



Abstract Data Types (ADT)

List is an Abstract Data Type

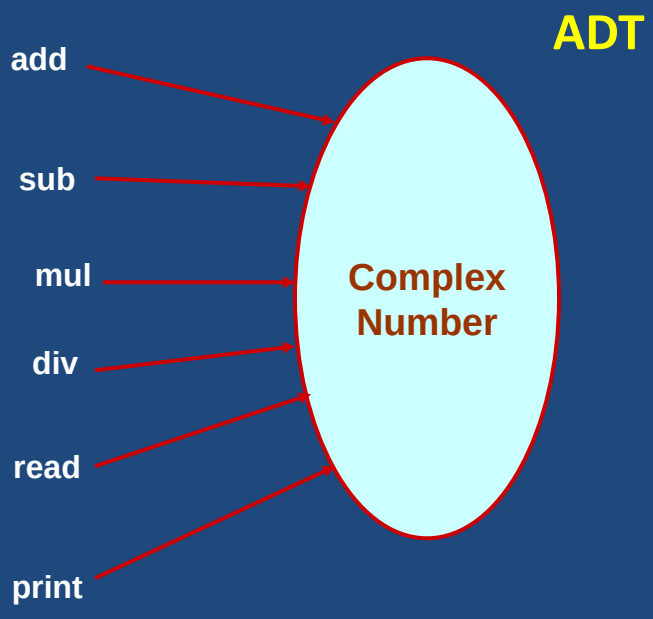
- A class of objects whose logical behavior is defined by a set of values and a set of operations.
- What is an abstract data type (ADT)?
  - It is a data type defined by the user.
  - It is defined by it's behavior (semantics)
  - Typically more complex than simple data types like *int*, *float*, etc.
- 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



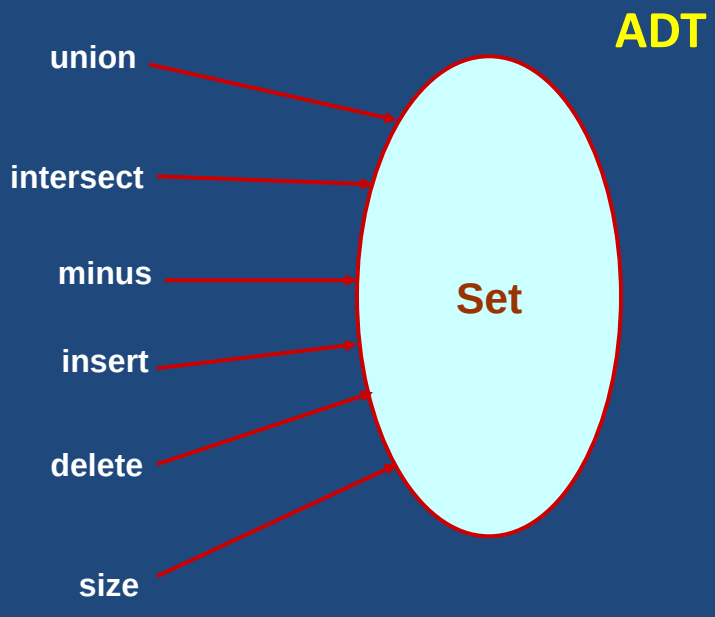
Example 2 :: Set manipulation

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

Structure definition

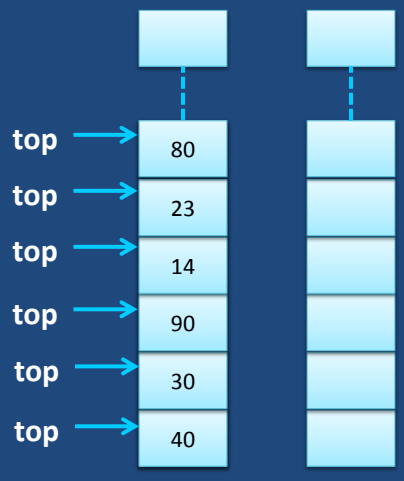
```
set *union (set a, set b);
set *intersect (set a, set b);
set *minus (set a, set b);
void insert (set a, int x);
void delete (set a, int x);
int size (set a);
```

Function prototypes



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

STACK USING ARRAY



PUSH

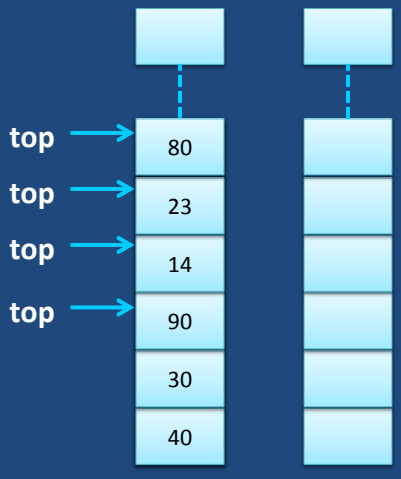
Increment top (array index)

```
#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

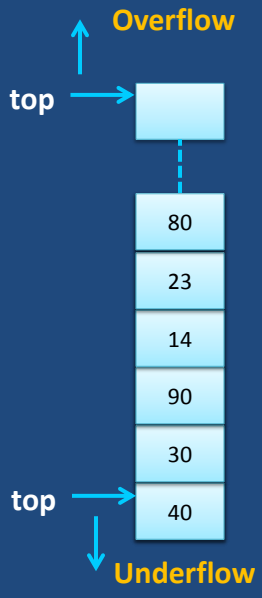


POP

Decrement top (array index)

```
#define MAXSIZE 100

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

STACK: 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;
   }
}
```

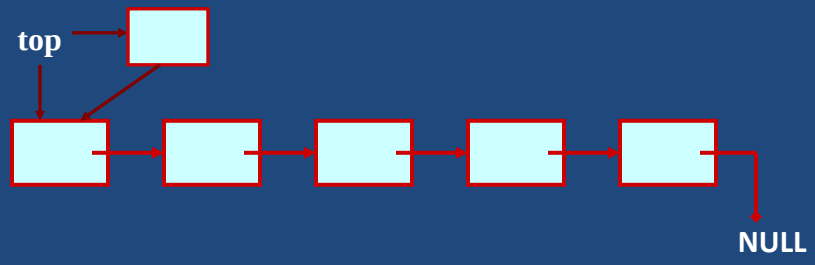
Stack Creation

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

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

Stack: Linked List Structure

PUSH

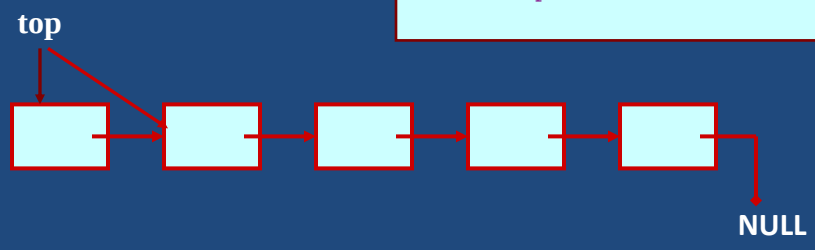
```
struct stack
{
   int value;
   struct stack *next;
};
typedef struct stack STACK;

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).

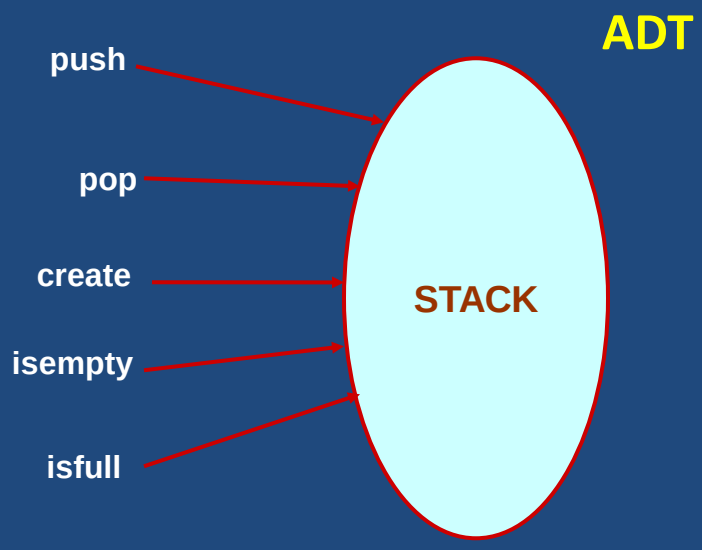
STACK using array

```
#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: 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 */
```



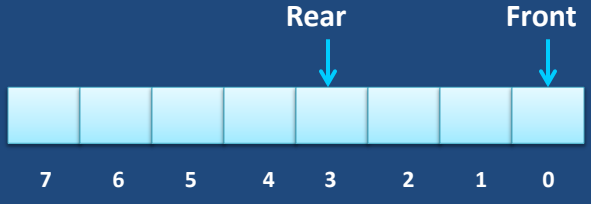
QUEUE: First-in-first-out (FIFO)

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.

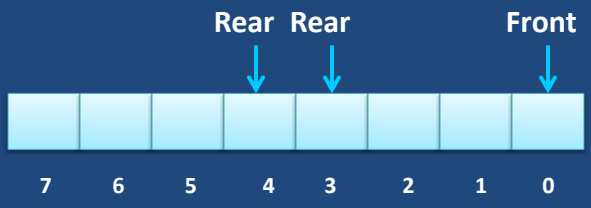
```
#define MAXSIZE 100
struct queue
{
   int que[MAXSIZE];
   int front,rear;
};
typedef struct queue QUEUE;
```



ENQUEUE

Increment rear (array index)

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



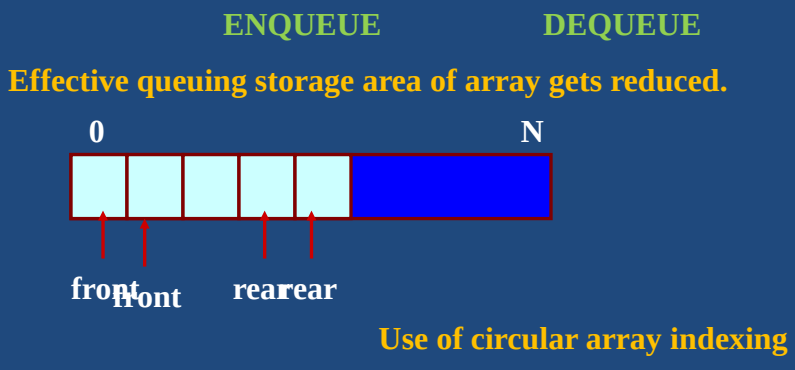
DEQUEUE

Increment front (array index)

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

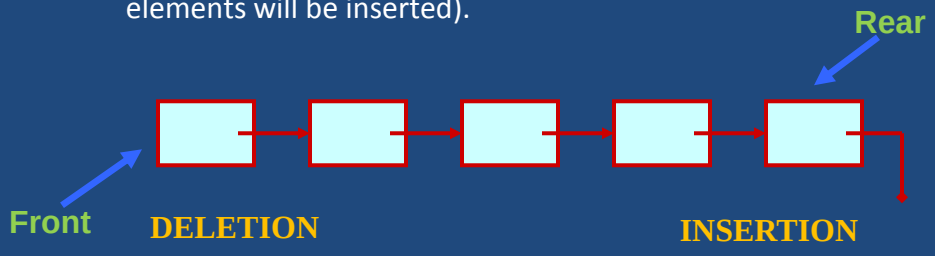
Problem With Array Implementation

- 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.



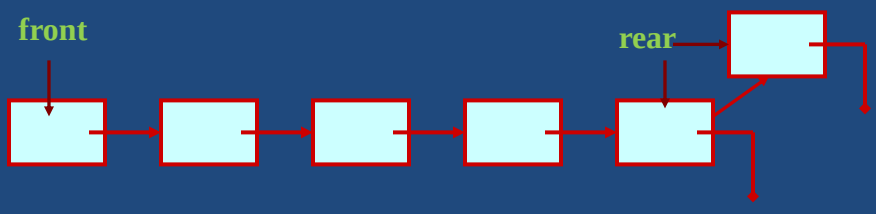
QUEUE USING LINKED LIST

- 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).



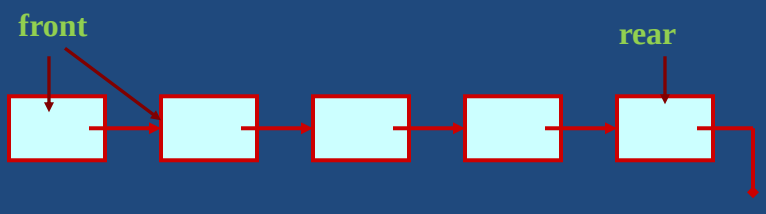
QUEUE: Insertion into a Linked List

ENQUEUE



QUEUE: Deletion from a Linked List

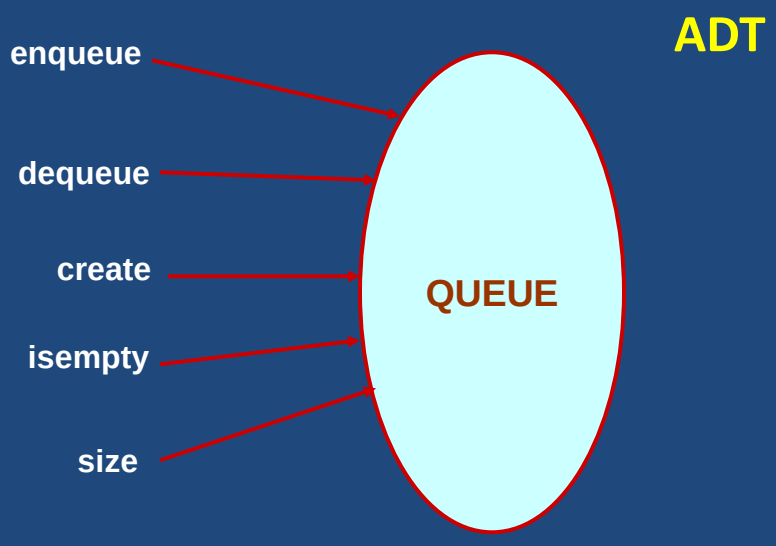
DEQUEUE



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*/
```



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 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

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

Mathematical Calculations – A Challenge

The precedence of operators affects the order of operations. A mathematical expression cannot simply be evaluated left to right.

A challenge when evaluating an expression.

Example: **A + B * C**

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 Rules

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 Pseudo Code

```
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);
}
```

Infix to Postfix Example

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 +

Homework

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

Evaluating Postfix Expression: Algorithm

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

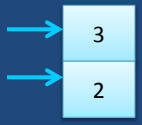
3) When the expression is ended, the number in the stack is the final answer

Evaluating Postfix Expression: Example

Infix Expression: 2 * 3 – 4 / 5

Postfix Expression: 2 3 * 4 5 / -

Evaluate Expression

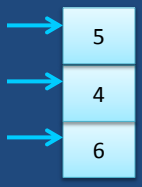


Evaluating Postfix Expression

Infix Expression: 2 * 3 – 4 / 5

Postfix Expression: 2 3 * 4 5 / -

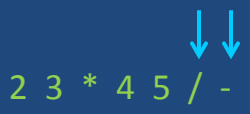
Evaluate Expression



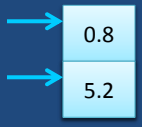
Evaluating Postfix Expression

Infix Expression: 2 * 3 – 4 / 5

Postfix Expression: 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.

Examples and Exercises

Example1: Program to delete vowels from a string

Input 1: Massachusetts
Output 1: Msschstts

Input 2: New Delhi
Output 2: Nw Dlh

```
#include <stdio.h>
#include <string.h>
int main()
{
        char s[100], t[100];
        int i, j = 0;
        printf("Enter a string to delete vowels\n");
        gets(s);
        for(i = 0; s[i] != '\0'; i++) {
           if(check_vowel(s[i]) == 0)
              t[j] = s[i];
              j++;
      }
     t[j] = '\0';
    printf("String after deleting vowels: %s\n", s);
    return 0;
}
```

```
int check_vowel (char c)
{
       switch(c)   {
            case 'a':
            case 'A':
            case 'e':
            case 'E':
            case 'i':
            case 'I':
            case 'o':
            case 'O':
            case 'u':
            case 'U':   return 1;
             default: return 0;
       }
}
```

Example 2: Check a string is palindrome or not using library function

Input 1: Civic
Output 1: Civic is a palindrome

Input 2: Palindrome
Output 2: Palindrome is not a palindrome.

Hint: strcmp, strlwr, strrev

```
#include <stdio.h>
#include <string.h>
int main()
{
  char input[100], r[100];
  printf("Enter the string for palindrome check \n");
  scanf("%s", input);
  strcpy(r, input);
  strrev(r);
  if(strcasecmp(input, r) == 0 )  // or if(strcmp(strlwr(input), strlwr(r)) == 0 )
    printf("%s is a palindrome.\n", input);
  else
    printf("%s is not a palindrome.\n", input);
  return 0;
}
```

Example 3: Print the abbreviated form of a person's name

Input 1: First name : Mohandas
Middle name : Karamchand
Last name : Gandhi

Output 1: M.K.Gandhi

```
#include <stdio.h>
int main()
{
   char  fname[20], mname[20], lname[20];
    printf("Enter full name (first middle last): ");
    scanf("%s %s %s", fname, mname, lname);
    printf("Abbreviated name: ");
    printf("%s \n", strcat( fname[0],'.', mname[0],'.', lname));
    return 0;
}
```

Example 4: Find Sum of lower triangular and upper triangular matrix (without diagonal)

Input :
1 2 3
4 5 6
1 0 7

Output:
Upper triangular matrix=11
Lower triangular matrix=5

```
#include<stdio.h>
 int main()
{
   int i,j,m,n,d1=0,d2=0,a[10][10];
   printf("How many rows and columns:");
   scanf("%d%d",&m,&n);
   printf("Enter matrix elements:\n");
   for(i=0;i<m;++i) {
      for(j=0;j<n;++j)  {
         scanf("%d",&a[i][j]);
            if(j>i)
               d1+=a[i][j];
            else
               if(i>j)
                  d2+=a[i][j];
     } }
    printf(\n"Sum of elements above the diagonal=%d\n",d1);
   printf("Sum of elements below the diagonal=%d",d2);
    return 0;
}
```

Example 5: Check if a matrix is symmetric or not

Input :
1 2 3
2 5 6
3 6 7

Output: Symmetric matrix.

```
#include<stdio.h>
int  main()
{  int m, n, i, j, matrix[10][10],
transpose[10][10];
   printf("Enter the number of rows and
columns of matrix\n");
   scanf("%d%d",&m,&n);
   printf("Enter the elements of matrix\n");
   for ( i = 0 ; i < m ; i++ ) {
      for ( j = 0 ; j < n ; j++ ) {
         scanf("%d",&matrix[i][j]);
    }}
   for ( i = 0 ; i < m ; i++ ) {
      for ( j = 0 ; j < n ; j++ ) {
         transpose[j][i] = matrix[i][j];
   }}
if ( m == n ) {
      for ( i = 0 ; i< m ; i++ ) {
         for ( j = 0 ; j < m ; j++ ) {
            if ( matrix[i][j] !=
transpose[i][j] )
               break;
          }
         if ( j != m )
         break;
       }
       if ( i == m )
          printf("Symmetric matrix.\n");
}
else
       printf("Not a symmetric
matrix.\n");
  return 0;
}
```

Example 6:Store temperature of two cities for a week and display it.

Example:

City 1, Day 1: 33
City 1, Day 2: 34
City 1, Day 3: 35
City 1, Day 4: 36
City 1, Day 5: 35
City 1, Day 6: 34
City 1, Day 7: 33
City 2, Day 1: 20
City 2, Day 2: 19
City 2, Day 3: 18
City 2, Day 4: 16
City 2, Day 5: 16
City 2, Day 6: 16
City 2, Day 7: 16

	D1	D2	D3	D4	D5	D6	D7
C1	33	34	35	36	35	34	33
C2	20	19	18	16	16	16	16

```
#include <stdio.h>
#define CITY 10
#define WEEK 10
void main()
{
   int city=5,week=7, i, j;
   float temperature[CITY][WEEK] ;
  for (i = 0; i < city; i++) {
     for(j = 0; j <week; j++) {
          printf("City %d, Day %d: ", i+1, j+1);
          scanf("%f", &temperature[i][j]);
     }
  }
 printf("\n Displaying values: \n\n");
 for (i = 0; i < city; i++) {
    for(j = 0; j < week; j++) {
        printf("City %d, Day %d = %f\n", i+1, j+1, temperature[i][j]);
    }
  }
}
```

Example 7: Write a function that perform binary search on two dimensional (2D) array given a key.

Input : 5
1 2 3
2 5 6
3 6 7

Output: Key 5 is found at index (2,2).

```
int binSearchOnMatrix(int matrix[][], int rowS. int colS, int key)
{
    int start = 0, mid, row, col, value;
     int end = rowS * colS - 1;
    while (start <= end)
    {
        mid = start + (end - start) / 2;
        row = mid / colS;
        col = mid % colS;
        value = matrix[row][col];
        if (value == key)
          { printf("Key %d is at index( %d, %d).\n", key, row, col );
                    return 1; }
        if (value > key)
            end = mid - 1;
        else
            start = mid + 1;
     }
     return 0;
}
```

Example 8: Given 2D array where row-wise it is sorted in non-decreasing order, write a function that prints all common elements in these arrays.

Input:
Ar1[0][] = {1, 5, 5, 20, 80}
Ar2[1][] = {3, 4, 5, 5, 10, 20, 45}
Ar3[2][] = {5, 5, 10, 20}

Output: 5, 5, 20

Example 8: Function that prints all common elements in these arrays.

```
void findCommon(int ar1[], int ar2[], int ar3[], int n1, int n2, int n3)
{
    int i = 0, j = 0, k = 0;
    while (i < n1 && j < n2 && k < n3){
        if (ar1[i] == ar2[j] && ar2[j] == ar3[k])
        {   printf("\n%d", ar1[i]);  i++; j++; k++; }
        else if (ar1[i] < ar2[j])
            i++;
        else if (ar2[j] < ar3[k])
            j++;
        else
            k++;
    }
}
```

Example 9

Write a C Program to store student course information using structures with Dynamically Memory Allocation.

Input:

Enter number of records: 2
Enter name of the subject and marks respectively:
Programming
22
Enter name of the subject and marks respectively:
Structure
33

Output:

Displaying Information:
Programming 22
Structure 33

Exercise

1. Reverse an array without affecting special characters

Sample output
Enter the string : a%gh
String after reversing: hg%a

2. Find the inverse of a 3x3 matrix

3. Store the pairwise distance between n points and store the distance in a 2D array

4. Find the third largest element in an array of distinct elements.

5. Find the largest & smallest word in a string (set of words).

6. Write a C program to print the number of words, lines and characters in a line text.