Introduction to Data Structure

Data Structure:

Logical and mathematical model of organization of data is called Data structure. Data structure consists of two things: A **collection of simple or structure data type** and a **set of rules** to organize and accessing the collection.

Basic Operations on Data structure

1. Traversal: Visiting each element of a list/collection exactly once
2. Searching: Finding location of an item in a list based on given key value
3. Insertion: Insertion of a new element into a list
4. Deletion: Removal of an element from a list
5. Sorting: Arranging the data element in some logical order
6. Merge: Combining the element of two or more sorted list into one sorted list

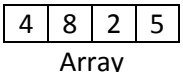
Examples of some data structure

Array :

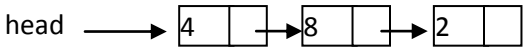
Array contains multiple numbers of data in adjacent memory location. It supports direct access to its elements.

Linked list:

Linked list data structure consists of nodes, each containing data and link pointing to the next node in the sequence. It does not allow direct access to a node.



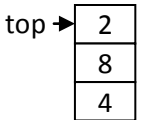
Array



Linked List

Stack :

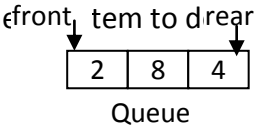
Stack is a restricted linear data structure in which insertion and deletion of data take place at one end. Stack is called a LIFO List (Last In First Out) as the items inserted last will be processed first.



Stack

Queue:

Queue is a restricted linear data structure in which insertion of data takes place at one end and deletion of data takes place in other end. Queue is a FIFO List (First In First Out) list as the items inserted first is the



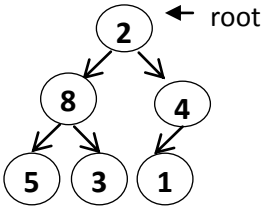
Queue

Tree :

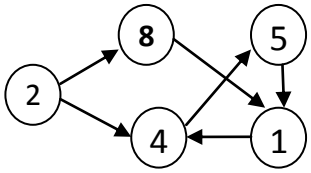
Tree is a non linear data structure that is used to represents the data in a hierarchical relationship. A node may have multiple successors (children nodes) but only one predecessor (parent node)

Graph :

Graph is a nonlinear data structure, consists of a set of nodes and a set of links. A node is called vertex. A link is called the edge that connects two vertices. In Graph, relationship among the nodes is less restricted. It means a node can have multiple predecessors.



Tree



Graph

Linear data structure:

The data items in linear data structure form a sequence and maintain a linear ordering. Data are arranged in linear fashion though their memory locations may not be sequential.

In this data structure, the data/elements are traversed sequentially one after another and only one element can be directly accessible at each step.

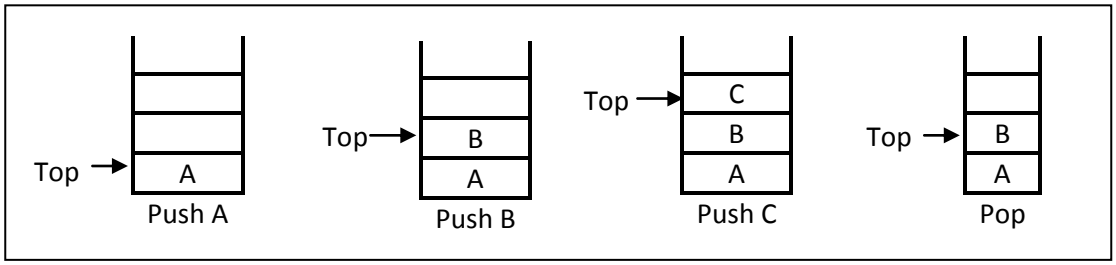
Example: Array, Linked list, stack, queue

Non-Linear Data structure:

In Non-linear data structure, data are not arranged in a sequence. It is constructed by attaching a data element to several other data element in such way that at one-step multiple data element can be accessible. Example Tree, Graph.

Stack

Stack is a restricted linear data structure in which insertion and deletion of data items take place at one end. The insertion and deletion end of the Stack called the TOP end. Stack is called a LIFO List (Last In First Out) as the items inserted last is deleted first.



Basic Stack Operations:

1. **PUSH:** This operation inserts an item at the top of the stack. Once an item is pushed, the TOP pointer now points to newly inserted item.
2. **POP:** This operation removed the item at top of the stack. Top is decremented on each pop operation.

State/ Error conditions in a Stack

1. **Overflow:** If a Stack is full and we want to push an item. This state of stack is called stack overflow.
2. **Underflow: When** a Stack is empty and we want a pop operation. Then this situation is called underflow.

Example: Consider an array implemented empty Stack of Size=5, show the status of the Stack stepwise after following operations are made

PUSH(10), PUSH(20),PUSH(30), POP, PUSH(40), PUSH(50), POP, PUSH(60),PUSH(70), PUSH(80), POP

Assuming Array index starts with 1

Operations	STACK					TOP
						TOP=0
PUSH(10)	10					TOP=1
PUSH(20)	10	20				TOP=2
PUSH(30)	10	20	30			TOP=3
POP	10	20				TOP=2
PUSH(40)	10	20	40			TOP=3
PUSH(50)	10	20	40	50		TOP=4
POP	10	20	40			TOP=3
PUSH(60)	10	20	40	60		TOP=4
PUSH(70)	10	20	40	60	70	TOP=5
PUSH(80)	10	20	40	60	70	Error! UNDERFLOW
POP	10	20	40	60		TOP=4

Algorithm to PUSH an element in a Stack.

```
PUSH (STACK , MAX , top , item)
//This Algorithm inserts an element item into STACK.
1. IF TOP = MAX  THEN    // Is Stack Full
2.     Write "OVERFLOW" and Return.
   //End IF
3. top = top + 1
4. STACK[top] = item  // inserts new element
5.return.
```

Algorithm to POP an element from a Stack.

```
POP (STACK , MAX , top)
```

//This algorithm deletes an element from STACK and assigns it to item

```
1. IF top = 0  THEN  // Stack is empty
2.         Write  "UNDERFLOW" and Return.
   //End of IF
3. item = STACK [top].
4. top = top - 1
5. return item.
```

Implementation of Stack using Array in C

Program: Write a menu design program to implement following operations on a Array implemented Stack

```
/* Array implemented stack*/
#include"stdio.h"
#include"stdlib.h"
#define MAX 10
int stack[MAX], top= -1;
void push(int);
int pop();
void display(int);
int  main()
  {
  char ch;
  int choice,num;
  do {
     printf("\n 1:PUSH.");
     printf("\n 2:POP.");
     printf("\n 3:DISPLAY.");
     printf("\n 4:EXIT.");
     printf("\n Enter Your Choice(1,2,3,4):");
     scanf("%d",&choice);
     switch(choice)
     {
     case 1:
       printf("Enter The Number To Be Insert:");
       scanf("%d",&num);
       push(num);
       break;
     case 2:
       printf(" %d is deleted",pop());
       break;
     case 3: display(top);
          break;
     case 4: exit(0);
     }
   printf("\nDo U want to Continue:");
   getchar();
   ch=getchar();
   }while(ch=='y');
return 0;
}
```

```
void push(int num)
   {
   if(top== MAX -1)
      {
      printf("Overflow");
      return;
      }
   top = top + 1;
   stack[top] = num;
   }

int pop()
  {
  int d;
  if(top == -1)
     {
     printf("Underflow!");
     return 0;
     }
  else
     {
     d=stack[top];
     top=top-1;
     return  d;
     }
  }

void display(int i)
   {
   while(i>=0)
      {
      printf(" %d", stack[i]);
      i= i- 1;
      }
   }
```

Arithmetic Expression

An arithmetic expression consists of operands, operators and parenthesis. Three commonly used arithmetic expressions are:

- Infix notation
- Prefix notation (Polish notation)
- Postfix notation (Reverse Polish notation)

Infix expression: In This notation, the operator is place between two operands (for binary operator). For example to add 6 and 8, we write 6 + 8.

Prefix expression: In this expression, operator is present before the operands. This expression is named after a Polish Mathematician **Jan Lukasiewicz,** so it is otherwise called **Polish Notation**. Example: To add 6 and 8, we write + 6 8

Postfix expression: n this expression, operator is present after the operands. As this expression is just the reverse of Polish notation**,** so it is otherwise called **Reverse Polish Notation**.

Example: To multiply 6 and 8, we write 6 8 *

For an Infix notation 5 * 3 + 9 / 3, the postfix and prefix form are as follows:

Prefix: + * 5 3 / 9 3

Postfix : 5 3 * 9 / 3 +

Conversion of Infix notation to postfix notation

Algorithm : InfixToPostfix (I)

//This algorithm converts an Infix notation (I) to postfix notation

1. P = ϕ
2. Push '(' and add ') ' to the end of I
3. FOR each symbol S in I repeat step 4 to 6.
4. IF symbol is an operand, print operand.
5. IF symbol is '(' , Push it into stack.

6. IF symbol is an operator($\otimes$) then
   i. Repeatedly pop from stack and add to P, each operator on the top which has same or higher precedence than $\otimes$
   ii. Add $\otimes$ into stack
7. IF symbol is ')' then
   i. Repeatedly pop from stack and add to P each operator on top of stack until '(' is encountered
   ii. Remove '('
8. Return P.

Convert **(A – B) * (C / D) + E**) to postfix

Step	Input	Action	Stack	Output (p)
1		Push (	(	
2	(	Push	((	
3	A	Print A	((	A
4	-	Push -	((-	A
5	B	Print B	((-	AB
6	)	Print –, Remove (	(	AB-
7	*	Push *	(*	AB-
8	(	Push (	(*(	AB-
9	C	Print C	(*(	AB-C
10	/	Push /	(*(/	AB-C
11	D	Print C	(*(/	AB-CD
12	)	Pop /, Print /, Remove (	(*	AB-CD/
13	+	Pop *, Print *, Push +	(+	AB-CD/ *
14	E	Print E	(+	AB-CD/ *E
15	)	Pop +, Print +, Remove (	EMPTY	**AB-CD/ *E+**

Exercise:

1. A * B + C / D * E * F ^ G - H
2. A - (B+C) * D ^ (E+F) + G
3. 12 + 6 / 3 * 7 – 9 ^ 3
4. A-B/C*D^E
5. A -(B / C + D) * E ^ (F + G))

Evaluation of postfix expression

Algorithm: PostfixEvaluation(P)

//This algorithm returns the value of postfix expression P

1. FOR each symbol(s) in P
2. IF Symbol(s) is an operand, PUSH(s)
3. IF Symbol(s) is an operator($\otimes$) Then
4. A = pop()
5. B = pop()
6. C = B $\otimes$ A
7. PUSH (C)
8. Value = pop()
9. Return value

Problem: Evaluate 5 3 – 9 3 / +

Solution:

Symbol	Stack	Action`
5	5	Push 5
3	5 3	Push 3
-	2	A=3,B=5,C=5-3=2, push C
9	2 9	Push 9
3	2 9 3	Push 3
/	2 3	A=3,B=9,C=9/3=3, push C
+	5	A=3,B=2,C=2+3=5,push C

Value of the expression = pop the stack **= 5**

Program:

1. Implement all the operations of a stack using linked list.
2. WAP to input an infix expression and convert to postfix
3. WAP to input a postfix expression and evaluate it.
4. WAP to input a string. Reverse it using a stack.

Self-Learning:

1. Define the **Tower-of-Hanoi** problem. Stepwise show the solution of ToH with 3 disks. Write a recursive algorithm to solve the problem. Write a program to implement it.
2. Parenthesis matching algorithm. Write algorithm and program to implement it.