DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institute Affiliated to VTU, Belagavi)

Unit – II

STACK

Stack is a linear non primitive data structure in which insertion and deletion are made through same end, according to last in first out (LIFO) principles.

The operation of stack

PUSH & POP

Push the item into the stack and **pop** to delete an item from a stack. A stack is a limited access data structure - elements can be added and removed from the stack only at the top. **Push** adds an item to the top of the stack, **pop** removes the item from the top. A helpful analogy is to think of a stack of books; you can remove only the top book, also you can add a new book on the top. The push and pop operation as shown in figure 1.

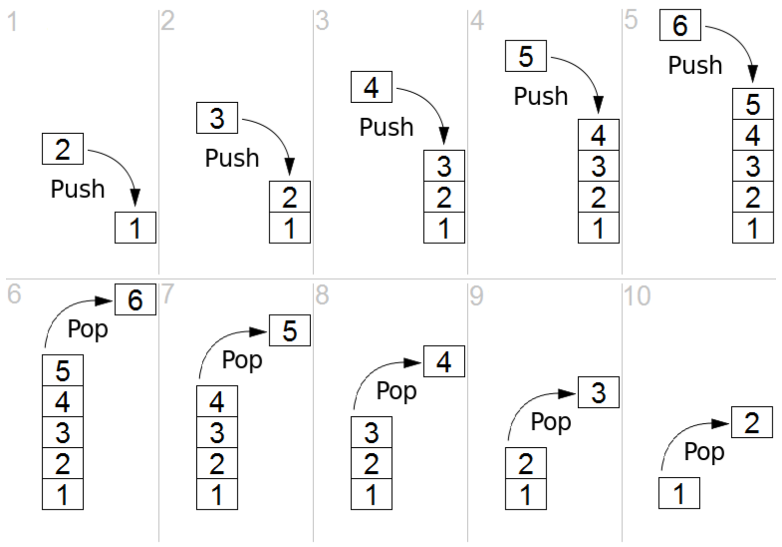


Figure 1. Stack operation push & pop

Function push & pop.

```
push()
{
        if(top ==size -1)
        printf("Stack Full")
        else
        s[++top]=item;
}
pop()
{
        if(top ==-1)
        printf("Stack Empty")
        else
        top--;
}
```

Write a C program to perform the operation of stack such as push, pop & display the status of stack.

```
#include<stdio.h>
#define SIZE 5
void push (int, int *, int [ ] );
void pop (int *, int [ ]);
void display (int, int [ ]);
void main ( )
{
        int s[SIZE], top = -1 ,item ,ch;
        clrscr ( );
        while (1)
         {
                printf (“\n 1: PUSH 2:POP 3:DISPLAY Otherwise EXIT\n”);
                scanf (“%d”, &ch);
                switch (ch)
                {
                        case 1: printf (“Enter the element \n”);
                                scanf (“%d”, &item);
                                push (item, &top, s); break;
                        case 2: pop (&top, s); break
                        case 3: display (top, s); break
                        default: printf (“Invalid operation”);
                                getch ( );
                                exit (0);
                }
        }
}

void push (int item , int *top , int s[ ])
{
        if (*top == SIZE – 1)
        printf (“Stack if Full “);
        else
        {
                *top = *top +1;
                S[*top]= item;
        }
}
void pop (int *top, int s[ ])
{

        if (*top == -1)
        printf (“Stack is Empty”);
        else
        {
                printf (“The deleted element is %d”, s[*top]);
                *top = *top – 1;
        }
}
```

```
Void display (int top, int s[ ])
{
        int i;
        if (top == -1)
        printf (“Stack is Empty”);
        else
        {
                printf (“The status of stack is \n”);
                for (i=0; i<=top; i++)
                printf (“%d “,s[i]);
        }
}
```

Application of stack

1. Conversion of expression
2. Recursion

Expression: An expression is a mathematical phrase that combines of numbers , variables and the mathematical operator.

Types of expression
1. infix expression
2. postfix expression
3. prefix expression

Infix expression: In infix expression the operator is placed between two or more operands.
Example:

Infix expression = operand1 operator operand 2
Postfix expression = operand1 operand 2 operator
Prefix expression = operator operand1 operand 2

Sl.no	Infix	postfix	prefix
1.	a+b	ab+	+ab
2.	a*b + c	ab*c+	+*abc
3.	(a-b) * (c –d)	ab-cd-*	*-ab-cd
4.	(a+b)*d+e/(f+p*h)+m	ab+d*efph*+/+m+	++*+abd /e+f*phm
5.	((a/(b-c+d))*(e-p)*h)	abc-d+/ep-*h*	**/a+-bcd-eph
6.	(1 + 2 * 3) / 4 + 2 – 4 * 6	123*+4/2+46*-	-+/+1*2342*46
7.	2+3*4/6+6*8	234*6/+68*+	++2/*346*68

I. Algorithm to convert the infix expression into postfix expression. It uses a stack; but in this case, the stack is used to hold operators rather than numbers. The purpose of the stack is to reverse the order of the operators in the expression. It also serves as a storage structure, since no operator can be printed until both of its operands have appeared.

1. Read the infix expression
2. Initially push # onto a stack, for # set priority as -1;
3. If the input symbol '(' push onto a stack
4. If the input symbol is a operand then place into a postfix expression
5. If the current input symbol is an operator then check the stack of top precedence value, if the input symbol precedence value is more than push onto stack otherwise pop from stack.
6. If the input symbol ')' then pop all the element from the stack until we get '(' and append popped operation to postfix expression. Finally just pop ')'.
7. Finally pop the remaining content of stack until empty and store pop element into postfix expression and the postfix expression end with null character.

The input priority

Symbol	Symbol	Value
(	#	1
+	-	2
*	/	3
^	$	4

Example: (a * b) +c / d

Input read	Action	Stack	output
(	Push (	(	
a	Print a	(	a
*	Push *	(*	a
b	Print b	(*	ab
)	Pop * & print it		ab*
+	Push +	+	ab*
c	Print c	+	ab*c
/	Push /	+ /	ab*c
D	Print d	+/	ab*cd
End of the input	Pop element by element and print it	Empty	ab*cd/+

Program in C to convert a given valid parenthesized infix arithmetic expression to postfix expression and then print both the expressions. The expression consists of single character operands and the binary operators +, - , * , /

```
#include<stdio.h>
#include<string.h>
#include<conio.h>

char stack[50];
int top=-1;

void push(char x)
{
    stack[++top]=x;
}
int pop()
{
    return(stack[top--]);
}
int prior(char x)
{
    int p;
    if(x=='('||x=='#') p=1;
    if(x=='+'||x=='-') p=2;
    if(x=='*'||x=='/') p=3;
    if(x=='^'||x=='$') p=4;
    return p;
}

void main()
{
    char infix[20], postfix[20];
    int i,j=0;
    clrscr();
    printf("Enter an infix expression:\n");
    gets(infix);
    push('#');
    for(i=0;i<strlen(infix);i++)
    {
            if(isalnum(infix[i]))
            postfix[j++]=infix[i];
            else if(infix[i]=='(' )
            push(infix[i]);
            else if(infix[i]==')')
            {
                    while(stack[top]!='(')
                    postfix[j++]=pop();
                    pop();
            }
            else
            {
                    while(prior(stack[top])>=prior(infix[i]))
                    postfix[j++]=pop();
                    push(infix[i]);
            }
    }
    while(stack[top]!='#')
      postfix[j++]=pop();
      postfix[j]='\0';
      printf("\nPostfix expression is:\n");
      puts(postfix);
      getch();
}
```

II. Algorithm to Evaluation of postfix expression

1. Read the postfix expression from left to right
2. If the input symbol is an operand push into a stack
3. If the input symbol is an operator and form the following operations
   + Then result = operand1 + operand2
   - Then result = operand1 - operand2
   * Then result = operand1 * operand2
   / Then result = operand 1 / operand2
4. Push the result onto stack
5. Repeat steps 1 to 4 until the postfix expression ends.

Input: 634+*98++

Input symbol	Action	Stack
6	Push 6	6
3	Push 3	6 3
4	Push 4	6 3 4
+	Pop 4 Pop 3 and perform 3+4 push 7	6 7
*	Pop 7 pop 6 and perform 6*7 push 42	42
9	Push 9	42 9
8	Push 8	42 9 8
+	Pop 8, pop 9 and perform 9+8 push 17	42 17
+	Pop 42, pop 17 and perform 17+42 push 59	59
Display stack[top] that is 59		

Program in C to evaluate a valid postfix expression using stack. Assume that the postfix expression is read as single line consisting of non-negative single digit operands and binary arithmetic operators. The arithmetic operators are +(add),-(sub),*(mul) and / (divide)

```
    #include<stdio.h>
    #include<math.h>
    #include<conio.h>
    int stack[20], top=-1;
    void push(int x)
    {
        stack[++top]=x;
    }

    int pop()
    {
        return(stack[top--]);
    }
    void main()
    {
        char s[20];
        int x, i, y;
        clrscr();
        printf("\n evaluation of postfix expression \n");
        printf("\n enter a postfix expression ");
        gets(s);
        for(i=0;i<strlen(s);i++)
        {
            if(isdigit(s[i]))
            push(s[i]-'0');
            else
            {
                y=pop();
                x=pop();
              switch(s[i])
                {
                    case '+':push(x+y); break;
                    case '-':push(x-y);  break;
                    case '*':push(x*y); break;
                    case '/':push(x/y); break;
                    case '^':
                    case '$':push(pow(x,y)); break;
                }
            }
        }
    printf("\n result is %d ",pop());
    getch();
}
```

III. Algorithm to convert postfix expression to infix expression

1. While there are input symbol left
2. Read the next symbol from input.
3. If the symbol is an operand
   Push it onto the stack.
4. Otherwise, the symbol is an operator, pop the two element from the stack and Perform the following
   P2 =pop();
   P1 =pop();
   Strcpy(infix,'(');
   Strcat(infix,p1);
   Strcat(infix,op);
   Strcat(infix,p2);
   Strcat(infix,')');
   Push(infix);
5. Push the resulted string back to stack.
7. Repeat step 2 to 4 till the postfix expression ends.
8. Finally display stack of top or the infix string.

Postfix expression: ab*cd/+

Current input symbol	Action	Stack
a	Push a	a
b	Push b	a b
+	Pop 2 values from the stack, P2 = b , p1= a, then step 4 in the algorithm.	(a + b)
c	Push c	(a + b) c
d	Push d	(a + b) c d
/	Pop 2 values from the stack, P2 = d, p1= c, then step 4 in the algorithm.	(a + b) (c / d)
+	Pop 2 values from the stack, P2 = (c / d), p1= (a + b) & then step 4 in the algorithm.	((a + b) + (c / d))
End of the input.	Finally display stack of top.	((a + b) + (c / d))

IV. Algorithm to convert prefix expression to infix expression

1. Reverse the input string
2. Read the input symbol from input.
3. If the symbol is an operand then Push it onto the stack.
4. If the symbol is an operator, pop the two element from the stack and Perform the following
   P2 =pop();
   P1 =pop();
   strcpy(infix,'(');
   strcat(infix,p1);
   strcat(infix,op);
   strcat(infix,p2);
   strcat(infix,')');
   push (infix);
5. Push the resulted string back to stack.
7. Repeat step 2 to 4 till the end of the input string.
9. Reverse the stack of top
10. Finally display stack of top or the infix string.

Example: Prefix expression: *+ab/cd
Reverse the prefix expression: dc/ba+*

Current input symbol	Action	Stack
d	Push d	d
c	Push c	d c
/	Pop 2 values from the stack, P2 = c , p1= d, then step 1 to 5 in the algorithm.	(d / c)
b	Push b	(d / c) b
a	Push a	(d / c) b a
+	Pop 2 values from the stack, P2 = a, p1= b, then step 1 to 5 in the algorithm.	(d / c) (b + a)
*	Pop 2 values from the stack, P2 = (a + b) , p1= (c / d) & then step 1 to 5 in the algorithm.	((d / c) * (b + a))
End of the input.	Then reverse stack of top , Finally display stack of top.	((a+b)*(c/d))

V. Algorithm conversion of postfix expression to prefix expression

1. Read the input string
2. scan the current input symbol from the left
3. If the scanned character is a digit, then push it into the stack.

4. If the scanned character is an operator, then pop two elements from the stack. Form a string containing scanned operator and two popped elements. Push the resultant string into the stack.
   ```
   P2=pop( ), p1 =pop( );
   strcpy(pre,op);
   strcat(pre,p1);
   strcat(pre,p2)
   Push (pre);
   ```
5. Finally print prefix string or stack of top.

Example: Postfix expression: ab*cd/+

Current input symbol	Action	Stack
a	Push a	a
b	Push b	a b
*	Pop 2 values from the stack, P2 = b , p1= a, then step 4 in the algorithm.	*ab
c	Push c	*ab c
d	Push d	*ab c d
/	Pop 2 values from the stack, P2 = d, p1= c, then step 4 in the algorithm.	*ab /cd
+	Pop 2 values from the stack, P2 = /cd , p1= *ab & then step 4 in the algorithm.	+*ab/cd
End of the input.	Finally display stack of top.	

VI. Algorithm conversion of prefix expression to postfix expression

1. Read the prefix expression
2. Reverse the input expression
3. If the input symbol is an operand then push into stack[top]
4. If the input symbol is an operator then perform the following, pop two elements from the stack p1=pop(),p2=pop()
   ```
   strcpy(post, p1);
   ```

```
    strcat(post, p2);
    strcat(post, op);
    push (post);
5.  Repeat step 3 to 4 till the end of the reversed string.
```

Example: Prefix expression: *+ab-cd

Reverse: dc-ba+*

Current input symbol	Action	Stack
d	Push d	d
c	Push c	d c
-	Pop 2 values from the stack, P1 = c, p2= d, then step 4 in the algorithm.	cd-
b	Push b	cd- b
a	Push a	cd- b a
+	Pop 2 values from the stack, P1 = a, p2= b, then step 4 in the algorithm.	cd- ab+
*	Pop 2 values from the stack, P1 =ab+, p2= cd- & then step 4 in the algorithm.	ab+cd-*
End of the input.	Finally display stack of top.	ab+cd-*

Recursion: A function calling itself is known as recursion.

1. The following example calculates the factorial of a given number using a recursive function

```
#include <stdio.h>
int factorial(unsigned int i)
{
   if(i <= 1) return 1;
   }
   return i * factorial(i - 1);
}

int  main()
{
   int i = 15;
   printf("Factorial of %d is %d\n", i, factorial(i));
   return 0;
}
```

2. The following example generates the Fibonacci series for a given number using a recursive function

```
#include <stdio.h>
int fibonaci(int i)
{
   if(i == 0)  return 0;
   if(i == 1)  return 1;
   return fibonaci(i-1) + fibonaci(i-2);
}

int  main()
{
   int i, n;
   printf("Enter the number \n");
  scanf("%d",&n);
   for (i = 0; i < n; i++)
   {
      printf("%d\t\n", fibonaci(i));
   }
   return 0;
}
```

3. Recursive function for sum of natural numbers

```
int sum( int n)
{
        if(n==1) return 1;
        Return n + sum (n-1)
}
```

4. Recursive function to find the gcd of two integer numbers.

```
#include<stdio.h>
void main()
{
        int m,n;
        printf(“Enter two integer numbers \n”);
        scanf(“%d %d”,&m,&n);
        printf(“The gcd of two number is %d”, gcd(m,n));
        getch();
}

int gcd(int m , int n)
{
        if(n==0)   return m;
        return gcd(n, m%n);
}
```

5. Recursive function to find the sum of all the array elements.

```
int sum(int n, int a[ ])
{
        if(n==1) return a[0];
        return a[n-1] + sum(n-1,a);
}
```

UNIT - III
QUEUE

Queue is a linear non primitive data structure in which insertion are made through one end and deletion are made through other end, according to First In First Out (FIFO) principles.

Types of queue

1. Ordinary queue
2. Circular queue
3. Double ended queue
4. Priority queue

Ordinary queue or linear queue

In ordinary queue the insertion of elements through rear end and deletion of elements from the front end.

1. Initial assume that rear =-1 and front =0 indicates the queue is empty

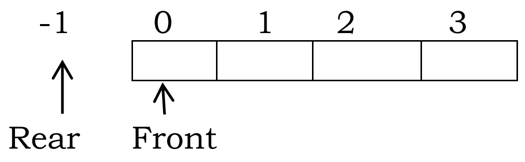
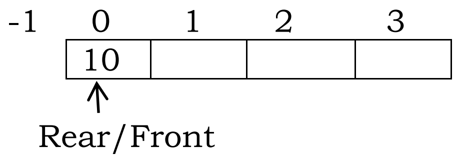
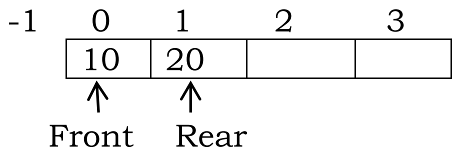


Figure1 represents the queue is empty

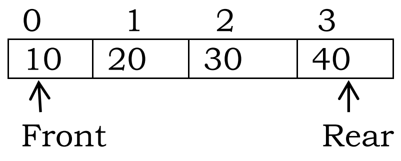
2. To insert an element into a queue increment the value of rear by 1 in that index store the element.



3. To insert an element into a queue increment the value of rear by 1 in that index store the element.



4. We can insert maximum 4 elements in the queue and then the queue is full.



Function to insert an element into a linear queue.

```
int qinsert( )
{
        if(rear == size-1)
        Printf("Queue is Full");
        else
        {
                rear = rear +1 ;
                q[rear]= item;
        }
}
```

In ordinary queue the deletion of elements through front end
.

1. Initial assume that rear =-1 and front =0 indicates the queue is empty, whenever front value is greater than rear the queue is empty.

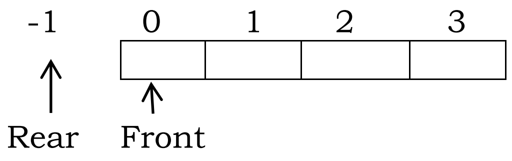
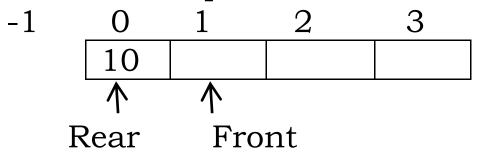
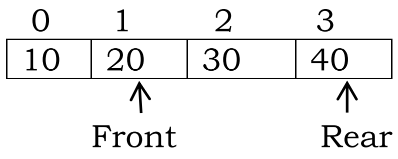


Figure1 represents the queue is empty

2. To delete an element from a queue increment the value of front by 1.



3. To delete an element from a queue increment the value of front by 1, there we can delete elements from a queue until front value is larger than rear.



Function to delete an element from a linear queue.

```
int qdelete( )
{
        if(front > rear )
        printf ("Queue is Empty");
        else
        front = front +1 ;
}
```

Function to display the element of a linear queue.

```
int qdisplay( )
{
        if(front > rear )
        printf (“Queue is Empty”);
        else
        {
                printf (“The status of Queue \n”);
                for(i= front ; i<=rear ;i++)
                printf (“%d”, q[i]);
        }
}
```

Write a C program to perform the operation of queue such as insert, delete & display the status of queue.

```
#include<stdio.h>
#define SIZE 5
void qinsert(int, int*, int *, int [ ] );
void qdelete(int *, int *, int [ ]);
void display (int,int, int [ ]);
void main ( )
{
        int q[SIZE], rear= -1, front =0, item , ch;
        clrscr ( );
        while (1)
         {
                printf ("\n 1: INSERT  2:DELETE 3:DISPLAY Otherwise EXIT\n");
                scanf ("%d", &ch);
                switch (ch)
                {
                        case 1: printf ("Enter the element \n");
                                scanf ("%d", &item);
                                qinsert(item, &rear, &front, q); break;
                        case 2: qdelete (&rear, &front, q); break
                        case 3: qdisplay (front, rear, q); break
                        default: printf ("Invalid operation");
                                getch ( );
                                exit (0);
                }
        }
}
 int qinsert( int item , int *rear, *front, int q[ ])
 {
        if(*rear == SIZE-1)
        Printf("Queue is Full");
        else
        {
                *rear = *rear +1;
                q[*rear]= item;
        }
         }
 int qdelete( int *rear, int *front, int q[ ])
 {
        if(*front > *rear )
        printf ("Queue is Empty");
        else
        {
                Printf("The deleted element is %d", q[*front]);
                *front = *front +1;
         }
 }


        int qdisplay( int front, int rear, int q[ ])
        {
                if(front > rear )
                printf (“Queue is Empty”);
                else
                {
                        printf (“The status of Queue \n”);
                        for(i= front ; i<=rear ;i++)
                        printf (“Queue[%d]=%d\n”, i, q[i]);
                }
        }
```

Write a C program to implement circular queue.

```
#include<stdio.h>
#define MAX 5
void cqinsert(int,int*,int*,int[]);
int cqdelete(int*, int *, int []);
void cqdisplay(int,int,int[]);
void main()
{
          int q[MAX],i,ch,item,f=0,r=-1,c=0;
           clrscr();
           while(1)
          {
                    printf("\n1:Insert 2:Delete 3:Display\n");
                    scanf("%d",&ch);
                    switch(ch)
                    {
                              case 1:printf("Enter the number \n");
                                        scanf("%d",&item);
                                        cqinsert(item,&c,&r,q);
                                        break;
                              case 2: cqdelete(&c,&f,q); break;
                              case 3: cqdisplay(c,f,q);break;
                              default: printf("Invalid choice \n"); getch();exit(0);
                    }
          }
}

void cqinsert(int item, int *c, int *r, int q[])
{
          if(*c==MAX)
          printf("Queue is full");
          else
          {
                    *c=*c+1;
                    *r=(*r+1)%MAX;
                    q[*r]=item;
          }
}
int cqdelete(int *c, int *f, int q[])
{
	if(*c==0)
	printf("Queue is Empty");
	else
	{
		*c=*c-1;
		printf("The deleted element is %d",q[*f]);
		*f=(*f+1)%MAX;
	}
	return 0;
}

void cqdisplay(int c, int f, int q[])
{
	int i;
	if(c==0)
	printf("Queue is Emplty");
	else
	{
		printf("The status of queue\n");
		for(i=1;i<=c;i++)
		{
			printf("%d ",q[f]);
			f=(f+1)%MAX;
		}
	}
}
```