13. Write a program in C to implement a stack using Linked List. Perform the following operations:

a) Push b) Pop c) Peek d) Display the stack content

Ans:

```c
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int info;
    struct node *ptr;
}*top,*top1,*temp;
int peek();
void push(int data);
void pop();
void display();
int count = 0;
void main()
{
    int no, ch, e;
    printf("\n 1 - Push");
    printf("\n 2 - Pop");
    printf("\n 3 - Peek");
    printf("\n 4 - Dipslay");
    printf("\n 5 - Exit");
    while (1)
    {
        printf("\n Enter choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("Enter data : ");
            scanf("%d", &no);
            push(no);
            break;
        case 2:
            pop();
            break;
        case 3:
            if (top == NULL)
                printf("No elements in stack");
            else
            {
                printf("\n Top element : %d", peek());
            }
            break;
        case 4:
            display();
```

```
            break;
        case 5:
            exit(0);
        default :
            printf(" Wrong choice, Please enter correct choice  ");
            break;
        }
    }
}
void push(int data)
{
    if (top == NULL)
    {
        top =(struct node *)malloc(1*sizeof(struct node));
        top->ptr = NULL;
        top->info = data;
    }
    else
    {
        temp =(struct node *)malloc(1*sizeof(struct node));
        temp->ptr = top;
        temp->info = data;
        top = temp;
    }
    count++;
}
void display()
{
    top1 = top;

    if (top1 == NULL)
    {
        printf("Stack is empty");
        return;
    }

    while (top1 != NULL)
    {
        printf("%d ", top1->info);
        top1 = top1->ptr;
    }
 }
void pop()
{
    top1 = top;

    if (top1 == NULL)
    {
```

```
        printf("\n Error : Trying to pop from empty stack");
        return;
    }
    else
        top1 = top1->ptr;
    printf("\n Popped value : %d", top->info);
    free(top);
    top = top1;
    count--;
}
int peek()
{
    return(top->info);
}
```

14) Write a C program to convert infix expressions to postfix expressions.

Ans:

```
#include<stdio.h>
#include<ctype.h>

char stack[100];
int top = -1;

void push(char x)
{
    stack[++top] = x;
}

char pop()
{
    if(top == -1)
        return -1;
    else
        return stack[top--];
}

int priority(char x)
{
    if(x == '(')
        return 0;
    if(x == '+' || x == '-')
        return 1;
```

```
    if(x == '*' || x == '/')
        return 2;
    return 0;
}

int main()
{
    char exp[100];
    char *e, x;
    printf("Enter the expression : ");
    scanf("%s",exp);
    printf("\n");
    e = exp;

    while(*e != '\0')
    {
        if(isalnum(*e))
            printf("%c ",*e);
        else if(*e == '(')
            push(*e);
        else if(*e == ')')
        {
            while((x = pop()) != '(')
                printf("%c ", x);
        }
        else
        {
            while(priority(stack[top]) >= priority(*e))
                printf("%c ",pop());
            push(*e);
        }
        e++;
    }

    while(top != -1)
    {
        printf("%c ",pop());
    }return 0;
}
```

15) What are the advantages of using dynamic memory allocation over static memory allocation?

Ans:

i) Dynamic memory allocation is done during program execution whereas static memory allocation is done before program execution.

ii) In dynamic memory allocation, there is memory re-usability and memory can be freed when not required which is not the case for static memory allocation.

iii) The memory size in dynamic memory allocation can be modified as and when required. in static memory allocation memory size cannot be modified.

iv) heap is used for managing the dynamic allocation of memory and stack is used for static memory allocation.

v) In dynamic memory allocation the allocated memory can be released at any time during the program. whereas In static memory allocation this allocated memory remains from start to end of the program.

vi) dynamic memory allocation is more efficient than static memory allocation.

16) Explain infix, post fix and prefix expressions with examples.

Ans: Infix, prefix and postfix are three different but equivalent notations of writing algebraic expressions. Let us discuss what they and how are they different from each other and how to obtain it.

i) infix:

a) The traditional method of our writing of mathematical expressions is called as the infix expressions.

b) It is of the form <operand><operand><operand>

c) As the name suggests, here the operator is fixed inside between the operands. e.g. A+B here the plus operator is placed inside between the two operators, (A*B)/Q.

ii) Post fix:

a) The post fix expression as the name suggests has the operator placed right after the two operands.

b) It is of the form <operand><operand><operand>

c) In the infix expressions, it is difficult to keep track of the operator precedence whereas here the postfix expression itself determines the precedence of operators (which is done by the placement of operators)i.e the operator which occurs

first operates on the operand.
d) E.g. PQ-C/, here - operation is done on P and Q and then / is applied on C and the previous result.
e) A postfix expression is parenthesis-free expression. For evaluation, we evaluate it from left-to-right.

Infix expression	Postfix expression
(P+Q)*(M-N)	PQ+MN-*
(P+Q) / (M-N) - (A*B)	PQ+MN-/AB*-

iii) Prefix:
a) The prefix expression as the name suggests has the operator placed before the operand is specified.
b) It is of the form < operator > < operand > < operand >.
c) It works entirely in same manner as the postfix expression.
d) While evaluating a prefix expression, the operators are applied to the operands immediately on the right of the operator.
e) For evaluation, we evaluate it from left-to-right. Prefix expressions are also called as polish notation.

Infix expression	Prefix expression
(P+Q)*(M-N)	*+PQ-MN
(P+Q) / (M-N) - (A*B)	-/+PQ-MN*AB

17) What is recursion? Write a C program to calculate the sum of 'n' natural numbers using recursion.

Ans: The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function.
if a program allows you to call a function inside the same function then it is called recursive function

C program for sum of n natural numbers using recursion

```
#include
int addNumbers(int n);
int main() {
int num;
printf("Enter a positive integer: ");
scanf("%d", &num);
printf("Sum = %d", addNumbers(num));
return 0;
}
int addNumbers(int n) {
if (n != 0)
return n + addNumbers(n - 1);
else
return n;
}
```

18) Write ADT for a stack. Give application of stack.

Ans: i) A stack is an ordered list of elements in which elements are always inserted and deleted at one end, say the beginning.

ii) In the terminology of stacks, this end is called the top of the stack, whereas the other end is called the bottom of the stack

iii) The insertion operation is called push and the deletion operation is called pop.

Push operarion:

- The push operation is used to insert an element into the stack.
- The new element is added at the topmost position of the stack. However, before inserting the value, we must first check if TOP=MAX-1, because if that is the case, then the stack is full and no more insertions can be done.
- If an attempt is made to insert a value in a stack that is already full, an OVERFLOW message is printed

Pop Operation

- The pop operation is used to delete the topmost element from the stack.

- Before deleting the value, we must first check if TOP=NULL because if that is the case, then it means the stack is empty and no more deletions can be done.
- If an attempt is made to delete a value from a stack that is already empty, an UNDERFLOW message is printed.

Peek Operation:
Peek operation gets the top data element of the stack without removing it.

code for push, pop and peek

```
void push(int max){
if (top == max-1){
   printf("Overflow");
}else{
printf("Enter the element: ");
scanf("%d",el);
stack[++top]=el;
}}
void pop(){
if(top==-1){
   print("UNDERFLOW");
}
```

```
else{
el = stack[top];
top--;
printf("Deleted element: %d",el)l;
}}
void peek(){
if(top==-1){
   printf("UNDERFLOW");
}else{
   printf("Top Element: %d", stack[top]);
}
}
```

Application of stack:

Reversing a list: • A list of numbers can be reversed by reading each number from an array starting from the first index and pushing it on a stack.

• Once all the numbers have been read, the numbers can be popped one at a time and then stored in the array starting from the first index.

• Example :

Enter the number of elements in the array : 5

Enter the elements of the array : 1 2 3 4 5

The reversed array is : 5 4 3 2 1

19) Explain recursion as an application of stack with examples.

Ans:
- A recursive function is defined as a function that calls itself to solve a smaller version of its task until a final call is made which does not require a call to itself.
- Since a recursive function repeatedly calls itself, it makes use of the system stack to temporarily store the return address and local variables of the calling function.
- Every recursive solution has two major cases:
  - Base case
  - Recursive case

Types of Recursion are
- Direct Recursion
- indirect Recursion
- Tail Recursion

Example:
Tower of Hanoi: • The tower of Hanoi is one of the main applications of recursion. It says, 'if you can solve n-1 cases, then you can easily solve the nth case'.