**Q: SINGLY LINKED LIST**

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

int data;

struct node \*next;

}node;

node \*create\_node(){

node \*new\_node;

new\_node = (node \*)malloc(sizeof(node));

new\_node -> next = NULL;

printf("Enter data for NEW NODE: ");

scanf("%d", &new\_node->data);

return new\_node;

}

void display(node \*start){

if(start==NULL) printf("No node present\n");

else{

node \*ptr;

ptr = start;

while(ptr!=NULL){

printf("Data = %d\t", ptr->data);

printf("Next = %d\n", ptr->next);

ptr=ptr->next;

}

}

}

void insert\_first(node \*\*start){

node \*inserting\_node;

inserting\_node = create\_node();

if(\*start == NULL) \*start = inserting\_node;

else{

inserting\_node -> next = \*start;

\*start = inserting\_node;

}

}

void insert\_last(node \*\*start){

node \*new\_2\_insert;

new\_2\_insert = create\_node();

if(\*start == NULL) \*start = new\_2\_insert;

else{

node \*ptr = \*start;

while(ptr->next == NULL){

ptr = ptr->next;

}

ptr->next = new\_2\_insert;

}

}

void delete\_first(node \*\*start){

if(\*start == NULL) printf("No linked list\n");

else{

node \*temp;

temp = \*start;

\*start = (\*start)->next;

free(temp);

}

}

void delete\_last(node \*\*start){

if(\*start == NULL) printf("No linked list\n");

else{

node \*pre\_ptr=\*start, \*ptr=\*start;

while(ptr->next != NULL){

pre\_ptr = ptr;

ptr = ptr->next;

}

pre\_ptr->next = NULL;

free(ptr);

}

}

void main(){

node \*temp, \*start;

int choice;

while(1){

printf("Press 1 for Create Node\n");

printf("Press 2 to INSERT at FIRST\n");

printf("Press 3 to INSERT at LAST \n");

printf("Press 4 to DELETE FIRST \n");

printf("Press 5 to DELETE at LAST \n");

printf("Press 6 for Display\n");

printf("Press 7 to EXIT\n");

printf("Enter Your Choice: ");

scanf("%d", &choice);

switch(choice)

{

case 1: temp = create\_node();

break;

case 2: insert\_first(&start);

break;

case 3: insert\_last(&start);

break;

case 4: delete\_first(&start);

break;

case 5: delete\_last(&start);

break;

case 6: display(start);

break;

case 7: exit(0);

break;

default: printf("Invalid!");

break;

}

}

}

**OUTPUT:**

|  |  |
| --- | --- |
| INPUT:  (Choice) | OUTPUT: |
| 1 | Enter Data for NEW NODE: 30 |
| 2 | Enter Data for NEW NODE: 20 |
| 2 | Enter Data for NEW NODE: 10 |
| 3 | Enter Data for NEW NODE: 40 |
| 3 | Enter Data for NEW NODE: 50 |
| 6 | Data= 10 Next= -2033092480  Data= 20 Next= -2033092352  Data= 30 Next= -2033091648  Data= 40 Next= -2033092416  Data= 50 Next= 0 |
| 4 |  |
| 5 |  |
| 6 | Data= 20 Next= -2033092352  Data= 30 Next= -2033091648  Data= 40 Next= 0 |

**Q: CIRCULAR LINKED LIST**

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

int data;

struct node \*next;

}node;

node \*create(){

node \*new\_node;

new\_node = (node \*)malloc(sizeof(node));

new\_node -> next = NULL;

printf("Enter data for NEW NODE: ");

scanf("%d", &new\_node->data);

return new\_node;

}

void display(node \*start){

if(start==NULL) printf("No CIRCULAR linked list present\n");

else{

node \*ptr;

ptr = start;

while(ptr->next!=start){

printf("Data = %d\t", ptr->data);

printf("Next = %d\n", ptr->next);

ptr=ptr->next;

}

printf("Data = %d\t", ptr->data);

printf("Next = %d\n", ptr->next);

}

}

void insert\_first(node \*\*start){

node \*inserting\_node;

inserting\_node = create\_node();

if(\*start == NULL){

\*start = inserting\_node;

inserting\_node->next = \*start;

}

else{

node \*ptr = \*start;

while(ptr->next != \*start){

ptr=ptr->next;

}

ptr->next = inserting\_node;

inserting\_node->next = \*start;

\*start = inserting\_node;

}

}

void insert\_last(node \*\*start){

node \*new\_2\_insert;

new\_2\_insert = create\_node();

if(\*start == NULL){

\*start = new\_2\_insert;

new\_2\_insert->next = \*start;

}

else{

node \*ptr = \*start;

while(ptr->next != \*start){

ptr = ptr->next;

}

ptr->next = new\_2\_insert;

new\_2\_insert->next = \*start;

}

}

void delete\_first(node \*\*start){

node \*ptr, \*temp;

if(\*start == NULL) printf("No linked list\n");

else if((\*start)->next == \*start){

ptr = \*start;

\*start = NULL;

free(ptr);

}

else{

ptr = \*start;

while(ptr->next != \*start){

ptr = ptr->next;

}

ptr->next = (\*start)->next;

temp = \*start;

\*start = (\*start)->next;

free(temp);

}

}

void delete\_last(node \*\*start){

node \*ptr, \*pre\_ptr, \*temp;

if(\*start == NULL) printf("No linked list\n");

else if((\*start)->next == \*start){

ptr = \*start;

\*start = NULL;

free(ptr);

}

else{

pre\_ptr=\*start;

ptr=\*start;

while(ptr->next != \*start){

pre\_ptr = ptr;

ptr = ptr->next;

}

pre\_ptr->next = \*start;

free(ptr);

}

}

void delete\_given(node \*\*start, int item){

node \*ptr, \*prev;

if(\*start==NULL) printf("No linked list\n");

else{

ptr=\*start, prev=\*start;

while(ptr->next != \*start){

if(ptr->data==item) break;

prev = ptr;

ptr = ptr->next;

}

}

if(ptr->data == item){

if(ptr == \*start) delete\_first(start);

else if(ptr->next == \*start) delete\_last(start);

else{

prev->next = ptr->next;

free(ptr);

}

}

else printf("Given item is not present\n");

}

void main(){

node \*start;

int choice, item;

while(1){

printf("Press 1 to INSERT at FIRST\n");

printf("Press 2 to INSERT at LAST \n");

printf("Press 3 to DELETE FIRST \n");

printf("Press 4 to DELETE at LAST \n");

printf("Press 5 to DELETE a given item \n");

printf("Press 6 for Display\n");

printf("Press 7 to EXIT\n");

printf("Enter Your Choice: ");

scanf("%d", &choice);

switch(choice)

{

case 1: insert\_first(&start);

break;

case 2: insert\_last(&start);

break;

case 3: delete\_first(&start);

break;

case 4: delete\_last(&start);

break;

case 5:

printf("Enter the item to be deleted: ");

scanf("%d", &item);

delete\_given(&start, item);

break;

case 6: display(start);

break;

case 7: exit(0);

break;

default: printf("Invalid!");

break;

}

}

}

**OUTPUT:**

|  |  |
| --- | --- |
| INPUT:  (Choice) | OUTPUT: |
| 1 | Enter Data for NEW NODE: 30 |
| 1 | Enter Data for NEW NODE: 20 |
| 1 | Enter Data for NEW NODE: 10 |
| 2 | Enter Data for NEW NODE: 40 |
| 2 | Enter Data for NEW NODE: 50 |
| 6 | Data= 10 Next= -2033092480  Data= 20 Next= -2033092352  Data= 30 Next= -2033091648  Data= 40 Next= -2033092416  Data= 50 Next= -2033092920 |
| 3 |  |
| 4 |  |
| 6 | Data= 20 Next= -2033092352  Data= 30 Next= -2033091648  Data= 40 Next= -2033092472 |
| 5 | Enter the item to be deleted: 30 |
| 6 | Data= 20 Next= -2033092352  Data= 40 Next= -2033092472 |

**Q: DOUBLY LINKED LIST**

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

    int data;

    struct node \*next, \*prev;

}node;

node \*create(){

    node \*new;

    new= (node \*)malloc(sizeof(node));

    printf("Enter Data for NEW NODE: ");

    scanf("%d", &new->data);

    new->next= NULL, new->prev= NULL;

    return new;

}

void display(node \*start){

    node \*ptr;

    if(start==NULL){

        printf("NO Circular Linked List!\n");

    }

    else{

        ptr= start;

        while(ptr != NULL){

            printf("Data= %d\t", ptr->data);

            printf("Prev= %d", ptr->prev);

            printf("Next= \t%d\n", ptr->next);

            ptr= ptr->next;

        }

    }

}

void insert\_first(node \*\*start){

    //creating new node to insert

    node \*new2insert;

    new2insert= create();

    //insertion algorithm

    if(\*start == NULL){

        \*start= new2insert;

    }

    else{

        new2insert->next= \*start;

        (\*start)->prev= new2insert;//since, DOUBLY Linked List

        \*start= new2insert;

    }

}

void insert\_last(node \*\*start){

    //creating new node

    node \*new2insert;

    new2insert= create();

    //insertion algorithm

    if(\*start == NULL){

        \*start= new2insert;

    }

    else{

        node \*ptr= \*start;

        while(ptr->next !=NULL){

            ptr= ptr->next;

        }

        ptr->next= new2insert;

        new2insert->prev= ptr;

    }

}

void delete\_first(node \*\*start){

    if(\*start == NULL){

        printf("No Linked list\n");

    }

    else{

        node \*temp= \*start;

        (\*start)->next->prev = NULL;

        \*start= (\*start)->next;

        free(temp);

    }

}

void delete\_last(node \*\*start){

    if(\*start == NULL){

        printf("No Linked list\n");

    }

    else{

        node \*ptr= \*start, \*prev= \*start;

        while(ptr->next != NULL){

            prev= ptr;

            ptr= ptr->next;

        }

        prev->next= NULL;

        free(ptr);

    }

}

int main(){

    node \*start;

    start= NULL;

    int choice;

    while(1){

        printf("Press 1 for Insert at FIRST POSITION of Doubly Linked List\n");

        printf("Press 2 for Insert at LAST POSITION of Doubly Linked List\n");

        printf("Press 3 for Delete at FIRST POSITION of Doubly Linked List\n");

        printf("Press 4 for Delete at LAST POSITION of Doubly Linked List\n");

        printf("Press 5 to Display The DOUBLY Linked List\n");

        printf("Press 6 for Exit\n");

        printf("Enter your Choice: ");

        scanf("%d", &choice);

        switch(choice){

            case 1:

                insert\_first(&start);

                break;

            case 2:

                insert\_last(&start);

                break;

            case 3:

                delete\_first(&start);

                break;

            case 4:

                delete\_last(&start);

                break;

            case 5:

                display(start);

                break;

            case 6:

                exit(0);

                break;

        }

    }

}

1. INPUT & OUTPUT:

Press 1 for Insert at FIRST POSITION of Doubly Linked List

Press 2 for Insert at LAST POSITION of Doubly Linked List

Press 3 for Delete at FIRST POSITION of Doubly Linked List

Press 4 for Delete at LAST POSITION of Doubly Linked List

Press 5 to Display The DOUBLY Linked List

Press 6 for Exit

Enter your Choice:

|  |  |
| --- | --- |
| INPUT:  (Choice) | OUTPUT: |
| 1 | Enter Data for NEW NODE: 30 |
| 1 | Enter Data for NEW NODE: 20 |
| 1 | Enter Data for NEW NODE: 10 |
| 2 | Enter Data for NEW NODE: 40 |
| 2 | Enter Data for NEW NODE: 50 |
| 5 | Data= 10 Prev= 0 Next= -2033092480  Data= 20 Prev= -2033092000 Next= -2033092352  Data= 30 Prev= -2033092480 Next= -2033091648  Data= 40 Prev= -2033092352 Next= -2033092416  Data= 50 Prev= -2033091648 Next= 0 |
| 3 |  |
| 4 |  |
| 5 | Data= 20 Prev= 0 Next= -2033092352  Data= 30 Prev= -2033092480 Next= -2033091648  Data= 40 Prev= -2033092352 Next= 0 |

**Q: Implementation of a Stack.**

#include <stdio.h>

#include <process.h>

#include <stdlib.h>

#define MAX 50

int top = -1, stack[MAX];

void push();

void pop();

void display();

void main()

{

int ch;

while (1)

{

printf("\n\*\*\* Stack Menu \*\*\*");

printf("\n\n1.Push\n2.Pop\n3.Display\n4.Exit");

printf("\n\nEnter your choice(1-4):");

scanf("%d", &ch);

switch (ch)

{

case 1:

push();

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

exit(0);

default:

printf("\nWrong Choice!!");

}

}

}

void push()

{

int val;

if (top == MAX - 1) printf("\nStack is full!!");

else{

printf("\nEnter element to push:");

scanf("%d", &val);

top = top + 1;

stack[top] = val;

}

}

void pop()

{

if (top == -1) printf("\nStack is empty!!");

else{

printf("\nDeleted element is %d", stack[top]);

top = top - 1;

}

}

void display()

{

int i;

if (top == -1) printf("\nStack is empty!!");

else

{

printf("\nStack is...\n");

for (i = top; i >= 0; --i)

printf("%d\n", stack[i]);

}

}

**OUTPUT:**

|  |  |
| --- | --- |
| INPUT:  (Choice) | OUTPUT: |
|  | \*\*\* Stack Menu \*\*\*  1.Push  2.Pop  3.Display  4.Exit  Enter your choice(1-4): |
| 1 | Enter element to push: 65 |
| 1 | Enter element to push: 25 |
| 1 | Enter element to push: 39 |
| 3 | Stack is...  39  25  65 |
| 2 | Deleted element is 39 |
| 3 | Stack is...  25  65 |

**Q: Postfix Evaluation.**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100

// Stack implementation

int stack[MAX\_SIZE];

int top = -1;

void push(int item) {

if (top >= MAX\_SIZE - 1) {

printf("Stack Overflow\n");

return;

}

top++;

stack[top] = item;

}

int pop() {

if (top < 0) {

printf("Stack Underflow\n");

return -1;

}

int item = stack[top];

top--;

return item;

}

int is\_operator(char sym) {

if (sym == '+' || sym == '-' || sym == '\*' || sym == '/' || sym == '%' || sym == '^')

return 1;

return 0;

}

int evaluate(char \*exp) {

int i = 0;

char sym = exp[i];

int b, a, res;

while (sym != '\0') {

if (sym >= '0' && sym <= '9') {

int num = sym - '0';

push(num);

}

else if (is\_operator(sym)) {

a = pop();

b = pop();

switch (sym) {

case '+': res = b + a;

break;

case '-': res = b - a;

break;

case '\*': res = b \* a;

break;

case '/': res = b / a;

break;

case '%': res = b % a;

break;

case '^': res = b ^ a;

break;

}

push(res);

}

i++;

sym = exp[i];

}

res = pop();

return res;

}

int main(){

char exp[50];

printf("Enter your postfix equation: ");

gets(exp);

int res = evaluate(exp);

printf("Result= %d\n", res);

return 0;

}

**OUTPUT:**

Enter your postfix equation: 567+\*8-

Result= 57

**Q: Tower of Hanoi.**

#include <stdio.h>

void towerOfHanoi(int n, char from\_rod, char to\_rod, char aux\_rod)

{

if (n == 1) {

printf("\n Move disk 1 from rod %c to rod %c", from\_rod, to\_rod);

return;

}

towerOfHanoi(n-1, from\_rod, aux\_rod, to\_rod);

printf("\n Move disk %d from rod %c to rod %c", n, from\_rod, to\_rod);

towerOfHanoi(n-1, aux\_rod, to\_rod, from\_rod);

}

int main() {

int n;

printf(“Enter the number of disks: “);

scanf(“%d”, &n);

towerOfHanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods

return 0;

}

**OUTPUT:**

Move disk 1 from rod A to rod B

Move disk 2 from rod A to rod C

Move disk 1 from rod B to rod C

Move disk 3 from rod A to rod B

Move disk 1 from rod C to rod A

Move disk 2 from rod C to rod B

Move disk 1 from rod A to rod B

Move disk 4 from rod A to rod C

Move disk 1 from rod B to rod C

Move disk 2 from rod B to rod A

Move disk 1 from rod C to rod A

Move disk 3 from rod B to rod C

Move disk 1 from rod A to rod B

Move disk 2 from rod A to rod C

Move disk 1 from rod B to rod C

**Q: Infix to Postfix expression.**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct stack {

int size;

int top;

char \*arr;

};

int stackTop(struct stack\* sp){

return sp->arr[sp->top];

}

int isEmpty(struct stack \*ptr) {

if (ptr->top == -1) {

return 1;

}

else {

return 0;

}

}

int isFull(struct stack \*ptr) {

if (ptr->top == ptr->size - 1) {

return 1;

}

else {

return 0;

}

}

void push(struct stack\* ptr, char val){

if(isFull(ptr)){

printf("Stack Overflow! Cannot push %d to the stack\n", val);

}

else {

ptr->top++;

ptr->arr[ptr->top] = val;

}

}

char pop(struct stack\* ptr){

if(isEmpty(ptr)){

printf("Stack Underflow! Cannot pop from the stack\n");

return -1;

}

else{

char val = ptr->arr[ptr->top];

ptr->top--;

return val;

}

}

int precedence(char ch){

if(ch == '\*' || ch=='/')

return 3;

else if(ch == '+' || ch=='-')

return 2;

else

return 0;

}

int isOperator(char ch){

if(ch=='+' || ch=='-' ||ch=='\*' || ch=='/')

return 1;

else

return 0;

}

char\* infixToPostfix(char\* infix){

struct stack \* sp = (struct stack \*) malloc(sizeof(struct stack));

sp->size = 10;

sp->top = -1;

sp->arr = (char \*) malloc(sp->size \* sizeof(char));

char \* postfix = (char \*) malloc((strlen(infix)+1) \* sizeof(char));

int i=0; // Track infix traversal

int j = 0; // Track postfix addition

while (infix[i]!='\0')

{

if(!isOperator(infix[i])){

postfix[j] = infix[i];

j++;

i++;

}

else{

if(precedence(infix[i])> precedence(stackTop(sp))){

push(sp, infix[i]);

i++;

}

else{

postfix[j] = pop(sp);

j++;

}

}

}

while (!isEmpty(sp)) {

postfix[j] = pop(sp);

j++;

}

postfix[j] = '\0';

return postfix;

}

int main() {

char \* infix = "x-y/z-k\*d";

printf("postfix is %s", infixToPostfix(infix));

return 0;

}

**OUTPUT:**

postfix is xyz/-kd\*-