27/08/23

**PROGRAM1: To implement linked list data structure.**

**Code:**

#include<stdlib.h>

#include <stdio.h>

void create();

void display();

void insert\_begin();

void insert\_end();

void insert\_pos();

void delete\_begin();

void delete\_end();

void delete\_pos();

struct node

{

int info;

struct node \*next;

};

struct node \*start=NULL;

int main()

{

int choice;

while(1){

printf("\n1.Create\n");

printf("2.Display \n");

printf("3.Insert at the beginning \n");

printf("4.Insert at the end \n");

printf("5.Insert at specified position \n");

printf("6.Delete from beginning \n");

printf("7.Delete from the end \n");

printf("8.Delete from specified position \n");

printf("9.Exit \n");

printf("Enter your choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

create();

break;

case 2:

display();

break;

case 3:

insert\_begin();

break;

case 4:

insert\_end();

break;

case 5:

insert\_pos();

break;

case 6:

delete\_begin();

break;

case 7:

delete\_end();

break;

case 8:

delete\_pos();

break;

case 9:

exit(0);

break;

default:

printf("Wrong Choice:\n");

break;

}

}

return 0;

}

void create()

{

struct node \*temp,\*ptr;

temp=(struct node \*)malloc(sizeof(struct node));

if(temp==NULL)

{

printf("Out of Memory Space:\n");

exit(0);

}

printf("Enter the data value for the node:");

scanf("%d",&temp->info);

temp->next=NULL;

if(start==NULL)

{

start=temp;

}

else

{

ptr=start;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=temp;

}

}

void display()

{

struct node \*ptr;

if(start==NULL)

{

printf("List is empty:\n");

return;

}

else

{

ptr=start;

printf("The List elements are:\n");

while(ptr!=NULL)

{

printf("%d ",ptr->info );

ptr=ptr->next ;

}

printf("\n");

}

}

void insert\_begin()

{

struct node \*temp;

temp=(struct node \*)malloc(sizeof(struct node));

if(temp==NULL)

{

printf("Out of Memory Space:\n");

return;

}

printf("Enter the data value for the node:\n" );

scanf("%d",&temp->info);

temp->next =NULL;

if(start==NULL)

{

start=temp;

}

else

{

temp->next=start;

start=temp;

}

}

void insert\_end()

{

struct node \*temp,\*ptr;

temp=(struct node \*)malloc(sizeof(struct node));

if(temp==NULL)

{

printf("Out of Memory Space:\n");

return;

}

printf("Enter the data value for the node:\n" );

scanf("%d",&temp->info );

temp->next =NULL;

if(start==NULL)

{

start=temp;

}

else

{

ptr=start;

while(ptr->next !=NULL)

{

ptr=ptr->next ;

}

ptr->next =temp;

}

}

void insert\_pos()

{

struct node \*ptr,\*temp;

int i,pos;

temp=(struct node \*)malloc(sizeof(struct node));

if(temp==NULL)

{

printf("Out of Memory Space:\n");

return;

}

printf("Enter the position for the new node to be inserted:\n");

scanf("%d",&pos);

printf("Enter the data value of the node:");

scanf("%d",&temp->info) ;

temp->next=NULL;

if(pos==0)

{

temp->next=start;

start=temp;

}

else

{

for(i=0,ptr=start;i<pos-1;i++) { ptr=ptr->next;

if(ptr==NULL)

{

printf("Position not found.");

return;

}

}

temp->next =ptr->next ;

ptr->next=temp;

}

}

void delete\_begin()

{

struct node \*ptr;

if(ptr==NULL)

{

printf("List is Empty\n");

return;

}

else

{

ptr=start;

start=start->next ;

printf("The deleted element is :%d\n",ptr->info);

free(ptr);

}

}

void delete\_end()

{

struct node \*temp,\*ptr;

if(start==NULL)

{

printf("List is Empty:\n");

exit(0);

}

else if(start->next ==NULL)

{

ptr=start;

start=NULL;

printf("The deleted element is:%d\n",ptr->info);

free(ptr);

}

else

{

ptr=start;

while(ptr->next!=NULL)

{

temp=ptr;

ptr=ptr->next;

}

temp->next=NULL;

printf("The deleted element is:%d\n",ptr->info);

free(ptr);

}

}

void delete\_pos()

{

int i,pos;

struct node \*temp,\*ptr;

if(start==NULL)

{

printf("The List is Empty\n");

exit(0);

}

else

{

printf("Enter the position of the node to be deleted:\n");

scanf("%d",&pos);

if(pos==0)

{

ptr=start;

start=start->next ;

printf("The deleted element is:%d\n",ptr->info );

free(ptr);

}

else

{

ptr=start;

for(i=0;i<pos;i++) { temp=ptr; ptr=ptr->next ;

if(ptr==NULL)

{

printf("Position not Found.\n");

return;

}

}

temp->next =ptr->next ;

printf("The deleted element is:%d\n",ptr->info );

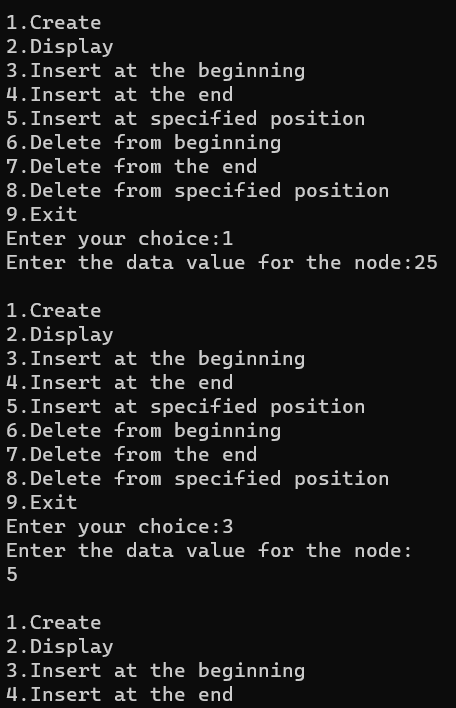
free(ptr);

}

}

}

**Sample Input&Output:**

****

**PROGRAM2: To merge two lists**

**Code:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node \*next;

} Node;

Node \*createNode(int data) {

Node \*newNode = (Node \*)malloc(sizeof(Node));

newNode->data = data;

newNode->next = NULL;

return newNode;

}

Node \*mergeLists(Node \*list1, Node \*list2) {

if (list1 == NULL)

return list2;

if (list2 == NULL)

return list1;

Node \*mergedList = NULL;

Node \*current = NULL;

if (list1->data <= list2->data) {

mergedList = list1;

list1 = list1->next;

} else {

mergedList = list2;

list2 = list2->next;

}

current = mergedList;

while (list1 != NULL && list2 != NULL) {

if (list1->data <= list2->data) {

current->next = list1;

list1 = list1->next;

} else {

current->next = list2;

list2 = list2->next;

}

current = current->next;

}

if (list1 != NULL)

current->next = list1;

else

current->next = list2;

return mergedList;

}

void printList(Node \*list) {

while (list != NULL) {

printf("%d -> ", list->data);

list = list->next;

}

printf("NULL\n");

}

int main() {

Node \*list1 = NULL, \*list2 = NULL;

int n1, n2, data;

printf("Enter the number of elements in list 1: ");

scanf("%d", &n1);

printf("Enter the elements of list 1:\n");

for (int i = 0; i < n1; i++) {

scanf("%d", &data);

Node \*newNode = createNode(data);

newNode->next = list1;

list1 = newNode;

}

printf("Enter the number of elements in list 2: ");

scanf("%d", &n2);

printf("Enter the elements of list 2:\n");

for (int i = 0; i < n2; i++) {

scanf("%d", &data);

Node \*newNode = createNode(data);

newNode->next = list2;

list2 = newNode;

}

Node \*mergedList = mergeLists(list1, list2);

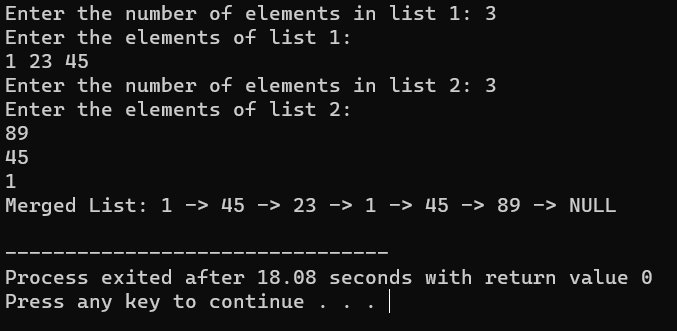
printf("Merged List: ");

printList(mergedList);

return 0;

}

**Sample Input & Output:**

****

**PROGRAM3: To implement stack operations**

**Code:**

#include <stdio.h>

#include <stdlib.h>

void push();

void pop();

void show();

int \*stack;

int top = -1;

int main() {

int size;

printf("Enter size of stack: ");

scanf("%d", &size);

stack = (int \*)malloc(size \* sizeof(int));

int choice;

while (1) {

printf("\nPerform operations on the stack:");

printf("\n1.Push the element\n2.Pop the element\n3.Show\n4.End");

printf("\n\nEnter the choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

push();

break;

case 2:

pop();

break;

case 3:

show();

break;

case 4:

exit(0);

default:

printf("\nInvalid choice!!");

}

}

}

void push() {

int x;

if (top == sizeof(stack) - 1) {

printf("\nOverflow!!");

} else {

printf("\nEnter the element to be added onto the stack: ");

scanf("%d", &x);

top = top + 1;

stack[top] = x;

}

}

void pop() {

if (top == -1) {

printf("\nUnderflow!!");

} else {

printf("\nPopped element: %d", stack[top]);

top = top - 1;

}

}

void show() {

if (top == -1) {

printf("\nUnderflow!!");

} else {

printf("\nElements present in the stack: \n");

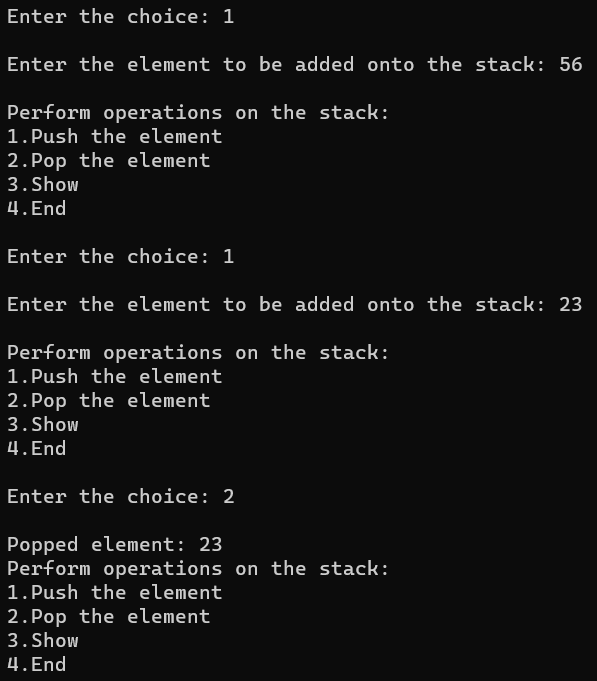
for (int i = top; i >= 0; --i)

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

}

}

**Sample Input & Output:**

****

**PROGRAM4: To implement queue operations**

**Code:**#include <stdio.h>

# define SIZE 100

void enqueue();

void dequeue();

void show();

int inp\_arr[SIZE];

int Rear = - 1;

int Front = - 1;

main()

{

int ch;

while (1)

{

printf("1.Enqueue Operation\n");

printf("2.Dequeue Operation\n");

printf("3.Display the Queue\n");

printf("4.Exit\n");

printf("Enter your choice of operations : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

show();

break;

case 4:

return 0;

default:

printf("Incorrect choice \n");

}

}

}

void enqueue()

{

int insert\_item;

if (Rear == SIZE - 1)

printf("Overflow \n");

else

{

if (Front == - 1)

Front = 0;

printf("Element to be inserted in the Queue : ");

scanf("%d", &insert\_item);

Rear = Rear + 1;

inp\_arr[Rear] = insert\_item;

}

}

void dequeue()

{

if (Front == - 1 || Front > Rear)

{

printf("Underflow \n");

return ;

}

else

{

printf("Element deleted from the Queue: %d\n", inp\_arr[Front]);

Front = Front + 1;

}

}

void show()

{

if (Front == - 1)

printf("Empty Queue \n");

else

{

printf("Queue: \n");

for (int i = Front; i <= Rear; i++)

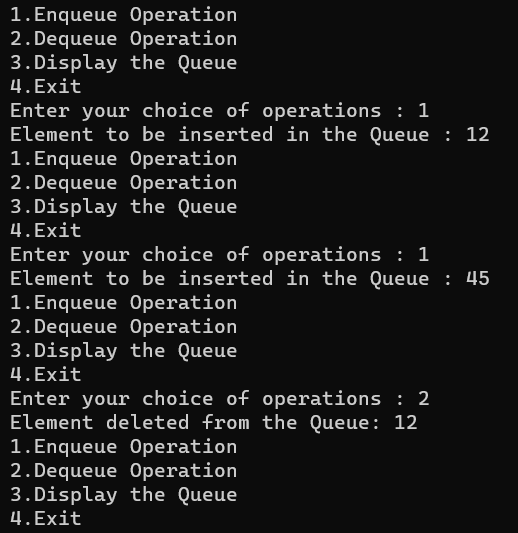
printf("%d ", inp\_arr[i]);

printf("\n");

}

}

**Sample Input & Output:**

****

**PROGRAM5: To convert infix to postfix using stack**

**Code:**

#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)

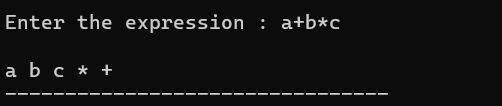
{

printf("%c ",pop());

}return 0;

}

**Sample Input & Output:**

****

**PROGRAM6: To evaluate postfix expression**

**Code:**

#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 symbol) {

if (symbol == '+' || symbol == '-' || symbol == '\*' || symbol == '/') {

return 1;

}

return 0;

}

int evaluate(char\* expression) {

int i = 0;

char symbol = expression[i];

int operand1, operand2, result;

while (symbol != '\0') {

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

int num = symbol - '0';

push(num);

}

else if (is\_operator(symbol)) {

operand2 = pop();

operand1 = pop();

switch(symbol) {

case '+': result = operand1 + operand2; break;

case '-': result = operand1 - operand2; break;

case '\*': result = operand1 \* operand2; break;

case '/': result = operand1 / operand2; break;

}

push(result);

}

i++;

symbol = expression[i];

}

result = pop();

return result;

}

int main() {

char expression[100];

printf("Enter the expression: ");

scanf("%s",expression);

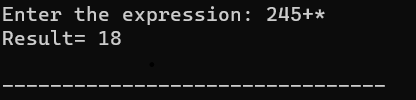
int result = evaluate(expression);

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

return 0;

}

**Sample Input & Output:**

****

**PROGRAM7: To implement tree traversals**

**Code:**

#include <stdio.h>

#include <stdlib.h>

// Definition of a binary tree node

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

// Function to create a new node

struct Node\* newNode(int data) {

struct Node\* node = (struct Node\*)malloc(sizeof(struct Node));

node->data = data;

node->left = NULL;

node->right = NULL;

return node;

}

// Function to perform in-order traversal

void inOrderTraversal(struct Node\* root) {

if (root != NULL) {

inOrderTraversal(root->left);

printf("%d ", root->data);

inOrderTraversal(root->right);

}

}

// Function to perform pre-order traversal

void preOrderTraversal(struct Node\* root) {

if (root != NULL) {

printf("%d ", root->data);

preOrderTraversal(root->left);

preOrderTraversal(root->right);

}

}

// Function to perform post-order traversal

void postOrderTraversal(struct Node\* root) {

if (root != NULL) {

postOrderTraversal(root->left);

postOrderTraversal(root->right);

printf("%d ", root->data);

}

}

int main() {

struct Node\* root = NULL;

// Construct the binary tree

root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

root->left->left = newNode(4);

root->left->right = newNode(5);

printf("In-order traversal:\n");

inOrderTraversal(root);

printf("\n");

printf("Pre-order traversal:\n");

preOrderTraversal(root);

printf("\n");

printf("Post-order traversal:\n");

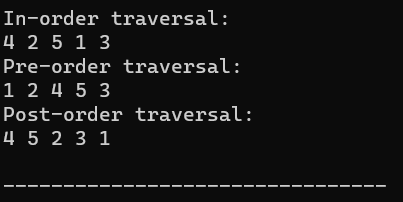
postOrderTraversal(root);

printf("\n");

return 0;

}

**Sample Input & Output:**

****