**Programs on functions**

**22) Write the sum of natural numbers using function:**

#include <stdio.h>

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;

}

**OUTPUT:**

**Enter a positive integer: 20**

**Sum = 210**

**23) Factorial using recursion function:**

#include<stdio.h>

long int multiplyNumbers(int n);

int main() {

int n;

printf("Enter a positive integer: ");

scanf("%d",&n);

printf("Factorial of %d = %ld", n, multiplyNumbers(n));

return 0;

}

long int multiplyNumbers(int n) {

if (n>=1)

return n\*multiplyNumbers(n-1);

else

return 1;

}

**OUTPUT:**

**Enter a positive integer: 6**

**Factorial of 6 = 720**

**24) Generating fibonacci series using function:**

#include<stdio.h>

int fibonacci(int);

int main(void)

{

int terms;

printf("Enter terms: ");

scanf("%d", &terms);

for(int n = 0; n < terms; n++)

{

printf("%d ", fibonacci(n));

}

return 0;

}

int fibonacci(int num)

{

if(num == 0 || num == 1)

{

return num;

}

else

{

return fibonacci(num-1) + fibonacci(num-2);

}

}

**OUTPUT:**

**Enter terms: 6**

**0 1 1 2 3 5**

**Programs on Structure, String and Pointers**

**25) Write a C Program using structure for entering details of the five students as name, admission number, Date of birth, department and display all the details:**

#include <stdio.h>

struct student

{

char name[50];

int roll, DOB;

} s[100];

int main()

{

int i,n;

struct student s[100];

printf("Enter total of students:\n");

scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("\n Enter information of student %d:\n",i+1);

printf("Enter name: ");

scanf("%s", s[i].name);

printf("Enter roll number: ");

scanf("%d", &s[i].roll);

printf("enter date of birth:");

scanf("%d", &s[i].DOB);

}

printf("Displaying Information:\n");

for(i=0;i<n;i++)

{

printf("\n %d no. student info\n",i+1);

printf("\tName:%s\n ",s[i].name);

printf("\t Roll number: %d\n",s[i].roll);

printf("\t date of birth: %d\n",s[i].DOB);

}

return 0;

}

**OUTPUT:**

**Enter total no.of students:**

**5**

**Enter information of student 1:**

**Enter name: pinky**

**Enter roll number: 552**

**Enter date of birth: 30-09-2001**

**Enter information of student 2:**

**Enter name: navi**

**Enter roll number: 553**

**Enter date of birth: 01-08-2000**

**Enter information of student 3:**

**Enter name: sony**

**Enter roll number: 554**

**Enter date of birth: 11-10-2001**

**Enter information of student 4:**

**Enter name: vinny**

**Enter roll number: 555**

**Enter date of birth: 10-04-2001**

**Enter information of student 5:**

**Enter name: ria**

**Enter roll number: 556**

**Enter date of birth: 22-03-2002**

1. **No. student info**

**Name: pinky**

**Roll number: 552**

**Date of birth: 30-09-2001**

1. **No. student info**

**Name: navi**

**Roll number: 553**

**Date of birth: 01-08-2000**

1. **No. student info**

**Name: sony**

**Roll number: 554**

**Date of birth: 11-10-2001**

1. **No. student info**

**Name: vinny**

**Roll number: 555**

**Date of birth: 10-04-2001**

1. **No. student info**

**Name: ria**

**Roll number: 556**

**Date of birth: 22-03-2002**

**26)Write a C program to find length of string using pointers:**

#include<stdio.h>

#include<conio.h>

int string\_ln(char\*);

void main() {

char str[20];

int length;

clrscr();

printf("\nEnter any string : ");

gets(str);

length = string\_ln(str);

printf("The length of the given string %s is : %d", str, length);

getch();

}

int string\_ln(char\*p) /\* p=&str[0] \*/

{

int count = 0;

while (\*p != '\0') {

count++;

p++;

}

return count;

}

**OUTPUT:**

**Enter the String : saisushma**

**Length of the given string pritesh is : 9**

**27)Write a C program to copy one string to another using pointers:**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum size of the string

int main()

{

char text1[MAX\_SIZE], text2[MAX\_SIZE];

char \* str1 = text1;

char \* str2 = text2;

/\* Input string from user \*/

printf("Enter any string: ");

gets(text1);

/\* Copy text1 to text2 character by character \*/

while(\*(str2++) = \*(str1++));

printf("First string = %s\n", text1);

printf("Second string = %s\n", text2)

return 0;

}

**OUTPUT:**

**Enter any string: sai sushma**

**First string = sai sushma**

**Second string = sai sushma**

**28)Write a C program to compare two strings using pointers:**

#include <stdio.h>

#define MAX 100

int main()

{

char str1[MAX]={0};

char str2[MAX]={0};

int loop;

int flag=1;

char \*pStr1=str1;

char \*pStr2=str2;

printf("Enter string 1: ");

scanf("%[^\n]s",pStr1);

printf("Enter string 2: ");

getchar(); //read & ignore extra character

scanf("%[^\n]s",pStr2);

printf("string1: %s\nstring2: %s\n",pStr1,pStr2);

for(loop=0; (\*(pStr1+loop))!='\0'; loop++)

{

if(\*(pStr1+loop) != \*(pStr2+loop))

{

flag=0;

break;

}

}

if(flag)

printf("Strings are same.\n");

else

printf("Strings are not same.\n");

return 0;

}

**OUTPUT:**

**Enter string 1: sai sushma**

**Enter string 2: sai Eesha**

**string 1: sai sushma**

**string 2: sai Eesha**

**Strings are not same**

**29) Write a C program to find the reverse of a string recursively and non-recursively:**

1. **Using recursion:**

#include <stdio.h>

void reversestring();

int main() {

printf("Enter a string: ");

reversestring();

return 0;

}

void reversestring() {

char c;

scanf("%c", &c);

if (c != '\n') {

reversestring();

printf("%c", c);

}

}

**OUTPUT:**

**Enter a string: sai sushma**

**amhsus ias**

**b) Using non-recursive:**

#include <stdio.h>

int main()

{

char s[1000], r[1000];

int begin, end, count = 0;

printf("Input a string\n");

gets(s);

while (s[count] != '\0')

count++;

end = count - 1;

for (begin = 0; begin < count; begin++) {

r[begin] = s[end];

end--;

}

r[begin] = '\0';

printf("%s\n", r);

return 0;

}

**OUTPUT:**

**Input a string: sai sushma**

**amhsus ias**

**Programs on Trees and Graphs**

**30)Create a binary tree and output the data with 3 tree traversals:**

**a)queue using linked list:**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*link;

}\*front=NULL,\*rear=NULL,\*temp,\*ptr;

void nque(int a)

{

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

if(ptr==NULL)

{

printf("Memory allocation fail\n");

return;

}

ptr->data=a;

if(front==NULL)

{

front=ptr;

rear=ptr;

rear->link = NULL;

}

else

{

rear->link=ptr;

rear=ptr;

rear->link=NULL;

}

}

void deque()

{

temp=front;

if(temp==NULL)

{

printf("No elements to delete\n");

return;

}

else

{

if(front==rear)

{

printf("%d is deleted\n",front->data);

free(front);

front=rear=NULL;

}

else

{

printf("%d is deleted\n",temp->data);

temp=temp->link;

free(front);

front=temp;

}

}

}

void display()

{

temp=front;

if(temp==NULL)

{

printf("No elements in the queue\n");

}

else

{

while(temp!=NULL)

{

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

temp=temp->link;

}

}

}

void main()

{

int c,a;

while(1)

{

printf("\n1.nque,2.deque,3.display,4.exit\n");

printf("Enter you choice: ");

scanf("%d",&c);

switch(c)

{

case 1:

printf("Enter a number to insert:-");

scanf("%d",&a);

nque(a);

break;

case 2:

deque();

break;

case 3:

display();

break;

case 4:

exit(0);

}

}

}

**OUTPUT:**

**1.enque**

**2. Deque**

**3.display**

**4.exit**

**Enter your choice:1**

**Enter number to insert:21**

**1.enque**

**2. Deque**

**3.display**

**4.exit**

**Enter your choice:4**

**b) Stack using linked list:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int info;

struct node \*ptr;

}\*top,\*top1,\*temp;

int count=0;

void create()

{

top = NULL;

}

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 peep()

{

return(top->info);

}

void main()

{

int no, ch, e;

printf("\n 1 - Push");

printf("\n 2 - Pop");

printf("\n 3 - Peep");

printf("\n 4 - Display");

printf("\n 5 - Exit");

create();

while (1)

{

printf("\nEnter 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

{

e = peep();

printf("\n Top element : %d", e);

}

break;

case 4:

display();

break;

case 5:

exit(0);

default :

printf("Wrong choice, Please enter correct choice ");

break;

}

}

}

**OUTPUT:**

**1-push**

**2-pop**

**3-peep**

**4-display**

**5-exit**

**Enter the choice:1**

**Enter data: 1 2 3 4**

**Enter the choice:5**

**31) Create a Binary Search Tree(BST) and search for a given value in BST:**

/\*Tree traversal in C

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node\* left;

struct node\* right;

};

// Inorder traversal

void inorder(struct node\* root) {

if (root == NULL) return;

inorder(root->left);

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

inorder(root->right);

}

// Preorder traversal

void preorder(struct node\* root) {

if (root == NULL) return;

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

preorder(root->left);

preorder(root->right);

}

// Postorder traversal

void postorder(struct node\* root) {

if (root == NULL) return;

postorder(root->left);

postorder(root->right);

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

}

// Create a new Node

struct node\* createNode(value) {

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

newNode->data = value;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

// Insert on the left of the node

struct node\* insertLeft(struct node\* root, int value) {

root->left = createNode(value);

return root->left;

}

// Insert on the right of the node

struct node\* insertRight(struct node\* root, int value) {

root->right = createNode(value);

return root->right;

}

int main() {

struct node\* root = createNode(1);

insertLeft(root, 21);

insertRight(root, 19);

insertLeft(root->left, 4);

insertRight(root->left, 16);

printf("Inorder traversal \n");

inorder(root);

printf("\nPreorder traversal \n");

preorder(root);

printf("\Postorder traversal \n");

postorder(root);

}

**OUTPUT:**

**Inorder traversal**

**4 ->21 ->16 ->1 ->19 ->**

**Preorder traversal**

**1 ->21 ->4 ->16 ->19 ->postorder traversal**

**4 ->16 ->21 ->19 ->1 ->**

**33) Write a program to find All-to-all Shortest paths in a Graph.**

#include<stdio.h>

int min(int,int);

void floyds(int p[10][10],int n)

{

int i,j,k;

for(k=1;k<=n;k++)

for(i=1;i<=n;i++)

for(j=1;j<=n;j++)

if(i==j)

p[i][j]=0;

else

p[i][j]=min(p[i][j],p[i][k]+p[k][j]);

}

int min(int a,int b)

{

if(a<b)

return(a);

else

return(b);

}

void main()

{

int p[10][10],w,n,e,u,v,i,j;;

printf("\n Enter the number of vertices:");

scanf("%d",&n);

printf("\n Enter the number of edges:\n");

scanf("%d",&e);

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

p[i][j]=999;

}

for(i=1;i<=e;i++)

{

printf("\n Enter the end vertices of edge%d with its weight \n",i);

scanf("%d%d%d",&u,&v,&w);

p[u][v]=w;

}

printf("\n Matrix of input data:\n");

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

printf("%d \t",p[i][j]);

printf("\n");

}

floyds(p,n);

printf("\n Transitive closure:\n");

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

printf("%d \t",p[i][j]);

printf("\n");

}

printf("\n The shortest paths are:\n");

for(i=1;i<=n;i++)

for(j=1;j<=n;j++)

{

if(i!=j)

printf("\n <%d,%d>=%d",i,j,p[i][j]);

}

}

**OUTPUT**:

**Enter the end vertices of edge1 with its weight**

**1 2 20**

**Enter the end vertices of edge2 with its weight**

**3 4 40**

**Enter the end vertices of edge3 with its weight**

**2 3 15**

**Enter the end vertices of edge4 with its weight**

**1 3 10**

**Matrix of input data:**

**999 20 10 999**

**999 0 15 55**

**999 999 0 40**

**999 999 999 0**

**Transitive closure:**

**0 20 10 50**

**999 0 15 55**

**999 999 0 40**

**999 999 999 0**

**The shortest paths are:**

**<1,2>=20**

**<1,3>=10**

**<1,4>=50**

**<2,1>=999**

**<2,3>=15**

**<2,4>=55**

**<3,1>=999**

**<3,2>=999**

**<3,4>=40**

**<4,1>=999**

**<4,2>=999**

**<4,3>=999**

**Programs on Linked-list:**

**34) Write a C program to implement the STACK operation using array as a data structure. Users must be given the following choices to perform relevant tasks.**

**a. Push an element on to the STACK.**

**b. Pop and element from the STACK.**

**c. Peek the STACK.**

**d. Display the STACK.**

**e. Exit the program.**

#include<stdio.h>

#define SIZE 10

void push(int);

void pop();

void peek();

void display();

int stack[SIZE], top = -1;

void main()

{

int value, choice;

while(1){

printf("\n\n\*\*\*\*\* MENU \*\*\*\*\*\n");

printf("1. Push\n2. Pop\n3. Peek \n 4. Display 5. Exit");

printf("\nEnter your choice: ");

scanf("%d",&choice);

switch(choice){

case 1: printf("Enter the value to be insert: ");

scanf("%d",&value);

push(value);

break;

case 2: pop();

break;

case 3: peek();

break;

case 4: display();

break;

case 5: exit(0);

default: printf("\nWrong selection!!! Try again!!!");

}

}

}

void push(int value){

if(top == SIZE-1)

printf("\nStack is Full!!! Insertion is not possible!!!");

else{

top++;

stack[top] = value;

printf("\nInsertion success!!!");

}

}

void pop(){

if(top == -1)

printf("\nStack is Empty!!! Deletion is not possible!!!");

else{

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

top--;

}

}

void peek(){

if(top == -1)

printf("\nStack is Empty!!!");

else{

int i;

printf("\nStack top most element is: %d\n",stack[top]);

}

}

void display(){

if(top == -1)

printf("\nStack is Empty!!!");

else{

int i;

printf("\nStack elements are:\n");

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

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

}

}

**OUTPUT:**

1. **Push**
2. **Pop**
3. **Peek**
4. **Display**
5. **Exit**

**Enter your choice: 1**

**Enter the value to be inserted: 34**

**Insertion success!!!**

1. **Push**
2. **Pop**
3. **Peek**
4. **Display**
5. **Exit**

**Enter your choice: 2**

**Deleted : 34**

1. **Push**
2. **Pop**
3. **Peek**
4. **Display**
5. **Exit**

**Enter your choice: 4**

**Stack is Empty!!!**

1. **Push**
2. **Pop**
3. **Peek**
4. **Display**
5. **Exit**

**Enter your choice:**

**56**

**Wrong selection!!! Try again!!!**

1. **Push**
2. **Pop**
3. **Peek**
4. **Display**
5. **Exit**

**Enter your choice: 5**

**35) Write a C program to reverse a string using STACK:**

#include <stdio.h>

#include <string.h>

#define max 100

int top,stack[max];

void push(char x){

// Inserting Element in stack(push)

if(top == max-1){

printf("stack overflow");

} else {

stack[++top]=x;

}

}

void pop(){

// Removing element from stack(pop)

printf("%c",stack[top--]);

}

main()

{

char str[]="sai sushma";

int len = strlen(str);

int i;

for(i=0;i<len;i++)

push(str[i]);

for(i=0;i<len;i++)

pop();

}

**Output:**

*amhsus ias*

**36) Write a C program to convert the given infix expression to post-fix expression using STACK:**

// C program to convert infix expression to postfix

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

struct Stack

{

int top;

unsigned capacity;

int\* array;

};

struct Stack\* createStack( unsigned capacity )

{

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

if (!stack)

return NULL;

stack->top = -1;

stack->capacity = capacity;

stack->array = (int\*) malloc(stack->capacity \* sizeof(int));

if (!stack->array)

return NULL;

return stack;

}

int isEmpty(struct Stack\* stack)

{

return stack->top == -1 ;

}

char peek(struct Stack\* stack)

{

return stack->array[stack->top];

}

char pop(struct Stack\* stack)

{

if (!isEmpty(stack))

return stack->array[stack->top--] ;

return '$';

}

void push(struct Stack\* stack, char op)

{

stack->array[++stack->top] = op;

}

int isOperand(char ch)

{

return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');

}

int Prec(char ch)

{

switch (ch)

{

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

}

return -1;

}

int infixToPostfix(char\* exp)

{

int i,stack;

struct Stack\* stack = createStack(strlen(exp));

if(!stack) /

return -1 ;

for (i = 0, k = -1; exp[i]; ++i)

{

if (isOperand(exp[i]))

exp[++k] = exp[i];

else if (exp[i] == '(')

push(stack, exp[i]);

else if (exp[i] == ')')

{

while (!isEmpty(stack) && peek(stack) != '(')

exp[++k] = pop(stack);

if (!isEmpty(stack) && peek(stack) != '(')

return -1;

else

pop(stack);

}

else

{

while (!isEmpty(stack) && Prec(exp[i]) <= Prec(peek(stack)))

exp[++k] = pop(stack);

push(stack, exp[i]);

}

}

while (!isEmpty(stack))

exp[++k] = pop(stack );

exp[++k] = '\0';

printf( "%s", exp );

}

int main()

{

char exp[] = "a+b\*(c^d-e)^(f+g\*h)-i";

infixToPostfix(exp);

return 0;

}

**Output:**

**abcd^e-fgh\*+^\*+i-**

**37)Write a C program to convert the given in-fix expression to pre-fix expression using STACK:**

#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<string.h>

#define SIZE 100

char stack[SIZE];

int top = -1;

void push(char c);

char pop();

int isoperator(char symbol);

int precedence(char symbol);

void InfixToPrefix(char infix\_exp[], char prefix\_exp[]);

void main()

{

char infix[SIZE], prefix[SIZE];

printf("\n\n Enter Infix expression : ");

gets(infix);

InfixToPrefix(infix,prefix);

printf("\n Prefix Expression: ");

puts(prefix);

}

void InfixToPrefix(char infix\_exp[], char prefix\_exp[])

{

int i, j, k, pos, len;

char item, x, rev[SIZE];

pos=0;

len=strlen(infix\_exp);

for(k=len-1;k>=0;k--)

{

rev[pos]=infix\_exp[k];

pos++;

}

rev[pos]='\0';

strcpy(infix\_exp,rev);

// Make Every “ ( ” as “ ) ” and every “ ) ” as “ ( ”

for(i=0; infix\_exp[i]!='\0'; i++)

{

if(infix\_exp[i] == ')')

infix\_exp[i] = '(';

else if(infix\_exp[i] == '(')

infix\_exp[i] = ')';

}

push('(');

strcat(infix\_exp,")");

i=0;

j=0;

item=infix\_exp[i];

while(item != '\0')

{

if(item == '(')

{

push(item);

}

else if( isdigit(item) || isalpha(item))

{

prefix\_exp[j] = item;

j++;

}

else if(isoperator(item) == 1)

{

x=pop();

while(isoperator(x) == 1 && precedence(x)>= precedence(item))

{

prefix\_exp[j] = x;

j++;

x = pop();

}

push(x);

push(item);

}

else if(item == ')')

{

x = pop();

while(x != '(')

{

prefix\_exp[j] = x;

j++;

x = pop();

}

}

else

{

printf("\nInvalid infix Expression.\n");

break;

}

i++;

item = infix\_exp[i];

}

if(top > 0)

printf("\n Invalid infix Expression.");

prefix\_exp[j] = '\0';

pos=0;

len=strlen(prefix\_exp);

for(k=len-1;k>=0;k--)

{

rev[pos]=prefix\_exp[k];

pos++;

}

rev[pos]='\0';

strcpy(prefix\_exp,rev);

}

void push(char c)

{

if(top >= SIZE-1)

printf("\n Stack Overflow.");

else

{

top++;

stack[top] = c;

}

}

char pop()

{

char c;

c='\0';

if(top < 0)

printf("\n Stack Underflow.");

else

{

c = stack[top];

top--;

}

return c;

}

int operator(char symbol)

{

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

return 1;

else

return 0;

}

int precedence(char symbol)

{

if(symbol == '^')

return(5);

else if(symbol == '/')

return(4);

else if(symbol == '\*')

return(3);

else if(symbol == '+')

return(2);

else if(symbol == '-')

return(1);

else

return(0);

}

**Output:**

**Enter Infix expression : abcd+(a\*b)-ef**

**Prefix Expression: -+abcd\*abef**

**38) Write a C program to evaluate the given pre-fix expression and post-fix expressions:**

**a)evaluating postfix expression:**

#include<stdio.h>

int stack[20];

int top = -1;

void push(int x)

{

stack[++top] = x;

}

int pop()

{

return stack[top--];

}

int main()

{

char exp[20];

char \*e;

int n1,n2,n3,num;

printf("Enter the expression :: ");

scanf("%s",exp);

e = exp;

while(\*e != '\0')

{

if(isdigit(\*e))

{

num = \*e - 48;

push(num);

}

else

{

n1 = pop();

n2 = pop();

switch(\*e)

{

case '+':

{

n3 = n1 + n2;

break;

}

case '-':

{

n3 = n2 - n1;

break;

}

case '\*':

{

n3 = n1 \* n2;

break;

}

case '/':

{

n3 = n2 / n1;

break;

}

}

push(n3);

}

e++;

}

printf("\nThe result of expression %s = %d\n\n",exp,pop());

return 0;

}

**Output:**

**Enter the expression :: a\*b**

**The result of expression a\*b = 0**

**b)evaluating prefix expression:**

#include<stdio.h>

#include<ctype.h>

#include<stdlib.h>

#define Max 20

int st[Max], top=-1;

void push(int ch)

{

if (top == Max-1)

{

printf("Stack is full\n");

}

else

{

top++;

st[top]=ch;

}

}

int pop()

{

int ch;

if (top==-1)

{

printf("Stack is empty\n");

}

else

{

ch=st[top];

top--;

}

return ch;

}

void dispstack()

{

int k;

printf("stack Content: ");

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

{

printf("%d, ", st[k]);

}

printf("\n");

}

int PreEval(char s[25])

{

char temp[25];

int i,val=0,ch1,ch2,j=0;

i=0; top=-1;

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

{

/\*if operand is countered print it\*/

if ( (s[i]>=48 && s[i]<=57) )

{

j=0;

temp[j]=s[i];

j++;

temp[j]='\0';

push(atoi(temp));

}

else

{

ch2=pop();

ch1=pop();

switch(s[i])

{

case '+' :{

val=ch2+ch1;

break;

}

case '-' :{

val=ch2-ch1;

break;

}

case '\*' :{

val=ch2\*ch1;

break;

}

case '/' :{

val=ch2/ch1;

break;

}

}

push(val);

}

i++;

}

val=pop();

return val;

}

void main()

{

char s[25],s1[25];

int val;

clrscr();

printf("enter a Prefix expression for evaluation\n");

scanf("%s",s);

strcpy(s1,strrev(s));

val= PreEval(s1);

printf("Value of Prefix Expression=%d\n", val);

getch();

}

**OUTPUT:**

**Enter a prefix expression evaluation A-B/ (C\*D^E)**

**Value of Prefix Expression= -A/B\*C^DE**

**Programs on Queues**

**39)Write a C program to implement a Linear-Queue, user must choose the following options:**

**a. Add an element to the Queue – EnQueue.**

**b. Remove an element from the Queue – DeQueue.**

**c. Display the elements of the Queue.**

**d. Terminate the program:**

#include<stdio.h>

#define SIZE 20

void enqueue(int);

void display();

int items[SIZE], front = -1, rear = -1;

int main()

{

int x,i,y;

printf("Enter no of elements in queue: ");

scanf("%d",&x);

for(i=0;i<x;i++){

printf("Enter Data: ");

scanf("%d",&y);

enqueue(y);

}

display();

}

void enqueue(int value){

if(rear == SIZE-1)

printf("\nQueue is Full!!");

else {

if(front == -1)

front = 0;

rear++;

items[rear] = value;

printf("\nInserted -> %d\n", value);

}

}

void display(){

int a[20],i;

if(rear == -1)

printf("\nQueue is Empty!!!");

else{

printf("\nQueue elements are:\n");

for(i=front; i<=rear; i++){

a[i]=items[i];

printf("%d\t",items[i]);

}

}

printf("\nQueue elements in reverse:\n");

for(i=rear;i>=front;i--)

{

printf("%d\t",a[i]);

}

printf("\nQueue elements in alternate Order:\n");

for(i=front;i<=rear;i+=2)

{

printf("%d\t",a[i]);

}

}

**OUTPUT:**

**Enter element in Queue: 4**

**Enter Data: 5 7 8 2**

**Queue elements are:**

**5 7 8 2**

**Queue elements in reverse:**

**2 8 7 5**

**Queue elements in alternate Order:**

**5 8**

**40) Write a C program to implement a Circular-Queue, user must choose the following options:**

**a. Add an element to the Queue – EnQueue.**

**b. Remove an element from the Queue – DeQueue.**

**c. Display the elements of the Queue.**

**d. Terminate the program.**

#include<stdio.h>

#define SIZE 10

void enQueue(int);

void deQueue();

void display();

int queue[10], front = -1, rear = -1;

void main() {

int value, choice;

while(1){

printf("\n\n\*\*\*\*\* MENU \*\*\*\*\*\n");

printf("1. Insertion\n2. Deletion\n3. Display\n4. Exit");

printf("\nEnter your choice: ");

scanf("%d",&choice);

switch(choice){

case 1: printf("Enter the value to be insert: ");

scanf("%d",&value);

enQueue(value);

break;

case 2: deQueue();

break;

case 3: display();

break;

case 4: exit(0);

default: printf("\nWrong selection!!! Try again!!!");

}

}}

void enQueue(int value){

if((front==0 &&rear == SIZE-1) || front==rear+1)

printf("\nQueue is Full!!! Insertion is not possible!!!");

else{

if(front == -1)

front = 0;

rear=(rear+1)%SIZE;

queue[rear] = value;

printf("\nInsertion success!!!");

}}

void deQueue(){

if(front == -1)

printf("\nQueue is Empty!!! Deletion is not possible!!!");

else{

printf("\nDeleted : %d", queue[front]);

front=(front+1)%SIZE;

if(front == rear)

front = rear = -1;

}}

void display(){

if(front == -1)

printf("\nQueue is Empty!!!");

else{

int i;

printf("\nQueue elements are:\n");

for(i=front; i!=rear; i=(i+1)%SIZE)

printf("%d\t",queue[i]);

}}

**OUTPUT:**

1. **Insertion**
2. **Deletion**
3. **Display**
4. **Exit**

**Enter your choice: 1**

**Enter your value to be insert: 45**

**Insertion success!!!**

1. **Insertion**
2. **Deletion**
3. **Display**
4. **Exit**

**Enter the choice: 2**

**Deleted : 45**

1. **Insertion**
2. **Deletion**
3. **Display**
4. **Exit**

**Enter your choice: 3**

**Queue elements are:**

**0 0 0 0 0 0 0 0**

1. **Insertion**
2. **Deletion**
3. **Display**
4. **Exit**

**Enter your choice: 4**

**Programs on linked-lists**

**41) Write a C program to create a single linked list with 5 nodes. (5 integers are taken from user input) and display the linked-list elements:**

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

void main()

{

struct node

{

int num;

struct node \*ptr;

};

typedef struct node NODE;

NODE \*head, \*first, \*temp = 0;

int count = 0;

int choice = 1;

first = 0;

while (choice)

{

head = (NODE \*)malloc(sizeof(NODE));

printf("Enter the data item\n");

scanf("%d", &head-> num);

if (first != 0)

{

temp->ptr = head;

temp = head;

}

else

{

first = temp = head;

}

fflush(stdin);

printf("Do you want to continue(Type 0 or 1)?\n");

scanf("%d", &choice);

}

temp->ptr = 0;

/\* reset temp to the beginning \*/

temp = first;

printf("\n status of the linked list is\n");

while (temp != 0)

{

printf("%d=>", temp->num);

count++;

temp = temp -> ptr;

}

printf("NULL\n");

printf("No. of nodes in the list = %d\n", count);

}

**OUTPUT**:

**Enter the data item**

**5**

**Do you want to continue(type 0 or 1)?**

**1**

**Enter the data item**

**3**

**Do you want to continue(type 0 or 1)?**

**1**

**Enter the data item**

**4**

**Do you want to continue(type 0 or 1)?**

**1**

**Enter the data item**

**6**

**Do you want to continue(type 0 or 1)?**

**1**

**Enter the data ite**

**7**

**Do you want to continue(type 0 or 1)?**

**0**

**Status of the linked list is**

**5=>3=>4=>6=>7=>NULL**

**NO. of nodes in the list = 5**

**42) Write a C program to search an element in a singly-linked list.**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int num;

struct node \*nextptr;

}

stnode, \*ennode;

int FindElement(int);

void main()

{

int n,i,FindElem,FindPlc;

stnode.nextptr=NULL;

ennode=&stnode;

printf("\n\n Linked List : Search an element in a Singly Linked List :\n");

printf("---------------------------------------------------------------\n");

printf(" Input the number of nodes : ");

scanf("%d", &n);

printf("\n");

for(i=0;i< n;i++)

{

ennode->nextptr=(struct node \*)malloc(sizeof(struct node));

printf(" Input data for node %d : ",i+1);

scanf("%d",&ennode->num);

ennode=ennode->nextptr;

}

ennode->nextptr=NULL;

printf("\n Data entered in the list are :\n");

ennode=&stnode;

while(ennode->nextptr!=NULL)

{

printf(" Data = %d\n",ennode->num);

ennode=ennode->nextptr;

}

printf("\n");

printf(" Input the element to be searched : ");

scanf("%d",&FindElem);

FindPlc=FindElement(FindElem);

if(FindPlc<=n)

printf(" Element found at node %d \n\n",FindPlc);

else

printf(" This element does not exists in linked list.\n\n");

}

int FindElement(int FindElem)

{

int ctr=1;

ennode=&stnode;

while(ennode->nextptr!=NULL)

{

if(ennode->num==FindElem)

break;

else

ctr++;

ennode=ennode->nextptr;

}

return ctr;

}

**OUTPUT:**

**Linked list : search an element in a singly linked list :**

**---------------------------------------------------------------**

**Input the number of nodes : 4**

**Input data for node 1 : 4**

**Input data for node 2 : 7**

**Input data for node 3 : 8**

**Input data for node 4 : 9**

**Data entered in the list are :**

**Data = 4**

**Data = 7**

**Data = 8**

**Data = 9**

**Input the element to be searched : 8**

**Element found at node 3**

**43)Write a C program to perform the following tasks:**

**a. Insert a node at the beginning of a singly-linked list.**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data; // Data

struct node \*next; // Address

}\*head;

void createList(int n);

void insertNodeAtBeginning(int data);

void displayList();

int main()

{

int n, data;

printf("Enter the total number of nodes: ");

scanf("%d", &n);

createList(n);

printf("\nData in the list \n");

displayList();

printf("\nEnter data to insert at beginning of the list: ");

scanf("%d", &data);

insertNodeAtBeginning(data);

printf("\nData in the list \n");

displayList();

return 0;

}

void createList(int n)

{

struct node \*newNode, \*temp;

int data, i;

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

if(head == NULL)

{

printf("Unable to allocate memory.");

}

else

{

printf("Enter the data of node 1: ");

scanf("%d", &data);

head->data = data; // Link data field with data

head->next = NULL; // Link address field to NULL

temp = head;

for(i=2; i<=n; i++)

{

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

if(newNode == NULL)

{

printf("Unable to allocate memory.");

break;

}

else

{

printf("Enter the data of node %d: ", i);

scanf("%d", &data);

newNode->data = data; // Link data field of newNode with data

newNode->next = NULL; // Link address field of newNode with NULL

temp->next = newNode; // Link previous node i.e. temp to the newNode

temp = temp->next;

}

}

printf("SINGLY LINKED LIST CREATED SUCCESSFULLY\n");

}

}

void insertNodeAtBeginning(int data)

{

struct node \*newNode;

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

if(newNode == NULL)

{

printf("Unable to allocate memory.");

}

else

{

newNode->data = data; // Link data part

newNode->next = head; // Link address part

head = newNode; // Make newNode as first node

printf("DATA INSERTED SUCCESSFULLY\n");

}

}

void displayList()

{

struct node \*temp;

if(head == NULL)

{

printf("List is empty.");

}

else

{

temp = head;

while(temp != NULL)

{

printf("Data = %d\n", temp->data); // Print data of current node

temp = temp->next; // Move to next node

}

}

}

**OUTPUT:**

**Enter the total number of nodes: 4**

**Enter the data of node 1: 20**

**Enter the data of node 2: 30**

**Enter the data of node 3: 40**

**Enter the data of node 4: 50**

**SINGLY LINKED LIST CREATED SUCCESSFULLY**

**Data in the list**

**Data = 20**

**Data = 30**

**Data = 40**

**Data = 50**

**Enter data to insert at the beginning of the list: 10**

**DATA INSERTED SUCCESSFULLY**

**Data in the list**

**Data = 10**

**Data = 20**

**Data = 30**

**Data = 40**

**Data = 50**

**b) Insert a node at end of a singly-linked list:**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data; // Data

struct node \*next; // Address

}\*head;

void createList(int n);

void insertNodeAtEnd(int data);

void displayList();

int main()

{

int n, data;

printf("Enter the total number of nodes: ");

scanf("%d", &n);

createList(n);

printf("\nData in the list \n");

displayList();

printf("\nEnter data to insert at end of the list: ");

scanf("%d", &data);

insertNodeAtEnd(data);

printf("\nData in the list \n");

displayList();

return 0;

}

void createList(int n)

{

struct node \*newNode, \*temp;

int data, i;

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

if(head == NULL)

{

printf("Unable to allocate memory.");

}

else

{

printf("Enter the data of node 1: ");

scanf("%d", &data);

head->data = data; // Link the data field with data

head->next = NULL; // Link the address field to NULL

temp = head;

for(i=2; i<=n; i++)

{

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

/\* If memory is not allocated for newNode \*/

if(newNode == NULL)

{

printf("Unable to allocate memory.");

break;

}

else

{

printf("Enter the data of node %d: ", i);

scanf("%d", &data);

newNode->data = data; // Link the data field of newNode with data

newNode->next = NULL; // Link the address field of newNode with NULL

temp->next = newNode; // Link previous node i.e. temp to the newNode

temp = temp->next;

}

}

printf("SINGLY LINKED LIST CREATED SUCCESSFULLY\n");

}

}

void insertNodeAtEnd(int data)

{

struct node \*newNode, \*temp;

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

if(newNode == NULL)

{

printf("Unable to allocate memory.");

}

else

{

newNode->data = data; // Link the data part

newNode->next = NULL;

temp = head;

while(temp->next != NULL)

temp = temp->next;

temp->next = newNode; // Link address part

printf("DATA INSERTED SUCCESSFULLY\n");

}

} displayList()

{

struct node \*temp;/

if(head == NULL)

{

printf("List is empty.");

}

else

{

temp = head;

while(temp != NULL)

{

printf("Data = %d\n", temp->data); // Print data of current node

temp = temp->next; // Move to next node

}

}

}

**OUTPUT:**

**Enter the total number of nodes: 3**

**Enter the data of node 1: 10**

**Enter the data of node 2: 20**

**Enter the data of node 3: 30**

**SINGLY LINKED LIST CREATED SUCCESSFULLY**

**Data in the list**

**Data = 10**

**Data = 20**

**Data = 30**

**Enter data to insert at end of the list: 40**

**DATA INSERTED SUCCESSFULLY**

**Data in the list**

**Data = 10**

**Data = 20**

**Data = 30**

**Data = 40**

**c) Insert a node at the middle of a singly-linked list:**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data; // Data

struct node \*next; // Address

}\*head;

void createList(int n);

void insertNodeAtMiddle(int data, int position);

void displayList();

int main()

{

int n, data, position;

printf("Enter the total number of nodes: ");

scanf("%d", &n);

createList(n);

printf("\nData in the list \n");

displayList();ntf("nEnter data to insert at middle of the list: ");

scanf("%d", &data);

printf("Enter the position to insert new node: " );

scanf("%d", &position);

insertNodeAtMiddle(data, position);

printf("\nData in the list \n");

displayList();

return 0;

}

void createList(int n)

{

struct node \*newNode, \*temp;

int data, i;

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

if(head == NULL)

{

printf("Unable to allocate memory.");

}

else

{

printf("Enter the data of node 1: ");

scanf("%d", &data);

head->data = data; // Link the data field with data

head->next = NULL; // Link the address field to NULL

temp = head;/

for(i=2; i<=n; i++)

{

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

if(newNode == NULL)

{

printf("Unable to allocate memory.");

break;

}

else

{

printf("Enter the data of node %d: ", i);

scanf("%d", &data);

newNode->data = data; // Link the data field of newNode with data

newNode->next = NULL; // Link the address field of newNode with NULL

temp->next = newNode; // Link previous node i.e. temp to the newNode

temp = temp->next;

}

}

printf("SINGLY LINKED LIST CREATED SUCCESSFULLY\n");

}

}

void insertNodeAtMiddle(int data, int position)

{

int i;

struct node \*newNode, \*temp;

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

if(newNode == NULL)

{

printf("Unable to allocate memory.");

}

else

{

newNode->data = data; // Link data part

newNode->next = NULL;

temp = head;

for(i=2; i<=position-1; i++)

{

temp = temp->next;

if(temp == NULL)

break;

}

if(temp != NULL)

{

newNode->next = temp->next;

temp->next = newNode;

printf("DATA INSERTED SUCCESSFULLY\n");

}

else

{

printf("UNABLE TO INSERT DATA AT THE GIVEN POSITION\n");

}

}

}

void displayList()

{

struct node \*temp;

if(head == NULL)

{

printf("List is empty.");

}

else

{

temp = head;

while(temp != NULL)

{

printf("Data = %d\n", temp->data); // Print data of current node

temp = temp->next; // Move to next node

}

}

}

**OUTPUT:**

**Enter the total number of nodes: 4**

**Enter the data of node 1: 10**

**Enter the data of node 2: 20**

**Enter the data of node 3: 40**

**Enter the data of node 4: 50**

**SINGLY LINKED LIST CREATED SUCCESSFULLY**

**Data in the list**

**Data = 10**

**Data = 20**

**Data = 40**

**Data = 50**

**Enter data to insert at middle of the list: 30**

**Enter the position to insert new node: 3**

**DATA INSERTED SUCCESSFULLY**

**Data in the list**

**Data = 10**

**Data = 20**

**Data = 30**

**Data = 40**

**Data = 50**

**d) Delete a node from the beginning of the singly-linked list:**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data; // Data

struct node \*next; // Address

}\*head;

void createList(int n);

void deleteFirstNode();

void displayList();

int main()

{

int n, choice;

printf("Enter the total number of nodes: ");

scanf("%d", &n);

createList(n);

printf("\nData in the list \n");

displayList();

printf("\nPress 1 to delete first node: ");

scanf("%d", &choice);

if(choice == 1)

deleteFirstNode();

printf("\nData in the list \n");

displayList();

return 0;

}

void createList(int n)

{

struct node \*newNode, \*temp;

int data, i;

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

if(head == NULL)

{

printf("Unable to allocate memory.");

}

else

{

printf("Enter the data of node 1: ");

scanf("%d", &data);

head->data = data; // Link the data field with data

head->next = NULL; // Link the address field to NULL

temp = head;

for(i=2; i<=n; i++)

{

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

if(newNode == NULL)

{

printf("Unable to allocate memory.");

break;

}

else

{

printf("Enter the data of node %d: ", i);

scanf("%d", &data);

newNode->data = data; // Link the data field of newNode with data

newNode->next = NULL; // Link the address field of newNode with NULL

temp->next = newNode; // Link previous node i.e. temp to the newNode

temp = temp->next;

}

}

printf("SINGLY LINKED LIST CREATED SUCCESSFULLY\n");

}

}

void deleteFirstNode()

{

struct node \*toDelete;

if(head == NULL)

{

printf("List is already empty.");

}

else

{

toDelete = head;

head = head->next;

printf("\nData deleted = %d\n", toDelete->data);

free(toDelete);

printf("SUCCESSFULLY DELETED FIRST NODE FROM LIST\n");

}

}

void displayList()

{

struct node \*temp;

if(head == NULL)

{

printf("List is empty.");

}

else

{

temp = head;

while(temp != NULL)

{

printf("Data = %d\n", temp->data); // Print data of current node

temp = temp->next; // Move to next node

}

}

}

**OUTPUT:**

**Enter the total number of nodes: 4**

**Enter the data of node 1: 10**

**Enter the data of node 2: 20**

**Enter the data of node 3: 40**

**Enter the data of node 4: 50**

**SINGLY LINKED LIST CREATED SUCCESSFULLY**

**Data in the list**

**Data = 10**

**Data = 20**

**Data = 40**

**Data = 50**

**Enter data to insert at middle of the list: 30**

**Enter the position to insert new node: 3**

**DATA INSERTED SUCCESSFULLY**

**Data in the list**

**Data = 10**

**Data = 20**

**Data = 30**

**Data = 40**

**Data = 50**

**e) Delete a node from the end of a singly-linked list:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int num; //Data of the node

struct node \*nextptr; //Address of the node

}\*stnode;

void createNodeList(int n); //function to create the list

void LastNodeDeletion(); //function to delete the last nodes

void displayList(); //function to display the list

int main()

{

int n,num,pos;

printf("\n\n Linked List : Delete the last node of Singly Linked List :\n");

printf("---------------------------------------------------------------\n");

printf(" Input the number of nodes : ");

scanf("%d", &n);

createNodeList(n);

printf("\n Data entered in the list are : \n");

displayList();

LastNodeDeletion();

printf("\n The new list after deletion the last node are : \n");

displayList();

return 0;

}

void createNodeList(int n)

{

struct node \*fnNode, \*tmp;

int num, i;

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

if(stnode == NULL) //check whether the stnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

}

else

{

printf(" Input data for node 1 : ");

scanf("%d", &num);

stnode-> num = num;

stnode-> nextptr = NULL; //Links the address field to NULL

tmp = stnode;

for(i=2; i<=n; i++)

{

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

if(fnNode == NULL) //check whether the fnnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

break;

}

else

{

printf(" Input data for node %d : ", i);

scanf(" %d", &num);

fnNode->num = num; // links the num field of fnNode with num

fnNode->nextptr = NULL; // links the address field of fnNode with NULL

tmp->nextptr = fnNode; // links previous node i.e. tmp to the fnNode

tmp = tmp->nextptr;

}

}

}

}

void LastNodeDeletion()

{

struct node \*toDelLast, \*preNode;

if(stnode == NULL)

{

printf(" There is no element in the list.");

}

else

{

toDelLast = stnode;

preNode = stnode;

while(toDelLast->nextptr != NULL)

{

preNode = toDelLast;

toDelLast = toDelLast->nextptr;

}

if(toDelLast == stnode)

{

stnode = NULL;

}

else

{

preNode->nextptr = NULL;

}

free(toDelLast);

}

}

void displayList()

{

struct node \*tmp;

if(stnode == NULL)

{

printf(" No data found in the empty list.");

}

else

{

tmp = stnode;

while(tmp != NULL)

{

printf(" Data = %d\n", tmp->num); // prints the data of current node

tmp = tmp->nextptr; // advances the position of current node

}

}

}

**OUTPUT**

**Input the number of nodes : 3**

**Input data for node 1 : 1**

**Input data for node 2 : 2**

**Input data for node 3 : 3**

**Data entered in the list are :**

**Data = 1**

**Data = 2**

**Data = 3**

**The new list after deletion the last node are :**

**Data = 1**

**Data = 2**

**44) Write a C program to create a doubly linked list with 5 nodes:**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \* prev;

struct node \* next;

}\*head, \*last;

void createList(int n);

void displayListFromFirst();

void displayListFromEnd();

int main()

{

int n, choice;

head = NULL;

last = NULL;

printf("Enter the number of nodes you want to create: ");

scanf("%d", &n);

createList(n); // Create list of n nodes

printf("\nPress 1 to display list from First");

printf("\nPress 2 to display list from End : ");

scanf("%d", &choice);

if(choice==1)

{

displayListFromFirst();

}

else if(choice == 2)

{

displayListFromEnd();

}

return 0;

}

void createList(int n)

{

int i, data;

struct node \*newNode;

if(n >= 1)

{

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

if(head != NULL)

{

printf("Enter data of 1 node: ");

scanf("%d", &data);

head->data = data;

head->prev = NULL;

head->next = NULL;

last = head;

for(i=2; i<=n; i++)

{

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

if(newNode != NULL)

{

printf("Enter data of %d node: ", i);

scanf("%d", &data);

newNode->data = data;

newNode->prev = last;

newNode->next = NULL;

last->next = newNode;

last = newNode;

}

else

{

printf("Unable to allocate memory.");

break;

}

}

printf("\nDOUBLY LINKED LIST CREATED SUCCESSFULLY\n");

}

else

{

printf("Unable to allocate memory");

}

}

}

void displayListFromFirst()

{

struct node \* temp;

int n = 1;

if(head == NULL)

{

printf("List is empty.");

}

else

{

temp = head;

printf("\n\nDATA IN THE LIST:\n");

while(temp != NULL)

{

printf("DATA of %d node = %d\n", n, temp->data);

N++;

/\* Move the current pointer to next node \*/

temp = temp->next;

}

}

}

void displayListFromEnd()

{

struct node \* temp;

int n = 0;

if(last == NULL)

{

printf("List is empty.");

}

else

{

temp = last;

printf("\n\nDATA IN THE LIST:\n");

while(temp != NULL)

{

printf("DATA of last-%d node = %d\n", n, temp->data);

n++;

/\* Move the current pointer to previous node \*/

temp = temp->prev;

}

}

}

**OUTPUT:**

**Enter the number of nodes you want to create: 5**

**Enter data of 1 node: 10**

**Enter data of 2 node: 20**

**Enter data of 3 node: 30**

**Enter data of 4 node: 40**

**Enter data of 5 node: 50**

**DOUBLY LINKED LIST CREATED SUCCESSFULLY**

**Press 1 to display list from First**

**Press 2 to display list from End : 1**

**DATA IN THE LIST:**

**DATA of 1 node = 10**

**DATA of 2 node = 20**

**DATA of 3 node = 30**

**DATA of 4 node = 40**

**DATA of 5 node = 50**

**45) Write a C program to create a circular linked list with 5 nodes:**

#include <stdio.h>

#include <stdlib.h>

struct node {

int num;

struct node \* nextptr;

}\*stnode;

void ClListcreation(int n);

void displayClList();

int main()

{

int n;

stnode = NULL;

printf("\n\n Circular Linked List : Create and display a circular linked list :\n");

printf("-----------------------------------------------------------------------\n");

printf(" Input the number of nodes : ");

scanf("%d", &n);

ClListcreation(n);

displayClList();

return 0;

}

void ClListcreation(int n)

{

int i, num;

struct node \*preptr, \*newnode;

if(n >= 1)

{

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

printf(" Input data for node 1 : ");

scanf("%d", &num);

stnode->num = num;

stnode->nextptr = NULL;

preptr = stnode;

for(i=2; i<=n; i++)

{

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

printf(" Input data for node %d : ", i);

scanf("%d", &num);

newnode->num = num;

newnode->nextptr = NULL;

preptr->nextptr = newnode;

preptr = newnode;

}

preptr->nextptr = stnode;

}

}

void displayClList()

{

struct node \*tmp;

int n = 1;

if(stnode == NULL)

{

printf(" No data found in the List yet.");

}

else

{

tmp = stnode;

printf("\n\n Data entered in the list are :\n");

do {

printf(" Data %d = %d\n", n, tmp->num);

tmp = tmp->nextptr;

n++;

}while(tmp != stnode);

}

**OUTPUT:**

**Circular Linked List : Create a circular linked list :**

**-----------------------------------------------------------------------**

**Input the number of nodes : 5**

**Input data for node 1 : 4**

**Input data for node 2 : 6**

**Input data for node 3 : 7**

**Input data for node 4 : 5**

**Input data for node 5 : 8**

**Data entered in the list are :**

**Data 1 = 4**

**Data 2 = 6**

**Data 3 = 7**

**Data 4 = 5**

**Data 5 = 8**

**46) Write a C program to implement the stack using linked list:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int info;

struct node \*ptr;

}\*top,\*top1,\*temp;

int topelement();

void push(int data);

void pop();

void empty();

void display();

void destroy();

void stack\_count();

void create();

int count = 0;

void main()

{

int no, ch, e;

printf("\n 1 - Push");

printf("\n 2 - Pop");

printf("\n 3 - Top");

printf("\n 4 - Empty");

printf("\n 5 - Exit");

printf("\n 6 - Display");

printf("\n 7 - Stack Count");

printf("\n 8 - Destroy stack");

create();

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

{

e = topelement();

printf("\n Top element : %d", e);

}

break;

case 4:

empty();

break;

case 5:

exit(0);

case 6:

display();

break;

case 7:

stack\_count();

break;

case 8:

destroy();

break;

default :

printf(" Wrong choice, Please enter correct choice ");

break;

}

}

}

void create()

{

top = NULL;

}

void stack\_count()

{

printf("\n No. of elements in stack : %d", count);

}

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 topelement()

{

return(top->info);

}

void empty()

{

if (top == NULL)

printf("\n Stack is empty");

else

printf("\n Stack is not empty with %d elements", count);

}

void destroy()

{

top1 = top;

while (top1 != NULL)

{

top1 = top->ptr;

free(top);

top = top1;

top1 = top1->ptr;

}

free(top1);

top = NULL;

printf("\n All stack elements destroyed");

count = 0;

}

**OUTPUT:**

**1 - Push**

**2 - Pop**

**3 - Top**

**4 - Empty**

**5 - Exit**

**6 - Display**

**7 - Stack Count**

**8 - Destroy stack**

**Enter choice : 1**

**Enter data : 22**

**Enter choice : 1**

**Enter data : 45**

**Enter choice : 2**

**Popped value : 45**

**Enter choice : 3**

**Top element : 22**

**Enter choice : 1**

**Enter data : 57**

**Enter choice : 1**

**Enter data : 75**

**Enter choice : 6**

**75 57 22**

**Enter choice : 7**

**No. of elements in stack : 3**

**Enter choice : 8**

**All stack elements destroyed**

**Enter choice : 4**

**Stack is empty**

**Enter choice : 5**

**47) Write a C program to implement the queue using a linked list:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int info;

struct node \*ptr;

}\*front,\*rear,\*temp,\*front1;

int frontelement();

void enq(int data);

void deq();

void empty();

void display();

void create();

void queuesize();

int count = 0;

void main()

{

int no, ch, e;

printf("\n 1 - Enque");

printf("\n 2 - Deque");

printf("\n 3 - Front element");

printf("\n 4 - Empty");

printf("\n 5 - Exit");

printf("\n 6 - Display");

printf("\n 7 - Queue size");

create();

while (1)

{

printf("\n Enter choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter data : ");

scanf("%d", &no);

enq(no);

break;

case 2:

deq();

break;

case 3:

e = frontelement();

if (e != 0)

printf("Front element : %d", e);

else

printf("\n No front element in Queue as queue is empty");

break;

case 4:

empty();

break;

case 5:

exit(0);

case 6:

display();

break;

case 7:

queuesize();

break;

default:

printf("Wrong choice, Please enter correct choice ");

break;

}

}

}

void create()

{

front = rear = NULL;

}

void queuesize()

{

printf("\n Queue size : %d", count);

}

void enq(int data)

{

if (rear == NULL)

{

rear = (struct node \*)malloc(1\*sizeof(struct node));

rear->ptr = NULL;

rear->info = data;

front = rear;

}

else

{

temp=(struct node \*)malloc(1\*sizeof(struct node));

rear->ptr = temp;

temp->info = data;

temp->ptr = NULL;

rear = temp;

}

count++;

}

void display()

{

front1 = front;

if ((front1 == NULL) && (rear == NULL))

{

printf("Queue is empty");

return;

}

while (front1 != rear)

{

printf("%d ", front1->info);

front1 = front1->ptr;

}

if (front1 == rear)

printf("%d", front1->info);

}

void deq()

{

front1 = front;

if (front1 == NULL)

{

printf("\n Error: Trying to display elements from empty queue");

return;

}

else

if (front1->ptr != NULL)

{

front1 = front1->ptr;

printf("\n Dequed value : %d", front->info);

free(front);

front = front1;

}

else

{

printf("\n Dequed value : %d", front->info);

free(front);

front = NULL;

rear = NULL;

}

count--;

}

int frontelement()

{

if ((front != NULL) && (rear != NULL))

return(front->info);

else

return 0;

}

void empty()

{

if ((front == NULL) && (rear == NULL))

printf("\n Queue empty");

else

printf("Queue not empty");

}

**OUTPUT:**

**1 - Enque**

**2 - Deque**

**3 - Front element**

**4 - Empty**

**5 - Exit**

**6 - Display**

**7 - Queue size**

**Enter choice : 1**

**Enter data : 16**

**Enter choice : 1**

**Enter data : 48**

**Enter choice : 1**

**Enter data : 33**

**Enter choice : 3**

**Front element : 16**

**Enter choice : 6**

**16 48 33**

**Enter choice : 7**

**Queue size : 3**

**Enter choice : 2**

**Dequed value : 16**

**Enter choice : 6**

**48 33**

**Enter choice : 7**

**Queue size : 2**

**Enter choice : 4**

**Queue not empty**

**Enter choice : 5**