Programs on Functions:

Program 22:

Title: Sum of Natural numbers

Objective: Write a C Program to find the sum of natural numbers using function.

Explanation: Given n where n>=0, we have sum = 1+2+....+n

Code:

#include <stdio.h>

//function to add numbers from 1 to n

int sum\_natural (int n)

{

int i,sum= 0 ;

//loops runs from 1 to n

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

sum += i;

return sum;

}

int main ()

{

int n;

//input n

printf( "Enter Number\n" );

scanf( "%d" ,&n);

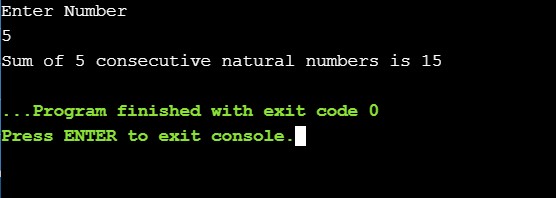
//print sum

printf( "Sum of %d consecutive natural numbers is %d" ,n,sum\_natural(n));

return 0 ;

}

Output:



Program 23:

Title: Factorial of Number

Objective: Write a C Program to find factorial of number using recursion.

Explanation: Given a number n, where n>=0, factorial of number n! = 1x2x3….(n-1)x(n)

Code:

#include <stdio.h>

int recfact( int n)

{

if (n == 0 || n== 1 )

return 1 ;

return n\*recfact(n -1 );

}

int main ()

{

int n;

printf( "Enter Number\n" );

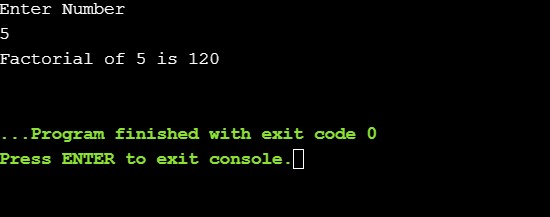
scanf( "%d" ,&n);

printf( "Factorial of %d is %d\n" ,n,recfact(n));

return 0 ;

}

Output:



Program 24:

Title: Fibonacci

Objective: Write a C Program to generate the Fibonacci series.

Explanation: For finding the nth fibonocci number, Fn = Fn-1 + Fn-2.

Code:

#include <stdio.h>

//prints the fibonocci series upto n numbers

void fibonnaci\_series( int n)

{

int i, a = 0 , b= 1 ,c;

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

{

c = a+b;

a = b;

b = c;

//prints fibonoccci number

printf( "%d\n" ,c);

}

}

int main ()

{

int n;

//inputting n

printf( "Enter Number\n" );

scanf( "%d" ,&n);

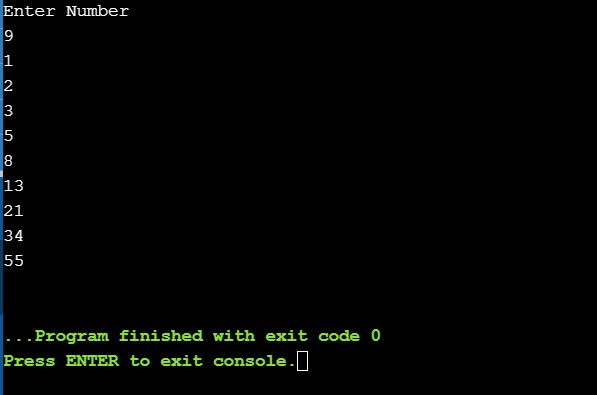
//calling function to print the first n fibonooci numbers.

fibonnaci\_series(n);

return 0 ;

}

Output:



Programs on Structure, String and Pointers:

Program 25:

Title: Implementation of Structure

Objective: C Program using the structure for entering details of the five students like name,

Admission number, date of the birth, department and display all the details.

Explanation:

Input and output of structure having some objects. In c we access the structure

objects by ‘.’ for example if we have structure student, which contains objects name,

roll no, let student be the variable initializing the structure then we can access the objects

by, stu.name, stu.roll no. We do the same with any number of objects.

Code:

#include <stdio.h>

//structure for date

typedef struct Date

{

int month;

int day;

int year;

}date;

//structure for student

typedef struct Student

{

char name[30];

int adminNo;

date DOB;

char dept[30];

}student;

int main()

{

student s[5];

int i;

//inputtting the details of 5 students

printf("Enter details: \n");

for(i=0;i<5;i++){

printf("Enter\n");

printf("Name:");

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

printf("Admin No: ");

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

printf("Date of Birth:\n");

printf("Date: ");

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

printf("Month: ");

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

printf("year: ");

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

printf("Department: ");

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

}

//printing details of five students

printf("\n Details: \n");

for(i=0;i<5;i++){

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

printf("Admin No: %d\n",s[i].adminNo );

printf("Date of Birth (DDMMYYYY): %d/%d/%d\n",s[i].DOB.day,s[i].DOB.month,s[i].DOB.year);

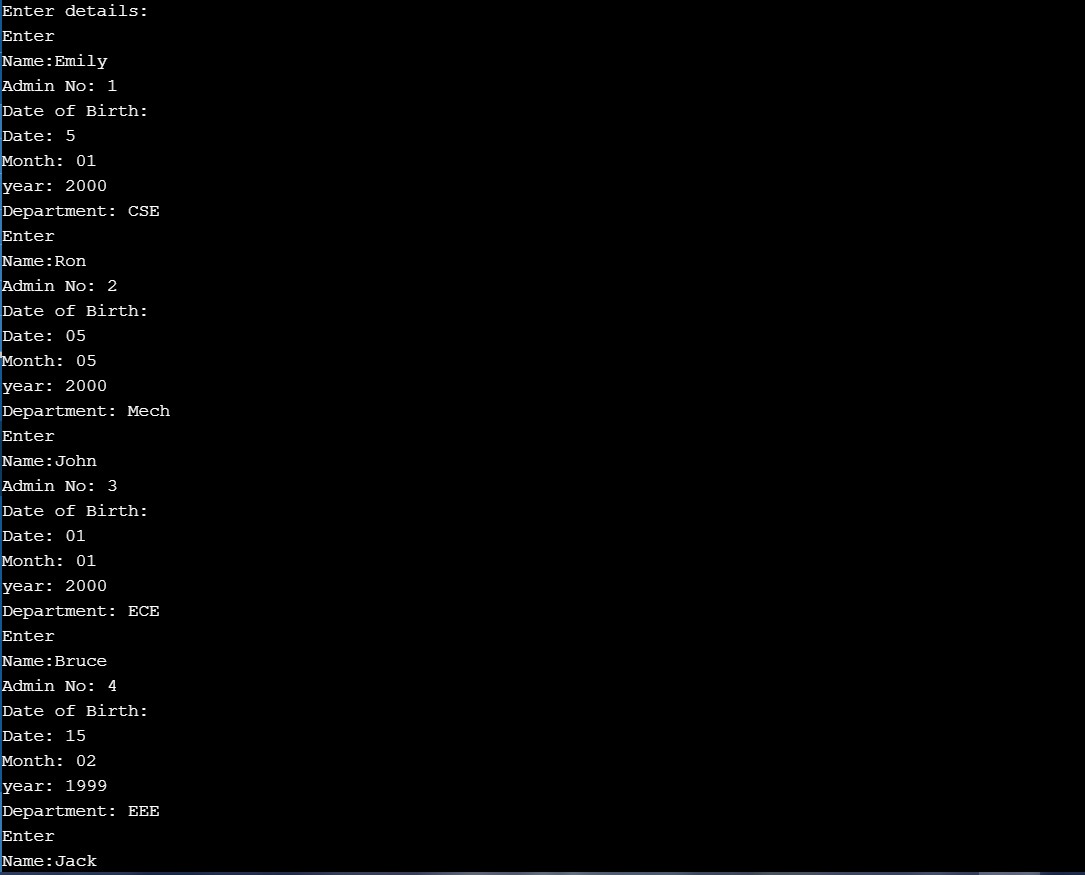
printf("Department: %s\n\n",s[i].dept);

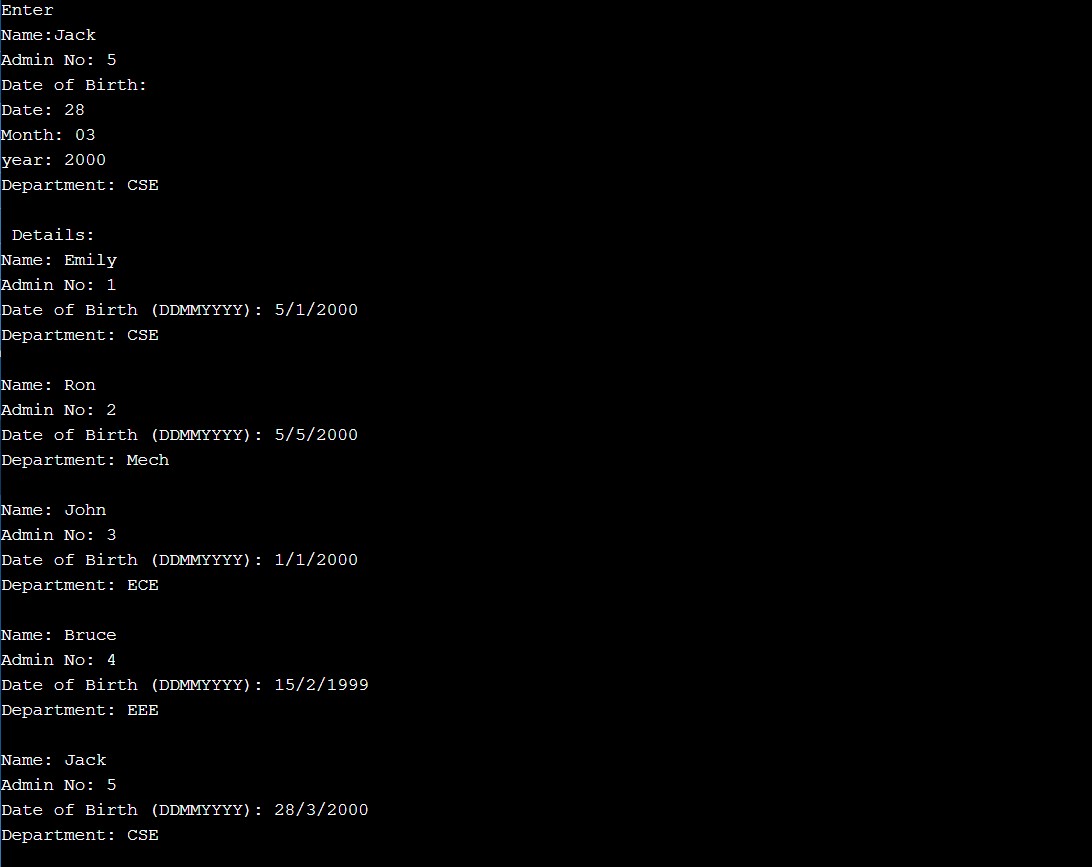
}

return 0;

}

Output:





Program 26:

Title: String Length

Objective: Write a C program to find length of string using pointers

Explanation:

In C programs, strings end as a null character, represented by ‘\0’. Thus to find the

String length we can simply count the characters till ‘\0’ from 0.

Code:

#include <stdio.h>

//calculates the length of the string and return the length

int string\_ln ( char \* i)

{

int count = 0 ;

//runs the loop till the end of the list

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

count++;

i++;

}

return count;

}

int main ()

{

char str[ 40 ];

//inputting string

printf( "Enter a string:\n" );

gets(str);

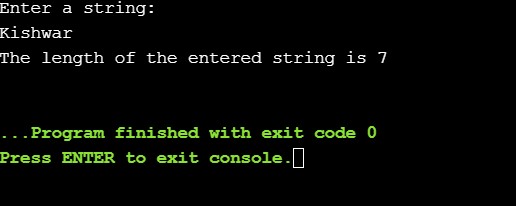
//printing the length

printf( "The length of the entered string is %d\n" ,string\_ln(str));

return 0 ;

}

Output:



Program 27:

Title: Copying String

Objective: Write a C program to copy one string to another using pointers

Explanation:

To copy a string to another we can just initialize every character of the string to the

Another, till end of the string which is represented by ‘\0’.

Code:

#include <stdio.h>

void copystr ( char \*dest, char \*src)

{

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

\*dest++=\*src++;

\*dest= '\0' ;

}

int main ()

{

char str1[ 30 ],str2[ 30 ];

//inputting string

printf( "Enter a string:\n" );

gets(str1);

copystr(str2,str1);

printf( "Entered String:\n" );

printf( "%s\n" ,str1 );

printf( "Copied String:\n" );

printf( "%s\n" ,str2 );

return 0 ;

}

Output:



Program 28:

Title: String Comparison

Objective: Write a C program to compare two strings using pointers.

Explanation:

For comparing string we traverse throught the string simultaneously if we bith string

reach ‘\0’ at same time it would mean they are of equal length. If one reached earlier

than other it would mean they are of unequal lengths.

Code:

int strcomp ( char \*str1, char \*str2)

{

while (\*str1 == \*str2){

if ( \*str1 == '\0' || \*str2 == '\0' )

break ;

str1++;

str2++;

}

if ( \*str1 == '\0' && \*str2 == '\0' )

return 1 ;

return 0 ;

}

int main ()

{

char str1[ 30 ],str2[ 30 ];

//inputting string

printf( "Enter string:\n" );

gets(str1);

printf( "Enter string to compare:\n" );

gets(str2);

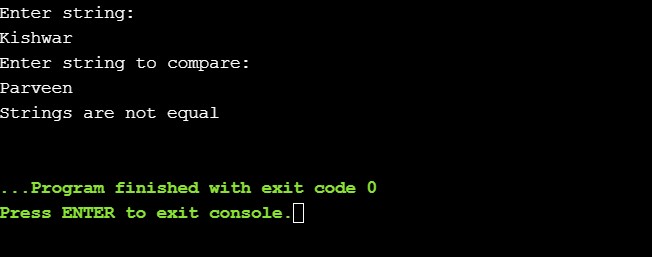
if (strcomp(str1,str2) == 1 ) printf( "Strings are equal\n" );

else printf( "Strings are not equal\n" );

return 0 ;

}

Output:



Program 29:

Title: Reverse String

Objective: Write a C program to find the reverse of a string recursively and non-recursively.

Explanation:

We can reverse string by swapping the elements of first half with the last half.

Code:

#include <stdio.h>

void recRev ( char \*x, int beg, int end)

{

char temp;

if (beg >= end)

return ;

temp = \*(x+beg);

\*(x+beg) = \*(x+end);

\*(x+end) = temp;

recRev(x, ++beg, --end);

}

int string\_ln ( char \* p){

int count = 0 ;

//runs the loop till the end of the list

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

count++;

p++;

}

return count;

}

void rev ( char \*str)

{

int len, i;

char \*beg, \*end, temp;

len = string\_ln(str);

beg = str;

end = str;

for (i = 0 ;i < (len - 1 ) ; i++ )

end++;

for ( i = 0 ; i < len/ 2 ; i++ )

{

temp = \*end;

\*end = \*beg;

\*beg = temp;

beg++;

end--;

}

}

int main ()

{

char str1[ 30 ];

//input string

printf( "Enter a string:\n" );

gets(str1);

printf( "Reversing the string using recursion\n" );

recRev(str1, 0 ,string\_ln(str1) -1 );

printf( "%s\n" ,str1);

printf( "Reversing the reversed string using iteration\n" );

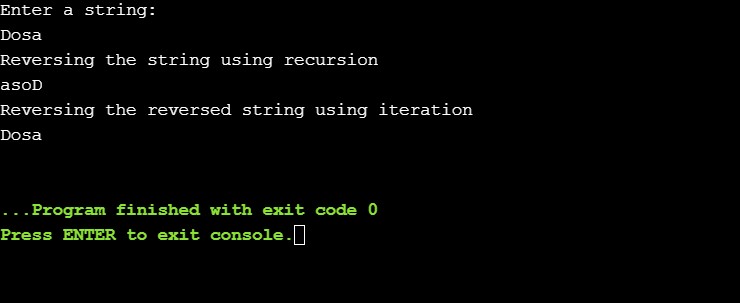
rev(str1);

printf( "%s\n" ,str1);

return 0 ;

}

Output:



Program 30:

Title: Binary Tree Transversal

Objective: Write a C program to find the reverse of a string recursively and non-recursively.

Explanation:

A binary tree, is a tree in which no node can have more than two children.

Pre-order: The traversal goes in format root, left, right.

In-order: The traversal goes in format left, root, and right.

Post-order: The traversal goes in format left, right, root.

Code:

#include <stdio.h>

#include <stdlib.h>

#define initmemory() (struct node\*)malloc(sizeof(struct node))

struct node {

int data;

struct node \*left;

struct node \*right;

};

struct node\* insert(){

struct node \*newnode;

int x;

printf( "Enter data:" );

scanf( "%d" ,&x);

if (x== -1 )

return NULL;

newnode = initmemory();

newnode->data = x;

printf( "left child of %d:\n" ,x);

newnode->left = insert();

printf( "right child of %d:\n" ,x);

newnode->right = insert();

return newnode;

}

void postOrder ( struct node \*root) {

if (root == NULL){

return ;

}

postOrder(root->left);

postOrder(root->right);

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

}

void inOrder ( struct node \*root) {

if (root == NULL) return ;

inOrder(root->left);

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

inOrder(root->right);

}

void preOrder ( struct node \*root) {

if (root == NULL) return ;

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

preOrder(root->left);

preOrder(root->right);

}

int main () {

struct node\* root = insert();

int num,i;

int data;

printf( "\nPost Order:\n" );

postOrder(root);

printf( "\nPre Order\n" );

preOrder(root);

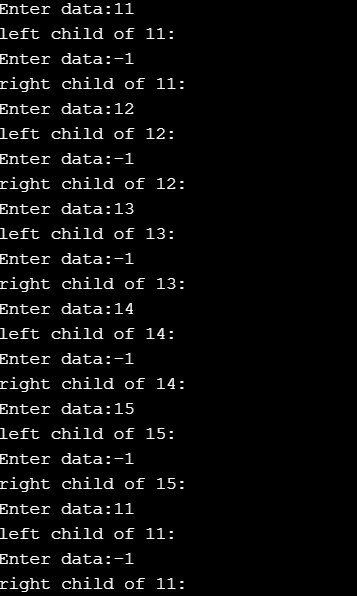
printf( "\nIn Order\n" );

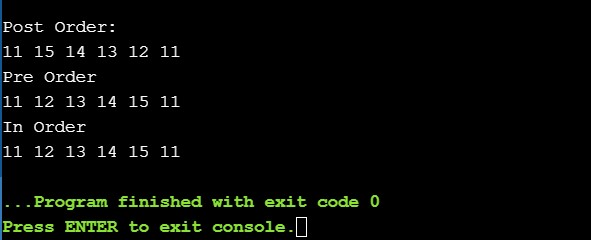
inOrder(root);

return 0 ;

}

Output:





Program 31:

Title: Binary Search Tree

Objective: Create a Binary Search Tree (BST) and search for a given value in BST.

Explanation:

Searching for a node is similar to inserting a node. We start from root, and then go left

Or right until we find (or not find the node).

Code:

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#define initmemory() (struct node\*)malloc(sizeof(struct node))

typedef struct node {

int data;

struct node \*left;

struct node \*right;

}node;

node\* insert(node\* root, int data) {

if(root == NULL) {

node\* node = initmemory();

node->data = data;

node->left = NULL;

node->right = NULL;

return node;

} else {

if(data <= root->data) {

root->left = insert(root->left, data);

}

else {

root->right = insert(root->right, data);;

}

return root;

}

}

int bstSearch(node\* root, int search)

{

if (root == NULL)

return 0;

if(root->data == search)

return 1;

if (root->data < search)

return 2\*(bstSearch(root->right, search));

return 2\*(bstSearch(root->left, search));

}

int main(int argc, char const \*argv[])

{

node\* root = NULL;

int num,i,search,data,pos;

printf("Enter initial tree size:\n");

scanf("%d", &num);

printf("Enter the elements in tree:\n");

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

scanf("%d", &data);

root = insert(root, data);

}

printf("\nEnter search element:\n");

scanf("%d",&search);

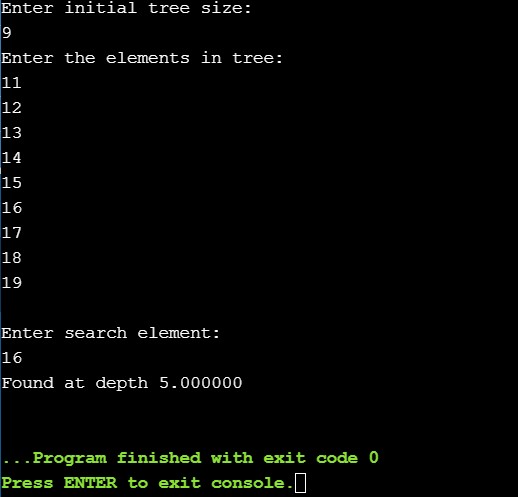
pos = bstSearch(root,search);

printf("Found at depth %f\n",log2(pos));

return 0;

}

Output:



Program 32:

Title: Single Source Shortest Path

Objective:

Write a program to implement a single source shortest path algorithm. Either Bellman-Ford or Dijkstra’s algorithm.

Code:

#include <stdio.h>

#include <stdlib.h>

int Bellman\_Ford(int G[20][20] , int V, int E, int edge[20][2])

{

int i,u,v,k,distance[20],parent[20],S,flag=1;

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

distance[i] = 1000 , parent[i] = -1 ;

printf("Enter source: ");

scanf("%d",&S);

distance[S-1]=0 ;

for(i=0;i<V-1;i++)

{

for(k=0;k<E;k++)

{

u = edge[k][0] , v = edge[k][1] ;

if(distance[u]+G[u][v] < distance[v])

distance[v] = distance[u] + G[u][v] , parent[v]=u ;

}

}

for(k=0;k<E;k++)

{

u = edge[k][0] , v = edge[k][1] ;

if(distance[u]+G[u][v] < distance[v])

flag = 0 ;

}

if(flag)

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

printf("Vertex %d -> cost = %d parent = %d\n",i+1,distance[i],parent[i]+1);

return flag;

}

int main()

{

int V,edge[20][2],G[20][20],i,j,k=0;

printf("BELLMAN FORD\n");

printf("Enter no. of vertices: ");

scanf("%d",&V);

printf("Enter graph in matrix form:\n");

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

for(j=0;j<V;j++)

{

scanf("%d",&G[i][j]);

if(G[i][j]!=0)

edge[k][0]=i,edge[k++][1]=j;

}

if(Bellman\_Ford(G,V,k,edge))

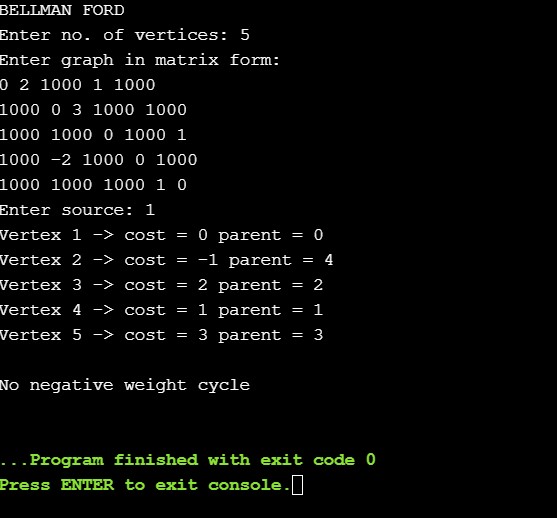
printf("\nNo negative weight cycle\n");

else printf("\nNegative weight cycle exists\n");

return 0;

}

Output:



Program 33:

Title: Shortest Path in Graph

Objective: Write a program to find All-to-all Shortest paths in a Graph.

Explanation:

BFS is a traversing algorithm where you should start traversing from a selected node

(Source or starting node) and traverse the graph layer wise thus exploring the

Neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes

Code:

#include <stdio.h>

#include <stdlib.h>

#define SIZE 40

struct queue {

int items[SIZE];

int front;

int rear;

};

struct queue\* createQueue();

void enqueue(struct queue\* q, int);

int dequeue(struct queue\* q);

void display(struct queue\* q);

int isEmpty(struct queue\* q);

void printQueue(struct queue\* q);

struct node {

int vertex;

struct node\* next;

};

struct node\* createNode(int);

struct Graph {

int numVertices;

struct node\*\* adjLists;

int\* visited;

};

void bfs(struct Graph\* graph, int startVertex) {

struct queue\* q = createQueue();

graph->visited[startVertex] = 1;

enqueue(q, startVertex);

while (!isEmpty(q)) {

printQueue(q);

int currentVertex = dequeue(q);

printf("Visited %d\n", currentVertex);

struct node\* temp = graph->adjLists[currentVertex];

while (temp) {

int adjVertex = temp->vertex;

if (graph->visited[adjVertex] == 0) {

graph->visited[adjVertex] = 1;

enqueue(q, adjVertex);

}

temp = temp->next;

}

}

}

// Creating a node

struct node\* createNode(int v) {

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

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

// Creating a graph

struct Graph\* createGraph(int vertices) {

struct Graph\* graph = malloc(sizeof(struct Graph));

graph->numVertices = vertices;

graph->adjLists = malloc(vertices \* sizeof(struct node\*));

graph->visited = malloc(vertices \* sizeof(int));

int i;

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

graph->adjLists[i] = NULL;

graph->visited[i] = 0;

}

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest) {

struct node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

newNode = createNode(src);

newNode->next = graph->adjLists[dest];

graph->adjLists[dest] = newNode;

}

struct queue\* createQueue() {

struct queue\* q = malloc(sizeof(struct queue));

q->front = -1;

q->rear = -1;

return q;

}

int isEmpty(struct queue\* q) {

if (q->rear == -1)

return 1;

else

return 0;

}

void enqueue(struct queue\* q, int value) {

if (q->rear == SIZE - 1)

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

else {

if (q->front == -1)

q->front = 0;

q->rear++;

q->items[q->rear] = value;

}

}

int dequeue(struct queue\* q) {

int item;

if (isEmpty(q)) {

printf("Queue is empty");

item = -1;

} else {

item = q->items[q->front];

q->front++;

if (q->front > q->rear) {

printf("Resetting queue ");

q->front = q->rear = -1;

}

}

return item;

}

void printQueue(struct queue\* q) {

int i = q->front;

if (isEmpty(q)) {

printf("Queue is empty");

} else {

printf("\nQueue contains \n");

for (i = q->front; i < q->rear + 1; i++) {

printf("%d ", q->items[i]);

}

}

}

int main() {

struct Graph\* graph = createGraph(6);

addEdge(graph, 0, 1);

addEdge(graph, 0, 2);

addEdge(graph, 1, 2);

addEdge(graph, 1, 4);

addEdge(graph, 1, 3);

addEdge(graph, 2, 4);

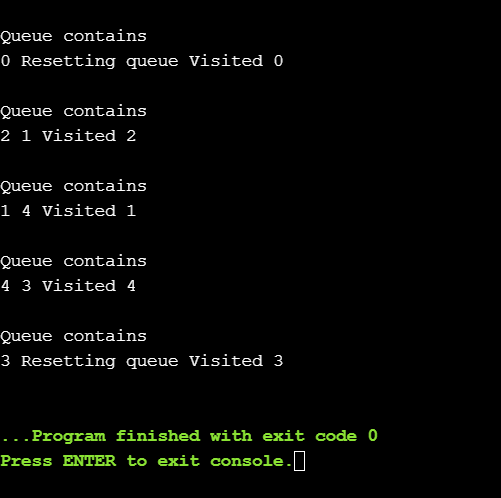
addEdge(graph, 3, 4);

bfs(graph, 0);

return 0;

}

Output:



Program 34:

Title: Implementing Stacks using Arrays

Objective:

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.

1. Push an element on to the STACK.

2. Pop and element from the STACK.

3. Peek the STACK.

4. Display the STACK.

5. Exit the program.

Explanation:

Push: Adds an item in the stack. If the stack is full, then it is said to be an Overflow

condition.

Pop: Removes an item from the stack. The items are popped in the reversed order in

which they are pushed. If the stack is empty, then it is said to be an Underflow

condition.

Peek: Returns top element of stack

Code:

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define max 1000//max elements in stack

//defining stack

typedef struct STACK{

int ar[max];

int top;

}stack;

//function to push elements into stack

void push(stack \*s, int data){

//overflow condition

if(s->top >= max-1){

printf("overflow\n");

return;

}

s->top++;

s->ar[s->top] = data;

}

//function to pop elements

int pop(stack \*s){

//underflow condition

if(s->top < 0) {

printf("Underflow\n");

return INT\_MIN;

//raise: if element stored == INT\_MIN , the function fails

}

int temp = s->ar[s->top];

s->top--;

return temp;

}

//return the top elemnt without disturbing the top

int peek(stack s){

return s.ar[s.top];

}

//displays elements from top

void display(stack s){

int i;

if(s.top == -1){

printf("Empty\n");

}

for(i =s.top;i>-1;i--){

printf("%d\n",s.ar[i]);

}

printf("\n");

}

//main

int main(int argc, char const \*argv[])

{

//intialized variables needed

stack s;

s.top = -1;

int choice,data;

//runs loop till user chooses exit --> 5

while(1){

//menu

printf("\n1. Push an element on to the STACK.\n"

"2. Pop and element from the STACK.\n"

"3. Peek the STACK.\n"

"4. Display the STACK.\n"

"5. Exit the program.\n");

scanf("%d",&choice);

//performs action according the choice

switch(choice){

case 1:{

printf("\nEnter an element to add\n");

scanf("%d",&data);

push(&s,data);

break;

}

case 2:{

data =pop(&s);

if(data != INT\_MIN)

printf("%d is removed\n",data);

break;

}

case 3:{

printf("%d is the top\n",peek(s) );

break;

}

case 4:{

display(s);

break;

}

case 5:{

exit(0);

break;

}

default: printf("no such option choose again\n");

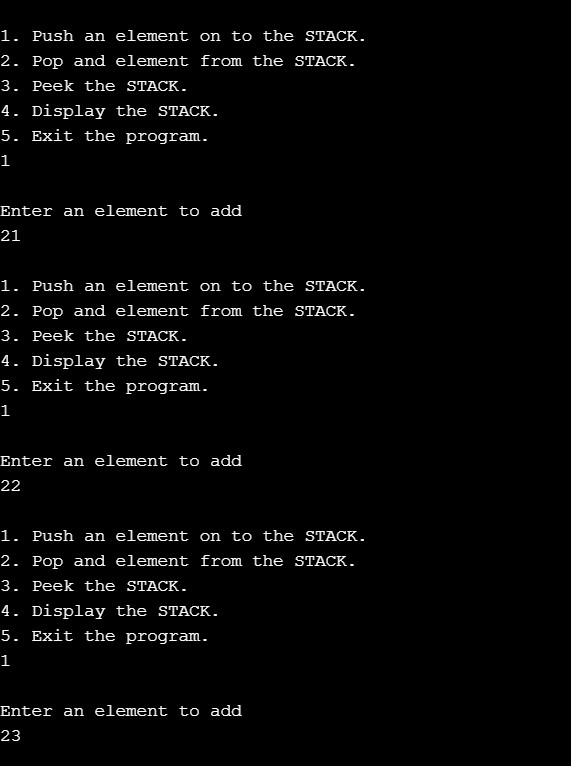
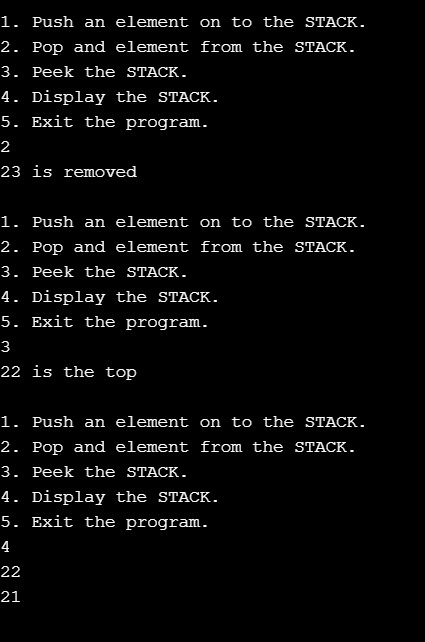
}

}

return 0;

}

Output:

Program 35:

Title: Reversing a string using Stacks

Objective:

Write a C program to reverse a string using STACK

Explanation:

When give with a string like “C Programming” it needs to be converted into

“gninmmargorp C”. We can achieve this by using STACK data structure since it fools

FILO(First In Last Out).

Code:

#include <stdio.h>

#define max 1000

//globally initalized stack

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

//function which return the top element of the global stack

char pop(){

//underflow condion

if(top == -1)

return '0';

char res = stack[top];

top--;

return res;

}

//fuction pushes <char> data into the global stack

void push(char data){

//overflow condition

if(top == max-1)

printf("overflow\n");

top++;

stack[top] = data;

}

//main

int main(){

char string[max];

//inputting the string

printf("Enter String:\n");

scanf("%[^\n]%\*c",string);

int i ;

//pushes all the elements in the string into stack

for(i = 0;string[i] !='\0';i++){

push(string[i]);

}

//pops all elements from stack a d intializes to string

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

int c = pop();

if(c == '0')

break;

string[i] = c;

}

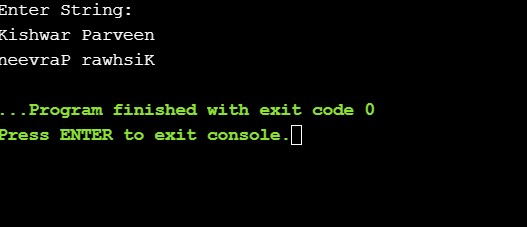
//prints the string after reversing

printf("%s",string);

return 0;

}

Output:



Program 36:

Title: Conversion of In-Fix to Post-Fix

Objective:

Write a C program to convert the given infix expression to post-fix expression using

STACK.

Explanation:

To convert infix expression to postfix expression, we will use the stack data structure.

By scanning the infix expression from left to right, when we will get any operand, simply add them to the postfix form, and for the operator and parenthesis, add them in the stack maintaining the precedence of them.

Code:

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define max 1000

//intializing stack

typedef struct Stack{

int top;

int arr[max];

}stack;

//pops a element

char pop(stack \*s){

if (s->top != -1)

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

return '#';

}

//push a element

void push(stack \*s, char op){

s->arr[++s->top] = op;

}

//preccedence

int Prec(char ch){

switch (ch) {

case '+': return 1;

case '-': return 1;

case '\*': return 2;

case '/': return 2;

case '^': return 3;

}

return -1;

}

//convert infix two postfix

void in2post(char\* exp) {

int i, k;

stack s; s.top = -1;

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

//if is a variable

if ((exp[i] >= 'a' && exp[i] <= 'z') || (exp[i] >= 'A' && exp[i] <= 'Z'))

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

//if a opening bracket

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

push(&s, exp[i]);

//if a closing bracket

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

//pops all elements form the stack till '('

while (s.top != -1 && s.arr[s.top] != '(')

exp[++k] = pop(&s);

//if there is no '('

if (s.top == -1 && s.arr[s.top] != '(')

printf("Invalid expression\n");

//pops '('

else

pop(&s);

}

//if is a operation

else {

//pops till the precendence of stack is greater than current element

while (s.top != -1 && Prec(exp[i]) <= Prec(s.arr[s.top]))

exp[++k] = pop(&s);

//pushes the current operation into the stack

push(&s, exp[i]);

}

}

exp[++k] = pop(&s);

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

printf( "%s", exp );

}

//main

int main() {

//driver code

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

printf( "Infix- expression: %s\n", exp );

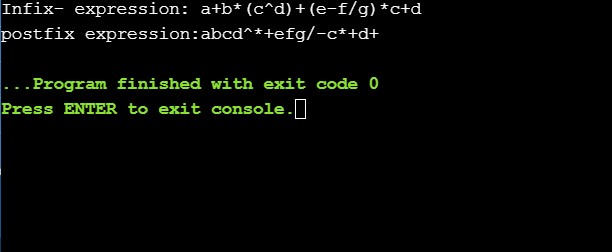
printf("%s","postfix expression:" );

in2post(exp);

return 0;

}

Output:



Program 37:

Title: Conversion of In-Fix to Pre-Fix

Objective:

Write a C program to convert the given infix expression to pre-fix expression using

STACK.

Explanation:

When the operators are before operands then it is a prefix expression. We can achieve to convert infix to prefix by reversing prefix expression and running through postfix function.

Code:

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define max 1000

//intializing stack

typedef struct Stack{

int top;

char arr[max];

}stack;

//pops a element

char pop(stack \*s){

if (s->top != -1)

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

return '#';

}

//push a element

void push(stack \*s, char op){

s->arr[++s->top] = op;

}

//preccedence

int Prec(char ch){

switch (ch) {

case '+': return 1;

case '-': return 1;

case '\*': return 2;

case '/': return 2;

case '^': return 3;

}

return -1;

}

//displays elements from top

void display(stack s){

int i;

if(s.top == -1){

printf("Empty\n");

}

for(i =s.top;i>-1;i--){

printf("%c",s.arr[i]);

}

printf("\n");

}

//convert infix two prefix

void in2pre(char\* exp) {

int i, n;

for(n = 0;exp[n];n++);

stack s; s.top = -1;

stack pre;pre.top = -1;

for (i = n-1; i>=0; i--) {

//if is a variable

if ((exp[i] >= 'a' && exp[i] <= 'z') || (exp[i] >= 'A' && exp[i] <= 'Z'))

push(&pre,exp[i]);

//if a opening bracket(reverse in prefix) therefore '('' = ')'

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

push(&s, exp[i]);

//if a closing bracket

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

//pops all elements form the stack till '('

while (s.top != -1 && s.arr[s.top] != ')')

push(&pre,pop(&s));

//if there is no '('

if (s.top == -1 && s.arr[s.top] != ')')

printf("Invalid expression\n");

//pops '('

else

pop(&s);

}

//if is a operation

else {

//pops till the precendence of stack is greater than current element

while (s.top != -1 && Prec(exp[i]) <= Prec(s.arr[s.top]))

push(&pre,pop(&s));

//pushes the current operation into the stack

push(&s, exp[i]);

}

}

push(&pre,pop(&s));

display(pre);

}

//main

int main() {

//driver code

char exp[] = "(a-b/c)\*(a/k-l)";

printf( "Infix- expression: %s\n", exp );

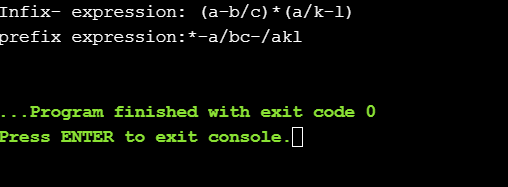
printf("%s","prefix expression:" );

in2pre(exp);

return 0;

}

Output:



Program 38:

Title: Evaluation of Post-fix and Pre-fix expressions.

Objective: Write a C program to evaluate the given pre-fix expression, post-fix expression

Explanation:

Post-fix evaluation: While reading the expression from left to right, push the element

in the stack if it is an operand.Pop the two operands from the stack, if the element is

an operator and then evaluate it.Push back the result of the evaluation. Repeat it till the end of the expression.

Pre-fix evaluation: We do the same thing as post-fix evalution but we read the

Expression from right to left.

Code:

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define max 1000

//intializing stack

typedef struct Stack{

int top;

int arr[max];

}stack;

//pops a element

int pop(stack \*s){

if (s->top > -1)

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

return -1;

}

//push a element

void push(stack \*s, int op){

s->arr[++s->top] = op;

}

//displays elements from top

void display(stack s){

int i;

if(s.top == -1){

printf("Empty\n");

}

for(i =s.top;i>-1;i--){

printf("%d\n",s.arr[i]);

}

printf("\n");

}

//evaluation of post fix expression where number are having spaces after them

void evaluatePost(char \*exp){

int i, num = 0;

stack operand; operand.top = -1;

for(i= 0;exp[i];i ++){

//if is a number

if(exp[i]-'0' >= 0 && exp[i]-'0' <=9){

//updates the number

if(num == 0) num = exp[i]-'0';

else num = num\*10 + exp[i]-'0';

}

//if is a space

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

//pushes the element

push(&operand, num);

num = 0;

}

//if is a operand

else{

//pops two number from stack and performs operations

int op2 = pop(&operand);

int op1 = pop(&operand);

if(op2 == -1 || op1 == -1){

printf("invalid\n");

return;

}

//pushes the result into the stack

switch(exp[i]){

case '\*':{

push(&operand, (op1\*op2));

break;

}

case '/':{

push(&operand,(op1/op2));

break;

}

case '+':{

push(&operand,(op1+op2));

break;

}

case '-':{

push(&operand,(op1-op2));

break;

}

}

}

}

//if the stack is not empty then the ex[ression is invalid

if(operand.top != 0){ printf("invalid\n");

//return;

}

display(operand);

}

//evaluation of pre fix expression where number are having spaces before them

//is same as post fix expect that it evaluate the expression backward

void evaluatePre(char \*exp){

int n;

for(n=0;exp[n];n++);

int i, num = 0;

stack operand; operand.top = -1;

for(i = n-1;i>-1;i--){

if(exp[i]-'0' >= 0 && exp[i]-'0' <=9){

if(num == 0) num = exp[i]-'0';

else num = num\*10 + exp[i]-'0';

}

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

push(&operand, num);

num = 0;

}

else{

int op2 = pop(&operand);

int op1 = pop(&operand);

if(op2 == -1 || op1 == -1){

printf("invalid\n");

return;

}

switch(exp[i]){

case '\*':{

push(&operand, (op1\*op2));

break;

}

case '/':{

push(&operand,(op1/op2));

break;

}

case '+':{

push(&operand,(op1+op2));

break;

}

case '-':{

push(&operand,(op1-op2));

break;

}

}

}

}

if(operand.top != 0){ printf("invalid\n");

return;

}

display(operand);

}

int main(){

//expression for post should have space after every number and no space anywhere else

//example "34 45 \*23 +/" is valid

//"3 4 5 \* 5 / -" is invalid

char exp[] = "7 8 +3 2 +/";

printf("expression = %s\n",exp);

evaluatePost(exp);

//expression for post should have space before every number and no space anywhere else

//example " 34 45\* 23+/" is valid

//" 3 4 5 \* 5 / - " is invalid

char exp2[] = "+ 9\* 2 6";

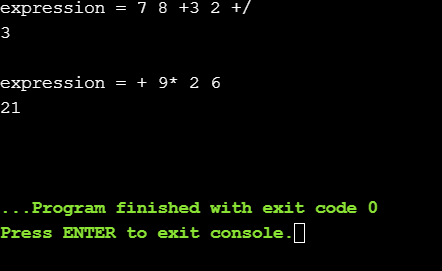
printf("expression = %s\n",exp2);

evaluatePre(exp2);

return 0;

}

Output:



Program 39:

Title: Implementation of Linear Queue using Stacks

Objective:

Write a C program to implement a Linear-Queue, user must choose the following

options:

1. Add an element to the Queue – EnQueue.

2. Remove an element from the Queue – DeQueue.

3. Display the elements of the Queue.

4. Terminate the program.

Explanation:

Like Stack, Queue is a linear structure which follows a particular order in which the

operations are performed. The order is First In First Out (FIFO).

Enqueue: Adds an item to the queue. If the queue is full, then it is said to be an

Overflow condition.

Dequeue: Removes an item from the queue. The items are popped in the same order

in which they are pushed. If the queue is empty, then it is said to be an Underflow

condition.

Code:

#include <stdio.h>

#include <stdlib.h>

# define MAX\_SIZE 100

//displays elements from front to back

void display(int \*queue, int rear, int front){

int i;

printf("Now queue (front.....to .....back:)\n");

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

printf("%d ",\*(queue+i));

}

printf("%d ",\*(queue+rear));

}

//adding a eleemt into the queue

void enqueue(int \*queue, int \*rear,int \*front){

//overflow condition

if(\*rear == MAX\_SIZE-1){

printf("Overflow\n Aborting...");

return;

}

//inserts a element in rear

int ele;

printf("Enter the element to insert:\n");

scanf("%d",&ele);

if (\*rear == -1)

{

\*front = 0;

}

\*rear += 1;

\*(queue+\*rear) = ele;

}

//deletes a element form front

void dequeue(int \*queue, int \*front,int \*rear){

//underflow condition

if((\*front)==-1){

printf("\nunderflow.. aborting");

return;

}

int temp = \*(queue+(\*front));

if(\*front== \*rear){\*front= -1; \*rear= -1;}

else{\*front = \*front + 1;}

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

}

int main(){

int ans, queue[MAX\_SIZE], front = -1, rear = -1;

//menu

while(1){

printf("\nMENU\n"

"\n1.Insert an element "

"\n2.Delete an element "

"\n3.Display queue"

"\n4.Exit\n");

scanf("%d",&ans);

//does according to option choosed

switch(ans){

case 1: enqueue(queue, &rear, &front);break;

case 2: dequeue(queue,&front,&rear);break;

case 3: display(queue, rear, front);break;

case 4: exit(0);break;

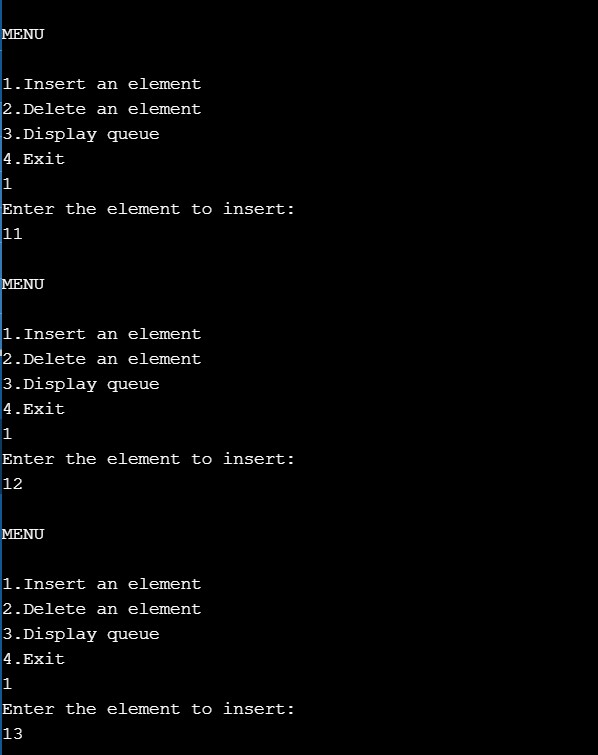
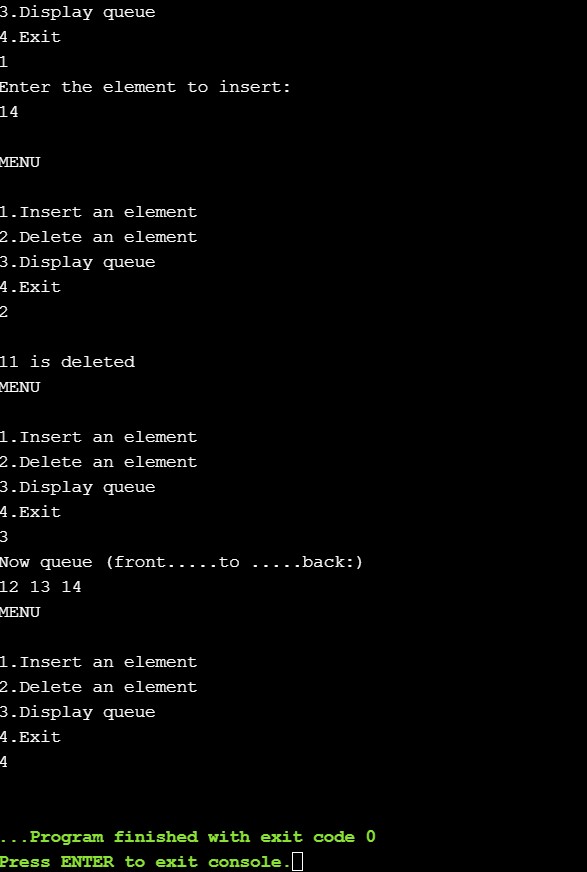
}

}

return 0;

}

Output:

Program 40:

Title: Implementation of Circular Queue

Objective:

Write a C program to implement a Circular-Queue, user must choose the following

options:

1. Add an element to the Queue – EnQueue.

2. Remove an element from the Queue – DeQueue.

3. Display the elements of the Queue.

4. Terminate the program.

Explanation:

A circular queue is a linear data structure in which the operations are performed

based on FIFO (First In First Out) principle and the last position is connected back to

the first position to make a circle.

Code:

#include <stdio.h>

#include <stdlib.h>

# define MAX\_SIZE 5

//adding a eleemt into the circular queue

void enqueue(int \*queue, int \*rear,int \*front){

//overflow condition

if((\*rear == MAX\_SIZE-1 && \*front == 0)|| (\*front== (\*rear)+1)){

printf("Overflow\n Aborting...");

return;

}

//inserts a element in rear

int ele;

printf("Enter the element to insert:\n");

if (\*front == -1){

\*front = 0;

}

\*rear = ((\*rear)+1)%MAX\_SIZE;

printf("%d is rear\n",\*rear );

scanf("%d",&ele);

\*(queue+\*rear) = ele;

}

//displays elements from front to back of circular queue

void display(int queue[], int rear, int front){

int i;

printf("Now queue (front.....to .....back:)\n");

for(i=(front);i!=rear;i= (i+1)%MAX\_SIZE){

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

}

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

}

//deletes a element form front of circular queue

void dequeue(int \*queue, int \*front,int \*rear){

//underflow condition

if((\*front)==-1){

printf("\nunderflow.. aborting");

return;

}

int temp = \*(queue+(\*front));

if(\*front== \*rear){\*front= -1; \*rear= -1;}

else{\*front = (\*front + 1)%MAX\_SIZE;}

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

}

int main(){

int ans, queue[MAX\_SIZE], front = -1, rear = -1;

//menu

while(1){

printf("\nMENU:circular queue\n"

"\n1.Insert an element "

"\n2.Delete an element "

"\n3.Display queue"

"\n4.Exit\n");

scanf("%d",&ans);

//does according to option choosed

switch(ans){

case 1: enqueue(queue, &rear, &front);break;

case 2: dequeue(queue, &front,&rear);break;

case 3: display(queue, rear, front);break;

case 4: exit(0);break;

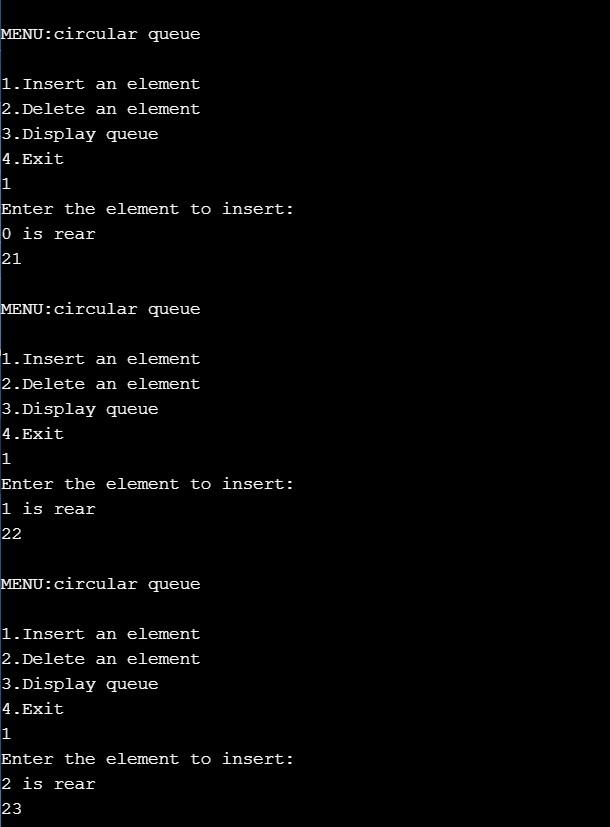
}

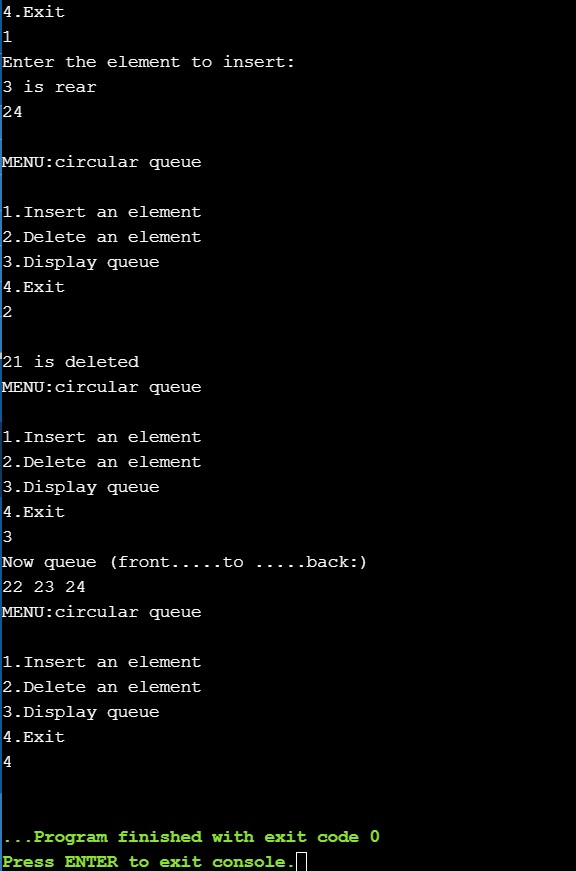
}

return 0;

}

Output:





Program 41:

Title: Intializing and displaying a node of five elements

Objective:

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.

Code:

#include <stdio.h>

#include <stdlib.h>

//declaration if a node

typedef struct Node{

int data;

struct Node \*next;

}node;

//function which creates nodes

node\* createNode( int data){

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

n->data = data;

n->next = NULL;

return n;

}

//function which creates a list of 5 nodes

node\* createList(){

int n= 5 ,data;

node \*p, \*head = NULL;

//runs loop 5 times

while (n--){

printf( "Enter a number\n" );

scanf( "%d" ,&data);

if (head == NULL){

//intializing newnode as head

head = createNode(data);

p = head;

}

else {

p->next = createNode(data);

p = p->next;

}

}

//return the list of 5nodes

return head;

}

//displays elemets in linked list till null

void display (node \*head){

while (head!=NULL){

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

head = head->next;

}

printf( "NULL\n" );

}

//main

int main ()

{

//driver code

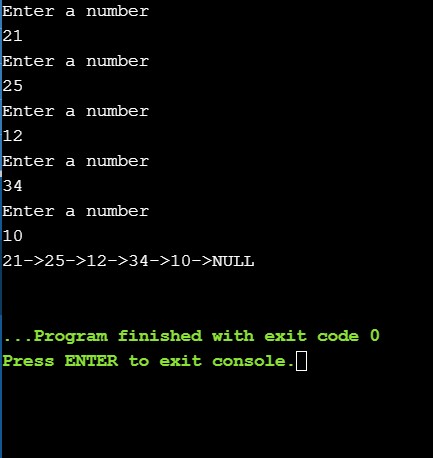
node \*head=createList();

display(head);

return 0 ;

}

Output:



Program 42:

Title: Searching a element in linked list

Objective: Write a C program to search an element in a singly-linked list.

Explanation:

When given with an integer we we search for the element in the linked list and print

the position (index+1) of the element. If not found should print -1 Here we will use

linear search because list might not be sorted

Code:

#include <stdio.h>

#include <stdlib.h>

//declaration if a node

typedef struct Node {

int data;

struct Node \* next ;

}node;

//function which creates nodes

node\* createNode ( int data){

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

n->data = data;

n->next = NULL ;

return n;

}

//function which creates a list of n nodes

node\* createList (){

int n,data;

node \*p, \*head = NULL ;

printf( "\n How many elements to enter?" );

scanf( "%d" , &n);

//runs loop 5 times

while (n--){

printf( "Enter a number\n" );

scanf( "%d" ,&data);

if (head == NULL ){

//intializing newnode as head

head = createNode(data);

p = head;

}

else {

p->next = createNode(data);

p = p->next;

}

}

//return the list of 5nodes

return head;

}

//return position of the search element

int search (node \*head, int search){

int pos= 1 ;

while (head != NULL ){

if (head->data == search) return pos;

head = head->next;

pos++;

}

//not found condition

return -1 ;

}

//main

int main ( int argc, char const \*argv[])

{

//driver code

node \*head = createList();

int s;

printf( "element to be searched\n" );

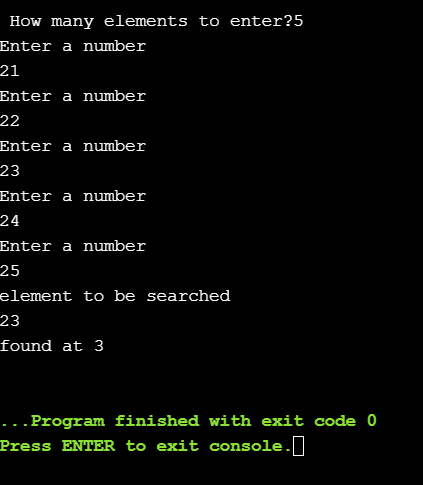
scanf( "%d" ,&s);

printf( "found at %d\n" ,search(head,s));

return 0 ;

}

Output:



Program 43:

Title: Implementation of Linked Lists

Objective:

Write a C program to perform the following tasks:

1. Insert a node at beginning of a singly-linked list.

2. Insert a node at end of a singly-linked list.

3. Insert a node at middle of a singly-linked list.

4. Delete a node from the beginning of the singly-linked list.

5. Delete a node from the end of a singly-linked list.

Explanation:

Insertion in beggining of a list: When we add a new node the new node becomes

the head of the list thus this function needs to return the head of the newnode

Insertion in end of a list: This function needs to add a new node at a=end of node

and if the list is emoty return a newnode.

Insertion in middle of a list: This function needs to add a node in n/2th postion

where n is the total nubmer of nodes in the linked-list.

Deletion in beginning of a list: This function needs to delete the free the memory of

first node and return the the next node of the head. If empty should return empty.

Deletion in end of a list: This function needs to delete a node at end if only one node

is present return NULL

Code:

#include <stdio.h>

#include <stdlib.h>

#define init() ((struct node\*)malloc(sizeof(struct node)))

typedef struct node

{

int data;

struct node \*next;

struct node \*prev;

}node;

//function which creates nodes

node\* createNode(int data){

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

n->data = data;

n->next = NULL;

return n;

}

//inserts a node in the begging

node\* insertBeg(node \*head,int data){

node \*newNode = createNode(data);

newNode->next = head;

return newNode;

}

//inserts at middle

//if nodes = even it adds at nodes/2

//if nodes = odd adds at nodes/2 + 1

void insertMiddle(node \*head,int data){

node \*ptr = head;

if(ptr == NULL){

//if list empty doesnot add eleemet

printf("empty\n");

return;

}

//runns till the end

while(head->next != NULL){

if(head->next->next != NULL){

head = head->next->next;//runs fast

ptr = ptr->next;//runs half the iterations

}

else{

break;

}

}

node \*temp = ptr->next;

ptr->next = createNode(data);

ptr->next->next = temp;

}

//recusive function which insert the node at the end

node\* insertEnd(node \*head, int data){

//if empty return a newlist with one element

if(head == NULL){

return createNode(data);

}

//calls insertend function

head->next = insertEnd(head->next, data);

return head;

}

//deletes node in the begging

node\* deleteBeg(node \*head){

node \*temp = head;

//node empty thus returns null

if(head == NULL){

printf("Empty\n");

return NULL;

}

//ideltes the node

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

free(temp);

//returns the head's next elements

return head->next;

}

//deletes the last last node(not recursive)

node\* deleteEnd(node \*head){

//if empty list return NULL

if(head == NULL ){

printf("empty\n");

return NULL;

}

//if a single element returns NULL

if (head->next == NULL){

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

free(head);

return NULL;

}

//last node deleted

head->next = deleteEnd(head->next);

return head;

}

//displays the list

void display(node \*head){

while(head!=NULL){

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

head = head->next;

}

printf("NULL\n");

}

//main

int main () {

int choice,data;

node \*head ;

while(1){

//menu

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

printf("\nChoose one option from the following list ...\n");

printf("\n===============================================\n");

printf("\n1.Insert in begining\n2.Insert at last\n3.Insert middle.\n4.Delete num at the begining \n5.Delete num at the end\n6.Display\n7.Exit\n");

printf("\nEnter your choice?\n");

scanf("\n%d",&choice);

//performs operation according to the choice

switch(choice){

case 1:{

printf("Enter the data to be inserted\n");

scanf("%d",&data);

head = insertBeg(head, data);

break;

}

case 2:{

printf("Enter the data to be inserted\n");

scanf("%d",&data);

head = insertEnd(head, data);

break;

}

case 3:{

printf("Enter the data to be inserted\n");

scanf("%d",&data);

insertMiddle(head, data);

break;

}

case 4:{

head = deleteBeg(head);

break;

}

case 5:{

head = deleteEnd(head);

break;

}

case 6:{

printf("The list:\n");

display(head);

break;

}

case 7: {exit(0);break;}

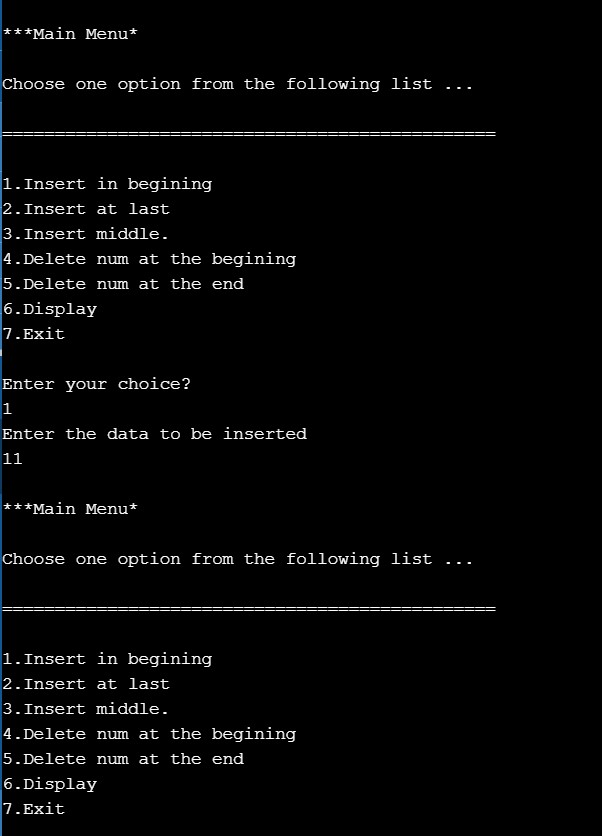
}

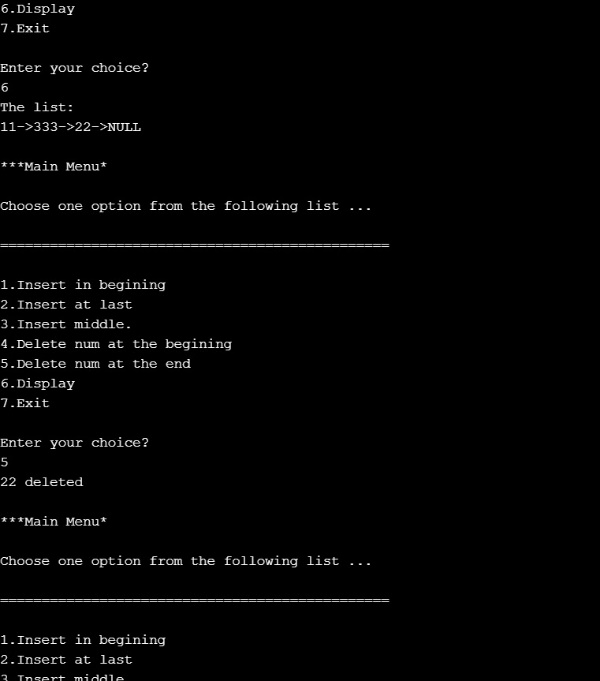
}

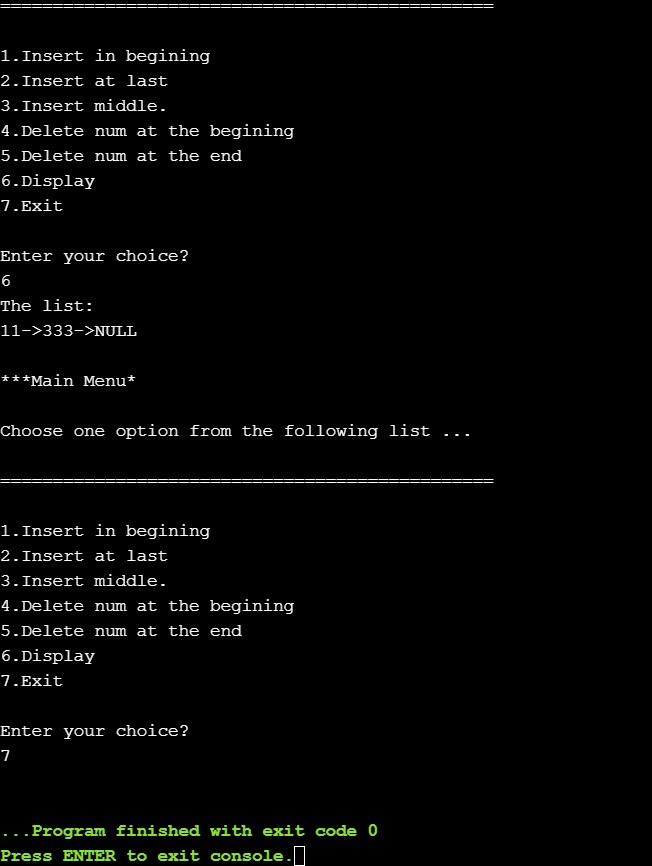
return 0;

}

Output:







Program 44:

Title: Creation of Doubly Linked list

Objective: Write a C program to create a doubly linked list with 5 nodes.

Explanation:

Doubly linked list is a type of linked list in which each node apart from storing its data

has two links. The first link points to the previous node in the list and the second link

points to the next node in the list. The first node of the list has its previous link

pointing to NULL similarly the last node of the list has its next node pointing to NULL

Code:

#include <stdio.h>

#include <stdlib.h>

//double linked list node

typedef struct node

{

int data;

struct node \*next;

struct node \*prev;

}node;

node\* createNode( int data){

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

newNode->data = data;

newNode->next = NULL;

newNode->prev = NULL;

return newNode;

}

//function which creates a double list of 5 nodes

node\* createList(){

int n= 5 ,data;

node \*p, \*head = NULL,\*temp;

//runs loop 5 times

while (n--){

printf( "Enter a number\n" );

scanf( "%d" ,&data);

//for the first node

if (head == NULL){

//intializing newnode as head

head = createNode(data);

p = head;

}

//fpr other nodes

else {

temp = createNode(data);

p->next = temp;

temp->prev = p;

p = p->next;

}

}

//return the list of 5nodes

return head;

}

//displays nodes from head to the end

void display (node \*head){

while (head != NULL){

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

head = head->next;

}

printf( "\n" );

} //main

int main (){

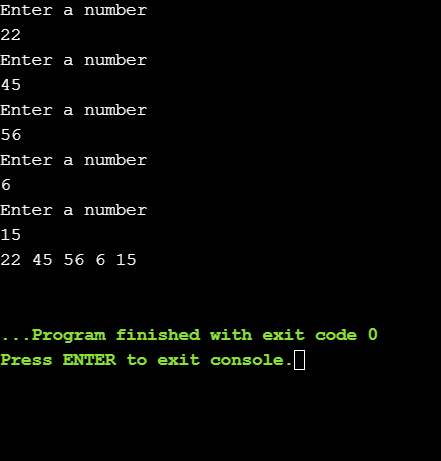
node \*head = createList();

display(head);

return 0 ;

}

Output:



Program 45:

Title: Creation of Circular Linked list

Objective: Write a C program to create a circular linked list with 5 nodes.

Explanation:

Circular Linked List is a variation of Linked list in which the first element points to the

last element and the last element points to the first element. So the only change in

creation function from linked list, is that the nth node instead of pointing to NULL. It

points to head.

Code:

#include <stdio.h>

#include <stdlib.h>

//declaration of a node

typedef struct Node{

int data;

struct Node \*next;

}node;

//function which creates nodes

node\* createNode( int data){

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

n->data = data;

n->next = NULL;

return n;

}

//function which creates a circualar list of 5 nodes

node\* createList(){

int n= 5 ,data;

node \*p, \*head = NULL;

//runs loop 5 times

while (n--){

printf( "Enter a number\n" );

scanf( "%d" ,&data);

if (head == NULL){

//intializing newnode as head

head = createNode(data);

p = head;

}

else {

p->next = createNode(data);

p = p->next;

}

}

//intializes the last eleemt next to head

p->next = head;

//return the list of 5nodes

return head;

}

//displays elemets in linked list till reaches the head

void display (node \*head){

node \*ptr = head;

//if list is not empty

if (head != NULL){

while (head->next != ptr){

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

head = head->next;

}

printf( "%d connected to %d" ,head->data,ptr->data);

}

//if list is empty

else

printf( "NULL\n" );

}

//main

int main ()

{

//driver code

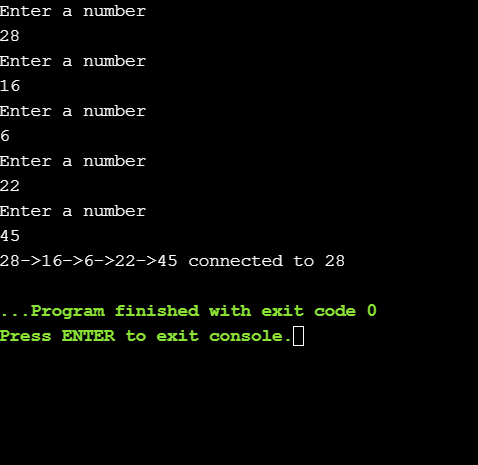
node \*head=createList();

display(head);

return 0 ;

}

Output:



Program 46:

Title: Implementation Stack Using Linked Lists

Objective: Write a C program to implement the stack using linked lists

Explanation:

The push operator is similar to the insertion in the begging in linked list. An dthe pop operator similar to the seletion from end of the linked list(refer problem 43)

Code:

#include <stdio.h>

#include <stdlib.h>

//declaration of a node

typedef struct Node{

int data;

struct Node \*next;

}node;

//function which creates nodes

node\* createNode( int data){

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

n->data = data;

n->next = NULL;

return n;

}

//pushes a node into the stack

node\* push(node \*head, int data){

node \*newNode = createNode(data);

newNode->next = head;

return newNode;

}

//pops an element from the stack

node\* pop(node \*head){

node \*temp = head;

//node empty thus returns null

if (head == NULL){

printf( "Empty\n" );

return NULL;

}

//deletes the node

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

free(temp);

//returns the stack after popping

return head->next;

}

//prints the top element

void peek (node \*head){

if (head == NULL) {

printf( "empty\n" );

return ;

}

printf( "%d\n" ,head->data);

}

//displays the stack from top to the end

void display (node \*head){

while (head!=NULL){

printf( "%d\n" ,head->data);

head = head->next;

}

}

int main (){

//intialized variables needed

node \*top = NULL;

int choice,data;

//runs loop till user chooses exit --> 5

while ( 1 ){

//menu

printf( "\n1. Push an element on to the STACK.\n"

"2. Pop and element from the STACK.\n"

"3. Peek the STACK.\n"

"4. Display the STACK.\n"

"5. Exit the program.\n" );

scanf( "%d" ,&choice);

//performs action according the choice

switch (choice){

case 1 :{

printf( "\nEnter an element to add\n" );

scanf( "%d" ,&data);

top = push(top,data);

break ;

}

case 2 :{

top = pop(top);

break ;

}

case 3 :{

peek(top);

break ;

}

case 4 :{

display(top);

break ;

}

case 5 :{

exit( 0 );

break ;

}

default : printf( "no such option choose again\n" );

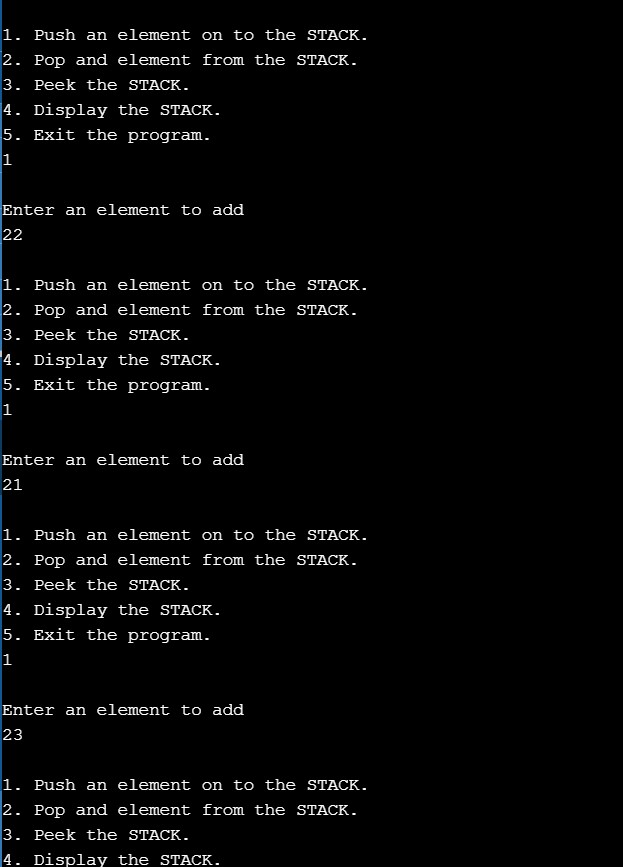
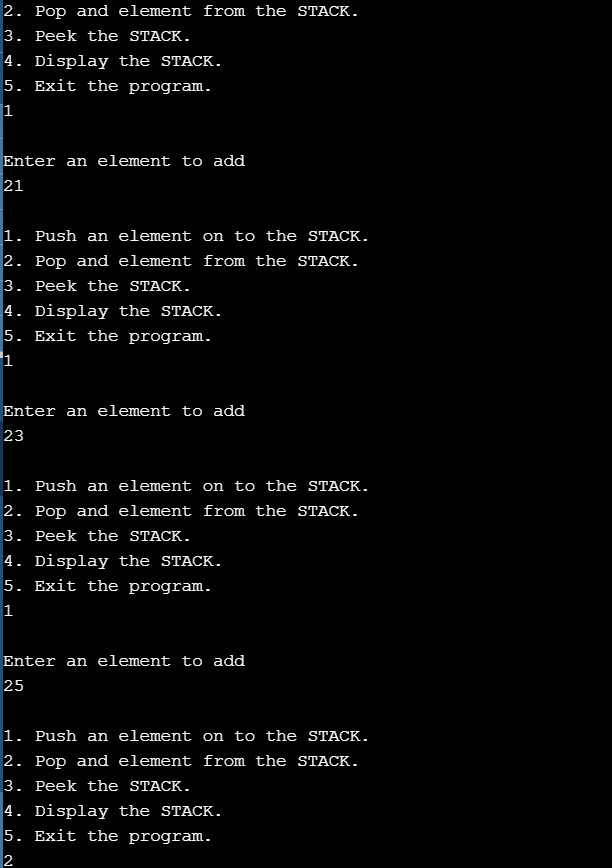
}

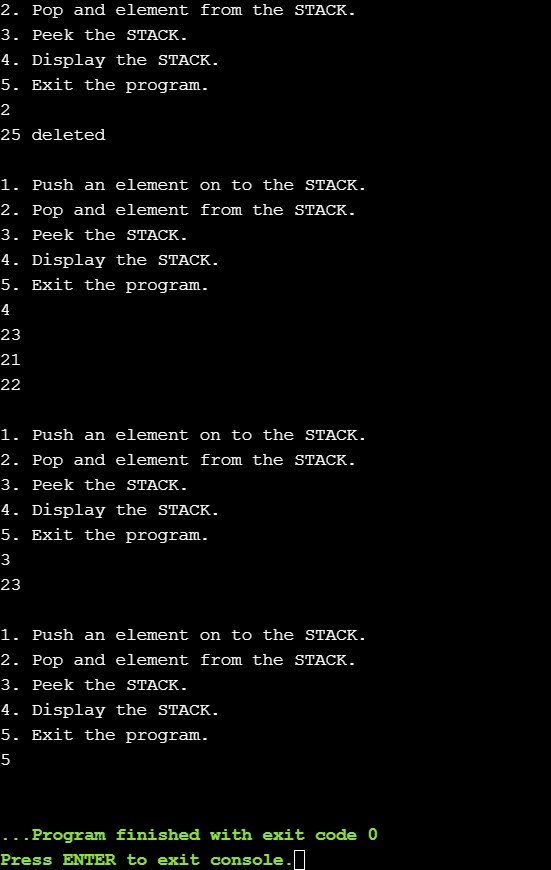
}

return 0 ;

}

Output:



Program 47:

Title: Implementing queues using LInked lists

Objective: Write a C program to implement the queue using linked list.

Explanation:

The enqueue is similar to insertion in begging and dequeue is similar to deletion in

beginning in linked list. SInce we have refernce point of where to delete in both cases

Code:

#include<stdio.h>

#include <stdlib.h>

//declaration of a node

typedef struct Node{

int data;

struct Node \*next;

}node;

struct Queue {

node \*front, \*rear;

};

//function which creates nodes

node\* createNode( int data){

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

n->data = data;

n->next = NULL;

return n;

}

void enQueue ( struct Queue\* q, int data)

{

// Create a new LL node

node\* temp = createNode(data);

if (q->rear == NULL) {

q->front = q->rear = temp;

return ;

}

q->rear->next = temp;

q->rear = temp;

}

void deQueue ( struct Queue\* q)

{

if (q->front == NULL)

return ;

node\* temp = q->front;

q->front = q->front->next;

if (q->front == NULL)

q->rear = NULL;

free(temp);

}

void display ( struct Queue \*q){

node \*save = q->front;

while (q->front != q->rear){

printf( "%d-> " ,q->front->data);

q->front = q->front->next;

}

printf( "%d\n" ,q->rear->data );

q->front = save;

}

int main ( int argc, char const \*argv[])

{

//intialized variables needed

struct Queue \*q = (( struct Queue\*)malloc( sizeof ( struct Queue)));

int choice,data;

//runs loop till user chooses exit --> 5

while ( 1 ){

//menu

printf( "\n1. EnQueue an element on to the STACK.\n"

"2. Dequeue and element from the STACK.\n"

"3. Display the STACK.\n"

"4. Exit the program.\n" );

scanf( "%d" ,&choice);

//performs action according the choice

switch (choice){

case 1 :{

printf( "\nEnter an element to add\n" );

scanf( "%d" ,&data);

enQueue(q, data);

break ;

}

case 2 :{

deQueue(q);

break ;

}

case 3 :{

display(q);

break ;

}

case 4 :{

exit( 0 );

break ;

}

default : printf( "no such option choose again\n" );

}

}

return 0 ;

}

Output:

