**Sorting : BUBBLE SORT**

***Aim : To implement bubble sort using c***

***Algorithm :***

1. *begin BubbleSort(list)*
2. *for all elements of list*
3. *if list[i] > list[i+1]*
4. *swap(list[i], list[i+1])*
5. *end if*
6. *end for*
7. *return list*
8. *end BubbleSort*

***Program :***

#include<stdio.h>

#include<conio.h>

int main(){

int a[100], i,j,temp;

printf("Enter 10 Elements\n");

for(i=0;i<10;i++){

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

}

printf("\nUnSorted : ");

for(i=0;i<10;i++){

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

}

printf("\nSorted : ");

for(i=0;i<10;i++){

for(j=0;j<10;j++){

if(a[j]>=a[j+1]){

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

}

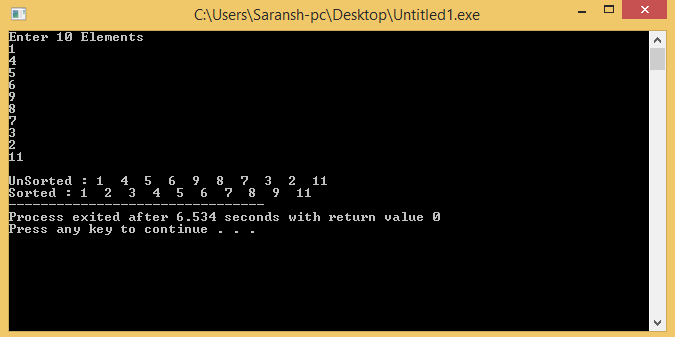
for(i=0;i<10;i++){

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

}

}

***OUTPUT :***



***RESULT :*** The program was compiled and executed successfully.

**SORTING : Insertion sort**

***Aim : To implement insertion sort in c***

***Algorithm*** :

*Step 1 − If it is the first element, it is already sorted. return 1;*

*Step 2 − Pick next element*

*Step 3 − Compare with all elements in the sorted sub-list*

*Step 4 − Shift all the elements in the sorted sub-list that is greater than the value to be sorted*

*Step 5 − Insert the value*

*Step 6 − Repeat until list is sorted*

***Program :***

#include<stdio.h>

#include<conio.h>

int main(){

int a[100], i,j,temp;

printf("Enter 10 Elements\n");

for(i=0;i<10;i++){

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

}

printf("\nUnSorted : ");

for(i=0;i<10;i++){

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

}

printf("\nSorted : ");

for(i=0;i<10;i++){

j=i;

while(a[j]<=a[j-1]&&j>0){

temp=a[j];

a[j]=a[j-1];

a[j-1]=temp;

j--;

}

}

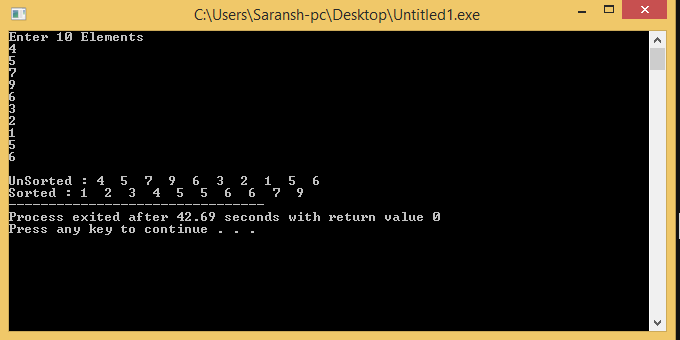
for(i=0;i<10;i++){

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

}

}

***Output :***



***Result :*** The program was compiled and executed successfully

**Search : Linear Search**

***Aim : To implement liner search algorithm in C***

***Algorithm :***

*Linear Search ( Array A, Value x)*

*Step 1: Set i to 1*

*Step 2: if i > n then go to step 7*

*Step 3: if A[i] = x then go to step 6*

*Step 4: Set i to i + 1*

*Step 5: Go to Step 2*

*Step 6: Print Element x Found at index i and go to step 8*

*Step 7: Print element not found*

*Step 8: Exit*

***Program :***

#include<stdio.h>

#include<conio.h>

int main(){

int i,count=0, key, a[10];

printf("Enter 10 Elements \n");

for(i=0;i<10;i++){

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

}

for(i=0;i<10;i++){

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

}

printf("\nEnter the element to be searched : ");

scanf("%d", &key);

for(i=0;i<10;i++){

if(key == a[i]){

count++;

}

}

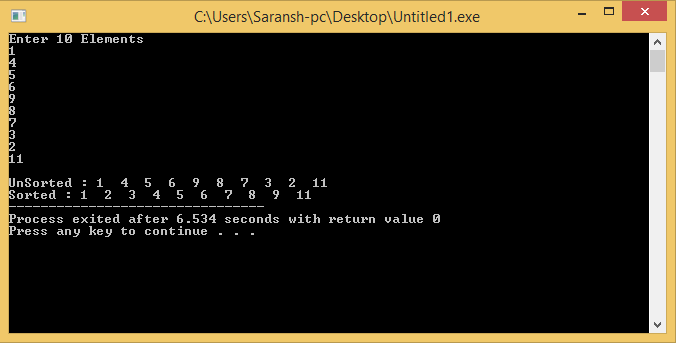
if(count!=0)

printf("The element was found\n");

else printf("Element was not found\n");

}

***Output :***



***Result :*** The program was compiled and executed successfully.

**Search : Binary Search**

***Aim : To implement Binary Search algorithm in C.***

***Algorithm :***

Procedure binary\_search

1. *Set lowerBound = 1, Set upperBound = n*
2. *while x not found*
3. *if upperBound < lowerBound*
4. *EXIT: x does not exists.*
5. *set midPoint = lowerBound + ( upperBound - lowerBound ) / 2*
6. *if A[midPoint] < x*
7. *set lowerBound = midPoint + 1*
8. *if A[midPoint] > x*
9. *set upperBound = midPoint - 1*
10. *if A[midPoint] = x*
11. *EXIT: x found at location midPoint*
12. *end while*

***Program :***

#include<stdio.h>

#include<conio.h>

int main(){

int i,count=0, key, a[10];

printf("Enter 10 Elements \n");

for(i=0;i<10;i++){

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

}

for(i=0;i<10;i++){

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

}

printf("\nEnter the element to be searched : ");

scanf("%d", &key);

for(i=0;i<10;i++){

if(key == a[i]){ count++; }

}

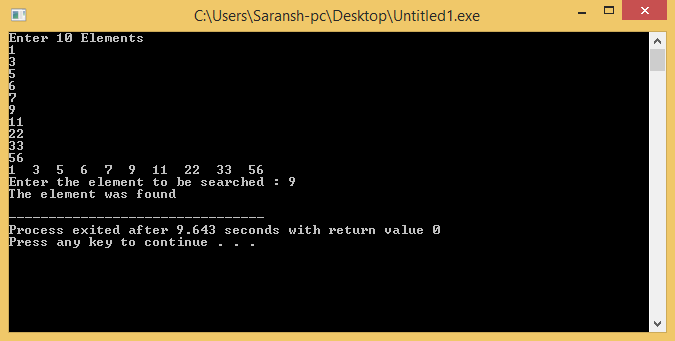
if(count!=0)

printf("The element was found\n");

else printf("Element was not found\n");

}

***Output :***



***Result :*** The program was compiled and executed successfully.

**Linked List : Singly**

***Aim : To implement singly linked list Algorithm in C***

***Algiorithm :***

*Traversal :*

1. *Initialize ptr = start;*
2. *Repeat step 3&4 until ptr != NULL*
3. *Print ptr->data;*
4. *Ptr=ptr->next;*
5. *Exit;*

*Insertion at the beginning :*

1. *Allocate memory fro temp*
2. *Temp->data=num;*
3. *temp->next=start;*
4. *Start=temp;*
5. *Exit;*

*Insertion At the end:*

1. *Initialize ptr=start;*
2. *Repeat step 3 until ptr->next !=NULL;*
3. *Ptr=ptr->next;*
4. *Allocate memory for temp;*
5. *Temp->data=num;*
6. *Temp->next=NULL;*
7. *Ptr->next =temp;*
8. *Exit;*

*Delete a node :*

1. *Initialize \*n and \*prev;*
2. *Repeat step 3 & 4 until n!=NULL;*
3. *If n->data == num;*

*Prev->next=n->next;*

1. *Else*

*Prev=n;*

*N=n->next;*

*Exit;*

***Program :***

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

struct node{

int data;

struct node \*next;

}\*head;

int delete\_start(struct node \*n)

{

n=head;

if(head==NULL)

{

printf("\nLIST IS EMPTY\n");

return 0;

}

n=n->next;

head=n;

}

int delete\_end(struct node \*n)

{

struct node \*prev;

n=head;

if(head==NULL)

{

printf("\nLIST IS EMPTY\n");

return 0;

}

while(n->next!=NULL)

{

prev=n;

n=n->next;

}

n=prev;

n->next=NULL;

};

int delete\_n(struct node \*n, int num)

{

struct node \*prev;

n=head;

if(head==NULL)

{

printf("\nLIST IS EMPTY\n");

return 0;

}

while(n!=NULL)

{

if(n->data==num)

{

prev->next=n->next;

return 0;

}

else

{

prev=n;

n=n->next;

}

}

};

int add\_at\_loc(int loc, int num)

{

int x=1;

struct node \*temp,\*ptr;

ptr=head;

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

temp->data=num;

if(head==NULL)

{

return 0;

}

while(x<=loc-2)

{

ptr=ptr->next;

if(ptr==NULL)

{

printf("Element not found in list");

}

x++;

}

temp->next=ptr->next;

ptr->next=temp;

ptr=temp;

}

int display(struct node \*n)

{

n=head;

if(n==NULL)

{

printf("List is empty\n");

return 0;

}

else

{

while(n != NULL)

{

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

n=n->next;

}

}

}

int add\_end(int num)

{

struct node \*temp, \*ptr;

ptr=head;

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

temp->data=num;

if(head==NULL)

{

printf("\nList already empty\nAdding element...\n");

temp->next=NULL;

head=temp;

return 0;

}

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=temp;

ptr=temp;

ptr->next=NULL;

}

int add\_start(int num)

{

struct node \*temp, \*ptr;

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

ptr=head;

temp->data=num;

if(head==NULL)

{

printf("\nList already empty\nAdding element...\n");

temp->next=NULL;

head=temp;

}

else

{

temp->next=ptr;

head=temp;

}

}

int main()

{

struct node \*n;

head = NULL;

int i,j, loc, num;

for(;;)

{

printf("\nYOUR CHOICE.....\n");

printf("\nPress 1 to add\nPress 2 to delete\nPress 3 to display\nPress 4 to exit\n");

scanf("%d", &i);

switch (i)

{

case 1 :

printf("SUB CHOICES....\n");

printf("Press 1 to Add at the start\n");

printf("Press 2 to Add at the end\n");

printf("Press 3 to Add in between\n");

scanf("%d", &j);

switch(j)

{

case 1 :

printf("Enter the number to be added : ");

scanf("%d", &num);

add\_start(num);

break;

case 2 :

printf("Enter the number to be added : ");

scanf("%d", &num);

add\_end(num);

break;

case 3 :

display(n);

printf("\nEnter the position\n");

scanf("%d", &loc);

printf("\nEnter the number to be entered : ");

scanf("%d", &num);

add\_at\_loc(loc,num);

break;

}

break;

case 2 :

printf("SUB CHOICES....\n");

printf("Press 1 to Delete at the start\n");

printf("Press 2 to Delete at the end\n");

printf("Press 3 to Delete the certain element\n");

scanf("%d", &j);

switch(j)

{

case 1 :

delete\_start(n);

break;

case 2 :

delete\_end(n);

break;

case 3 :

printf("Enter the element to be deleted\n");

scanf("%d", &num);

delete\_n(n, num);

break;

}

break;

case 3 :

display(n);

break;

case 4 :

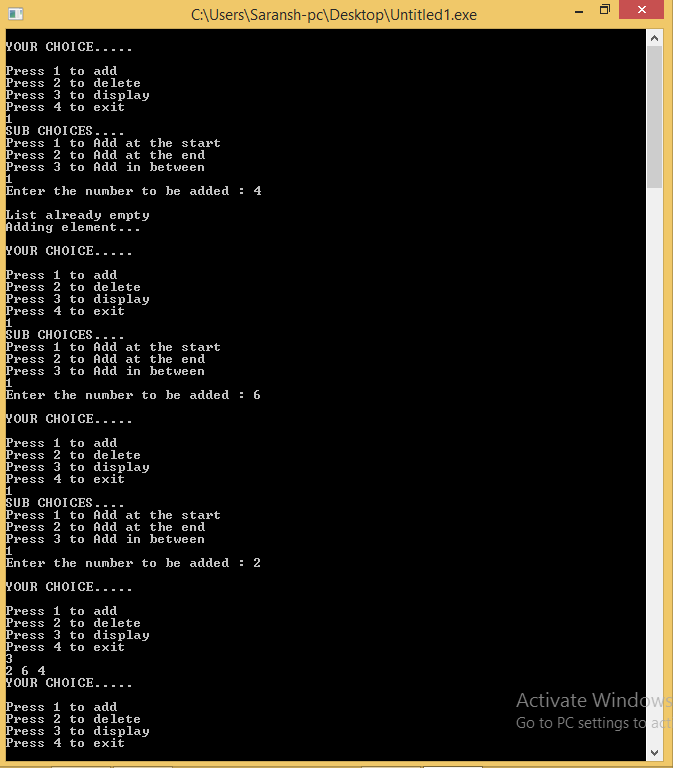
return 0;

}

}

}

***Output :***



***Result*** : The program was compiled and executed successfully.

**Linked list : Doubly**

***Aim : To implement Doubly linked list algorithm in C***

***Algorithm :***

*Insertion :*

1. *link->key = key;*
2. *link->data = data;*
3. *If isEmpty is true*
4. *last = link;*
5. *else head->prev = link;*
6. *link->next = head;*
7. *head = link;*

*Deletion :*

1. *Set tempLink = head;*
2. *If head->next == NULL*
3. *last = NULL;*
4. *}else {*
5. *head->next->prev = NULL;*
6. *head = head->next;*
7. *return tempLink;*

***Program :***

#include<iostream>

#include<stdlib.h>

using namespace std;

struct node{

int data;

struct node \*prev, \*next;

}\*head=NULL, \*ptr, \*temp, \*last=NULL;

int display\_forward(){

ptr=head;

if(head==NULL){

printf("Lst is Empty\n");

return 0;

}

while(ptr!=NULL){

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

ptr=ptr->next;

}

}

int display\_reverse(){

ptr=last;

if(last==NULL){

printf("Lst is Empty\n");

return 0;

}

while(ptr!=NULL){

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

ptr=ptr->prev;

}

}

int add\_start(int x){

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

temp->data=x;

if(head==NULL){

last=temp;

head=temp;

head->prev=NULL;

head->next=NULL;

}else{

temp->next=head;

head->prev=temp;

temp->prev=NULL;

head=temp;

}

}

int add\_last(int x){

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

temp->data=x;

if(head==NULL){

last=temp;

head=temp;

head->prev=NULL;

head->next=NULL;

}else{

last->next=temp;

temp->prev=last;

temp->next=NULL;

last=temp;

}

}

int delete\_num(int x){

ptr=head;

if(head==NULL){

printf("List is empty\n");

return 0;

}

while(ptr->data!=x){

ptr=ptr->next;

}

if(ptr==last){

ptr=ptr->prev;

last=ptr;

last->next=NULL;

}

if(ptr==head){

head=NULL;

last=NULL;

}

else{

ptr->next->prev=ptr->prev;

ptr->prev->next=ptr->next;

}

}

int main(){

int x, num;

add\_start(4);

add\_start(5);

add\_start(6);

add\_start(8);

add\_start(3);

add\_start(7);

printf("\nForward : ");

display\_forward();

printf("\nReverse : ");

display\_reverse();

printf("\n\nDeleteting a value : 6\n");

delete\_num(6);

printf("\nForward : ");

display\_forward();

printf("\nReverse : ");

display\_reverse();

printf("\n\nDeleteting a value : 8\n");

delete\_num(8);

printf("\nForward : ");

display\_forward();

printf("\nReverse : ");

display\_reverse();

printf("\n\nAfter adding at last::\n");

add\_last(45);

add\_last(66);

add\_last(99);

add\_last(21);

add\_last(34);

printf("\nForward : ");

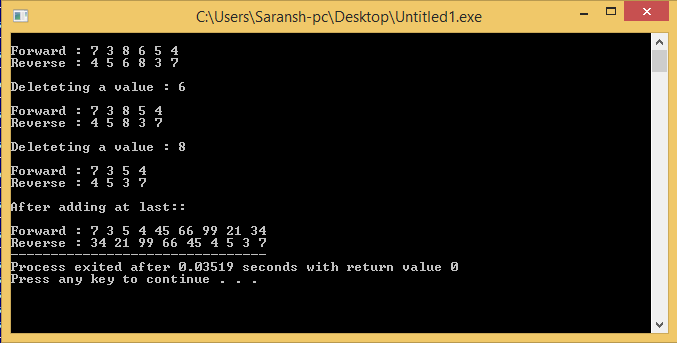
display\_forward();

printf("\nReverse : ");

display\_reverse();

}

***Output :***



***Result :*** The program compiled end executed successfully.

**Linked List : Circular**

***Aim : To Implement circular Linked list algorithm in C***

***Algorithm :***

*Insertion* :

1. *link->key = key;*
2. *link->data= data;*
3. *if (isEmpty()) {*
4. *head = link;*
5. *head->next = head;*
6. *}else {*
7. *link->next = head;*
8. *head = link;*

*Deletion :*

1. *set \*tempLink = head;*
2. *if(head->next == head){*
3. *head = NULL;*
4. *return tempLink;*
5. *head = head->next;*
6. *return tempLink;*

*Display :*

1. *set \*ptr = head;*
2. *if(head != NULL) {*
3. *while(ptr->next != ptr)*
4. *Print ptr->key,ptr->data*
5. *ptr = ptr->next;*

***Program :***

#include<iostream>

#include<stdlib.h>

using namespace std;

struct node{

int data;

struct node \*prev, \*next;

}\*head=NULL, \*ptr, \*temp, \*last=NULL;

int display\_forward(){

ptr=head;

if(head==NULL){

printf("Lst is Empty\n");

return 0;

}

while(ptr!=NULL){

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

ptr=ptr->next;

}

}

int display\_reverse(){

ptr=last;

if(last==NULL){

printf("Lst is Empty\n");

return 0;

}

while(ptr!=NULL){

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

ptr=ptr->prev;

}

}

int add\_start(int x){

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

temp->data=x;

if(head==NULL){

last=temp;

head=temp;

head->prev=NULL;

head->next=NULL;

}else{

temp->next=head;

head->prev=temp;

temp->prev=NULL;

head=temp;

}

}

int add\_last(int x){

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

temp->data=x;

if(head==NULL){

last=temp;

head=temp;

head->prev=NULL;

head->next=NULL;

}else{

last->next=temp;

temp->prev=last;

temp->next=NULL;

last=temp;

}

}

int delete\_num(int x){

ptr=head;

if(head==NULL){

printf("List is empty\n");

return 0;

}

while(ptr->data!=x){

ptr=ptr->next;

}

if(ptr==last){

ptr=ptr->prev;

last=ptr;

last->next=NULL;

}

if(ptr==head){

head=NULL;

last=NULL;

}

else{

ptr->next->prev=ptr->prev;

ptr->prev->next=ptr->next;

}

}

int main(){

int x, num;

add\_start(4);

add\_start(5);

add\_start(6);

add\_start(8);

add\_start(3);

add\_start(7);

printf("\nForward : ");

display\_forward();

printf("\nReverse : ");

display\_reverse();

printf("\n\nDeleteting a value : 6\n");

delete\_num(6);

printf("\nForward : ");

display\_forward();

printf("\nReverse : ");

display\_reverse();

printf("\n\nDeleteting a value : 8\n");

delete\_num(8);

printf("\nForward : ");

display\_forward();

printf("\nReverse : ");

display\_reverse();

printf("\n\nAfter adding at last::\n");

add\_last(45);

add\_last(66);

add\_last(99);

add\_last(21);

add\_last(34);

printf("\nForward : ");

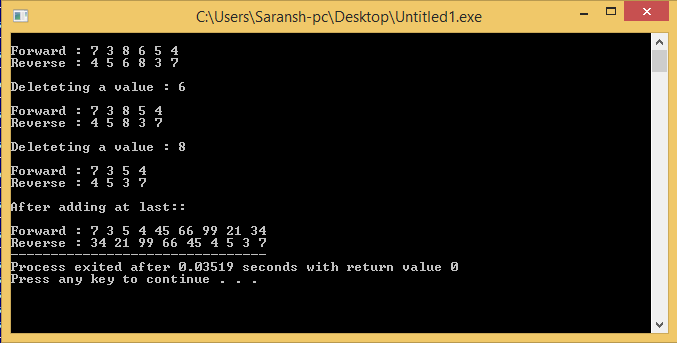
display\_forward();

printf("\nReverse : ");

display\_reverse();

}

***Output :***



***Result :*** The program compiled and executed successfully.

**STACKS : Using Linked list**

***Aim :*** To implement stacks algorithm in C.

***Algorithm :***

*PUSH* :

*begin procedure push: stack, data*

1. *if stack is full*
2. *return null*
3. *endif*
4. *top ← top + 1*
5. *stack[top] ← data*
6. *end procedure*

*POP :*

*begin procedure pop: stack*

1. *if stack is empty*
2. *return null*
3. *endif*
4. *data ← stack[top]*
5. *top ← top - 1*
6. *return data*
7. *end procedure*

***Program :***

#include<iostream>

#include<stdlib.h>

using namespace std;

struct node

{

int data;

struct node \*next;

}\*head;

int display()

{

struct node \*n;

n=head;

if(head==NULL)

{

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

return 0;

}

while(n!=NULL)

{

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

n=n->next;

}

}

int push()

{

struct node \*n, \*temp;

int num;

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

printf("Enter the number to be pushed : ");

scanf("%d", &num);

temp->data=num;

if(head==NULL)

{

head=temp;

head->next=NULL;

return 0;

}

temp->next=head;

head=temp;

}

int pop()

{

struct node \*n;

n=head;

if(head==NULL)

{

printf("\nThe stack is already empty\n");

return 0;

}

n=n->next;

head=n;

}

int main()

{

int x;

for(;;)

{

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

scanf(“%d”,&x);

switch(x)

{

case 1:

push();

break;

case 2 :

pop();

break;

case 3 :

display();

break;

case 4 :

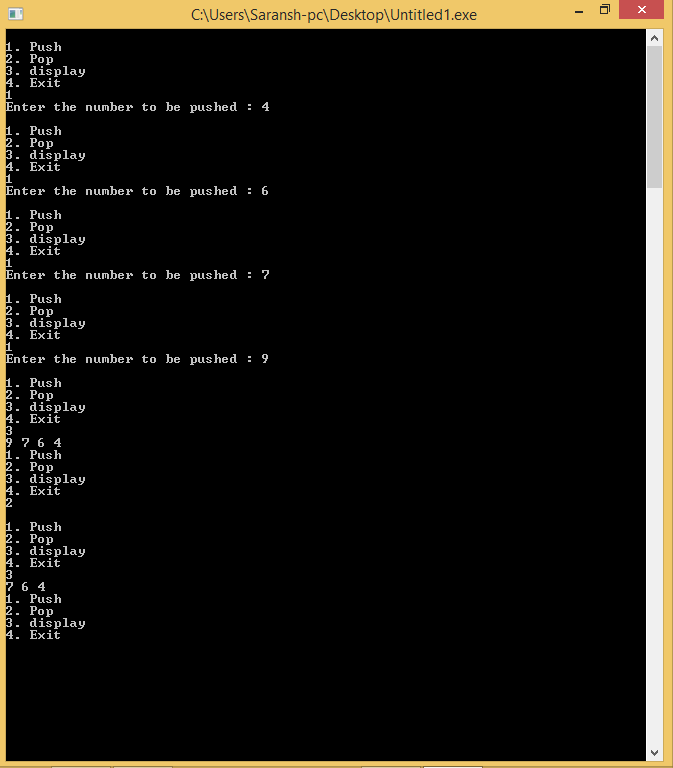
return 0;

}

}

}

***Output :***



***Result :*** The program was compiled and executed successfully.

**STACK : Using Array**

***Aim:*** To implement stacks algorithm using C.

***Algorithm :***

*PUSH :*

*begin procedure push: stack, data*

1. *if stack is full*
2. *return null*
3. *endif*
4. *top ← top + 1*
5. *stack[top] ← data*
6. *end procedure*

*POP :*

*begin procedure pop: stack*

1. *if stack is empty*
2. *return null*
3. *endif*
4. *data ← stack[top]*
5. *top ← top - 1*
6. *return data*
7. *end procedure*

***Program :***

#include <stdio.h>

#include<iostream>

using namespace std;

int MAXSIZE = 8;

int stack[8];

int top = -1;

int isempty() {

if(top == -1)

return 1;

else

return 0;

}

int isfull() {

if(top == MAXSIZE)

return 1;

else

return 0;

}

int peek() {

return stack[top];

}

int pop() {

int data;

if(!isempty()) {

data = stack[top];

top = top - 1;

return data;

}else {

printf("Could not retrieve data, Stack is empty.\n");

}

}

int push(int data) {

if(!isfull()) {

top = top + 1;

stack[top] = data;

}else {

printf("Could not insert data, Stack is full.\n");

}

}

int main() {

int x,num;

for(;;){

printf("Press 1 to Insert\n");

scanf(”%d”,&x);

if(x==1){

printf("\nEnter the number to be entered : ");

scanf(“%d”,&num);

push(num);

if(isfull())

break;

}

else break;

}

printf("Element at top of the stack: %d\n" ,peek());

printf("Elements: \n");

// print stack data

while(!isempty()) {

int data = pop();

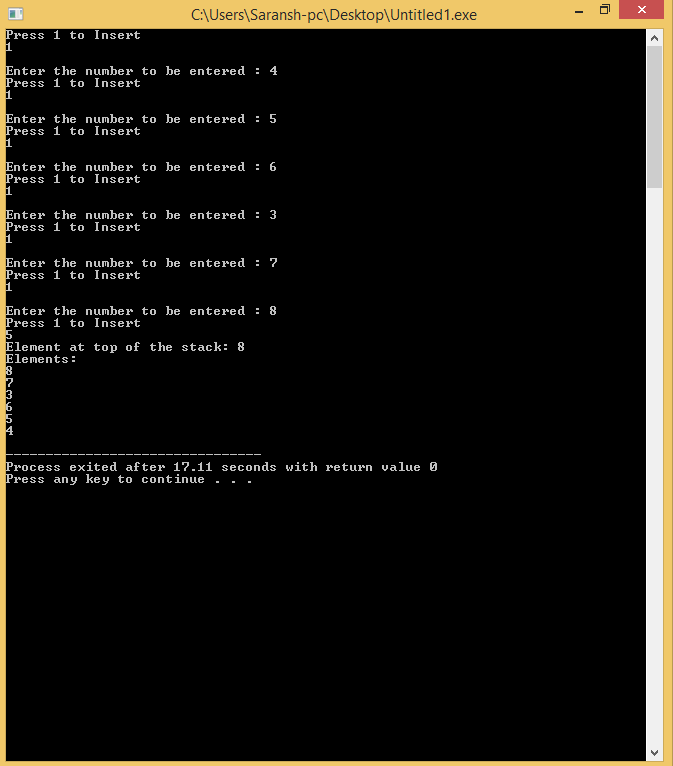
printf("%d\n",data);

}

return 0;

}

***Output :***



***Result :*** The program was compiled and executed successfully.

**QUEUE : Array implementation**

***Aim : To implement queue algorithm in C.***

***Algorithm*** :

*Enqueue :*

1. *if queue is full*
2. *return overflow*
3. *endif*
4. *rear ← rear + 1*
5. *queue[rear] ← data*
6. *return true*
7. *end procedure*

*Dequeue :*

1. *int enqueue(int data)*
2. *if(isfull())*
3. *return 0;*
4. *rear = rear + 1;*
5. *queue[rear] = data;*
6. *return 1;*
7. *end procedure*

***Program :***

#include <stdio.h>

#define MAX 10

int queue\_array[MAX];

int rear = - 1;

int front = - 1;

void display();

void insert();

void delet();

main()

{

int choice;

while (1) {

printf("1.Insert element to queue \n");

printf("2.Delete element from queue \n");

printf("3.Display all elements of queue \n");

printf("4.Quit \n");

printf("Enter your choice : ");

scanf("%d", &choice);

switch (choice) {

case 1:

insert();

break;

case 2:

delet();

break;

case 3:

display();

break;

case 4:

return 0;

default:

printf("Wrong choice \n");

} /\*End of switch\*/

} /\*End of while\*/

} /\*End of main()\*/

void insert(){

int add\_item;

if (rear == MAX - 1)

printf("Queue Overflow \n");

else{

if (front == - 1)

/\*If queue is initially empty \*/

front = 0;

printf("Inset the element in queue : ");

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

rear = rear + 1;

queue\_array[rear] = add\_item;

}

}

void delet(){

if (front == - 1 || front > rear){

printf("Queue Underflow \n");

return ;

}

else{

printf("Element deleted from queue is : %d\n", queue\_array[front]);

front = front + 1;

}

}

void display(){

int i;

if (front == - 1)

printf("Queue is empty \n");

else{

printf("Queue is : \n");

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

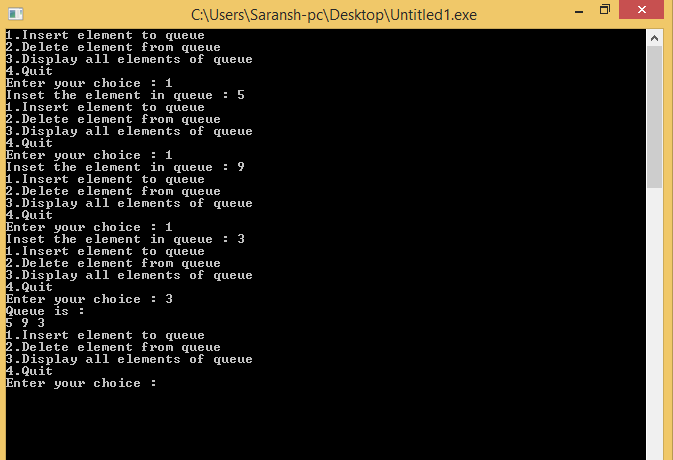
printf("%d ", queue\_array[i]);

printf("\n");

}

}

***Output :***



***Result :*** The program compiled and executed successfully.

**Application of Stacks and Queue : Infix to postfix.**

***Aim : To implement infix to postfix in C.***

***Algorithm :***

*Step 1 − scan the expression from left to right*

*Step 2 − if it is an operand push it to stack*

*Step 3 − if it is an operator pull operand from stack and perform operation*

*Step 4 − store the output of step 3, back to stack*

*Step 5 − scan the expression until all operands are consumed*

*Step 6 − pop the stack and perform operation*

***Program*** :

#include<stdio.h>

#include<string.h>

#include<ctype.h>

char stack[25];

int top = -1;

int stack\_int[25];

int top\_int = -1;

void push(char item){

stack[++top] = item;

}

char pop(){

return stack[top--];

}

int precedence(char symbol){

switch(symbol){

case '+':

case '-':

return 2;

break;

case '\*':

case '/':

return 3;

break;

case '^':

return 4;

break;

case '(':

case ')':

case '#':

return 1;

break;

}

}

int isOperator(char symbol){

switch(symbol){

case '+':

case '-':

case '\*':

case '/':

case '^':

case '(':

case ')':

return 1;

break;

default:

return 0;

}

}

void convert(char infix[],char postfix[]){

int i,symbol,j = 0;

stack[++top] = '#';

for(i = 0;i<strlen(infix);i++){

symbol = infix[i];

if(isOperator(symbol) == 0){

postfix[j] = symbol;

j++;

}

else{

if(symbol == '('){

push(symbol);

}

else{

if(symbol == ')') {

while(stack[top] != '(') {

postfix[j] = pop();

j++;

}

pop();

}

else{

if(precedence(symbol)>precedence(stack[top])) {

push(symbol);

}

else{

while(precedence(symbol)<=precedence(stack[top])){

postfix[j] = pop();

j++;

}

push(symbol);

}

}

}

}

}

while(stack[top] != '#') {

postfix[j] = pop();

j++;

}

postfix[j]='\0';

}

void push\_int(int item){

stack\_int[++top\_int] = item;

}

char pop\_int() {

return stack\_int[top\_int--];

}

int evaluate(char \*postfix){

char ch;

int i = 0,operand1,operand2;

while( (ch = postfix[i++]) != '\0') {

if(isdigit(ch)) {

push\_int(ch-'0');

}

else {

operand2 = pop\_int();

operand1 = pop\_int();

switch(ch) {

case '+':

push\_int(operand1+operand2);

break;

case '-':

push\_int(operand1-operand2);

break;

case '\*':

push\_int(operand1\*operand2);

break;

case '/':

push\_int(operand1/operand2);

break;

}

}

}

return stack\_int[top\_int];

}

int main() {

char infix[25] = "1\*(2+3)",postfix[25];

convert(infix,postfix);

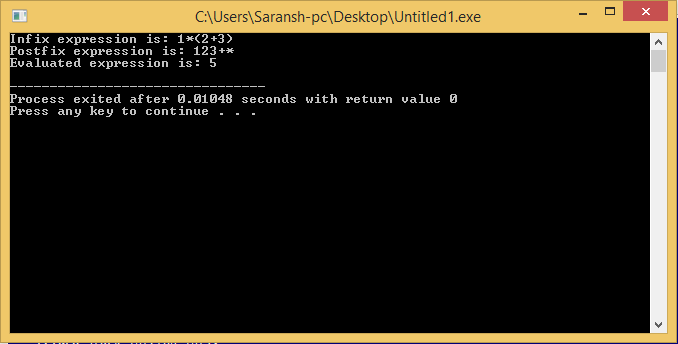
printf("Infix expression is: %s\n" , infix);

printf("Postfix expression is: %s\n" , postfix);

printf("Evaluated expression is: %d\n" , evaluate(postfix));

}

***Output :***



***Result :*** The program compiled and executed successfully.

**Application of Stacks and Queue : Priority Queue**

***Aim : To implement priority queue algorithm in C.***

***Algorithm :***

1. *Temp->info=item;*
2. *Temp->priority=priorty;*
3. *Check if front ==NULL || priority<front->priority*
4. *Do temp->link=front; front =temp;*
5. *Else q=front;*
6. *Repeat step 7 till q->link !=NULL && q->link->priority<=priority*
7. *q=q->link;*
8. *q->link=temp;*

***Program :***

#include <iostream>

#include <cstdio>

#include <cstring>

#include <cstdlib>

using namespace std;

struct node{

int priority;

int info;

struct node \*link;

}\*front;

void insert(int item, int priority){

node \*tmp=new node, \*q;

tmp->info = item;

tmp->priority = priority;

if (front == NULL || priority < front->priority){

tmp->link = front;

front = tmp;

}

else{

q = front;

while (q->link != NULL && q->link->priority <= priority)

q=q->link;

tmp->link = q->link;

q->link = tmp;

}}

void del(){

node \*tmp;

if(front == NULL)

printf(“Queue Underflow\n");

else{

tmp = front;

cout<<"Deleted item is: "<<tmp->info<<endl;

front = front->link;

free(tmp);

}}

void display(){

node \*ptr;

ptr = front;

if (front == NULL)

printf(“Queue is empty\n");

else{

printf(“Queue is :\n");

cout<<"Priority Item\n";

while(ptr != NULL){

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

ptr = ptr->link;

}}}

int main(){

front = NULL;

int choice, item, priority;

do{

printf(“1.Insert\n"<<"2.Delete\n"<<"3.Display\n"<<"4.Quit\n”);

printf(“Enter your choice : ");

scanf(“%d”,&choice);

switch(choice){

case 1:

printf(“Input the item value to be added in the queue : ");

scanf(“%d”,&item);

cout<<"Enter its priority : ";

scanf(“%d”,&priority);

insert(item, priority);

break;

case 2:

del(); break;

case 3:

display(); break;

case 4:

break;

default :

printf("Wrong choice\n"); break;

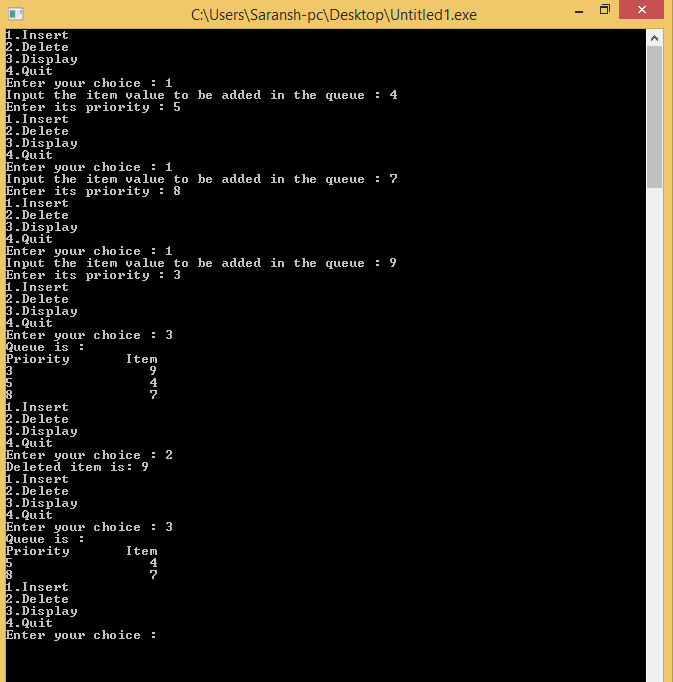
}}

while(choice != 4);

return 0;

}

***Output :***



***Result :*** The Program compiled and executed successfully.

**Tree Traversal : Binary Tree**

***Aim : To implement Binary tree traversal in C.***

***Algorithm :***

*In-order Traversal*

*Step 1 − Recursively traverse left subtree.*

*Step 2 − Visit root node.*

*Step 3 − Recursively traverse right subtree.*

*Pre-order Traversal*

*Step 1 − Visit root node.*

*Step 2 − Recursively traverse left subtree.*

*Step 3 − Recursively traverse right subtree.*

*Post-order Traversal*

*Step 1 − Recursively traverse left subtree.*

*Step 2 − Recursively traverse right subtree.*

*Step 3 − Visit root node.*

***Program :***

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*leftChild;

struct node \*rightChild;

};

struct node \*root = NULL;

void insert(int data) {

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

struct node \*current;

struct node \*parent;

tempNode->data = data;

tempNode->leftChild = NULL;

tempNode->rightChild = NULL;

//if tree is empty

if(root == NULL) {

root = tempNode;

}else {

current = root;

parent = NULL;

while(1) {

parent = current;

//go to left of the tree

if(data < parent->data) {

current = current->leftChild;

//insert to the left

if(current == NULL) {

parent->leftChild = tempNode;

return;

}

}//go to right of the tree

else {

current = current->rightChild;

//insert to the right

if(current == NULL) {

parent->rightChild = tempNode;

return;

}

}

}

}

}

struct node\* search(int data) {

struct node \*current = root;

printf("Visiting elements: ");

while(current->data != data) {

if(current != NULL)

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

//go to left tree

if(current->data > data) {

current = current->leftChild;

}

//else go to right tree

else {

current = current->rightChild;

}

if(current==NULL)

{

return NULL;

}

else

{

return current;

}

}

}

void pre\_order\_traversal(struct node\* root) {

if(root != NULL) {

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

pre\_order\_traversal(root->leftChild);

pre\_order\_traversal(root->rightChild);

}

}

void inorder\_traversal(struct node\* root) {

if(root != NULL) {

inorder\_traversal(root->leftChild);

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

inorder\_traversal(root->rightChild);

}

}

void post\_order\_traversal(struct node\* root) {

if(root != NULL) {

post\_order\_traversal(root->leftChild);

post\_order\_traversal(root->rightChild);

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

}

}

int main()

{

int i;

int array[7] = { 27, 14, 35, 10, 19, 31, 42 };

for(i = 0; i < 7; i++)

insert(array[i]);

i = 31;

struct node \* temp = search(i);

if(temp != NULL)

{

printf("[%d] Element found.", temp->data);

printf("\n");

}

else

{

printf("[ x ] Element not found (%d).\n", i);

}

i = 15;

temp = search(i);

if(temp != NULL)

{

printf("[%d] Element found.", temp->data);

printf("\n");

}

else

{

printf("[ x ] Element not found (%d).\n", i);

}

printf("\nPreorder traversal: ");

pre\_order\_traversal(root);

printf("\nInorder traversal: ");

inorder\_traversal(root);

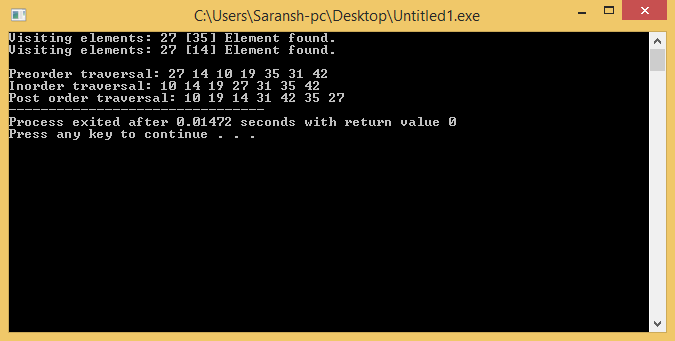
printf("\nPost order traversal: ");

post\_order\_traversal(root);

return 0;

}

***Output :***



***Result :*** The program compiled and executed successfully.

**Minimum Spanning tree : Prims Algorithm**

***Aim : To implement concept of minimum spanning tree using prims algorithm in C.***

***Algorithm*** :

1. *Create a set.*
2. *Assign elements to the matrix.*
3. *Assign 0 to first vertex so that it is picked first.*
4. *While set doesn’t include all the elements,*
5. *Pick a vertex u which is not there in set and has minimum value.*
6. *Update all the values with the new vertexes.*

***Program :***

#include<stdio.h>

int a,b,u,v,n,i,j,ne=1;

int visited[10]={0},min,mincost=0,cost[10][10];

int main()

{

printf("\nEnter the number of nodes:");

scanf("%d",&n);

printf("\nEnter the adjacency matrix:\n");

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

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

{

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

if(cost[i][j]==0)

cost[i][j]=999;

}

visited[1]=1;

printf("\n");

while(ne < n)

{

for(i=1,min=999;i<=n;i++)

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

if(cost[i][j]< min)

if(visited[i]!=0)

{

min=cost[i][j];

a=u=I;

b=v=j;

}

if(visited[u]==0 || visited[v]==0)

{

printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);

mincost+=min;

visited[b]=1;

}

cost[a][b]=cost[b][a]=999;

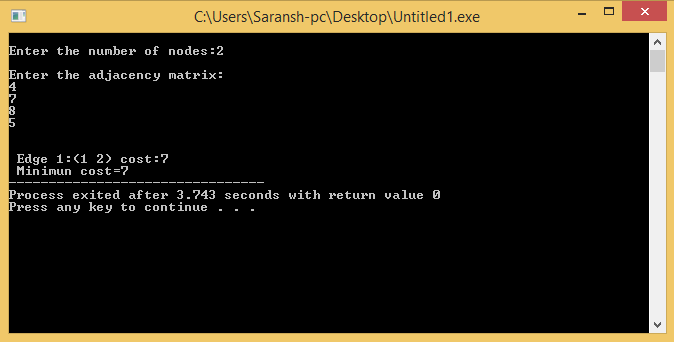
}

printf("\n Minimun cost=%d",mincost);

return 0;

}

***Output :***



***Result : The program compiled and executed successfully.***