**Program -1**

**Aim:** To Implement Bubble Sort.

**Source Code:**

#include<stdio.h>

#include<conio.h>

void main(){

int arr[20], n, i ,j ,temp;

printf("Enter the size of the array:\n");

scanf("%d",&n);

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

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

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

}

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

for(j = 0; j < n - 1 -i; j++) {

if(arr[j] > arr[j+1]) {

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

printf("Sorted Array:\n");

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

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

}

}

**Input:**

Enter the size of the array:

5

Enter the elements of the array:

1 -4 9 3 10

**Output:**

Sorted Array:

-4 1 3 9 10

**Program -3**

**Aim:** To Implement Insertion Sort.

**Source Code:**

#include<stdio.h>

#include<conio.h>

void main(){

int arr[20], n, i ,j ,temp, curr, prev;

printf("Enter the size of the array:\n");

scanf("%d",&n);

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

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

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

}

for(int i = 1; i < num.length; i++) {

curr = num[i];

prev = i-1;

while(prev >= 0 && num[prev] > curr) {

num[prev + 1] = num[prev];

prev--;

}

num[prev + 1] = curr;

}

printf("Sorted Array:\n");

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

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

}

}

**Input:**

Enter the size of the array:

5

Enter the elements of the array:

1 -4 9 3 10

**Output:**

Sorted Array:

-4 1 3 9 10

**Program -2**

**Aim:** To Implement Selection Sort.

**Source Code:**

#include<stdio.h>

#include<conio.h>

void main(){

int arr[20], n, i ,j ,temp, curr, prev;

printf("Enter the size of the array:\n");

scanf("%d",&n);

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

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

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

}

for(int i = 0; i < num.length - 1; i++) {

int minIdx = i;

for(int j = i+1; j < num.length; j++) {

if(num[minIdx] > num[j]){

minIdx = j;

}

}

int temp = num[i];

num[i] = num[minIdx];

num[minIdx] = temp;

}

printf("Sorted Array:\n");

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

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

}

}

**Input:**

Enter the size of the array:

5

Enter the elements of the array:

1 -4 9 3 10

**Output:**

Sorted Array:

-4 1 3 9 10

**Program -4**

**Aim:** To Implement Linear Search.

**Source Code:**

#include <stdio.h>

#include <conio.h>

void main(){

int arr[20], n, i, key;

printf("Enter the size of the Array:\n");

scanf("%d",&n);

printf("Enter The Elements of the Array:\n");

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

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

}

printf("Enter the No. to be found:\n");

scanf("%d",&key);

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

if(key == arr[i]) {

printf("Number Found on %d place\n",i+1);

break;

}

}

if(i >= n){

printf("Number not found");

}

getch();

}

**Input:**

Enter the size of the Array:

3

Enter The Elements of the Array:

1 3 4

Enter the No. to be found:

5

**Output:**

Number not found

**Program -5**

**Aim:** To Implement Binary Search.

**Source Code:**

#include<stdio.h>

#include<conio.h>

void binarySearch(int arr[], int n){

int x;

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

scanf("%d", &x);

int l = 0;

int r = n;

while(l < r){

int mid = (l+r)/2;

if(arr[mid] == x){

printf("Element found at %d position", mid+1);

break;

}else if(arr[mid] > x){

r = mid - 1;

}else{

l = mid + 1;

}

}

if(l > r){

printf("Element Not Found!");

}

}

int main()

{

int n;

printf("Enter the size of arr:\n");

scanf("%d", &n);

int arr[n+1];

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

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

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

}

binarySearch(arr, n);

return 0;

}

**Input:**

Enter the size of arr:

5

Enter the elements of the Array:

1 4 6 8 9

Enter the element:

6

**Output:**

Element found at 3 position

**Program -6**

**Aim:** Write a Program to insert an element at any position in the array.

**Source Code:**

#include <stdio.h>

#include <conio.h>

void insertion(int arr[], int n){

int idx;

int item;

printf("Enter the position:\n");

scanf("%d", &idx);

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

scanf("%d", &item);

for(int i = n; i >= idx; i--){

arr[i] = arr[i-1];

}

arr[idx - 1] = item;

}

void traverse(int arr[], int n){

printf("Array Elements are:\n");

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

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

}

}

int main()

{

int n;

printf("Enter the size of arr:\n");

scanf("%d", &n);

int arr[n+1];

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

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

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

}

insertion(arr, n);

return 0;

}

**Input:**

Enter the size of arr:

5

Enter the elements of the Array:

2 4 5 2 6

Enter the position:

3

Enter the item:

6

**Output:**

Array Elements are:

2 4 6 5 2 6

**Program -7**

**Aim:** Write a Program to delete an element at any position in the array.

**Source Code:**

#include <stdio.h>

#include <conio.h>

void deletion(int arr[], int n){

int idx;

printf("Enter the position:\n");

scanf("%d", &idx);

for(int i = idx - 1; i < n; i++){

arr[i] = arr[i + 1];

}

}

void traverse(int arr[], int n){

printf("Array Elements are:\n");

for(int i = 0; i < n-1; i++){

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

}

}

int main()

{

int n;

printf("Enter the size of arr:\n");

scanf("%d", &n);

int arr[n+1];

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

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

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

}

deletion(arr, n);

traverse(arr, n);

return 0;

}

**Input:**

Enter the size of arr:

5

Enter the elements of the Array:

2 4 5 6 7

Enter the position:

2

**Output:**

Array Elements are:

2 5 6 7

**Program -8**

**Aim:** Write a Program to implement push, pop and print in the stack using Array.

**Source Code:**

#include<stdio.h>

#include<conio.h>

#define max 100

static int stack[max];

static int top = -1;

void push();

void pop();

void print();

void push(){

int item;

if(top == max - 1){

printf("Stack Overflow!");

}else{

top = top+1;

printf("Enter the no -> ");

scanf("%d", &item);

stack[top] = item;

}

}

void pop(){

int item;

if(top == -1){

printf("Stack Underflow!");

}else{

item = stack[top];

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

top = top - 1;

}

}

void print(){

if(top == -1){

printf("Stack is Empty!");

}

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

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

}

}

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

int ch;

while (1)

{

printf("\nPress 1 to push()");

printf("\nPress 2 to pop()");

printf("\nPress 3 to print the stack");

printf("\nPress 4 to exit\n");

scanf("%d", &ch);

switch (ch)

{

case 1: push();

break;

case 2: pop();

break;

case 3: print();

break;

case 4: return 0;

default: printf("Invalid Choice!");

break;

}

}

return 0;

}

**Input & Output:**

Press 1 to push()

Press 2 to pop()

Press 3 to print the stack

Press 4 to exit

1

Enter the no -> 5

Press 1 to push()

Press 2 to pop()

Press 3 to print the stack

Press 4 to exit

1

Enter the no -> 3

Press 1 to push()

Press 2 to pop()

Press 3 to print the stack

Press 4 to exit

1

Enter the no -> 6

Press 1 to push()

Press 2 to pop()

Press 3 to print the stack

Press 4 to exit

1

Enter the no -> 7

Press 1 to push()

Press 2 to pop()

Press 3 to print the stack

Press 4 to exit

1

Enter the no -> 8

Press 1 to push()

Press 2 to pop()

Press 3 to print the stack

Press 4 to exit

3

**8 7 6 3 5**

Press 1 to push()

Press 2 to pop()

Press 3 to print the stack

Press 4 to exit

2

**8**

Press 1 to push()

Press 2 to pop()

Press 3 to print the stack

Press 4 to exit

3

**7 6 3 5**

Press 1 to push()

Press 2 to pop()

Press 3 to print the stack

Press 4 to exit

4

**Program -9**

**Aim:** Write a Program to implement push, pop and print in the stack using Linked List.

**Source Code:**

# include <stdio.h>

# include <stdlib.h>

int top =0;

int n=100;

int k=1;

struct Node{

    int data;

    struct Node\* next;

};

void push(struct Node\*\* head , int \*val){

    top++;

    if (top > n){

        k=0;

    }

    else{

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

    newnode->data = \*val;

    newnode->next = NULL;

    struct Node\* temp=\*head;

    struct Node\* prev= NULL;

    while(temp != NULL){

        temp = temp -> next;

        if (prev == NULL){

            prev = \*head;

        }

        else{

            prev = prev -> next;

        }

    }

    prev -> next = newnode;

    }

}

void pop(struct Node\*\* head){

    int x = top - 1;

    top = top -1;

    if (top < 0){

        k=0;

    }

    else{

    struct Node\* temp = \*head;

    while(temp != NULL && x>0){

        temp = temp -> next;

        x--;

    }

    temp -> next = NULL;

    }

}

void print (struct Node\*\* head){

    struct Node\* temp = \*head;

    while(temp != NULL){

        printf("%d ",temp -> data);

        temp = temp -> next;

    }

    printf("\n");

}

int main(){

    struct Node\* node1 = (struct Node\*) malloc(sizeof(struct Node));

    int d1;

    printf("first data to be pushed : ");

    scanf("%d",&d1);

    node1 -> data = d1;

    node1 -> next = NULL;

    struct Node\* head = node1;

    int check;

    while(1){

        printf("1: push \n2: pop \n3:print \ncheck : ");

        scanf("%d",&check);

        if (check == 1){

            int val;

            scanf("%d",&val);

            push(&head,&val);

            if (k==0){

                printf("stack overflow");

                break;

            }

        }

        else if (check == 2){

            pop(&head);

            if (k==0){

                printf("stack underflow");

                break;

            }

        }

        else if (check == 3){

            print(&head);

        break;

}

        else{

            printf("wrong check");

            break;

        }

    }

}

**Input & Output:**

first data to be pushed : 8

1: push

2: pop

3:print

check : 1

5

1: push

2: pop

3:print

check : 2

1: push

2: pop

3:print

check : 1

9

1: push

2: pop

3:print

check : 3

**8 9**

**Program -10**

**Aim:** Write a Program to implement Queue using Array.

**Source Code:**

#include <stdio.h>

#include <math.h>

#define max 10

static int Q[max];

static int front = -1;

static int rear = -1;

void add();

void delete();

void print();

void add()

{

if (front == 0 && rear == max - 1)

{

printf("Overflow");

return;

}

else if (front == rear + 1)

{

printf("Overflow");

return;

}

int item;

printf("Enter the item you want to add:");

scanf("%d", &item);

if (front == -1)

{

front = 0;

rear = 0;

}

else if (rear == max - 1)

{

rear = 0;

}

else

{

rear = rear + 1;

}

Q[rear] = item;

}

void delete()

{

if (front == -1)

{

printf("Underflow");

return;

}

int item = Q[front];

if (front == rear)

{

front = -1;

rear = -1;

}

else if (front == max - 1)

{

front = 0;

}

else

{

front = front + 1;

}

printf("Deleted item is: %d", item);

}

void print()

{

if (front == -1)

{

printf("Queue is Empty");

return;

}

printf("Queue Elements are: ");

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

{

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

}

}

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

{

int ch;

while (1)

{

printf("\nPress 1 to add()");

printf("\nPress 2 to delete()");

printf("\nPress 3 to print the queue");

printf("\nPress 4 to exit\n");

scanf("%d", &ch);

switch (ch)

{

case 1:

add();

break;

case 2:

delete ();

break;

case 3:

print();

break;

case 4:

return 0;

default:

printf("Invalid Choice!");

break;

}

}

return 0;

}

**Input & Output:**

Press 1 to add()

Press 2 to delete()

Press 3 to print the queue

Press 4 to exit

1

**Enter the item you want to add:6**

Press 1 to add()

Press 2 to delete()

Press 3 to print the queue

Press 4 to exit

1

**Enter the item you want to add:7**

Press 1 to add()

Press 2 to delete()

Press 3 to print the queue

Press 4 to exit

1

**Enter the item you want to add:8**

Press 1 to add()

Press 2 to delete()

Press 3 to print the queue

Press 4 to exit

1

**Enter the item you want to add:7**

Press 1 to add()

Press 2 to delete()

Press 3 to print the queue

Press 4 to exit

1

**Enter the item you want to add:10**

Press 1 to add()

Press 2 to delete()

Press 3 to print the queue

Press 4 to exit

1

**Enter the item you want to add:9**

Press 1 to add()

Press 2 to delete()

Press 3 to print the queue

Press 4 to exit

3

**Queue Elements are: 6 7 8 7 10 9**

Press 1 to add()

Press 2 to delete()

Press 3 to print the queue

Press 4 to exit

2

**Deleted item is: 6**

Press 1 to add()

Press 2 to delete()

Press 3 to print the queue

Press 4 to exit

2

**Deleted item is: 7**

Press 1 to add()

Press 2 to delete()

Press 3 to print the queue

Press 4 to exit

3

**Queue Elements are: 8 7 10 9**

Press 1 to add()

Press 2 to delete()

Press 3 to print the queue

Press 4 to exit

4

**Program -11**

**Aim:** Write a Program to convert the infix expression to postfix and prefix expression.

**Source Code:**

#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<limits.h>

#include<stdlib.h>

#define MAX 100

int top = -1;

char stack[MAX];

int isFull (){

return top == MAX - 1;

}

int isEmpty (){

return top == -1;

}

void push (char item){

if (isFull ())

return;

top++;

stack[top] = item;

}

int pop (){

if (isEmpty ())

return INT\_MIN;

return stack[top--];

}

int peek (){

if (isEmpty ())

return INT\_MIN;

return stack[top];

}

int checkIfOperand (char ch){

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

(ch >= 'A' && ch <= 'Z');

}

int precedence (char ch){

switch (ch){

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

}

return -1;

}

int getPostfix (char \*expression){

int i, j;

for (i = 0, j = -1; expression[i]; ++i){

if (checkIfOperand

(expression[i])){

expression[++j] =

expression[i];}

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

push (expression[i]); }

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

while (!isEmpty (stack) &&

peek (stack) != '(')

expression[++j] = pop

(stack);

if (!isEmpty (stack) &&

peek (stack) != '(')

return -1; // invalid

expression

else

pop (stack); }

else{

while (!isEmpty (stack) &&

precedence (expression[i]) <=

precedence (peek (stack)))

expression[++j] = pop

(stack);

push (expression[i]);

}}

while (!isEmpty (stack))

expression[++j] = pop (stack);

expression[++j] = '\0';

}

void reverse (char \*exp){

int size = strlen (exp);

int j = size, i = 0;

char temp[size];

temp[j--] = '\0';

while (exp[i] != '\0'){

temp[j] = exp[i];

j--;

i++; }

strcpy (exp, temp);

}

void brackets (char \*exp){

int i = 0;

while (exp[i] != '\0'){

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

exp[i] = ')';

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

exp[i] = '(';

i++;

}}

void InfixtoPrefix (char \*exp){

int size = strlen (exp);

reverse (exp);

brackets (exp);

getPostfix (exp);

reverse (exp);

}

int main (){

char expression[20];

int choice;

printf("Enter the infix

expression:");

gets(expression);

printf("1.Infix to Postfix\n2.Infix to

Prefix\nEnter your choice:");

scanf("%d",&choice);

switch(choice){

case 1: getPostfix(expression);

printf ("The postfix expression is:

");

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

break;

case 2: InfixtoPrefix (expression);

printf ("The prefix expression is:

");

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

break;

default: printf("Invalid choice");

}

}

**Input & Output:**

Enter the infix expression: A+(B\*C-(D/E^F)\*G)+H

1.Infix to Postfix

2.Infix to Prefix

Enter your choice : 1

The postfix expression is: ABC\*DEF^/G\*-+H+

Enter your choice : 2

The prefix is: +A+-\*BC\*/D^EFGH

**Program -12**

**Aim:** Write a Program to implement Queue using Linked List.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \* next;

};

struct node \* front = NULL;

struct node \* rear = NULL;

void enqueue(int value) {

struct node \* ptr;

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

ptr->data = value;

ptr->next = NULL;

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

front = rear = ptr;

} else {

rear->next = ptr;

rear = ptr;

}

}

int dequeue() {

if (front == NULL) {

printf("\nUnderflow\n");

return -1;

} else {

struct node \* temp = front;

int temp\_data = front->data;

front = front->next;

free(temp);

return temp\_data;

}

}

void display() {

struct node \* temp;

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

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

} else {

printf("The queue is \n");

temp = front;

while (temp) {

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

temp = temp->next;

}

printf("NULL\n");

}

}

int main() {

int choice, value;

printf("\nImplementation of Queue using Linked

List\n");

while (choice != 4) {

printf("1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");

printf("Enter your choice : ");

scanf("%d", & choice);

switch (choice) {

case 1:

printf("Enter the value to insert: ");

scanf("%d", & value);

enqueue(value);

break;

case 2:

printf("Popped element is :%d\n",

dequeue());

break;

case 3:

display();

break;

case 4:

exit(0);

break;

default:

printf("\nInvalid choice\n");

}

}

return 0;

}

**Input & Output:**

Implementation of Queue using Linked List

1. Enqueue

2. Dequeue

3. Display

4. Exit.

Enter your choice : 1

Enter the value to insert : 10

1. Enqueue

2. Dequeue

3. Display

4. Exit.

Enter your choice : 1

Enter the value to insert : 20

1. Enqueue

2. Dequeue

3. Display

4. Exit.

Enter your choice : 1

Enter the value to insert : 30

1. Enqueue

2. Dequeue

3. Display

4. Exit.

Enter your choice : 1

Enter the value to insert : 40

1. Enqueue

2. Dequeue

3. Display

4. Exit.

Enter your choice : 2

Popped element is : 10

1. Enqueue

2. Dequeue

3. Display

4. Exit.

Enter your choice : 3

The queue is

20--->30--->40--->NULL

1. Enqueue

2. Dequeue

3. Display

4. Exit.

Enter your choice : 4

**Program - 13**

**Aim:** Write a Program to implement Quick Sort.

**Source Code:**

#include <stdio.h>

int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; j++) {

if (arr[j] <= pivot) {

i++;

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

int temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

void quickSort(int arr[], int low, int high) {

if (low < high) {

int pivotIndex = partition(arr, low, high);

quickSort(arr, low, pivotIndex - 1);

quickSort(arr, pivotIndex + 1, high);

}

}

void printArray(int arr[], int size) {

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

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

}

printf("\n");

}

int main() {

int arr[] = {64, 34, 25, 12, 22, 11, 90};

int arrSize = sizeof(arr) / sizeof(arr[0]);

printf("Original array: ");

printArray(arr, arrSize);

quickSort(arr, 0, arrSize - 1);

printf("Sorted array: ");

printArray(arr, arrSize);

return 0;

}

**Input & Output:**

Original array: 64 34 25 12 22 11 90

Sorted array: 11 12 22 25 34 64 90

**Program - 14**

**Aim:** Write a Program to implement Heap Sort.

**Source Code:**

#include <stdio.h>

int heapSize;

int arr[] = {53, 79, 21, 10, 99};

void heapSort(int arrLength);

void buildMaxHeap(int arrLength);

void maxHeapify(int i);

int main()

{

int arrLength = sizeof(arr) / sizeof(int);

heapSort(arrLength);

for (int i = 0; i < arrLength; i++)

{

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

}

printf("\n");

return 0;

}

void heapSort(int arrLength)

{

buildMaxHeap(arrLength);

for (int i = arrLength - 1; i >= 1; i--)

{

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

heapSize--;

maxHeapify(0);

}

}

void buildMaxHeap(int arrLength)

{

heapSize = arrLength;

for (int i = arrLength / 2 - 1; i >= 0; i--)

{

maxHeapify(i);

}

}

void maxHeapify(int i)

{

int largest;

int l = 2 \* i + 1;

int r = 2 \* i + 2;

if (l < heapSize && arr[l] > arr[i])

largest = l;

else

largest = i;

if (r < heapSize && arr[r] > arr[largest])

largest = r;

if (largest != i)

{

int temp = arr[i];

arr[i] = arr[largest];

arr[largest] = temp;

maxHeapify(largest);

}

}

**Input:**

10 45 98 37 56

**Output:**

10 37 45 56 98

**Program - 15**

**Aim:** Write a program to implement Hashing Technique

**Source Code:**

#include <stdio.h>

#include <math.h>

#define max 5

int arr[max];

void init()

{

int i;

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

{

arr[i] = -1;

}

}

void in\_div(int value)

{

int key = value % max;

if (arr[key] == -1)

{

arr[key] = value;

printf("%d inserted at arr[%d]\n", value, key);

}

else

{

printf("Collision: arr[%d] has element %d already\n", key, arr[key]);

printf("Unable to insert %d\n", value);

}

}

void in\_multi(int value)

{

float A = 0.40;

int key = (int)(max \* (remainder(A \* value, 1)));

if (arr[key] == -1)

{

arr[key] = value;

printf("%d inserted at arr[%d]\n", value, key);

}

else

{

printf("Collision: arr[%d] has element %d already\n", key, arr[key]);

printf("Unable to insert %d\n", value);

}

}

void in\_midsq(int value)

{

int keySqaure = value \* value;

int digits = (int)log10(value) + 1;

int key = (int)(keySqaure / pow(10, digits / 2)) % 10;

if (arr[key] == -1)

{

arr[key] = value;

printf("%d inserted at arr[%d]\n", value, key);

}

else

{

printf("Collision: arr[%d] has element %d already\n", key, arr[key]);

printf("Unable to insert %d\n", value);

}

}

void in\_digfold()

{

int arr2[15];

for (int i = 0; i < 15; i++)

{

arr2[i] = -1;

}

int value = 1102;

int key = 11 + 02;

arr2[key] = value;

for (int i = 0; i < 15; i++)

{

printf("%d", arr2[i]);

printf(" ");

}

}

int main()

{

printf("\nHashing using Division method:\n");

init();

in\_div(20);

in\_div(16);

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

{

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

printf(" ");

}

printf("\nHashing using Multiplication method:\n");

init();

in\_multi(123);

in\_multi(100);

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

{

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

printf(" ");

}

printf("\nHashing using Mid Square method:\n");

init();

in\_midsq(10);

in\_midsq(12);

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

{

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

printf(" ");

}

printf("\nHashing using Mid Square method:\n");

in\_digfold();

return 0;

}

**Input & Output:**

**Hashing using Division method:**

20 inserted at arr[0]

16 inserted at arr[1]

20 16 -1 -1 -1

**Hashing using Multiplication method:**

123 inserted at arr[1]

100 inserted at arr[0]

100 123 -1 -1 -1

**Hashing using Mid Square method:**

10 inserted at arr[0]

12 inserted at arr[4]

10 -1 -1 -1 12

**Hashing using Mid Square method:**

-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1102 -1

**Program - 16**

**Aim:** To implement circular queue using array

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

#define size 6

int queue[size];

int front = 0;

int rear = 0;

int isEmpty();

int isFull();

void enQueue(int);

int deQueue();

void display();

int main()

{

int choice;

do

{

printf("1. enQueue: \n");

printf("2. deQueue: \n");

printf("3. Display: \n");

printf("4. Exit: \n");

printf("Enter choice: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

{

int val;

printf("Enter item: ");

scanf("%d", &val);

enQueue(val);

break;

}

case 2:

{

if (!isEmpty())

printf("Removing item: %d\n", deQueue());

else

deQueue();

break;

}

case 3:

display();

break;

case 4:

break;

default:

printf("Invalid choice\n");

}

} while (choice != 4);

return 0;

}

int isEmpty()

{

if (rear == front)

return 1;

return 0;

}

int isFull()

{

if ((rear + 1) % size == front)

return 1;

return 0;

}

void enQueue(int val)

{

if (isFull())

printf("Queue is full\n");

else

{

queue[rear] = val;

// circular increment

rear = (rear + 1) % size;

printf("Item %d added.\n", val);

}

}

int deQueue()

{

int removed;

if (isEmpty())

printf("Queue is empty\n");

else

{

removed = queue[front];

// circular increment

front = (front + 1) % size;

}

return removed;

}

void display()

{

if (isEmpty())

printf("Queue is empty\n");

else

{

int i = front;

while (i != rear)

{

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

// circular increment

i = (i + 1) % size;

}

printf("\n");

}

}

**Input & Outputs:**

1. enQueue:

2. deQueue:

3. Display:

4. Exit:

Enter choice: 1

Enter item: 10

Item 10 added.

1. enQueue:

2. deQueue:

3. Display:

4. Exit:

Enter choice: 1

Enter item: 20

Item 20 added.

1. enQueue:

2. deQueue:

3. Display:

4. Exit:

Enter choice: 1

Enter item: 30

Item 30 added.

1. enQueue:

2. deQueue:

3. Display:

4. Exit:

Enter choice: 3

10 20 30

1. enQueue:

2. deQueue:

3. Display:

4. Exit:

Enter choice: 2

Removing item: 10

1. enQueue:

2. deQueue:

3. Display:

4. Exit:

Enter choice: 3

20 30

1. enQueue:

2. deQueue:

3. Display:

4. Exit:

Enter choice: 4

**Program - 17**

**Aim:** To implement circular queue using linked list.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*front = NULL;

struct node \*rear = NULL;

void EnQueue();

void DeQueue();

void Display();

int main()

{

int choice;

do

{

printf("1.EnQueue\n2.DeQueue\n3.Display\n4.Exit\nEnter choice: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

EnQueue();

break;

case 2:

DeQueue();

break;

case 3:

Display();

break;

case 4:

break;

default:

printf("Invalid choice\n");

}

} while (choice != 4);

return 0;

}

void EnQueue()

{

int val;

printf("Enter item: ");

scanf("%d", &val);

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

newNode->data = val;

newNode->next = NULL;

// if first node

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

{

front = rear = newNode;

rear->next = front;

}

else

{

rear->next = newNode;

rear = newNode;

rear->next = front;

}

}

void DeQueue()

{

struct node \*temp;

if (front == NULL)

printf("Queue is empty.\n");

else

{

temp = front;

front = front->next;

rear->next = front;

if (front == NULL)

rear = NULL;

printf("Removing element: %d\n", temp->data);

}

free(temp);

}

void Display()

{

struct node \*temp = front;

while (temp->next != front)

{

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

temp = temp->next;

}

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

}

**Input & Outputs:**

1.EnQueue

2.DeQueue

3.Display

4.Exit

Enter choice: 1

Enter item: 10

1.EnQueue

2.DeQueue

3.Display

4.Exit

Enter choice: 1

Enter item: 20

1.EnQueue

2.DeQueue

3.Display

4.Exit

Enter choice: 1

Enter item: 30

1.EnQueue

2.DeQueue

3.Display

4.Exit

Enter choice: 3

10->20->30

1.EnQueue

2.DeQueue

3.Display

4.Exit

Enter choice: 2

Removing element: 10

1.EnQueue

2.DeQueue

3.Display

4.Exit

Enter choice: 3

20->30

1.EnQueue

2.DeQueue

3.Display

4.Exit

Enter choice: 4

**Program - 18**

**Aim:** Write a Program to implement binary tree using linked list.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct btnode

{

int data;

struct btnode \*left;

struct btnode \*right;

};

struct btnode \*create();

void inorder(struct btnode \*node);

void preorder(struct btnode \*node);

void postorder(struct btnode \*node);

int main()

{

struct btnode \*root;

int choice;

do

{

printf("\n1.CreateTree\n2.Inorder\n3.Preorder\n4.Postorder\n5.Exit\nEnter choice:\n");

scanf("%d", &choice);

switch (choice)

{

case 1:

root = create();

break;

case 2:

inorder(root);

break;

case 3:

preorder(root);

break;

case 4:

postorder(root);

break;

case 5:

break;

}

} while (choice != 5);

return 0;

}

struct btnode \*create()

{

struct btnode \*newNode;

int data;

printf("\nEnter data(-1 for null value): ");

scanf("%d", &data);

if (data == -1)

{

return 0;

}

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

newNode->data = data;

printf("Enter left child of %d: ", data);

newNode->left = create();

printf("Enter right child of %d: ", data);

newNode->right = create();

return newNode;

}

void inorder(struct btnode \*node)

{

if (node == NULL)

{

return;

}

inorder(node->left);

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

inorder(node->right);

}

void preorder(struct btnode \*node)

{

if (node == NULL)

{

return;

}

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

preorder(node->left);

preorder(node->right);

}

void postorder(struct btnode \*node)

{

if (node == NULL)

{

return;

}

postorder(node->left);

postorder(node->right);

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

}

**Input & Output:**

1.CreateTree

2.Inorder

3.Preorder

4.Postorder

5.Exit

Enter choice:

1

Enter data(-1 for null value): 6

Enter left child of 6:

Enter data(-1 for null value): 4

Enter left child of 4:

Enter data(-1 for null value): 3

Enter left child of 3:

Enter data(-1 for null value): -1

Enter right child of 3:

Enter data(-1 for null value): -1

Enter right child of 4:

Enter data(-1 for null value): 5

Enter left child of 5:

Enter data(-1 for null value): -1

Enter right child of 5:

Enter data(-1 for null value): -1

Enter right child of 6:

Enter data(-1 for null value): 8

Enter left child of 8:

Enter data(-1 for null value): 7

Enter left child of 7:

Enter data(-1 for null value): -1

Enter right child of 7:

Enter data(-1 for null value): -1

Enter right child of 8:

Enter data(-1 for null value): 9

Enter left child of 9:

Enter data(-1 for null value): -1

Enter right child of 9:

Enter data(-1 for null value): -1

1.CreateTree

2.Inorder

3.Preorder

4.Postorder

5.Exit

Enter choice:

2

3 4 5 6 7 8 9

1.CreateTree

2.Inorder

3.Preorder

4.Postorder

5.Exit

Enter choice: 3

6 4 3 5 8 7 9

1.CreateTree

2.Inorder

3.Preorder

4.Postorder

5.Exit

Enter choice: 4

3 5 4 7 9 8 6

1.CreateTree

2.Inorder

3.Preorder

4.Postorder

5.Exit

Enter choice: 5

**Program - 19**

**Aim:** Write a program to implement binary search tree using linked list.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*left;

struct node \*right;

};

struct node \*getNewNode(int val);

struct node \*search(struct node \*root, int val);

int getRightMin(struct node \*root);

struct node \*removeNode(struct node \*root, int val);

struct node \*insert(struct node \*root, int val);

void inorder(struct node \*root);

int main()

{

struct node \*root;

root = NULL;

int choice;

int val;

do

{

printf("\n1.Insert\n2.Delete\n3.Search\n4.Inorder\n5.Exit\nEnter choice:\n");

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("Enter element: ");

scanf("%d", &val);

root = insert(root, val);

printf("Element added\n");

break;

case 2:

printf("Enter element: ");

scanf("%d", &val);

root = removeNode(root, val);

printf("Item %d deleted.\n", val);

break;

case 3:

printf("Enter element: ");

scanf("%d", &val);

root = search(root, val);

printf("%p", root);

break;

case 4:

inorder(root);

printf("\n");

break;

case 5:

break;

}

} while (choice != 5);

return 0;

}

struct node \*getNewNode(int val)

{

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

newNode->data = val;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

struct node \*search(struct node \*root, int val)

{

if (root == NULL || root->data == val)

{

return root;

}

else if (root->data < val)

{

return search(root->right, val);

}

else

{

return search(root->left, val);

}

}

int getRightMin(struct node \*root)

{

struct node \*temp = root;

while (temp->left != NULL)

{

temp = temp->left;

}

return temp->data;

}

struct node \*removeNode(struct node \*root, int val)

{

if (root == NULL)

return NULL;

if (root->data < val)

{

root->right = removeNode(root->right, val);

}

else if (root->data > val)

{

root->left = removeNode(root->left, val);

}

else

{

if (root->left == NULL && root->right == NULL)

{

free(root);

return NULL;

}

else if (root->left == NULL)

{

struct node \*temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL)

{

struct node \*temp = root->left;

free(root);

return temp;

}

else

{

int rightMin = getRightMin(root->right);

root->data = rightMin;

root->right = removeNode(root->right, rightMin);

}

}

return root;

}

struct node \*insert(struct node \*root, int val)

{

if (root == NULL)

return getNewNode(val);

else if (root->data < val)

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

else if (root->data > val)

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

return root;

}

void inorder(struct node \*root)

{

if (root == NULL)

{

return;

}

inorder(root->left);

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

inorder(root->right);

}

**Input & Output:**

1.Insert

2.Delete

3.Search

4.Inorder

5.Exit

Enter choice:

1

Enter element: 100

Element added

1.Insert

2.Delete

3.Search

4.Inorder

5.Exit

Enter choice:

1

Enter element: 50

Element added

1.Insert

2.Delete

3.Search

4.Inorder

5.Exit

Enter choice:

1

Enter element: 200

Element added

1.Insert

2.Delete

3.Search

4.Inorder

5.Exit

Enter choice:

1

Enter element: 150

Element added

1.Insert

2.Delete

3.Search

4.Inorder

5.Exit

Enter choice:

1

Enter element: 300

Element added

1.Insert

2.Delete

3.Search

4.Inorder

5.Exit

Enter choice:

4

50 100 150 200 300

1.Insert

2.Delete

3.Search

4.Inorder

5.Exit

Enter choice:

2

Enter element: 200

Item 200 deleted.

1.Insert

2.Delete

3.Search

4.Inorder

5.Exit

Enter choice:

4

50 100 150 300

1.Insert

2.Delete

3.Search

4.Inorder

5.Exit

Enter choice:

5

**Program - 20**

**Aim:** Write a Program to solve Tower of Hanoi Problem.

**Source Code:**

void towerOfHanoi(int n, char A, char B, char C){

if(n == 1){

printf("Move %c --> %c \n",A,C);

return;

}

towerOfHanoi(n-1, A, C, B);

printf("Move %c --> %c \n", A, C);

towerOfHanoi(n-1, B,A,C);

}

int main(){

int n;

printf("Enter number of disks: ");

scanf("%d",&n)

towerOfHanoi(n, 'S', 'H', 'D');

}

**Input**

3

**Output:**

Move S --> D

Move S --> H

Move D --> H

Move S --> D

Move H --> S

Move H --> D

Move S --> D

**Program - 21**

**Aim:**  Write a Program to implement Radix Sort.

**Source Code:**

#include <stdio.h>

int getMax(int arr[], int n)

{

int mx = arr[0];

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

if (arr[i] > mx)

mx = arr[i];

return mx;

}

void countSort(int arr[], int n, int exp)

{

int output[n];

int i, count[10] = {0};

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

count[(arr[i] / exp) % 10]++;

for (i = 1; i < 10; i++)

count[i] += count[i - 1];

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

{

output[count[(arr[i] / exp) % 10] - 1] = arr[i];

count[(arr[i] / exp) % 10]--;

}

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

arr[i] = output[i];

}

void radixsort(int arr[], int n)

{

// Find the maximum number to know number of digits

int m = getMax(arr, n);

for (int exp = 1; m / exp > 0; exp \*= 10)

countSort(arr, n, exp);

}

void print(int arr[], int n)

{

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

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

}

int main()

{

int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};

int n = sizeof(arr) / sizeof(arr[0]);

// Radix sort function

radixsort(arr, n);

print(arr, n);

return 0;

}

**Output**

2 24 45 66 75 90 170 802

**Program - 22**

**Aim** : Write a Program to implement tree traversals using linked list.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct Node

{

int data;

struct Node \*left;

struct Node \*right;

};

struct Node \*createNode(int data)

{

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

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

void inorderTraversal(struct Node \*root)

{

if (root == NULL)

return;

inorderTraversal(root->left);

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

inorderTraversal(root->right);

}

void preorderTraversal(struct Node \*root)

{

if (root == NULL)

return;

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

preorderTraversal(root->left);

preorderTraversal(root->right);

}

void postorderTraversal(struct Node \*root)

{

if (root == NULL)

return;

postorderTraversal(root->left);

postorderTraversal(root->right);

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

}

int main()

{

struct Node \*root = createNode(1);

root->left = createNode(2);

root->right = createNode(3);

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

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

printf("Inorder Traversal: ");

inorderTraversal(root);

printf("\n");

printf("Preorder Traversal: ");

preorderTraversal(root);

printf("\n");

printf("Postorder Traversal: ");

postorderTraversal(root);

printf("\n");

return 0;

}

**Output:**

Inorder Traversal: 4 2 5 1 3

Preorder Traversal: 1 2 4 5 3

Postorder Traversal: 4 5 2 3 1

**Program - 23**

**Aim** : Write a Program to implement MST (Minimum Spanning Tree).

**Source Code:**

#include <stdio.h>

#include <stdbool.h>

#include <limits.h>

#define V 5 // Number of vertices in the graph

int minKey(int key[], bool mstSet[])

{

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

{

if (mstSet[v] == false && key[v] < min)

{

min = key[v];

min\_index = v;

}

}

return min\_index;

}

void printMST(int parent[], int graph[V][V])

{

printf("Edge \tWeight\n");

for (int i = 1; i < V; i++)

printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);

}

void primMST(int graph[V][V])

{

int parent[V]; // Array to store constructed MST

int key[V]; // Key values used to pick minimum weight edge in cut

bool mstSet[V]; // To represent set of vertices included in MST

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

{

key[i] = INT\_MAX;

mstSet[i] = false;

}

key[0] = 0; // Make key 0 so that this vertex is picked as the first vertex

parent[0] = -1; // First node is always root of MST

for (int count = 0; count < V - 1; count++)

{

int u = minKey(key, mstSet);

mstSet[u] = true;

for (int v = 0; v < V; v++)

{

if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])

{

parent[v] = u;

key[v] = graph[u][v];

}

}

}

printMST(parent, graph);

}

int main()

{

int graph[V][V] = {

{0, 2, 0, 6, 0},

{2, 0, 3, 8, 5},

{0, 3, 0, 0, 7},

{6, 8, 0, 0, 9},

{0, 5, 7, 9, 0}};

primMST(graph);

return 0;

}

**Output:**

Edge Weight

0 – 1 2

1 – 2 3

0 – 3 6

1 – 4 5

**Program - 24**

**Aim** : Write a Program to evaluate Prefix and Postfix expression.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_SIZE 100

struct Stack

{

int top;

unsigned capacity;

int \*array;

};

struct Stack \*createStack(unsigned capacity)

{

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

stack->capacity = capacity;

stack->top = -1;

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

return stack;

}

int isFull(struct Stack \*stack)

{

return stack->top == stack->capacity - 1;

}

int isEmpty(struct Stack \*stack)

{

return stack->top == -1;

}

void push(struct Stack \*stack, int item)

{

if (isFull(stack))

return;

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

}

int pop(struct Stack \*stack)

{

if (isEmpty(stack))

return INT\_MIN;

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

}

int evaluatePostfix(char \*postfix)

{

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

int i, operand1, operand2;

for (i = 0; postfix[i]; ++i)

{

if (isdigit(postfix[i]))

{

push(stack, postfix[i] - '0');

}

else

{

operand2 = pop(stack);

operand1 = pop(stack);

switch (postfix[i])

{

case '+':

push(stack, operand1 + operand2);

break;

case '-':

push(stack, operand1 - operand2);

break;

case '\*':

push(stack, operand1 \* operand2);

break;

case '/':

push(stack, operand1 / operand2);

break;

}

}

}

return pop(stack);

}

int evaluatePrefix(char \*prefix)

{

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

int i, operand1, operand2;

for (i = strlen(prefix) - 1; i >= 0; --i)

{

if (isdigit(prefix[i]))

{

push(stack, prefix[i] - '0');

}

else

{

operand1 = pop(stack);

operand2 = pop(stack);

switch (prefix[i])

{

case '+':

push(stack, operand1 + operand2);

break;

case '-':

push(stack, operand1 - operand2);

break;

case '\*':

push(stack, operand1 \* operand2);

break;

case '/':

push(stack, operand1 / operand2);

break;

}

}

}

return pop(stack);

}

int main()

{

char postfix[MAX\_SIZE];

char prefix[MAX\_SIZE];

printf("Enter the postfix expression: ");

fgets(postfix, MAX\_SIZE, stdin);

postfix[strcspn(postfix, "\n")] = '\0'; // Remove newline character

int resultPostfix = evaluatePostfix(postfix);

printf("Result of postfix expression: %d\n", resultPostfix);

printf("\nEnter the prefix expression: ");

fgets(prefix, MAX\_SIZE, stdin);

prefix[strcspn(prefix, "\n")] = '\0'; // Remove newline character

int resultPrefix = evaluatePrefix(prefix);

printf("Result of prefix expression: %d\n", resultPrefix);

return 0;

}

**Output:**

Enter the postfix expression: 53\*82/+

Result of postfix expression: 17

Enter the prefix expression: +\*53/82

Result of prefix expression: 17