**DS Practicals**

1. **Write a C++ program to store the elements in 1-D array and perform the operations like searching, sorting and reversing the elements.**

#include<iostream>

#include<conio.h>

using namespace std;

void search(int A[], int n, int val)

{

int flag=0;

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

{

if(A[i]==val)

{

cout<<val<<"Present at location:"<<i<<endl;

flag=1;

}

}

if(flag==0)

cout<<val<<"Not present in array";

}

int\* sort(int A[], int n)

{

int \*B=new int[n];

for(int i=0; i<n; i++) B[i]=A[i];

int tmp;

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

{

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

{

if(B[i]>B[j])

{

tmp=B[i];

B[i]=B[j];

B[j]=tmp;

}

}

}

return B;

}

int\* reverse(int A[], int n)

{

int \*B=new int[n];

for(int i=0; i<n; i++) B[i]=A[i];

int tmp;

for(int i=0, j=n-1; i<n/2; i++, j--)

{

tmp=B[i];

B[i]=B[j];

B[j]=tmp;

}

return B;

}

void display(int A[], int n)

{

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

cout<<A[i]<<" ";

}

int main()

{

int n;

system("cls");

cout<<"Enter size of array:";

cin>>n;

int \*A=new int[n];

cout<<"Enter"<<n<<"numbers...\n";

for(int i=0; i<n; i++) cin>>A[i];

int ch;

int \*X;

do

{

system("cls");

cout<<"Original Array:";

display(A,n);

cout<<"\nMenu(1.Search, 2.Sort, 3.Reverse, 4.Exit):";

cout<<"\nEnter choice:";

cin>>ch;

switch(ch)

{

case 1:

int val;

cout<<"Enter no. to search:";

cin>>val;

search(A, n, val);

break;

case 2:

X=sort(A,n);

cout<<"\nSorted Array:";

display(X,n);

delete []X;

break;

case 3:

X=reverse(A,n);

cout<<"\nReverse Array:";

display(X,n);

delete []X;

break;

case 4:

cout<<"\nStop processing...";

break;

default:

cout<<"\nInvalid Entry!!!";

}

getch();

}

while(ch!=4);

delete []A;

return 0;

}

1. **Write a C++ program to merge two sorted arrays in a single sorted array.**

#include<iostream>

#include<conio.h>

using namespace std;

int\* merge(int A[], int lb1, int ub1, int B[], int lb2, int ub2)

{

int \*C=new int[ub1+ub2+1];

int i=lb1, j=lb2, k=0;

while(i<=ub1 and j<=ub2)

{

if(A[i]<B[j])

{

C[k]=A[i]; i++; k++;

}

else

{

C[k]=B[j]; j++; k++;

}

}

if(i>ub1)

{

while(j<ub2)

{

C[k]=B[j]; j++; k++;

}

}

else if(j>ub2)

{

while(i<=ub1)

{

C[k]=A[i]; i++; k++;

}

}

return C;

}

void display(int A[], int lb, int ub)

{

for(int i=lb; i<=ub; i++)

cout<<A[i]<<" ";

}

int main()

{

int X[]={10,20,30,40,50,100};

int Y[]={0,5,15,25,35,45,55,65,75,85,95,100};

int ub1=5, ub2=11;

cout<<"\nFirst Array:";

display(X,0,ub1);

cout<<"\nSecond Array:";

display(Y,0,ub2);

int \*Z=merge(X,0,ub1,Y,0,ub2);

cout<<"\nMerged Array:";

display(Z,0,ub1+ub2);

delete []X;

delete []Y;

delete []Z;

getch();

}

1. **Write a C++ program to search an element in the single linked list.**

#include<iostream>

#include<stdlib.h>

using namespace std;

template <typename type>

class LinkedList

{

struct Node

{

type data;

Node\* next;

};

Node\* begin=NULL;

public:

void traverse()

{

if(begin==NULL)

cout<<"Empty..."<<endl;

else

{

Node \*ptr=begin;

cout<<"\nList:";

while(ptr!=NULL)

{

cout<<"\t"<<ptr->data;

ptr=ptr->next;

}

}

}

void insertBeg(type x)

{

Node \*tmp=new Node;

tmp->data=x;

tmp->next=begin;

begin=tmp;

}

void insertEnd(type x)

{

Node \*tmp=new Node;

tmp->data=x;

tmp->next=NULL;

if(begin==NULL) begin=tmp;

else

{

Node \*ptr=begin;

while(ptr->next!=NULL)

ptr=ptr->next;

ptr->next=tmp;

}

}

void search(type val)

{

if(begin==NULL)

cout<<"Empty..."<<endl;

else

{

int flag=0;

Node \*ptr=begin;

while(ptr!=NULL)

{

if(ptr->data==val)

{

cout<<val<<"Present at address:"<<ptr<<endl;

flag=1;

}

ptr=ptr->next;

}

if(flag==0)

cout<<val<<"Not present";

}

}

};

int main()

{

system("cls");

LinkedList<int> obj;

int val, ch;

while(true)

{

cout<<"\n\tLinked List(1.insertBeg, 2.insertEnd, 3.Search, 4.Exit):";

cin>>ch;

switch(ch)

{

case 1:

cout<<"\n\tEnter integer:";

cin>>val;

obj.insertBeg(val);

break;

case 2:

cout<<"\n\tEnter integer:";

cin>>val;

obj.insertEnd(val);

break;

case 3:

cout<<"\n\tEnter integer:";

cin>>val;

obj.search(val);

break;

case 4:

exit(0);

}

obj.traverse();

}

return 0;

}

1. **Write a C++ program to sort doubly linked list or two way linked list.**

#include<iostream>

#include<stdlib.h>

using namespace std;

template <typename type>

class DLinkedList

{

struct Node

{

type data;

Node \*next;

Node \*prev;

};

Node\* begin=NULL, \*end=NULL;

public:

void forward\_traverse()

{

if(begin==NULL)

cout<<"Empty..."<<endl;

else

{

Node \*ptr=begin;

cout<<"\nForward List:";

while(ptr!=NULL)

{

cout<<"\t"<<ptr->data;

ptr=ptr->next;

}

}

}

void backward\_traverse()

{

if(end==NULL)

cout<<"Empty..."<<endl;

else

{

Node \*ptr=end;

cout<<"\nBackward List:";

while(ptr!=NULL)

{

cout<<"\t"<<ptr->data;

ptr=ptr->prev;

}

}

}

void insertBeg(type x)

{

Node \*tmp=new Node;

tmp->prev=NULL;

tmp->data=x;

if(begin==NULL)

{

tmp->next=NULL;

end=tmp;

}

else

{

tmp->next=begin;

begin->prev=tmp;

}

begin=tmp;

}

void insertEnd(type x)

{

Node \*tmp=new Node;

tmp->data=x;

tmp->next=NULL;

if(end==NULL)

{

tmp->prev=NULL;

begin=end=tmp;

}

else

{

tmp->prev=end;

end->next=tmp;

}

end=tmp;

}

void sortlist()

{

Node \*i=begin, \*j=NULL;

type tmp;

while(i!=NULL)

{

j=i->next;

while(j!=NULL)

{

if(i->data>j->data)

{

tmp=i->data;

i->data=j->data;

j->data=tmp;

}

j=j->next;

}

i=i->next;

}

}

};

int main()

{

system("cls");

DLinkedList<int> obj;

int val, ch;

while(true)

{

cout<<"\n\tDLinked List(1.insertBeg, 2.insertEnd, 3.Sort, 4.Exit):";

cin>>ch;

switch(ch)

{

case 1:

cout<<"\n\tEnter integer:";

cin>>val;

obj.insertBeg(val);

break;

case 2:

cout<<"\n\tEnter integer:";

cin>>val;

obj.insertEnd(val);

break;

case 3:

obj.sortlist();

break;

case 4:

exit(0);

}

obj.forward\_traverse();

obj.backward\_traverse();

}

return 0;

}

1. **Write a C++ program array implementation of stack data structure with push, pop, peek and display operations.**

#include<iostream>

#include<stdlib.h>

using namespace std;

template<class type>

class stack

{

private:

int Max, Top;

type \*List;

public:

stack(int stacksize)

{

if(stacksize<=0)

{

cout<<"Stack size cannot be negative. Default size is 10";

stacksize=10;

}

Max=stacksize;

Top=-1;

List=new type(Max);

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

List[i]=NULL;

}

~stack()

{

delete []List;

}

bool isEmpty()

{

return(Top==-1);

}

bool isFull()

{

return(Top==Max-1);

}

void push(type newitem)

{

if(isFull())

cout<<"Stack Overflow"<<endl;

else

{

List[++Top]=newitem;

}

}

void pop()

{

if(isEmpty())

cout<<"Stack Underflow"<<endl;

else

Top--;

}

type peek()

{

if(isEmpty())

{

cout<<"Stack Underflow"<<endl;

return NULL;

}

else return List[Top];

}

void display()

{

cout<<"\nTop:"<<Top;

cout<<"\nStack:";

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

cout<<List[i]<<"\t";

}

};

int main()

{

stack<int> obj(5);

int ch, val;

while(true)

{

cout<<"\n\tStack Operations(1.Push, 2.Pop, 3.Peek, 4.Exit):";

cin>>ch;

switch(ch)

{

case 1:

cout<<"\n\tEnter integer to push:";

cin>>val;

obj.push(val);

break;

case 2:

obj.pop();

break;

case 3:

val=obj.peek();

if(val!=NULL)

cout<<"\n\tTop element of the stack:"<<val;

break;

case 4:

exit(0);

}

obj.display();

}

return 0;

}

1. **Write a c++ program to create a single linked list, reverse list and display list.**

#include<iostream>

#include<stdlib.h>

using namespace std;

template<typename type>

class LinkedList

{

struct Node

{

type data;

Node\* next;

};

Node\* begin=NULL;

public:

void traverse()

{

if(begin==NULL)

cout<<"Empty..."<<endl;

else

{

Node \*ptr=begin;

cout<<"\nList:";

while(ptr!=NULL)

{

cout<<"\t"<<ptr->data;

ptr=ptr->next;

}

}

}

void insertBeg(type x)

{

Node \*tmp=new Node;

tmp->data=x;

tmp->next=begin;

begin=tmp;

}

void insertEnd(type x)

{

Node \*tmp=new Node;

tmp->data=x;

tmp->next=NULL;

if(begin==NULL) begin=tmp;

else

{

Node \*ptr=begin;

while(ptr->next!=NULL)

ptr=ptr->next;

ptr->next=tmp;

}

}

void reverse()

{

Node \*current=begin;

Node \*prev=NULL, \*next=NULL;

while(current!=NULL)

{

next=current->next;

current->next=prev;

prev=current;

current=next;

}

begin=prev;

}

};

int main()

{

system("cls");

LinkedList<int> obj;

int val,ch;

while(true)

{

cout<<"\n\tLinked List(1.insertBeg, 2.insertEnd, 3.reverse, 4.exit):";

cin>>ch;

switch(ch)

{

case 1:

cout<<"\n\tEnter integer:";

cin>>val;

obj.insertBeg(val);

break;

case 2:

cout<<"\n\tEnter integer:";

cin>>val;

obj.insertEnd(val);

break;

case 3:

obj.reverse();

break;

case 4:

exit(0);

}

obj.traverse();

}

return 0;

}

1. **Design an employee class for reading and displaying the employee information (employee ID and employee name) with the getInfo() and displayInfo() methods:**

#include<iostream>

using namespace std;

class Employee

{

int empid;

char empname[20];

void getInfo()

{

cout<<"ID";

cin>>empid;

cout<<"Name:";

cin>>empname;

}

public:

Employee()

{

getInfo();

}

void displayInfo()

{

cout<<"\n"<<empid<<"\t"<<empname;

}

};

int main()

{

cout<<"Enter employee details of First Employee...\n";

Employee e1;

cout<<"Enter employee details of Second Employee..\n";

Employee e2;

cout<<"\n id \t Name";

e1.displayInfo();

e2.displayInfo();

return 0;

}

1. **Write a C++ to convert an infix expression to postfix expression using stack data structure:**

#include<bits/stdc++.h>

using namespace std;

int precedence(char m)

{

if(m == '^')

return 3;

else if(m == '\*' || m == '/')

return 2;

else if(m == '+' || m == '-')

return 1;

}

void infix\_to\_postfix(string t)

{

stack<char> s;

int l = t.length();

string ans;

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

{

if((t[i] >= 'a' && t[i] <= 'z') || (t[i] >= 'A' && t[i] <= 'Z'))

ans+=t[i];

else if(t[i] == '(')

s.push('(');

else if(t[i] == ')')

{

while(s.top() != '(')

{

char c = s.top();

ans += c;

s.pop();

}

if(s.top() == '(')

{

char c = s.top();

s.pop();

}

}

else{

while(s.empty()==false && precedence(t[i]) <= precedence(s.top()))

{

char c = s.top();

ans += c;

s.pop();

}

s.push(t[i]);

}

}

while(s.empty() == false)

{

char c = s.top();

ans += c;

s.pop();

}

cout << ans << endl;

}

int main()

{

string s = "a+b\*c";

infix\_to\_postfix(s);

return 0;

}

1. **Write a C++ program for the array implementation of queue data structure with insert enqueue , delete dequeue, getFront & traverse display operations:**

#include <iostream>

using namespace std;

int cqueue[5];

int front = -1, rear = -1, n=5;

void insertCQ(int val) {

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

cout<<"Queue Overflow \n";

return;

}

if (front == -1) {

front = 0;

rear = 0;

} else {

if (rear == n - 1)

rear = 0;

else

rear = rear + 1;

}

cqueue[rear] = val ;

}

void deleteCQ() {

if (front == -1) {

cout<<"Queue Underflow\n";

return ;

}

cout<<"Element deleted from queue is : "<<cqueue[front]<<endl;

if (front == rear) {

front = -1;

rear = -1;

} else {

if (front == n - 1)

front = 0;

else

front = front + 1;

}

}

void displayCQ() {

int f = front, r = rear;

if (front == -1) {

cout<<"Queue is empty"<<endl;

return;

}

cout<<"Queue elements are :\n";

if (f <= r) {

while (f <= r){

cout<<cqueue[f]<<" ";

f++;

}

} else {

while (f <= n - 1) {

cout<<cqueue[f]<<" ";

f++;

}

f = 0;

while (f <= r) {

cout<<cqueue[f]<<" ";

f++;

}

}

cout<<endl;

}

int main() {

int ch, val;

cout<<"1)Insert\n";

cout<<"2)Delete\n";

cout<<"3)Display\n";

cout<<"4)Exit\n";

do {

cout<<"Enter choice : "<<endl;

cin>>ch;

switch(ch) {

case 1:

cout<<"Input for insertion: "<<endl;

cin>>val;

insertCQ(val);

break;

case 2:

deleteCQ();

break;

case 3:

displayCQ();

break;

case 4:

cout<<"Exit\n";

break;

default: cout<<"Incorrect!\n";

}

} while(ch != 4);

return 0;

}

1. **Write a C++ program to implement bubble sort:**

#include <iostream>

using namespace std;

void bubblesort(int arr[], int n)

{

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

{

return;

}

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

{

if (arr[i] > arr[i + 1])

{

swap(arr[i], arr[i + 1]);

}

}

bubblesort(arr, n - 1);

}

int main()

{

int arr[5] = {2, 5, 1, 6, 9};

bubblesort(arr, 5);

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

{

cout << arr[i] << " ";

}

return 0;

}

1. **Write a C++ program to implement selection sort:**

#include <bits/stdc++.h>

using namespace std;

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

void selectionSort(int arr[], int n)

{

int i, j, min\_idx;

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

{

min\_idx = i;

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

if (arr[j] < arr[min\_idx])

min\_idx = j;

if(min\_idx!=i)

swap(&arr[min\_idx], &arr[i]);

}

}

void printArray(int arr[], int size)

{

int i;

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

cout << arr[i] << " ";

cout << endl;

}

int main()

{

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

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

selectionSort(arr, n);

cout << "Sorted array: \n";

printArray(arr, n);

return 0;

}

1. **Write a C++ program to implement merge sort:**

#include <iostream>

using namespace std;

void merge(int arr[], int p, int q, int r)

{

int n1 = q - p + 1;

int n2 = r - q;

int L[n1], M[n2];

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

L[i] = arr[p + i];

for (int j = 0; j < n2; j++)

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

int i, j, k;

i = 0;

j = 0;

k = p;

while (i < n1 && j < n2)

{

if (L[i] <= M[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = M[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = M[j];

j++;

k++;

}

}

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

void printArray(int arr[], int size)

{

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

cout << arr[i] << " ";

cout << endl;

}

int main()

{

int arr[] = {6, 5, 12, 10, 9, 1};

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

mergeSort(arr, 0, size - 1);

cout << "Sorted array: \n";

printArray(arr, size);

return 0;

}

1. **Write a C++ program to search an element using sequential search:**

#include <iostream>

using namespace std;

int main()

{

int arr[5], n, i, pos;

i=0;

while(i<=4)

{

cout<<" enter value in element "<<i<<" :";

cin>>arr[i];

i++;

}

pos = 0;

cout<<" enter any value :";

cin>>n;

i=0;

while(i<=5)

{

if(n==arr[i])

{

pos=i+1;

break;

}

i++;

}

if(pos==0)

cout<<" value not found"<<endl;

else

cout<<" Value found at position = "<<pos<<endl;

}

1. **Write a C++ program to search an element using binary search:**

#include<iostream>

using namespace std;

int main()

{

int i, arr[10], num, first, last, middle;

cout<<"Enter 10 Elements (in ascending order): ";

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

cin>>arr[i];

cout<<"\nEnter Element to be Search: ";

cin>>num;

first = 0;

last = 9;

middle = (first+last)/2;

while(first <= last)

{

if(arr[middle]<num)

first = middle+1;

else if(arr[middle]==num)

{

cout<<"\nThe number, "<<num<<" found at Position "<<middle+1;

break;

}

else

last = middle-1;

middle = (first+last)/2;

}

if(first>last)

cout<<"\nThe number, "<<num<<" is not found in given Array";

cout<<endl;

return 0;

}

1. **Write a C++ program binary tree traversal preorder, inorder and postorder:**

#include <iostream>

using namespace std;

struct Node {

int data;

struct Node \*left, \*right;

Node(int data) {

this->data = data;

left = right = NULL;

}

};

void preorderTraversal(struct Node\* node)

{

if (node == NULL)

return;

cout << node->data << "->";

preorderTraversal(node->left);

preorderTraversal(node->right);

}

void postorderTraversal(struct Node\* node)

{

if (node == NULL)

return;

postorderTraversal(node->left);

postorderTraversal(node->right);

cout << node->data << "->";

}

void inorderTraversal(struct Node\* node)

{

if (node == NULL)

return;

inorderTraversal(node->left);

cout << node->data << "->";

inorderTraversal(node->right);

}

int main()

{

struct Node\* root = new Node(1);

root->left = new Node(12);

root->right = new Node(9);

root->left->left = new Node(5);

root->left->right = new Node(6);

cout << "Inorder traversal ";

inorderTraversal(root);

cout << "\nPreorder traversal ";

preorderTraversal(root);

cout << "\nPostorder traversal ";

postorderTraversal(root);

}

1. **Write a c++ program to insert the element into maximum heap:**

#include <iostream>

using namespace std;

void max\_heap(int \*a, int m, int n) {

int j, t;

t = a[m];

j = 2 \* m;

while (j <= n) {

if (j < n && a[j+1] > a[j])

j = j + 1;

if (t > a[j])

break;

else if (t <= a[j]) {

a[j / 2] = a[j];

j = 2 \* j;

}

}

a[j/2] = t;

return;

}

void build\_maxheap(int \*a,int n) {

int k;

for(k = n/2; k >= 1; k--) {

max\_heap(a,k,n);

}

}

int main() {

int n, i;

cout<<"enter no of elements of array\n";

cin>>n;

int a[30];

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

cout<<"enter elements"<<" "<<(i)<<endl;

cin>>a[i];

}

build\_maxheap(a,n);

cout<<"Max Heap\n";

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

cout<<a[i]<<endl;

}

}

1. **Write a c++ program to insert the element into minimum heap:**

#include <iostream>

#include <conio.h>

using namespace std;

void min\_heap(int \*a, int m, int n){

int j, t;

t= a[m];

j = 2 \* m;

while (j <= n) {

if (j < n && a[j+1] < a[j])

j = j + 1;

if (t < a[j])

break;

else if (t >= a[j]) {

a[j/2] = a[j];

j = 2 \* j;

}

}

a[j/2] = t;

return;

}

void build\_minheap(int \*a, int n) {

int k;

for(k = n/2; k >= 1; k--) {

min\_heap(a,k,n);

}

}

int main() {

int n, i;

cout<<"enter no of elements of array\n";

cin>>n;

int a[30];

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

cout<<"enter element"<<" "<<(i)<<endl;

cin>>a[i];

}

build\_minheap(a, n);

cout<<"Min Heap\n";

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

cout<<a[i]<<endl;

}

getch();

}

1. **Write a C++ program to implement the concept of collision handling:**

#include <iostream>

const int T\_S = 200;

using namespace std;

struct HashTableEntry {

int v, k;

HashTableEntry \*n;

HashTableEntry \*p;

HashTableEntry(int k, int v) {

this->k = k;

this->v = v;

this->n = NULL;

}

};

class HashMapTable {

public:

HashTableEntry \*\*ht, \*\*top;

HashMapTable() {

ht = new HashTableEntry\*[T\_S];

for (int i = 0; i < T\_S; i++)

ht[i] = NULL;

}

int HashFunc(int key) {

return key % T\_S;

}

void Insert(int k, int v) {

int hash\_v = HashFunc(k);

HashTableEntry\* p = NULL;

HashTableEntry\* en = ht[hash\_v];

while (en!= NULL) {

p = en;

en = en->n;

}

if (en == NULL) {

en = new HashTableEntry(k, v);

if (p == NULL) {

ht[hash\_v] = en;

} else {

p->n = en;

}

} else {

en->v = v;

}

}

void Remove(int k) {

int hash\_v = HashFunc(k);

HashTableEntry\* en = ht[hash\_v];

HashTableEntry\* p = NULL;

if (en == NULL || en->k != k) {

cout<<"No Element found at key "<<k<<endl;

return;

}

while (en->n != NULL) {

p = en;

en = en->n;

}

if (p != NULL) {

p->n = en->n;

}

delete en;

cout<<"Element Deleted"<<endl;

}

void SearchKey(int k) {

int hash\_v = HashFunc(k);

bool flag = false;

HashTableEntry\* en = ht[hash\_v];

if (en != NULL) {

while (en != NULL) {

if (en->k == k) {

flag = true;

}

if (flag) {

cout<<"Element found at key "<<k<<": ";

cout<<en->v<<endl;

}

en = en->n;

}

}

if (!flag)

cout<<"No Element found at key "<<k<<endl;

}

~HashMapTable() {

delete [] ht;

}

};

int main() {

HashMapTable hash;

int k, v;

int c;

while (1) {

cout<<"1.Insert element into the table"<<endl;

cout<<"2.Search element from the key"<<endl;

cout<<"3.Delete element at a key"<<endl;

cout<<"4.Exit"<<endl;

cout<<"Enter your choice: ";

cin>>c;

switch(c) {

case 1:

cout<<"Enter element to be inserted: ";

cin>>v;

cout<<"Enter key at which element to be inserted: ";

cin>>k;

hash.Insert(k, v);

break;

case 2:

cout<<"Enter key of the element to be searched: ";

cin>>k;

hash.SearchKey(k);

break;

case 3:

cout<<"Enter key of the element to be deleted: ";

cin>>k;

hash.Remove(k);

break;

case 4:

exit(1);

default:

cout<<"\nEnter correct option\n";

}

}

return 0;

}

1. **Write a program to implement the concept of graph in data structure:**

#include <stdio.h>

#include <stdlib.h>

#define N 6

struct Graph

{

struct Node\* head[N];

};

struct Node

{

int dest;

struct Node\* next;

};

struct Edge

{

int src, dest;

};

struct Graph\* createGraph(struct Edge edges[], int n)

{

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

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

{

graph->head[i] = NULL;

}

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

{

int src = edges[i].src;

int dest = edges[i].dest;

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

newNode->dest = dest;

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

graph->head[src] = newNode;

}

return graph;

}

void printGraph(struct Graph\* graph)

{

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

{

struct Node\* ptr = graph->head[i];

while (ptr != NULL)

{

printf("(%d —> %d)\t", i, ptr->dest);

ptr = ptr->next;

}

printf("\n");

}

}

int main(void)

{

struct Edge edges[] =

{

{0, 1}, {1, 2}, {2, 0}, {2, 1}, {3, 2}, {4, 5}, {5, 4}

};

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

struct Graph \*graph = createGraph(edges, n);

printGraph(graph);

return 0;

}

**Array:**

* Array is a linear collection of finite number of homogeneous data elements.
* When an array is stored in memory its elements gets stored in consecutive memory locations.
* Elements of the array can be accessed with an index.

**Linked list:**

* A linked list can be defined an the linear collection of elements where each element is stored in a node and the linear order between elements is given by the means of pointers instead of sequential memory locations.
* In linked list or one-way list, each node is divided into two parts:

1. First part of the node contain the element itself.
2. Second part which is termed as next field or pointer field contain the address of the next node in the list.

**Single or one-way linked list:**

* In this each node has atleast two parts

1. First part is known as info part which load the element.
2. Second part is known as next part which hold the address of the next node.

* Address of first node is stored in as special variable known as begin.
* Next part of the last node contains null indicating the end of the linked list.

**Stack:**

* In arrays the insertion and deletion of an element can take place at any position of the array but in the case of stack the insertion and deletion of an element can occur only at one end which is known as top.
* In stack insertion operation is known as push and deletion operation is known as pop.
* Stack is also called last in first out(LIFO) list which means that last item added to the stack will be the first item to be removed from the stack.