**TERMWORK-1**

#include<stdio.h>

#include<stdlib.h>

void read\_array(int \*,int);

void display\_array(int\*,int);

int \* max\_element(int \*,int);

void reverse\_array(int \*,int);

int search\_element(int \*,int);

void main()

{

int n,arr[10],i,\*max\_ele;

char option;

int \*p,s\_element;

p=arr;

do{

printf("Enter 1:Read the array elements\n");

printf("Enter 2:display the array elements\n");

printf("Enter 3:find max element\n");

printf("Enter 4:To print reverse array\n");

printf("Enter 5:To find key element\n ");

printf("Enter 0:To Terminate the program\n");

scanf(" %c",&option);

switch(option)

{

case '1': printf("Enter the size of an array:\n");

scanf("%d",&n);

read\_array(arr,n);

break;

case '2': display\_array(arr,n);

break;

case '3': max\_ele=max\_element(arr,n);

printf("The max element is %d\n",\*max\_ele);

break;

case '4': reverse\_array(arr,n);

break;

case '5': s\_element=search\_element(arr,n);

if (s\_element==0)

{

printf("The element not found\n");

}

else

{

printf("The element is found at %d location\n",s\_element);

}

break;

default : printf("You entered the wrong choice!\n");

}

}

while(option!='0');

}

void read\_array(int \*p,int n)

{

int i;

printf("Enter the %d array elements\n",n);

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

{

scanf("%d",(p+i));

}

}

void display\_array(int \*p,int n)

{

int i;

printf("The array elements are\n:");

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

{

printf("a[%d]=%d\n",i,\*(p+i));

}

}

int \* max\_element(int \*p,int n)

{

int max,i,j;

max=\*p;

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

{

if (\*(p+i)>max)

{

max=\*(p+i);

j=i;

}

}

return (p+j);

}

void reverse\_array(int \*p,int n)

{

int i;

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

{

printf("a[%d]=%d\n",i,\*(p+i));

}

}

int search\_element(int \*p,int n)

{

int key,i,flag=0;

printf("Enter the element you want to search\n");

scanf("%d",&key);

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

{

if(\*p==key)

{

flag=1;

printf("The key element is found at %d index\n",i);

return (p);

break;

}

p++;

}

if (flag==0)

{

return 0;

}

}

**TERMWORK-2**

//demonstration of structures

#include<stdio.h>

#include<string.h>

struct student

{

char name[50];

int cetRank;

char branch[20];

};

void input(struct student s[],int n);

void segregateStudents(struct student[] ,int ,char targetBranch[]);

void display(struct student s[],int n);

void main()

{

int n;

printf("Enter the no. of students :\n");

scanf("%d",&n);

struct student s[n];

input(s,n);

display(s,n);

segregateStudents(s,n,"cse");

segregateStudents(s,n,"enc");

segregateStudents(s,n,"eee");

}

void input(struct student s[],int n)

{

printf("Input student details:\n");

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

{

printf("Enter the details of %d Student\n",i+1);

printf("Enter name:\n");

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

printf("Enter CET Rank:\n");

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

printf("Enter Branch:\n");

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

}

}

void display(struct student s[],int n)

{

printf("Student's details:\n");

printf("sl.no\t\t\t\tName\t\t\t\tCetRank\t\t\t\tBranch\n");

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

{

printf("%d\t\t\t\t%s\t\t\t\t%d\t\t\t\t%s\n",i+1,s[i].name,s[i].cetRank,s[i].branch);

}

}

void segregateStudents(struct student s[] ,int n,char targetBranch[])

{

printf("\nlist of %s students\n\n",targetBranch);

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

{

if(strcasecmp(targetBranch,s[i].branch)==0)

{

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

}

}

}

**TERMWORK-3**

//conversion of infix to postfix exp

#include<stdio.h>

#include<ctype.h>

#include<string.h>

int top=-1;

char stack[20];

char postfix[20];

void push(char ch)

{

stack[++top]=ch;

}

char pop()

{

return stack[top--];

}

int priority(int ch)

{

if (ch=='(')

return 0;

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

return 1;

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

return 2;

return -1;

}

int main()

{

char infix[20];

char ch;

int p=0;

printf("Enter any valid infix expression:\n");

gets(infix);

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

{

ch=infix[i];

if (isalnum(ch))

{

postfix[p++]=ch;

}

else if (ch=='(')

{

push(ch);

}

else if ((ch=='+'|| ch=='\*' || ch=='/'||ch=='-'))

{

while (top!=-1 && priority(stack[top])>=priority(ch))

{

postfix[p++]=pop();

}

push(ch);

}

else

{ while(top!=-1 && stack[top]!='(')

{

postfix[p++]=pop();

}

if(top!= -1 && stack[top]=='('){

pop();}

}

}

while(top!=-1){

postfix[p++]=pop();

}

printf("The postfix exp is :\n");

puts(postfix);

}

**TERMWORK-5**

#include<stdio.h>

#include<stdlib.h>

typedef struct Node{

int data;

struct Node\* next;

}NODE;

NODE\* add(NODE\* head,int item)

{

NODE\* prev,\*curr;

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

if(newNode==NULL){

printf("\nMalloc failure!");

exit(1);

}

newNode->data=item;

newNode->next=NULL;

if(head==NULL)

head=newNode;

else if(item<head->data){

newNode->next=head;

head=newNode;

}

else{

prev=head;

curr=head;

curr=curr->next;

while (curr && item>curr->data){

prev=prev->next;

curr=curr->next;

}

newNode->next=curr;

prev->next=newNode;

}

return head;

}

int search(NODE \*head,int item)

{

if(head==NULL){

printf("\nWarehouse is empty");

return 0;

}

while(head && (item>head->data))

head=head->next;

if (head==NULL)

return 0;

else if (item==head->data)

return 1;

else return 0;

}

void disp(NODE \*head)

{

if(head==NULL){

printf("\nWarehouse is empty");

return ;

}

printf("\nThe warehouse items are:");

while(head){

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

head=head->next;

}

}

int main()

{

NODE\* head =NULL;

int opt,item;

while(1){

printf("\n1.Add item\n2.Display Warehouse\n3.Search item\n4.Exit");

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

scanf("%d",&opt);

switch(opt)

{

case 1:printf("\nEnter item to add\n");

scanf("%d",&item);

head=add(head,item);

break;

case 2:disp(head);

break;

case 3:printf("\nEnter the item to search");

scanf("%d",&item);

if(search(head,item))

printf("\nitem %d is present in the warehouse",item);

else

printf("\nitem %d is not present in the warehouse",item);

break;

case 4: exit(0);

}

}

return 0;

}

**TERMWORK 5(singly linked list)**

#include<stdio.h>

#include<stdlib.h>

typedef struct Node{

int data;

struct Node\* next;

}NODE;

NODE\* add(NODE\* head,int item)

{

NODE\* prev,\*curr;

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

if(newNode==NULL){

printf("\nMalloc failure!");

exit(1);

}

newNode->data=item;

newNode->next=NULL;

if(head==NULL)

head=newNode;

else if(item<head->data){

newNode->next=head;

head=newNode;

}

else{

prev=head;

curr=head;

curr=curr->next;

while (curr&&item>curr->data){

prev=prev->next;

curr=curr->next;

}

newNode->next=curr;

prev->next=newNode;

}

return head;

}

int search(NODE \*head,int item)

{

if(head==NULL){

printf("\nWarehouse is empty");

return 0;

}

while(head && (item>head->data))

head=head->next;

if (head==NULL)

return 0;

else if (item==head->data)

return 1;

else return 0;

}

void disp(NODE \*head)

{

if(head==NULL){

printf("\nWarehouse is empty");

return ;

}

printf("\nThe warehouse items are:");

while(head){

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

head=head->next;

}

}

int main()

{

NODE\* head =NULL;

int opt,item;

while(1){

printf("\n1.Add item\n2.Display Warehouse\n3.Search item\n4.Exit");

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

scanf("%d",&opt);

switch(opt)

{

case 1:printf("\nEnter item to add\n");

scanf("%d",&item);

head=add(head,item);

break;

case 2:disp(head);

break;

case 3:printf("\nEnter the item to search");

scanf("%d",&item);

if(search(head,item))

printf("\nitem %d is present in the warehouse",item);

else

printf("\nitem %d is not present in the warehouse",item);

break;

case 4: exit(0);

}

}

return 0;

}

**TERMWORK 6(doubly linked list)**

#include <stdio.h>

#include <stdlib.h>

// Node structure for doubly linked list

struct Node {

int item;

struct Node\* prev;

struct Node\* next;

};

struct Node \*head,\*newNode,\*temp,\*tail;

// Function to create a new node

void create() {

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

printf("Enter the item\n");

scanf("%d",&newNode->item);

newNode->prev = NULL;

newNode->next = NULL;

if(head==NULL){

head=temp=newNode;

}

else{

temp->next=newNode;

newNode->prev=temp;

temp=newNode;

}

}

// Function to insert a node at the front of the list

void insertAtFront() {

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

printf("Enter the item");

scanf("%d",&newNode->item);

newNode->prev = NULL;

newNode->next = NULL;

head->prev=newNode;

newNode->next=head;

head=newNode;

}

void insertAtEnd() {

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

printf("Enter the item");

scanf("%d",&newNode->item);

newNode->prev = NULL;

newNode->next = NULL;

tail=head;

while(tail->next!=NULL)

tail=tail->next;

tail->next=newNode;

newNode->prev=tail;

tail=newNode;

printf("Ele added");

}

// Function to display the content of the list

void display() {

temp=head;

printf("List content: ");

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

// Function to search for a value in the list

void search() {

int key;

temp=head;

int position = 1;

printf("enter the key:");

scanf("%d",&key);

while (temp!=NULL&&temp->item!=key) {

temp=temp->next;

position++;

}

if(temp==NULL){

printf("%d not found in list",key);

}

else{

printf("%d found t position %d in list",key,position);

}

}

// Function to perform bubble sort on the list

void sortList() {

struct Node \*i ,\*j;

int temp,current;

if (head == NULL) {

printf("List is empty. Nothing to sort.\n");

return;

}

for (i = head; i != NULL; i= i->next) {

for (j= i->next; j != NULL; j = j->next) {

if (i->item > j->item) {

// Swap data

temp = i->item;

i->item = j->item;

j->item = temp;

}

}

}

printf("List sorted in ascending order.\n");

}

int main() {

int choice;

head=temp=tail=newNode=NULL;

do {

printf("\n1. Create a doubly linked list \n2. Insert a new node at front \n3. insert at end\n4. display the list\n 5. to search \n 6. to sort\n 7. Exit\n");

printf("Enter your choice:\n ");

scanf("%d", &choice);

switch (choice) {

case 1:create();

break;

case 2:

insertAtFront();

break;

case 3:

insertAtEnd();

break;

case 4:

display();

break;

case 5:

search();

break;

case 6:

sortList();

break;

case 7:exit(0);

default:

printf("Invalid choice. Please enter a valid option.\n");

}

} while (choice != 7);

}

**TERMWORK 7**

#include <stdio.h>

#include <stdlib.h>

// Define a Node structure for the binary search tree

struct Node {

int key;

struct Node \*left, \*right;

};

// Function to create a new Node

struct Node\* createNode(int key) {

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

newNode->key = key;

newNode->left = newNode->right = NULL;

return newNode;

}

// Function to insert a key into the BST

struct Node\* insert(struct Node\* root, int key) {

if (root == NULL)

return createNode(key);

if (key < root->key)

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

else if (key > root->key)

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

return root;

}

// Function to search for a key in the BST

struct Node\* search(struct Node\* root, int key) {

if (root == NULL || root->key == key)

return root;

if (key < root->key)

return search(root->left, key);

return search(root->right, key);

}

// Function for in-order traversal of the BST

void inorderTraversal(struct Node\* root) {

if (root != NULL) {

inorderTraversal(root->left);

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

inorderTraversal(root->right);

}

}

// Function for pre-order traversal of the BST

void preorderTraversal(struct Node\* root) {

if (root != NULL) {

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

preorderTraversal(root->left);

preorderTraversal(root->right);

}

}

// Function for post-order traversal of the BST

void postorderTraversal(struct Node\* root) {

if (root != NULL) {

postorderTraversal(root->left);

postorderTraversal(root->right);

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

}

}

int main() {

struct Node\* root = NULL;

int choice, rollno;

struct Node\* result;

do {

printf("\nBinary Search Tree Operations:\n");

printf("1. Insert a node\n");

printf("2. Search for a node\n");

printf("3. In-order traversal\n");

printf("4. Pre-order traversal\n");

printf("5. Post-order traversal\n");

printf("0. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the roll number to insert: ");

scanf("%d", &rollno);

root = insert(root, rollno);

break;

case 2:

printf("Enter the roll number to search: ");

scanf("%d", &rollno);

result = search(root, rollno);

if (result != NULL)

printf("Node with roll number %d found in the tree.\n", rollno);

else

printf("Node with roll number %d not found in the tree.\n", rollno);

break;

case 3:

printf("In-order traversal: ");

inorderTraversal(root);

printf("\n");

break;

case 4:

printf("Pre-order traversal: ");

preorderTraversal(root);

printf("\n");

break;

case 5:

printf("Post-order traversal: ");

postorderTraversal(root);

printf("\n");

break;

case 0:

printf("Exiting the program.\n");

break;

default:

printf("Invalid choice. Please enter a valid option.\n");

}

} while (choice != 0);

return 0;

}

**TERMWORK 8**

#include <stdio.h>

#include <string.h>

#define TABLE\_SIZE 10

// Structure to store customer information

struct Customer {

int custid;

char custname[50];

char custphno[15];

};

// Hash table to store customer records

struct Customer hashTable[TABLE\_SIZE];

// Function to calculate the hash value for a given customer id

int hashFunction(int custid) {

return custid % TABLE\_SIZE;

}

// Function to insert a new customer record into the hash table

void insertCustomer(int custid, const char \*custname, const char \*custphno) {

int index = hashFunction(custid);

// Linear probing to handle collisions

while (hashTable[index].custid != -1) {

index = (index + 1) % TABLE\_SIZE;

}

// Insert the new customer record

hashTable[index].custid = custid;

strcpy(hashTable[index].custname, custname);

strcpy(hashTable[index].custphno, custphno);

}

// Function to search for customer information using custid

void searchCustomer(int custid) {

int index = hashFunction(custid);

int originalIndex = index;

// Linear probing to find the correct record

while (hashTable[index].custid != -1) {

if (hashTable[index].custid == custid) {

printf("Customer found:\n");

printf("Customer ID: %d\n", hashTable[index].custid);

printf("Customer Name: %s\n", hashTable[index].custname);

printf("Customer Phone Number: %s\n", hashTable[index].custphno);

return;

}

index = (index + 1) % TABLE\_SIZE;

if (index == originalIndex) {

// Customer not found

printf("Customer not found for ID %d\n", custid);

return;

}

}

// Customer not found

printf("Customer not found for ID %d\n", custid);

}

// Function to display all customer records

void displayRecords() {

printf("Customer Records:\n");

for (int i = 0; i < TABLE\_SIZE; i++) {

if (hashTable[i].custid != -1) {

printf("Index %d: (ID: %d, Name: %s, Phone: %s)\n", i, hashTable[i].custid, hashTable[i].custname, hashTable[i].custphno);

}

}

}

int main() {

int data;

// Initialize hash table with -1 indicating empty slots

for (int i = 0; i < TABLE\_SIZE; i++) {

hashTable[i].custid = -1;

}

// Insert some sample data

insertCustomer(101, "Ajay", "9562385665");

insertCustomer(205, "Pooja", "9874523610");

insertCustomer(312, "Swayam", "7895463210");

// Display all records

displayRecords();

// Search for a customer

printf("\n Enter Item to search:");

scanf("%d",&data);

searchCustomer(data);

return 0;

}