1. Write a C program to implement a AVL tree with all rotations.

#include <stdio.h>

#include <stdlib.h>

struct Node

{

int key;

struct Node \*left;

struct Node \*right;

int height;

};

int getHeight(struct Node \*n){

if(n==NULL)

return 0;

return n->height;

}

struct Node \*createNode(int key){

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

node->key = key;

node->left = NULL;

node->right = NULL;

node->height = 1;

return node;

}

int max (int a, int b){

return (a>b)?a:b;

}

int getBalanceFactor(struct Node \* n){

if(n==NULL){

return 0;

}

return getHeight(n->left) - getHeight(n->right);

}

struct Node\* rightRotate(struct Node\* y){

struct Node\* x = y->left;

struct Node\* T2 = x->right;

x->right = y;

y->left = T2;

x->height = max(getHeight(x->right), getHeight(x->left)) + 1;

y->height = max(getHeight(y->right), getHeight(y->left)) + 1;

return x;

}

struct Node\* leftRotate(struct Node\* x){

struct Node\* y = x->right;

struct Node\* T2 = y->left;

y->left = x;

x->right = T2;

x->height = max(getHeight(x->right), getHeight(x->left)) + 1;

y->height = max(getHeight(y->right), getHeight(y->left)) + 1;

return y;

}

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

if (node == NULL)

return createNode(key);

if (key < node->key)

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

else if (key > node->key)

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

node->height = 1 + max(getHeight(node->left), getHeight(node->right));

int bf = getBalanceFactor(node);

if(bf>1 && key < node->left->key){

return rightRotate(node);

}

if(bf<-1 && key > node->right->key){

return leftRotate(node);

}

if(bf>1 && key > node->left->key){

node->left = leftRotate(node->left);

return rightRotate(node);

}

if(bf<-1 && key < node->right->key){

node->right = rightRotate(node->right);

return leftRotate(node);

}

return node;

}

void preOrder(struct Node \*root)

{

if(root != NULL)

{

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

preOrder(root->left);

preOrder(root->right);

}

}

int main(){

struct Node \* root = NULL;

root = insert(root, 1);

root = insert(root, 2);

root = insert(root, 4);

root = insert(root, 5);

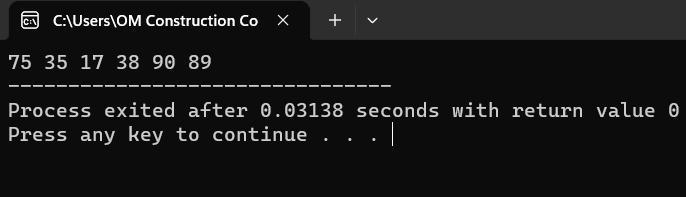
root = insert(root, 6);

root = insert(root, 3);

preOrder(root);

return 0;

}



1. Write a C program to traverse a Binary tree.

#include <stdio.h>

#include <stdlib.h>

struct tnode {

int data;

struct tnode \*left, \*right;

};

struct tnode \*root = NULL;

struct tnode \* createNode(int data) {

struct tnode \*newNode;

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

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return (newNode); }

void insertion(struct tnode \*\*node, int data) {

if (!\*node) {

\*node = createNode(data);

} else if (data < (\*node)->data) {

insertion(&(\*node)->left, data);

} else if (data > (\*node)->data) {

insertion(&(\*node)->right, data); } }

void postOrder(struct tnode \*node) {

if (node) {

postOrder(node->left);

postOrder(node->right);

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

return; }

void preOrder(struct tnode \*node) {

if (node) {

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

preOrder(node->left);

preOrder(node->right); }

return; }

void inOrder(struct tnode \*node) {

if (node) {

inOrder(node->left);

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

inOrder(node->right); }

return; }

int main() {

int data, ch;

while (1) {

printf("\n1. Insertion\n2. Pre-order\n");

printf("3. Post-order\n4. In-order\n");

printf("5. Exit\nEnter your choice:");

scanf("%d", &ch);

switch (ch) {

case 1:

printf("Enter ur data:");

scanf("%d", &data);

insertion(&root, data);

break;

case 2:

preOrder(root);

break;

case 3:

postOrder(root);

break;

case 4:

inOrder(root);

break;

case 5:

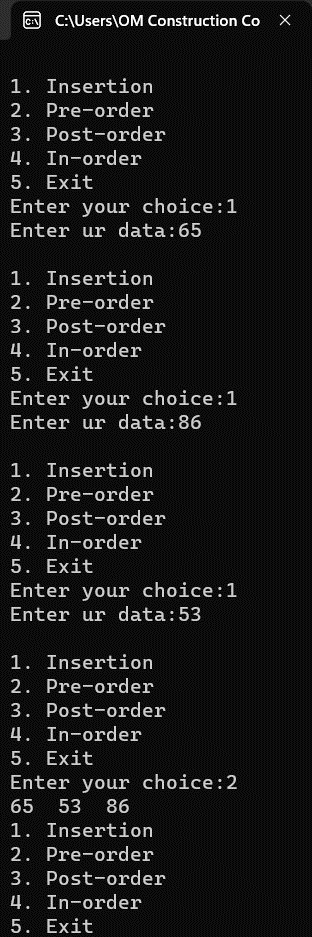
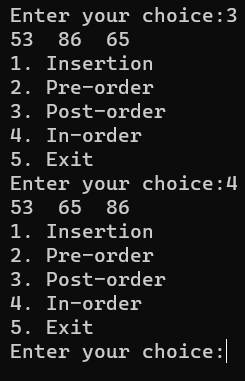
exit(0);

default:

printf("U've entered wrong opetion\n");

break; } }

return 0; }

1. Write a C program to implement hashing using linear probing.

#include <stdio.h>

#include<stdlib.h>

#define TABLE\_SIZE 10

int h[TABLE\_SIZE]={NULL};

void insert()

{

int key,index,i,flag=0,hkey;

printf("\nenter a value to insert into hash table\n");

scanf("%d",&key);

hkey=key%TABLE\_SIZE;

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

{

index=(hkey+i)%TABLE\_SIZE;

if(h[index] == NULL)

{

h[index]=key;

break; } }

if(i == TABLE\_SIZE)

printf("\nelement cannot be inserted\n");

}

void search()

{ int key,index,i,flag=0,hkey;

printf("\nenter search element\n");

scanf("%d",&key);

hkey=key%TABLE\_SIZE;

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

{ index=(hkey+i)%TABLE\_SIZE;

if(h[index]==key)

{

printf("value is found at index %d",index);

break;

}

}

if(i == TABLE\_SIZE)

printf("\n value is not found\n");

}

void display()

{ int i;

printf("\nelements in the hash table are \n");

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

printf("\nat index %d \t value = %d",i,h[i]);

}

main()

{ int opt,i;

while(1)

{

printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");

scanf("%d",&opt);

switch(opt)

{

case 1:

insert();

break;

case 2:

display();

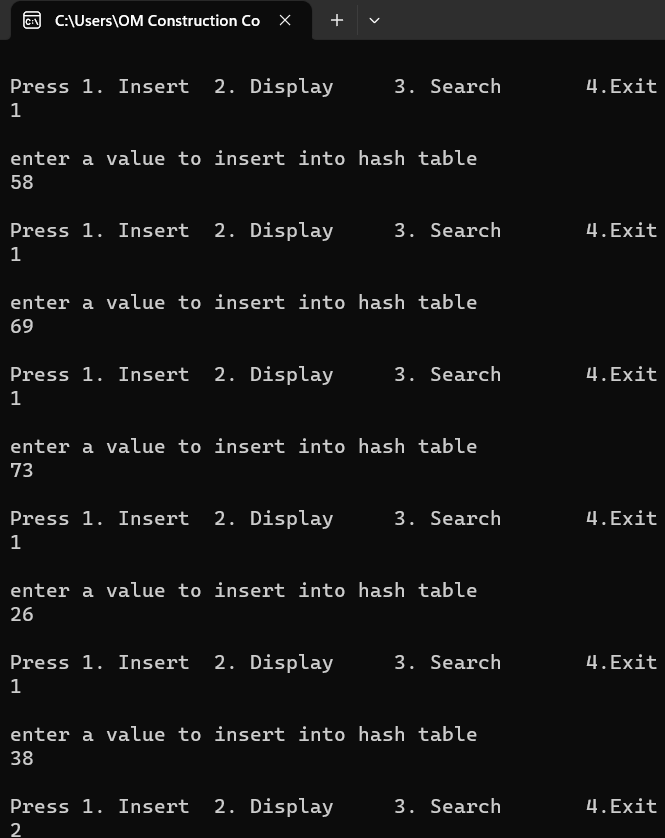
break;

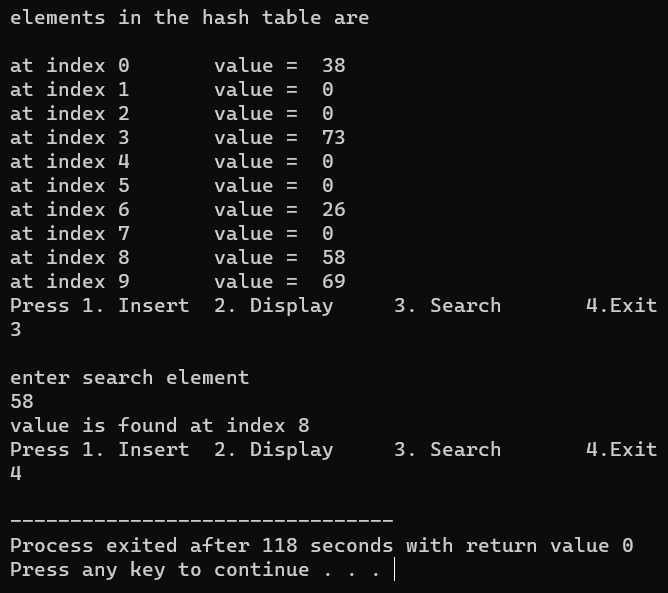
case 3:

search();

break;

case 4:exit(0); } } }





4.

A Write a C program to implement selection sort.

#include <stdio.h>

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

int i, j, small;

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

small = i;

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

if (arr[j] < arr[small])

small = j;

int temp = arr[small];

arr[small] = arr[i];

arr[i] = temp;

} }

void printArr(int a[], int n) {

int i;

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

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

}

int main() {

int a[] = { 12, 31, 25, 8, 32, 17 };

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

printf("Before sorting array elements are - \n");

printArr(a, n);

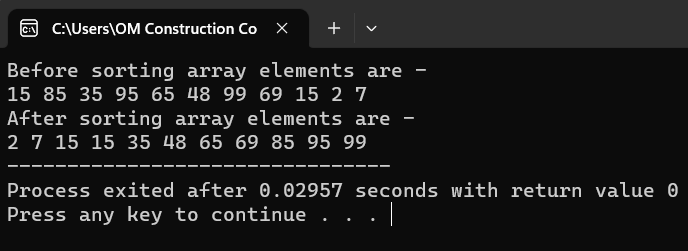
selection(a, n);

printf("\nAfter sorting array elements are - \n");

printArr(a, n);

return 0;

}



B Write a C program to implement insertion sort.

#include <stdio.h>

void insert(int a[], int n)

{ int i, j, temp;

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

temp = a[i];

j = i - 1;

while(j>=0 && temp <= a[j])

{

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

j = j-1;

}

a[j+1] = temp;

}

} void printArr(int a[], int n)

{

int i;

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

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

}

int main()

{

int a[] = { 21, 71, 28, 8, 2, 19 };

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

printf("Before sorting array elements are - \n");

printArr(a, n);

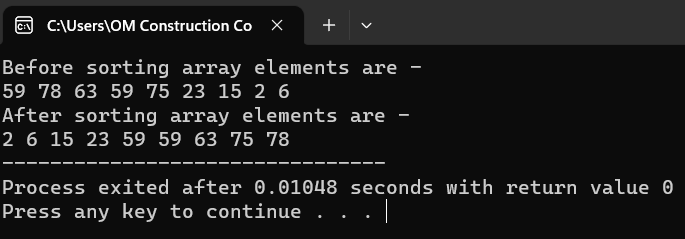
insert(a, n);

printf("\nAfter sorting array elements are - \n");

printArr(a, n);

return 0;

}



c. write a c program to implement quick sort

#include <stdio.h>

void swap(int\* a, int\* b) {

int t = \*a;

\*a = \*b;

\*b = t;

}

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

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high - 1; j++) {

if (arr[j] < pivot) {

i++;

swap(&arr[i], &arr[j]);

}

}

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

return (i + 1);

}

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

if (low < high) {

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

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

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

int i;

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

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

printf("\n");

}

int main() {

int arr[] = { 12, 17, 6, 25, 1, 5 };

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

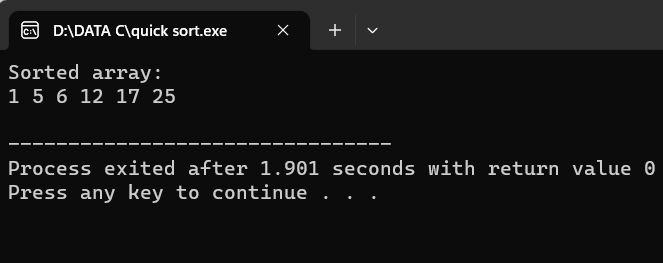
quickSort(arr, 0, n - 1);

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}



d. write a c program to implement Bubble sort

#include <stdio.h>

void bubble\_sort(int arr[], int n) {

int i, j;

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

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

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

int temp = arr[j];

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

arr[j + 1] = temp;

}

}

}

}

int main() {

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

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

bubble\_sort(arr, n);

printf("Sorted array: ");

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

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

}

return 0;

}

