Coding Assignment

Assignment-1:

1. **You are given an integer n consisting of digits 1, 2, and 3 and you can flip one digit to a 3. Return the maximum number you can make.**

#include<stdio.h>

#include<math.h>

int solve(int n) {

int index,add,x = 0,k = n;

while (k > 0) {

if ((k % 10) < 3) {

add = 3 - k % 10;

index += x + 1;

x = 0;

}

else x++;

k /= 10;

}

return (n + (pow(10, index - 1) \* add));

}

int main(){

int num;

printf("Enter a number for flipping one digit to 3's to get maximum number:");

scanf("%d",&num);

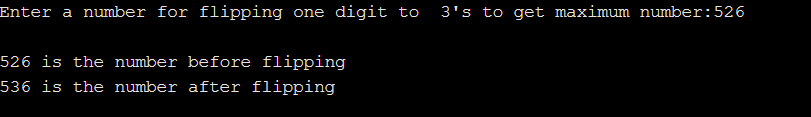
printf("\n%d is the number before flipping ",num);

printf("\n%d is the number after flipping",solve(num));

return 0;

}

Output:



1. Rotate matrix 90 degrees clockwise direction.

#include <stdio.h>

int main()

{

printf("Enter M and N of matrix A: ");

int m, n;

int p, q;

scanf("%d %d", &m, &n);

printf("Enter matrix elements:\n");

int i, j, matA[101][101];

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

{

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

{

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

}

}

printf("\nThe 90 degrees rotated matrix:\n");

int temp;

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

{

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

{

temp = matA[i][j];

matA[i][j] = matA[j][i];

matA[j][i] = temp;

}

}

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

{

for (j = 0; j < n / 2; j++)

{

temp = matA[i][j];

matA[i][j] = matA[i][n - 1 - j];

matA[i][n - 1 - j] = temp;

}

}

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

{

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

printf("%2d ", matA[i][j]);

printf("\n");

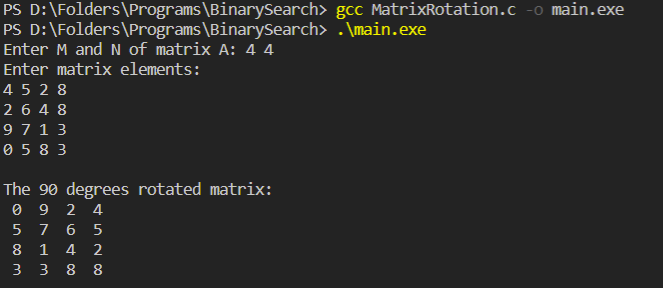
}

printf("\n");

return 0;

}

Output:



1. Given a list of integers nums, remove numbers that appear multiple times in the list, while maintaining the order of the appearance in the original list.

vector<int> solve(vector<int>& nums) {

unordered\_map<int, int> numsh;

vector<int> res={};

vector<int>:: iterator j;

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

{

if(numsh.find(nums[i])!=numsh.end())

{

auto it = numsh.find(nums[i]);

if(it != numsh.end())

it -> second++;

}

else

{

numsh.insert(make\_pair(nums[i],1));

}

}

for(auto itr:nums)

{

if(numsh.find(itr)!=numsh.end())

{

auto it = numsh.find(itr);

if(it->second>1)

{

continue;

}

else

{

res.insert(res.end(),itr);

}

}

}

return res;

}

Assignment-2:

1. Find length of the longest palindrome substring in a given string

#include <stdio.h>

#include <string.h>

void SubStr(char\* str, int low, int high){

int i;

for (i = low; i <= high; ++i)printf("%c", str[i]);

}

int result(char\* str)

{

int maxLength = 1,start = 0,len = strlen(str),i,low, high;

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

low = i - 1;

high = i;

while (low >= 0 && high < len && str[low] == str[high]) {

if (high - low + 1 > maxLength) {

start = low;

maxLength = high - low + 1;

}

--low;

++high;

}

low = i - 1;

high = i + 1;

while (low >= 0 && high < len && str[low] == str[high]) {

if (high - low + 1 > maxLength) {

start = low;

maxLength = high - low + 1;

}

--low;

++high;

}

}

printf("Longest palindrome substring is: ");

SubStr(str, start, start + maxLength - 1);

return maxLength;

}

int main()

{

char str[50];

printf("Enter a string to check for longest substring palindrome: ");

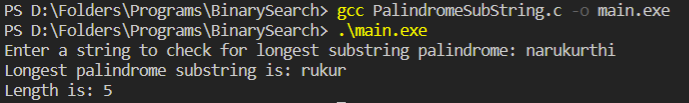
gets(str);

printf("\nLength is: %d", result(str));

return 0;

}

Output:



1. Given a string s of words delimited by spaces, reverse the order of words.

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100

int main(){

char str[100], reverse[100];

int len, i, index, wordStart, wordEnd;

printf("Enter any string: ");

gets(str);

len = strlen(str);

index = 0;

wordStart = len - 1;

wordEnd = len - 1;

while(wordStart > 0){

if(str[wordStart] == ' '){

i = wordStart + 1;

while(i <= wordEnd){

reverse[index] = str[i];

i++;

index++;

}

reverse[index++] = ' ';

wordEnd = wordStart - 1;

}

wordStart--;

}

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

reverse[index] = str[i];

index++;

}

reverse[index] = '\0';

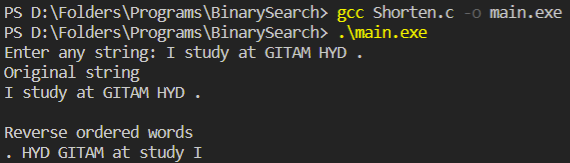
printf("Original string \n%s\n\n", str);

printf("Reverse ordered words \n%s", reverse);

return 0;

}

Output:



1. Given two lowercase alphabet strings s1 and s2, determine if s1 is a subsequence of s2.

bool solve(string s1, string s2) {

int i = 0;

int j = 0;

int n = s1.length();

int m = s2.length();

int k = 0;

while (i < n && j < m)

{

if (s1[i] == s2[j])

{

i++;

j++;

k++;

}

else

{

j++;

}

}

return n == k;

}

1. Given a lowercase alphabet string s, return the minimum number of characters that we need to delete such that the frequency of each character occurs a unique number of times.

int solve(string s) {

unordered\_map<char, int> freq;

for (auto& c : s) freq[c]++;

vector<int> freqCounts;

for (auto& p : freq) freqCounts.push\_back(p.second);

sort(freqCounts.rbegin(), freqCounts.rend());

int ret = 0;

int lastFreq = 1e9;

for (int freq : freqCounts)

{

if (freq > lastFreq)

{

int rem = freq - lastFreq;

ret += rem;

freq -= rem;

}

lastFreq = max(0, freq - 1);

}

return ret;

}

1. Given two strings s0 and s1, return the number of substrings where s1 contains any anagram of s0.

int solve(string s0, string s1) {

int tbl[26] = {0}, count = 0, ans = 0;

for (char c : s0)

{

tbl[c - 'a']++;

if (tbl[c - 'a'] == 1) count++;

}

for (int i = 0; i < s1.size(); i++) {

tbl[s1[i] - 'a']--;

if (tbl[s1[i] - 'a'] == 0)

count--;

else if (tbl[s1[i] - 'a'] == -1)

count++;

if (i >= s0.size())

{

tbl[s1[i - s0.size()] - 'a']++;

if (tbl[s1[i - s0.size()] - 'a'] == 1)

count++;

else if (tbl[s1[i - s0.size()] - 'a'] == 0)

count--;

}

if (!count) ans++;

}

return ans;

}

Assignment-3:

1. GCD of two numbers using recursion.

#include <stdio.h>

int gcd(int n1, int n2)

{

if (n2 != 0)

{

return gcd(n2, n1 % n2);

}

else

{

return n1;

}

}

int main()

{

int n1, n2;

printf("Enter two positive integers: ");

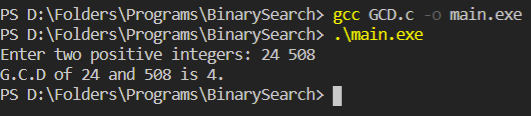
scanf("%d %d", &n1, &n2);

printf("G.C.D of %d and %d is %d.", n1, n2, gcd(n1, n2));

return 0;

}

Output:



1. Sum of digits using recursion.

#include <stdio.h>

int sum(int num)

{

if (num != 0)

{

return (num % 10 + sum(num / 10));

}

else

{

return 0;

}

}

int main()

{

int num, result;

printf("Enter the number: ");

scanf("%d", &num);

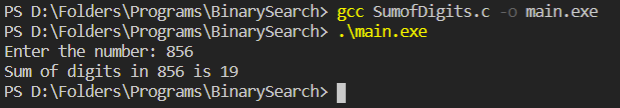
result = sum(num);

printf("Sum of digits in %d is %d\n", num, result);

return 0;

}

Output:



1. Power of any number using recursion.

#include <stdio.h>

int power(int base, int pow)

{

if (pow != 0)

{

return (base \* power(base, pow - 1));

}

else

{

return 1;

}

}

int main()

{

int base, pow, result;

printf("Enter base and positive power: ");

scanf("%d %d", &base, &pow);

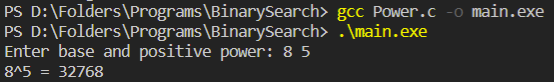
result = power(base, pow);

printf("%d^%d = %d", base, pow, result);

return 0;

}

Output:



1. Decimal number to binary number conversion using recursion.

#include <stdio.h>

int convert(int dec)

{

if (dec == 0)

{

return 0;

}

else

{

return (dec % 2 + 10 \* convert(dec / 2));

}

}

int main()

{

int dec;

printf("Enter a decimal number: ");

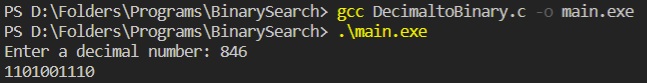
scanf("%d", &dec);

printf("%d", convert(dec));

return 0;

}

Output:



1. Check given string is palindrome or not.

#include <stdio.h>

#include <string.h>

void check(char word[], int index)

{

int len = strlen(word) - (index + 1);

if (word[index] == word[len])

{

if (index + 1 == len || index == len)

{

printf("The entered word is a palindrome\n");

return;

}

check(word, index + 1);

}

else

{

printf("The entered word is not a palindrome\n");

}

}

int main()

{

char word[15];

printf("Enter a word to check if it is a palindrome: ");

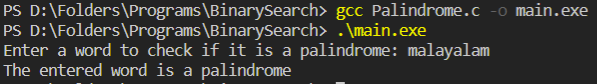
scanf("%s", word);

check(word, 0);

return 0;

}

Output:



Assignment-4:

1. Implement binary tree traversals.

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node\* left;

struct node\* right;

};

struct node\* newNode(int data)

{

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

node->data = data;

node->left = NULL;

node->right = NULL;

return (node);

}

void printPostorder(struct node\* node)

{

if (node == NULL)return;

printPostorder(node->left);

printPostorder(node->right);

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

}

void printInorder(struct node\* node)

{

if (node == NULL)return;

printInorder(node->left);

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

printInorder(node->right);

}

void printPreorder(struct node\* node)

{

if (node == NULL)return;

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

printPreorder(node->left);

printPreorder(node->right);

}

int main(){

struct node\* root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

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

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

printf("\nPreorder traversal of binary tree is \n");

printPreorder(root);

printf("\nInorder traversal of binary tree is \n");

printInorder(root);

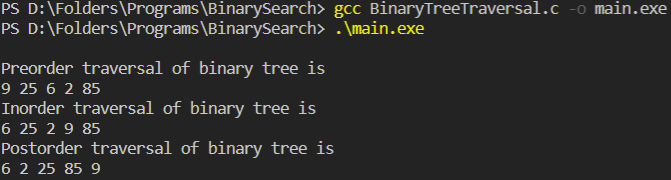
printf("\nPostorder traversal of binary tree is \n");

printPostorder(root);

return 0;

}

Output:



1. Construct binary search tree and apply insert, deletion, and display operations.

#include <stdio.h>

#include <stdlib.h>

typedef struct node {

int data;

struct node \*left;

struct node \*right;

struct node \*parent;

}node;

typedef struct binary\_search\_tree {

node \*root;

}binary\_search\_tree;

node\* new\_node(int data) {

node \*n = malloc(sizeof(node));

n->data = data;

n->left = NULL;

n->right = NULL;

n->parent = NULL;

return n;

}

binary\_search\_tree\* new\_binary\_search\_tree() {

binary\_search\_tree \*t = malloc(sizeof(binary\_search\_tree));

t->root = NULL;

return t;

}

node\* minimum(binary\_search\_tree \*t, node \*x) {

while(x->left != NULL)x = x->left;

return x;

}

void insert(binary\_search\_tree \*t, node \*n) {

node \*y = NULL;

node \*temp = t->root;

while(temp != NULL) {

y = temp;

if(n->data < temp->data)temp = temp->left;

else temp = temp->right;

}

n->parent = y;

if(y == NULL) t->root = n;

else if(n->data < y->data)y->left = n;

else y->right = n;

}

void transplant(binary\_search\_tree \*t, node \*u, node \*v) {

if(u->parent == NULL) t->root = v;

else if(u == u->parent->left) u->parent->left = v;

else u->parent->right = v;

if(v != NULL)v->parent = u->parent;

}

void delete(binary\_search\_tree \*t, node \*z) {

if(z->left == NULL) {

transplant(t, z, z->right);

free(z);

}

else if(z->right == NULL) {

transplant(t, z, z->left);

free(z);

}

else {

node \*y = minimum(t, z->right); //minimum element in right subtree

if(y->parent != z) {

transplant(t, y, y->right);

y->right = z->right;

y->right->parent = y;

}

transplant(t, z, y);

y->left = z->left;

y->left->parent = y;

free(z);

}

}

void inorder(binary\_search\_tree \*t, node \*n) {

if(n != NULL) {

inorder(t, n->left);

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

inorder(t, n->right);

}

}

int main() {

binary\_search\_tree \*t = new\_binary\_search\_tree();

node \*a, \*b, \*c, \*d, \*e, \*f, \*g, \*h, \*i, \*j, \*k, \*l, \*m;

a = new\_node(10);

b = new\_node(20);

c = new\_node(30);

d = new\_node(100);

e = new\_node(90);

f = new\_node(40);

g = new\_node(50);

h = new\_node(60);

i = new\_node(70);

j = new\_node(80);

k = new\_node(150);

l = new\_node(110);

m = new\_node(120);

insert(t, a);

insert(t, b);

insert(t, c);

insert(t, d);

insert(t, e);

insert(t, f);

insert(t, g);

insert(t, h);

insert(t, i);

insert(t, j);

insert(t, k);

insert(t, l);

insert(t, m);

delete(t, a);

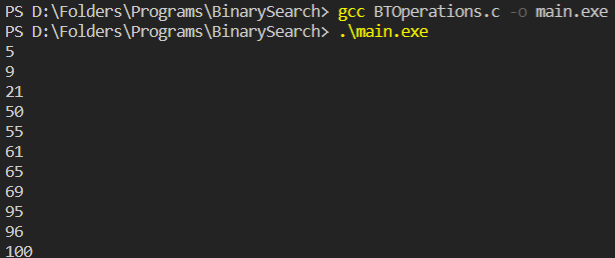
delete(t, m);

inorder(t, t->root);

return 0;

}

Output:



1. Stack - Infix to postfix expression

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

struct Stack{

int top;

unsigned capacity;

int\* array;

};

struct Stack\* createStack( unsigned capacity ){

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

if (!stack)return NULL;

stack->top = -1;

stack->capacity = capacity;

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

return stack;

}

int isEmpty(struct Stack\* stack){

return stack->top == -1 ;

}

char peek(struct Stack\* stack){

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

}

char pop(struct Stack\* stack){

if (!isEmpty(stack))

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

return '$';

}

void push(struct Stack\* stack, char op){

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

}

int isOperand(char ch){

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

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

}

int Prec(char ch){

switch (ch){

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':return 3;

}

return -1;

}

int infixToPostfix(char\* exp){

int i, k;

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

if(!stack) return -1 ;

for (i = 0, k = -1; exp[i]; ++i){

if (isOperand(exp[i])) exp[++k] = exp[i];

else if (exp[i] == '(') push(stack, exp[i]);

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

while (!isEmpty(stack) && peek(stack) != '(') exp[++k] = pop(stack);

if (!isEmpty(stack) && peek(stack) != '(') return -1;

else pop(stack);

}

else{

while (!isEmpty(stack) &&

Prec(exp[i]) <= Prec(peek(stack)))

exp[++k] = pop(stack);

push(stack, exp[i]);

}

}

while (!isEmpty(stack))exp[++k] = pop(stack );

exp[++k] = '\0';

printf( "%s", exp );

}

int main(){

char exp[50];

printf("Enter an infix expression");

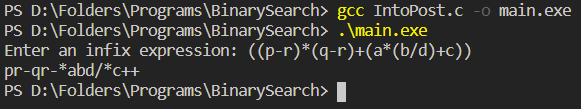
gets(exp);

infixToPostfix(exp);

return 0;

}

Output:



1. Circular queue

#include <stdio.h>

#define SIZE 5

int items[SIZE];

int front = -1, rear = -1;

int isFull() {

if ((front == rear + 1) || (front == 0 && rear == SIZE - 1)) return 1;

return 0;

}

int isEmpty() {

if (front == -1) return 1;

return 0;

}

void Enq(int element) {

if (isFull())printf("\n Queue is full!! \n");

else {

if (front == -1) front = 0;

rear = (rear + 1) % SIZE;

items[rear] = element;

printf("\n Inserted -> %d", element);

}

}

int Deq() {

int element;

if (isEmpty()) {

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

return (-1);

}

else {

element = items[front];

if (front == rear) {

front = -1;

rear = -1;

}

else front = (front + 1) % SIZE;

printf("\n Deleted element -> %d \n", element);

return (element);

}

}

void display() {

int i;

if (isEmpty())printf(" \n Empty Queue\n");

else {

printf("\n Front -> %d ", front);

printf("\n Items -> ");

for (i = front; i != rear; i = (i + 1) % SIZE) printf("%d ", items[i]);

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

printf("\n Rear -> %d \n", rear);

}

}

int main() {

Deq();

Enq(1);

Enq(2);

Enq(3);

Enq(4);

Enq(5);

Enq(6);

display();

Deq();

display();

Enq(7);

display();

Enq(8);

return 0;

}

Output:

