**PART B**

1. **Valid Parenthesis**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include<stdbool.h>

bool test(char \*s, int len) {

}

int main() {

char string1[80];

int n, i, x;

printf("Input a string: ");

scanf("%s", string1);

n = strlen(string1);

printf("Check whether brackets in the string are valid or not: %d", test(string1, n));

return 0;

}

1. **Bitwise Operator**

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

void calculate\_the\_maximum(int n, int k) {

}

int main() {

    int n, k;

    scanf("%d %d", &n, &k);

    calculate\_the\_maximum(n, k);

    return 0;

1. **Happy Numbers**

#include <stdio.h>

#include <stdbool.h>

// Function to calculate the sum of the squares of digits of a number

int sumOfSquares(int num) {

}

return sum;

}

// Function to determine if a number is a happy number

bool isHappy(int n) {

int slow = n;

int fast = n;

do {

slow = sumOfSquares(slow); // Move slow pointer by one step

fast = sumOfSquares(sumOfSquares(fast)); // Move fast pointer by two steps

} while (slow != fast);

return slow == 1; // If slow equals 1, it's a happy number

}

int main() {

int n1 = 19;

int n2 = 2;

printf("Is %d a happy number? %s\n", n1, isHappy(n1) ? "true" : "false");

printf("Is %d a happy number? %s\n", n2, isHappy(n2) ? "true" : "false");

return 0;

}

1. **Large & small triangle**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

struct triangle

{

int a;

int b;

int c;

};

typedef struct triangle triangle;

double area(triangle tr) {

double p = (tr.a + tr.b + tr.c) / 2.0;

return sqrt(p \* (p - tr.a) \* (p - tr.b) \* (p - tr.c));

}

int compare(const void \*a, const void \*b) {

}

void sort\_by\_area(triangle\* tr, int n) {

qsort(tr, n, sizeof(triangle), compare);

}

int main()

{

int n;

scanf("%d", &n);

triangle \*tr = malloc(n \* sizeof(triangle));

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

scanf("%d%d%d", &tr[i].a, &tr[i].b, &tr[i].c);

}

sort\_by\_area(tr, n);

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

printf("%d %d %d\n", tr[i].a, tr[i].b, tr[i].c);

}

return 0;

}

1. **Variadic Functions**

#include <stdarg.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define MIN\_ELEMENT 1

#define MAX\_ELEMENT 1000000

int sum (int count,...) {

va\_end(args);

return result;

}

int min(int count,...) {

va\_end(args);

return result;

}

int max(int count,...) {

}

va\_end(args);

return result;

}

int test\_implementations\_by\_sending\_three\_elements() {

srand(time(NULL));

int elements[3];

elements[0] = rand() % (MAX\_ELEMENT - MIN\_ELEMENT + 1) + MIN\_ELEMENT;

elements[1] = rand() % (MAX\_ELEMENT - MIN\_ELEMENT + 1) + MIN\_ELEMENT;

elements[2] = rand() % (MAX\_ELEMENT - MIN\_ELEMENT + 1) + MIN\_ELEMENT;

fprintf(stderr, "Sending following three elements:\n");

for (int i = 0; i < 3; i++) {

fprintf(stderr, "%d\n", elements[i]);

}

int elements\_sum = sum(3, elements[0], elements[1], elements[2]);

int minimum\_element = min(3, elements[0], elements[1], elements[2]);

int maximum\_element = max(3, elements[0], elements[1], elements[2]);

fprintf(stderr, "Your output is:\n");

fprintf(stderr, "Elements sum is %d\n", elements\_sum);

fprintf(stderr, "Minimum element is %d\n", minimum\_element);

fprintf(stderr, "Maximum element is %d\n\n", maximum\_element);

int expected\_elements\_sum = 0;

for (int i = 0; i < 3; i++) {

if (elements[i] < minimum\_element) {

return 0;

}

if (elements[i] > maximum\_element) {

return 0;

}

expected\_elements\_sum += elements[i];

}

return elements\_sum == expected\_elements\_sum;

}

int test\_implementations\_by\_sending\_five\_elements() {

srand(time(NULL));

int elements[5];

elements[0] = rand() % (MAX\_ELEMENT - MIN\_ELEMENT + 1) + MIN\_ELEMENT;

elements[1] = rand() % (MAX\_ELEMENT - MIN\_ELEMENT + 1) + MIN\_ELEMENT;

elements[2] = rand() % (MAX\_ELEMENT - MIN\_ELEMENT + 1) + MIN\_ELEMENT;

elements[3] = rand() % (MAX\_ELEMENT - MIN\_ELEMENT + 1) + MIN\_ELEMENT;

elements[4] = rand() % (MAX\_ELEMENT - MIN\_ELEMENT + 1) + MIN\_ELEMENT;

fprintf(stderr, "Sending following five elements:\n");

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

fprintf(stderr, "%d\n", elements[i]);

}

int elements\_sum = sum(5, elements[0], elements[1], elements[2], elements[3], elements[4]);

int minimum\_element = min(5, elements[0], elements[1], elements[2], elements[3], elements[4]);

int maximum\_element = max(5, elements[0], elements[1], elements[2], elements[3], elements[4]);

fprintf(stderr, "Your output is:\n");

fprintf(stderr, "Elements sum is %d\n", elements\_sum);

fprintf(stderr, "Minimum element is %d\n", minimum\_element);

fprintf(stderr, "Maximum element is %d\n\n", maximum\_element);

int expected\_elements\_sum = 0;

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

if (elements[i] < minimum\_element) {

return 0;

}

if (elements[i] > maximum\_element) {

return 0;

}

expected\_elements\_sum += elements[i];

}

return elements\_sum == expected\_elements\_sum;

}

int test\_implementations\_by\_sending\_ten\_elements() {

srand(time(NULL));

int elements[10];

elements[0] = rand() % (MAX\_ELEMENT - MIN\_ELEMENT + 1) + MIN\_ELEMENT;

elements[1] = rand() % (MAX\_ELEMENT - MIN\_ELEMENT + 1) + MIN\_ELEMENT;

elements[2] = rand() % (MAX\_ELEMENT - MIN\_ELEMENT + 1) + MIN\_ELEMENT;

elements[3] = rand() % (MAX\_ELEMENT - MIN\_ELEMENT + 1) + MIN\_ELEMENT;

elements[4] = rand() % (MAX\_ELEMENT - MIN\_ELEMENT + 1) + MIN\_ELEMENT;

elements[5] = rand() % (MAX\_ELEMENT - MIN\_ELEMENT + 1) + MIN\_ELEMENT;

elements[6] = rand() % (MAX\_ELEMENT - MIN\_ELEMENT + 1) + MIN\_ELEMENT;

elements[7] = rand() % (MAX\_ELEMENT - MIN\_ELEMENT + 1) + MIN\_ELEMENT;

elements[8] = rand() % (MAX\_ELEMENT - MIN\_ELEMENT + 1) + MIN\_ELEMENT;

elements[9] = rand() % (MAX\_ELEMENT - MIN\_ELEMENT + 1) + MIN\_ELEMENT;

fprintf(stderr, "Sending following ten elements:\n");

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

fprintf(stderr, "%d\n", elements[i]);

}

int elements\_sum = sum(10, elements[0], elements[1], elements[2], elements[3], elements[4],

elements[5], elements[6], elements[7], elements[8], elements[9]);

int minimum\_element = min(10, elements[0], elements[1], elements[2], elements[3], elements[4],

elements[5], elements[6], elements[7], elements[8], elements[9]);

int maximum\_element = max(10, elements[0], elements[1], elements[2], elements[3], elements[4],

elements[5], elements[6], elements[7], elements[8], elements[9]);

fprintf(stderr, "Your output is:\n");

fprintf(stderr, "Elements sum is %d\n", elements\_sum);

fprintf(stderr, "Minimum element is %d\n", minimum\_element);

fprintf(stderr, "Maximum element is %d\n\n", maximum\_element);

int expected\_elements\_sum = 0;

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

if (elements[i] < minimum\_element) {

return 0;

}

if (elements[i] > maximum\_element) {

return 0;

}

expected\_elements\_sum += elements[i];

}

return elements\_sum == expected\_elements\_sum;

}

int main ()

{

int number\_of\_test\_cases;

scanf("%d", &number\_of\_test\_cases);

while (number\_of\_test\_cases--) {

if (test\_implementations\_by\_sending\_three\_elements()) {

printf("Correct Answer\n");

} else {

printf("Wrong Answer\n");

}

if (test\_implementations\_by\_sending\_five\_elements()) {

printf("Correct Answer\n");

} else {

printf("Wrong Answer\n");

}

if (test\_implementations\_by\_sending\_ten\_elements()) {

printf("Correct Answer\n");

} else {

printf("Wrong Answer\n");

}

}

return 0;

}

1. Nth Tribonacci number

#include <stdio.h>

int tribonacci(int n)

{

// write your code here

return c;

}

int main()

{

int n;

// Input the value of N

printf("Enter the value of N: ");

scanf("%d", &n);

// Calculate and display the Nth Tribonacci number

printf("The %dth Tribonacci number is: %d\n", n, tribonacci(n));

return 0;

}

1. **Binary Search Tree: Insertion**

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

struct node {

int data;

struct node \*left;

struct node \*right;

};

void preOrder( struct node \*root)

{

if( root == NULL )

return;

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

preOrder(root->left);

preOrder(root->right);

}

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

{

// write your code here

return root;

}

int main()

{

struct node\* root = NULL;

int t;

int data;

scanf("%d", &t);

while(t-- > 0) {

scanf("%d", &data);

root = insert(root, data);

}

preOrder(root);

return 0;

}

1. **Remove duplicates from sorted array**

#include <stdio.h>

int removeDup(int arr[], int n)

{

//write your code here

}

int main()

{

int arr[] = {1, 1, 2, 2, 3, 4, 4, 5};

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

n = removeDup(arr, n);

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

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

return 0;

}

1. **Printing Tokens**

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

int main()

{

char s[1001];

scanf("%[^\n]%\*c", s);

//write your code here

return 0;

}

1. **Find the index of first occurrence in a string**

#include <stdio.h>

#include <string.h>

// Function to find the first occurrence of a substring in a string

int findFirstOccurrence(const char \*str, const char \*subStr) {

int strLen = strlen(str);

int subStrLen = strlen(subStr);

// Loop through the main string

//write your code here

}

// If the entire substring matched

if (j == subStrLen) {

return i; // Return the starting index

}

}

return -1; // Substring not found

}

int main() {

char str[100], subStr[50];

// Input main string

printf("Enter the main string: ");

fgets(str, sizeof(str), stdin);

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

// Input substring

printf("Enter the substring: ");

fgets(subStr, sizeof(subStr), stdin);

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

// Find and display the first occurrence

int index = findFirstOccurrence(str, subStr);

if (index != -1) {

printf("The first occurrence of the substring is at index: %d\n", index);

} else {

printf("Substring not found in the main string.\n");

}

return 0;

}