#include <stdio.h> OP.C FILE

#include <math.h>

#include <stdlib.h>

#include <string.h>

#include "practice\_problems.h"

// 1.

char upper\_to\_lower(char ch){

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

return ch + 32; // A = 65, a = 97

else

return '!'; // Invalid input received

}

// 2.

float area\_of\_circle(float radius){

return 3.14 \* radius \* radius; // Pi \* r^2

}

// 3.

int char\_between\_char(char ch\_1, char ch\_2){

if(ch\_1 > ch\_2)

return ch\_1 - ch\_2 - 1;

else

return ch\_2 - ch\_1 - 1;

}

// 4.

float celsius\_to\_fahrenheit(float temperature){

return temperature \* 1.8 + 32; // Multiply by 1.8 (or 9/5) and add 32

}

// 5.

unsigned int odd\_or\_even(int num){

return num % 2; // Return 1 if odd, 0 if even

}

// 6.

int check\_leap\_year(int year){

// Must be exactly divisible by 4

// Century years must also be divisible by 400

// Return 1 if its leap year else return 0

return (year % 400 == 0) || ((year % 4 == 0) && (year % 100 != 0));

}

// 7.

int pow\_of\_2(int raise){

return 1 << raise;

}

// 8.

int char\_or\_int(char ch){

if((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z'))

return 0;

else if(ch >= '0' && ch <= '9')

return 1;

else

return -1;

}

// 9.

unsigned int add\_rand(unsigned int x, unsigned int y){

return x + y;

}

// 10.

unsigned int sum\_of\_digits(unsigned int num){

unsigned int sum = 0;

while(num > 0){

sum += num % 10; // 5, 4, 3, 2, 1

num = num / 10; // 1234, 123, 12, 1

}

return sum;

}

// 11.

unsigned int reverse\_of\_num(unsigned int num){

unsigned int rev = 0;

while(num > 0){

rev = (rev \* 10) + (num % 10);

num = num / 10;

}

return rev;

}

// 12.

unsigned int count\_digit(unsigned int num, unsigned int digit){

unsigned int rem = 0;

unsigned int count = 0;

while(num > 0){

rem = num % 10;

num = num / 10;

if(digit == rem){

count++;

}

}

return count;

}

// 13.

unsigned int check\_palindrome(unsigned int num){

unsigned int rev = 0;

unsigned int temp = num;

while(temp > 0){

rev = (rev \* 10) + (temp % 10);

temp = temp / 10;

}

return rev == num; // Return 1 if palindrome, 0 if not a palindrome

}

// 14.

// Function to check if given number is prime or not

// Return 1 if number is prime else return 0

unsigned int is\_prime(unsigned int num){

if(num < 2){

return 0;

}

for(int k=2; k<= num/2; k++){

if(num % k == 0){

// i is completely divisible by k

return 0;

}

}

return 1;

}

// 15.

unsigned int sum\_of\_series(unsigned int num){

unsigned int count = 1, sum = 0;

for(int i=1; i<=num; i++){

printf("%d ", count);

if(i < num){

printf("+ ");

}

sum = sum + count; // Calculate sum

count = (count\*10)+1; // 1, 11, 111, 1111

}

printf("= %d", sum);

return 0;

}

// 16.

unsigned int check\_if\_armstrong(unsigned int num){

unsigned long int sum\_of\_cubes = 0;

unsigned int rem, temp = num;

while(temp > 0){

rem = temp % 10;

temp = temp / 10;

sum\_of\_cubes += rem \* rem \* rem;

}

// Return 1 if number is Armstrong number

return (num == sum\_of\_cubes);

}

// 17.

unsigned int check\_if\_amicable(unsigned int num\_1, unsigned int num\_2){

unsigned int sum\_1 = 0, sum\_2 = 0;

for(int i=1; i<= num\_1/2; i++){

if(num\_1%i == 0){

sum\_1 = sum\_1 + i;

}

}

for(int j=1; j<= num\_2/2; j++){

if(num\_2%j == 0){

sum\_2 = sum\_2 + j;

}

}

if((num\_1 == sum\_2) && (num\_2 == sum\_1)){

return 1;

}

else{

return 0;

}

}

// 18.

unsigned int menu\_driven\_calculator(unsigned int num\_1, unsigned int num\_2, char operation){

switch(operation){

case '+':

return num\_1 + num\_2;

break;

case '-':

return num\_1 - num\_2;

break;

case '\*':

return num\_1 \* num\_2;

break;

default :

return -1;

break;

}

}

// 19.

float volume\_of\_cube(float length){

return length \* length \* length;

}

float volume\_of\_cuboid(float length, float breadth, float height){

return length \* breadth \* height;

}

float volume\_of\_sphere(float radius){

return 4.188\* radius \* radius \* radius; // 4/3 \* Pi \* r^3

}

float volume\_of\_cylinder(float radius, float height){

return 3.14 \* radius \* radius \* height;

}

float volume\_of\_cone(float radius, float height){

return (3.14 \* radius \* radius \* height)/3;

}

void main\_menu(void){

unsigned int shape;

float dimension\_1, dimension\_2, dimension\_3;

printf("Select the Shape\n");

printf("1 : Cube\n");

printf("2 : Cuboid\n");

printf("3 : Sphere\n");

printf("4 : Cylinder\n");

printf("5 : Cone\n");

scanf("%d", &shape);

switch(shape){

case 1:

printf("Enter the Length: ");

scanf("%f", &dimension\_1);

printf("Volume of Cube: %f", volume\_of\_cube(dimension\_1));

break;

case 2:

printf("Enter the Length: ");

scanf("%f", &dimension\_1);

printf("Enter the Breadth: ");

scanf("%f", &dimension\_2);

printf("Enter the Height: ");

scanf("%f", &dimension\_3);

printf("Volume of Cuboid: %f", volume\_of\_cuboid(dimension\_1, dimension\_2, dimension\_3));

break;

case 3:

printf("Enter the Radius: ");

scanf("%f", &dimension\_1);

printf("Volume of Sphere: %f", volume\_of\_sphere(dimension\_1));

break;

case 4:

printf("Enter the Radius: ");

scanf("%f", &dimension\_1);

printf("Enter the Height: ");

scanf("%f", &dimension\_2);

printf("Volume of Cylinder: %f", volume\_of\_cylinder(dimension\_1, dimension\_2));

break;

case 5:

printf("Enter the Radius: ");

scanf("%f", &dimension\_1);

printf("Enter the Height: ");

scanf("%f", &dimension\_2);

printf("Volume of Cone: %f", volume\_of\_cone(dimension\_1, dimension\_2));

break;

default:

printf("\nInvalid Option Selected");

}

}

// 20.

float electricity\_bill(float units\_consumed){

float rem, amount = 0;

if(units\_consumed > 300){

rem = units\_consumed - 300;

units\_consumed = 300;

amount += rem \* 10;

}

if(units\_consumed > 200){

rem = units\_consumed - 200;

units\_consumed = 200;

amount += rem \* 7;

}

if(units\_consumed >= 0){

amount += units\_consumed \* 5;

}

return amount;

}

// 21.

unsigned int bin\_2\_dec(unsigned long int num){

unsigned int rem, dec = 0, pow = 1; // 2^0 = 1

while(num > 0){

rem = num % 10; // Get the LSB Digit

num = num/10; // Neglect the LSB Digit

dec += rem \* pow;

pow = pow \* 2; // 2^1 = 2, 2^2 = 4, 2^3 = 8 ...

}

return dec;

}

unsigned long int dec\_2\_bin(unsigned int num){

unsigned int rem, bin = 0;

while(num > 0){

rem = num % 2;

num = num / 2;

bin = bin\*10 + rem;

}

// Actual Binary number is reverse of the result

return reverse\_of\_num(bin);

}

// 22.

int generate\_series(int arr[], int size){

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

arr[i+3] = arr[i+0] + arr[i+1] + arr[i+2];

}

return 1; // Operation Success

}

// 23.

void print\_pattern\_1(int n){

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

for(int j=n; j>i; j--){

printf("\*");

}

printf("\n");

}

}

// 24.

void print\_pattern\_2(int n){

int count=0;

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

if(count <= i){

printf("\*");

count++;

}

if(count > i){

printf("\n");

count = 0;

i++;

}

}

}

// 25.

void generate\_series\_2(void){

for(int i=1; i<9; i++){

printf("%d%d ", i, 9-i);

}

}

// 26.

int power(int base, int exponent){

if(exponent == 1)

return base;

else

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

}

// 27.

int factorial(int num){

if(num == 1)

return 1;

else

return num \* factorial(num-1);

}

// 28. F(x) = x + x3/3! + x5/5! + x7/7!+

float series\_evaluation(int x, int n){

if(n == 1){

return x;

}

else{

return ((((float)power (x, ((2\*n)-1))) / (float)factorial((2\*n)-1)) + series\_evaluation(x, n-1));

}

}

// 29.

// Macro defined in .h file

// 30.

// Macro defined in .h file

// 31.

// Using Functions from math.h

/\*

void main\_menu\_math\_operations(void){

int operation;

float x, n;

printf("Select the Required Operation\n");

printf("1 : Natural Log(x)\n");

printf("2 : Log10(x)\n");

printf("3 : Power(x, n)\n");

printf("4 : Sin(x)\n");

printf("5 : Cos(x)\n");

scanf("%d", &operation);

switch(operation){

case 1:

printf("Enter x: ");

scanf("%f", &x);

printf("Natural Log(%f) = %f", x, log(x));

break;

case 2:

printf("Enter x: ");

scanf("%f", &x);

printf("Log10(%f) = %f", x, log10(x));

break;

case 3:

printf("Enter x: ");

scanf("%f", &x);

printf("Enter n: ");

scanf("%f", &n);

printf("(%f)^(%f) = %f", x, n, pow(x, n));

break;

case 4:

printf("Enter x: ");

scanf("%f", &x);

printf("Sin(%f) = %f", x, sin(x));

break;

case 5:

printf("Enter x: ");

scanf("%f", &x);

printf("Cos(%f) = %f", x, cos(x));

break;

default:

printf("\nInvalid Option Selected");

}

}

\*/

// 32.

// Function to print integer array

int print\_int\_array(int arr[], int arr\_size){

for(int i=0; i<arr\_size; i++){

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

}

printf("\n");

return 1; // Operation Successful

}

int ascending\_sort(int arr[], int arr\_size){

int temp;

for(int i=0; i<arr\_size; i++){

for(int j=i+1; j<arr\_size; j++){

if(arr[i] > arr[j]){

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

return 1; // Return Success

}

int descending\_sort(int arr[], int arr\_size){

int temp;

for(int i=0; i<arr\_size; i++){

for(int j=i+1; j<arr\_size; j++){

if(arr[i] < arr[j]){

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

return 1; // Return Success

}

// 33.

void swap\_by\_ref(int \*ptr\_1, int \*ptr\_2){

int temp = \*ptr\_1;

\*ptr\_1 = \*ptr\_2;

\*ptr\_2 = temp;

}

// 34.

int min\_max\_array(int arr[], int arr\_size, int \*min, int \*max){

if(arr\_size == 0){

return -1; // Invalid Array Size

}

\*min = \*max = arr[0];

for(int i=1; i<arr\_size; i++){

if(\*min > arr[i]){

\*min = arr[i];

}

if(\*max < arr[i]){

\*max = arr[i];

}

}

return 1; // Return Success

}

// 35.

// I. A union B (Write function set\_union())

int set\_union(int arr\_a[], int size\_a, int arr\_b[], int size\_b, int arr\_c[], int size\_c){

int already\_added=0;

int count = 0;

// Add Elements of array A to union array without repetition

for(int i=0; i<size\_a; i++){

already\_added = 0;

for(int j=0; j<count; j++){

if(arr\_a[i] == arr\_c[j]){

already\_added = 1;

break;

}

}

if(!already\_added){

arr\_c[count++] = arr\_a[i];

}

}

// Add Elements of array B to union array without repetition

for(int i=0; i<size\_b; i++){

already\_added = 0;

for(int j=0; j<count; j++){

if(arr\_b[i] == arr\_c[j]){

already\_added = 1;

break;

}

}

if(!already\_added){

arr\_c[count++] = arr\_b[i];

}

}

return count; // Return number of elements in array C

}

// II. A intersection B (Write function set\_intersection())

int set\_intersection(int arr\_a[], int size\_a, int arr\_b[], int size\_b, int arr\_c[], int size\_c){

int already\_added=0;

int count = 0;

for(int i=0; i<size\_a; i++){

// For element of array A

for(int j=0; j<size\_b; j++){

// Check against each element of array B

if(arr\_a[i] == arr\_b[j]){

// If element is present in both array A and array B

already\_added = 0;

for(int k=0; k<count; k++){

// Check if element is already added to array C

if(arr\_a[i] == arr\_c[k]){

already\_added = 1;

break;

}

}

if(!already\_added){

arr\_c[count++] = arr\_a[i];

}

}

}

}

return count; // Return number of elements in array C

}

// III. A-B and B-A (Write function set\_difference())

int set\_difference(int arr\_a[], int size\_a, int arr\_b[], int size\_b, int arr\_c[], int size\_c){

int is\_present = 0;

int count = 0;

for(int i=0; i<size\_a; i++){

// For each element of array B

for(int j=0; j<size\_b; j++){

// Check if that element is present in array A

is\_present = 0;

if(arr\_a[i] == arr\_b[j]){

is\_present = 1;

break;

}

}

if(!is\_present){

arr\_c[count++] = arr\_a[i];

}

}

return count;

}

// 36.

int remove\_duplicate\_in\_array(int arr[], int arr\_size){

int count = arr\_size;

int is\_repeated = 0;

int i, j, k;

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

for(j=i+1; j<count; j++){

is\_repeated = 0;

if(arr[i] == arr[j]){

is\_repeated = 1;

break;

}

}

if(is\_repeated){

for(k=j; k<=count-1; k++){

arr[k] = arr[k+1];

}

count--;

i--;

}

}

return count;

}

// 37.

int linear\_search(int arr[], int arr\_size, int element){

for(int i=0; i<arr\_size; i++){

if(arr[i] == element){

return i; // Return Element Index

}

}

return -1; // Return Element not Found

}

// 38.

// Assumption input array is sorted in ascending order

int binary\_search(int arr[], int arr\_size, int element){

int mid, low = 0, high = arr\_size;

while(low <= high){

mid = low + ((high - low) / 2);

// Return index if element is found in middle

if(arr[mid] == element)

return mid;

// If element is greater ignore lower part

if(arr[mid] < element)

low = mid + 1;

// If element is smaller ignore heigher part

if(arr[mid] > element)

high = mid - 1;

}

return -1; // Element not found

}

// 39.

int sum\_of\_product(int arr[], int arr\_size){

int sop = 0;

for(int i=0; i<arr\_size-1; i++){

sop += arr[i] \* arr[i+1];

}

return sop;

}

// 40.

int find\_string\_length(char str[]){

int length = 0;

while(str[length] != '\0'){

length++;

}

return length;

}

// 41.

void worded\_date(int day, int month, int year, char str[]){

char year\_str[5];

sprintf(str, "%d", day);

sprintf(year\_str, "%d", year);

switch(day){

case 1:

case 21:

case 31:

strcat(str, "st");

break;

case 2:

case 22:

strcat(str, "nd");

break;

case 3:

case 23:

strcat(str, "rd");

break;

default:

strcat(str, "th");

break;

}

switch(month){

case 1:

strcat(str, " January ");

break;

case 2:

strcat(str, " February ");

break;

case 3:

strcat(str, " March ");

break;

case 4:

strcat(str, " April ");

break;

case 5:

strcat(str, " May ");

break;

case 6:

strcat(str, " June ");

break;

case 7:

strcat(str, " July ");

break;

case 8:

strcat(str, " August ");

break;

case 9:

strcat(str, " September ");

break;

case 10:

strcat(str, " October ");

break;

case 11:

strcat(str, " November ");

break;

case 12:

strcat(str, " December ");

break;

default:

break;

}

strcat(str, year\_str);

}

// 42.

void lower\_to\_upper(char str[]){

int i=0;

while(str[i] != '\0'){

if(str[i] >= 'a' && str[i] <= 'z'){

str[i] -= 32;

}

i++;

}

}

// 43.

void reverse\_string(char str[]){

int str\_len = find\_string\_length(str)-1;

int len = str\_len/2;

char temp;

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

temp = str[i];

str[i] = str[str\_len - i];

str[str\_len - i] = temp;

}

}

// 44.

int string\_palindrome(char \*str){

int str\_len = find\_string\_length(str)-1;

int i = 0;

while(i <= str\_len){

// Using != in while loop will not work for even length palindrome

if(str[i] != str[str\_len]){

return 0; // Not a palindrome

}

i++;

str\_len--;

}

return 1; // palindrome

}

// 45.

void string\_concat(char \*str\_1, char \*str\_2){

int len\_1 = 0, len\_2 = 0;

// Move to end of string 1

while(str\_1[len\_1] != '\0'){

len\_1++;

}

// Copy string 2 to string 1

while(str\_2[len\_2] != '\0'){

str\_1[len\_1++] = str\_2[len\_2++];

}

// Add null terminator to concatenated string

str\_1[len\_1] = '\0';

}

// 46.

// Create 2D matrix using double pointers

int \*\*create\_2d\_matrix(int rows, int columns){

int \*\*matrix\_ptr;

int i;

// Allocate memory for an array of row pointers

matrix\_ptr = (int \*\*)malloc(sizeof(int \*) \* rows);

// Allocate memory for each row

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

\*(matrix\_ptr + i) = (int \*)malloc(sizeof(int) \* columns);

}

// Return the double pointer to the matrix

return matrix\_ptr;

}

// Initialize 2D matrix

int initialize\_2d\_matrix(int \*\*my\_matrix, int rows, int columns){

int i, j;

// Check if double pointer has valid address

if(my\_matrix == NULL)

return -1; // Return Failure

// Check if each row pointers have valid address

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

if(\*(my\_matrix + i) == NULL)

return -1;

}

// Insert elements

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

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

\*(\*(my\_matrix + i)+j) = rand() % 40;

}

return 1; // Return Success

}

// Print 2D matrix

int print\_2d\_matrix(int \*\*my\_matrix, int rows, int columns){

int i, j;

// Check if double pointer has valid address

if(my\_matrix == NULL)

return -1; // Return Failure

// Check if each row pointers have valid address

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

if(\*(my\_matrix + i) == NULL)

return -1;

}

// Print elements

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

for(j=0; j<columns; j++){

printf("%02d ", \*(\*(my\_matrix + i)+j));

}

printf("\n");

}

return 1; // Return Success

}

// 47.

// Assumption works for square matrix

int transpose\_2d\_matrix(int \*\*my\_matrix, int rows, int columns){

int temp, i, j;

// Check if double pointer has valid address

if(my\_matrix == NULL)

return -1; // Return Failure

// Check if each row pointers have valid address

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

if(\*(my\_matrix + i) == NULL)

return -1;

}

// Run swapping for upper traingular matrix

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

for(j=i+1; j<columns; j++){

if(i != j){

temp = \*(\*(my\_matrix + i)+j);

\*(\*(my\_matrix + i)+j) = \*(\*(my\_matrix + j)+i);

\*(\*(my\_matrix + j)+i) = temp;

}

}

}

return 1; // Return Success

}

// 48.

int print\_column\_sum(int \*\*my\_matrix, int rows, int columns){

int col\_sum, i, j;

// Check if double pointer has valid address

if(my\_matrix == NULL)

return -1; // Return Failure

// Check if each row pointers have valid address

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

if(\*(my\_matrix + i) == NULL)

return -1;

}

for(j=0; j<columns; j++){

col\_sum = 0;

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

col\_sum += \*(\*(my\_matrix + i)+j);

}

printf("%03d ", col\_sum);

}

return 1; // Return Success

}

// 49.

int check\_if\_sparse(int rows, int columns, int my\_matrix[rows][columns]){

int i, j, count = 0;

int valid = (rows \* columns)/2;

// Check if double pointer has valid address

if(my\_matrix == NULL)

return -1; // Return Failure

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

for(j=0; j<columns; j++){

if(my\_matrix[i][j] == 0)

count++;

}

}

return count > valid;

}

/\*

// 50.

// Function to Initialize Array

Student\_Array initialize\_array(int size){

Student\_Array my\_arr;

my\_arr.c\_size = 0;

my\_arr.t\_size = (size > 0 && size <= MAX\_SIZE) ? size : MAX\_SIZE;

return my\_arr;

}

// Function to store data in to array

Student\_Array store\_data(Student\_Array std\_arr, Student std\_data){

if(std\_arr.c\_size == std\_arr.t\_size)

return std\_arr; // Return Failure

std\_arr.std[std\_arr.c\_size++] = std\_data;

return std\_arr; // Return Success

}

// Function to print student array

void print\_data(Student\_Array std\_arr){

Student std;

for(int i=0; i<std\_arr.c\_size; i++){

std = std\_arr.std[i];

printf("Roll No.: %d\n", std.roll\_no);

printf("Name : %s\n", std.name);

printf("Age : %d\n", std.age);

printf("Marks : %f\n\n", std.marks);

}

}

\*/

// 51.

Complex\_Num add\_complex(Complex\_Num a, Complex\_Num b){

Complex\_Num res;

res.real = a.real + b.real;

res.img = a.img + b.img;

return res;

}

Complex\_Num sub\_complex(Complex\_Num a, Complex\_Num b){

Complex\_Num res;

res.real = a.real - b.real;

res.img = a.img - b.img;

return res;

}

// (a+bi)(c+di)=(ac-bd)+(ad+bc)i

Complex\_Num mul\_complex(Complex\_Num a, Complex\_Num b){

Complex\_Num res;

res.real = (a.real\*b.real) - (a.img\*b.img);

res.img = (a.real\*b.img) + (a.img\*b.real);

return res;

}

// 57.

int disp\_file\_content(FILE \*ptr){

char ch;

if(ptr == NULL)

return -1; // Return Failure

do{

ch = fgetc(ptr);

printf("%c", ch);

}while(ch != EOF);

return 1; // Return Success

}

// 58.

int copy\_file\_content(FILE \*dest, FILE \*src){

char ch;

if(dest == NULL)

return -1; // Return Failure

if(src == NULL)

return -1; // Return Failure

// https://stackoverflow.com/questions/31630072/extra-character-at-end-while-copying

while(1){

ch = fgetc(src);

if(ch == EOF)

break;

fputc(ch, dest);

}

return 1; // Return Success

}

// 59.

int char\_count\_in\_file(FILE \*fptr){

char ch;

int char\_count = 0;

int line\_count = 0;

int tab\_count = 0;

int space\_count = 0;

if(fptr == NULL)

return -1; // Return Failure

do{

ch = fgetc(fptr);

if(ch == '\n')

line\_count++;

else if(ch == '\t')

tab\_count++;

else if(ch == ' ')

space\_count++;

else

char\_count++;

}while(ch != EOF);

printf("Character Count: %d\n", char\_count);

printf("Space Count : %d\n", space\_count);

printf("Tab Count : %d\n", tab\_count);

printf("Line Count : %d\n", line\_count);

return 1; // Return Success

}

// 60.

int odd\_even\_sorting(FILE \*num\_ptr, FILE \*evn\_ptr, FILE \*odd\_ptr){

char ch;

int num = 0;

if(num\_ptr == NULL || evn\_ptr == NULL || odd\_ptr == NULL)

return -1; // Return Failure

do{

num = 0;

while(1){

ch = fgetc(num\_ptr);

// Break if end of line or end of file

if(ch == '\n' || ch == EOF)

break;

// Convert to integer from character

num = (num \* 10) + ((int)ch - '0');

}

if(num%2)

fprintf(odd\_ptr, "%d\n", num);

else

fprintf(evn\_ptr, "%d\n", num);

}while(ch != EOF);

return 1; // Return Success

}

// 61.

// Add record

int add\_record(FILE \*fptr, Contact \*contact){

if(fptr == NULL)

return -1; // Return Failure

fwrite(contact, sizeof(Contact), 1, fptr);

return 1; // Return Success

}