1.Pointers

Pointer Arithmetic:

Write a C program to create an integer array of size 5, initialize it with values from 1 to 5, and then use pointer arithmetic to print each element of the array.

#include <stdio.h>

int main() {

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

int \*ptr = arr;

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

printf("Element %d: %d\n", i+1, \*(ptr + i));

}

return 0;

}



Q.2.Pointer to Pointer:

Write a C program to create a pointer to a pointer for an integer variable. Initialize the integer variable with a value, and then print its value using both the single pointer and the pointer to pointer.

#include <stdio.h>

int main() {

int value = 42;

int \*ptr = &value;

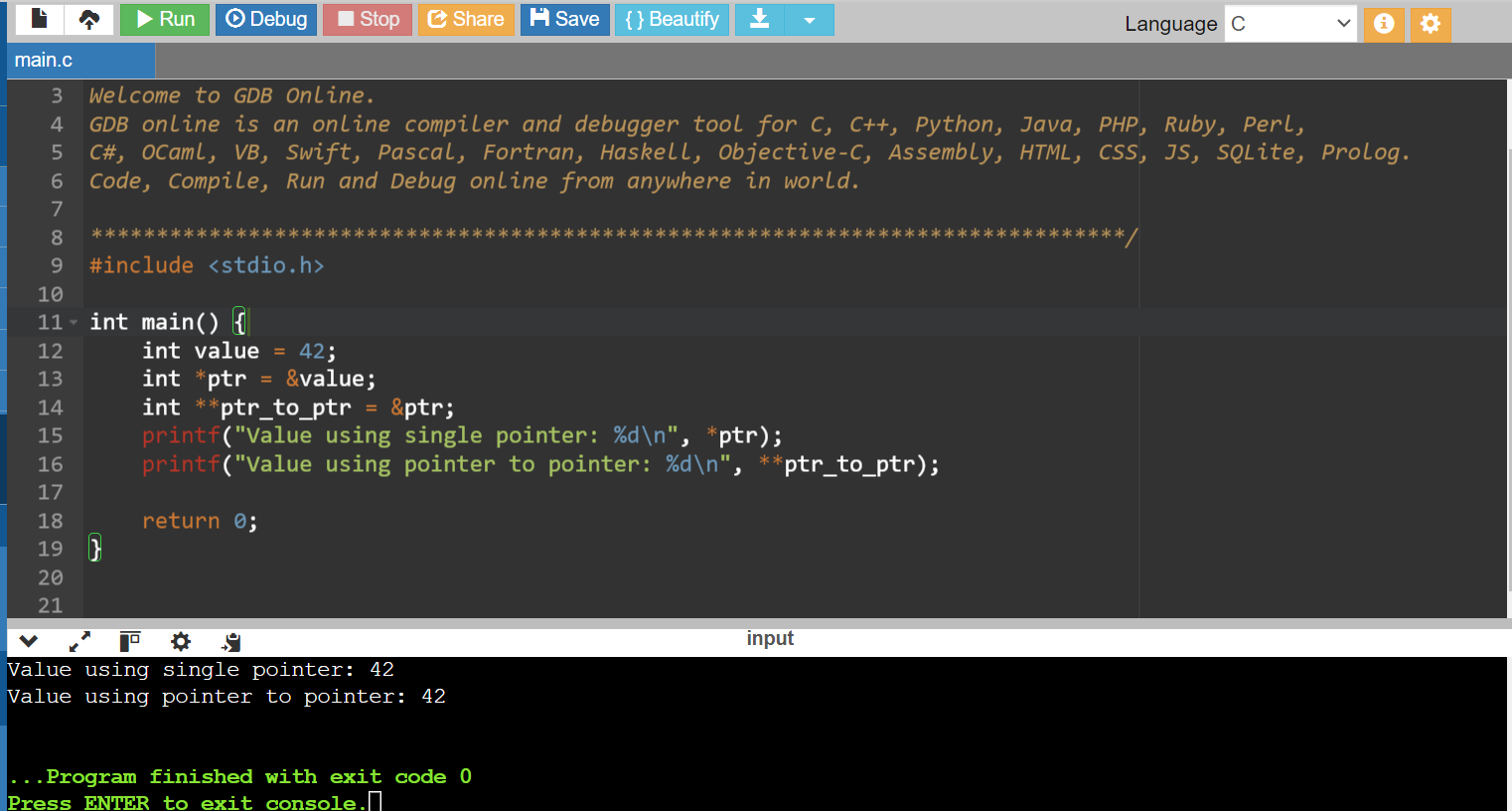
int \*\*ptr\_to\_ptr = &ptr;

printf("Value using single pointer: %d\n", \*ptr);

printf("Value using pointer to pointer: %d\n", \*\*ptr\_to\_ptr);

return 0;

}



3.Pointer Function Parameters:

Write a C function void swap(int \*a, int \*b) that swaps the values of two integers. Then, write a main function to test this swap function using pointer arguments.

#include <stdio.h>

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

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main() {

int x = 10;

int y = 20;

printf("Before swapping:\n");

printf("x = %d, y = %d\n", x, y);

swap(&x, &y);

printf("After swapping:\n");

printf("x = %d, y = %d\n", x, y);

return 0;

}



4.Dynamic Memory Allocation:

Write a C program to dynamically allocate memory for an array of integers of size 10. Initialize the array with values from 1 to 10, then print the values and free the allocated memory.

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*arr;

arr = (int\*)malloc(10 \* sizeof(int));

if (arr == NULL) {

printf("Memory allocation failed\n");

return 1;

}

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

arr[i] = i + 1;

}

printf("Array values:\n");

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

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

}

printf("\n");

free(arr);

return 0;

}



5.Pointer to Function:

Write a C program to create a function pointer that points to a function int add(int, int). Use the function pointer to call the add function and print the result.

#include <stdio.h>

int add(int a, int b) {

return a + b;

}

int main() {

int (\*func\_ptr)(int, int);

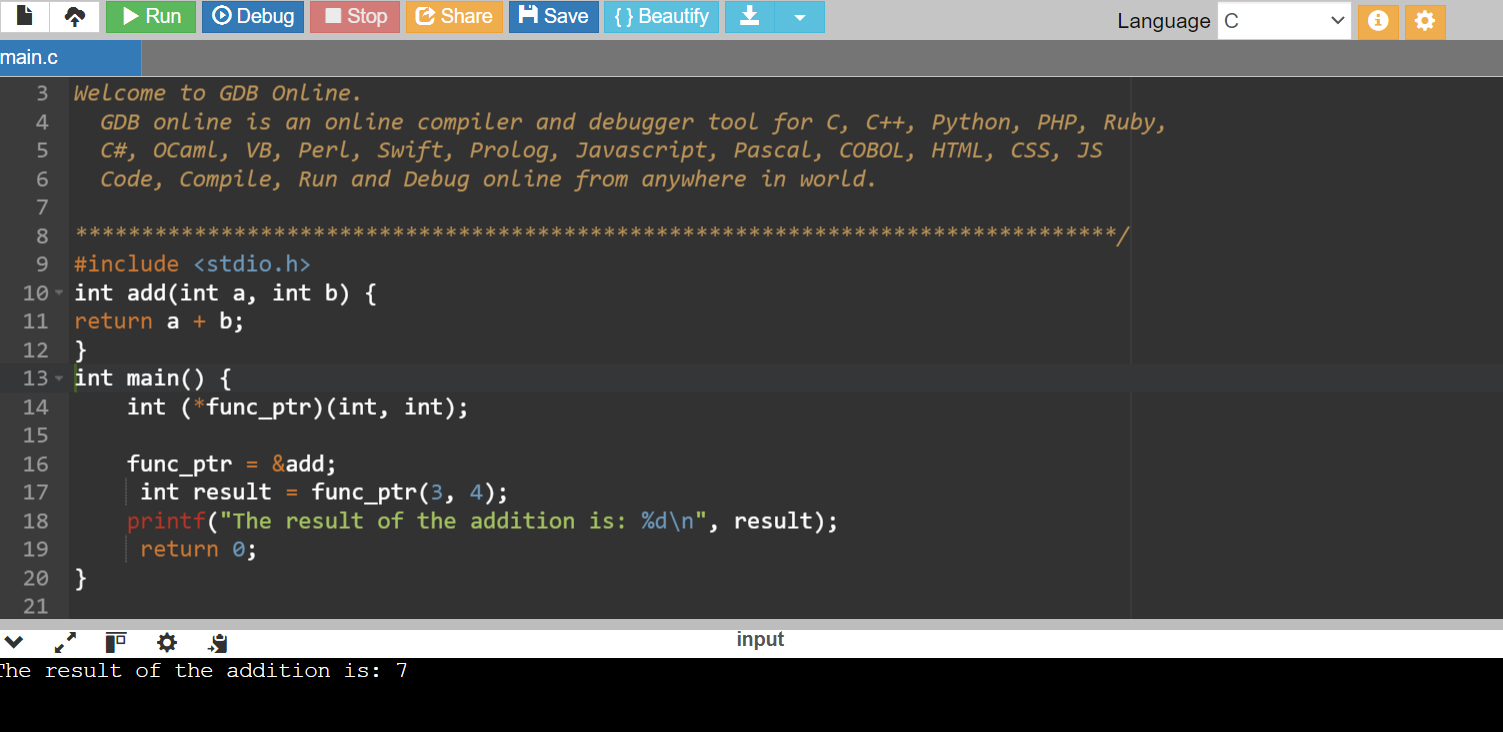
func\_ptr = &add;

int result = func\_ptr(3, 4);

printf("The result of the addition is: %d\n", result);

return 0;

}



Q.6 Functions

Recursive Function:

Write a C function int factorial(int n) that calculates the factorial of a given number using recursion. Test this function in the main program by calculating and printing the factorial of 5.

#include <stdio.h>

int factorial(int n) {

if (n == 0 || n == 1) {

return 1;

}

return n \* factorial(n - 1);

}

int main() {

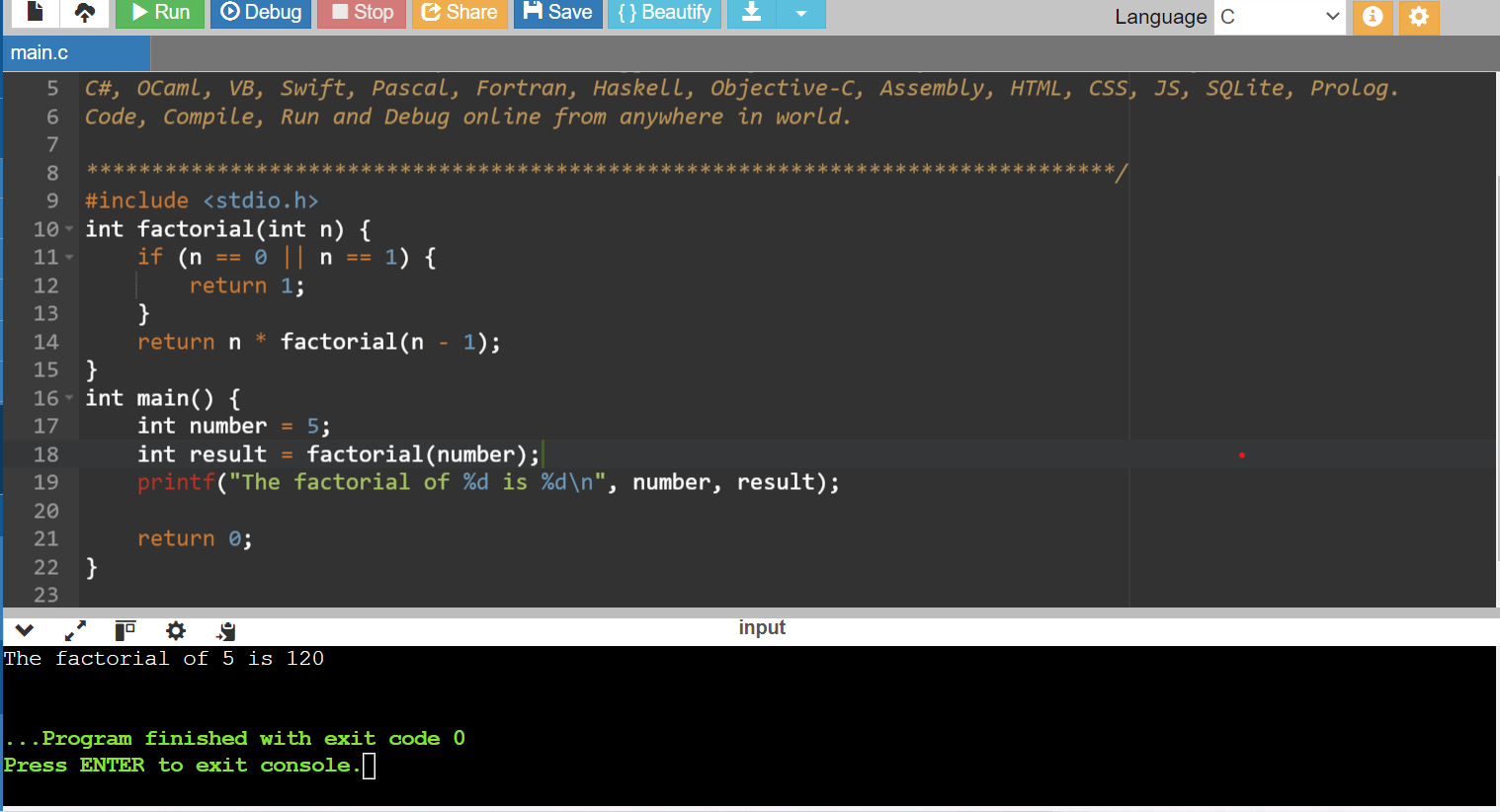
int number = 5;

int result = factorial(number);

printf("The factorial of %d is %d\n", number, result);

return 0;

}



Q.7 Static Variables in Functions:

Write a C function that uses a static variable to count how many times the function has been called. Test this function in the main program by calling it multiple times and printing the count.

#include <stdio.h>

void count\_calls() {

static int count = 0;

count++;

printf("The function has been called %d times\n", count);

}

int main() {

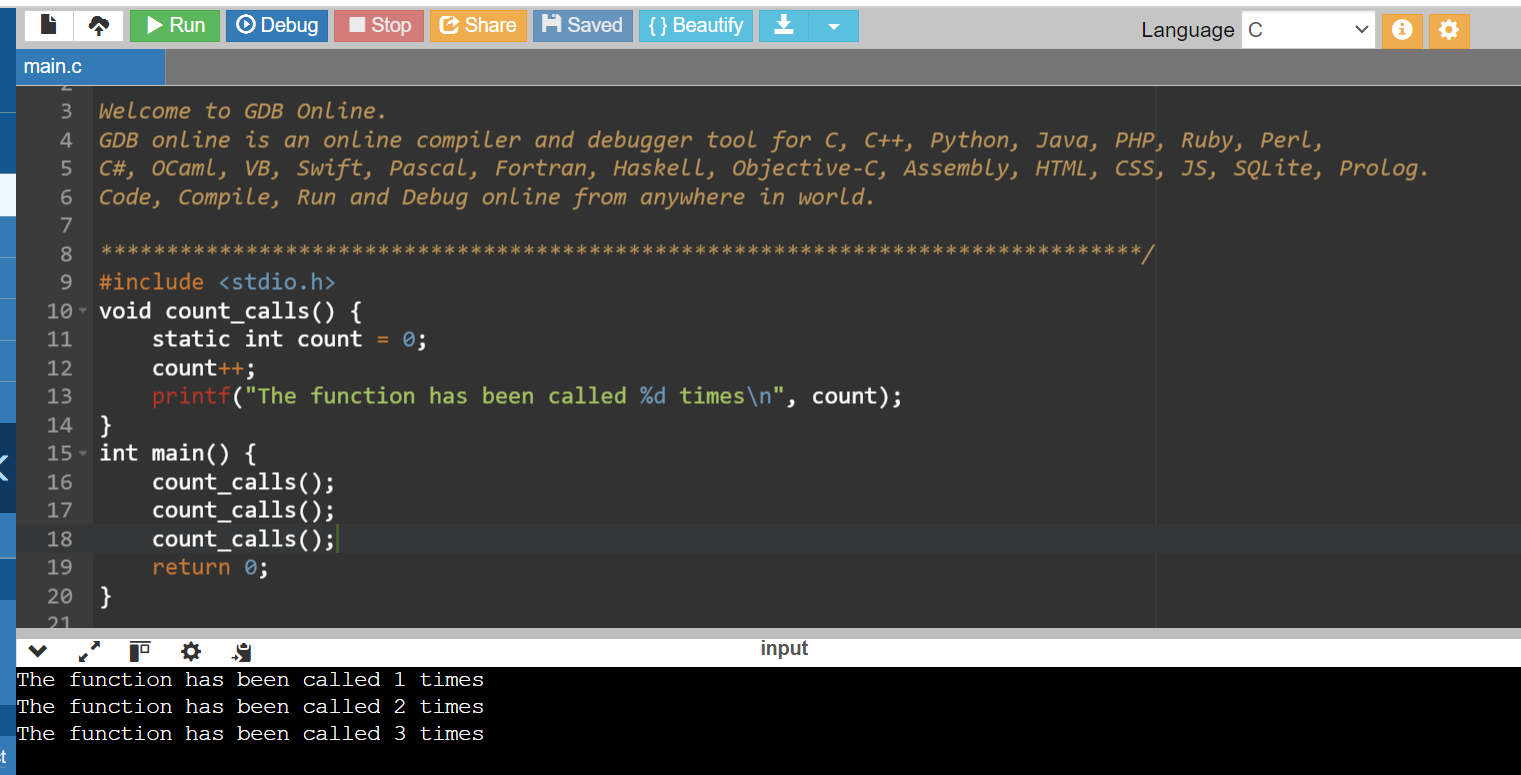
count\_calls();

count\_calls();

count\_calls();

return 0;

}



Q.8 Structure Basics:

Define a structure struct Point with two integer members x and y. Write a C program to create a Point variable, initialize it with values, and print the values.

#include <stdio.h>

struct Point {

int x;

int y;

};

int main() {

struct Point p1;

p1.x = 10;

p1.y = 20;

printf("value of x and y:\n");

printf("x = %d\n", p1.x);

printf("y = %d\n", p1.y);

return 0;

}

