Cprogramstask:

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;

}

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 num = 10;

int \*ptr = &num;

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 = 5;

int y = 10;

// Print the values before swapping

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

// Call the swap function with the addresses of x and y

swap(&x, &y);

// Print the values after swapping

printf("After swapping: 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 = (int \*)malloc(10 \* sizeof(int));

if (arr == NULL) {

printf("Memory allocation failed\n");

return 1; // Exit the program if memory allocation fails

}

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) = &add;

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

printf("Result: %d\n", result);

return 0;

}

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) {

return 1; // Base case: factorial of 0 is 1

} else {

return n \* factorial(n - 1); // Recursive case

}

}

int main() {

int result = factorial(5);

printf("Factorial of 5 is: %d\n", result);

return 0;

}

7. Function Returning Pointer:

Write a C function int\* createArray(int size) that dynamically allocates an array of integers of the given size and returns a pointer to the array. Initialize the array with values from 1 to size and print the array in the main function.

#include <stdio.h>

#include <stdlib.h>

int\* createArray(int size) {

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

if (arr == NULL) {

printf("Memory allocation failed\n");

return NULL; // Return NULL if allocation failed

}

// Step 2: Initialize the array with values from 1 to size

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

arr[i] = i + 1;

}

// Step 3: Return the pointer to the array

return arr;

}

int main() {

// Define the size of the array

int size = 10;

// Call the createArray function and get the pointer to the array

int \*array = createArray(size);

// Check if array creation was successful

if (array != NULL) {

// Print the values of the array

printf("Array values:\n");

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

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

}

printf("\n");

// Free the allocated memory

free(array);

}

return 0;

}

8. Array of Function Pointers:

Write a C program to create an array of function pointers, where each function takes two integers as arguments and returns an integer. Include functions for addition, subtraction, multiplication, and division. Use the array to perform these operations on two integers and print the results

#include <stdio.h>

// Function for addition

int add(int a, int b) {

return a + b;

}

// Function for subtraction

int subtract(int a, int b) {

return a - b;

}

// Function for multiplication

int multiply(int a, int b) {

return a \* b;

}

// Function for division

int divide(int a, int b) {

if (b != 0) {

return a / b;

} else {

printf("Error: Division by zero\n");

return 0; // Return 0 if division by zero

}

}

int main() {

int (\*operations[4])(int, int) = {add, subtract, multiply, divide};

int x = 20;

int y = 4;

printf("Addition: %d + %d = %d\n", x, y, operations[0](x, y));

printf("Subtraction: %d - %d = %d\n", x, y, operations[1](x, y));

printf("Multiplication: %d \* %d = %d\n", x, y, operations[2](x, y));

printf("Division: %d / %d = %d\n", x, y, operations[3](x, y));

return 0;

}

10. 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>

// Function to count how many times it has been called

void countCalls() {

static int count = 0;

count++;

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

}

int main() {

// Call the countCalls function multiple times

countCalls();

countCalls();

countCalls();

countCalls();

return 0;

}

11. Structures

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 p;

p.x = 10;

p.y = 20;

printf("Point p: x = %d, y = %d\n", p.x, p.y);

return 0;

}

12.