**POINTERS**

A pointer is defined as a derived data type that can store the address of other C variables or a memory location.

Ex:int digit=42

**1.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++) {

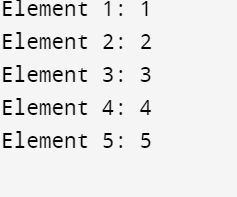
printf("Element %d: %d\n", i + 1, \*(ptr + i));

}

return 0;

}

**OUTPUT:**



**2.POINTER TO POINTER**

To store the address of int variable var, we have the pointer to int ptr\_var. We would need another pointer to store the address of ptr\_var.

int var = 6; int \*ptr\_var = &var;

**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 var = 10;

int \*ptr = &var;

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

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

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

return 0;

}

**OUTPUT:**

Value of var using single pointer: 10

Value of var using pointer to pointer: 10

**3.POINTER FUNCTION PARAMETRS**

Using pointers as function parameters in C allows functions to modify the values of variables that are passed to them. This can be useful for modifying variables directly, passing large structures or arrays efficiently, and more.

**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;

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

swap(&x, &y);

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

return 0;

}

**OUTPUT:**

Before swap: x = 5, y = 10

After swap: x = 10, y = 5

1. **POINTER TO FUNCTION**

A pointer to function or function pointer stores the address of the function. Though it doesn't point to any data. It points to the first instruction in the 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 x = 5;

int y = 10;

int result;

result = func\_ptr(x, y);

printf("The result of adding %d and %d is: %d\n", x, y, result);

return 0;

}

**OUTPUT:**

The result of adding 5 and 10 is: 15

**FUNCTIONS:**

A **function in C**is a set of statements that when called perform some specific task. It is the basic building block of a C program that provides modularity and code reusability.

Syntax of Functions in C:

The syntax of function can be divided into 3 aspects:

Function Declaration

Function Definition

Function Calls

**5.RECURSIVE FUNCTION:**

A recursive function is a function that solves a problem by solving smaller instances of the same problem.

**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;

result = factorial(number);

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

return 0;

}

**OUTPUT:**

The factorial of 5 is: 120

**STRUCTURE**

structure is a collection of elements of different data types.

Declaration of structure

Syntax:

struct struct\_name{

data\_type1 member1;

data\_type2 member2;

data\_typen membern;

};

**6.STRUCTURE BASIC:**

**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("Point p1: x = %d, y = %d\n", p1.x, p1.y);

return 0;

}

**OUTPUT:**

Point p1: x = 10, y = 20

1. **ARRAY OF STRUCTURE:**

An array whose elements are of type structure is called array of structure. It is generally useful when we need multiple structure variables in our program.

**Write a C program to define a structure struct Student with members name, age, and marks. Create an array of 3 students, initialize them with values, and print the details of each student.**

#include <stdio.h>

struct student {

char firstName[50];

int roll;

float marks;

} s[3];

int main() {

int i;

printf("Enter information of students:\n");

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

s[i].roll = i + 1;

printf("\nFor roll number%d,\n", s[i].roll);

printf("Enter first name: ");

scanf("%s", s[i].firstName);

printf("Enter marks: ");

scanf("%f", &s[i].marks);

}

printf("Displaying Information:\n\n");

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

printf("\nRoll number: %d\n", i + 1);

printf("First name: ");

puts(s[i].firstName);

printf("Marks: %.1f", s[i].marks);

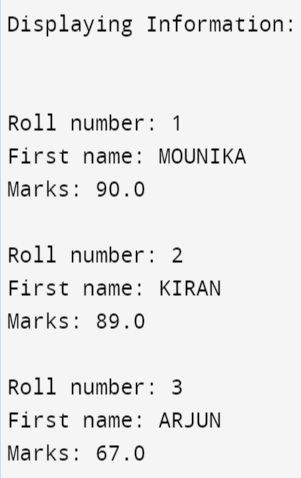
printf("\n");

}

return 0;

}

**OUTPUT**



1. **STRUCTURE POINTERS:**

A structure pointer is defined as the [pointer](https://www.geeksforgeeks.org/pointers-in-c-and-c-set-1-introduction-arithmetic-and-array/) which points to the address of the memory block that stores a [structure](https://www.geeksforgeeks.org/structures-c/) known as the structure pointer. Complex data structures like Linked lists, trees, graphs, etc. are created with the help of structure pointers. The structure pointer tells the address of a structure in memory by pointing the variable to the structure variable.

**Write a C program to define a structure struct Rectangle with members length and width. Create a pointer to a Rectangle variable, dynamically allocate memory for it, initialize the members, and print the values.**

#include <stdio.h>

#include <stdlib.h>

struct Rectangle {

float length;

float width;

};

int main() {

struct Rectangle \*ptrRect;

ptrRect = (struct Rectangle \*) malloc(sizeof(struct Rectangle));

if (ptrRect == NULL) {

printf("Memory allocation failed. Exiting...");

return 1;

}

ptrRect->length = 10.5;

ptrRect->width = 5.7;

printf("Rectangle Details:\n");

printf("Length: %.2f\n", ptrRect->length);

printf("Width: %.2f\n", ptrRect->width);

free(ptrRect);

return 0;

}

**OUTPUT**

Rectangle Details:

Length: 10.50

Width: 5.70