**Lab: Function Pointers and Their Applications in C**

**Objective:**

This lab will introduce function pointers, arrays of function pointers, functions returning pointers to arrays, calling functions via pointers, and passing functions as arguments.

**1. Definitions and Initializations of Function Pointers**

**Concept:**

* A function pointer stores the address of a function and can be used to call it.
* Syntax:

c

return\_type (\*pointer\_name)(parameter\_list);

**Task:**

* Declare and initialize a function pointer.

c

#include <stdio.h>

// Function prototype

int add(int a, int b);

int main() {

int (\*func\_ptr)(int, int); // Function pointer declaration

func\_ptr = add; // Assign function address

printf("Sum: %d\n", func\_ptr(5, 10)); // Call function via pointer

return 0;

}

int add(int a, int b) {

return a + b;

}

**2. Arrays of Pointers to Functions**

**Concept:**

* An array of function pointers allows selecting different functions dynamically.
* Useful for implementing menus and callbacks.

**Task:**

* Create an array of function pointers and call different functions.

c

#include <stdio.h>

// Function prototypes

int add(int a, int b);

int subtract(int a, int b);

int multiply(int a, int b);

int main() {

// Array of function pointers

int (\*operations[3])(int, int) = {add, subtract, multiply};

int choice, a = 10, b = 5;

printf("Choose operation: 0-Add, 1-Subtract, 2-Multiply: ");

scanf("%d", &choice);

if (choice >= 0 && choice < 3) {

printf("Result: %d\n", operations[choice](a, b)); // Call selected function

} else {

printf("Invalid choice\n");

}

return 0;

}

int add(int a, int b) { return a + b; }

int subtract(int a, int b) { return a - b; }

int multiply(int a, int b) { return a \* b; }

**3. Functions Returning Pointers to Arrays**

**Concept:**

* A function can return a pointer to an array, usually allocated dynamically.
* This is useful for returning collections of data.

**Task:**

* Implement a function returning a pointer to an array.

c

#include <stdio.h>

#include <stdlib.h>

// Function returning pointer to an array

int\* generateArray(int size) {

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

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

arr[i] = i \* 10;

}

return arr;

}

int main() {

int size = 5;

int \*arr = generateArray(size);

printf("Generated array: ");

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

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

}

free(arr); // Free allocated memory

return 0;

}

**4. Calling Functions Using Function Pointers**

**Concept:**

* Instead of calling functions directly, function pointers can be used.
* Enables flexible function selection.

**Task:**

* Use a function pointer to call a function.

c

#include <stdio.h>

void greet() {

printf("Hello, welcome to function pointers!\n");

}

int main() {

void (\*func\_ptr)(); // Function pointer declaration

func\_ptr = greet; // Assign address of greet function

func\_ptr(); // Call function via pointer

return 0;

}

**5. Passing Functions as Arguments to Another Function**

**Concept:**

* Higher-order functions allow passing a function as an argument.
* Useful for sorting, filtering, or callbacks.

**Task:**

* Implement a function that accepts another function as an argument.

c

#include <stdio.h>

// Function to operate on two numbers

int operate(int x, int y, int (\*func)(int, int)) {

return func(x, y); // Call function passed as argument

}

// Functions to pass

int add(int a, int b) { return a + b; }

int multiply(int a, int b) { return a \* b; }

int main() {

printf("Addition: %d\n", operate(5, 3, add));

printf("Multiplication: %d\n", operate(5, 3, multiply));

return 0;

}

**Lab Questions**

1. What is a function pointer and how is it different from a normal pointer?
2. How does an array of function pointers work?
3. How can a function return a pointer to an array?
4. Why is it useful to pass a function as an argument to another function?
5. What happens if we try to dereference a function pointer before assigning it a valid function?