**TRINITY INTERNATIONAL SS & COLLEGE**

**Dillibazar Height, Kathmandu, Nepal**

****

**LAB WORK #3: C-Programming (Pointers)**

**(COMPUTER SCIENCE)**

**SUBMITTED BY: SUBMITTED TO:**

**NAME: Prashim Timsina**

**GRADE: XII (MA2)**

**DATE : [2023/08/18] DAYA RAM YADAV**

**Faculty of Computer Science**

**KATHMANDU, NEPAL**

**2023 Table of Contents**

S. No. Page No.

1. Table of Contents 1

2. Objectives 2

3. Theoretical Background 2

3.1 C-Programming

3.1.1 Pointers

4. Work Done

5. Conclusion

Page: 1

**2. Objective**

The main objectives of the lab work are as follows:

1. To become familiar with the concept of a pointer.
2. To understand that variables are stored in memory locations that can be referred to by their addresses.
3. To understand pointer and array.
4. To understand passing pointer as function argument.

**3. Theoretical Background**

In the realm of C programming, pointers are dynamic tools that hold the memory address of variables, offering direct memory access and dynamic memory management. Declared using the data type followed by an asterisk (\*), like int \*ptr;, pointers allow interaction with memory locations.

Syntax and Purpose:

Pointers are initialized by assigning them the memory address of a variable using the address-of operator (&). The dereference operator (\*) retrieves the value stored at the memory location pointed to by the pointer. Pointers find use in dynamic memory allocation through functions like malloc(), facilitating flexible data structures. They also play a crucial role in passing data by reference to functions, enabling modifications outside the function's scope.

Dynamic Memory Allocation and Arrays:

Pointers are essential for dynamic memory allocation using functions like malloc(), as well as for navigating arrays through pointer arithmetic. Proper usage of pointers enhances code efficiency and memory management.

Null Pointers and Error Handling:

Utilizing null pointers (NULL or 0) is crucial for error handling, preventing crashes caused by uninitialized pointers. Their correct usage contributes to robust and reliable code.

Page: 2

Page: 2

Q1. WAP to perform arithmetic calculations (sum,difference, multiplication and division) of two numbers using pointers.

|  |
| --- |
| Source Code |

Q2. WAP to know a number is even or odd using pointer.

|  |
| --- |
| Source Code |

Q3. WAP to find sum and average of ‘n’ natural numbers using pointer.

|  |
| --- |
| Source Code |

Q4. Use array to input 10 elements and print them.Use array as pointer.

|  |
| --- |
| Source Code |

Q5. WAP to input 10 elements and print maximum and minimum value. Use array as pointer.

|  |
| --- |
| Source Code |

Q6. WAP to swap two values using call by reference and call by value.

|  |
| --- |
| Source Code |

Q7. WAP to sort 10 numbers stored in an array using pointer.

|  |
| --- |
| Source Code |

Q8. WAP to print multiplication table of a number using pointer.

|  |
| --- |
| Source Code |

**5. Conclusion**

Concluding our lab work centered around pointers in C has been an illuminating journey into the core of memory management and dynamic data manipulation. Navigating the syntax and intricacies of pointers has empowered us to directly access memory locations, efficiently manage dynamic memory, and create flexible data structures. As we wrap up this exploration, we emerge with a heightened appreciation for the precision and power of pointers, understanding their pivotal role in optimizing code efficiency and enabling advanced programming techniques. Our newfound mastery of pointers equips us to craft more sophisticated, memory-efficient, and dynamic programs in the realm of C programming.