

EXPERIMENT 3

Aim:

1. Write a program in C to implement arrays of pointers and pointers to arrays.
2. Write a program in C to implement pointers to structures.
3. Write a program in C to perform swapping of two numbers by passing addresses of the variables to the functions.

Theory (Arrays of Pointers and Pointers to Arrays):

- An array of pointers is a collection where each element is a pointer to a variable rather than a direct value. In this scenario, instead of storing actual values, the array stores addresses of variables. For example, if we have several integer variables and want to create an array that holds the addresses of these variables, we can use an array of pointers. This allows us to access and modify the values of the original variables indirectly by dereferencing the pointers stored in the array. This approach is particularly useful when dealing with dynamic data structures or when we need to manage multiple variables efficiently.
- A pointer to an array is a pointer that points to the first element of an array. By using this pointer, we can traverse and access elements of the array through pointer arithmetic. Instead of directly accessing the array elements by their index, we can use the pointer to move across the array. This method is beneficial when passing arrays to functions, as it allows the function to work with the original array without needing to copy it. It also simplifies operations on arrays, especially when dealing with large data sets or when the array needs to be modified.

Program:

```
#include <stdio.h>

// * - "value at" operator
// & - "address of" operator

int main(void)
{
    int arr[10] = {11,23,32,45,54,63,72,81,90,10};

    int *ptrtoarr = &arr[0];
    int arrlen = sizeof(arr) / sizeof(arr[0]);
    for (int i = 0; i < arrlen; i++)
    {
        printf("%d element: %d\n", i + 1, *(ptrtoarr + i));
    }

    printf("\n");

    int a = 112;
    int b = 123;
    int c = 34;
    int d = 234;
    int e = 123;

    int *arrofptr[5] = {&a, &b, &c, &d, &e};
```

```

int aoplen = sizeof(arrofptr) / sizeof(arrofptr[0]);
for (int i = 0; i < aoplen; i++)
{
    printf("%d element: %d\n", i + 1, *arrofptr[i]);
}
}

```

Output:

```

PS B:\sem3\ds\23bcp153_dsa\lab3> ./aoppoa
1 element: 11
2 element: 23
3 element: 32
4 element: 45
5 element: 54
6 element: 63
7 element: 72
8 element: 81
9 element: 90
10 element: 10

1 element: 112
2 element: 123
3 element: 34
4 element: 234
5 element: 123

```

Theory (Pointers to Structures):

In C, pointers to structures allow us to efficiently manage and manipulate data stored within structures. A structure is a user-defined data type that groups different data types under a single name, making it easier to manage related data. When we use pointers with structures, we can directly access and modify the members of the structure through the pointer, which holds the address of the structure.

In the code below, a structure named Student is defined, which contains the student's name, roll number, and CGPA. A pointer to the structure ptr is declared and assigned the address of the structure variable s1. Using this pointer, the members of the structure can be accessed using the -> operator. This approach allows us to work with the structure efficiently, especially when passing it to functions or working with dynamic data.

Program:

```

#include <stdio.h>
#include <string.h>

```

```

typedef struct Student {
    char name[30];
    int roll_no;
    float cgpa;
}

```

Student;

```
int main(void)
{
    Student s1;

    Student *ptr;

    ptr = &s1;

    printf("Enter Student's Name: ");
    fgets(ptr->name, sizeof(ptr->name), stdin);
    ptr->name[strcspn(ptr->name, "\n")] = 0;

    printf("Enter Student's Roll No.: ");
    scanf("%d", &ptr->roll_no);

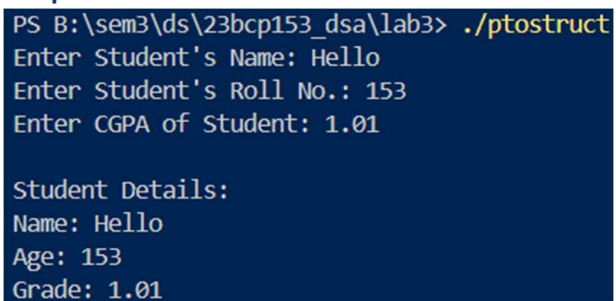
    printf("Enter CGPA of Student: ");
    scanf("%f", &ptr->cgpa);

    printf("\nStudent Details:\n");
    printf("Name: %s\n", ptr->name);
    printf("Age: %d\n", ptr->roll_no);
    printf("Grade: %.2f\n", ptr->cgpa);

    return 0;
}
```

// <https://stackoverflow.com/questions/2693776/removing-trailing-newline-character-from-fgets-input>

Output:



```
PS B:\sem3\ds\23bcp153_dsa\lab3> ./ptostruct
Enter Student's Name: Hello
Enter Student's Roll No.: 153
Enter CGPA of Student: 1.01

Student Details:
Name: Hello
Age: 153
Grade: 1.01
```

Theory (Swapping by Addresses):

In C, swapping two numbers by passing their addresses to a function is an example of call by reference. When we pass the addresses of variables to a function, the function can directly access and modify the values stored at those addresses. This is different from call by value, where a copy of the variable is passed, and changes made within the function do not affect the original variables.

In the code below, the swap function takes two pointers as parameters, which hold the addresses of the variables a and b. Inside the function, the values at these addresses are swapped using a temporary variable temp. This operation changes the actual values of a and b in the main function, demonstrating the effectiveness of call by reference in such scenarios.

Program:

```
#include <stdio.h>

void swap(int *x, int *y);

int main(void)
{
    int a = 1;
    int b = 2;

    printf("Value of a is %d and value of b is %d\n", a, b);

    swap(&a, &b);

    printf("Value of a is %d and value of b is %d\n", a, b);
    return 0;
}

void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;

    return;
}
```

Output:

```
PS B:\sem3\ds\23bcp153_dsa\lab3> ./swapint
Value of a is 1 and value of b is 2
Value of a is 2 and value of b is 1
```