

Data Structures and Algorithms

CIS-318

Lab Report no.1



Spring 2023

Obtained Marks	
Total Marks	
Lab Engineer Signature & Comments	

Student Name

1. Anas Zohrab

Section: A (Electronics)	
Experiment No: 01	Date of Submission: Feburary-20-2023
Experiment Title: Introduction to pointers and structures in C	
Batch: BSEE 2019-23	Teacher: Dr. Kamran Safdar
Semester 7th	Lab Engineer: Mr. Ghaznooq Ahmad

1.1 Title: Introduction to pointers and structures in C

1.2 Introduction:

In programming, a pointer is a variable that stores the memory address of another variable. Pointers allow you to work with memory directly and manipulate data in ways that would not be possible otherwise.

Here is an example in C programming:

```
int a = 5; // create a variable 'a' and assign it the value 5
int *p;   // create a pointer 'p' that points to an integer

p = &a;   // assign the address of variable 'a' to the pointer 'p'

printf("%d", *p); // dereference the pointer to get the value stored at the address it points to
```

In this example, the * operator is used to dereference the pointer and get the value stored at the address it points to. The output of this code would be 5, which is the value stored in variable a. Pointers are useful for various tasks such as dynamic memory allocation, accessing array elements, and passing variables by reference to functions. However, they can also be a source of errors such as null pointer exceptions and memory leaks if not used carefully.

here are some common pointer symbols and their definitions:

- *(asterisk): Used to declare a pointer variable and to dereference a pointer variable to access the value it points to. Example: `int *ptr;` declares a pointer to an integer, and `*ptr` would be used to access the value it points to.
- & (ampersand): Used to get the memory address of a variable. Example: `int x = 10; int *ptr = &x;` declares a pointer to an integer and assigns it the memory address of x.
- -> (arrow): Used to access the member of a structure or class that is pointed to by a pointer. Example: `struct Person { int age; char *name; };` defines a struct with age and name members, and `Person *ptr = &person;` declares a pointer to a Person struct. To access the name member through the pointer, you can use `ptr->name`.

1.3 Objectives:

1. Understanding of syntax of C syntax.
2. Understanding pointers and structures.
3. Solving problems using pointers and structures.

1.4 Code:

1. Code:

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

// Define the Employee struct
typedef struct {
    char name[100];
    float taxRate;
    float (*calcSalary)(void*);
} Employee;
```

```
// Define the SalariedEmployee struct that inherits from the Employee struct
typedef struct {
    Employee employee;
    float salary;
} SalariedEmployee;

// Define the HourlyEmployee struct that inherits from the Employee struct
typedef struct {
    Employee employee;
    float hours;
    float hourlyRate;
} HourlyEmployee;

// Define the CommissionedEmployee struct that inherits from the Employee struct
typedef struct {
    Employee employee;
    float sales;
    float commissionRate;
} CommissionedEmployee;

// Define the calcSalary_SalariedEmployee function
float calcSalary_SalariedEmployee(void* var) {
    SalariedEmployee* e = (SalariedEmployee*)var;
    return e->salary * (1 - e->employee.taxRate);
}

// Define the calcSalary_HourlyEmployee function
float calcSalary_HourlyEmployee(void* var) {
    HourlyEmployee* e = (HourlyEmployee*)var;
    return e->hours * e->hourlyRate * (1 - e->employee.taxRate);
}

// Define the calcSalary_CommissionedEmployee function
float calcSalary_CommissionedEmployee(void* var) {
    CommissionedEmployee* e = (CommissionedEmployee*)var;
    return e->sales * e->commissionRate * (1 - e->employee.taxRate);
}

int main() {
    int i;
    // Create a SalariedEmployee object
    SalariedEmployee salariedEmployee = {
        { "Talha", 0.1, calcSalary_SalariedEmployee },
        5000
    };

    // Create an HourlyEmployee object
    HourlyEmployee hourlyEmployee = {
        { "Ali", 0.15, calcSalary_HourlyEmployee },
        40,
        20
    };
}
```

```

// Create a CommissionedEmployee object
CommissionedEmployee commissionedEmployee = {
    { "Asad", 0.2, calcSalary_CommissionedEmployee },
    10000,
    0.05
};

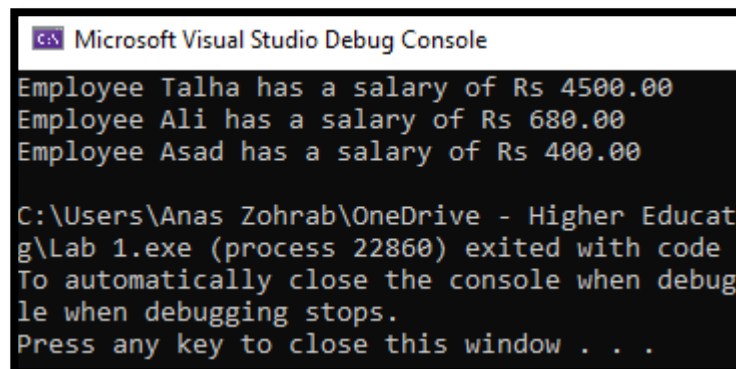
// Create an array of pointers to Employee objects
Employee* employees[] = {
    (Employee*)&salariedEmployee,
    (Employee*)&hourlyEmployee,
    (Employee*)&commissionedEmployee
};

// Loop through the array of Employee objects
for (i = 0; i < 3; i++) {
    Employee* e = employees[i];
    float salary = e->calcSalary(e);
    printf("Employee %s has a salary of Rs %.2f\n", e->name, salary);
}

return 0;
}

```

2. Result:



```

Microsoft Visual Studio Debug Console
Employee Talha has a salary of Rs 4500.00
Employee Ali has a salary of Rs 680.00
Employee Asad has a salary of Rs 400.00

C:\Users\Anas Zohrab\OneDrive - Higher Education\Lab 1.exe (process 22860) exited with code 0
To automatically close the console when debugging stops,
press Ctrl+Shift+D.
Press any key to close this window . . .

```

Figure 1 Output for the code

1.5 Discussion & Conclusion:

In this lab, we were introduced to the concept of pointers and structures in the C programming language. Pointers are variables that store the memory address of another variable as its value. We learned how to create pointers in C using the `*` operator and assign the address of the variable to the pointer. We also learned how to use the reference operator (`&`) to access the memory address of a variable. We also learned about structures, which are a way to group related variables into one place. Each variable in the structure is known as a member of the structure. Unlike an array, a structure can contain many different data types such as `int`, `float`, `char`, etc. We learned how to create structures in C and initialize their members.

The lab exercise involved developing a simple payroll application for a company with three types of employees: salaried, hourly, and commissioned. We had to create four structures, `Employee`, `Salaried Employee`, `Hourly Employee`, and `Commissioned Employee`, each

inheriting from the Employee structure and having their own set of data variables. We also had to create three functions for calculating the salary for each type of employee. In conclusion, this lab helped us to gain a better understanding of the syntax of C and the concept of pointers and structures. By implementing the payroll application, we were able to apply these concepts in a practical way. The exercise also helped us to improve our problem-solving skills as we had to create different structures and functions to calculate the salaries of the different types of employees.

1.6 References:

- GeeksforGeeks. (2022). Structures in C Programming. Retrieved February 17, 2023, from <https://www.geeksforgeeks.org/structures-c/>
- Tutorials Point. (n.d.). C Programming - Structures. Retrieved February 17, 2023, from https://www.tutorialspoint.com/cprogramming/c_structures.html
- W3Schools. (n.d.). C Pointers. Retrieved February 17, 2023, from https://www.w3schools.com/c/c_pointers.php