Lab # 11 To become familiar with the Structures

Objectives

After performing this lab, students will be able to

- Understand structure and use them in programs.

Structures:

Structure is a collection of variables of different data types under a single name. It is similar to a class in that, both hold a collection of data of different data types.

For example: You want to store some information about a person: his/her name, citizenship number and salary. You can easily create different variables name, citNo, salary to store this information separately.

However, in the future, you would want to store information about multiple persons. Now, you'd need to create different variables for each information per person: name1, citNo1, salary1, name2, citNo2, salary2.

You can easily visualize how big and messy the code would look. Also, since no relation between the variables (information) would exist, it's going to be a daunting task.

A better approach will be to have a collection of all related information under a single name Person, and use it for every person. Now, the code looks much cleaner, readable and efficient as well.

This collection of all related information under a single name Person is a structure.

How to declare a structure in C++ programming?

The **struct** keyword defines a structure type followed by an **identifier** (name of the structure).

Then inside the curly braces, you can declare one or more members (declare variables inside curly braces) of that structure.

For example:

```
struct Person
{
   char name[50];
   int age;
   float salary;
};
```

Here a structure person is defined which has three members: name, age and salary.

When a structure is created, no memory is allocated.

The structure definition is only the blueprint for the creating of variables. You can imagine it as a datatype. When you define an integer as below:

```
int abc;
```

The int specifies that, variable abc can hold integer element only. Similarly, structure definition only specifies that, what property a structure variable holds when it is defined.

Note: Remember to end the declaration with a semicolon (;)

How to define a structure variable?
Once you declare a structure person as above. You can define a structure variable as:

```
Person bill;
```

Here, a structure variable bill is defined which is of type structure Person.

When structure variable is defined, only then the required memory is allocated by the compiler.

Considering you have either 32-bit or 64-bit system, the memory of float is 4 bytes, memory of int is 4 bytes and memory of char is 1 byte.

Hence, 58 bytes of memory is allocated for structure variable bill.

How to access members of a structure?
The members of structure variable is accessed using a dot (.) operator.

Suppose, you want to access age of structure variable bill and assign it 50 to it. You can perform this task by using following code below:

```
bill.age = 50;
```

Example: C++ Structure
C++ Program to assign data to members of a structure variable and display it.

```
#include <iostream>
using namespace std;

struct Person
{
   char name[50];
   int age;
   float salary;
};

int main()
```

```
{
   Person p1;

   cout << "Enter Full name: ";
   cin.get(p1.name, 50);
   cout << "Enter age: ";
   cin >> p1.age;
   cout << "Enter salary: ";
   cin >> p1.salary;

   cout << "\nDisplaying Information." << endl;
   cout << "Name: " << p1.name << endl;
   cout <<"Age: " << p1.age << endl;
   cout << "Salary: " << p1.salary;

   return 0;
}
```

Output

```
Enter Full name: Zainab Umair
Enter age: 30
Enter salary: 1024.4

Displaying Information.
Name: Zainab Umair
Age: 31
Salary: 1024.4
```

Here a structure Person is declared which has three members name, age and salary.

Inside main() function, a structure variable p1 is defined. Then, the user is asked to enter information and data entered by user is displayed.

C++ Structure and Function
Structure variables can be passed to a function and returned in a similar way as normal arguments.

Passing structure to function in C++
A structure variable can be passed to a function in similar way as normal argument. Consider this example:

Example 1: C++ Structure and Function

```
#include <iostream>
using namespace std;

struct Person {
    char name[50];
    int age;
    float salary;
};

void displayData(Person);   // Function declaration

int main() {
    Person p;

    cout << "Enter Full name: ";
    cin.get(p.name, 50);
    cout << "Enter age: ";
    cin >> p.age;
    cout << "Enter salary: ";
    cin >> p.salary;

    // Function call with structure variable as an argument
    displayData(p);

    return 0;
}

void displayData(Person p) {
    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p.name << endl;
    cout <<"Age: " << p.age << endl;
    cout << "Salary: " << p.salary;
}
```

Output

```
Enter Full name: Umair Ayaz
Enter age: 31
Enter salary: 34233.4

Displaying Information.
Name: Umair Ayaz
Age: 31
Salary: 34233.4
```

In this program, user is asked to enter the name, age and salary of a Person inside main() function.

Then, the structure variable p is to passed to a function using.

```
displayData(p);
```

The return type of displayData() is void and a single argument of type structure Person is passed.

Then the members of structure p is displayed from this function.

Example 2: Returning structure from function in C++

```
#include <iostream>
using namespace std;

struct Person {
   char name[50];
   int age;
   float salary;
};

Person getData(Person);
void displayData(Person);

int main() {

   Person p, temp;

   temp = getData(p);
   p = temp;
   displayData(p);

   return 0;
}

Person getData(Person p) {

   cout << "Enter Full name: ";
   cin.get(p.name, 50);

   cout << "Enter age: ";
   cin >> p.age;

   cout << "Enter salary: ";
   cin >> p.salary;

   return p;
}

void displayData(Person p) {
   cout << "\nDisplaying Information." << endl;
   cout << "Name: " << p.name << endl;
   cout <<"Age: " << p.age << endl;
```

```
    cout << "Salary: " << p.salary;
}
```

Run Code
The output of this program is the same as the program above.
In this program, we have created two structure variables p and temp of type Person under the main() function.

The structure variable p is passed to getData() function which takes input from the user which is then stored in the temp variable.

```
temp = getData(p);
```

We then assign the value of temp to p.

```
p = temp;
```

Then the structure variable p is passed to displayData() function, which displays the information.

Note: We don't really need to use the temp variable for most compilers and C++ versions. Instead, we can simply use the following code:

```
p = getData(p);
```

C++ Pointers to Structure
A pointer variable can be created not only for native types like (int, float, double etc.) but they can also be created for user defined types like structure.

Here is how you can create pointer for structures:

```
#include <iostream>
using namespace std;

struct temp {
    int i;
    float f;
};

int main() {
    temp *ptr;
    return 0;
}
```

This program creates a pointer ptr of type structure temp.

Example: Pointers to Structure

```
#include <iostream>
using namespace std;
```

```
struct Distance {
   int feet;
   float inch;
};

int main() {
   Distance *ptr, d;

   ptr = &d;

   cout << "Enter feet: ";
   cin >> (*ptr).feet;
   cout << "Enter inch: ";
   cin >> (*ptr).inch;

   cout << "Displaying information." << endl;
   cout << "Distance = " << (*ptr).feet << " feet " << (*ptr).inch << " inches";

   return 0;
}
```

Output

```
Enter feet: 4
Enter inch: 3.5
Displaying information.
Distance = 4 feet 3.5 inches
```

In this program, a pointer variable ptr and normal variable d of type structure Distance is defined.

The address of variable d is stored to pointer variable, that is, ptr is pointing to variable d. Then, the member function of variable d is accessed using pointer.

Notes:

Since pointer ptr is pointing to variable d in this program, (*ptr).inch and d.inch are equivalent. Similarly, (*ptr).feet and d.feet are equivalent.
However, if we are using pointers, it is far more preferable to access struct members using the -> operator, since the . operator has a higher precedence than the * operator.

Hence, we enclose *ptr in brackets when using (*ptr).inch. Because of this, it is easier to make mistakes if both operators are used together in a single code.

ptr->feet is same as (*ptr).feet
ptr->inch is same as (*ptr).inc

Exercises

1. Write a C++ program that maintain record of students. Student contains following details:
   a) ID
   b) Name
   c) Department
   d) Email
   e) Phone number

   Create a structure names **Student**. Ask the user to enter record for 5 students and store those details in variables of **Student** type. Finally, print those records on screen.

2. Write a C++ program to do:
   a) Create a structure/record to store products; each product has a name, a model number and a price.
   b) Choose appropriate types for these fields/attributes.
   c) Your program should contain a loop which allows entry via the keyboard of up to 10 products, and stores them in an array
   d) Finally, the program should include a function to display all the products details after the entry loop has finished.

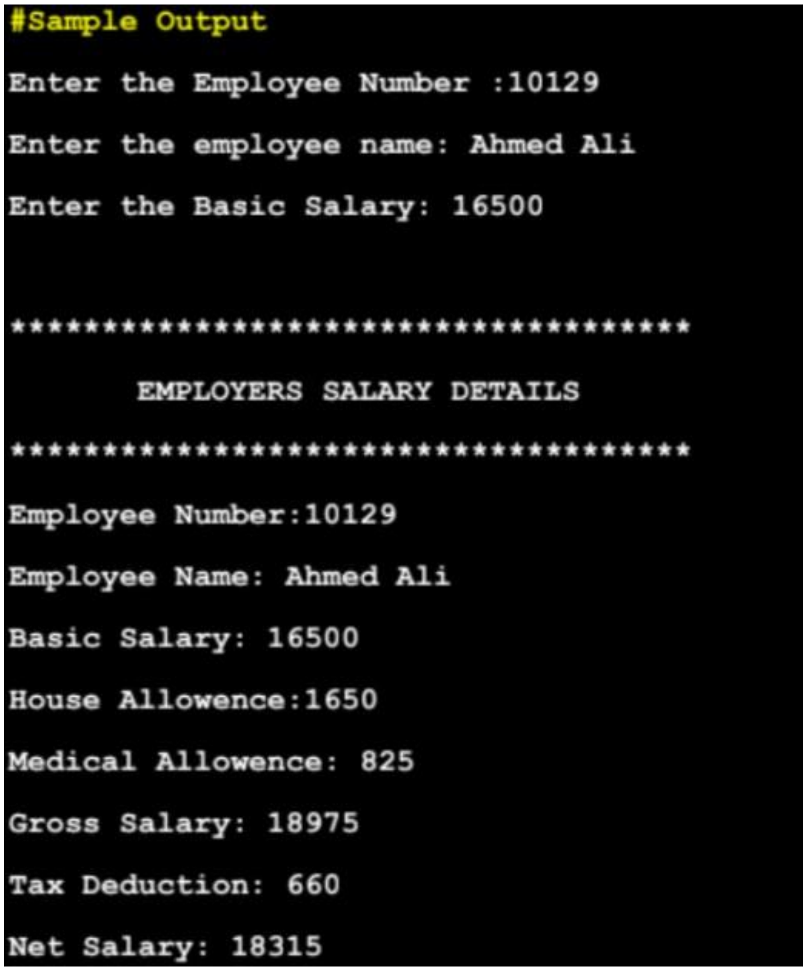
3. Write a C++ program that compute Net Salary of Employee. Program should contain two user defined functions **empSalary( )** and **display( )**.

   a) Create a structure of Employee that contains following data members:
      i. EmployeeNumber, Name, BasicSalary, HouseAllowance, MedicalAllowance, Tax, GrossPay and NetSalary

   b) EmployeeNumber, Name, and BasicSalary should be taken input from the user

   c) The function **empSalary( )** should compute the Employee salary with given criteria
      i. HouseAllowence = 10% of BasicSalary
      ii. Medical Allowence = 5% of Basic Salary
      iii. Tax = 4 % of Basic Salary
      iv. GrossSalary = Basic+HouseAllowence+MedicalAllowence
      v. NetSalary = GrossSalary – Tax

   d) The function display() should display the details of Employee, like shown below:



```
#Sample Output

Enter the Employee Number :10129

Enter the employee name: Ahmed Ali

Enter the Basic Salary: 16500


************************************

       EMPLOYERS SALARY DETAILS

************************************

Employee Number:10129

Employee Name: Ahmed Ali

Basic Salary: 16500

House Allowence:1650

Medical Allowence: 825

Gross Salary: 18975

Tax Deduction: 660

Net Salary: 18315
```

4. Consider the following code:

```cpp
#include <string>
#include <iostream>
using namespace std;
struct Student {
   string name;
   int id;
   int mark[3];
};
void inputStudent(Student* ptr); //function prototype for getting input
// some other function for printing the details
//*********************** Main Function ***********************//
int main () {
   Student stu;            // declaring an Student object
   Student* stuPtr = &stu; // defining a pointer for the object
   inputStudent(&stu); // calling function to get input into the object
   // calling the other function to print the details of the object return 0;
} // end main
```

In above given code there is a prototype, and a call to a function to input the values into a Student instance. Note that the parameter to this function is the address of a structure instance.

a) Write the definition for this function at the bottom of the file.
b) Then, write a function that takes a pointer to an instance of the Student structure and displays the contents

5. Making and Using Dynamically Allocated Arrays.
   Let us say that you want to choose how many marks to enter for the student and have the array change size accordingly. This gives you the opportunity to work with dynamic pointers.
   You will have to make the following changes to the code previously given:

a) Change **mark** within the **Student** struct to an integer pointer.
b) Ask the user from main how many marks he/she would like to enter
c) Change both the functions to have an additional parameter which is the number of marks.
d) For the **inputStudent** function, dynamically allocate the marks according to the number required.