CC-112L

Programming Fundamentals

Laboratory 11

Structures

Version: 1.0.0

Release Date: 30-09-2022

Department of Information Technology

University of the Punjab

Lahore, Pakistan

Contents:

Learning Objectives:

- Introduction
- Structure Definitions
- Initializing Structures
- Accessing Structure member with **“.”** and **“->”**
- Using Structures with Functions
- typedef
- Unions

Resources Required:

- Desktop Computer or Laptop
- Microsoft ® Visual Studio 2022

General Instructions:

- In this Lab, you are **NOT** allowed to discuss your solution with your colleagues, even not allowed to ask how is s/he doing, this may result in negative marking. You can **ONLY** discuss with your Teaching Assistants (TAs) or Lab Instructor.
- Your TAs will be available in the Lab for your help. Alternatively, you can send your queries via email to one of the followings.

Teachers:		
Course Instructor	Prof. Dr. Syed Waqar ul Qounain	swjaffry@pucit.edu.pk
Lab Instructor	Madiha Khalid	madiha.khalid@pucit.edu.pk
Teacher Assistants	Usman Ali	bitf19m007@pucit.edu.pk
	Saad Rahman	bsef19m021@pucit.edu.pk

Background and Overview:

Structure Definitions:

Structures are collections of related variables under one name, known as aggregates in the C standard. Structures may contain many variables of different types. That's in contrast to arrays, which contain only elements of the same type.

Structures are derived data types; they're constructed using objects of other types. The keyword struct introduces a structure definition, as in

```
struct Person {

        const char *name;

        int age;

};
```

Initializing Structures:

Like arrays, you can initialize a struct variable via an initializer list. For example, the following statement creates variable person1 using type struct Person and initializes member name to "Bob" and member age to 20:

```
struct Person person1 = {"Bob", 1};
```

If there are fewer initializers than members, the remaining members are automatically initialized to 0 or NULL (for pointer members).

Accessing Structure member with "." and "->":

You can access structure members with:

- The structure member operator (.), or dot operator, and
- The structure pointer operator (->), or arrow operator.

Using Structures with Functions:

With structures, you can pass to functions:

- individual structure members,
- entire structure objects, or
- pointers to structure objects.

Typedef:

The keyword typedef enables you to create synonyms (or aliases) for previously defined types. It's commonly used to create shorter names for struct types and simplify declarations of types like function pointers. For example, the following typedef defines person as a synonym for type struct Person

```
typedef struct Person person;
```

Unions:

Like a structure, a union is a derived data type, but its members share the same memory. At different times during program execution, some variables may not be relevant when others are. So, a union shares the space rather than wasting storage on variables that are not in use. A union's members can be of any type. The number of bytes used to store a union must be at least enough to hold its largest member.

Activities:

Pre-Lab Activities:

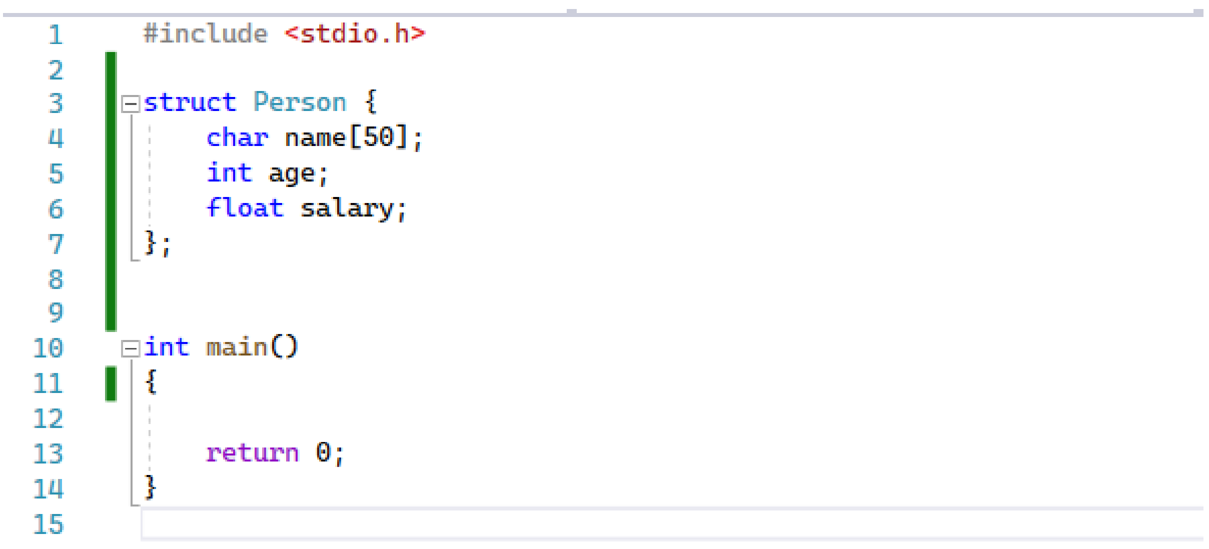
Define Structures:

In C programming, a struct (or structure) is a collection of variables (can be of different types) under a single name. Before you can create structure variables, you need to define its data type. To define a struct, the struct keyword is used.

Syntax of struct:

```
struct structureName {
  dataType member1;
  dataType member2;
  …
};
```

For example:



```
#include <stdio.h>

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


int main()
{

    return 0;
}

```

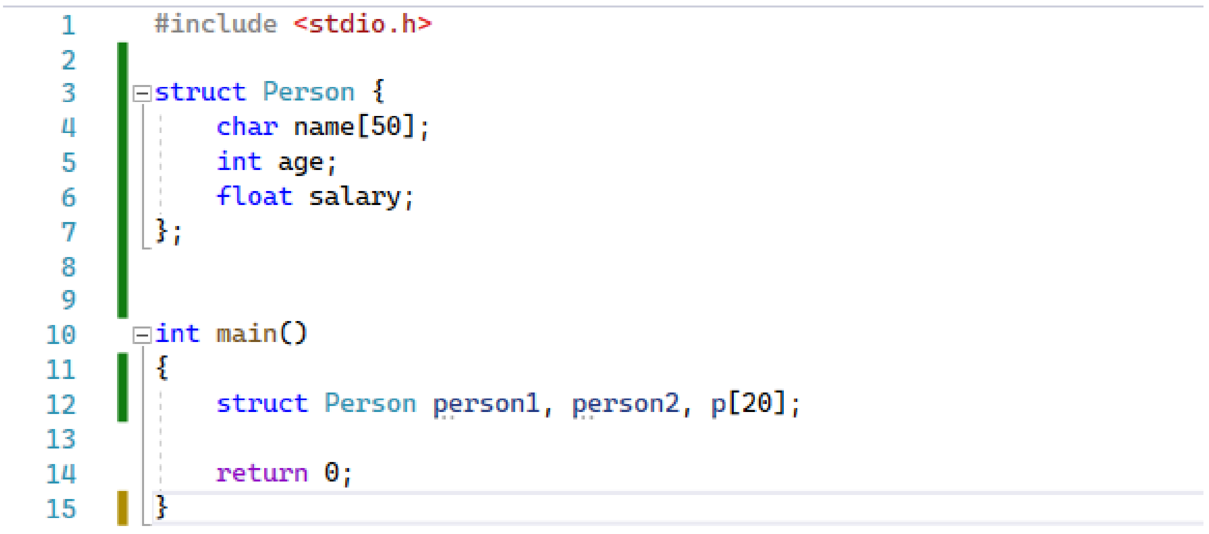
Fig. 01 (Struct Syntax)

Here, a derived type struct Person is defined. Now, you can create variables of this type.

Create struct Variables:

When a struct, type is declared, no storage or memory is allocated. To allocate memory of a given structure type and work with it, we need to create variables.

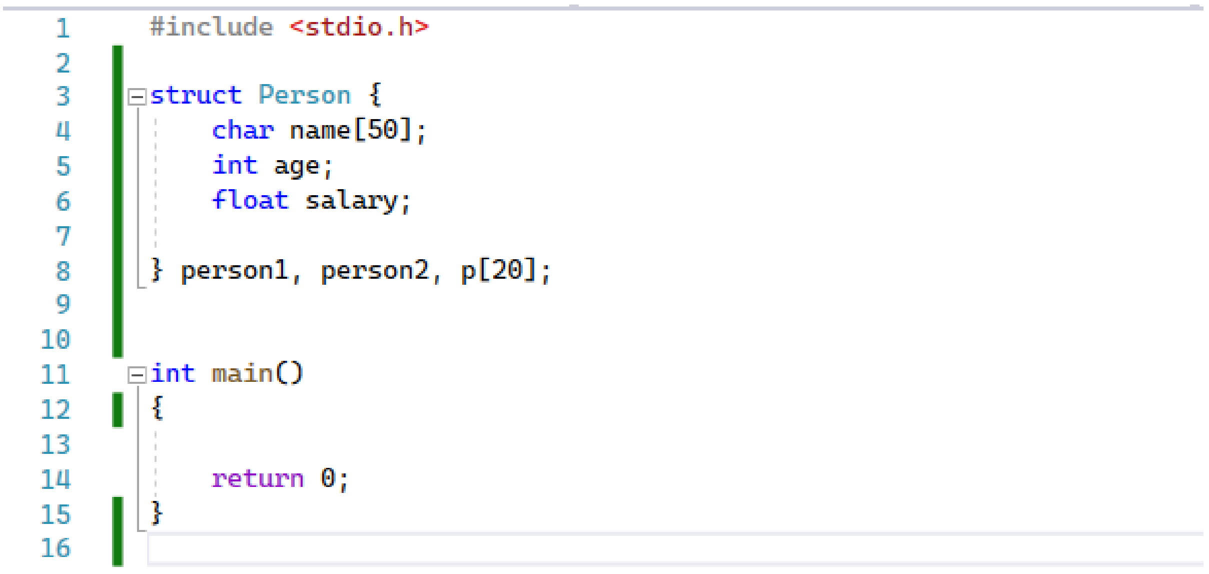
Here's how we create structure variables:



```
#include <stdio.h>

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


int main()
{
    struct Person person1, person2, p[20];

    return 0;
}
```

Fig. 02 (Create Struct - I)

Another way of creating a struct variable is:



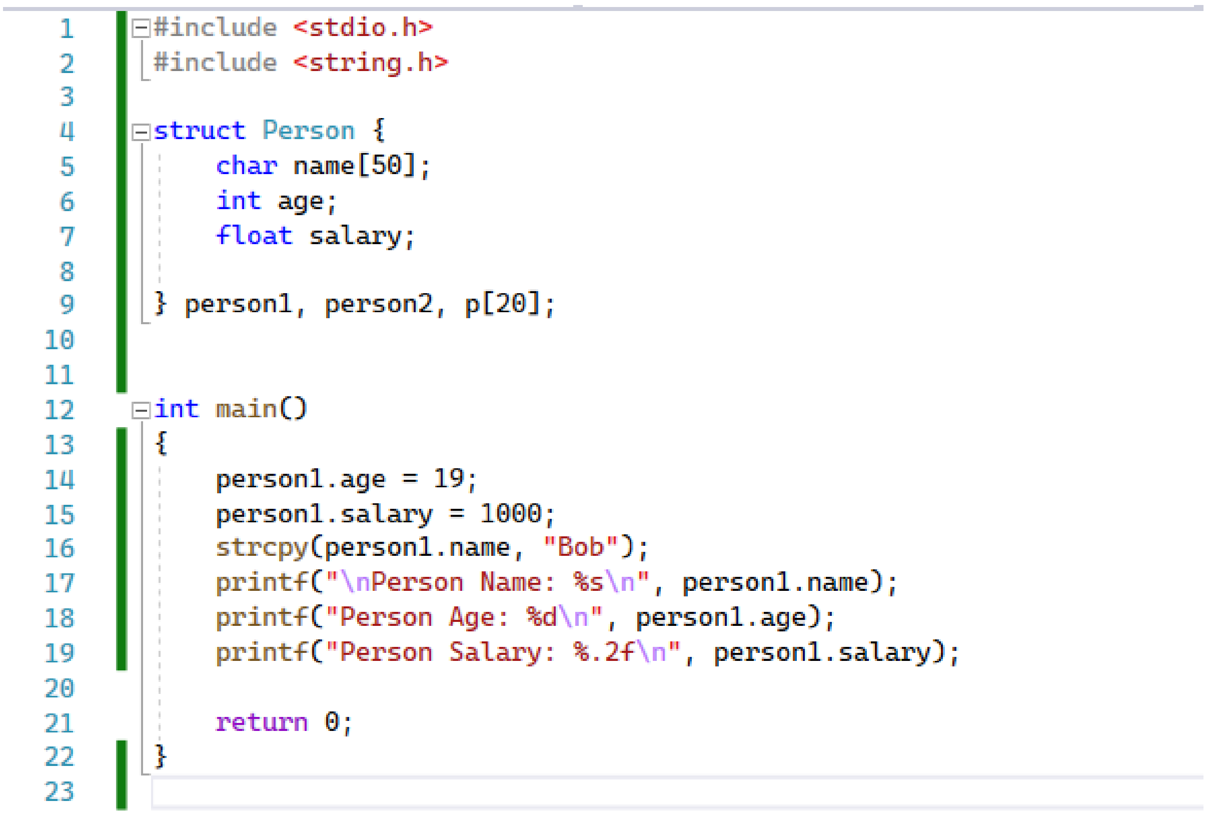
```
#include <stdio.h>

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

} person1, person2, p[20];


int main()
{

    return 0;
}

```

Fig. 03 (Create Struct - II)

In both cases,

- person1 and person2 are struct Person variables
- p [] is a struct Person array of size 20.

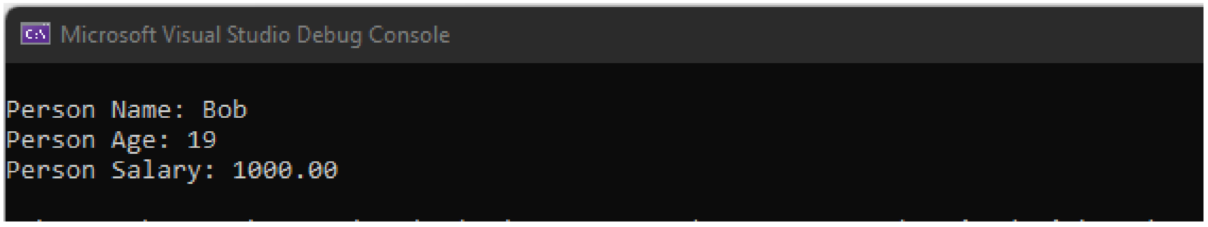
Example Code of struct:



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

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

} person1, person2, p[20];


int main()
{
    person1.age = 19;
    person1.salary = 1000;
    strcpy(person1.name, "Bob");
    printf("\nPerson Name: %s\n", person1.name);
    printf("Person Age: %d\n", person1.age);
    printf("Person Salary: %.2f\n", person1.salary);

    return 0;
}
```

Fig. 04 (Example Code Struct)

Output:



Microsoft Visual Studio Debug Console

```
Person Name: Bob
Person Age: 19
Person Salary: 1000.00
```

Fig. 05 (Struct Example Code Output)

Access Members of a Structure:

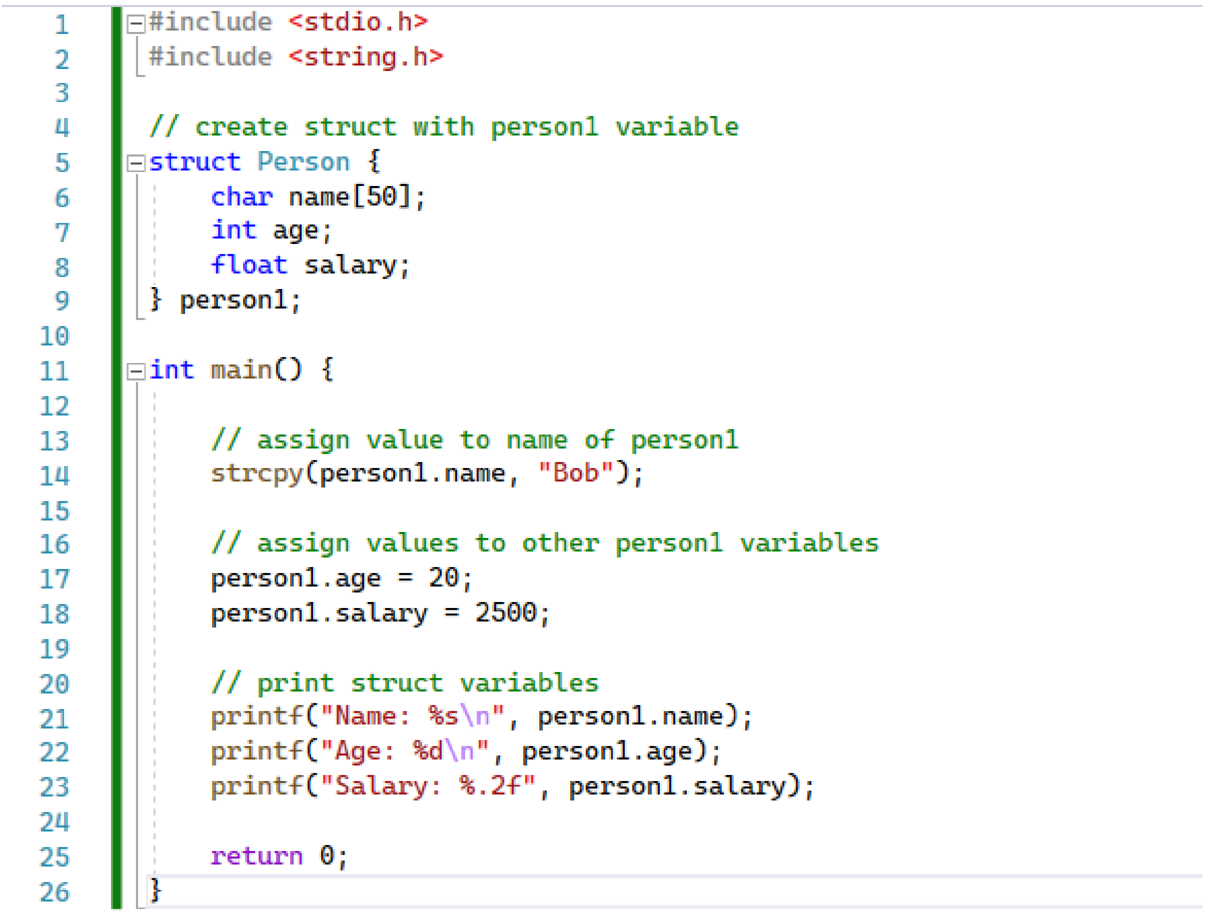
There are two types of operators used for accessing members of a structure.

- "**.**": Member operator
- "->": Structure pointer operator

Suppose, you want to access the salary of person1. Here's how you can do it.

```c
printf("Person Salary: %.2f\n", person1.salary);
```

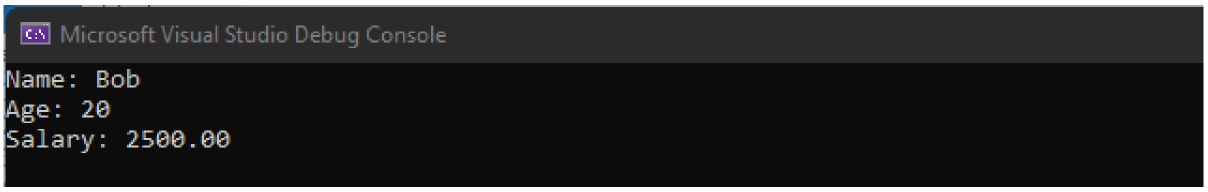
Example C struct:



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

// create struct with person1 variable
struct Person {
    char name[50];
    int age;
    float salary;
} person1;

int main() {

    // assign value to name of person1
    strcpy(person1.name, "Bob");

    // assign values to other person1 variables
    person1.age = 20;
    person1.salary = 2500;

    // print struct variables
    printf("Name: %s\n", person1.name);
    printf("Age: %d\n", person1.age);
    printf("Salary: %.2f", person1.salary);

    return 0;
}
```

Fig. 06 (Access Members of struct)

Output:

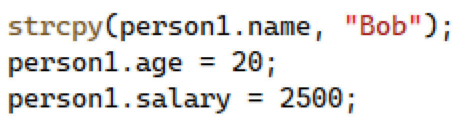


```
Name: Bob
Age: 20
Salary: 2500.00
```

Fig. 07 (Output Access Members of struct)

In this program, we have created a struct named Person. We have also created a variable of Person named person1.

In main (), we have assigned values to the variables defined in Person for the person1 object.



```c
strcpy(person1.name, "Bob");
person1.age = 20;
person1.salary = 2500;
```

Fig. 08 (Accessing members of struct)

Notice that we have used strcpy () function to assign the value to person1.name.

This is because name is a char array (C-string) and we cannot use the assignment operator = with it after we have declared the string.

Finally, we printed the data of person1.

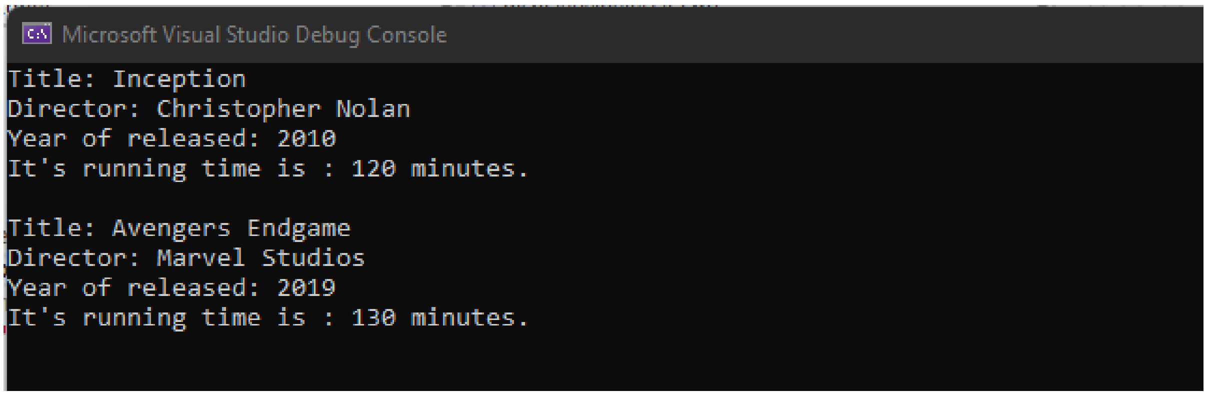
Task 01: Movie Data Struct **[30 minutes / 20 marks]**

Write a program that uses a structure named MovieData to store the following information about a movie:

- Title
- Director
- Year Released
- Running Time (in minutes)

The program should create two MovieData variables, store values in their members, and pass each one, in turn, to a function that displays the information about the movie in a clearly formatted manner.

Sample Output:



```
Title: Inception
Director: Christopher Nolan
Year of released: 2010
It's running time is : 120 minutes.

Title: Avengers Endgame
Director: Marvel Studios
Year of released: 2019
It's running time is : 130 minutes.
```

Fig. 09 (Pre-Lab Task 01)

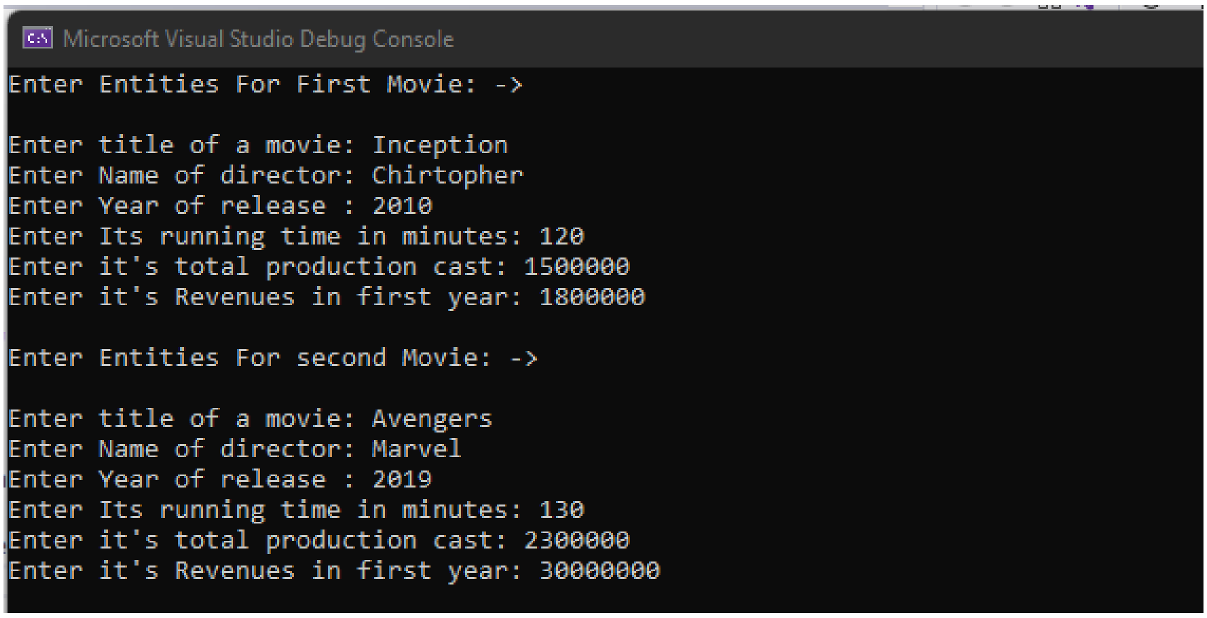
Submit **".c"** file named your **"MovieData1.c"** on Google Classroom.

Task 02: Update Movie Data Struct **[20 minutes / 20 marks]**

Modify the program you have written in task 1 such that MovieData structure includes two additional members that hold the movie's production costs and first-year revenues. Modify the function that displays the movie data to display the title, director, release year, running time, and first year's profit or loss. To complete this task, you have to implement following two functions:

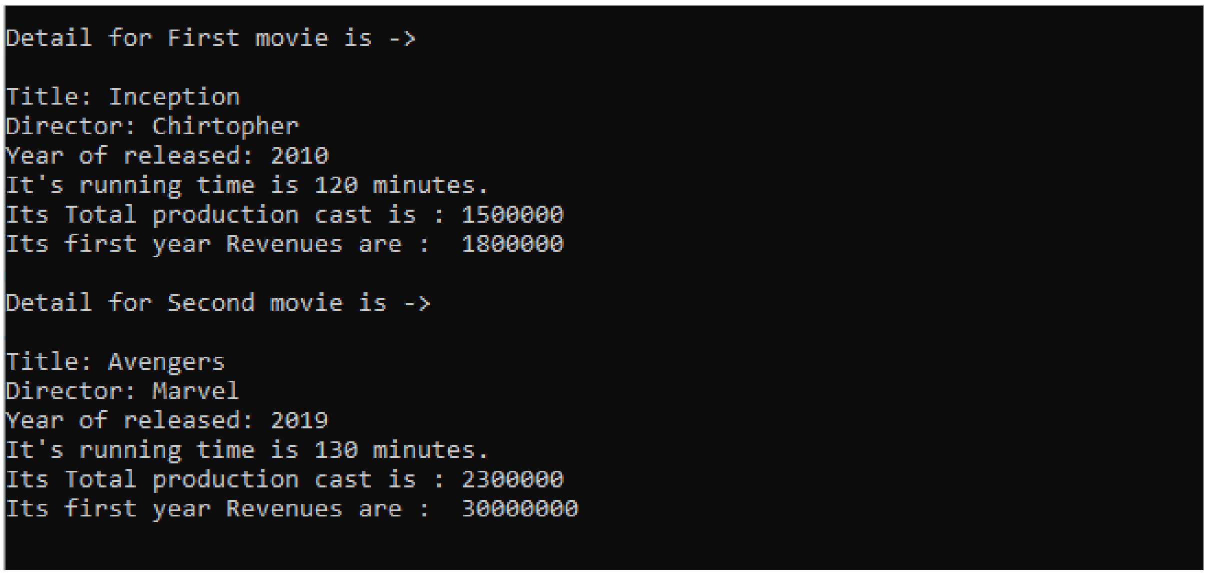
- void inputMovieData (MovieData &mv);
- void displayMovieData (MovieData mv);

Sample Output:



```
Enter Entities For First Movie: ->

Enter title of a movie: Inception
Enter Name of director: Chirtopher
Enter Year of release : 2010
Enter Its running time in minutes: 120
Enter it's total production cast: 1500000
Enter it's Revenues in first year: 1800000

Enter Entities For second Movie: ->

Enter title of a movie: Avengers
Enter Name of director: Marvel
Enter Year of release : 2019
Enter Its running time in minutes: 130
Enter it's total production cast: 2300000
Enter it's Revenues in first year: 30000000
```

Fig. 10 (Pre-Lab Task 02)



```
Detail for First movie is ->

Title: Inception
Director: Chirtopher
Year of released: 2010
It's running time is 120 minutes.
Its Total production cast is : 1500000
Its first year Revenues are :  1800000

Detail for Second movie is ->

Title: Avengers
Director: Marvel
Year of released: 2019
It's running time is 130 minutes.
Its Total production cast is : 2300000
Its first year Revenues are :  30000000
```

Fig. 11 (Pre-Lab Task 02)

Submit **".c"** file named your **"MovieData2.c"** on Google Classroom.

In-Lab Activities:

Why structs in C?

Suppose you want to store information about a person: his/her name, citizenship number, and salary. You can create different variables name, age and salary to store this information.

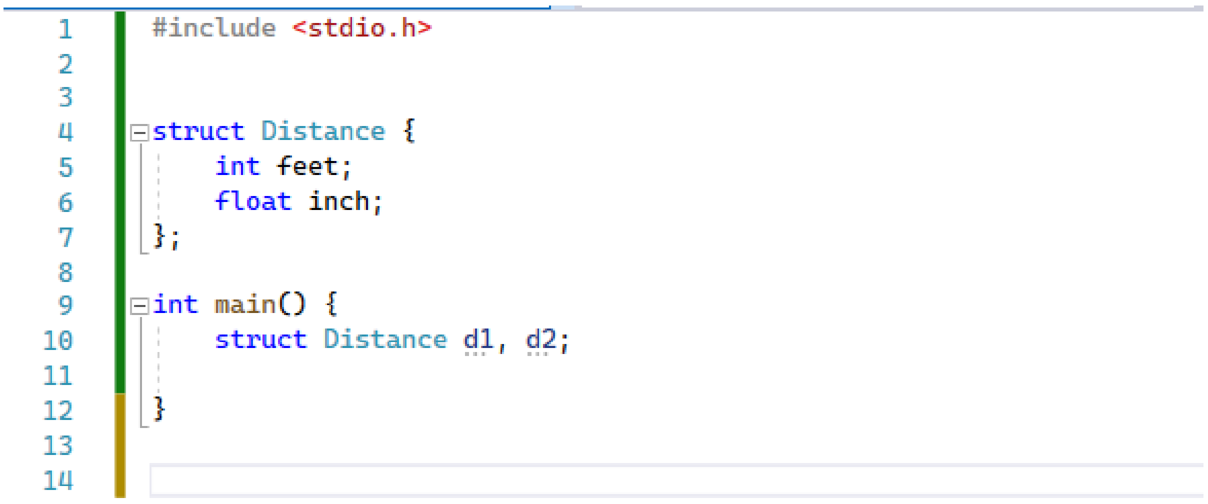
What if you need to store information of more than one person? Now, you need to create different variables for each information per person: name1, cage1, salary1, name2, age2, salary2, etc.

A better approach would be to have a collection of all related information under a single name Person structure and use it for every person.

Keyword typedef:

We use the typedef keyword to create an alias name for data types. It is commonly used with structures to simplify the syntax of declaring variables.

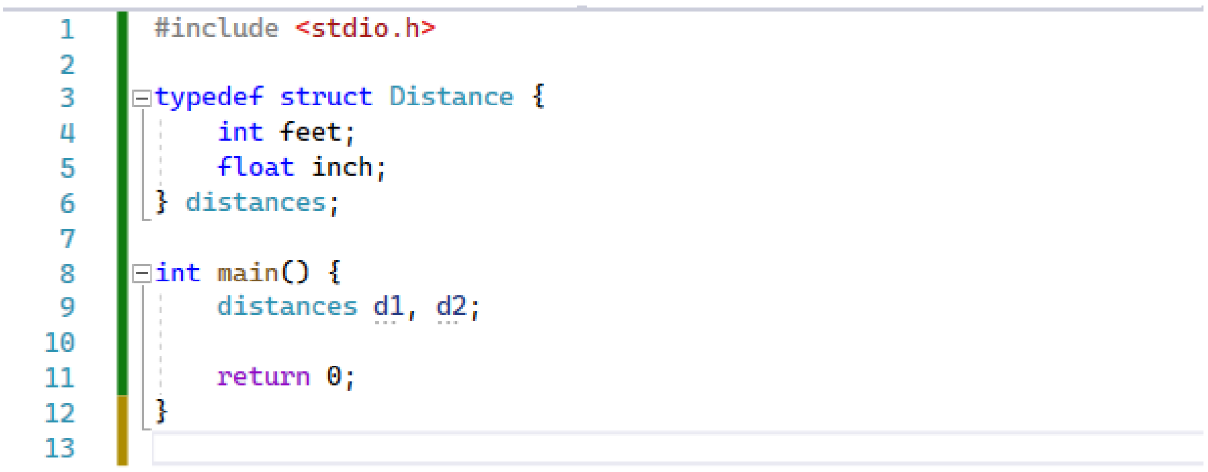
For example, let us look at the following code:



```
#include <stdio.h>


struct Distance {
    int feet;
    float inch;
};

int main() {
    struct Distance d1, d2;

}

```

Fig. 12 (Keyword typedef)

We can use typedef to write an equivalent code with a simplified syntax:

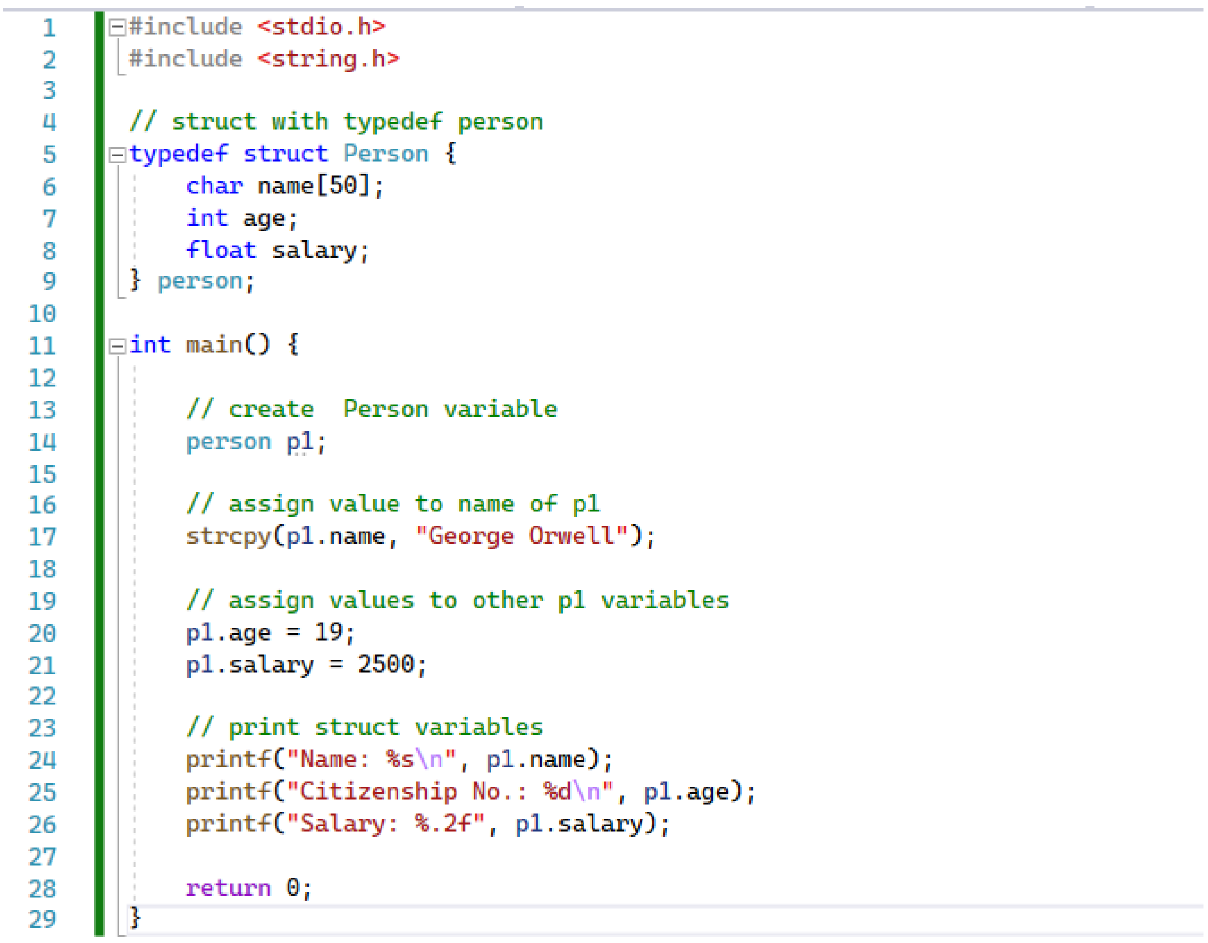


```c
#include <stdio.h>

typedef struct Distance {
    int feet;
    float inch;
} distances;

int main() {
    distances d1, d2;

    return 0;
}
```

Fig. 13 (typdef Example Code)

Typedef struct in C Example 2:

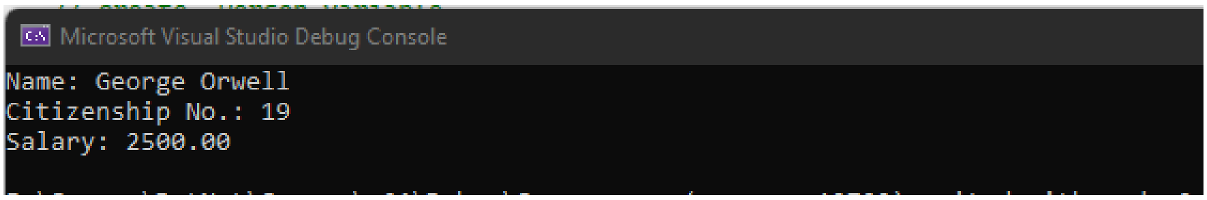
Code:



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

// struct with typedef person
typedef struct Person {
    char name[50];
    int age;
    float salary;
} person;

int main() {

    // create  Person variable
    person p1;

    // assign value to name of p1
    strcpy(p1.name, "George Orwell");

    // assign values to other p1 variables
    p1.age = 19;
    p1.salary = 2500;

    // print struct variables
    printf("Name: %s\n", p1.name);
    printf("Citizenship No.: %d\n", p1.age);
    printf("Salary: %.2f", p1.salary);

    return 0;
}
```

Fig. 14 (Typedef Example Code in C)

Output:


Microsoft Visual Studio Debug Console

```
Name: George Orwell
Citizenship No.: 19
Salary: 2500.00
```

Fig. 15 (Output of Typedef Example Code in C)

Here, we have used **typedef** with the **Person** structure to create an alias **person.**

// struct with typedef person

typedef struct Person {

  // code

} person;

Now, we can simply declare a **Person** variable using the **person** alias:

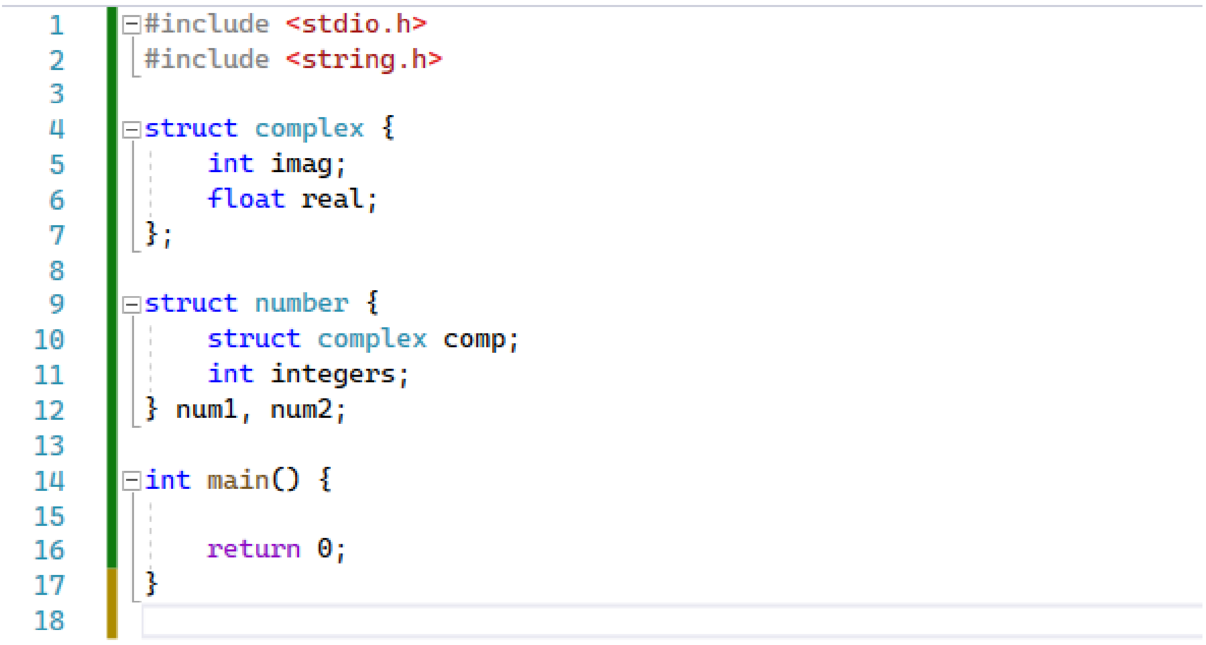
// equivalent to struct Person p1

person p1;

Nested Structures:

You can create structures within a structure in C programming.

For example,



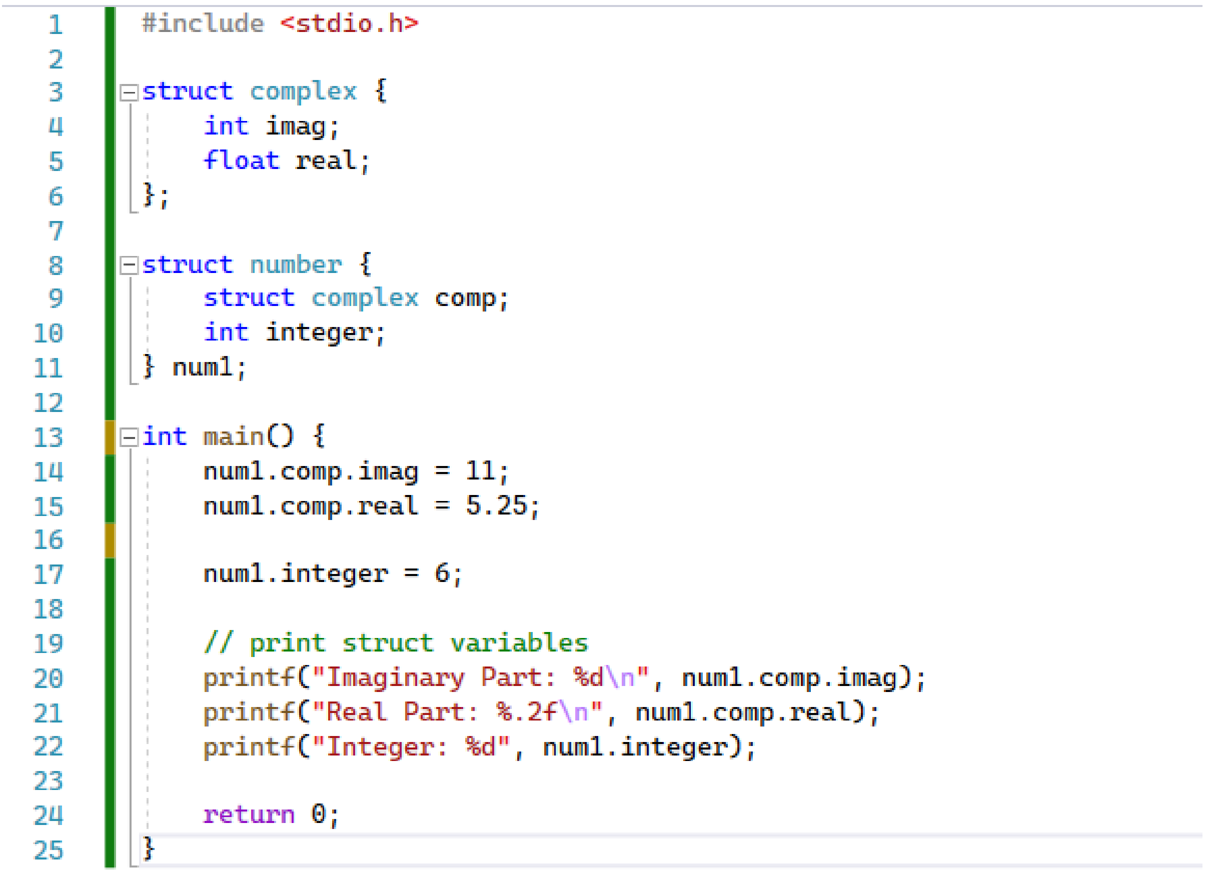
```c
#include <stdio.h>
#include <string.h>

struct complex {
    int imag;
    float real;
};

struct number {
    struct complex comp;
    int integers;
} num1, num2;

int main() {

    return 0;
}

```

Fig. 16 (Nested Struct in C)

Suppose, you want to set imag of num2 variable to 11. Here's how you can do it:

num2.comp.imag = 11;

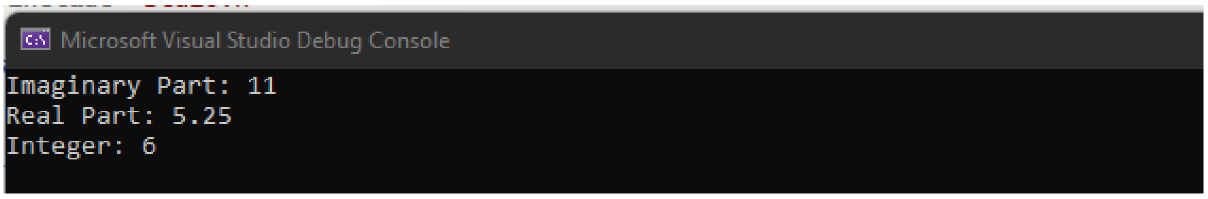
Example Code:



```c
#include <stdio.h>

struct complex {
    int imag;
    float real;
};

struct number {
    struct complex comp;
    int integer;
} num1;

int main() {
    num1.comp.imag = 11;
    num1.comp.real = 5.25;

    num1.integer = 6;

    // print struct variables
    printf("Imaginary Part: %d\n", num1.comp.imag);
    printf("Real Part: %.2f\n", num1.comp.real);
    printf("Integer: %d", num1.integer);

    return 0;
}
```

Fig. 17 (Nested Struct Example Code in C)

Output:



```
Imaginary Part: 11
Real Part: 5.25
Integer: 6
```

Fig. 18 (Output of Nest Struct Example in C)

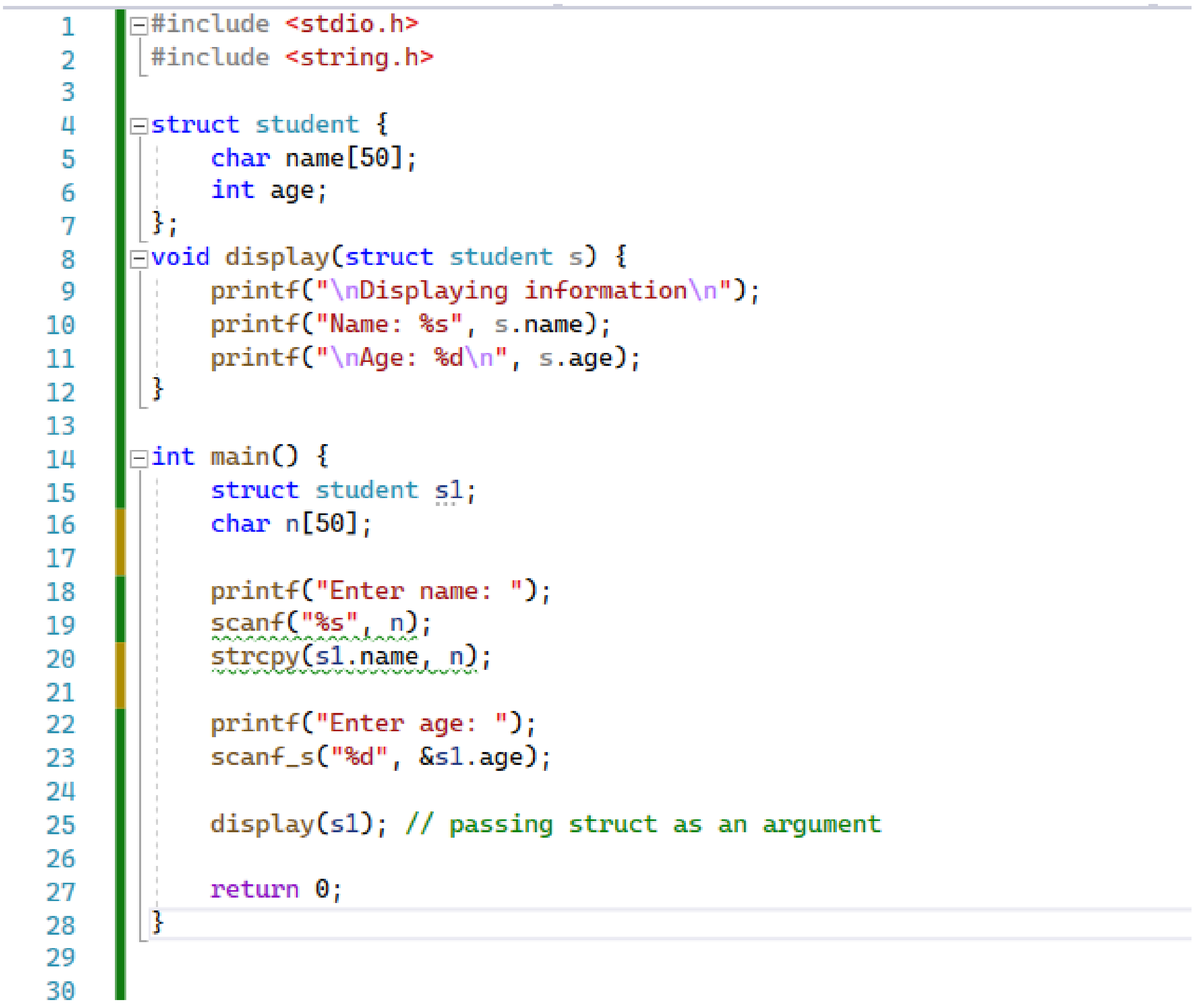
C Structure and Function:

Similar to variables of built-in types, you can also pass structure variables to a function.

Passing structs to functions:

Here's how you can pass structures to a function.

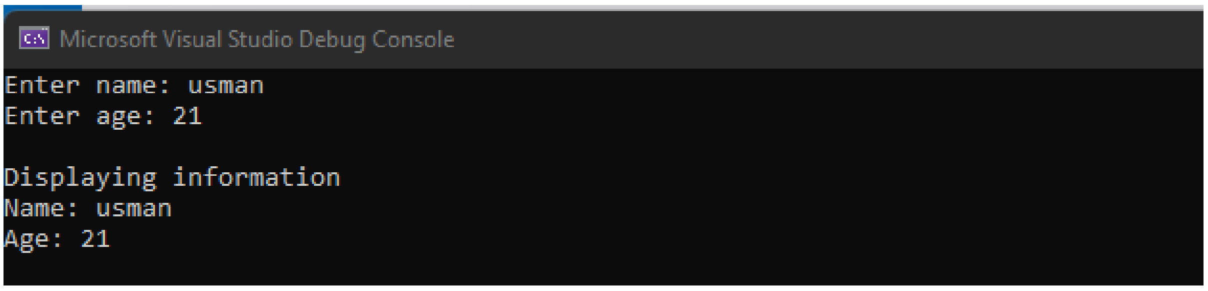
Example Code:



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

struct student {
    char name[50];
    int age;
};
void display(struct student s) {
    printf("\nDisplaying information\n");
    printf("Name: %s", s.name);
    printf("\nAge: %d\n", s.age);
}

int main() {
    struct student s1;
    char n[50];

    printf("Enter name: ");
    scanf("%s", n);
    strcpy(s1.name, n);

    printf("Enter age: ");
    scanf_s("%d", &s1.age);

    display(s1); // passing struct as an argument

    return 0;
}
```

Fig. 19 (Passing structs to function Example in C)

Output:



Microsoft Visual Studio Debug Console

```
Enter name: usman
Enter age: 21

Displaying information
Name: usman
Age: 21
```

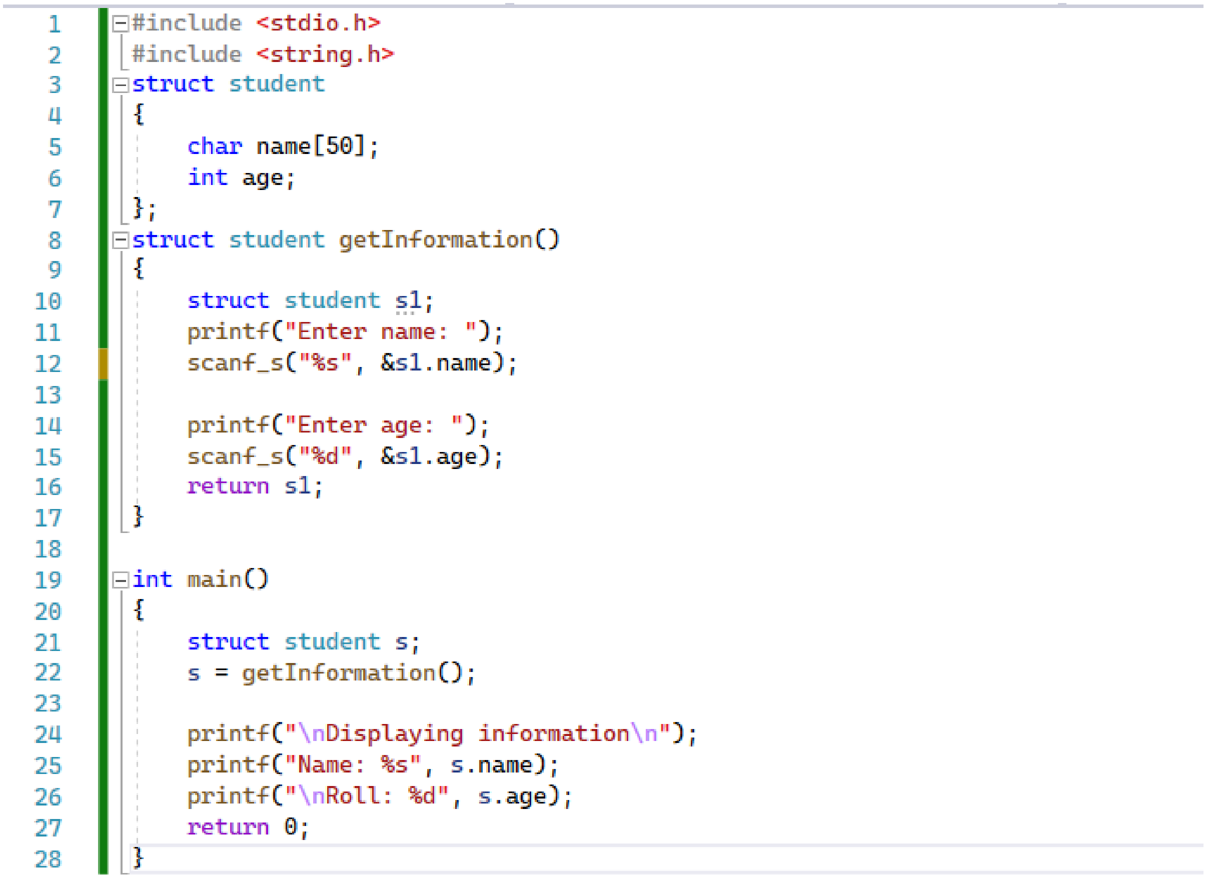
Fig. 20 (Output of Passing structs to function Example in C)

Here, a struct variable s1 of type struct student is created. The variable is passed to the display () function using **display(s1);** statement.

Return structs from a function:

Here's how you can return structure from a function:

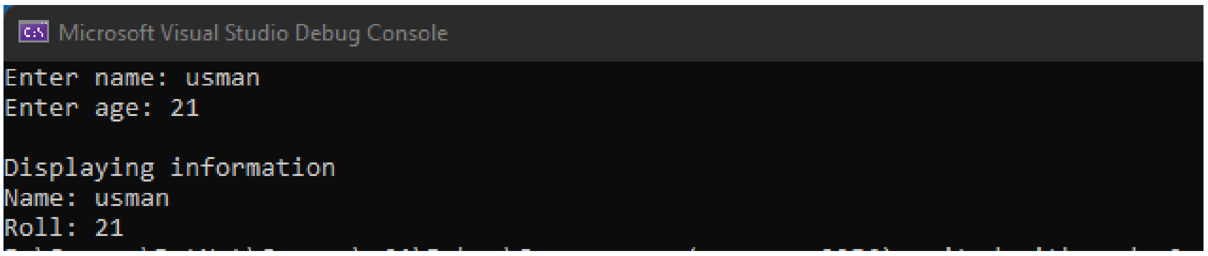
Example Code:



```c
#include <stdio.h>
#include <string.h>
struct student
{
    char name[50];
    int age;
};
struct student getInformation()
{
    struct student s1;
    printf("Enter name: ");
    scanf_s("%s", &s1.name);

    printf("Enter age: ");
    scanf_s("%d", &s1.age);
    return s1;
}

int main()
{
    struct student s;
    s = getInformation();

    printf("\nDisplaying information\n");
    printf("Name: %s", s.name);
    printf("\nRoll: %d", s.age);
    return 0;
}
```

Fig. 21 (Returning struct from function Example in C)

Output:



```
Enter name: usman
Enter age: 21

Displaying information
Name: usman
Roll: 21
```

Fig. 22 (Output of returning struct Example in C)

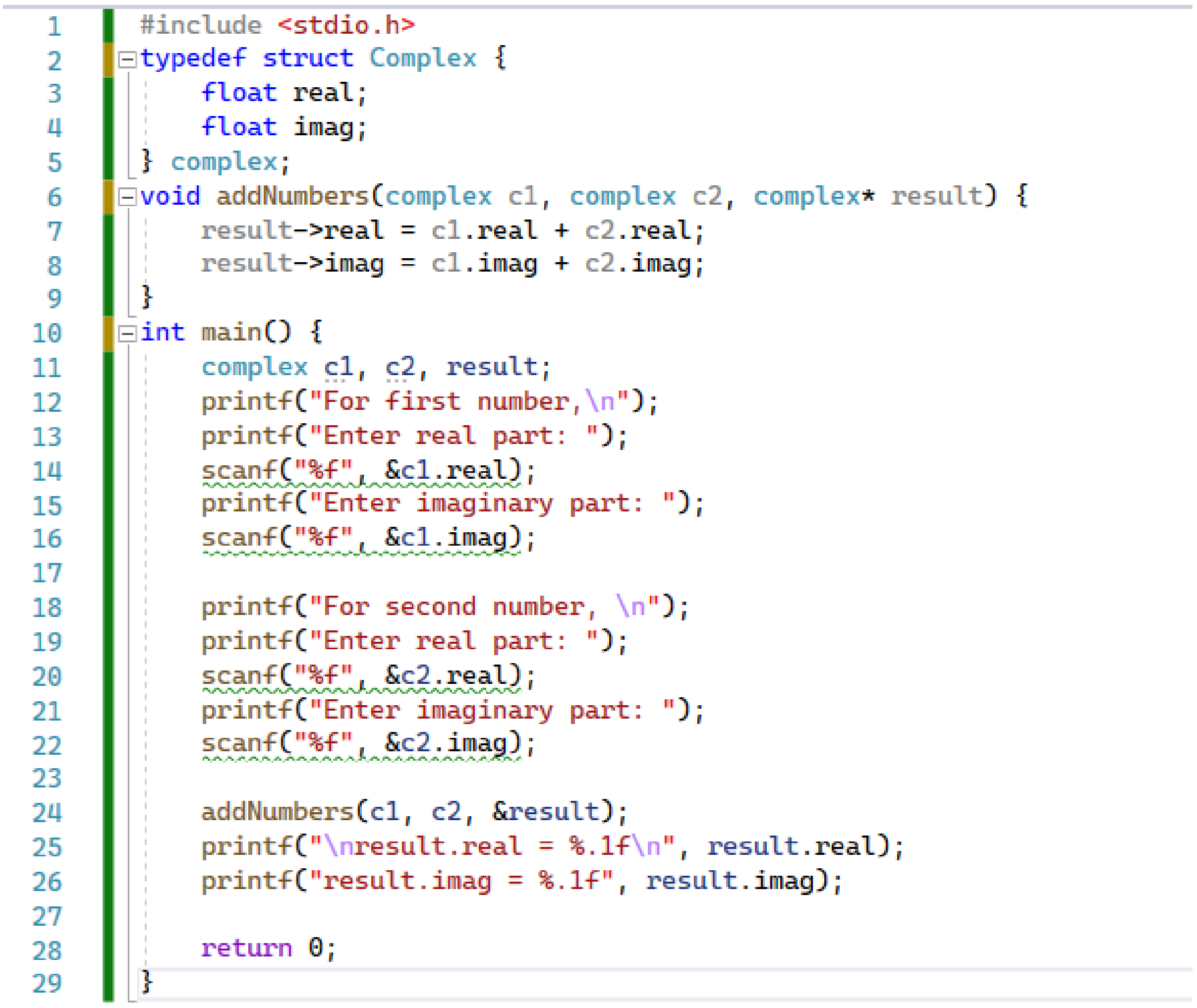
Here, the **getInformation ()** function is called using **s = getInformation ();** statement. The function returns a structure of type struct student. The returned structure is displayed from the main () function.

Notice that, the return type of **getInformation ()** is also struct student.

Passing struct by reference:

You can also pass structs by reference (in a similar way like you pass variables of built-in type by reference). We suggest you to read pass by reference tutorial before you proceed. During pass by reference, the memory addresses of struct variables are passed to the function.

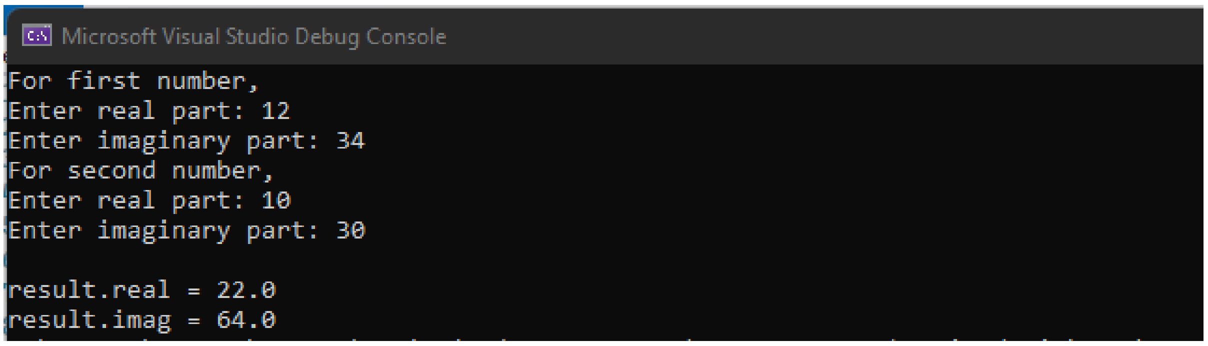
Example Code:



```c
#include <stdio.h>
typedef struct Complex {
    float real;
    float imag;
} complex;
void addNumbers(complex c1, complex c2, complex* result) {
    result->real = c1.real + c2.real;
    result->imag = c1.imag + c2.imag;
}
int main() {
    complex c1, c2, result;
    printf("For first number,\n");
    printf("Enter real part: ");
    scanf("%f", &c1.real);
    printf("Enter imaginary part: ");
    scanf("%f", &c1.imag);

    printf("For second number, \n");
    printf("Enter real part: ");
    scanf("%f", &c2.real);
    printf("Enter imaginary part: ");
    scanf("%f", &c2.imag);

    addNumbers(c1, c2, &result);
    printf("\nresult.real = %.1f\n", result.real);
    printf("result.imag = %.1f", result.imag);

    return 0;
}
```

Fig. 23 (Passing struct to function by reference in C)

Output:



```
For first number,
Enter real part: 12
Enter imaginary part: 34
For second number,
Enter real part: 10
Enter imaginary part: 30

result.real = 22.0
result.imag = 64.0
```

Fig. 24 (Output of passing struct by reference in C)

In the above program, three structure variables c1, c2 and the address of result is passed to the **addNumbers ()** function. Here, result is passed by reference.

When the result variable inside the **addNumbers ()** is altered, the result variable inside the main () function is also altered accordingly.

Task 01: Score Board **[40 minutes / 30 marks]**

Write a program that stores a team of score players where the team has 12 players in it. Create a structure that stores following data about a soccer player:

- Player's Name
- Player's Number
- Points Scored by Player

To store all players of the team, the program should keep an array of 12 of these structures. Each element is for a different player on a team. Write a function that populates the array of structures in a function by user given data for each player. It should then show a table that lists each player's number, name, and points scored. The program should also calculate and display the total points earned by the team. The number and name of the player who has earned the most points should also be displayed.

To complete this task, you have to implement following functions:

PopulateArray: to populate the array of structures by user provided data.

DisplayScoreBoard: to display the name, number and score of each player.

FindTopScorer: this function should return the player with highest score.

Call each of these functions in main () to demonstrate your work.

Input Validation: Do not accept negative values for player's numbers or points scored.

Fig. 16 (In-Lab Task 01)

Submit **".c"** file named your **"ScoreBoard.c"** on Google Classroom.

Task 02: Student Details **[40 minutes / 30 marks]**

Write and program that should create a structure to store following data about a student.

- Name
- rollNo
- Date of birth
- Number of courses the student is studying
- Marks of all courses the student is studying
- CGPA

Date of birth of the student should be of type DATE, therefore you should implement another structure to represent the date of birth. Marks of student should be stored in an array where size of array should be given by the user. Create 3 variables of type Student in the main () function. Identify any student who is going to dropout due to low CGPA (consider the bare minimum CGPA to continue is 2.5).

Minimum marks to pass a course are 50. Display all the failed courses for each student.

Input Validation: Do not accept negative values and values greater than 100 for marks of each course. Do not accept CGPA greater than 4.00 and less than 0.00.

Submit **".c"** file named your **"Student.c"** on Google Classroom.

Post-Lab Activities:

Unions:

A union is a user-defined type similar to structs in C except for one key difference.

Structures allocate enough space to store all their members, whereas unions can only hold one member value at a time.

How to define a union?

We use the union keyword to define unions. Here's an example:

```
union car

{

  char name [50];

  int price;

};
```

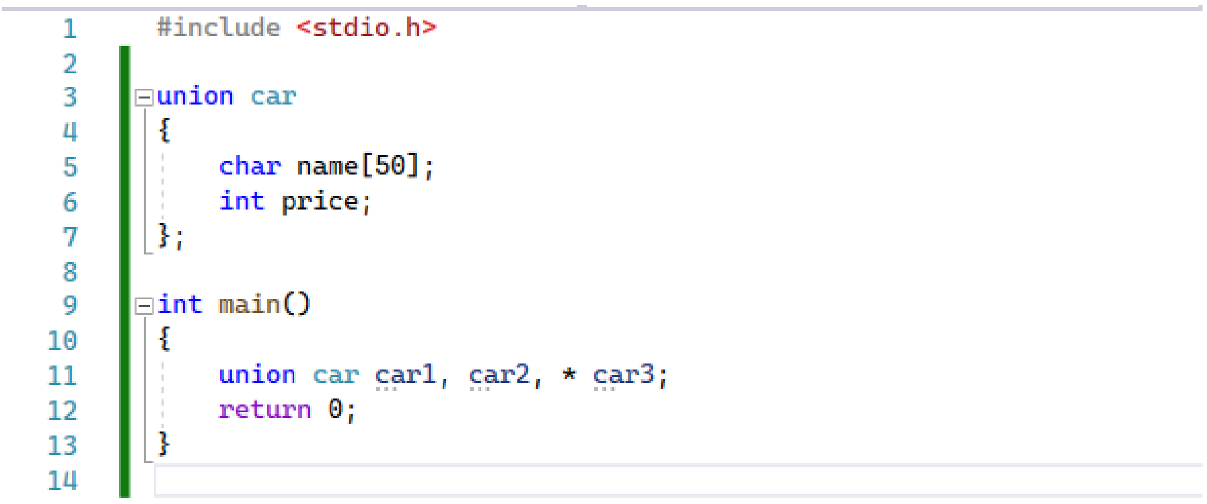
The above code defines a derived type union car.

Create union variables:

When a union is defined, it creates a user-defined type. However, no memory is allocated. To allocate memory for a given union type and work with it, we need to create variables.

Here's how we create union variables.

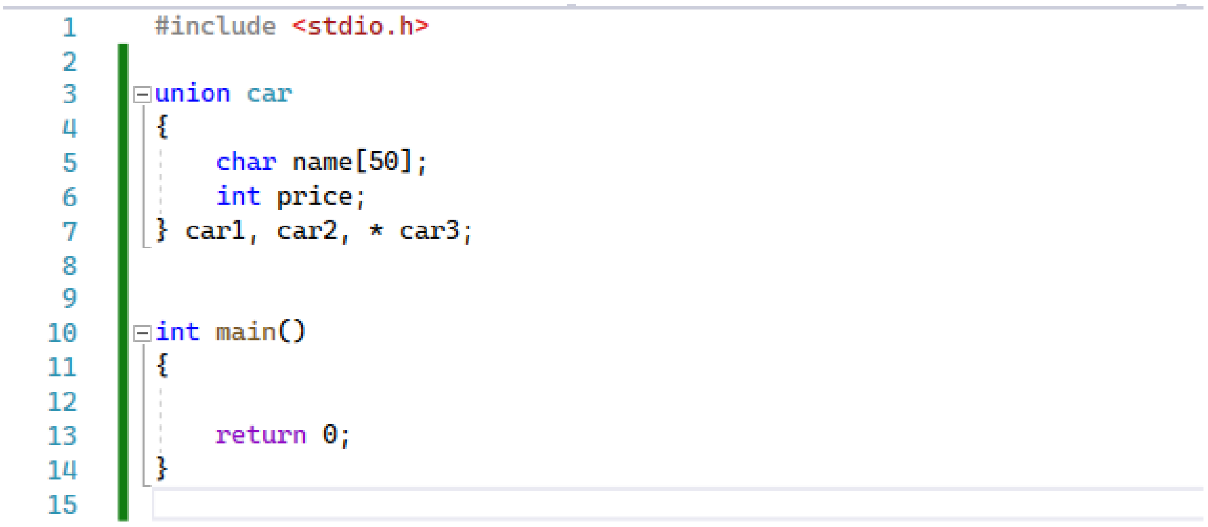
Example Code:



```
#include <stdio.h>

union car
{
    char name[50];
    int price;
};

int main()
{
    union car car1, car2, * car3;
    return 0;
}
```

Fig. 27 (Creating Unions in C)

Another way of creating union variables is:



```c
#include <stdio.h>

union car
{
    char name[50];
    int price;
} car1, car2, * car3;


int main()
{

    return 0;
}

```

Fig. 28 (Creating Unions in C)

In both cases, union variables car1, car2, and a union pointer car3 of union car type are created.

Access members of a union:

We use the **“.”** operator to access members of a union. And to access pointer variables, we use the **“->”** operator.

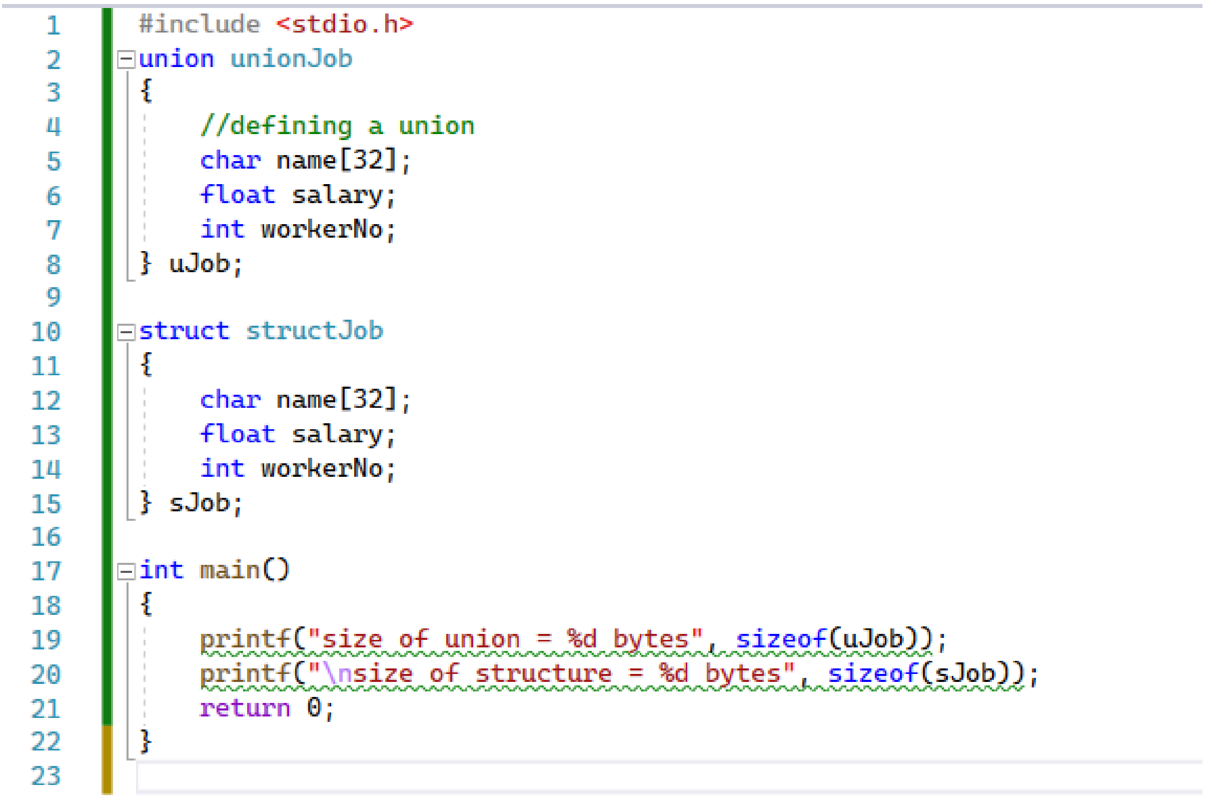
In the above example,

- To access price for car1, car1.price is used.
- To access price using car3, either (*car3).price or car3->price can be used.

Difference between unions and structures:

Let's take an example to demonstrate the difference between unions and structures:

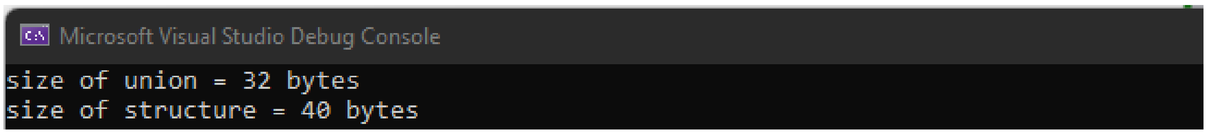
Example Code:



```c
#include <stdio.h>
union unionJob
{
    //defining a union
    char name[32];
    float salary;
    int workerNo;
} uJob;

struct structJob
{
    char name[32];
    float salary;
    int workerNo;
} sJob;

int main()
{
    printf("size of union = %d bytes", sizeof(uJob));
    printf("\nsize of structure = %d bytes", sizeof(sJob));
    return 0;
}
```

Fig. 29 (Difference between Union and struct in C)

Output:



```
size of union = 32 bytes
size of structure = 40 bytes
```

Fig. 30 (Output Difference between Union and struct in C)

Why this difference in the size of union and structure variables?

Here, the size of sJob is 40 bytes because

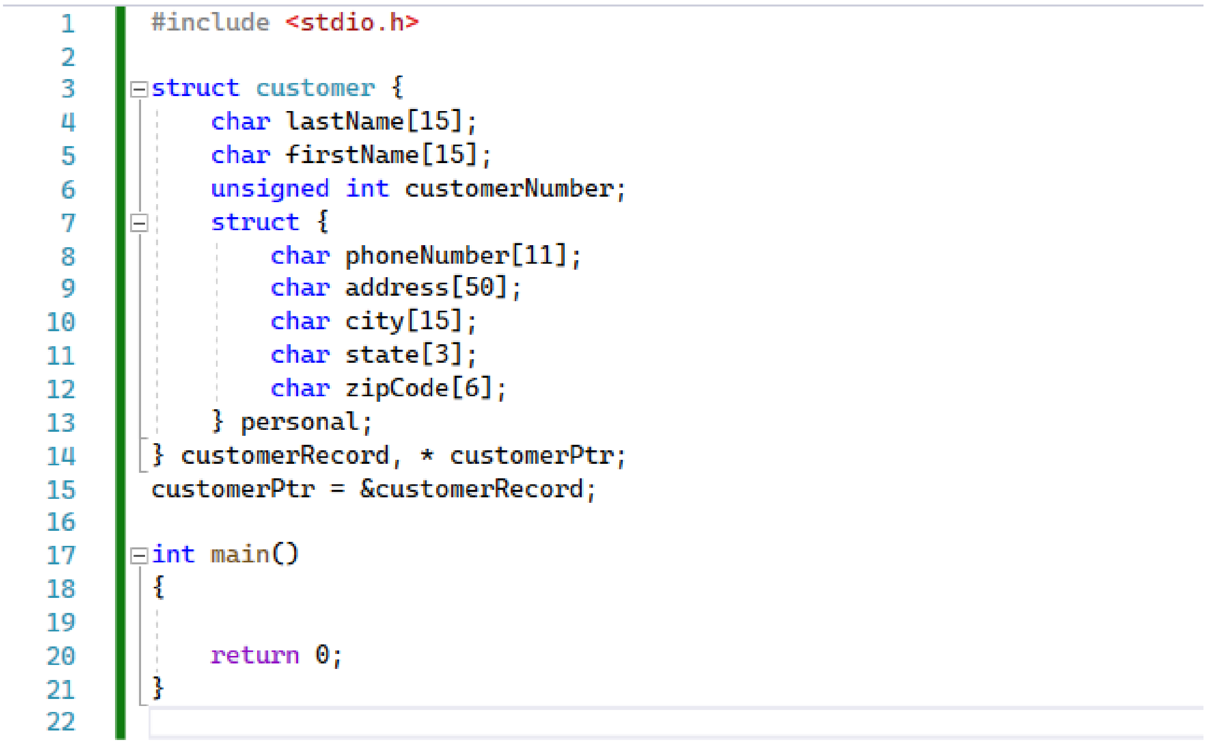
- the size of name [32] is 32 bytes
- the size of salary is 4 bytes
- the size of workerNo is 4 bytes

However, the size of uJob is 32 bytes. It's because the size of a union variable will always be the size of its largest element. In the above example, the size of its largest element, (name [32]), is 32 bytes.

With a union, all members share the same memory.

Task 01: Score Board **[40 minutes / 30 marks]**

Code:



```c
#include <stdio.h>

struct customer {
    char lastName[15];
    char firstName[15];
    unsigned int customerNumber;
    struct {
        char phoneNumber[11];
        char address[50];
        char city[15];
        char state[3];
        char zipCode[6];
    } personal;
} customerRecord, * customerPtr;
customerPtr = &customerRecord;

int main()
{

    return 0;
}

```

Fig. 31 (Post-Lab Task 01)

Write an expression that accesses the struct members in each of the following parts:

a) Member lastName of struct customerRecord.

b) Member lastName of the struct pointed to by customerPtr.

c) Member firstName of struct customerRecord.

d) Member firstName of the struct pointed to by customerPtr.

e) Member customerNumber of struct customerRecord.

f) Member customerNumber of the struct pointed to by customerPtr.

g) Member phoneNumber of member personal of struct customerRecord.

h) Member phoneNumber of member personal of the struct pointed to by customerPtr.

i) Member address of member personal of struct customerRecord.

j) Member address of member personal of the struct pointed to by customerPtr.

k) Member city of member personal of struct customerRecord.

l) Member city of member personal of the struct pointed to by customerPtr.

m)Member state of member personal of struct customerRecord.

n) Member state of member personal of the struct pointed to by customerPtr.

o) Member zipCode of member personal of struct customerRecord.

p) Member zipCode of member personal of the struct pointed to by customerPtr.

Write code in the main function of above given screenshot.

Submit **".c"** file named your **"Accessing.c"** on Google Classroom.

Submissions:

- For In-Lab Activity:
  - Save the files on your PC.
  - TA's will evaluate the tasks offline.
- For Pre-Lab & Post-Lab Activity:
  - Submit the .c file on Google Classroom and name it to your roll no.

Evaluations Metric:

• All the lab tasks will be evaluated offline by TA's	
• **Division of Pre-Lab marks:**	**[50 marks]**
▪ Task 01: Movie Data Struct	[30 marks]
▪ Task 02: Update Movie Data Struct	[20 marks]
• **Division of In-Lab marks:**	**[60 marks]**
▪ Task 01: Score Board	[30 marks]
▪ Task 02: Student Details	[30 marks]
• **Division of Post-Lab marks:**	**[30 marks]**
▪ Task 01: Accessing struct members	[30 marks]

References and Additional Material:

- Structure
  https://www.programiz.com/c-programming/c-structures

- Structure & Function
  https://www.programiz.com/c-programming/c-structure-function

- Unions
  https://www.programiz.com/c-programming/c-unions

Lab Time Activity Simulation Log:

• Slot – 01 – 00:00 – 00:15:	Class Settlement
• Slot – 02 – 00:15 – 00:30:	Execute C on Visual Studio
• Slot – 03 – 00:30 – 00:45:	Execute C on Visual Studio
• Slot – 04 – 00:45 – 01:00:	In-Lab Task
• Slot – 05 – 01:00 – 01:15:	In-Lab Task
• Slot – 06 – 01:15 – 01:30:	In-Lab Task
• Slot – 07 – 01:30 – 01:45:	In-Lab Task
• Slot – 08 – 01:45 – 02:00:	In-Lab Task
• Slot – 09 – 02:00 – 02:15:	In-Lab Task
• Slot – 10 – 02:15 – 02:30:	In-Lab Task
• Slot – 11 – 02:30 – 02:45:	Evaluation of Lab Tasks
• Slot – 12 – 02:45 – 03:00:	Discussion on Post-Lab Task