Unit 9

Structure and Union

What is structure?

- A structure is a collection of variables under a single name.
- Arrays allow to define type of variables that can hold several data items of the same kind.
- Similarly structure is another user defined data type available in C that allows to combine data items of different kinds.
- A structure is a convenient way of grouping several pieces of related information together.

What is structure?

- A structure is a convenient tool for handling a group of logically related data items.

- For ex: name, roll, fee, marks, address, gender and phone are related information of a student.

- To store information about a student, we would require to store the name of the student which is array of characters, roll of student which is integer type of data and so on.

- These attributes of students can be grouped into a single entity, ***student***. Here, student is known as structure which organizes different data times in a more meaningful way.

Defining a structure

- Syntax:

  struct *structure_name*{

  data_type member_variable1;

  data_type member_variable1;

  data_type member_variable1;

  ………..

  ………..

  data_type member_variable1;

  };

- Once ***structure_name*** is declared as new data type, then variables of that type can be declared as

  struct ***structure_name*** structure_variable;

Example for structure

- Let us create a structure named student that has name, roll, marks and remarks as members.

```
struct student{
        char name[20];
        int roll;
        float marks;
        char remarks;
    };
```

- Here, ***student*** is structure name and its members are name, roll, marks and remarks.

- So from before example, student is new data type and various variables of type ***struct student*** can be declared as:

  struct student st;

- Similarly multiple variables can also be declared:

  struct student st1, st2,st3;

- The member variables are accessed using dot (.)operator.

- Each member variable of structure has its own copy of member variables.

- For ex: st1.name is member variable, *name* of st1 structure variable.

  st2.roll is member variable, roll of st2 structure variable.

- We can also write as:

```
struct student{
        char name[20];
        int roll;
        float marks;
        char remarks;
 }st1, st2;
```

DIY

- Create structure date which has property day, year and month.
- Create structure book which has property …, …. ,….,….
- Create structure employee which has property …, …. ,….,….
- Create structure account which has property …, …. ,….,….
- Create structure department store which has property …, …. ,….,….

YOU HAVE 5 MINUTES!!!!

examples

```
struct date {
   int day;
   int month;
   int year;
}
```

```
struct book{
   char title[25];
   char author[20];
   int pages;
   float price;
}b1,b2;
```

```
struct epmployee{
   int emp_id;
   char emp_name[20];
   int age;
   char gender;
   float salary;
}b1,b2;
```

```
struct account{
   int acc_no;
   char acc_type[20];
   float balance;
};
```

Structure Initialization

- The values to be initialized must appear in order as in the definition of structure within braces and separated by commas.

- C does not allow the initialization of individual structure members within its definition.

```
struct student{
            char name[20];
            int roll;
            float marks;
      };
```

- A variable of this type can be initialized during its declaration as shown below

```
struct student st={“Ram”, 12, 55.4};
```

- This line is similar to

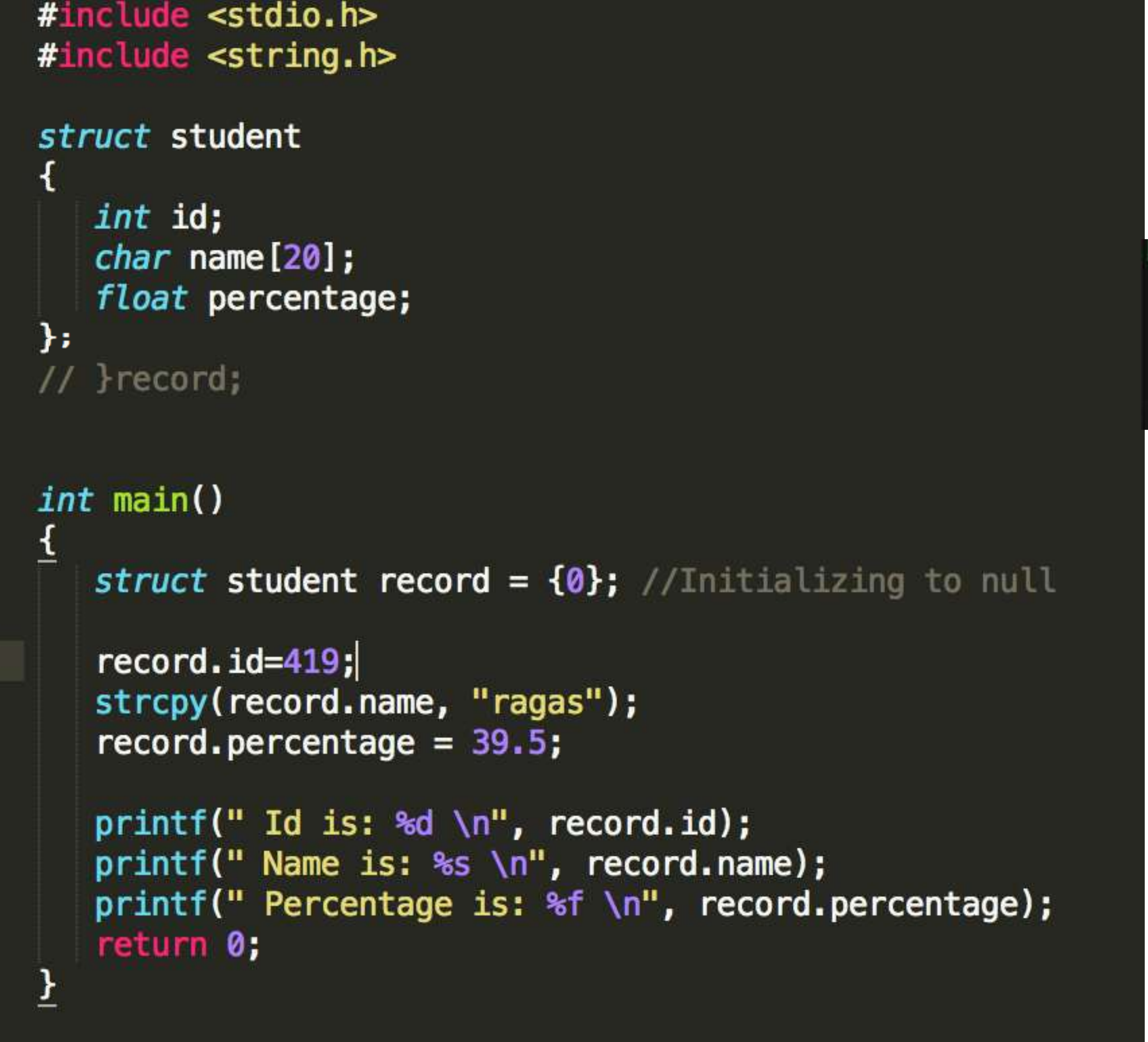
```
struct student st;
st.name=“Ram”;
st.roll=12;
st.marks=55.4;
```

How Structure Elements Are Stored

- The elements of a structure are always stored in contiguous memory location.
- A structure variable reserves number of bytes equal to sum of bytes needed to each of its members.
- For ex:

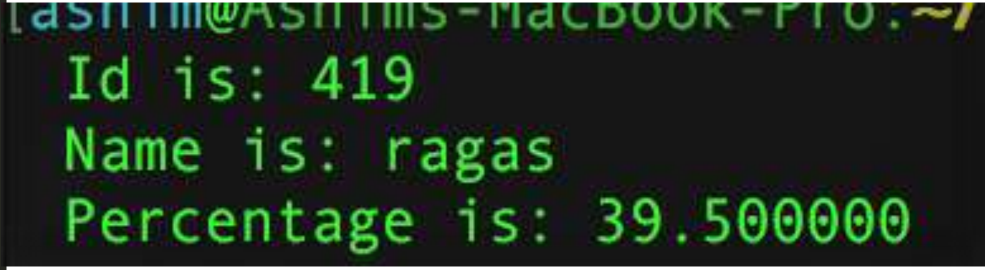
```
struct student{
        int roll;
        float marks;
        char remarks;
        }st;
```

- Here, structure ***student's*** variables ***st*** takes 7 bytes in memory as its member variable roll needs 2bytes, marks needs 4 bytes and remarks needs 1 byte.

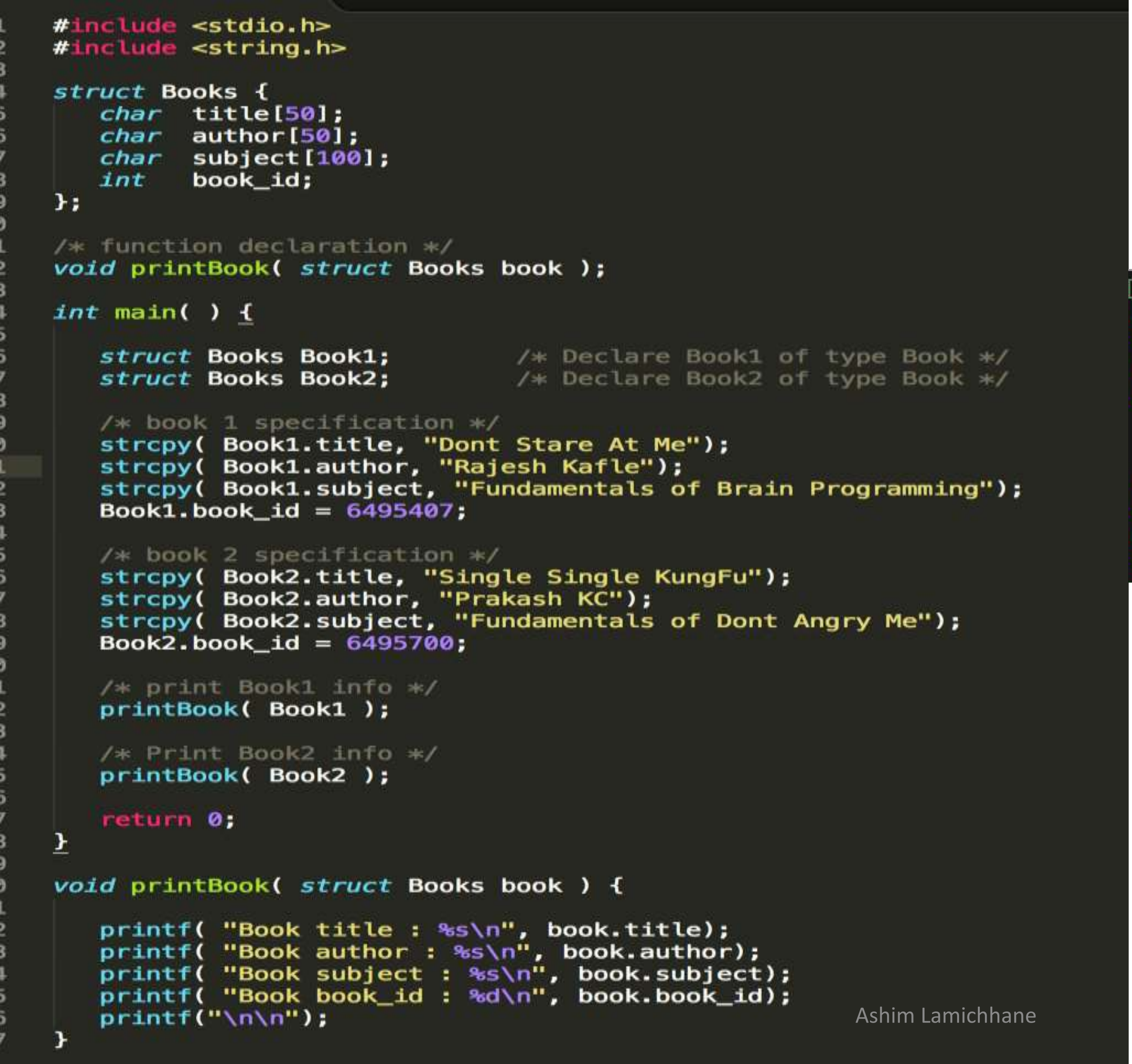


```c
#include <stdio.h>
#include <string.h>

struct student
{
    int id;
    char name[20];
    float percentage;
};
// }record;


int main()
{
    struct student record = {0}; //Initializing to null

    record.id=419;
    strcpy(record.name, "ragas");
    record.percentage = 39.5;

    printf(" Id is: %d \n", record.id);
    printf(" Name is: %s \n", record.name);
    printf(" Percentage is: %f \n", record.percentage);
    return 0;
}
```

OUTPUT

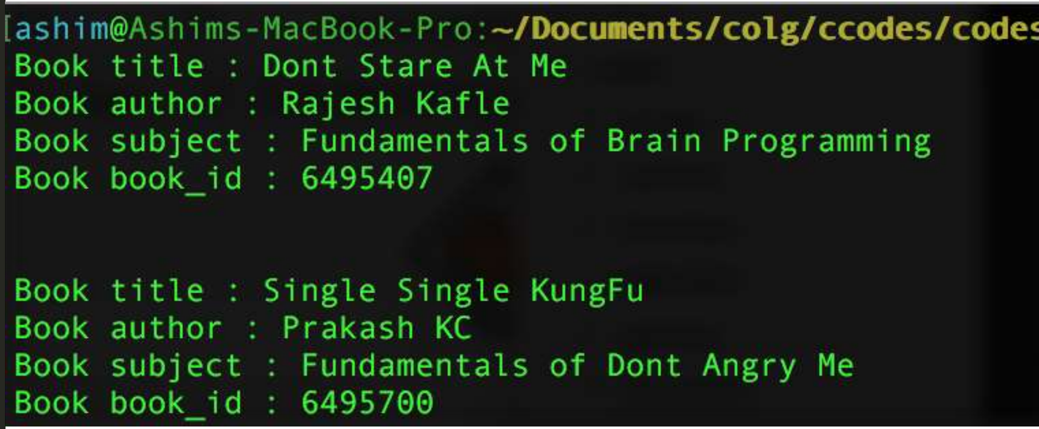


```
 Id is: 419
 Name is: ragas
 Percentage is: 39.500000
```



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

struct Books {
    char  title[50];
    char  author[50];
    char  subject[100];
    int   book_id;
};

/* function declaration */
void printBook( struct Books book );

int main( ) {

    struct Books Book1;        /* Declare Book1 of type Book */
    struct Books Book2;        /* Declare Book2 of type Book */

    /* book 1 specification */
    strcpy( Book1.title, "Dont Stare At Me");
    strcpy( Book1.author, "Rajesh Kafle");
    strcpy( Book1.subject, "Fundamentals of Brain Programming");
    Book1.book_id = 6495407;

    /* book 2 specification */
    strcpy( Book2.title, "Single Single KungFu");
    strcpy( Book2.author, "Prakash KC");
    strcpy( Book2.subject, "Fundamentals of Dont Angry Me");
    Book2.book_id = 6495700;

    /* print Book1 info */
    printBook( Book1 );

    /* Print Book2 info */
    printBook( Book2 );

    return 0;
}

void printBook( struct Books book ) {

    printf( "Book title : %s\n", book.title);
    printf( "Book author : %s\n", book.author);
    printf( "Book subject : %s\n", book.subject);
    printf( "Book book_id : %d\n", book.book_id);
    printf("\n\n");
}
```

Output



```
[ashim@Ashims-MacBook-Pro:~/Documents/colg/ccodes/codes
Book title : Dont Stare At Me
Book author : Rajesh Kafle
Book subject : Fundamentals of Brain Programming
Book book_id : 6495407


Book title : Single Single KungFu
Book author : Prakash KC
Book subject : Fundamentals of Dont Angry Me
Book book_id : 6495700
```

DIY

- Create a structure named student that has name, roll, marks and remarks as members. WAP to read and display data entered by the user.

- Create a structure named employee as members. WAP to read and display data entered by the user.

Array Of Structure

- In our previous structure examples, if we want to keep record of 50 students, we have to make 50 structure variables like st1,st2….st50. (WORST Technique)

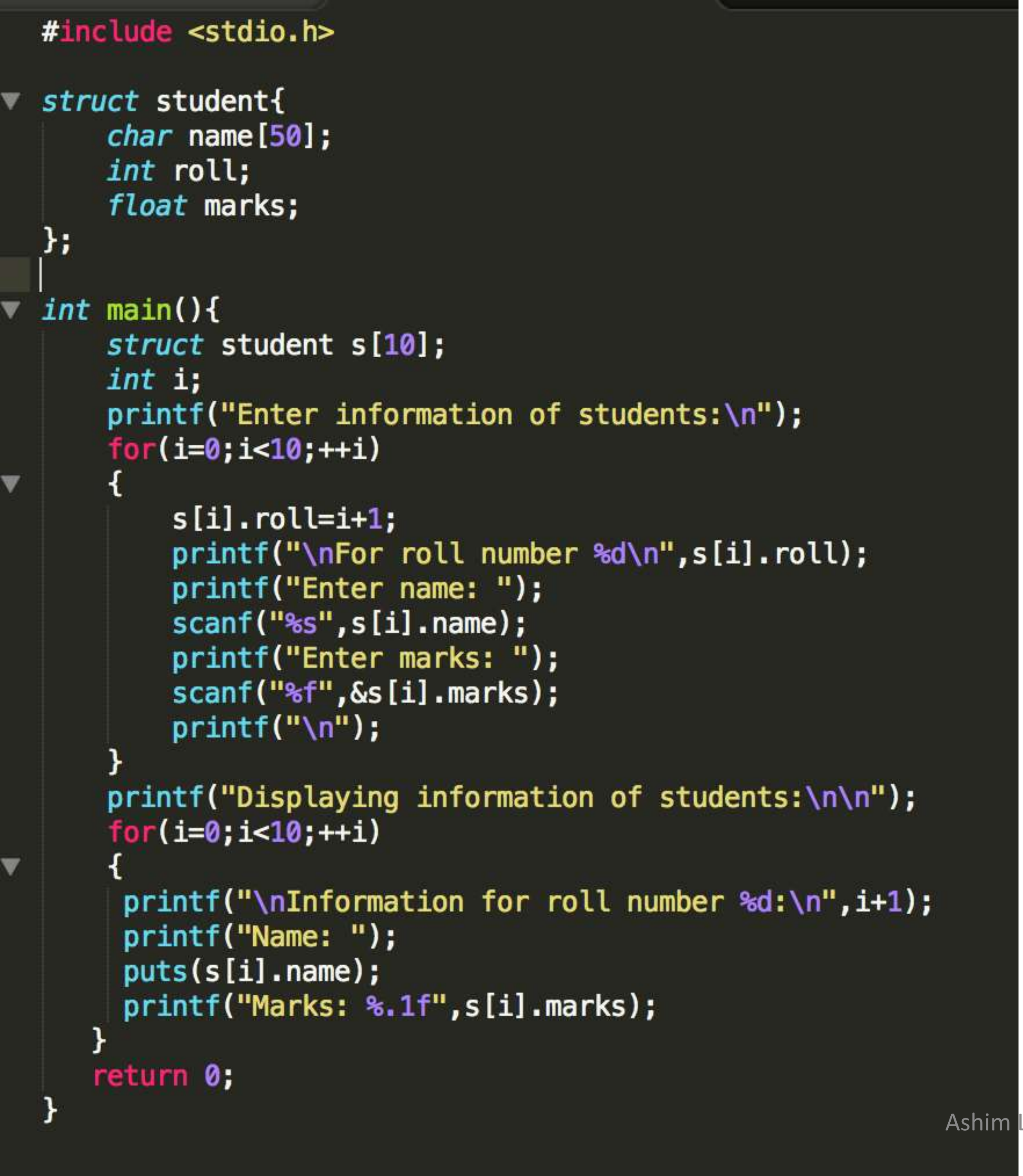
- To tackle this we can use array of structure to store records of 50 students.

- An array of structure can be declared in two ways as illustrated below

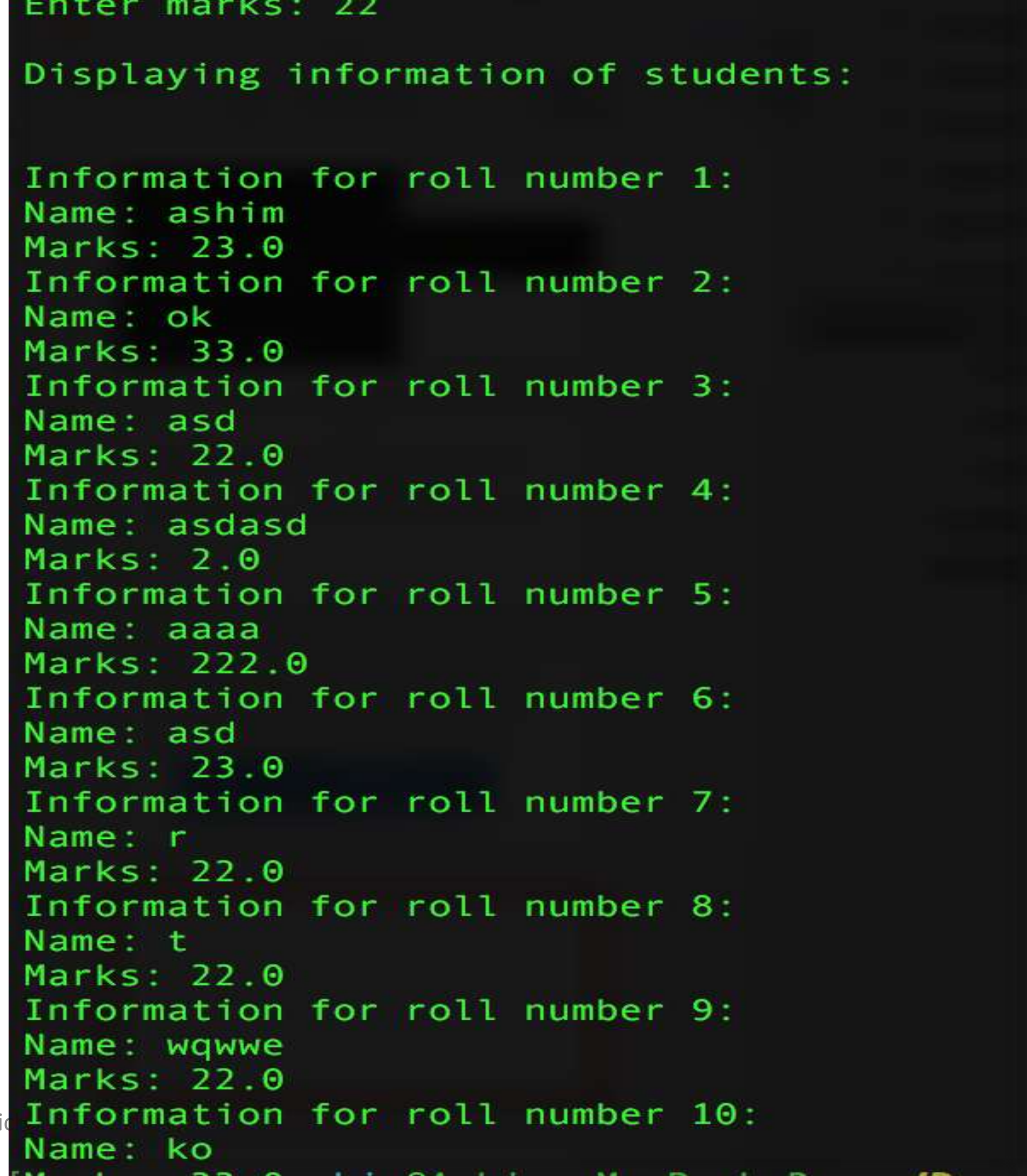
```
struct Employee{
    char name[20];
    int empID;
    float salary;
}emp[10];
```

```
struct Employee{
    char name[20];
    int empID;
    float salary;
};
struct Employee emp[10];
```

Here emp is an array of 10 Employee structures. Each element of the array emp will contain the structure of the type Employee.



```c
#include <stdio.h>

struct student{
    char name[50];
    int roll;
    float marks;
};

int main(){
    struct student s[10];
    int i;
    printf("Enter information of students:\n");
    for(i=0;i<10;++i)
    {
        s[i].roll=i+1;
        printf("\nFor roll number %d\n",s[i].roll);
        printf("Enter name: ");
        scanf("%s",s[i].name);
        printf("Enter marks: ");
        scanf("%f",&s[i].marks);
        printf("\n");
    }
    printf("Displaying information of students:\n\n");
    for(i=0;i<10;++i)
    {
     printf("\nInformation for roll number %d:\n",i+1);
     printf("Name: ");
     puts(s[i].name);
     printf("Marks: %.1f",s[i].marks);
   }
   return 0;
}
```



```
Enter marks: 22

Displaying information of students:


Information for roll number 1:
Name: ashim
Marks: 23.0
Information for roll number 2:
Name: ok
Marks: 33.0
Information for roll number 3:
Name: asd
Marks: 22.0
Information for roll number 4:
Name: asdasd
Marks: 2.0
Information for roll number 5:
Name: aaaa
Marks: 222.0
Information for roll number 6:
Name: asd
Marks: 23.0
Information for roll number 7:
Name: r
Marks: 22.0
Information for roll number 8:
Name: t
Marks: 22.0
Information for roll number 9:
Name: wqwwe
Marks: 22.0
Information for roll number 10:
Name: ko
```

Initializing array of structure

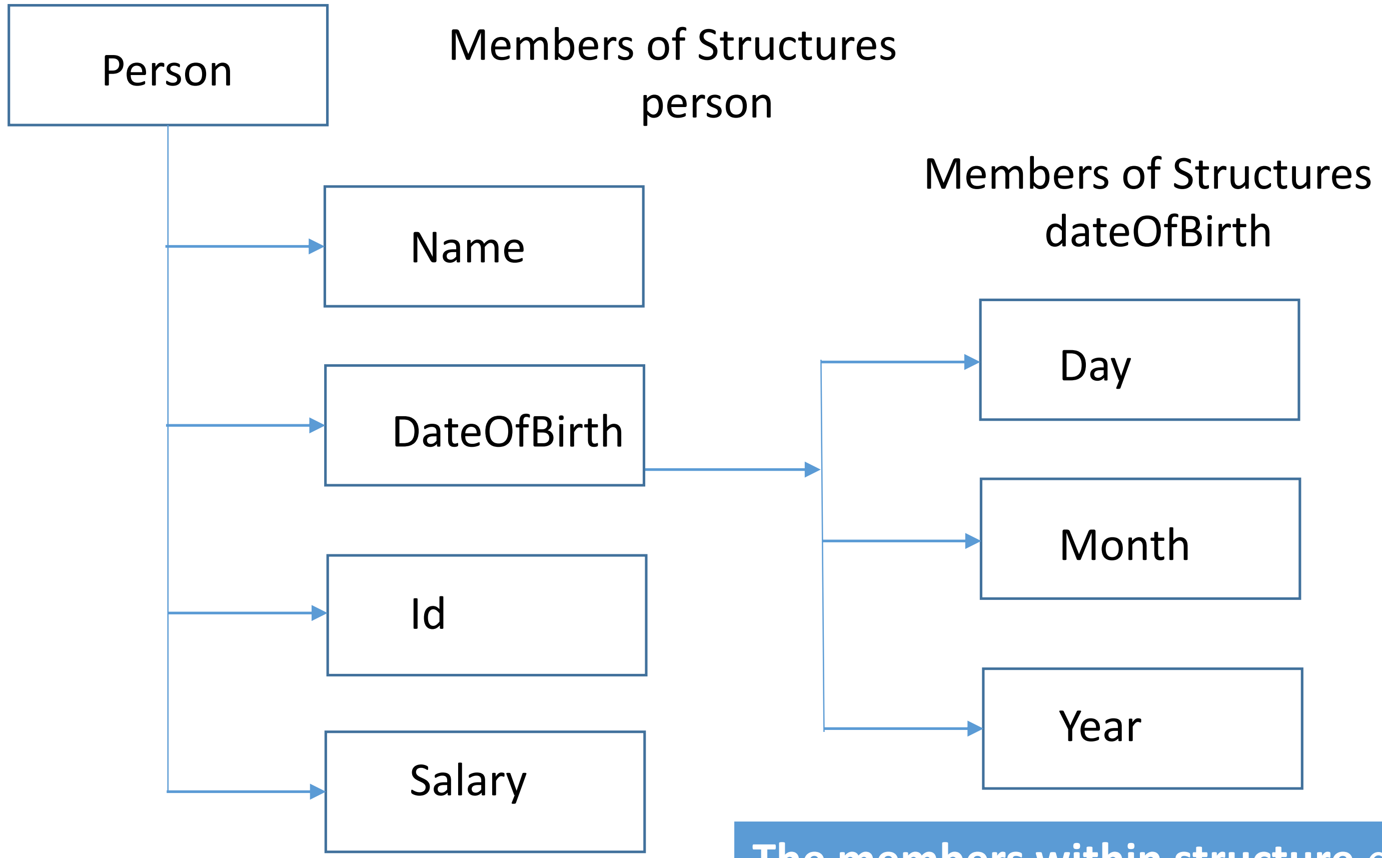
```
struct student{
        char name[50];
        int roll;
        float marks;
};
struct student stu[5]={
        “Ram”, 200,150.5;
        “Rahim”, 220,250.5;
        “kasprowich”, 210,250.4;
        “Smith”, 300,350.1;
        “Steven”, 400,120.5;
}
```

Structure within structure (Nested Structure)

- Structure written inside another structure is called as nesting of two structures.
- Nested Structures are allowed in C Programming Language.
- We can write one Structure inside another structure as member of another structure.
- Ex:

```
struct date
            {
               int date;
               int month;
               int year;
            };
```

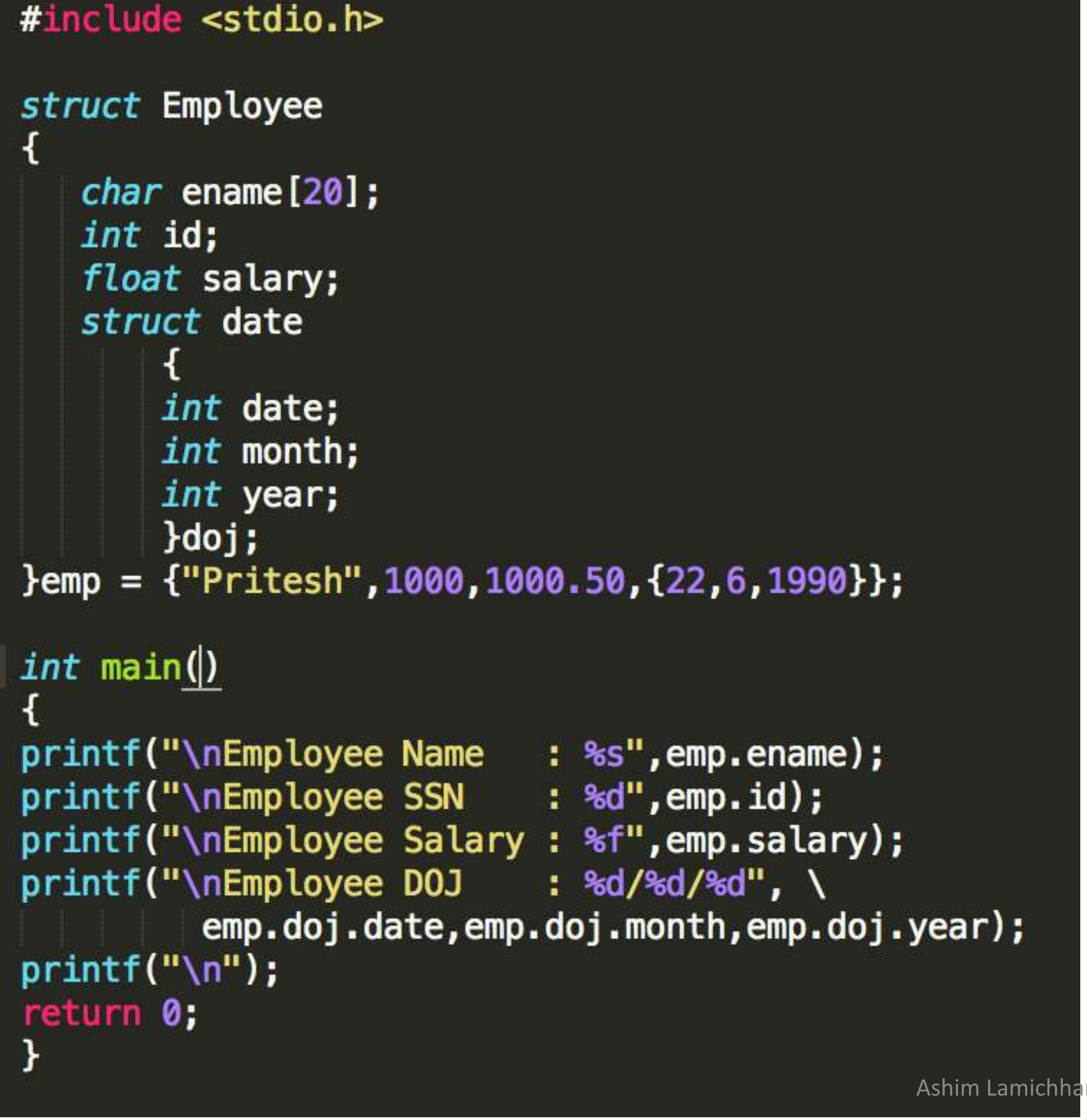
```
struct Employee
             {
             char ename[20];
             int ssn;
             float salary;
             struct date doj;
           }emp1;
```


Person
Members of Structures person
Name
DateOfBirth
Id
Salary
Members of Structures dateOfBirth
Day
Month
Year

```
struct date{

    int day;
    int month;
    int year;
};
```

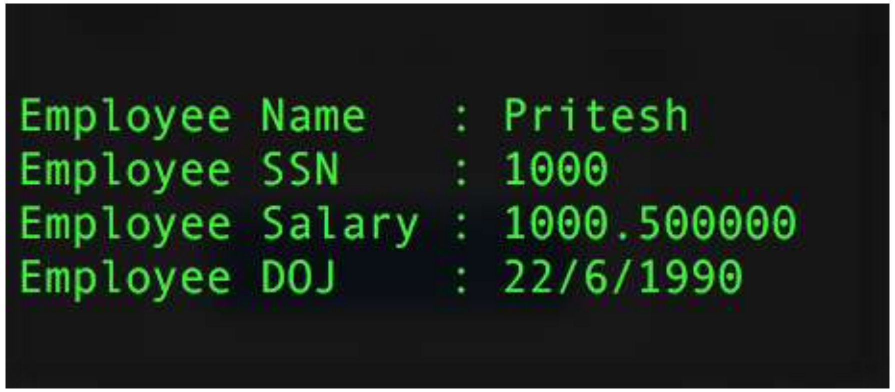
```
struct person{

char name[20];
int id;
struct date dateOfBirthday;
float salary;
}p;
```

The members within structure *date* can be accessed as:	The members within *person* structure are accessed as:
p.dateOfBirthday.day p.dateOfBirthday.month p.dateOfBirthday.year	p.Name p.Id p.salary

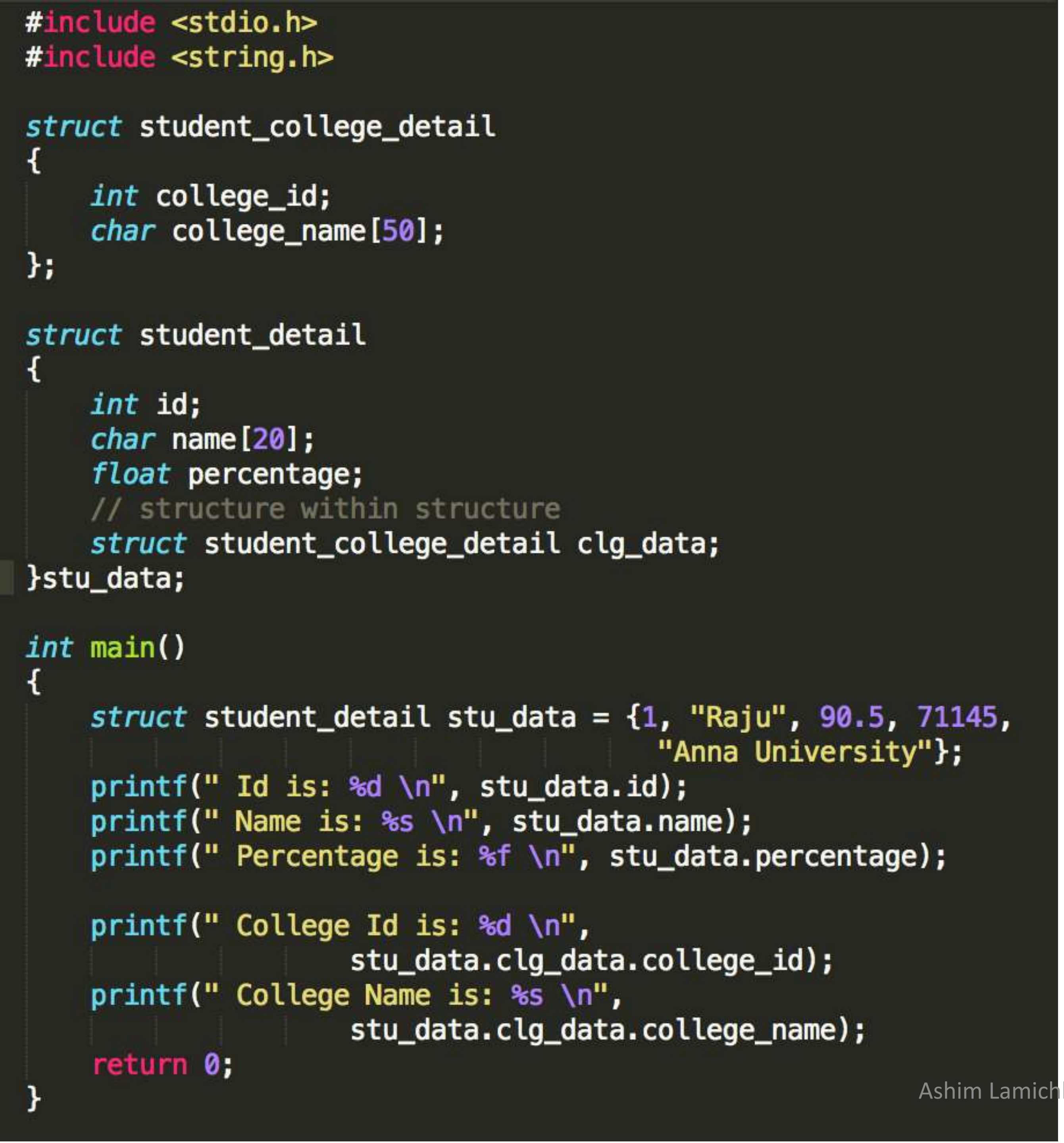


```c
#include <stdio.h>

struct Employee
{
    char ename[20];
    int id;
    float salary;
    struct date
        {
        int date;
        int month;
        int year;
        }doj;
}emp = {"Pritesh",1000,1000.50,{22,6,1990}};

int main()
{
printf("\nEmployee Name   : %s",emp.ename);
printf("\nEmployee SSN    : %d",emp.id);
printf("\nEmployee Salary : %f",emp.salary);
printf("\nEmployee DOJ    : %d/%d/%d", \
         emp.doj.date,emp.doj.month,emp.doj.year);
printf("\n");
return 0;
}
```

Output

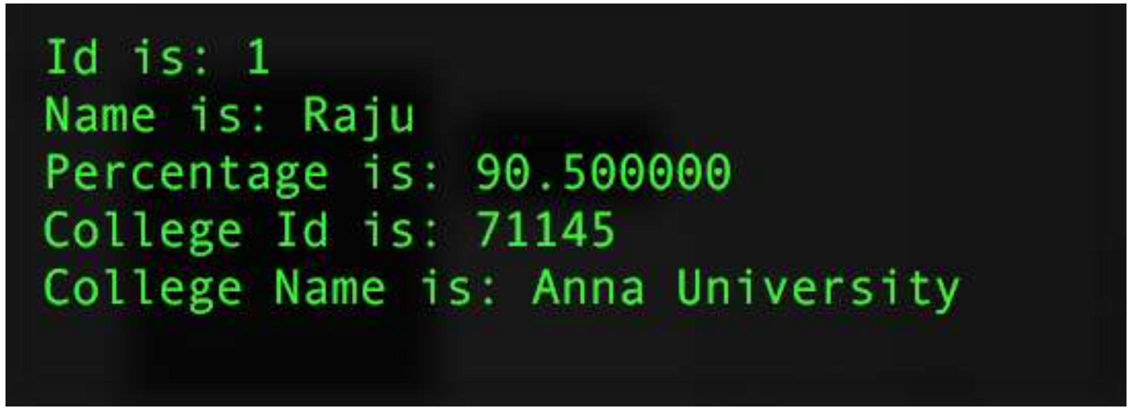


```
Employee Name   : Pritesh
Employee SSN    : 1000
Employee Salary : 1000.500000
Employee DOJ    : 22/6/1990
```



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

struct student_college_detail
{
    int college_id;
    char college_name[50];
};

struct student_detail
{
    int id;
    char name[20];
    float percentage;
    // structure within structure
    struct student_college_detail clg_data;
}stu_data;

int main()
{
    struct student_detail stu_data = {1, "Raju", 90.5, 71145,
                                       "Anna University"};
    printf(" Id is: %d \n", stu_data.id);
    printf(" Name is: %s \n", stu_data.name);
    printf(" Percentage is: %f \n", stu_data.percentage);

    printf(" College Id is: %d \n",
                    stu_data.clg_data.college_id);
    printf(" College Name is: %s \n",
                    stu_data.clg_data.college_name);
    return 0;
}
```

OUTPUT



```
Id is: 1
Name is: Raju
Percentage is: 90.500000
College Id is: 71145
College Name is: Anna University
```

Pointer to structure

- Pointers can be used also with structure.
- To store address of a structure type variable, we can define a structure type pointer variable as normal way.
- Let us consider a structure book that has members name, page and price.

```
struct book{
   char name[10];
   int page;
   float price;
}
```

```
struct book b;                // b is a structure variable
struct book *bptr;            // b is a pointer variable of structure type
```

- The declaration for a pointer to structure doesn't allocate any memory for a structure but allocates only for pointer.

- To use structure members through pointer bptr, memory must be allocated for a structure by using function malloc() or by adding declaration and assignment as given below.

- bptr=&b; //here the base address of b can be assigned to bptr pointer.

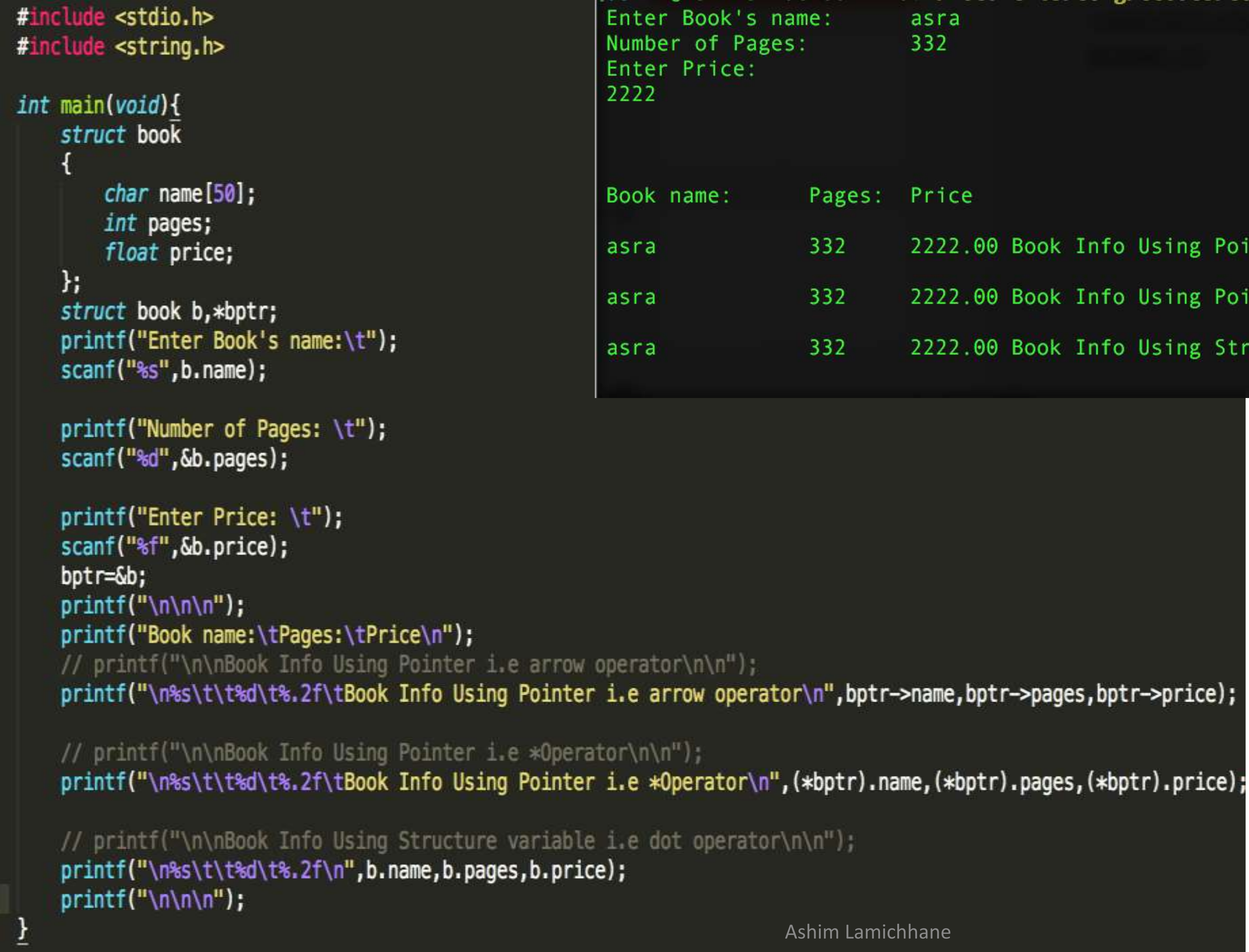
- An individual structure member can be accessed in terms of its corresponding pointer variable by writing

  ptr_variable -> member;

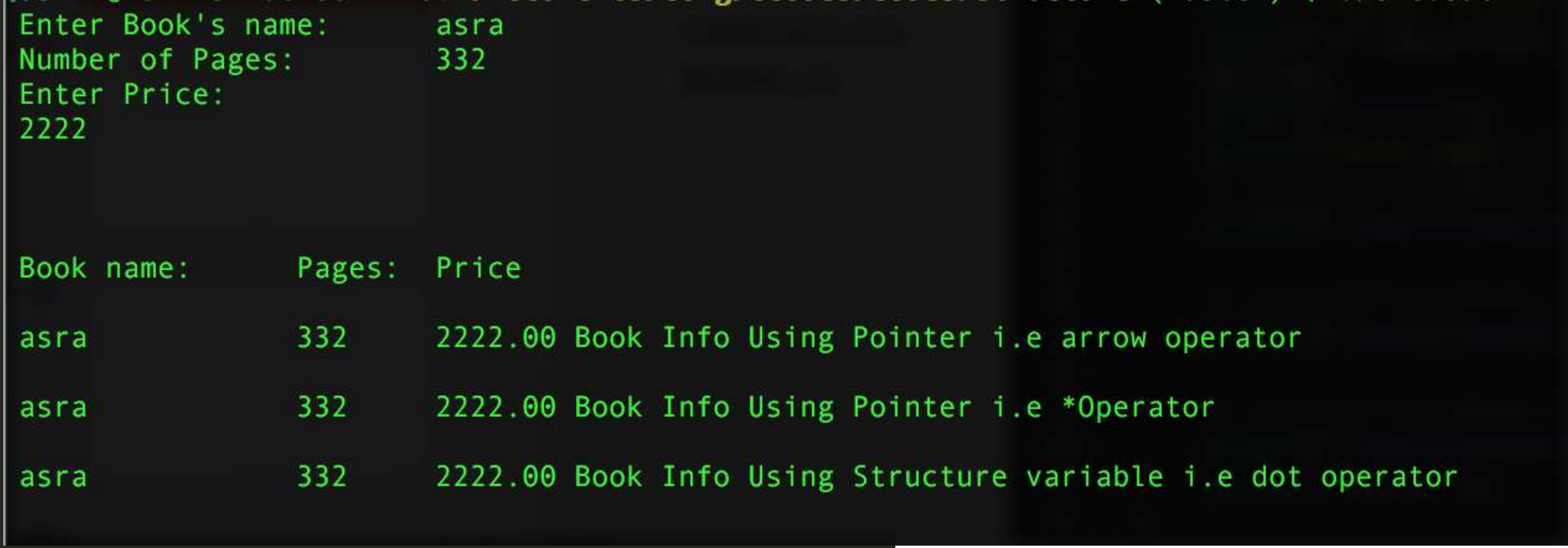
- Here -> is called arrow operator and there must be pointer to the structure on the left side of this operator.

- i.e. Now the members name, pages and price of book can be accessed as

b.name;	OR	bptr-> name	OR	(*bptr).name
b.page;	OR	bptr-> page	OR	(*bptr). page
b.price;	OR	bptr-> price	OR	(*bptr). price



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

int main(void){
    struct book
    {
        char name[50];
        int pages;
        float price;
    };
    struct book b,*bptr;
    printf("Enter Book's name:\t");
    scanf("%s",b.name);

    printf("Number of Pages: \t");
    scanf("%d",&b.pages);

    printf("Enter Price: \t");
    scanf("%f",&b.price);
    bptr=&b;
    printf("\n\n\n");
    printf("Book name:\tPages:\tPrice\n");
    // printf("\n\nBook Info Using Pointer i.e arrow operator\n\n");
    printf("\n%s\t\t%d\t%.2f\tBook Info Using Pointer i.e arrow operator\n",bptr->name,bptr->pages,bptr->price);

    // printf("\n\nBook Info Using Pointer i.e *Operator\n\n");
    printf("\n%s\t\t%d\t%.2f\tBook Info Using Pointer i.e *Operator\n",(*bptr).name,(*bptr).pages,(*bptr).price);

    // printf("\n\nBook Info Using Structure variable i.e dot operator\n\n");
    printf("\n%s\t\t%d\t%.2f\n",b.name,b.pages,b.price);
    printf("\n\n\n");
}
```

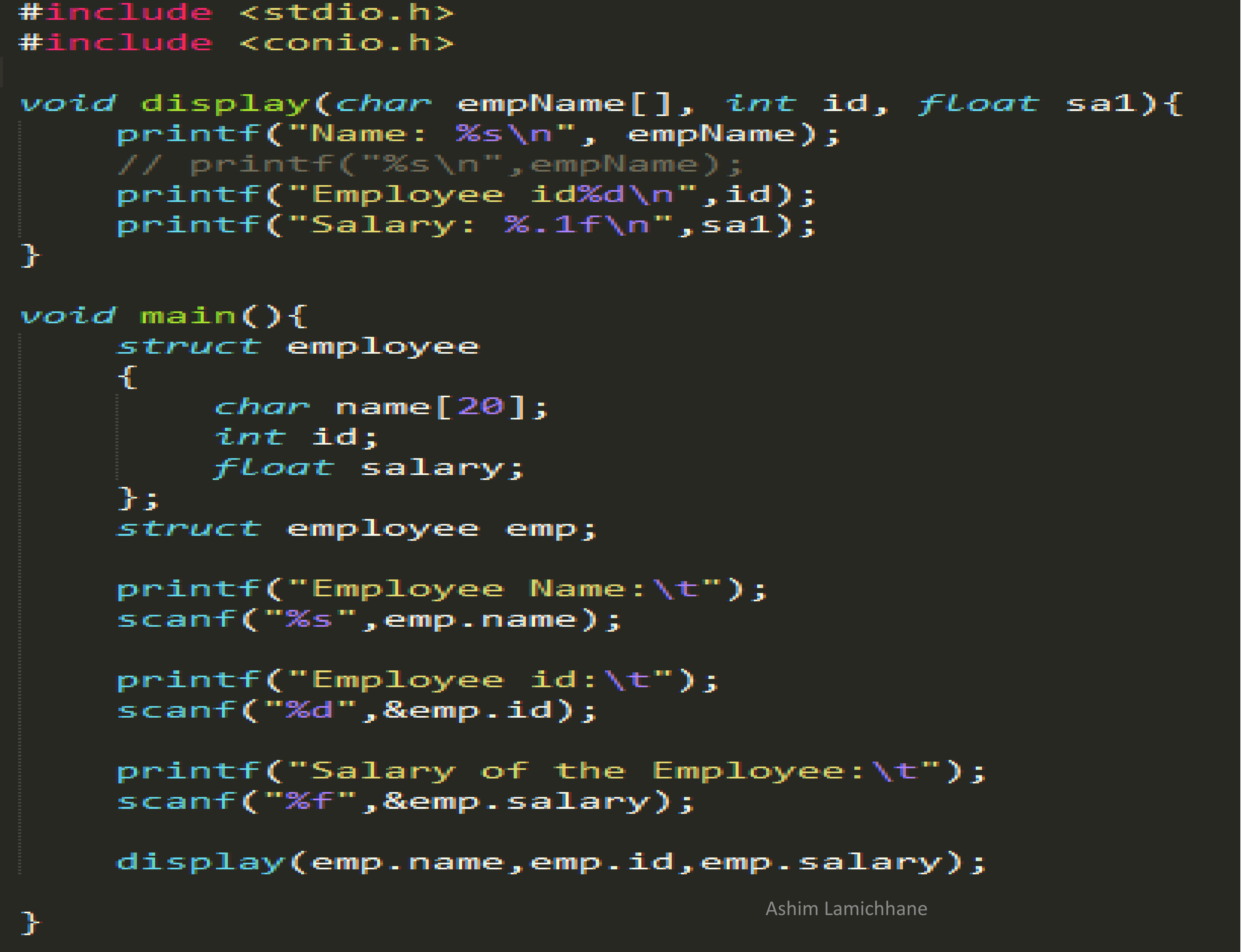


```
Enter Book's name:      asra
Number of Pages:        332
Enter Price:
2222



Book name:      Pages:  Price

asra            332     2222.00 Book Info Using Pointer i.e arrow operator

asra            332     2222.00 Book Info Using Pointer i.e *Operator

asra            332     2222.00 Book Info Using Structure variable i.e dot operator
```

Function and Structure

- Like an ordinary variable, a structure variable can also be passed to a function.

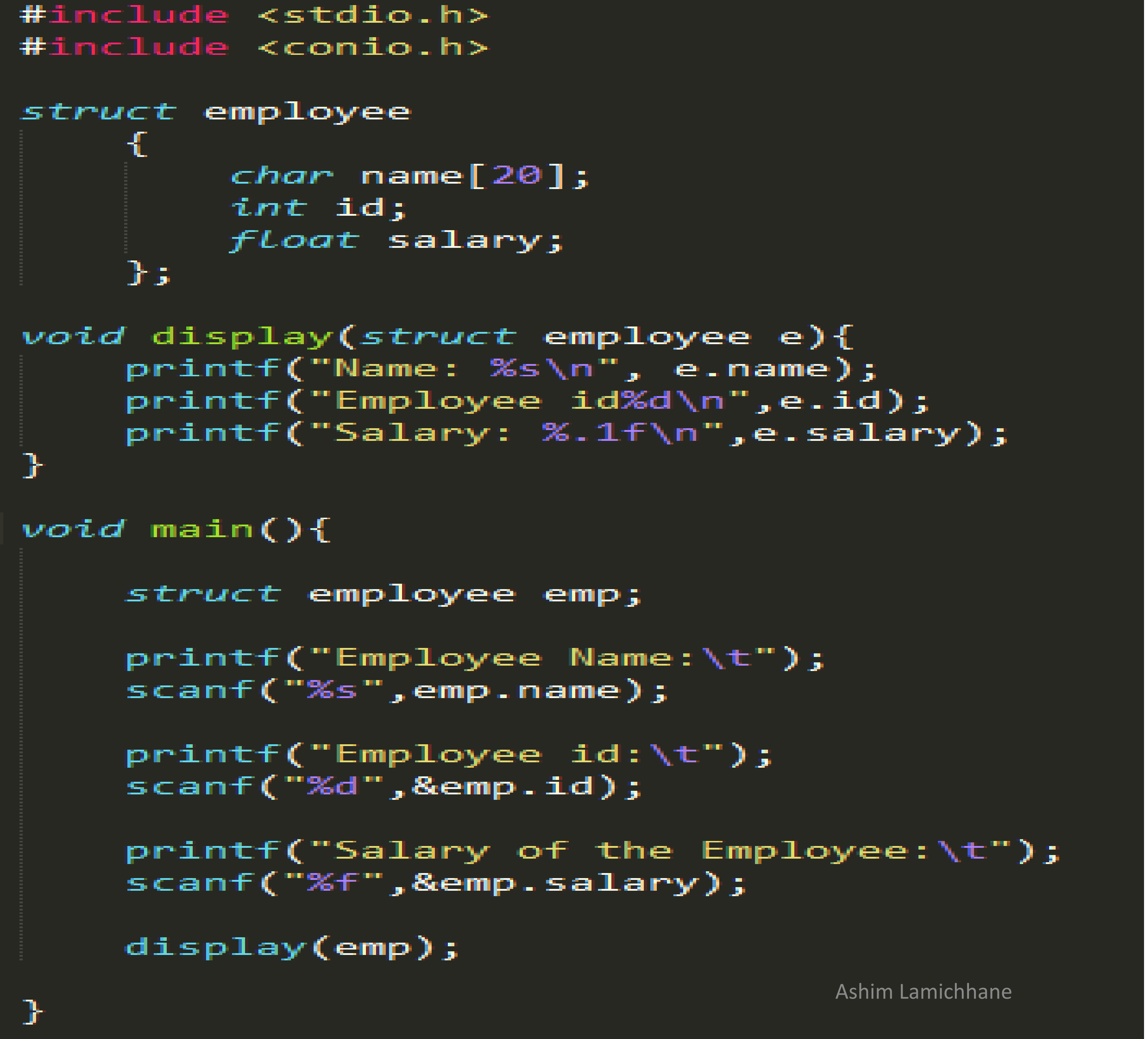
1. Passing structure members to functions
2. Passing whole structure to functions
3. Passing structure pointer to functions
4. Passing array of structure to functions



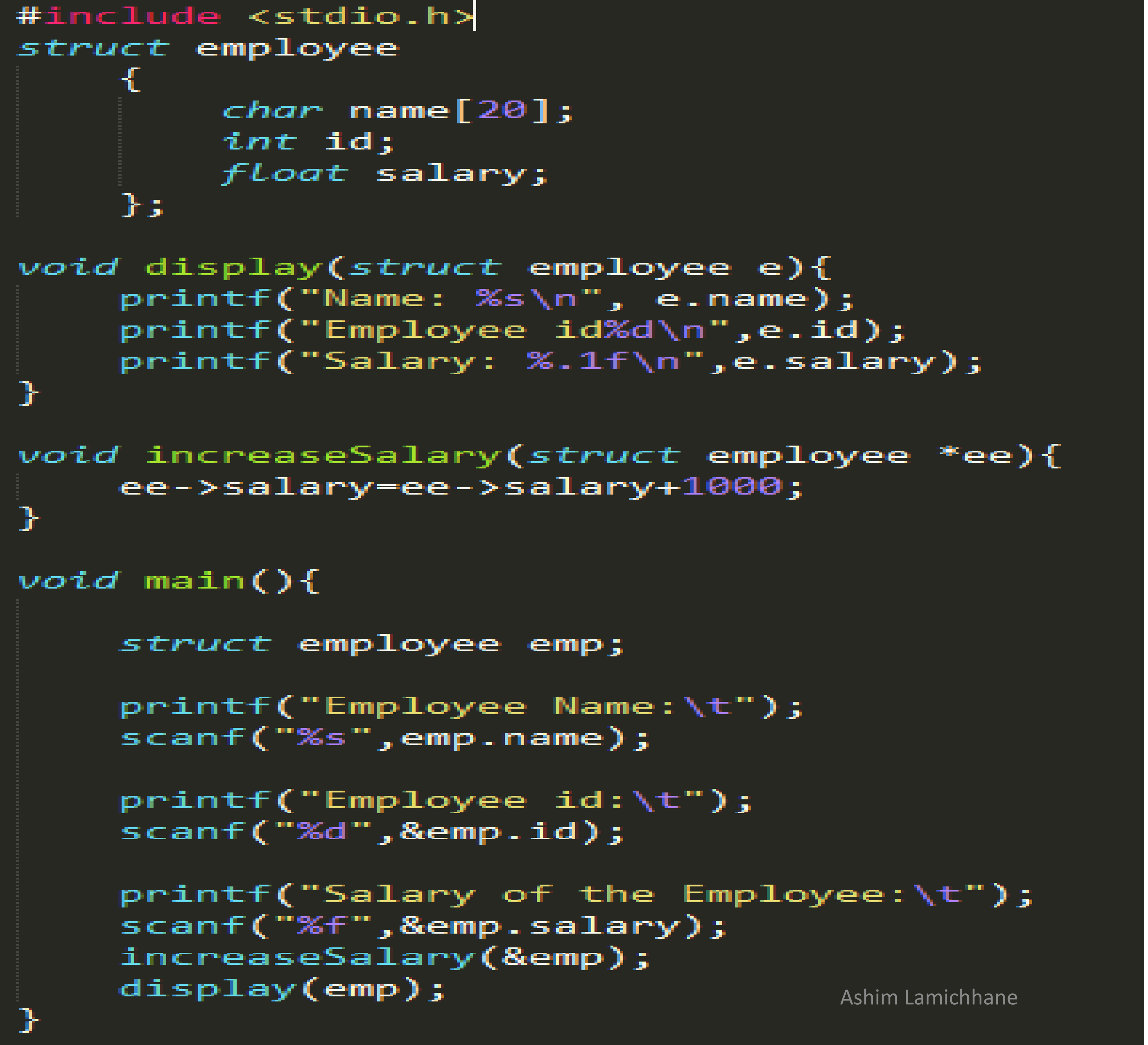
```c
#include <stdio.h>
#include <conio.h>

void display(char empName[], int id, float sal){
    printf("Name: %s\n", empName);
    // printf("%s\n",empName);
    printf("Employee id%d\n",id);
    printf("Salary: %.1f\n",sal);
}

void main(){
    struct employee
    {
        char name[20];
        int id;
        float salary;
    };
    struct employee emp;

    printf("Employee Name:\t");
    scanf("%s",emp.name);

    printf("Employee id:\t");
    scanf("%d",&emp.id);

    printf("Salary of the Employee:\t");
    scanf("%f",&emp.salary);

    display(emp.name,emp.id,emp.salary);

}
```

Passing structure members to functions

Passing whole structure to functions



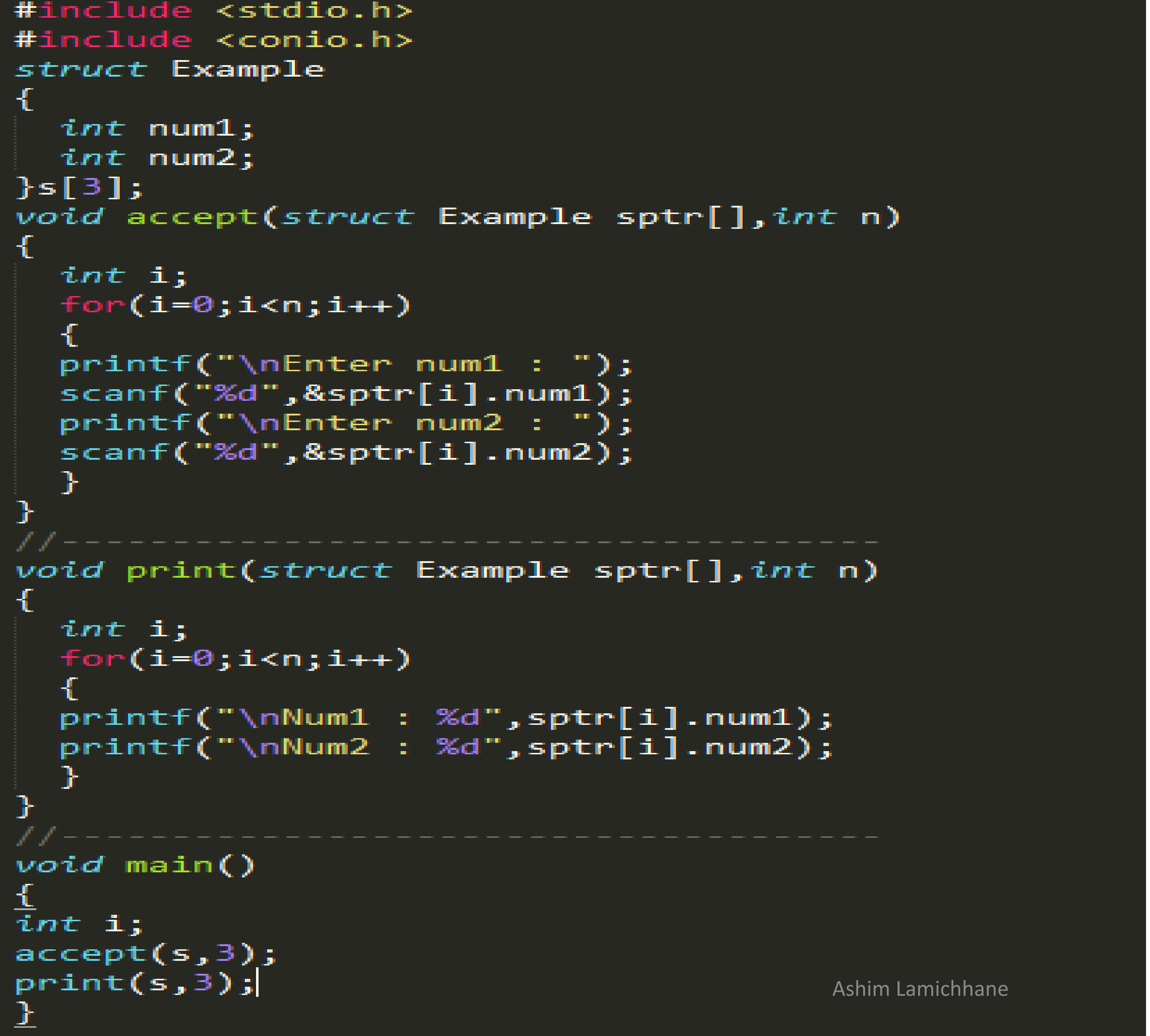
```
#include <stdio.h>
#include <conio.h>

struct employee
    {
        char name[20];
        int id;
        float salary;
    };

void display(struct employee e){
    printf("Name: %s\n", e.name);
    printf("Employee id%d\n",e.id);
    printf("Salary: %.1f\n",e.salary);
}

void main(){

    struct employee emp;

    printf("Employee Name:\t");
    scanf("%s",emp.name);

    printf("Employee id:\t");
    scanf("%d",&emp.id);

    printf("Salary of the Employee:\t");
    scanf("%f",&emp.salary);

    display(emp);

}
```



```
#include <stdio.h>
struct employee
      {
            char name[20];
            int id;
            float salary;
      };

void display(struct employee e){
      printf("Name: %s\n", e.name);
      printf("Employee id%d\n",e.id);
      printf("Salary: %.1f\n",e.salary);
}

void increaseSalary(struct employee *ee){
      ee->salary=ee->salary+1000;
}

void main(){

      struct employee emp;

      printf("Employee Name:\t");
      scanf("%s",emp.name);

      printf("Employee id:\t");
      scanf("%d",&emp.id);

      printf("Salary of the Employee:\t");
      scanf("%f",&emp.salary);
      increaseSalary(&emp);
      display(emp);
}
```

Passing Structure pointer to functions

Passing array of structure to functions



```
#include <stdio.h>
#include <conio.h>
struct Example
{
   int num1;
   int num2;
}s[3];
void accept(struct Example sptr[],int n)
{
   int i;
   for(i=0;i<n;i++)
   {
   printf("\nEnter num1 : ");
   scanf("%d",&sptr[i].num1);
   printf("\nEnter num2 : ");
   scanf("%d",&sptr[i].num2);
   }
}
//------------------------------------
void print(struct Example sptr[],int n)
{
   int i;
   for(i=0;i<n;i++)
   {
   printf("\nNum1 : %d",sptr[i].num1);
   printf("\nNum2 : %d",sptr[i].num2);
   }
}
//------------------------------------
void main()
{
int i;
accept(s,3);
print(s,3);
}
```

UNION

- Unions are similar to structure. Its syntax and use is similar to structure.
- The distinction is that all members within union share the same storage area of computer memory, whereas each member within a structure is assigned its own unique storage.
- Thus unions are used to conserve memory. Since same memory is shared by all members, one variable can reside into memory at a time.
- When another variable is set into memory, the previous is replaced i.e. previous can not persist.
- Thus unions are useful for applications involving multiple members where values need to be assigned to members at the same time.

Unions

- Therefore, although a union may contain many members of different types, it can handle only one member at a time.

- The compiler allocates a piece of storage that is large enough to hold the largest variable type in the union.

- For ex:

```
union student{
        int roll;
        float marks;
};
```

```
union student{
        int roll;
        float marks;
};
```

- Here, the union student has members roll and marks.

- Roll requires 2 bytes in memory and the marks contains 4 bytes in memory.

- As all union members share same memory, the compiler allocates larges memory (i.e 4 bytes in this case)

- The declaration of union and its variable is similar to that of structure.

Example

```
#include <stdio.h>

int main(){
            union student{
                        int roll;
                        float marks;
            };
union student st;
st.roll=455;
printf("Roll=%d\n", st.roll);

st.marks=67;
printf("marks=%d\n", st.marks);
}
```

OUTPUT:

```
Roll =455
Marks=67
```

```
#include <stdio.h>

int main(){
            union student{
                        int roll;
                        float marks;
            };
union student st;
st.roll=455;
st.marks=67;
printf("Roll=%d\n", st.roll);
printf("marks=%d\n", st.marks);
}
```

OUTPUT:

```
Roll =0
Marks=67
```

$\leftarrow$ Here, roll is zero as memory is replaced by another variable st.marks

Difference between Structure and Union

	Structure	Union
1.	Each member within a structure is assigned its own unique storage. It takes more memory than union	All members within union share the same storage area of computer memory. It takes less memory than structure
2.	The amount of memory required to store a structure is the sum of the sizes of all members	The amount of memory required to store an union is same as member that occupies largest memory
3.	All the structure members can be accessed at any point of time	Only one member of union can be accessed at any given time.
4.	Structure is declared as: `struct student{ int roll; float marks; }st;`	Union is declared as: `union student{ int roll; float marks; }st;`

References:

- http://www.studytonight.com/c/structures-in-c.php
- http://www.tutorialspoint.com/cprogramming/c_structures.htm
- http://www.javakode.com/c-programming/c-structures/
- http://c.learncodethehardway.org/book/ex16.html