SNS COLLEGE OF TECHNOLOGY

Coimbatore – 35

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A++' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF INFORMATION TECHNOLOGY

PROGRAMMING IN C AND DATA STRUCTURES

I YEAR - II SEM

UNIT III – ARRAYS AND INTRODUCTION TO DATA STRUCTURES

TOPIC – NESTED STRUCTURES

Introduction – Nested Structure

- The C programming language allows the nesting of the structure.
- This can be done by using one structure into the body of another structure.
- Nesting of multiple separate structures enables the creation of complex data types.
    - For instance, you need to store the information of multiple students.
    - Now the data members of the structure may be student name, student class, and student roll number.
    - Now class can have multiple subparts like standard, section, and stream.
    - Such types of cases can use nested structures to a great extent.

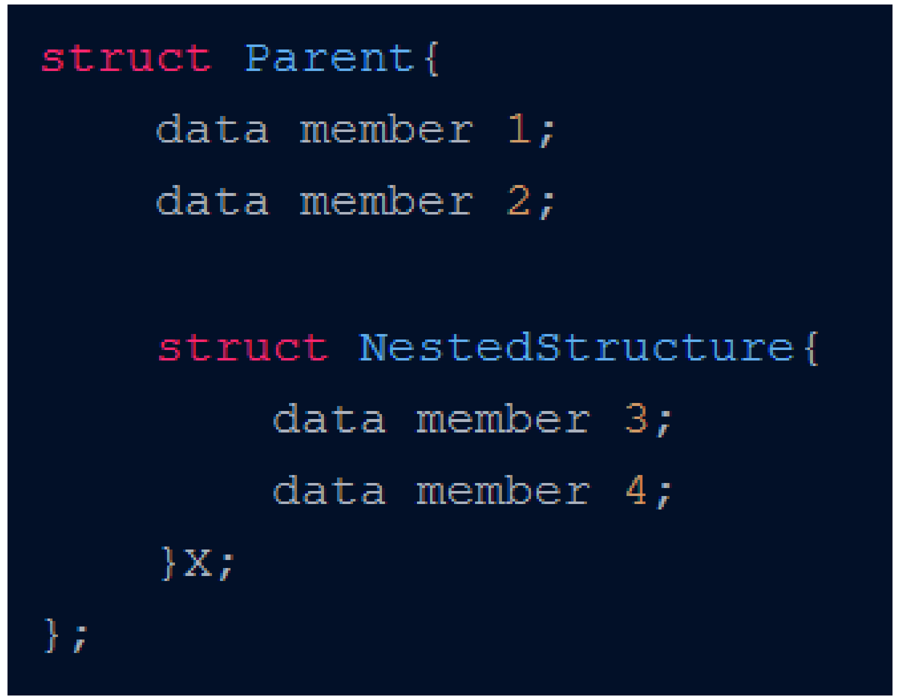
Nested Structure Declaration

A structure in C can be nested in two ways:

By Embedded Structure

As the name suggests, this method is used to embed one structure inside another, i.e., defining one structure in the definition of another structure. Thus using this method, the nested structure will be declared inside the parent structure.

Embedded structure follows the below format:



```
struct Parent{
    data member 1;
    data member 2;

    struct NestedStructure{
        data member 3;
        data member 4;
    }X;
};
```

Example

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

// Declaration of the main structure
struct employee {
  char name[30];
  int employeeid;

  //Declaration of dependent structure
  struct incomeInLPA {
    int base_salary;
    int ctc;
    int bonus;
  }
  inc; // Variable is created which acts as a member to the parent structure
};

int main() {
  struct employee emp;
  return 0;
}
```

1. The structure **incomeInLPA** has been embedded within structure **employee**.
2. Thus parent structure **'employee'** has three variables name, employeeid, and inc.
3. The further variable inc contains **base_salary, CTC**, and **bonus**.
4. The structure **incomeInLPA** is defined inside the parent structure employee.

By Separate Structure

The two structures are created separately in this method. The Dependent structure is used inside the Main/Parent structure by taking a member of the dependent structure type in the definition of the parent structure.

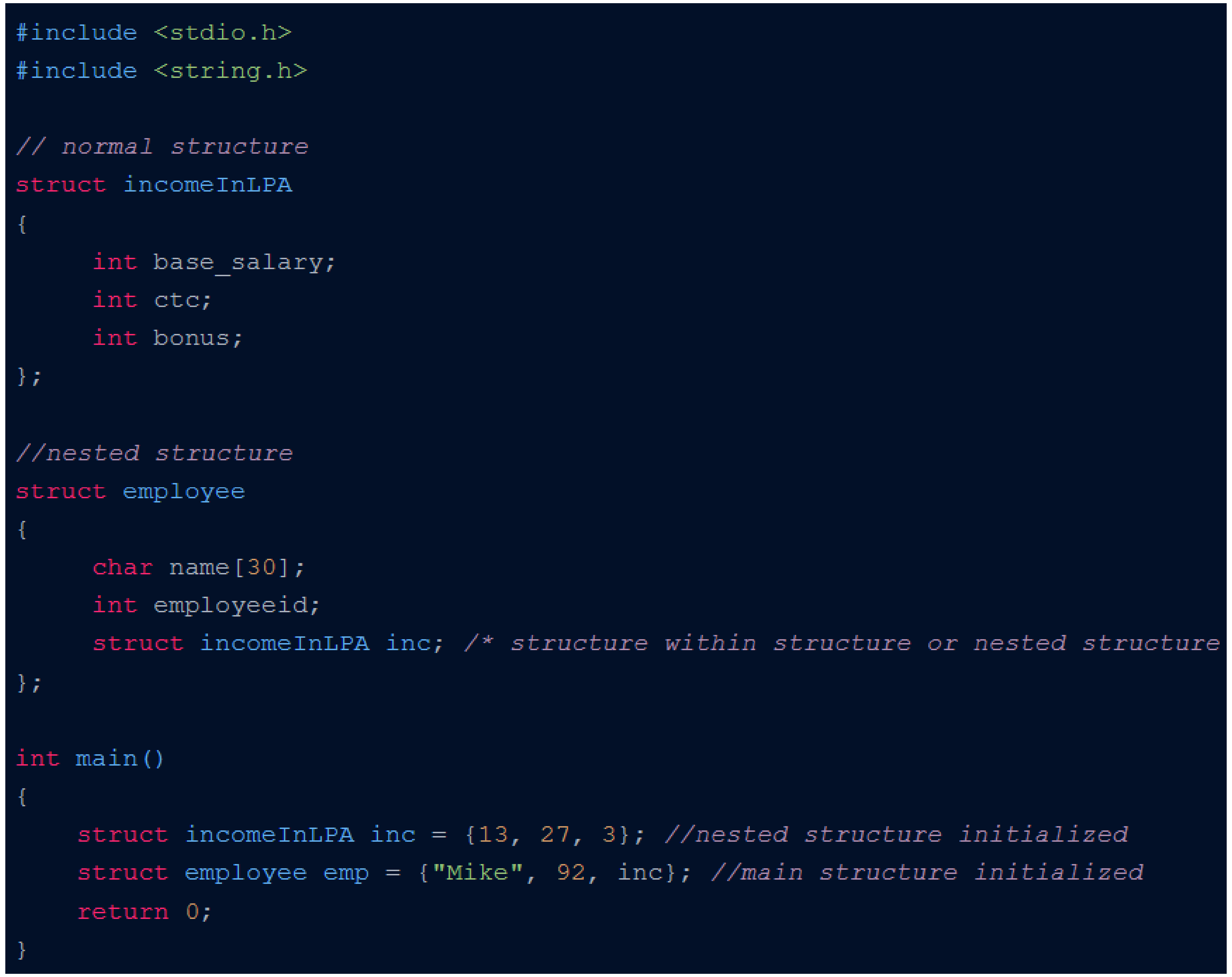
```
struct DependentStructure{
    data member 1;
    data member 2;
};
struct ParentStructure{
    data member 1;
    data member 2;
    struct DependentStructure X;
};
```

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


// Dependent structure
struct incomeInLPA {
  int base_salary;
  int ctc;
  int bonus;
};

// Main structure
struct employee {
  char name[30];
  int employeeid;
  struct incomeInLPA inc; /* structure within structure or nested structure */
};

int main() {
  struct employee emp;
  return 0;
}
```

Initializing Nested Structures

Nested Structures in C can be initialized at the time of declaration. There are two methods by which we can initialize nested structures:

Method 1 : The embed structure is initialized first using the structure variable of that structure, then the parent structure is initialized next using that already initialized member of the embed structure.



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

// normal structure
struct incomeInLPA
{
     int base_salary;
     int ctc;
     int bonus;
};

//nested structure
struct employee
{
     char name[30];
     int employeeid;
     struct incomeInLPA inc; /* structure within structure or nested structure
};

int main()
{
    struct incomeInLPA inc = {13, 27, 3}; //nested structure initialized
    struct employee emp = {"Mike", 92, inc}; //main structure initialized
    return 0;
}
```

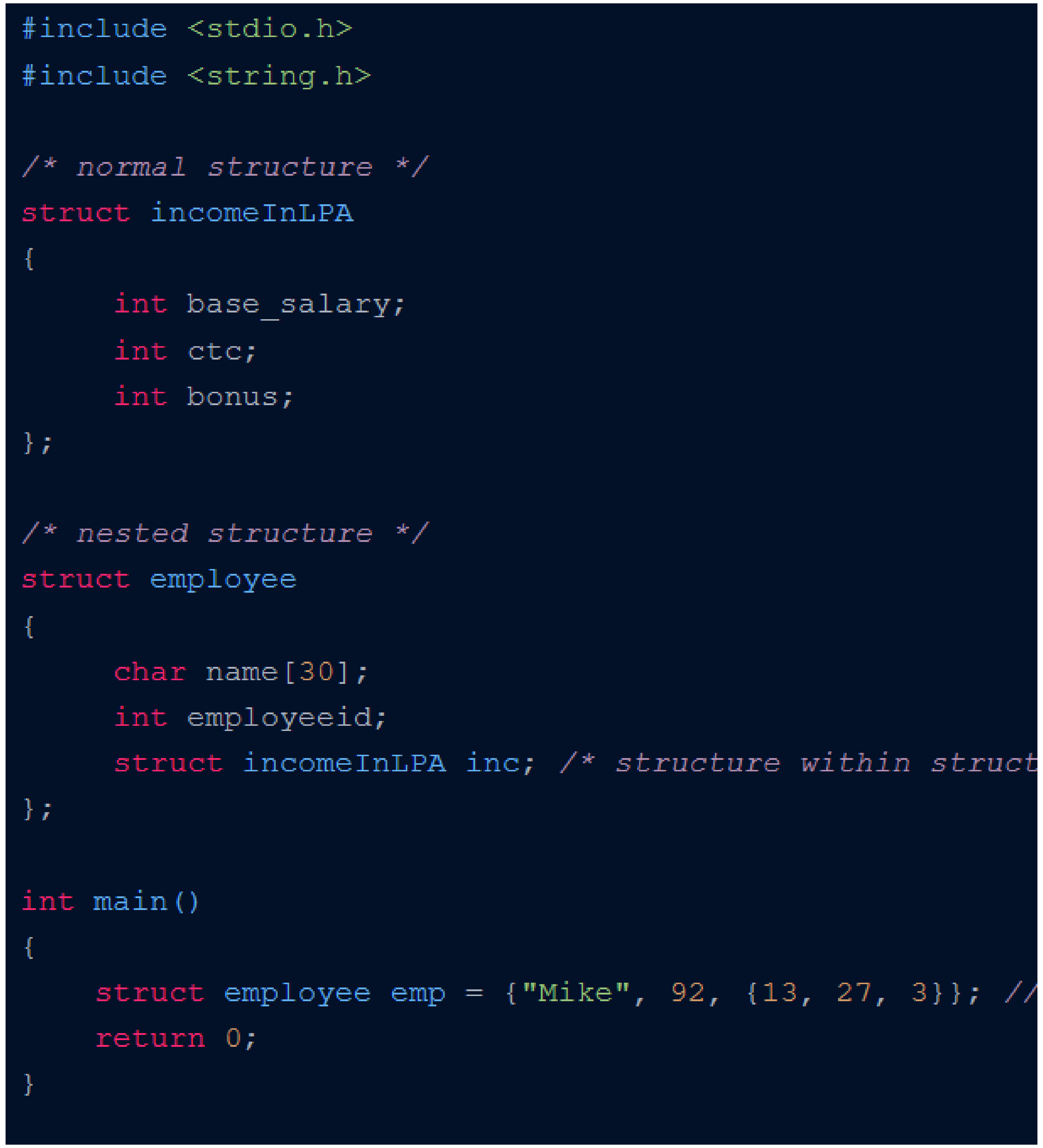
Here, employee is our parent structure and incomeInLPA is our nested/embedded structure.

We initialize our nested structure first and create the structure variable inc.

This structure variable is then used for initializing the parent structure employee variable emp.

Initializing Nested Structures

Method 2 : Both the parent structure and the nested structure are initialized together.



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

/* normal structure */
struct incomeInLPA
{
     int base_salary;
     int ctc;
     int bonus;
};

/* nested structure */
struct employee
{
     char name[30];
     int employeeid;
     struct incomeInLPA inc; /* structure within struct
};

int main()
{
    struct employee emp = {"Mike", 92, {13, 27, 3}}; //
    return 0;
}
```

1. Here, the incomeInLPA structure hasn't been initialized separately. "Mike", and 92 will be stored as name and employeeid in the employee structure.
2. {13, 27,13} will be stored as base-salary, ctc, and bonus in the incomeInLPA structure variable, and all three variables are a member of emp, which is of type employee.t

Accessing Nested Structure

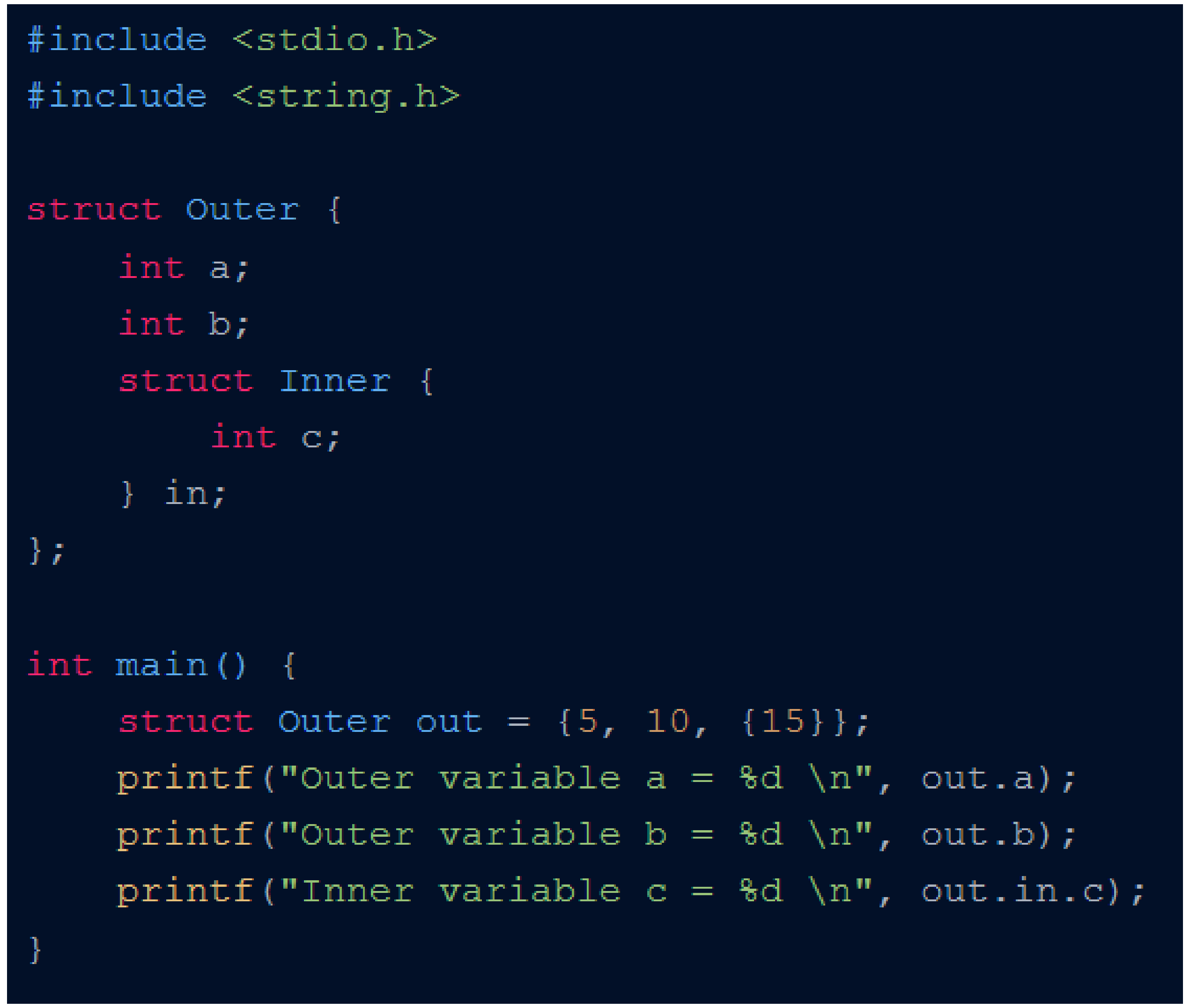
We can access Nested Structure in C in the following two ways:

✓ Using Normal variable.

✓ Using Pointer variable.

Using Normal Variable

Here the data members of the outer structure are accessed using a single dot, while data members of the inner structure are accessed using two dots. Here is a code for accessing nested structure using a normal variable:

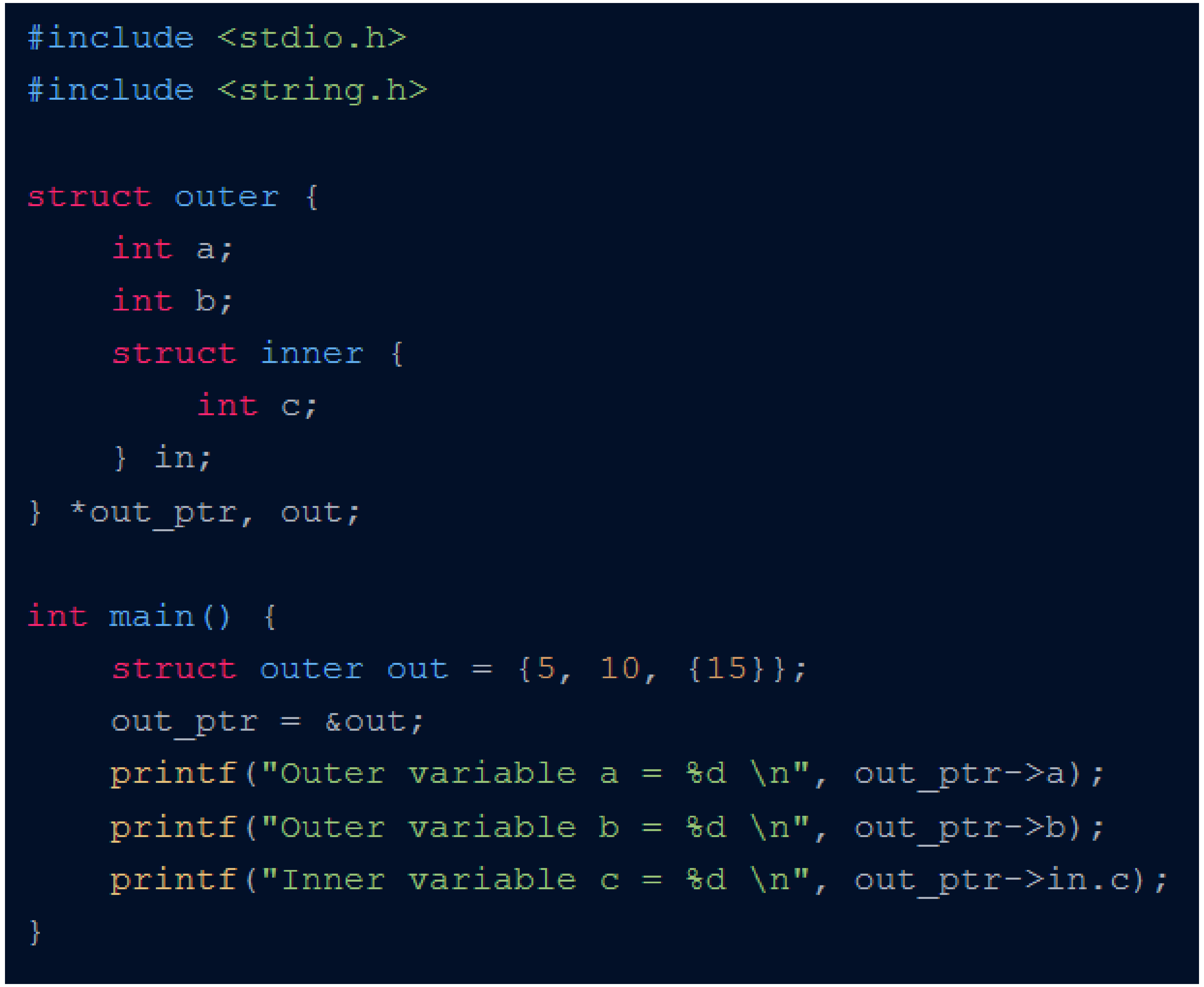


```c
#include <stdio.h>
#include <string.h>

struct Outer {
    int a;
    int b;
    struct Inner {
        int c;
    } in;
};

int main() {
    struct Outer out = {5, 10, {15}};
    printf("Outer variable a = %d \n", out.a);
    printf("Outer variable b = %d \n", out.b);
    printf("Inner variable c = %d \n", out.in.c);
}
```

1. Here the outer structure has 3 variables a, b, and in, while the further variable in contains c in it.
2. We initialise a = 5, b = 10 and c = 15.
3. If we want to access variables a and b, we can access them using a single dot as they are members of the outer structure.
4. However for the inner structure variable c, we have to use two dots as: **outerStructure.innerStructure.variable** i.e, **out.in.c**, to access the variable c.
5. This is because for accessing the inner structure variable c, we first have to access 'in' variable of the outer structure 'out'.
6. Now to access any variable inside 'in' variable, we'll have to specify which variable we want to use using another dot.
7. Thus, we'll have to use **out.in.c, to access the inner variable c**.

Using Pointer Variable

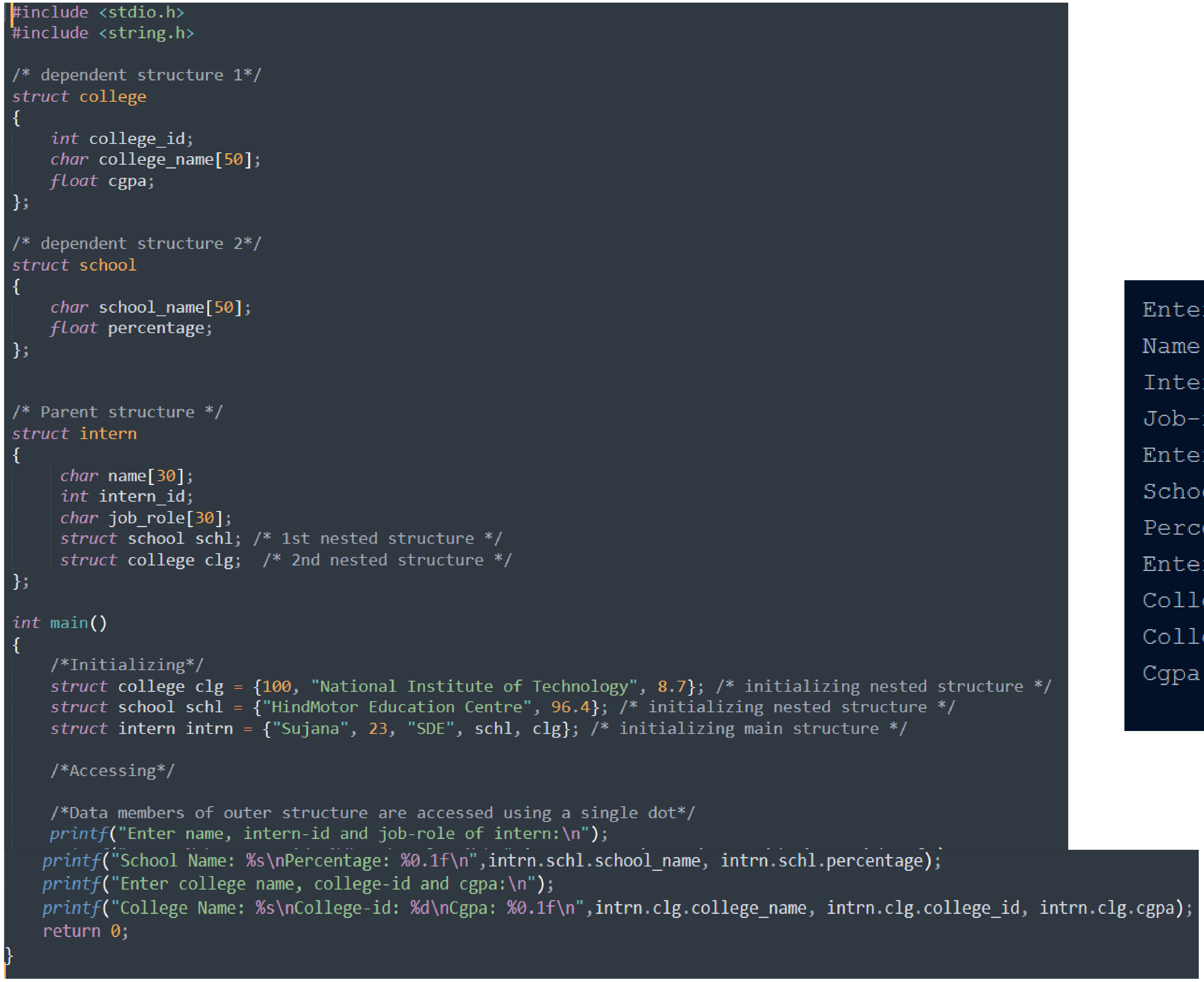
We can also use pointer variables for accessing any data member of a structure. In the below example, we use one pointer variable for the outer structure while a normal variable for the inner structure.



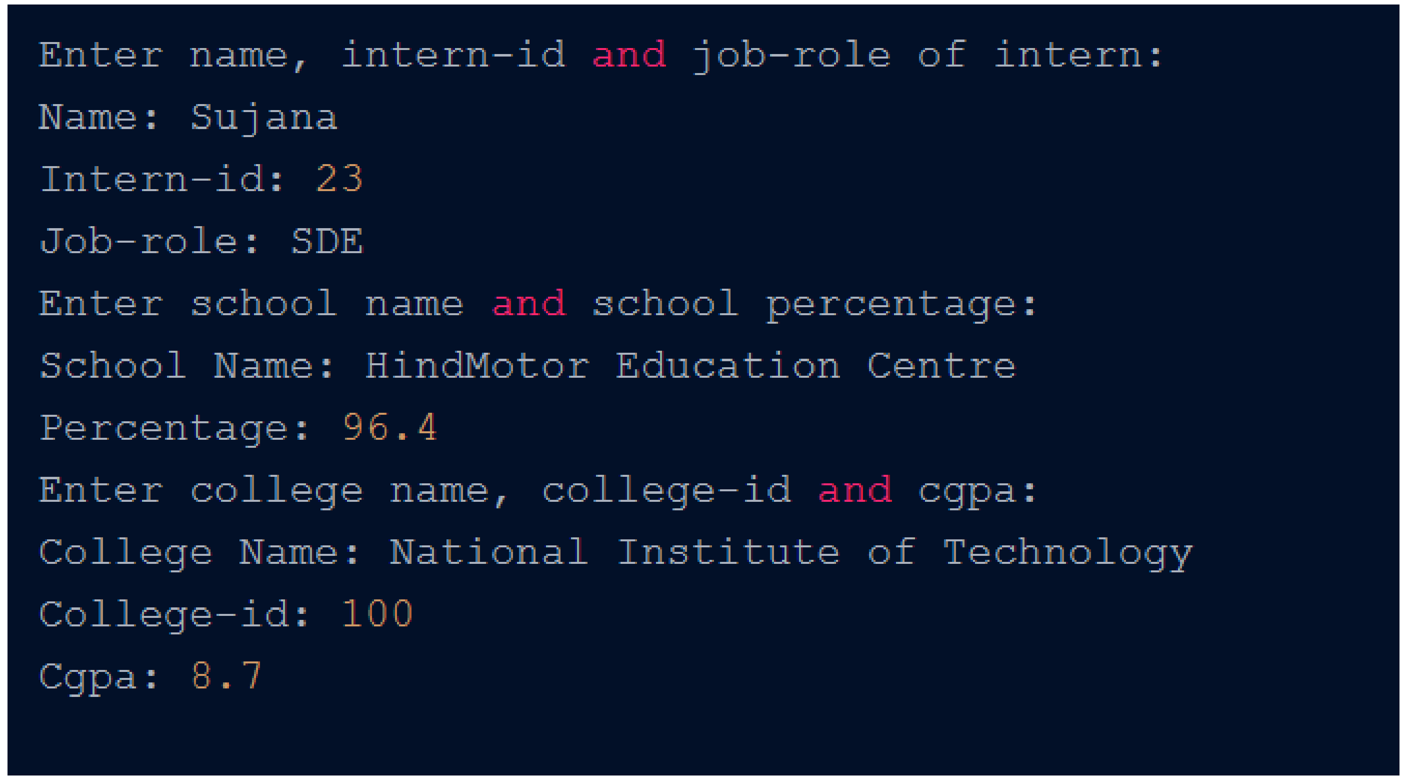
```
#include <stdio.h>
#include <string.h>

struct outer {
    int a;
    int b;
    struct inner {
        int c;
    } in;
} *out_ptr, out;

int main() {
    struct outer out = {5, 10, {15}};
    out_ptr = &out;
    printf("Outer variable a = %d \n", out_ptr->a);
    printf("Outer variable b = %d \n", out_ptr->b);
    printf("Inner variable c = %d \n", out_ptr->in.c);
}
```

1. Here due to the usage of both normal (*in*) and pointer(*out_ptr*) variables, we use both dot(.) and arrow(->) for accessing the inner and outer structure data members respectively.
2. As out_ptr is a pointer variable, so to access the member of the structure pointing by this pointer, we have to use ->, but to access 'c' we have to use dot further as in is a normal variable, and c could be accessed directly by in due to which we write out_ptr->in.c.

Example



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

/* dependent structure 1*/
struct college
{
    int college_id;
    char college_name[50];
    float cgpa;
};

/* dependent structure 2*/
struct school
{
    char school_name[50];
    float percentage;
};


/* Parent structure */
struct intern
{
     char name[30];
     int intern_id;
     char job_role[30];
     struct school schl; /* 1st nested structure */
     struct college clg;  /* 2nd nested structure */
};

int main()
{
    /*Initializing*/
    struct college clg = {100, "National Institute of Technology", 8.7}; /* initializing nested structure */
    struct school schl = {"HindMotor Education Centre", 96.4}; /* initializing nested structure */
    struct intern intrn = {"Sujana", 23, "SDE", schl, clg}; /* initializing main structure */

    /*Accessing*/

    /*Data members of outer structure are accessed using a single dot*/
    printf("Enter name, intern-id and job-role of intern:\n");
    printf("School Name: %s\nPercentage: %0.1f\n",intrn.schl.school_name, intrn.schl.percentage);
    printf("Enter college name, college-id and cgpa:\n");
    printf("College Name: %s\nCollege-id: %d\nCgpa: %0.1f\n",intrn.clg.college_name, intrn.clg.college_id, intrn.clg.cgpa);
    return 0;
}
```



```
Enter name, intern-id and job-role of intern:
Name: Sujana
Intern-id: 23
Job-role: SDE
Enter school name and school percentage:
School Name: HindMotor Education Centre
Percentage: 96.4
Enter college name, college-id and cgpa:
College Name: National Institute of Technology
College-id: 100
Cgpa: 8.7
```

Program Explanation

1. In the given example program, intern is our parent structure and school, and college are our dependent structures.
2. We initialize our dependent structures first, and create the structure variables schl and clg, for structure school and college, respectively.
3. These two structure variables are then used for initializing the parent structure intern variable intrn.
4. While printing the details of an intern, a structure variable intrn is used and the data members of outer structure intern, are accessed using a single dot, eg - intrn.name, intrn.intern_id etc.
5. While printing the details of the nested structure variables, like school and college details of an intern, we'll have to follow the format outerStructure.innerStructure.variable.
6. Since the outer structure variable is intrn, and nested structure variables are scl and clg, to access the school name and college name of an intern we have used intrn.schl.school_name, intrn.clg.college_name.