CSE 109
Computer Programming
Bitwise operators and Structure

Prepared by
Madhusudhan Basak
Assistant Professor
CSE, BUET

Modified by
Shashata Sawmya
Lecturer
CSE, BUET

Bitwise Operators

- Performs operation on a bit-by-bit level.

Operator	Type	Name
&	Binary	Bitwise AND
|	Binary	Bitwise OR
^	Binary	Bitwise XOR
~	Unary	1’s Complement
<<	Binary	Left Shift Operator
>>	Binary	Right Shift Operator

- Works with character type and integer types only.

Bitwise Operators

- Bitwise AND

unsigned int i, j;

i=11

j=1

- Determine i & j

00000000 00000000 00000000 00001011

00000000 00000000 00000000 00000001

00000000 00000000 00000000 00000001

- So, i & j results in 1

bitwise_intro.c

Bitwise Operators

- Bitwise OR

unsigned int i, j;

i=11

j=1

- Determine i | j

```
00000000 00000000 00000000 00001011
00000000 00000000 00000000 00000001
------------------------------------------------------------------------
00000000 00000000 00000000 00001011
```

- So, i | j results in 11

bitwise_intro.c

Bitwise Operators

- Bitwise XOR

unsigned int i, j;

i=11

j=1

- Determine i ^ j

00000000 00000000 00000000 00001011

00000000 00000000 00000000 00000001

00000000 00000000 00000000 00001010

- So, i ^ j results in 10

bitwise_intro.c

Bitwise Operators

- Bitwise XOR

unsigned int i, j;

i=11

- Determine ~i

00000000 00000000 00000000 00001011

11111111 11111111 11111111 11110100

Which the unsigned binary representation of 4294967285

- So, ~i results in 4294967285

bitwise_intro.c

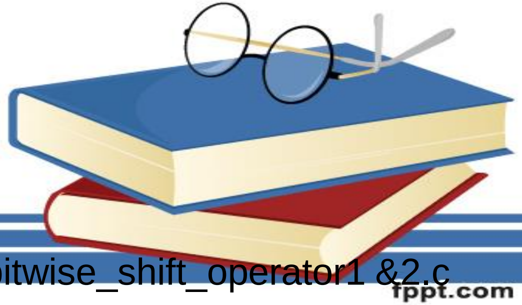
Shift Operators

- Left shift operator (<<)
- Right shift operator (>>)
- Binary operators
- Applies only to character or integer operands
- Format

$$value \ll number\ of\ bits$$

$$value \gg number\ of\ bits$$

- Left shift causes left shifting bits and adding 0's to right
- Right shift causes right shifting bits and 1) adding 0's to the left if unsigned 2) adding 1's to the left if signed



bitwise_shift_operator1 &2.c

Bitwise Operators

- The left shift and right shift operators should not be used for negative numbers.
- The bitwise operators should not be used in place of logical operators.
- The left-shift and right-shift operators are equivalent to multiplication and division by 2 respectively.
- The & operator can be used to quickly check if a number is odd or even.
- The ~ operator should be used carefully.

- See the following link for details
- https://www.geeksforgeeks.org/bitwise-operators-in-c-cpp/

bitwise_intro.c

Structure

- Aggregate data type
- Composed of two or more related variables
  - Called member
  - Each member can be of different types

Structure (General Form)

```
struct tag-name {
  type member1;
  type member2;
  type member3;
  .
  .
  .
  type memberN;
  } variable-list;
```

- The keyword **struct** means that a structure *type* is defined
- *tag-name* is the name of the *type*
- Either *tag-name* or *variable-list* is optional

Structure Example

```
struct point {
   int x;
   int y;
} p1, p2;

struct {
   int x;
   int y;
} p1, p2;
```

Structure Example

```
struct point {
   int x;
   int y;
};
```

struct point p1, p2;

- Keyword **struct** before variable declaration is necessary
- Each instance of a structure contains its own copy of the members
- Structure declaration without any variable name does not reserve any storage
- Describes template or shape of a structure

Structure Declaration

- Structure can be declared
  - Locally (inside a function)
  - Globally (outside of any function)

- Structure declaration and structure variable declaration are different.
  - Variables can be declared during structure declaration or later

- It is possible to declare structure globally and declare the variables of the structure locally

- See structure_declaration.c for detailed declaration.

Structure Initialization

```
p1.x=10;
p1.y=5;

struct point p3={5, 2};

p2={10, 5};//error, not possible
```

Structure Size

- The size of a structure element is greater than or equal to the summation of sizes of its fields (members).
- Why may it be greater?
  - For the ease of memory access! Memory is not access bytewise rather accessed 4 bytes per operation or 8 bytes per operation. So, it is common to find the size of the structure as the multiple 4 or 8.
  - Memory access is higher topic, not for now. Just remember the reason larger structure size.
- See for the following page details

https://www.geeksforgeeks.org/is-sizeof-for-a-struct-equal-to-the-sum-of-sizeof-of-each-member/

Structure Assignment

- Possible when type of the both objects are same

```
p2=p3;
```

Accessing Structure Member

- Use *dot operator* to access the member of a structure

```
StructureVariable.member
```

For example, if p3 is a structure variable,

```
printf("%d, %d\n", p3.x, p3.y);
```

- For scanf or other cases where address of a member is required, use & operator before structure variable name not before member name.

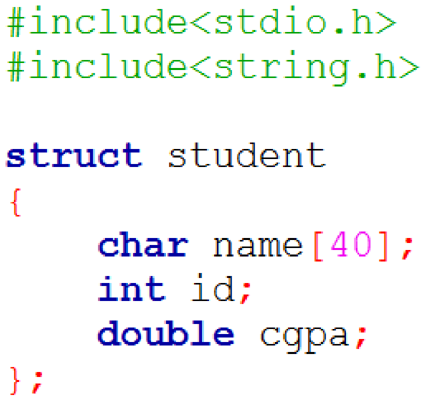
```
&StructureVariable.member
```

For example, if p3 is a structure variable,

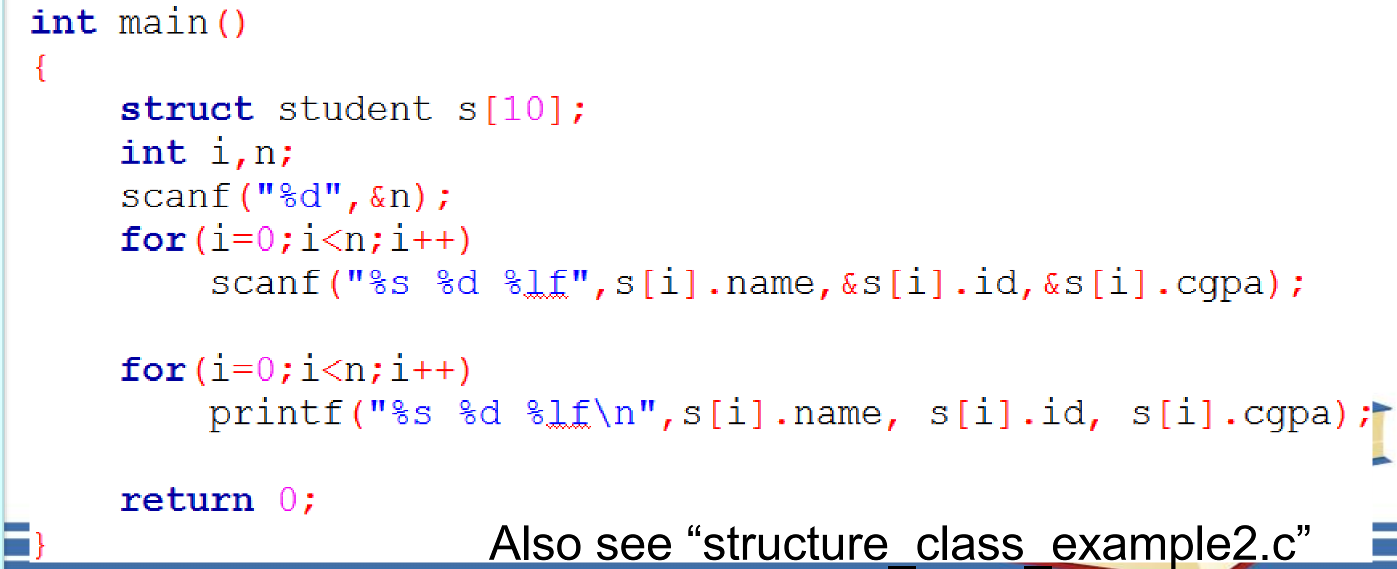
```
scanf("%d %d\n", &p3.x, &p3.y);
```

Structure Array

- A structure array can be declared like the process of a normal variable declaration.
- Index must be used to access an element of the array
- To access the member of that element, use *dot operator* after the index



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

struct student
{
    char name[40];
    int id;
    double cgpa;
};
```

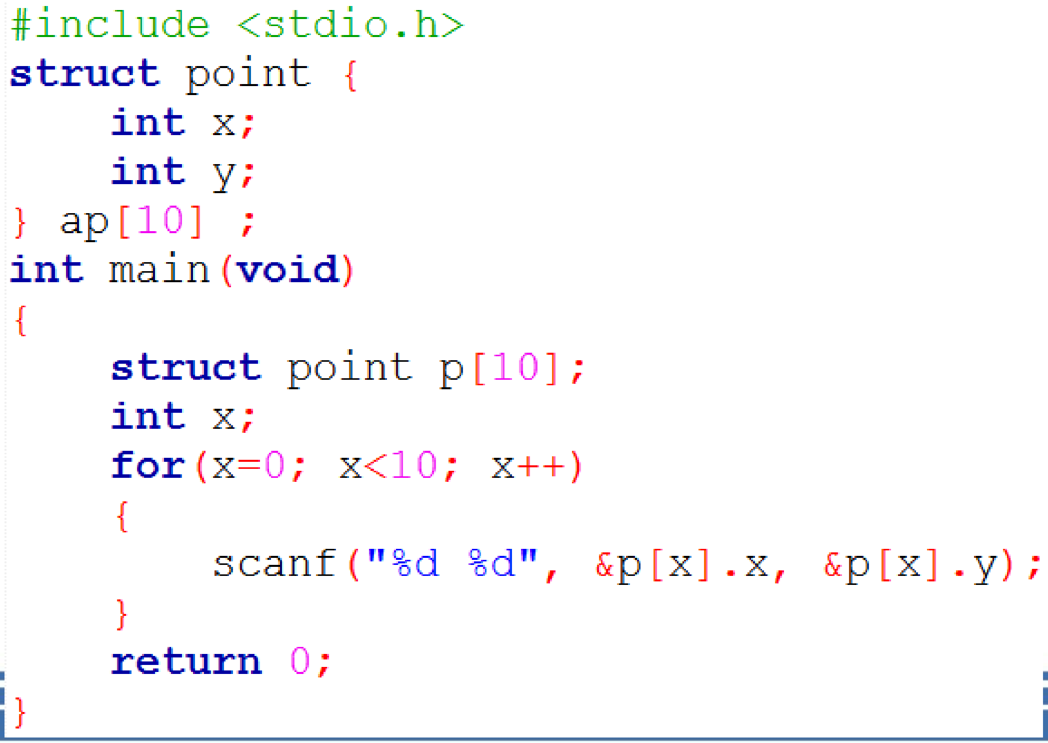


```c
int main()
{
    struct student s[10];
    int i,n;
    scanf("%d",&n);
    for(i=0;i<n;i++)
        scanf("%s %d %lf",s[i].name,&s[i].id,&s[i].cgpa);

    for(i=0;i<n;i++)
        printf("%s %d %lf\n",s[i].name, s[i].id, s[i].cgpa);

    return 0;
}
```

Also see “structure_class_example2.c”

Structure Array

- The member of a structure and a normal variable in a function can have the same name.



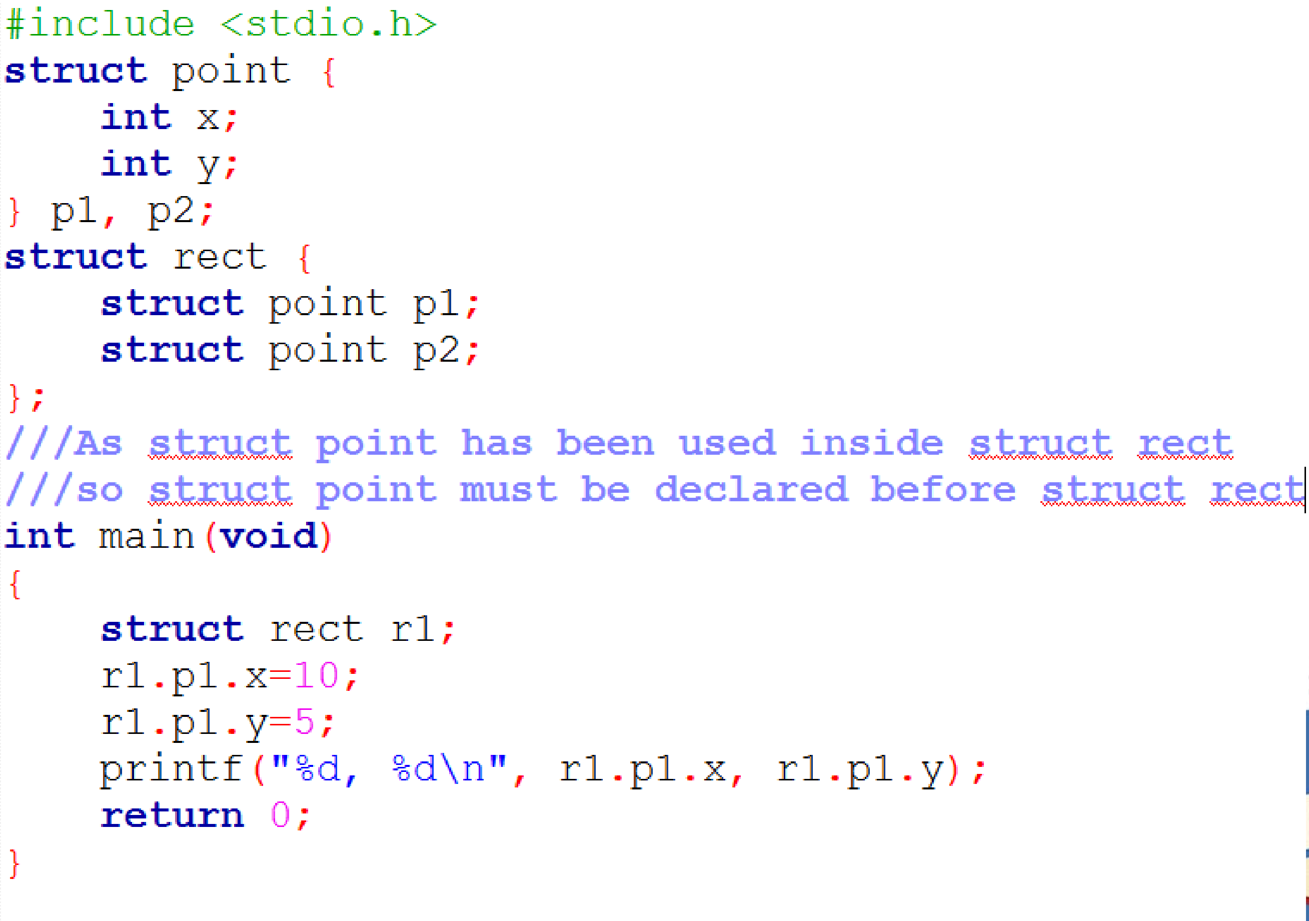
```c
#include <stdio.h>
struct point {
    int x;
    int y;
} ap[10] ;
int main(void)
{
    struct point p[10];
    int x;
    for(x=0; x<10; x++)
    {
        scanf("%d %d", &p[x].x, &p[x].y);
    }
    return 0;
}
```

Member x and int x are different

Nested Structure

- A structure can be used inside another structure and so on.
- If structure A is used inside structure B, then structure A must be declared before structure B.
- If structure A is used inside structure B, then B is the outer structure and A is inner structure.
- During accessing the elements of nested structure object, use the outermost structure variable first, then a dot, then the inner one, and so on.

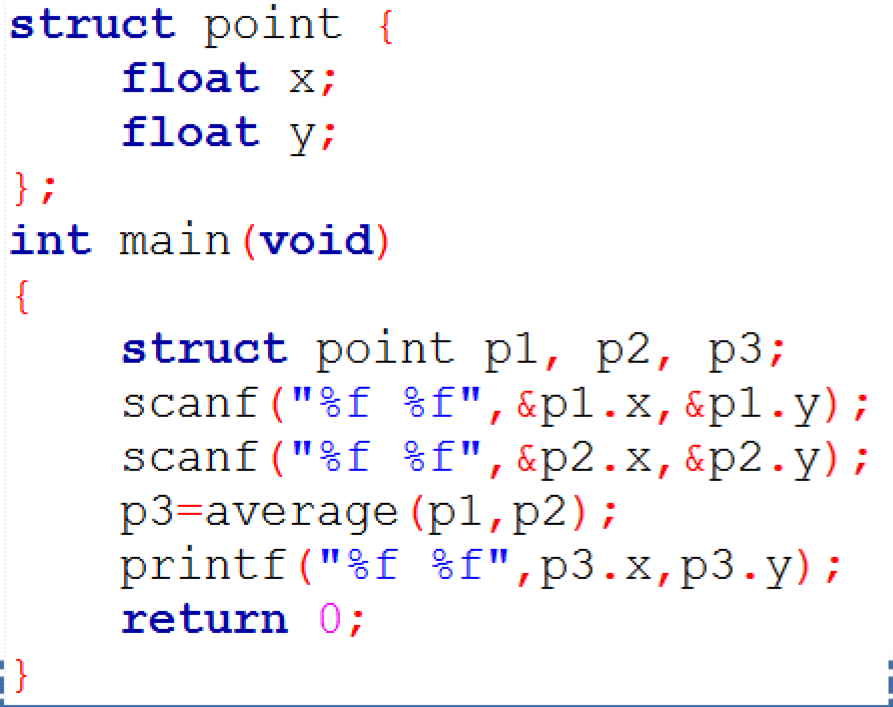
Nested Structure



```c
#include <stdio.h>
struct point {
    int x;
    int y;
} p1, p2;
struct rect {
    struct point p1;
    struct point p2;
};
///As struct point has been used inside struct rect
///so struct point must be declared before struct rect
int main(void)
{
    struct rect r1;
    r1.p1.x=10;
    r1.p1.y=5;
    printf("%d, %d\n", r1.p1.x, r1.p1.y);
    return 0;
}
```

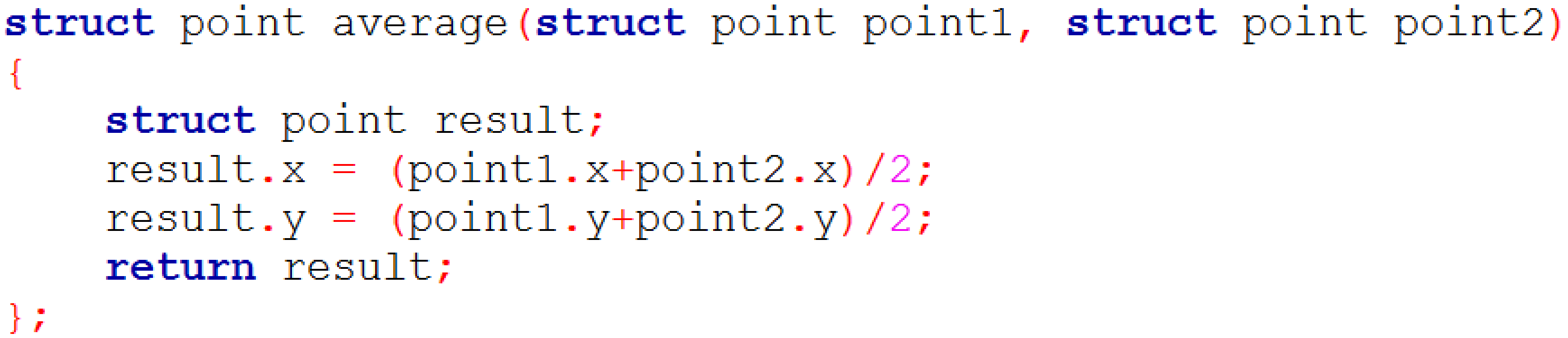
Structure in Function

- A structure object can be passed into a function and a structure object can be returned from a function.



```
struct point {
    float x;
    float y;
};
int main(void)
{
    struct point p1, p2, p3;
    scanf("%f %f",&p1.x,&p1.y);
    scanf("%f %f",&p2.x,&p2.y);
    p3=average(p1,p2);
    printf("%f %f",p3.x,p3.y);
    return 0;
}
```

Structure in Function



```
struct point average(struct point point1, struct point point2)
{
    struct point result;
    result.x = (point1.x+point2.x)/2;
    result.y = (point1.y+point2.y)/2;
    return result;
};
```

- The function prototype can be written above the structure definition but the full definition of the function must be written below the definition.

Pointer to Structure

- Pointer of a structure can be declared in the same way pointers to other variables are declared.
- When accessing a member using a structure, use the *dot operator.* When accessing a member using a pointer, use the *arrow operator.*
- See “structure_pointer.c” and “structure_pointer1.c” for the use of pointer to structure
- See structure_pointer2.c to see the use of pointer for the case of nested structure

typedef (user defined data types)

- `typedef type new-type`
- `type`: an existing data type
- Example:
  - `typedef int age`
    - `age` is a user defined data type, equivalent to `int`
  - `typedef double height[100];`

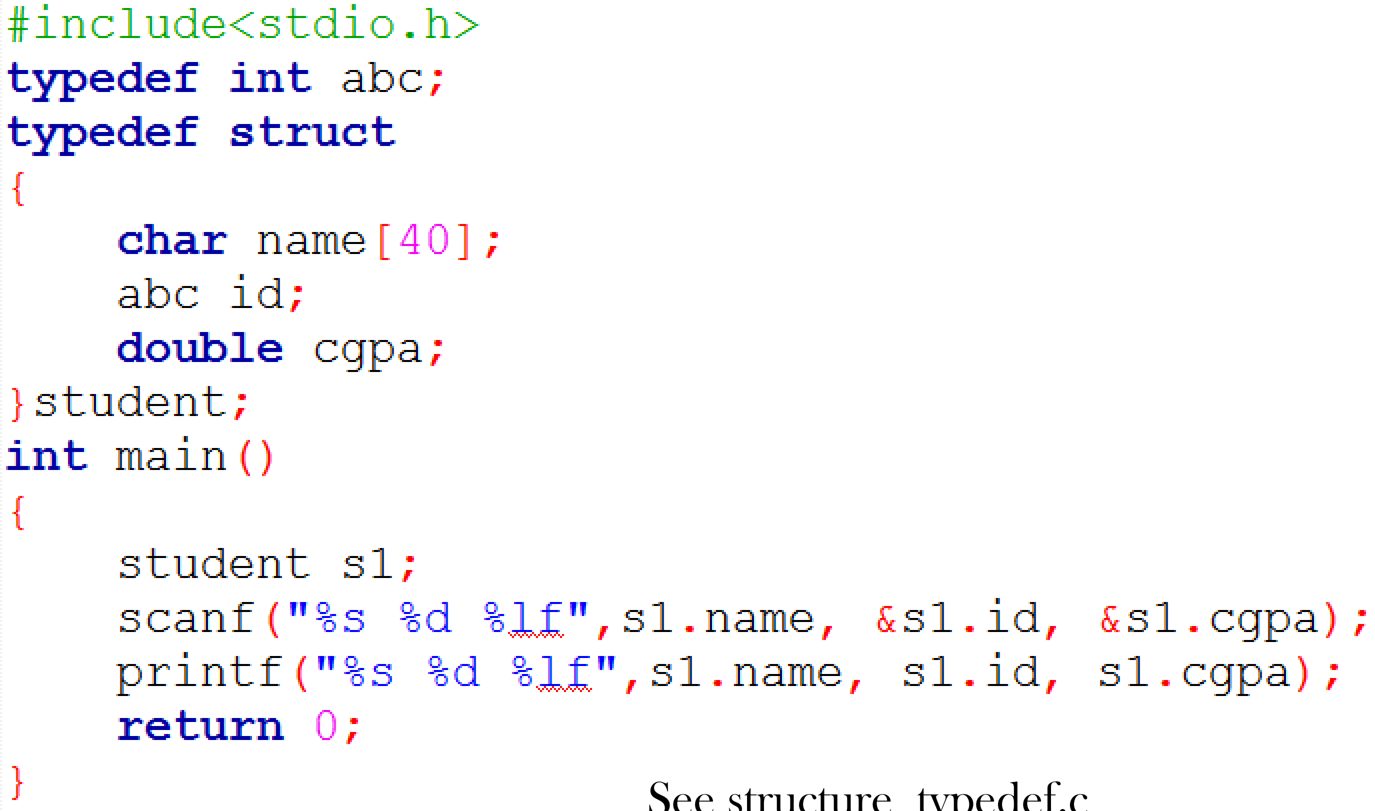
    `height male, female;`

typedef with Structure

```
typedef struct
{
        member1;
        member2;
} new-type;
```

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

typedef with Structure



```
#include<stdio.h>
typedef int abc;
typedef struct
{
    char name[40];
    abc id;
    double cgpa;
}student;
int main()
{
    student s1;
    scanf("%s %d %lf",s1.name, &s1.id, &s1.cgpa);
    printf("%s %d %lf",s1.name, s1.id, s1.cgpa);
    return 0;
}
```

See structure_typedef.c

References

- Teach Yourself C by Herbert Schildt (Third Edition)
  - Chapter 10 (10.1-10.3)
  - Chapter 11 (11.4-11.6)
- https://www.geeksforgeeks.org/structures-c/

Thank You ☺