Structures

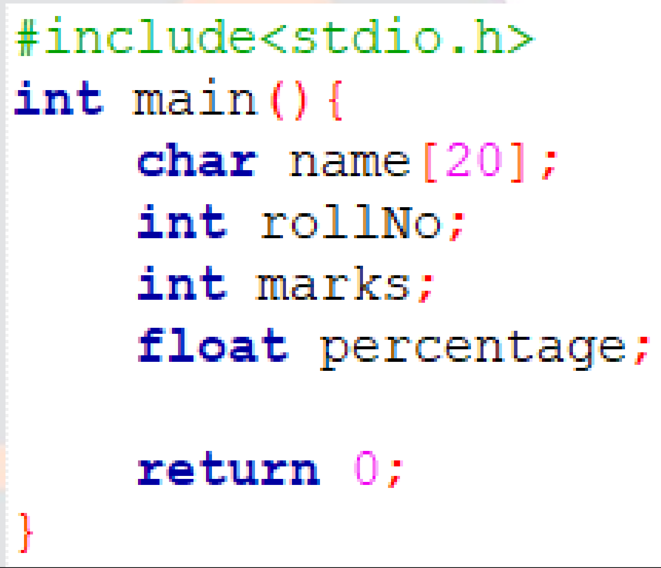
CHAPTER 33

SURESH TECHS

C PROGRAMMING COURSE

Program to take details about a student

Name
Roll number
Marks
Percentage



```
#include<stdio.h>
int main(){
    char name[20];
    int rollNo;
    int marks;
    float percentage;

    return 0;
}
```

But what if you want to represent 60 students?

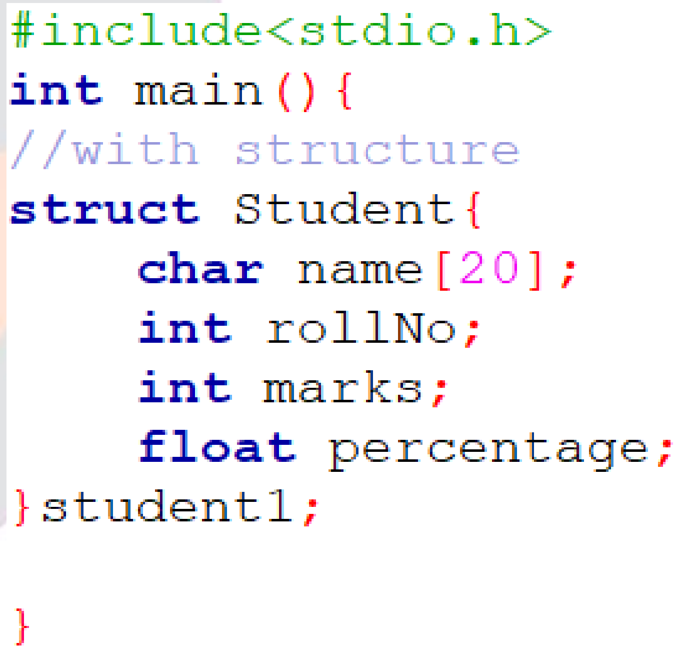
But array's store values of same data type 🤣🤣🤣

- To store integers, characters, decimal values we have int, char, float or double data types(primitive data types)
- Arrays and strings are used to store similar types of data together
- What if you want to stores two different types of data together
- Ex: student variable want to store his name, roll no, marks, percentage etc
- We can do that by creating different variables but that is not the feasible solution
- That’s where structure comes into the picture

How to create a structure?

- To create a structure, the **struct** keyword is used followed by the name of the structure. Then the body of the structure is defined in which the required data members are added

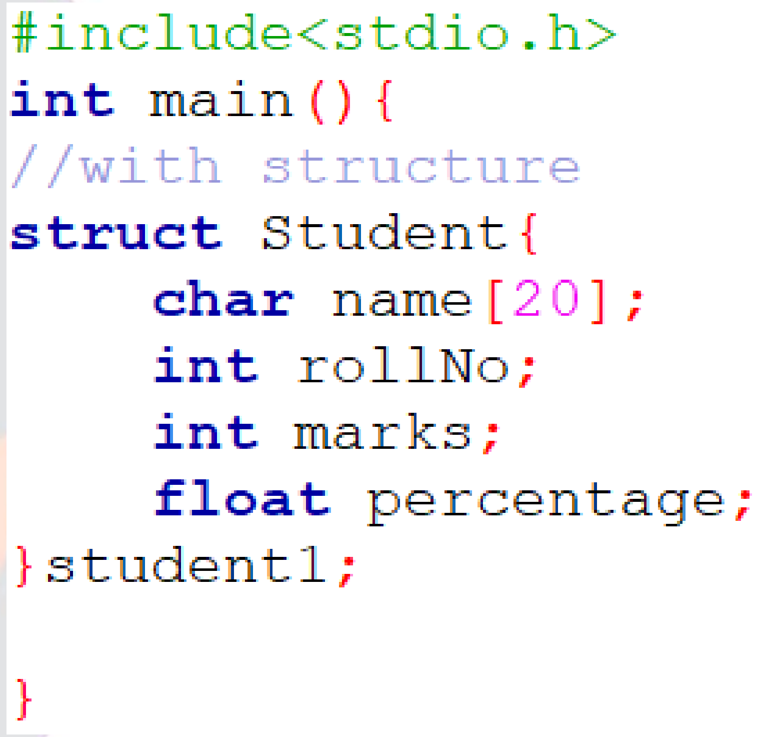
```
struct structure_name
{
    Data_member_type data_member_defination;
    Data_member_type data_member_defination;
    Data_member_type data_member_defination;
    ...
    ...
}sturcture_variables;
```



```
#include<stdio.h>
int main(){
//with structure
struct Student{
    char name[20];
    int rollNo;
    int marks;
    float percentage;
}student1;

}
```

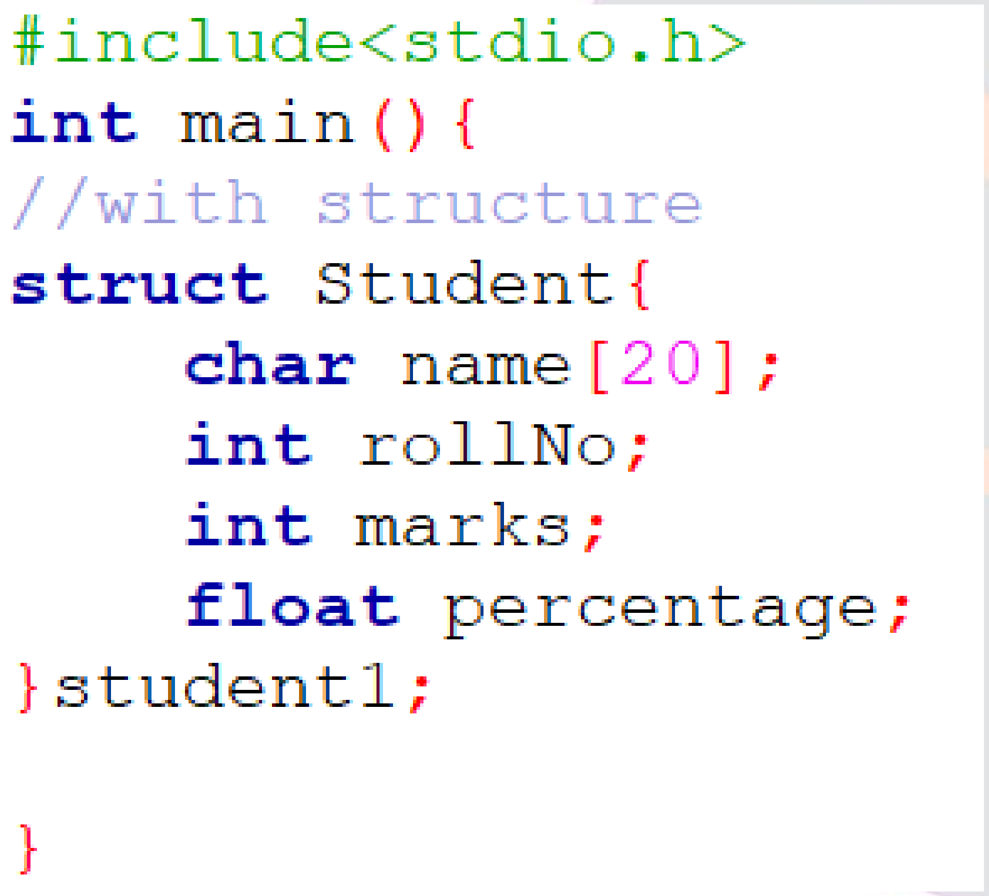
What is structure

- It is a **user defined data type** used to store two or more data types together
- Structures **don’t take up space in memory** unless and until we define some variables for it
- When **we define variables**, they **take up memory space depends up on the type of the data members**

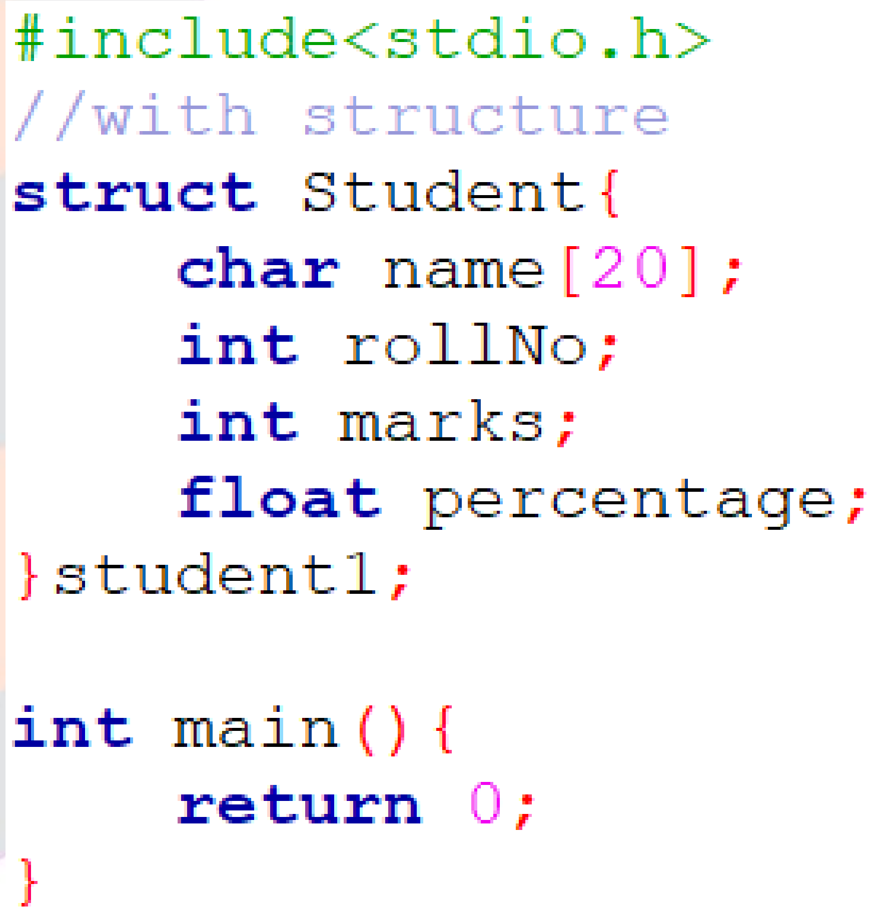


```
#include<stdio.h>
int main(){
//with structure
struct Student{
    char name[20];
    int rollNo;
    int marks;
    float percentage;
}student1;

}
```

To make the structure global we will define it outside of the function

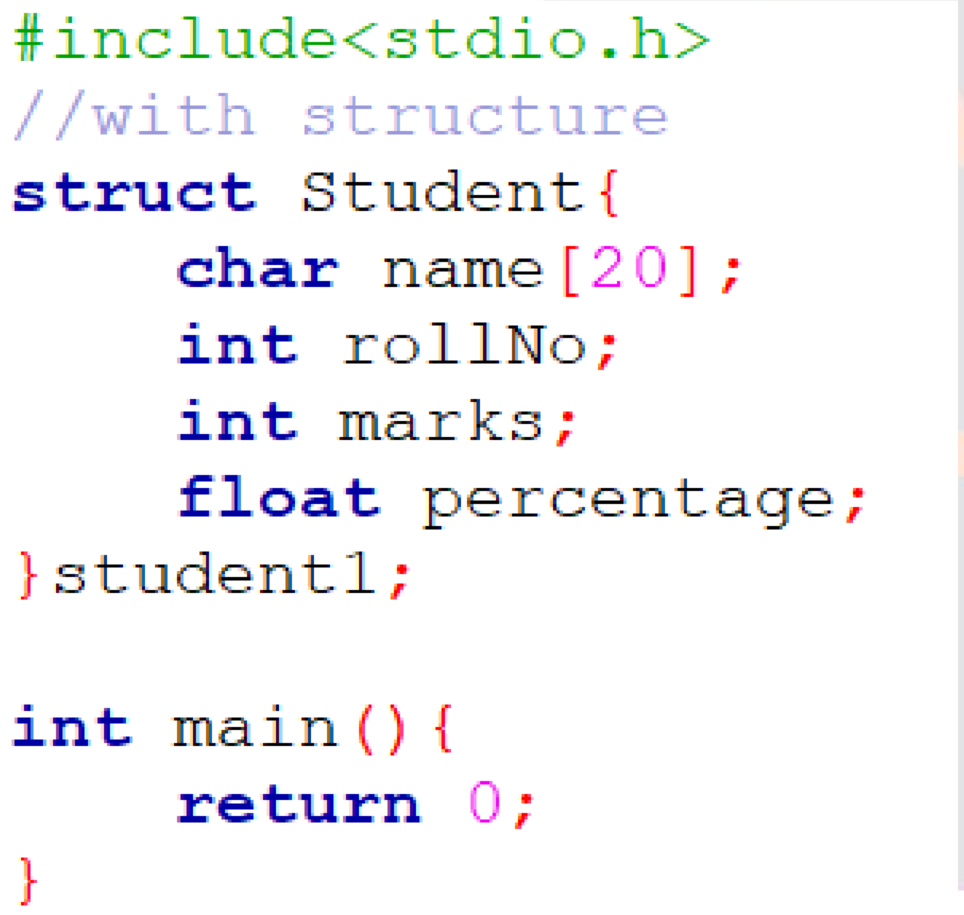


```c
#include<stdio.h>
int main(){
//with structure
struct Student{
    char name[20];
    int rollNo;
    int marks;
    float percentage;
}student1;

}
```

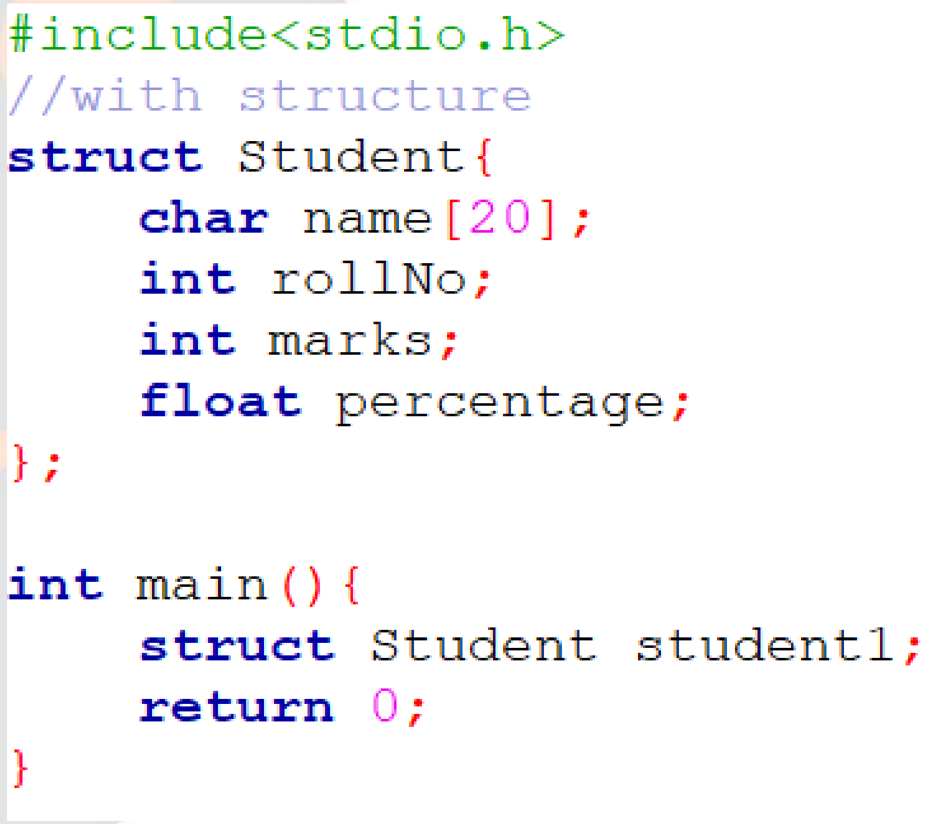


```c
#include<stdio.h>
//with structure
struct Student{
    char name[20];
    int rollNo;
    int marks;
    float percentage;
}student1;

int main(){
    return 0;
}
```

Another way of defining/declaring structure variables



```c
#include<stdio.h>
//with structure
struct Student{
    char name[20];
    int rollNo;
    int marks;
    float percentage;
}student1;

int main(){
    return 0;
}
```



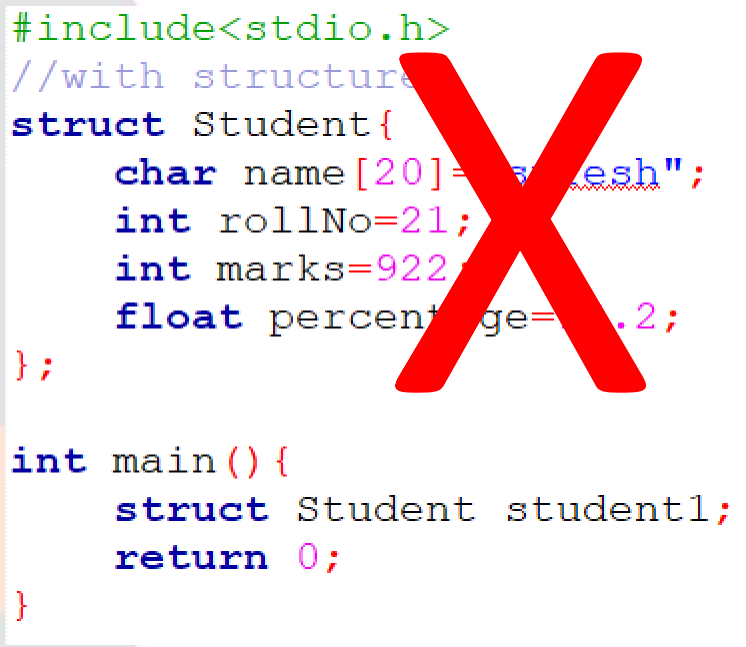
```c
#include<stdio.h>
//with structure
struct Student{
    char name[20];
    int rollNo;
    int marks;
    float percentage;
};

int main(){
    struct Student student1;
    return 0;
}
```

Which approach is better

- First approach is global where as second approach is local to the function where we defined the structure variable
- If we need to global variable then use first approach otherwise second approach is better

Initializing values to structure members

- Assigning values to the structure members
- As I told earlier declaration of a structure doesn’t allocate memory to the structure so we can’t assign values during declaration

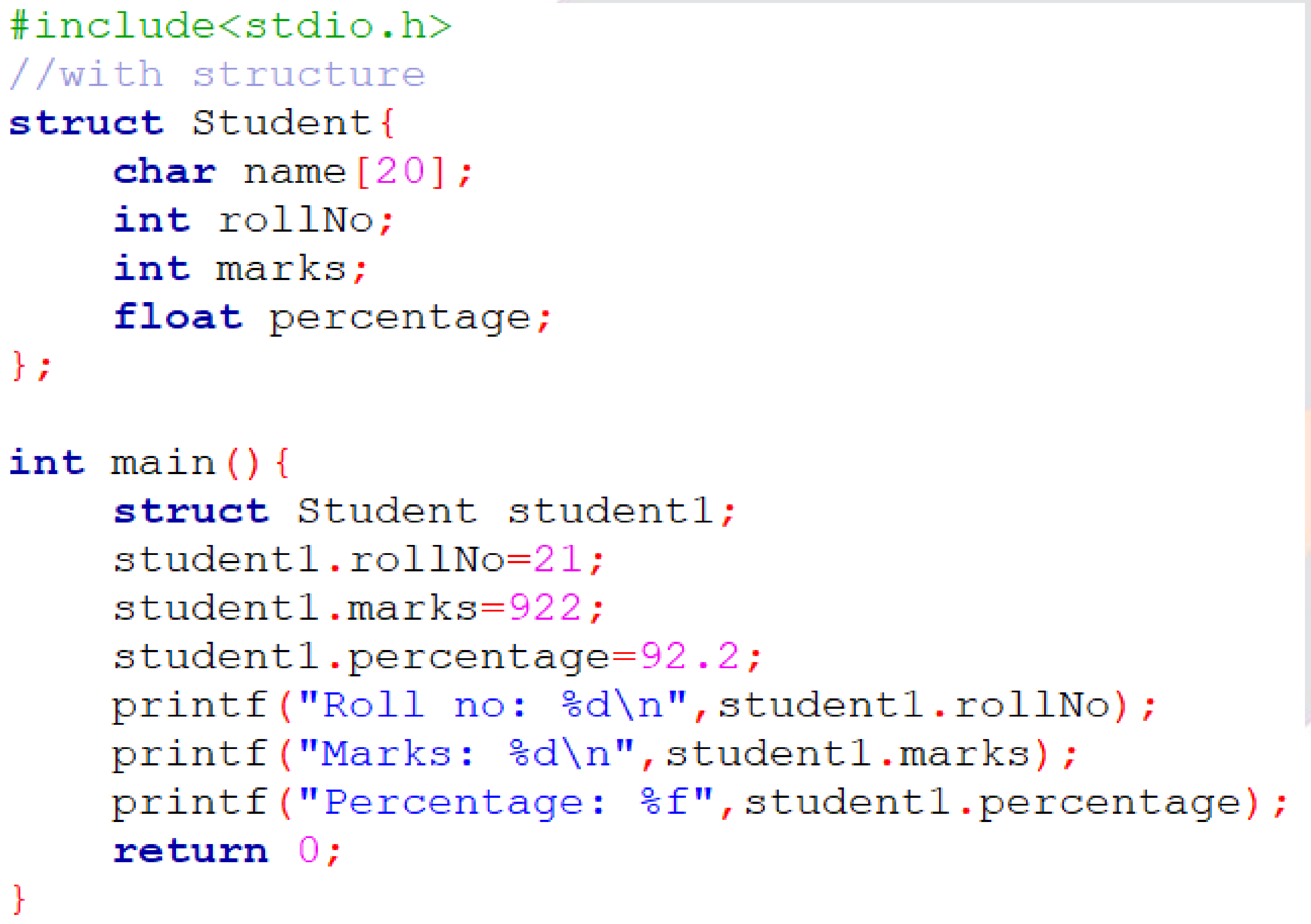


```
#include<stdio.h>
//with structure[illegible]
struct Student{
    char name[20]=[illegible]esh";
    int rollNo=21;
    int marks=922[illegible]
    float percent[illegible]ge=[illegible].2;
};

int main(){
    struct Student student1;
    return 0;
}
```

Three ways to initialize structure members

- Using dot '.' operator
- Using curly braces ‘{}’
- Designated initializers

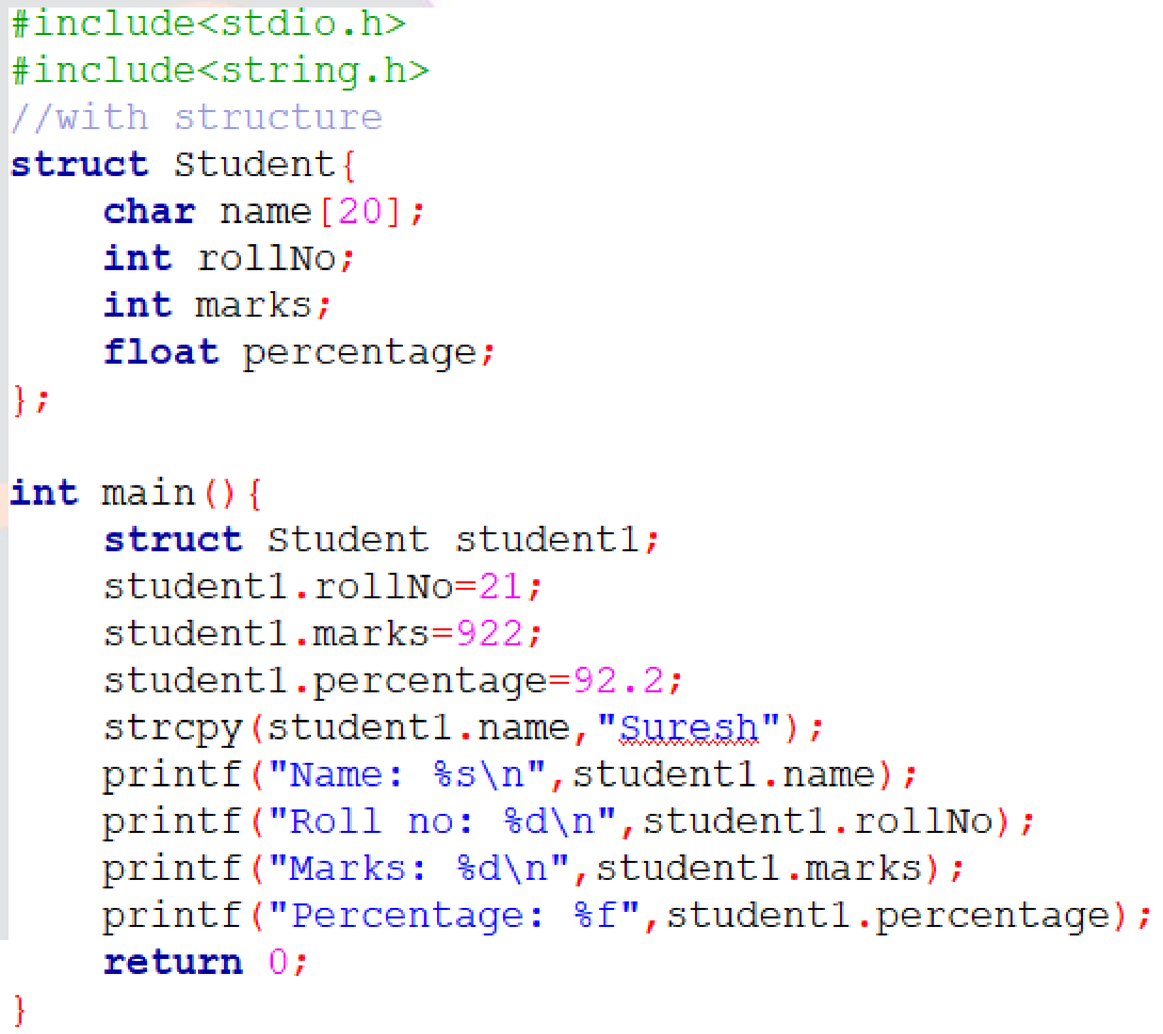
Using dot (.) to initialize and access members of a structure



```
#include<stdio.h>
//with structure
struct Student{
    char name[20];
    int rollNo;
    int marks;
    float percentage;
};

int main(){
    struct Student student1;
    student1.rollNo=21;
    student1.marks=922;
    student1.percentage=92.2;
    printf("Roll no: %d\n",student1.rollNo);
    printf("Marks: %d\n",student1.marks);
    printf("Percentage: %f",student1.percentage);
    return 0;
}
```

How to assign string value

- student1.name="suresh"; //Error
- strcpy(holder,value); //string.h
- strcpy is used to assign value of one string to another string



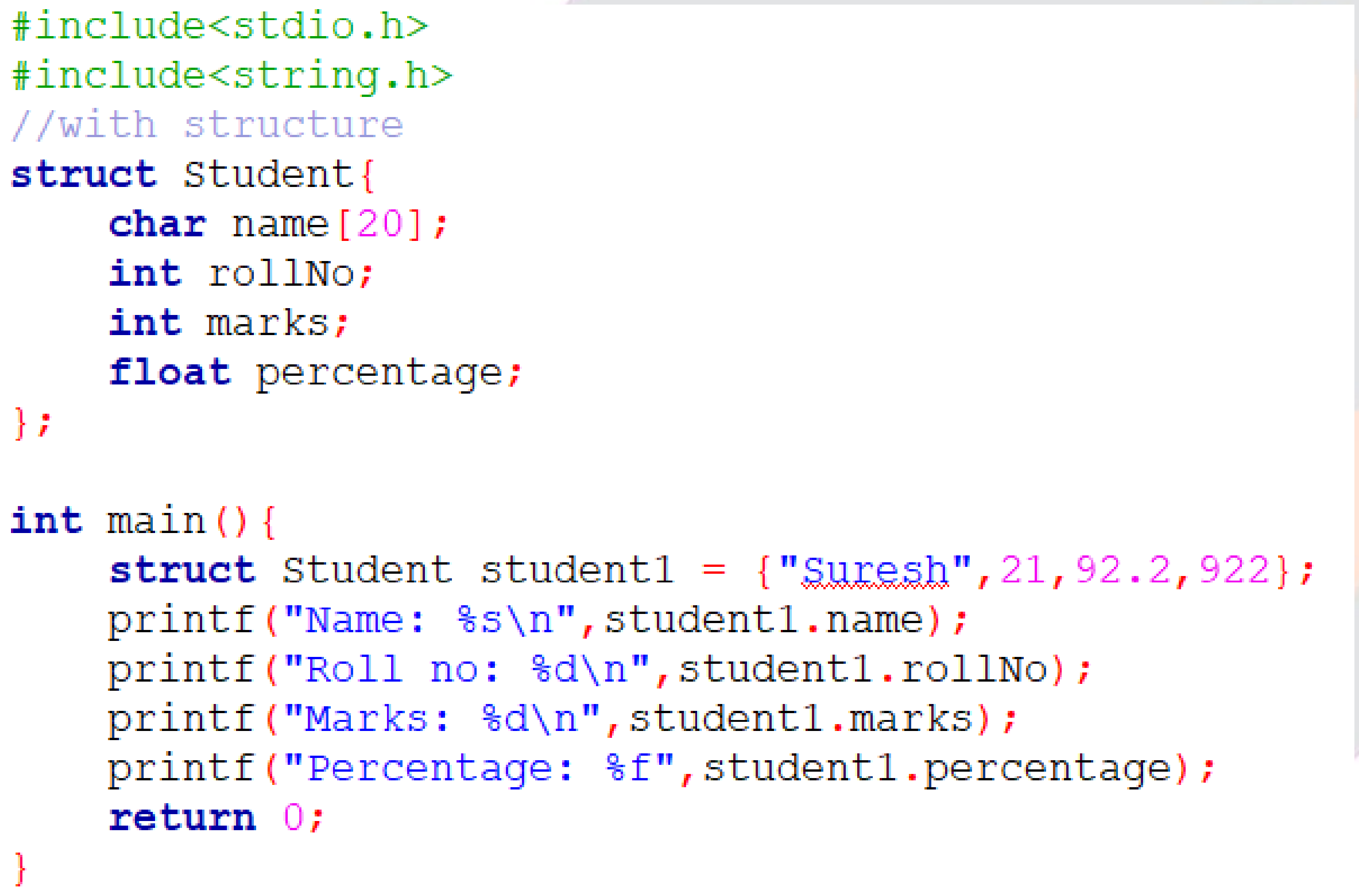
```
#include<stdio.h>
#include<string.h>
//with structure
struct Student{
    char name[20];
    int rollNo;
    int marks;
    float percentage;
};

int main(){
    struct Student student1;
    student1.rollNo=21;
    student1.marks=922;
    student1.percentage=92.2;
    strcpy(student1.name,"Suresh");
    printf("Name: %s\n",student1.name);
    printf("Roll no: %d\n",student1.rollNo);
    printf("Marks: %d\n",student1.marks);
    printf("Percentage: %f",student1.percentage);
    return 0;
}
```

Using curly braces { }

- If we want to initialize all the members during the structure variable declaration, we can declare using curly braces.
- struct stucture_name v1 = {value, value, value, ..};
- To initialize the data members by this method, the comma separated values should be provided in the **same order as the members are declared in the structure**.
- Also, this method is beneficial to use when we have to initialize all the data members.

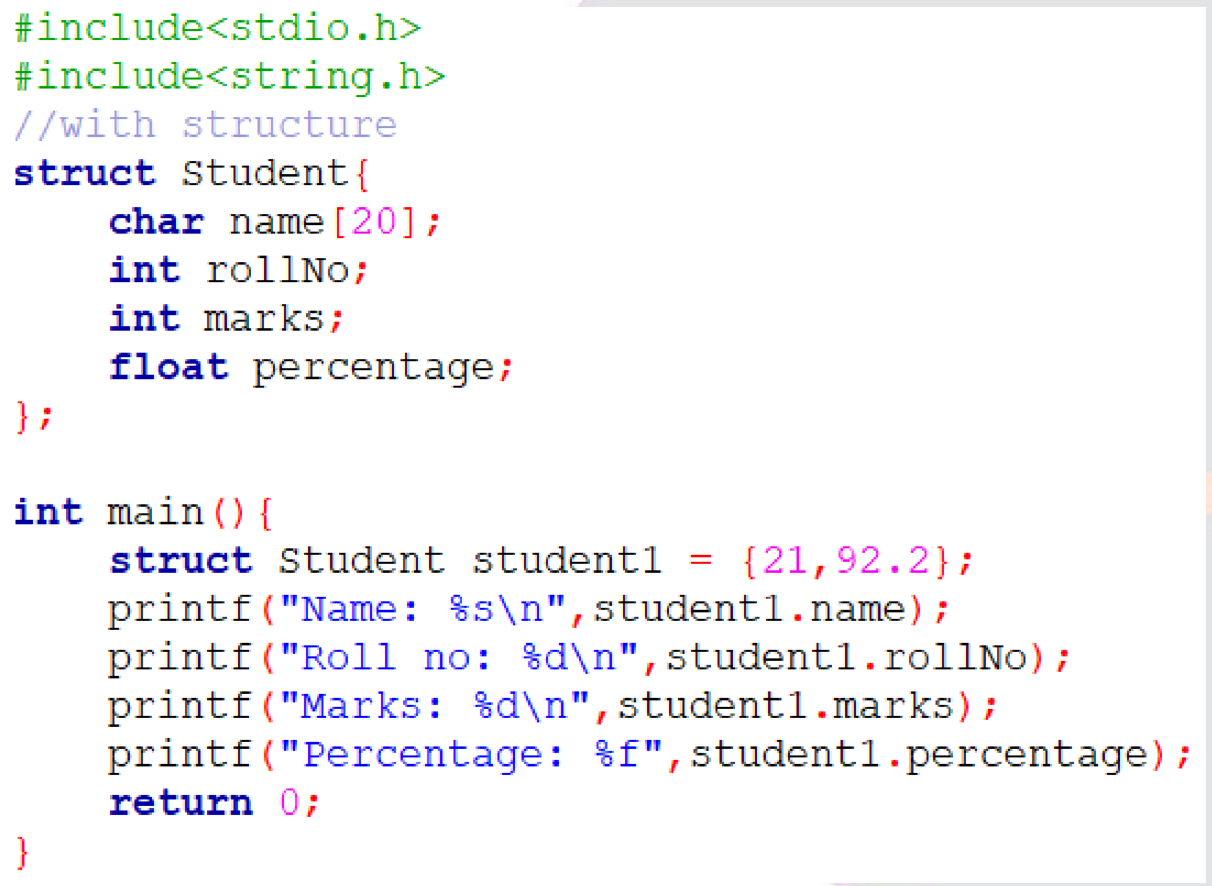
```
#include<stdio.h>
#include<string.h>
//with structure
struct Student{
    char name[20];
    int rollNo;
    int marks;
    float percentage;
};

int main(){
    struct Student student1 = {"Suresh",21,922,92.2};
    printf("Name: %s\n",student1.name);
    printf("Roll no: %d\n",student1.rollNo);
    printf("Marks: %d\n",student1.marks);
    printf("Percentage: %f",student1.percentage);
    return 0;
}
```

Should follow the order, look at the error here



```c
#include<stdio.h>
#include<string.h>
//with structure
struct Student{
    char name[20];
    int rollNo;
    int marks;
    float percentage;
};

int main(){
    struct Student student1 = {"Suresh",21,92.2,922};
    printf("Name: %s\n",student1.name);
    printf("Roll no: %d\n",student1.rollNo);
    printf("Marks: %d\n",student1.marks);
    printf("Percentage: %f",student1.percentage);
    return 0;
}
```

What happens here?



```c
#include<stdio.h>
#include<string.h>
//with structure
struct Student{
    char name[20];
    int rollNo;
    int marks;
    float percentage;
};

int main(){
    struct Student student1 = {21,92.2};
    printf("Name: %s\n",student1.name);
    printf("Roll no: %d\n",student1.rollNo);
    printf("Marks: %d\n",student1.marks);
    printf("Percentage: %f",student1.percentage);
    return 0;
}
```

Designated initializers

- We can use designated initializers when we want to initialize **only a few structure members.**
- Will discuss in sometime

Bit Fields are used for altering the size of the data type to save the memory

```
#include<stdio.h>
struct dob{
    int day;
    int month;
    int year;
};

int main(){
    struct dob sureshDOB;
    printf("%d",sizeof(sureshDOB));
    return 0;
}
```

How much memory does these three variables takes?

day: 1 to 31 values
month: 1 to 12
What is the point of wasting so much memory when we don’t use any other values for day and month

How about changing the default memory for a data type? From 4 bytes to 1 byte 🤔🤔?

4 bytes – 32 bits
2 bytes – 16 bits
1 byte – 8 bits

Bit Fields

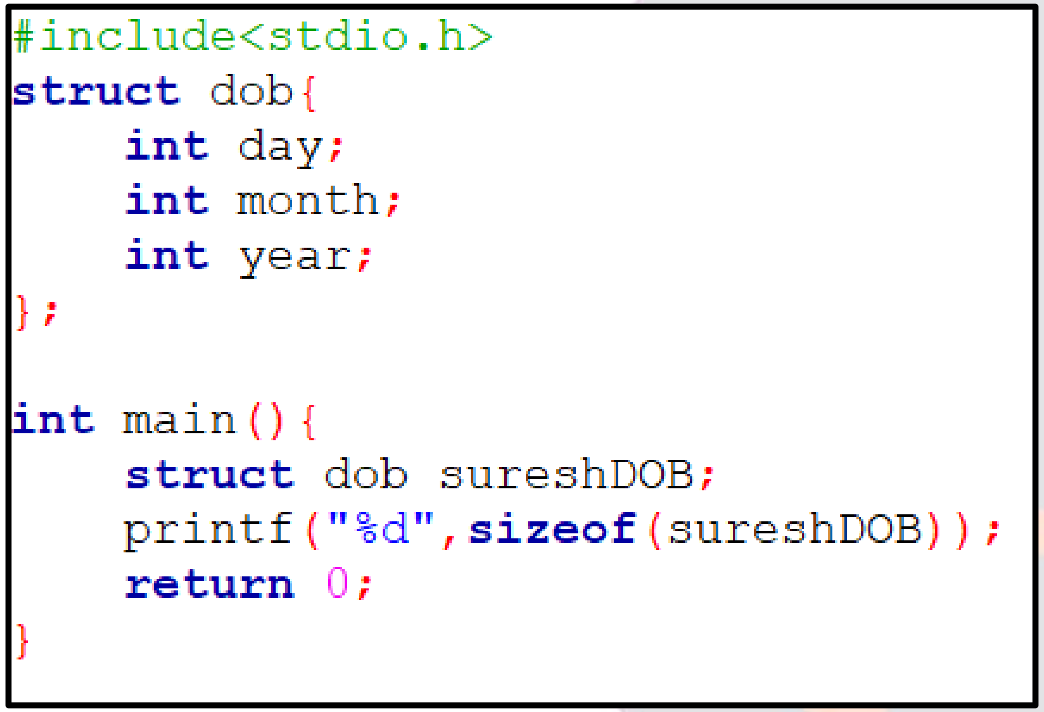
Bit Fields

- Memory is allocated in bits to store every data type
- int – 4 bytes – 32bits
- char – 1 byte – 8 bits
- Bit fields is the **concept of Structure in C** in which **we can define how many bits we have to allocate to the particular data member** of Structure to save memory
- int ->10 bits
- We can define the number of bits for a particular member using the **colon (:) operator**

Bit Fields

- Bit fields is the **concept of Structure in C** in which **we can define how many bits we have to allocate to the particular data member** of Structure to save memory
- int ->32 -> 5 bits->2^5 ->32->0 to 31 numbers
- 5 bits = how many bytes?
- Let us see any example

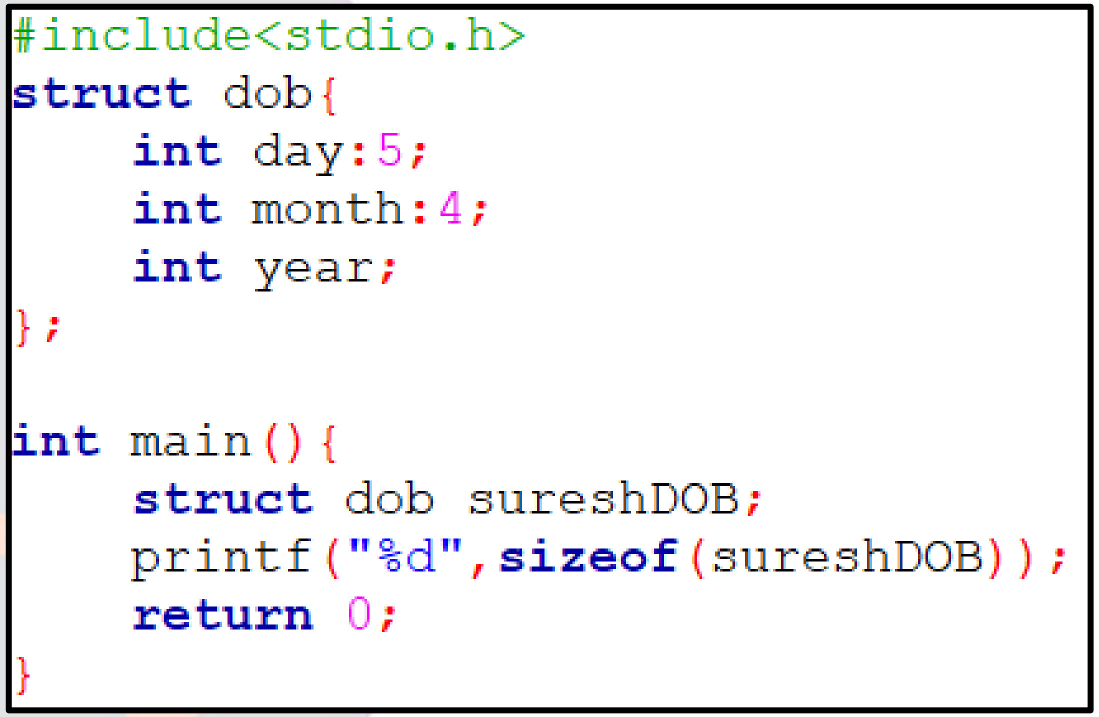
Syntax



```c
#include<stdio.h>
struct dob{
    int day;
    int month;
    int year;
};

int main(){
    struct dob sureshDOB;
    printf("%d",sizeof(sureshDOB));
    return 0;
}
```

12 bytes = 4+4+4

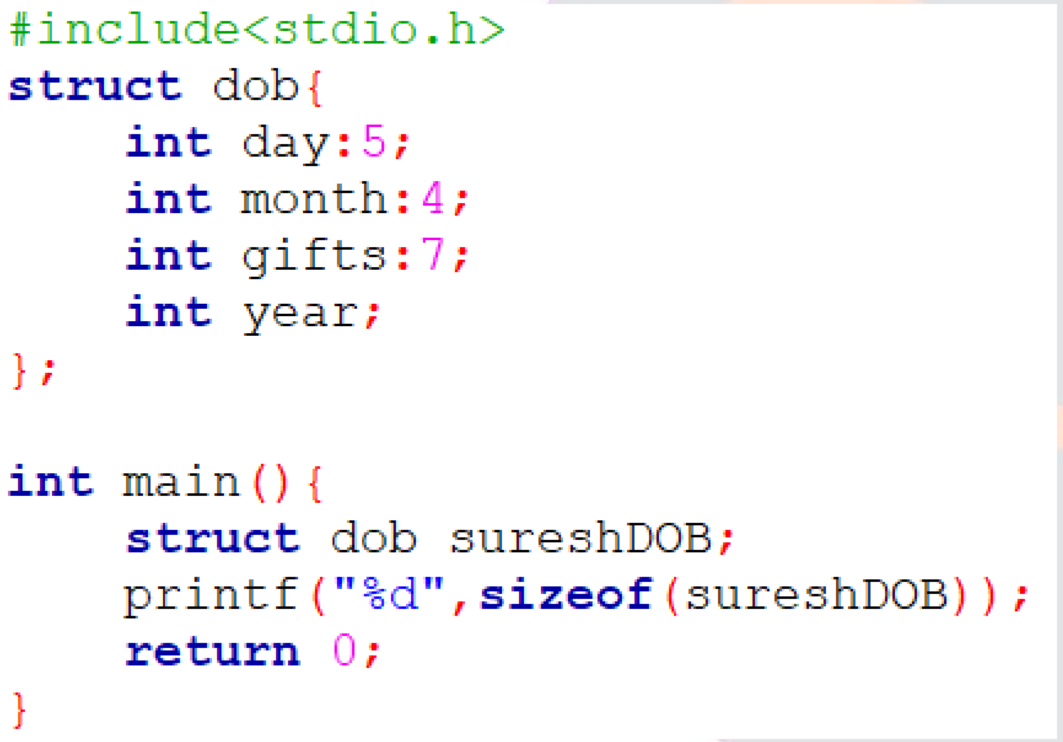
9 bits + 4 bytes



```c
#include<stdio.h>
struct dob{
    int day:5;
    int month:4;
    int year;
};

int main(){
    struct dob sureshDOB;
    printf("%d",sizeof(sureshDOB));
    return 0;
}
```

Since int data type uses 32 bits, day and month will take 4 bytes of memory. So 4+4 = 8 bytes

What is the size of this structure?



```c
#include<stdio.h>
struct dob{
    int day:5;
    int month:4;
    int gifts:7;
    int year;
};

int main(){
    struct dob sureshDOB;
    printf("%d",sizeof(sureshDOB));
    return 0;
}
```

8 bytes

How much memory saved?

8 bytes saved😍 😍 😍

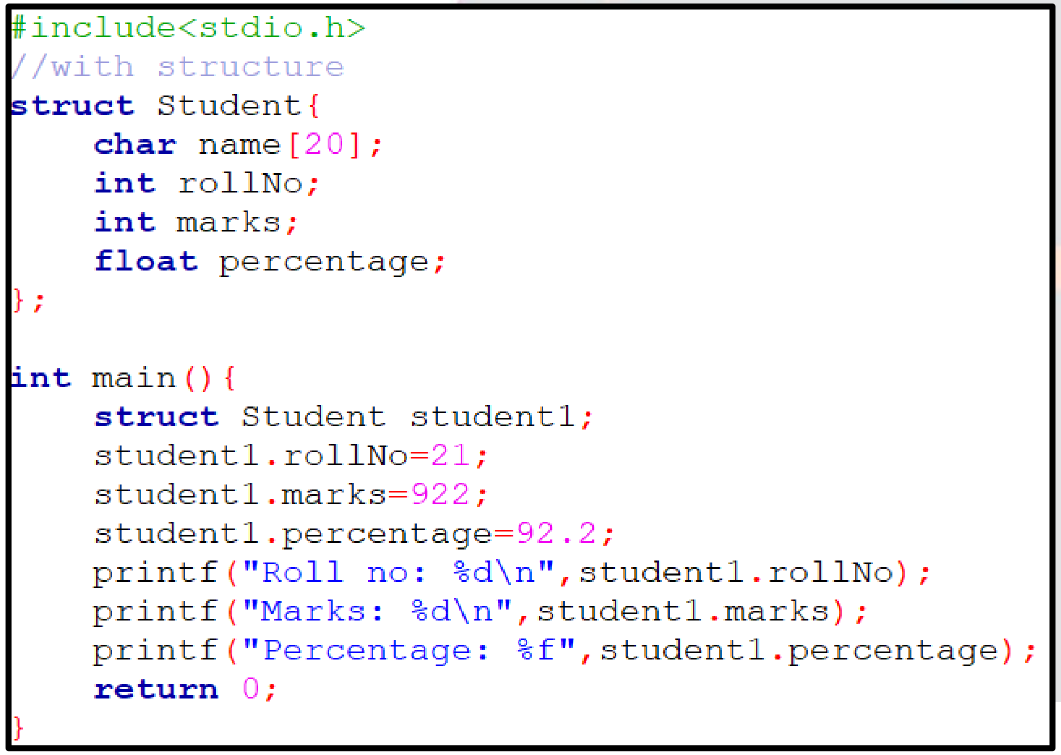
Syntax

```
struct structure_name
{
data_member : number_of_bits;
    ...
    ...
}
```

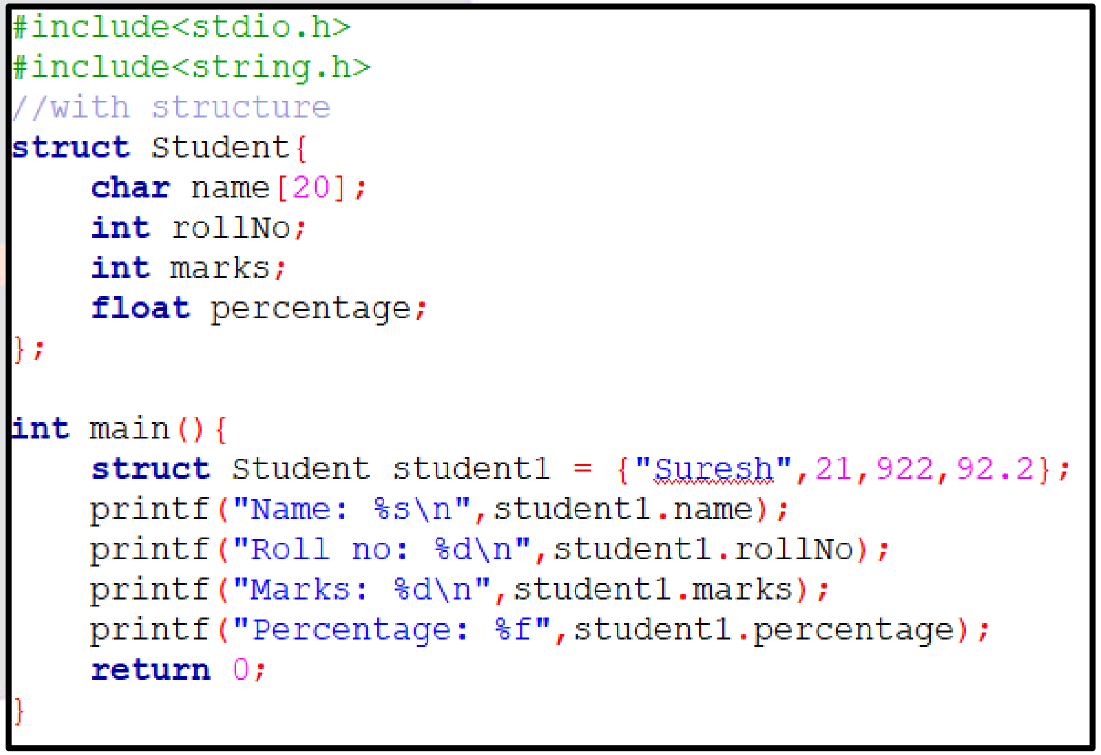
- We can’t use bitfields with float and double due to fractional parts and it’s a long way to explain them, not needed now 😊

Designated initialization

- Previous two ways: 1. dot 2. curly braces



```c
#include<stdio.h>
//with structure
struct Student{
    char name[20];
    int rollNo;
    int marks;
    float percentage;
};

int main(){
    struct Student student1;
    student1.rollNo=21;
    student1.marks=922;
    student1.percentage=92.2;
    printf("Roll no: %d\n",student1.rollNo);
    printf("Marks: %d\n",student1.marks);
    printf("Percentage: %f",student1.percentage);
    return 0;
}
```



```c
#include<stdio.h>
#include<string.h>
//with structure
struct Student{
    char name[20];
    int rollNo;
    int marks;
    float percentage;
};

int main(){
    struct Student student1 = {"Suresh",21,922,92.2};
    printf("Name: %s\n",student1.name);
    printf("Roll no: %d\n",student1.rollNo);
    printf("Marks: %d\n",student1.marks);
    printf("Percentage: %f",student1.percentage);
    return 0;
}
```

Designated initialization

Used when we want to initialize only a few structure members

```
struct strucutre_name structure_varialbe = {
        .structure_member = value,
        .structure_member = value
};
```

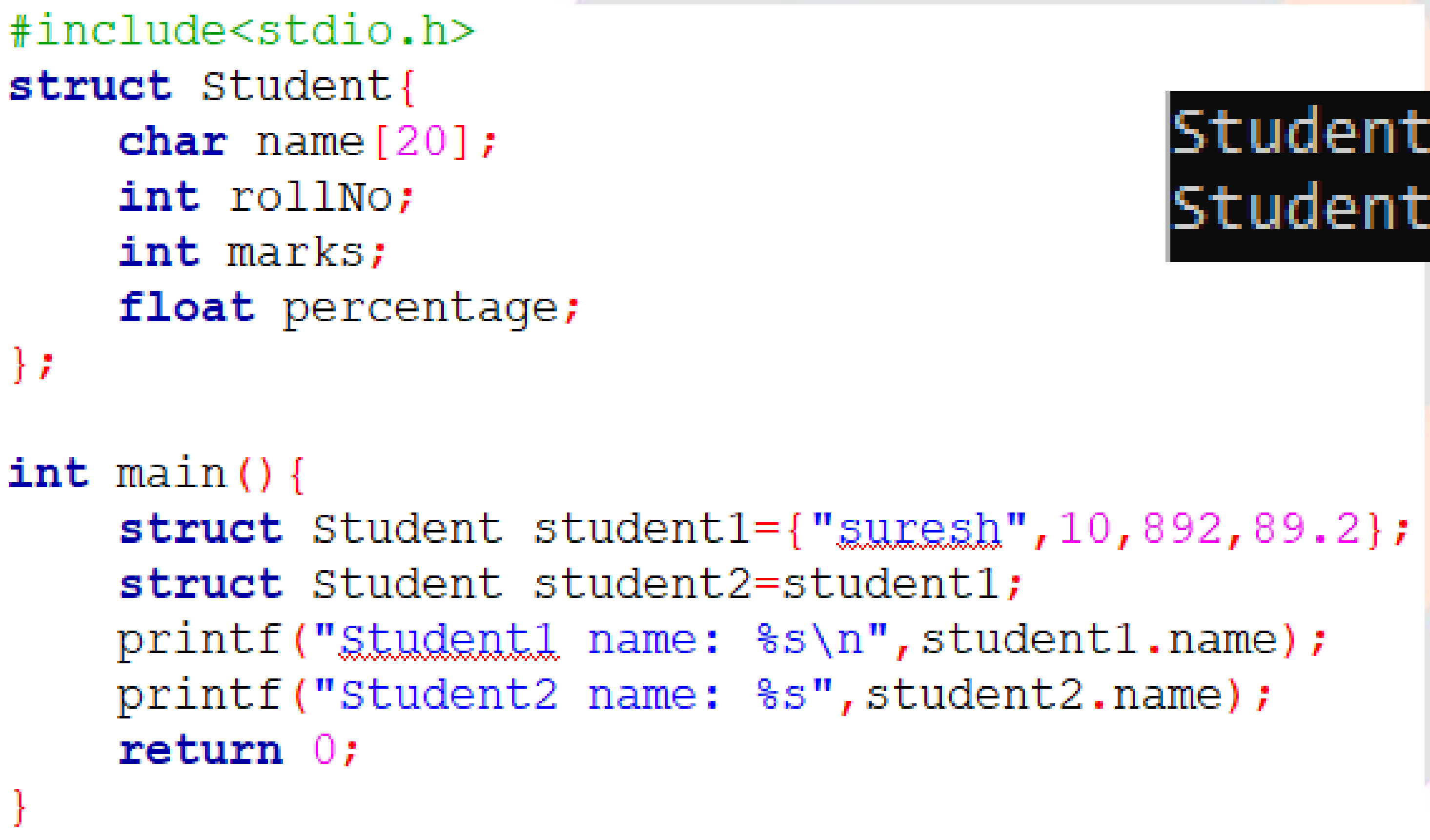
Most important thing is that we can initialize members in any order. It is **not compulsory to maintain the same order as the members are declared in the structure**.

```c
#include<stdio.h>
struct dob{
    int day:5;
    int month:4;
    int gifts:7;
    int year;
};

int main(){
    struct dob sureshDOB;
    //1. dot operator
    sureshDOB.day=1;
    sureshDOB.month=6;
    sureshDOB.gifts=10;
    sureshDOB.year=1994;
    //2. curly braces
    struct dob johnDOB={29,12,5,1993};
    //3. designated initialization
    struct dob jerryDOB={.month=1,.gifts=6,.year=1993,.day=15};

    return 0;
}
```

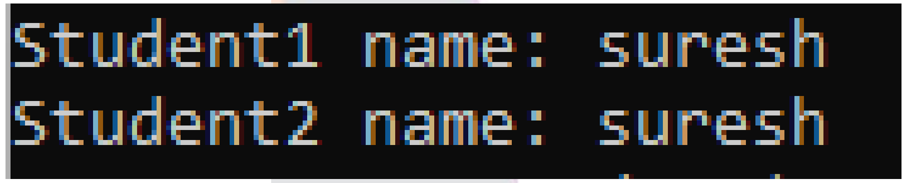
Let us represent 60 students using structures

```c
#include<stdio.h>
struct Student{
    char name[20];
    int rollNo;
    int marks;
    float percentage;
};

int main(){
    struct Student students[60];
    for(int i=0;i<60;i++){
        printf("Enter name of %d student: ",i+1);
        scanf("%s",students[i].name);
        printf("Enter roll no of %d student: ",i+1);
        scanf("%d",&students[i].rollNo);
        printf("Enter marks of %d student: ",i+1);
        scanf("%d",&students[i].marks);
        printf("Enter percentage of %d student: ",i+1);
        scanf("%f",&students[i].percentage);
    }
    printf("========ALL STUDENTS========\n");
    for(int i=0;i<60;i++){
        printf("Name: %s\nRoll no: %d\nMarks: %d\nPercentage:%f\n",students[i].name,
               students[i].rollNo,students[i].marks,students[i].percentage);
        printf("============\n");
    }
    return 0;
}
```

Can we assign one structure to another structure?

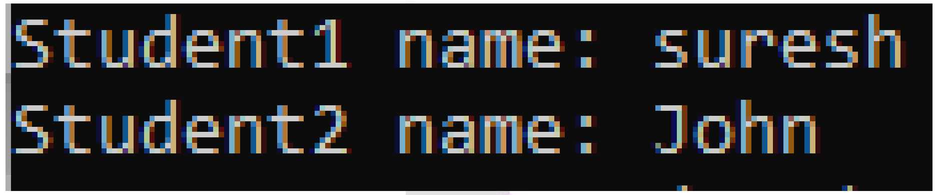


```c
#include<stdio.h>
struct Student{
    char name[20];
    int rollNo;
    int marks;
    float percentage;
};

int main(){
    struct Student student1={"suresh",10,892,89.2};
    struct Student student2=student1;
    printf("Student1 name: %s\n",student1.name);
    printf("Student2 name: %s",student2.name);
    return 0;
}
```



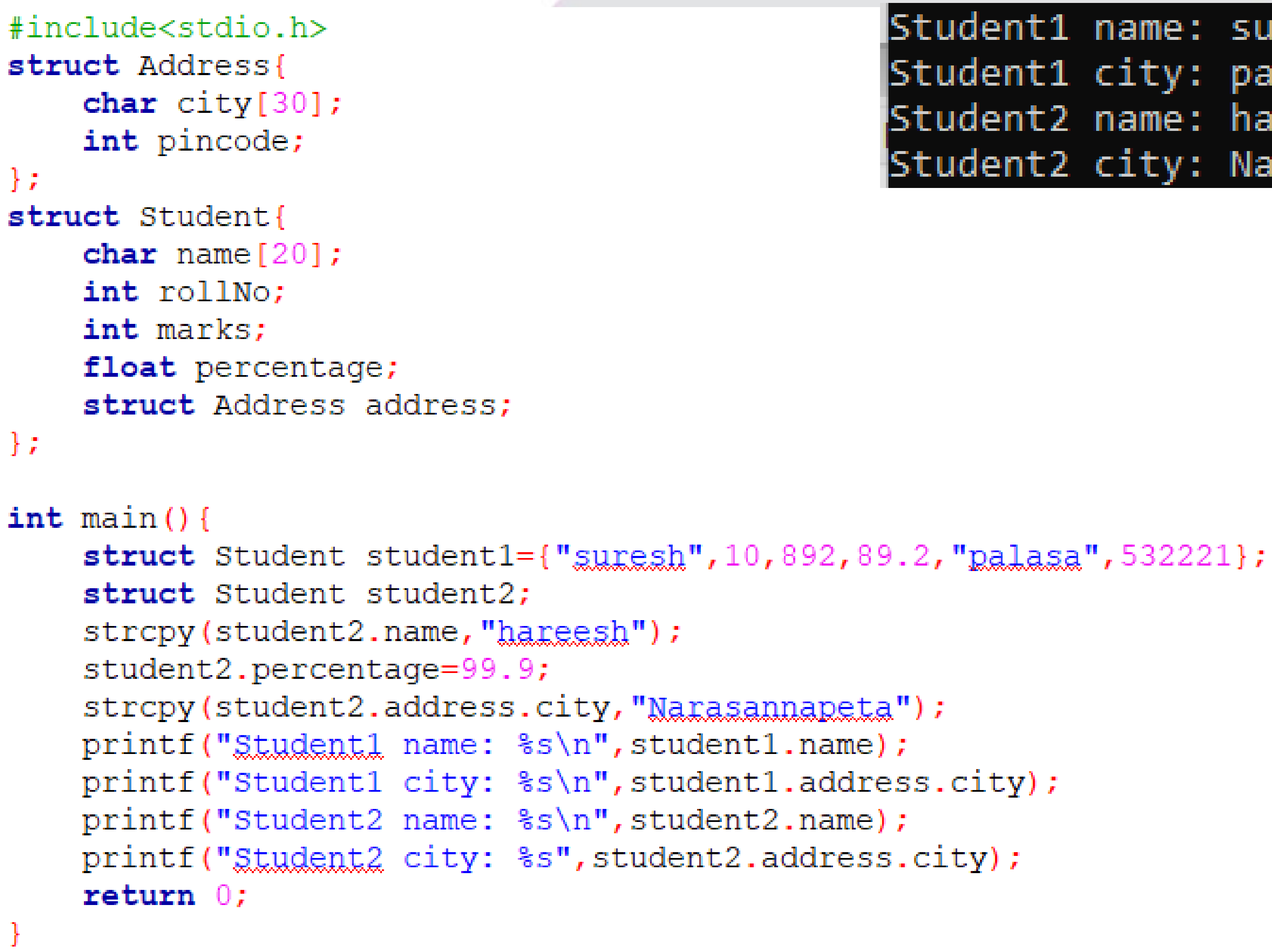
```
Student1 name: suresh
Student2 name: suresh
```

```c
#include<stdio.h>
struct Student{
    char name[20];
    int rollNo;
    int marks;
    float percentage;
};

int main(){
    struct Student student1={"suresh",10,892,89.2};
    struct Student student2=student1;
    strcpy(student2.name,"John");
    printf("Student1 name: %s\n",student1.name);
    printf("Student2 name: %s",student2.name);
    return 0;
}
```

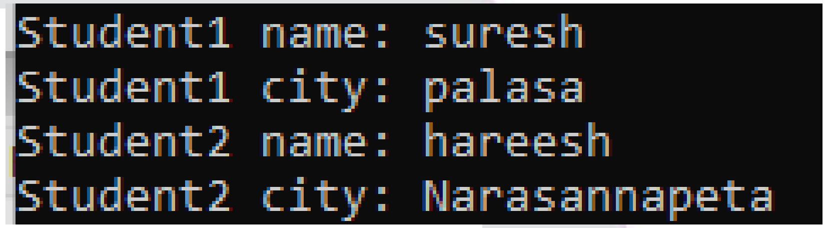


```
Student1 name: suresh
Student2 name: John
```

Nested structures



```c
#include<stdio.h>
struct Address{
    char city[30];
    int pincode;
};
struct Student{
    char name[20];
    int rollNo;
    int marks;
    float percentage;
    struct Address address;
};

int main(){
    struct Student student1={"suresh",10,892,89.2,"palasa",532221};
    struct Student student2;
    strcpy(student2.name,"hareesh");
    student2.percentage=99.9;
    strcpy(student2.address.city,"Narasannapeta");
    printf("Student1 name: %s\n",student1.name);
    printf("Student1 city: %s\n",student1.address.city);
    printf("Student2 name: %s\n",student2.name);
    printf("Student2 city: %s",student2.address.city);
    return 0;
}
```



```
Student1 name: suresh
Student1 city: palasa
Student2 name: hareesh
Student2 city: Narasannapeta
```

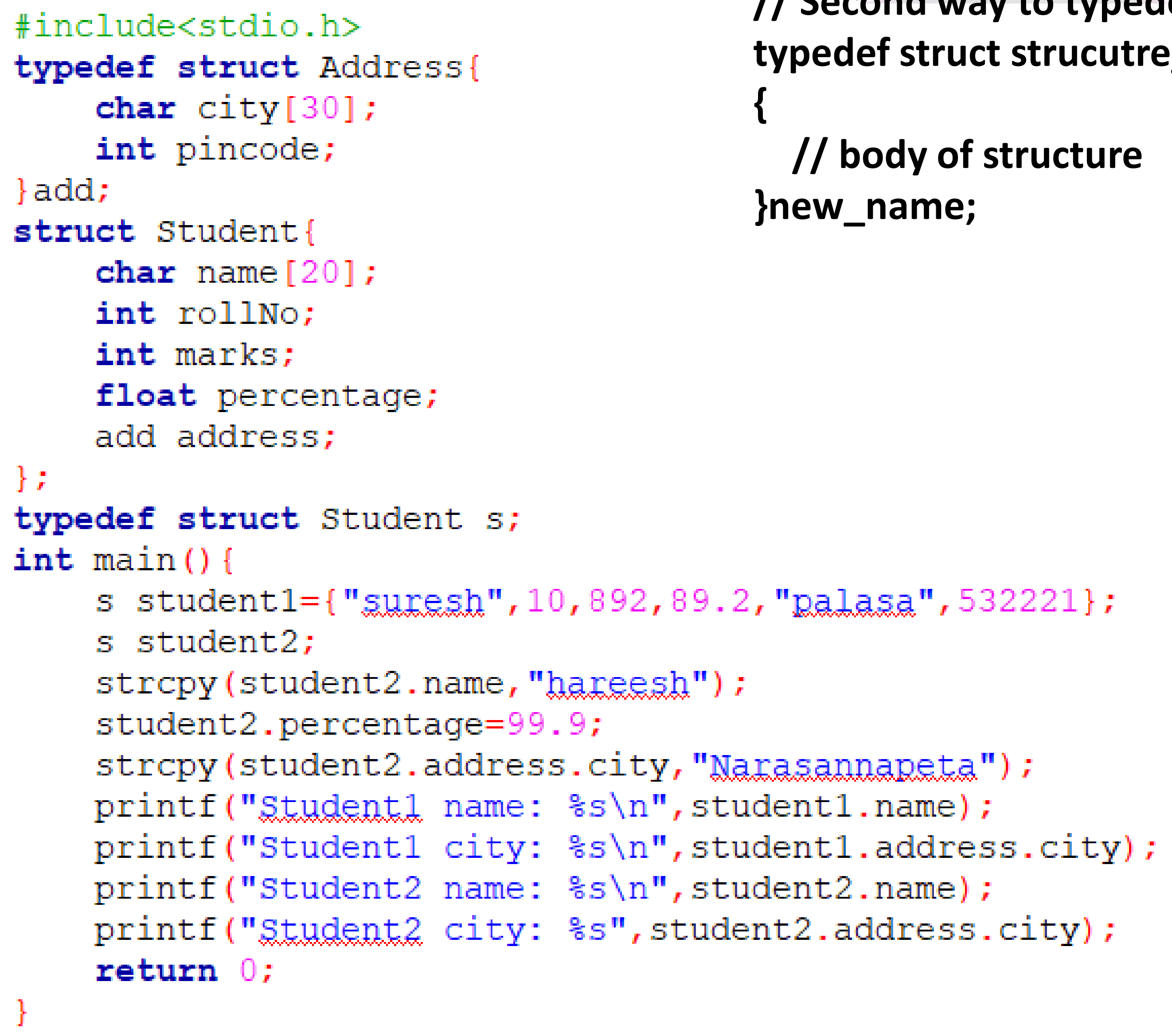
Typedef importance in structures

- Used to give nickname to the data type

```
// First way to typedef
typedef struct strucutre_name new_name;

// Second way to typedef
typedef struct strucutre_name
{
   // body of structure
}new_name;
```

```c
#include<stdio.h>
struct Address{
    char city[30];
    int pincode;
};
struct Student{
    char name[20];
    int rollNo;
    int marks;
    float percentage;
    struct Address address;
};
typedef struct Student s;
int main(){
    s student1={"suresh",10,892,89.2,"palasa",532221};
    s student2;
    strcpy(student2.name,"hareesh");
    student2.percentage=99.9;
    strcpy(student2.address.city,"Narasannapeta");
    printf("Student1 name: %s\n",student1.name);
    printf("Student1 city: %s\n",student1.address.city);
    printf("Student2 name: %s\n",student2.name);
    printf("Student2 city: %s",student2.address.city);
    return 0;
}
```

// First way to typedef
typedef struct strucutre_name new_name;



```c
#include<stdio.h>
typedef struct Address{
    char city[30];
    int pincode;
}add;
struct Student{
    char name[20];
    int rollNo;
    int marks;
    float percentage;
    add address;
};
typedef struct Student s;
int main(){
    s student1={"suresh",10,892,89.2,"palasa",532221};
    s student2;
    strcpy(student2.name,"hareesh");
    student2.percentage=99.9;
    strcpy(student2.address.city,"Narasannapeta");
    printf("Student1 name: %s\n",student1.name);
    printf("Student1 city: %s\n",student1.address.city);
    printf("Student2 name: %s\n",student2.name);
    printf("Student2 city: %s",student2.address.city);
    return 0;
}
```

```
// Second way to typedef
typedef struct strucutre_name
{
   // body of structure
}new_name;
```

What next?

- Unions