C - STRUCTURES

Dr. Sheak Rashed Haider Noori

Professor & Associate Head

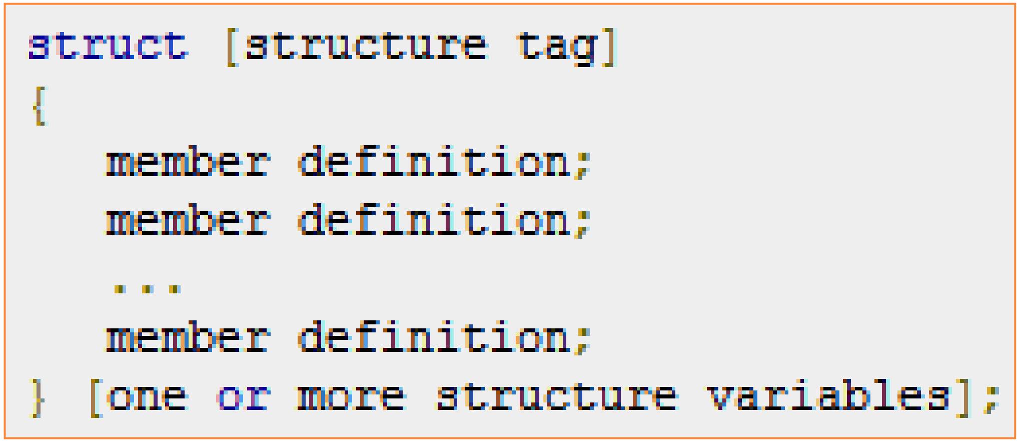
Department of Computer Science

C - STRUCTURES

- C arrays allow you to define type of variables that can hold several data items of the same kind but **structure** is another user defined data type available in C programming, which allows you to combine data items of different kinds.
- Structure is the collection of variables of different types under a single name for better handling.
- Structures are used to represent a record, Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book:
  - Title
  - Author
  - Subject
  - Book ID

DEFINING A STRUCTURE

- To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member for your program.
- The format of the **struct** statement is this:



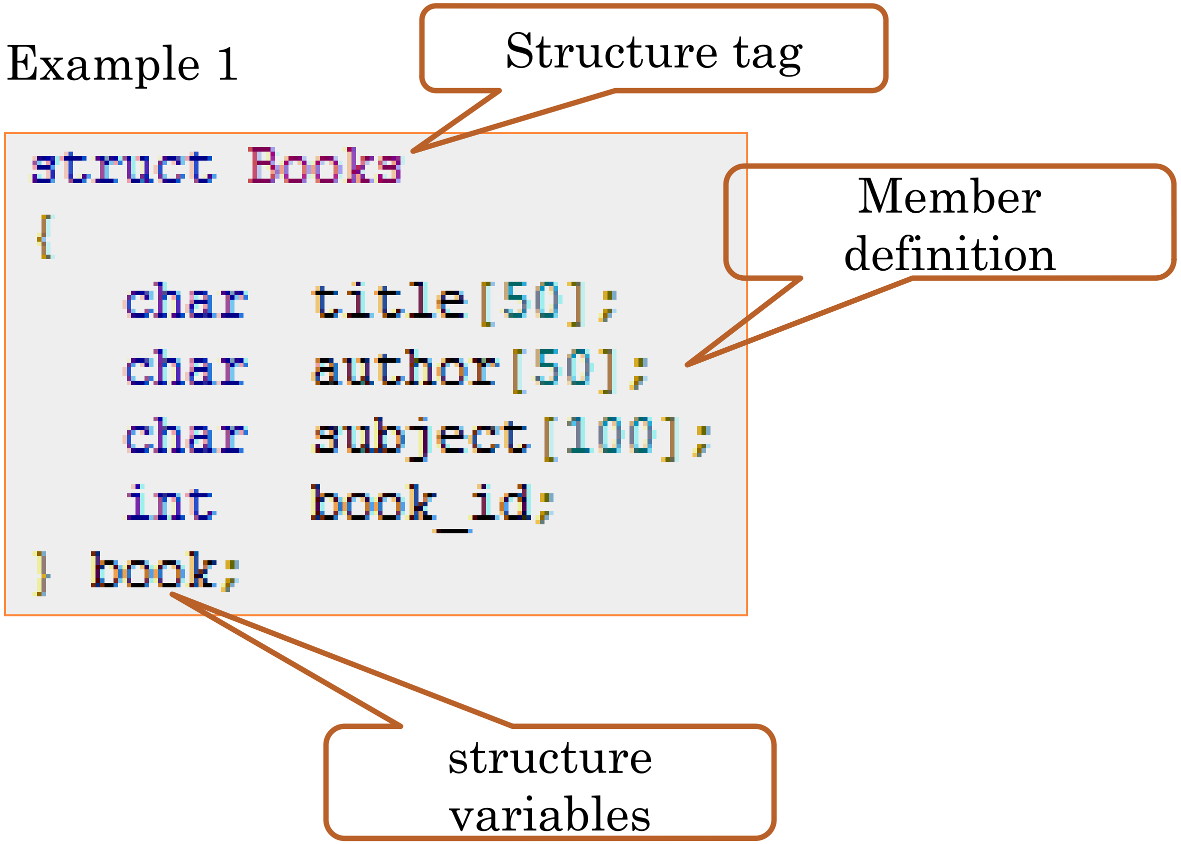
```
struct [structure tag]
{
   member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
```

- The **structure tag** is optional and each member definition is a normal variable definition, such as *int i*; or *float f*; or any other valid variable definition.
- At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional.

DEFINING A STRUCTURE

- Here is the way you would declare the Book structure:

Example 1



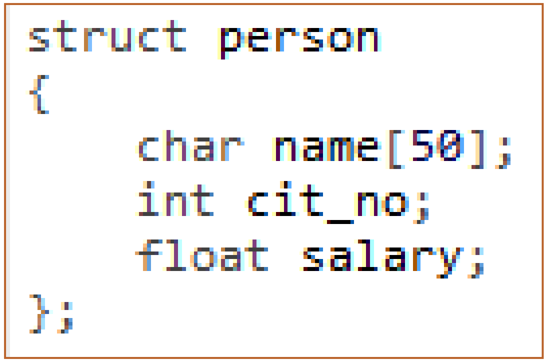
```
struct Books
{
    char  title[50];
    char  author[50];
    char  subject[100];
    int   book_id;
} book;
```

Structure tag

Member definition

structure variables

Example 2

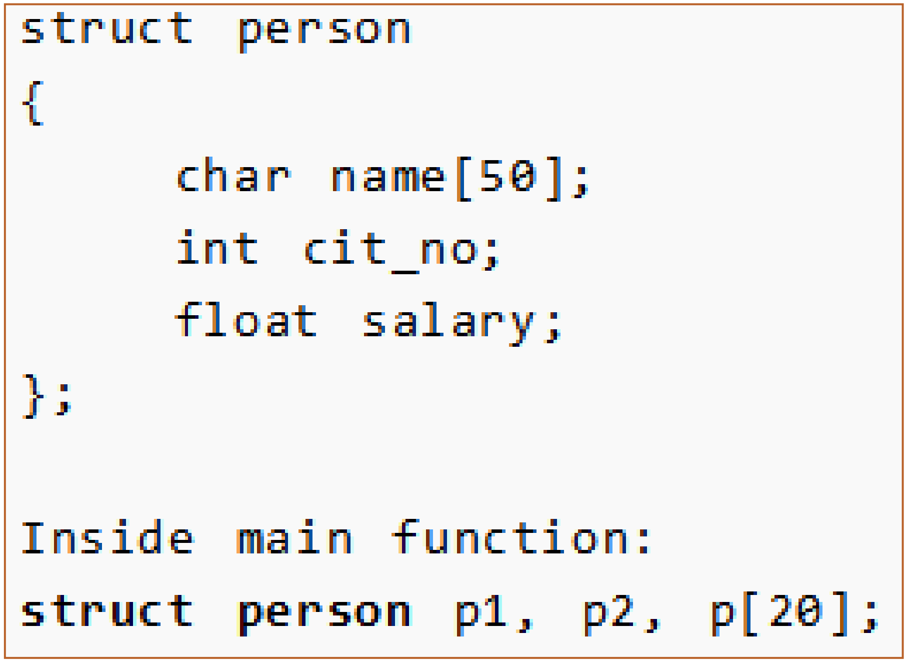


```
struct person
{
        char name[50];
        int cit_no;
        float salary;
};
```

- With the declaration of the structure you have created a new type, called **Books**.

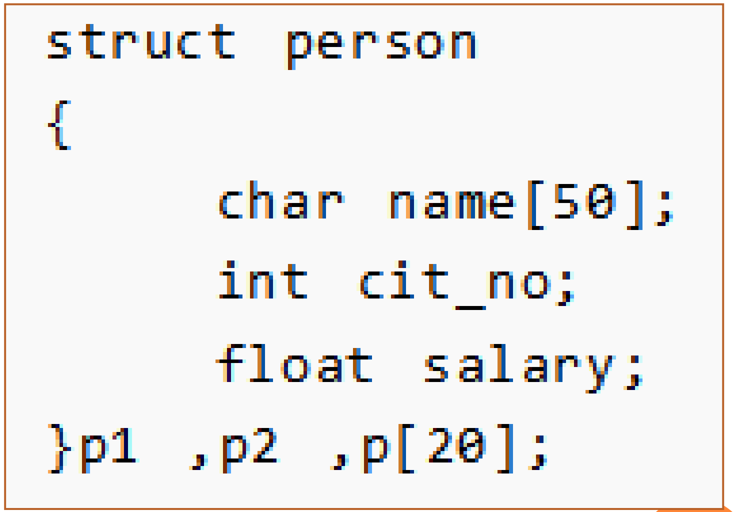
STRUCTURE VARIABLE DECLARATION

- When a structure is defined, it creates a user-defined type but, no storage is allocated.
- For the above structure of person, variable can be declared as:



```
struct person
{
      char name[50];
      int cit_no;
      float salary;
};

Inside main function:
struct person p1, p2, p[20];
```

Another way of creating structure variable is:

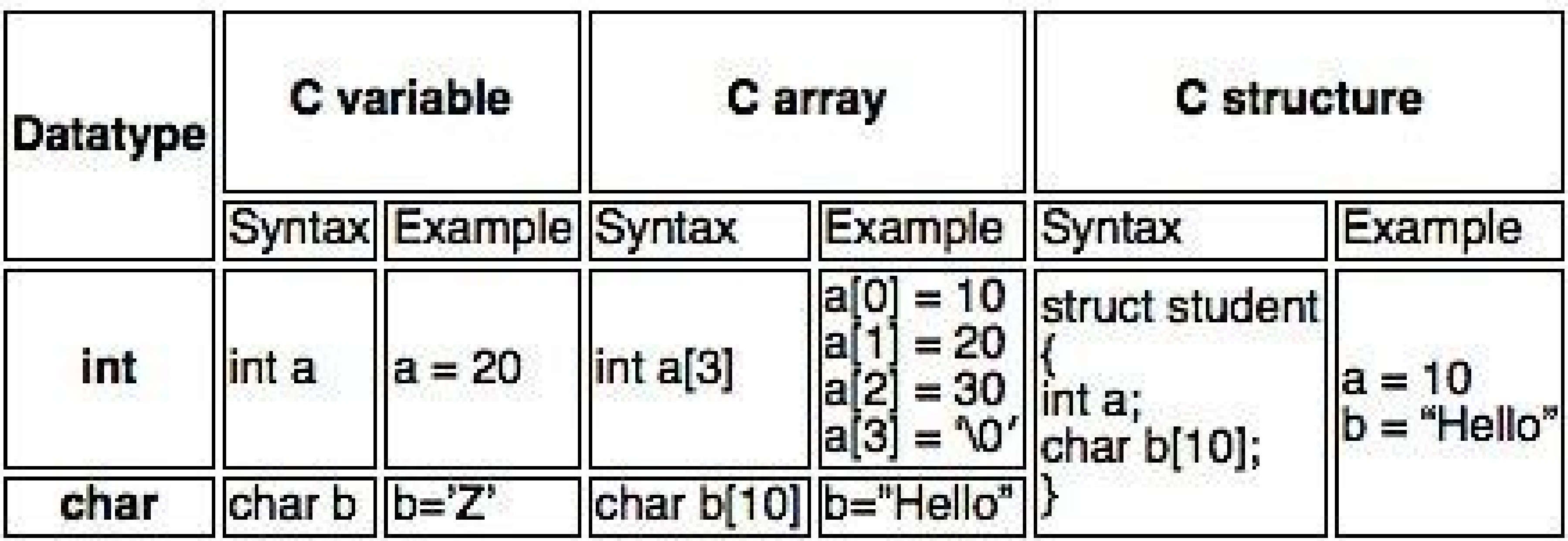


```
struct person
{
      char name[50];
      int cit_no;
      float salary;
}p1 ,p2 ,p[20];
```

In both cases, 2 variables *p1*, *p2* and array *p* having 20 elements of type **struct person** are created.

DIFFERENCE BETWEEN C VARIABLE, ARRAY AND STRUCTURE

- A normal C variable can hold only one data of one data type at a time.
- An array can hold group of data of same data type.
- A structure can hold group of data of different data types
- Data types can be int, char, float, double and long double etc.



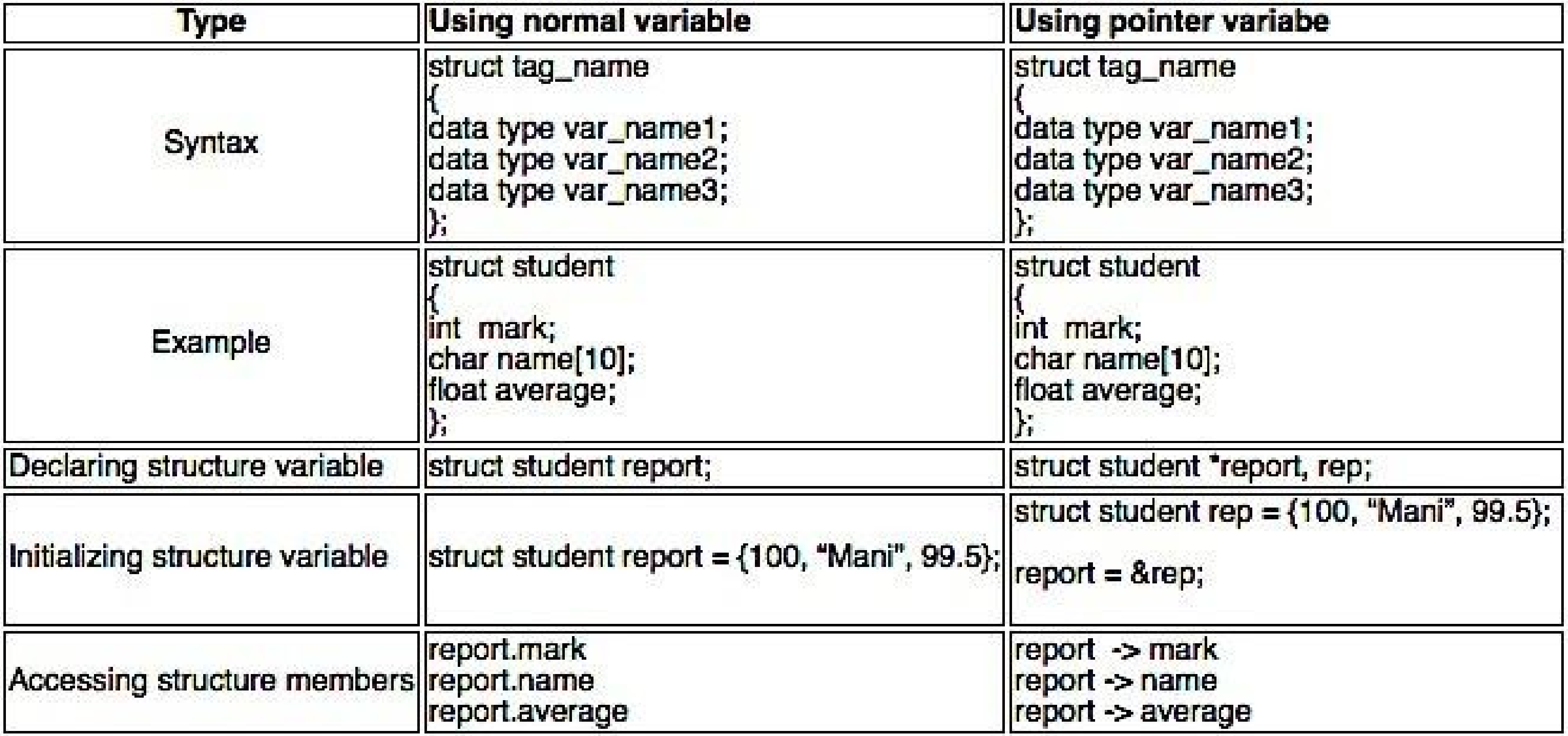
Datatype	C variable		C array		C structure	
	Syntax	Example	Syntax	Example	Syntax	Example

int	int a	a = 20	int a[3]	a[0] = 10
				a[1] = 20
				a[2] = 30
				a[3] = '\0'
					struct student
					{
					int a;
					char b[10];
					}
						a = 10
						b = "Hello"

char	char b	b='Z'	char b[10]	b="Hello"		

BELOW TABLE EXPLAINS FOLLOWING CONCEPTS IN C STRUCTURE

- How to declare a C structure?
- How to initialize a C structure?
- How to access the members of a C structure?



Type	Using normal variable	Using pointer variabe

Syntax	struct tag_name
	{
	data type var_name1;
	data type var_name2;
	data type var_name3;
	};
		struct tag_name
		{
		data type var_name1;
		data type var_name2;
		data type var_name3;
		};

Example	struct student
	{
	int mark;
	char name[10];
	float average;
	};
		struct student
		{
		int mark;
		char name[10];
		float average;
		};

Declaring structure variable	struct student report;	struct student *report, rep;

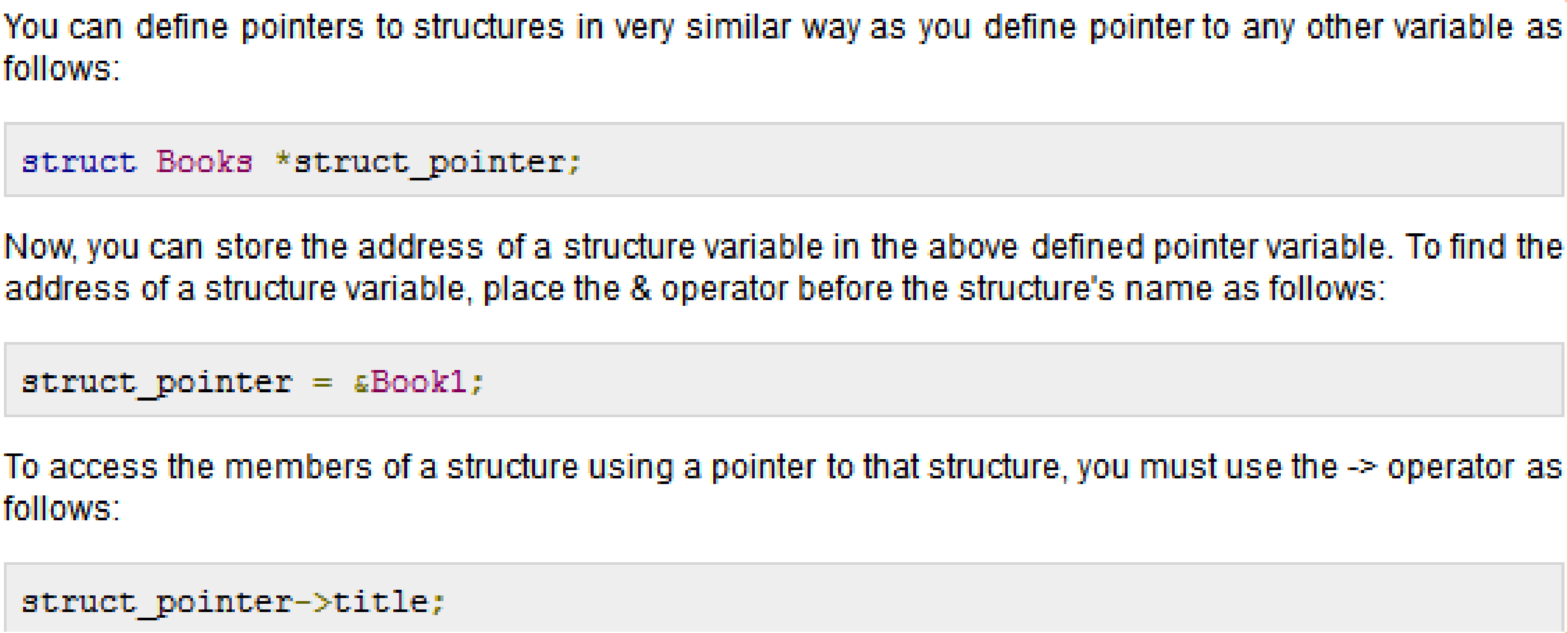
Initializing structure variable	struct student report = {100, “Mani”, 99.5};	struct student rep = {100, “Mani”, 99.5};
		report = &rep;

Accessing structure members	report.mark report.name report.average	report -> mark report -> name report -> average

ACCESSING MEMBERS OF A STRUCTURE

- Structure can be accessed in 2 ways. They are,
  1. Using normal structure variable
  2. Using pointer variable
- There are two types of operators used for accessing members of a structure.
  - Member operator(.)
  - Structure pointer operator(->)
- Dot(.) operator is used to access the data using normal structure variable.
- Arrow (->) is used to access the data using pointer variable.
- We already have learnt how to access structure data using normal variable. So, we are showing here how to access structure data using pointer variable.

POINTERS TO STRUCTURES



You can define pointers to structures in very similar way as you define pointer to any other variable as follows:

```
struct Books *struct_pointer;
```

Now, you can store the address of a structure variable in the above defined pointer variable. To find the address of a structure variable, place the & operator before the structure's name as follows:

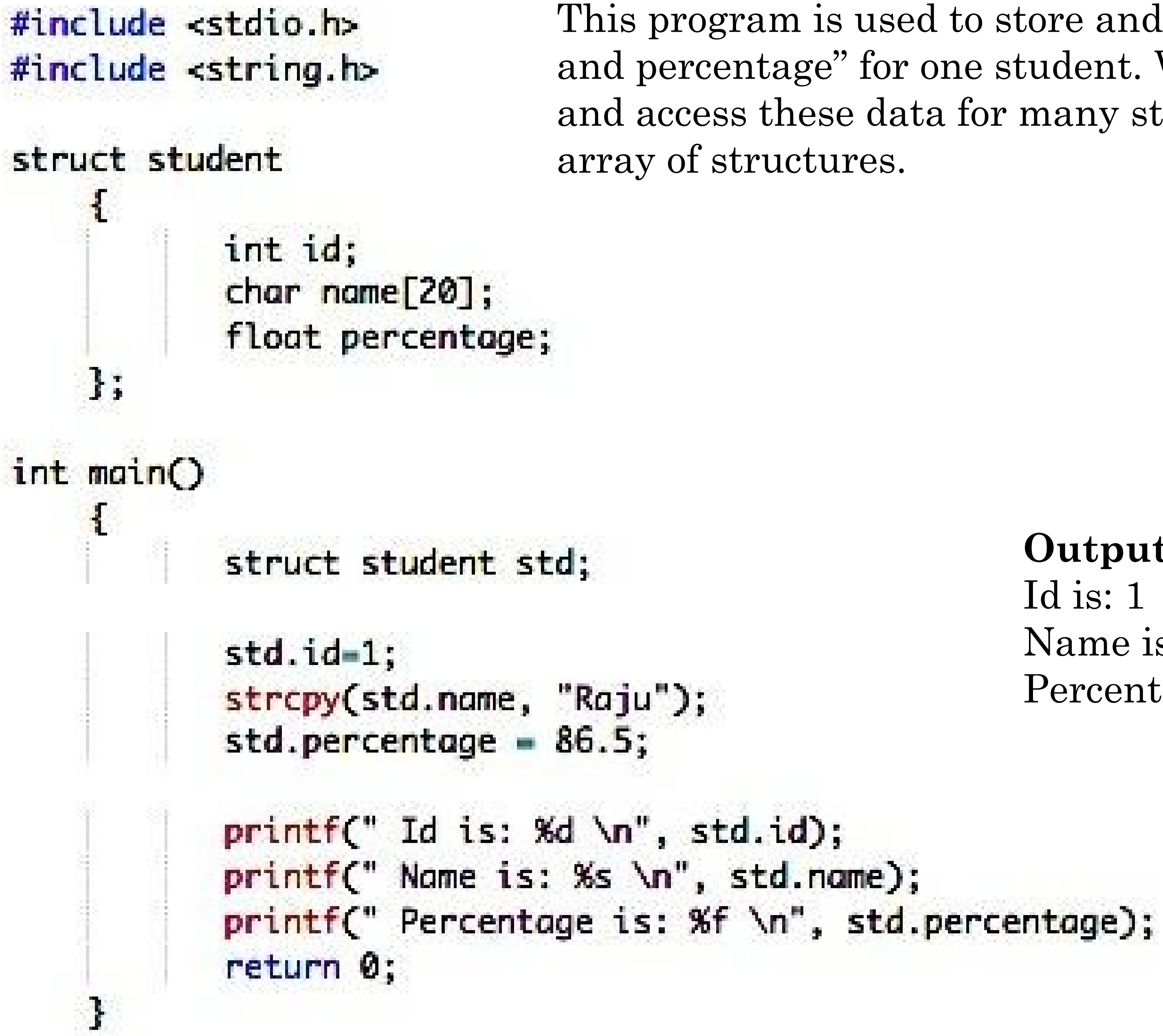
```
struct_pointer = &Book1;
```

To access the members of a structure using a pointer to that structure, you must use the -> operator as follows:

```
struct_pointer->title;
```

EXAMPLE PROGRAM FOR C STRUCTURE

This program is used to store and access “id, name and percentage” for one student. We can also store and access these data for many students using array of structures.



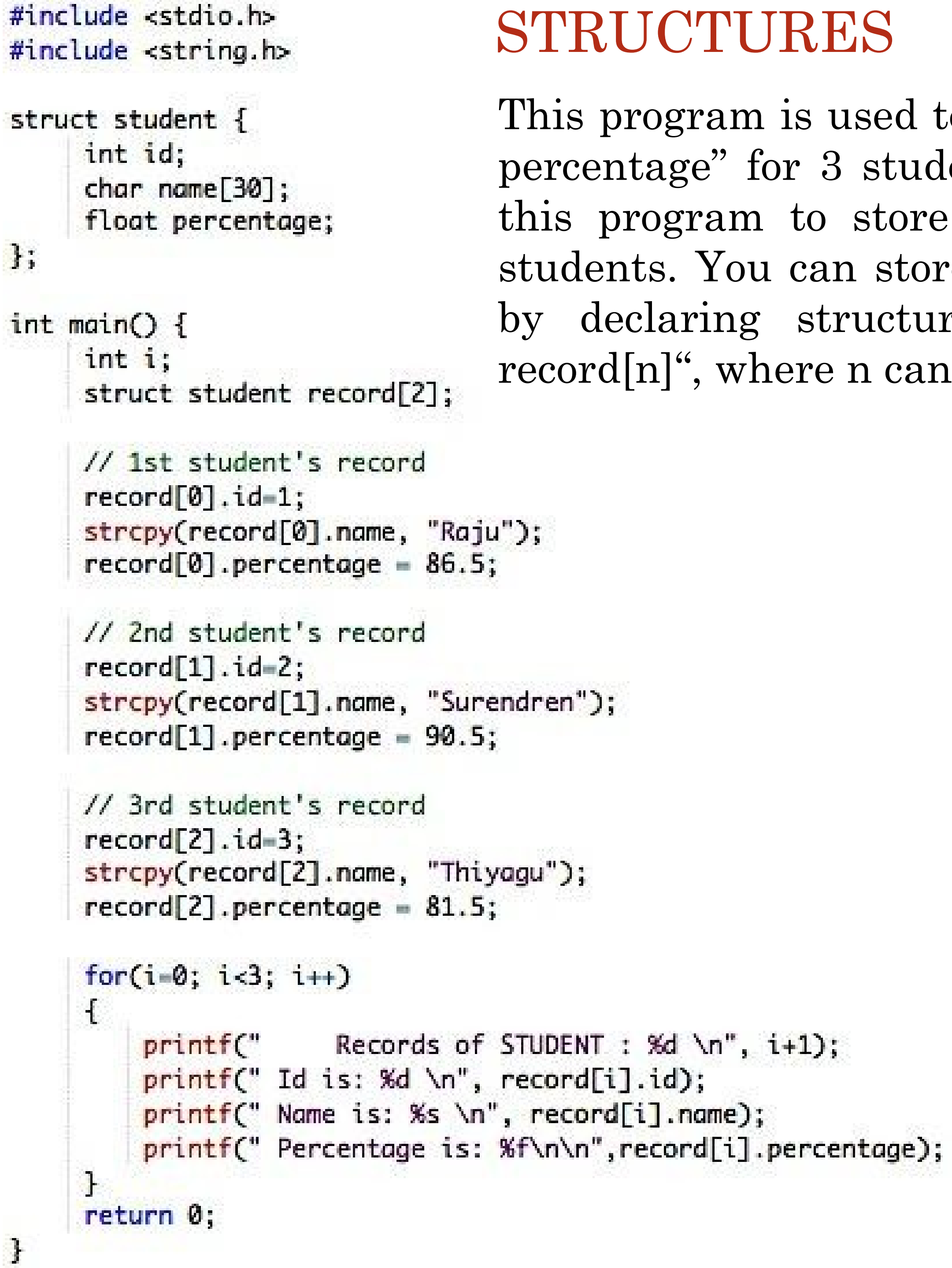
```
#include <stdio.h>
#include <string.h>

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

int main()
    {
            struct student std;

            std.id=1;
            strcpy(std.name, "Raju");
            std.percentage = 86.5;

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

Output:

```
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

STRUCTURES

This program is used to store and access “id, name and percentage” for 3 students. Structure array is used in this program to store and display records for many students. You can store “n” number of students record by declaring structure variable as ‘struct student record[n]“, where n can be 1000 or 5000 etc.



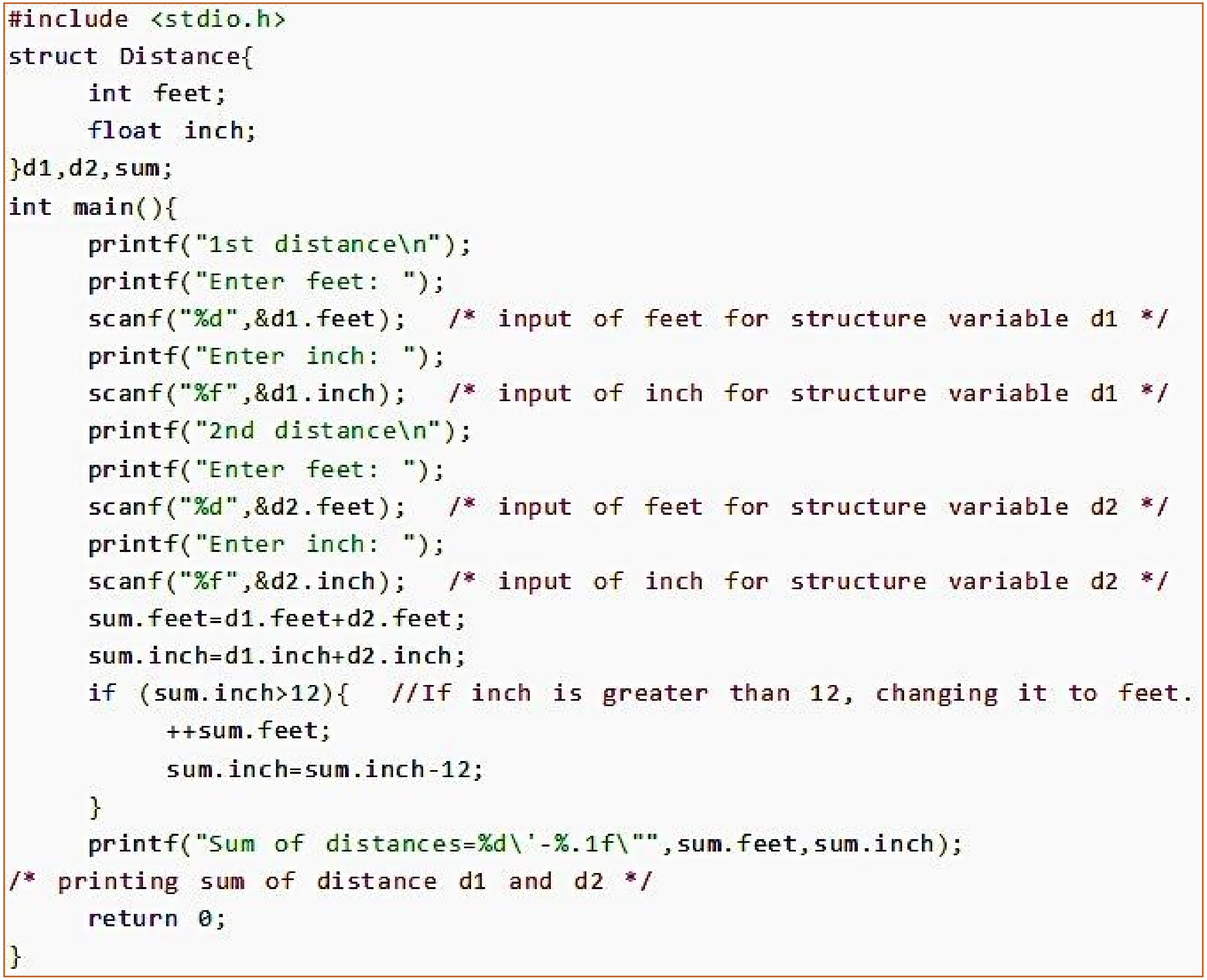
```c
#include <stdio.h>
#include <string.h>

struct student {
    int id;
    char name[30];
    float percentage;
};

int main() {
    int i;
    struct student record[2];

    // 1st student's record
    record[0].id=1;
    strcpy(record[0].name, "Raju");
    record[0].percentage = 86.5;

    // 2nd student's record
    record[1].id=2;
    strcpy(record[1].name, "Surendren");
    record[1].percentage = 90.5;

    // 3rd student's record
    record[2].id=3;
    strcpy(record[2].name, "Thiyagu");
    record[2].percentage = 81.5;

    for(i=0; i<3; i++)
    {
        printf("     Records of STUDENT : %d \n", i+1);
        printf(" Id is: %d \n", record[i].id);
        printf(" Name is: %s \n", record[i].name);
        printf(" Percentage is: %f\n\n",record[i].percentage);
    }
    return 0;
}
```

Output:

```
   Records of STUDENT : 1
Id is: 1
Name is: Raju
Percentage is: 86.500000
   Records of STUDENT : 2
Id is: 2
Name is: Surendren
Percentage is: 90.500000
   Records of STUDENT : 3
Id is: 3 Name is: Thiyagu
Percentage is: 81.500000
```

EXAMPLE PROGRAM OF STRUCTURE

Write a C program to add two distances entered by user. Measurement of distance should be in inch and feet.(Note: 12 inches = 1 foot)



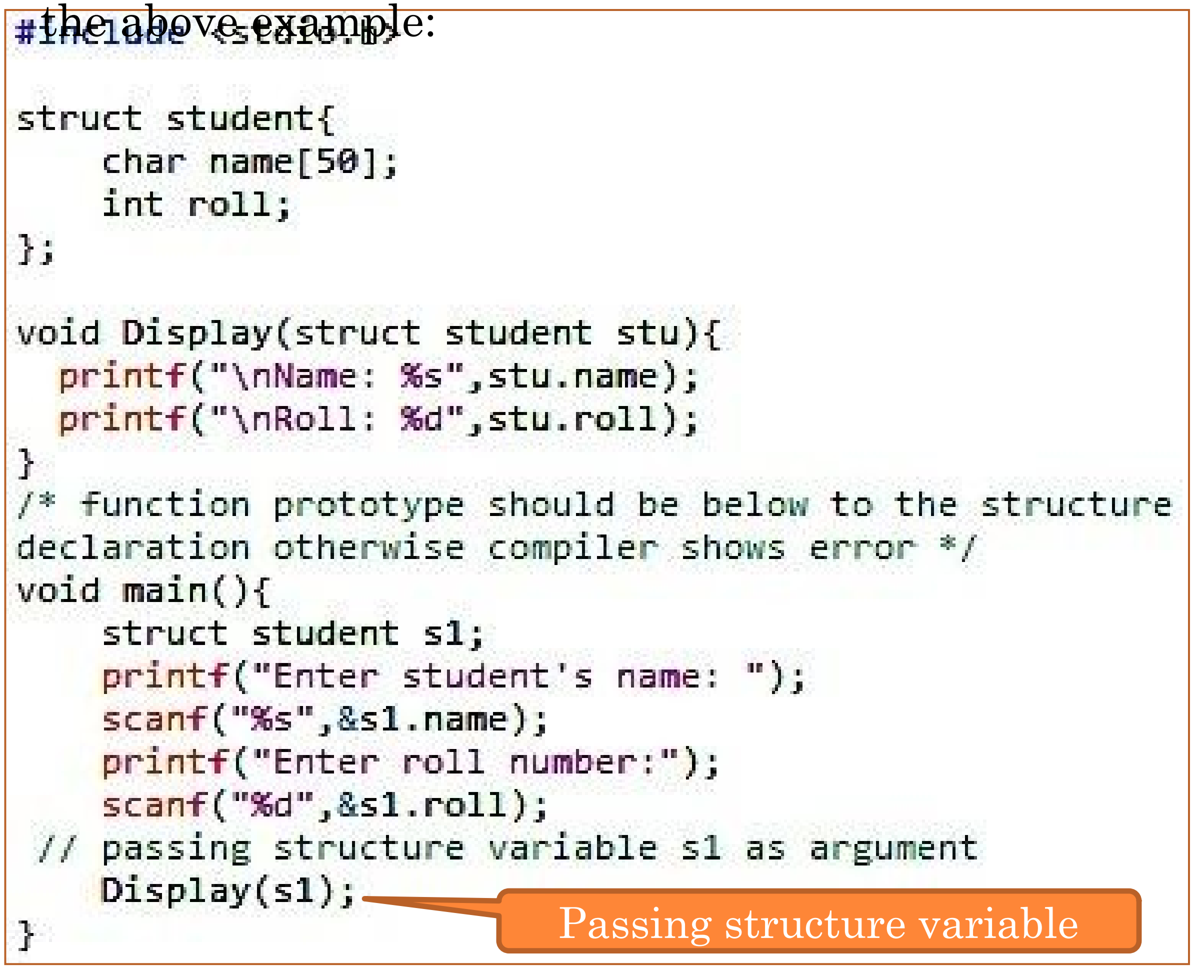
```c
#include <stdio.h>
struct Distance{
     int feet;
     float inch;
}d1,d2,sum;
int main(){
     printf("1st distance\n");
     printf("Enter feet: ");
     scanf("%d",&d1.feet);   /* input of feet for structure variable d1 */
     printf("Enter inch: ");
     scanf("%f",&d1.inch);   /* input of inch for structure variable d1 */
     printf("2nd distance\n");
     printf("Enter feet: ");
     scanf("%d",&d2.feet);   /* input of feet for structure variable d2 */
     printf("Enter inch: ");
     scanf("%f",&d2.inch);   /* input of inch for structure variable d2 */
     sum.feet=d1.feet+d2.feet;
     sum.inch=d1.inch+d2.inch;
     if (sum.inch>12){   //If inch is greater than 12, changing it to feet.
          ++sum.feet;
          sum.inch=sum.inch-12;
     }
     printf("Sum of distances=%d\'-%.1f\"",sum.feet,sum.inch);
/* printing sum of distance d1 and d2 */
     return 0;
}
```

PASSING STRUCTURE TO FUNCTION

- A structure can be passed to any function from main function or from any sub function.
- Structure definition will be available within the function only.
- It won't be available to other functions unless it is passed to those functions by value or by address(reference).
- Else, we have to declare structure variable as global variable. That means, structure variable should be declared outside the main function. So, this structure will be visible to all the functions in a C program.
- **Passing structure to function in C:** It can be done in below 3 ways.
  1. Passing structure to a function by value
  2. Passing structure to a function by address(reference)
  3. No need to pass a structure – Declare structure variable as global

EXAMPLE – PASSING STRUCTURE TO FUNCTION BY VALUE

- A structure variable can be passed to the function as an argument as normal variable.
- If structure is passed by value, change made in structure variable in function definition does not reflect in original structure variable in calling function.
- You would access structure variables in the similar way as you have accessed in the above example:



```
#include <stdio.h>

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

void Display(struct student stu){
  printf("\nName: %s",stu.name);
  printf("\nRoll: %d",stu.roll);
}
/* function prototype should be below to the structure
declaration otherwise compiler shows error */
void main(){
    struct student s1;
    printf("Enter student's name: ");
    scanf("%s",&s1.name);
    printf("Enter roll number:");
    scanf("%d",&s1.roll);
 // passing structure variable s1 as argument
    Display(s1);
}
```

Passing structure variable

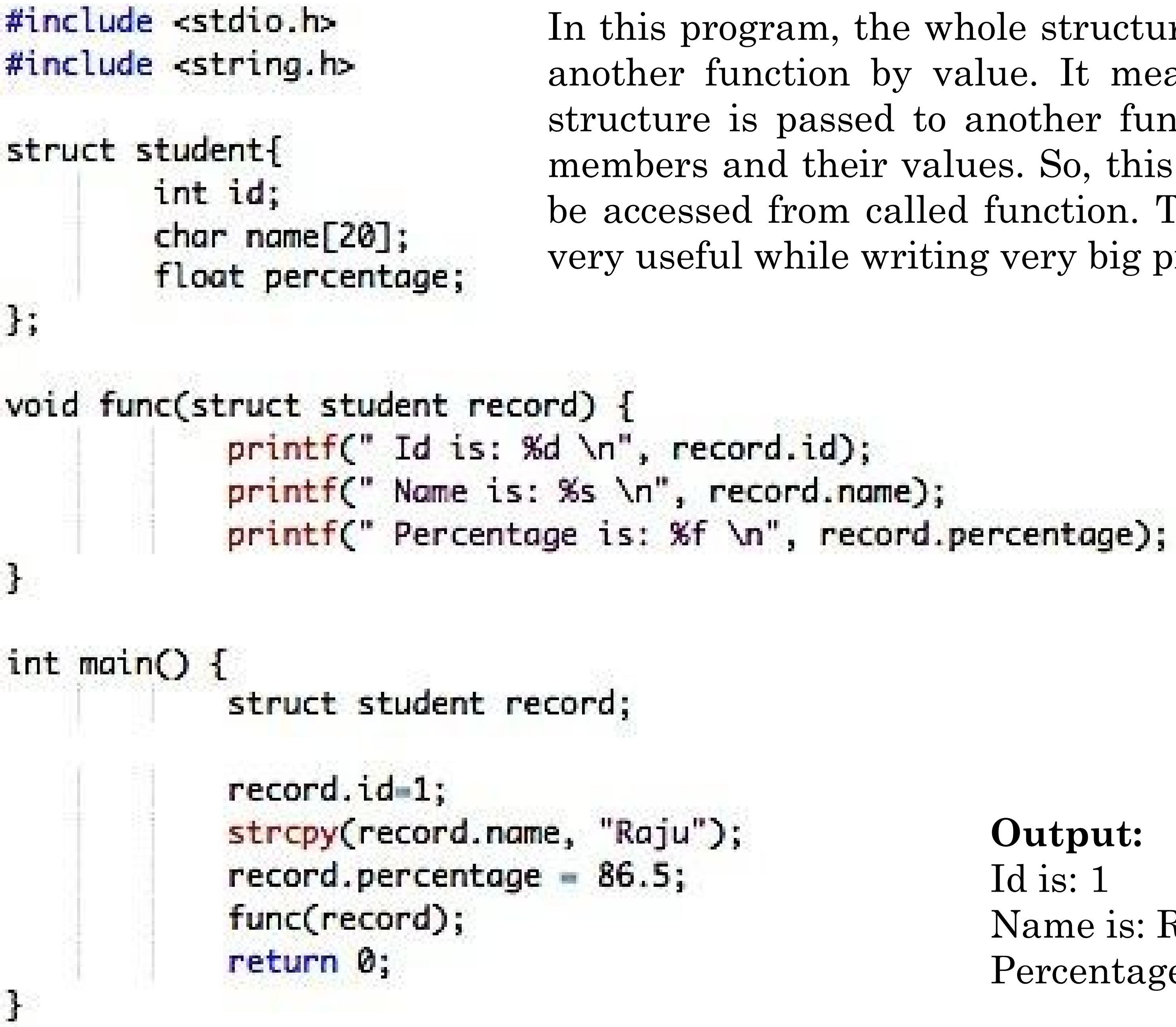
Write a C program to create a structure student, containing name and roll. Ask user the name and roll of a student in main function. Pass this structure to a function and display the information in that function.

Output:

```
Enter student's name: Kevin
Enter roll number: 149
Name: Kevin
Roll: 149
```

EXAMPLE – PASSING STRUCTURE TO FUNCTION BY VALUE

In this program, the whole structure is passed to another function by value. It means the whole structure is passed to another function with all members and their values. So, this structure can be accessed from called function. This concept is very useful while writing very big programs in C.

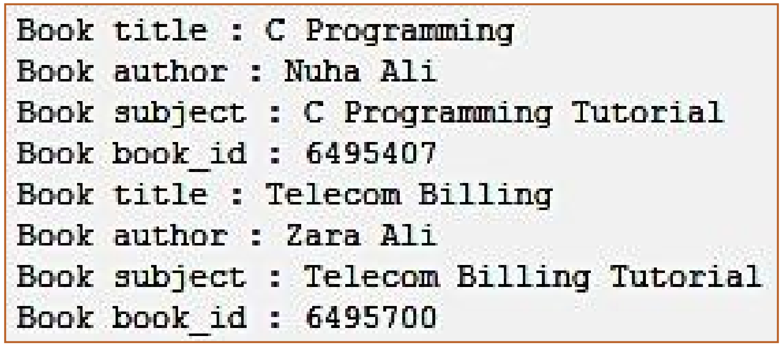


```c
#include <stdio.h>
#include <string.h>

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

void func(struct student record) {
            printf(" Id is: %d \n", record.id);
            printf(" Name is: %s \n", record.name);
            printf(" Percentage is: %f \n", record.percentage);
}

int main() {
            struct student record;

            record.id=1;
            strcpy(record.name, "Raju");
            record.percentage = 86.5;
            func(record);
            return 0;
}
```

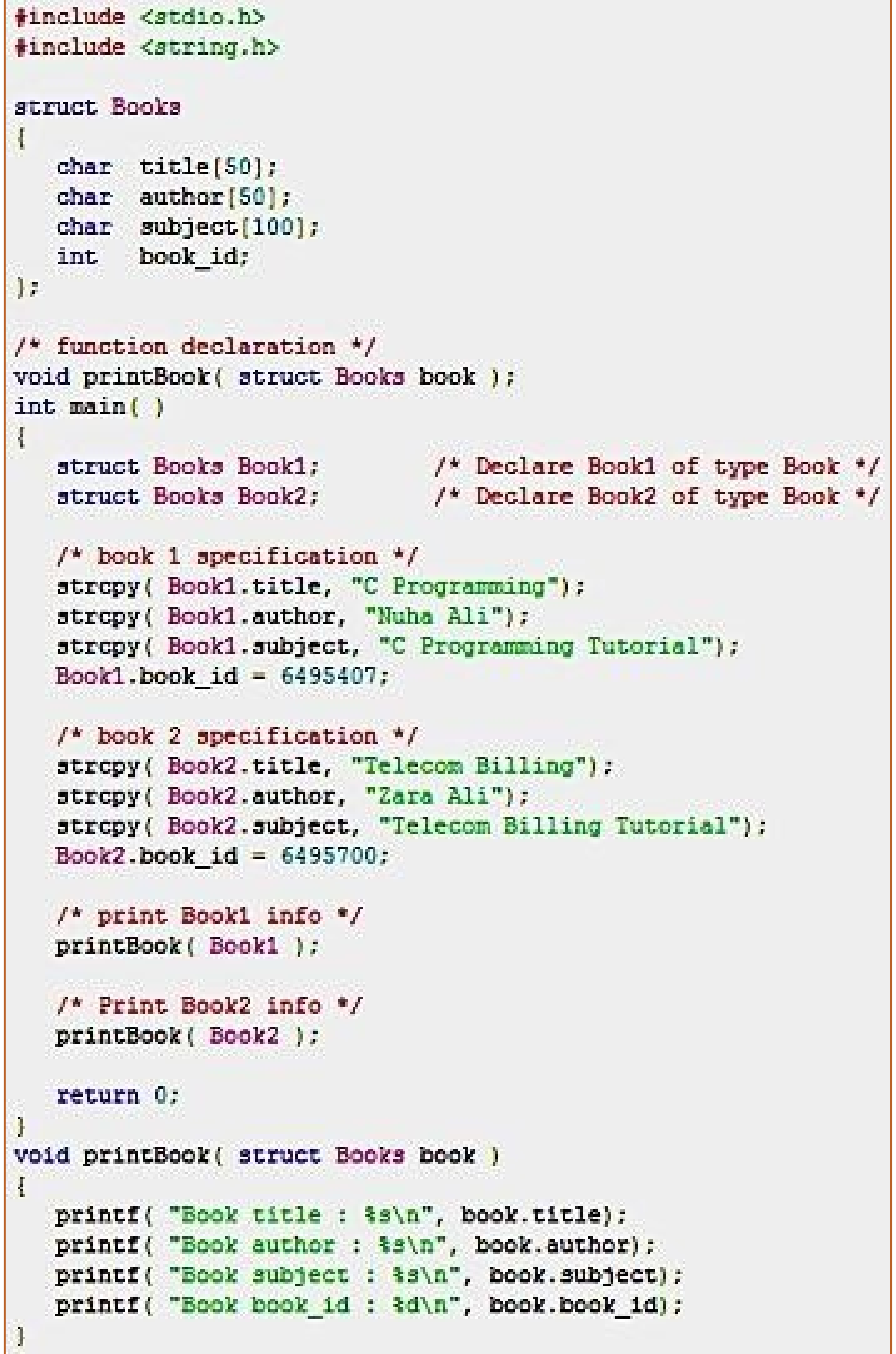
Output:

```
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

EXAMPLE – PASSING STRUCTURE TO FUNCTION BY VALUE



```
Book title : C Programming
Book author : Nuha Ali
Book subject : C Programming Tutorial
Book book_id : 6495407
Book title : Telecom Billing
Book author : Zara Ali
Book subject : Telecom Billing Tutorial
Book book_id : 6495700
```

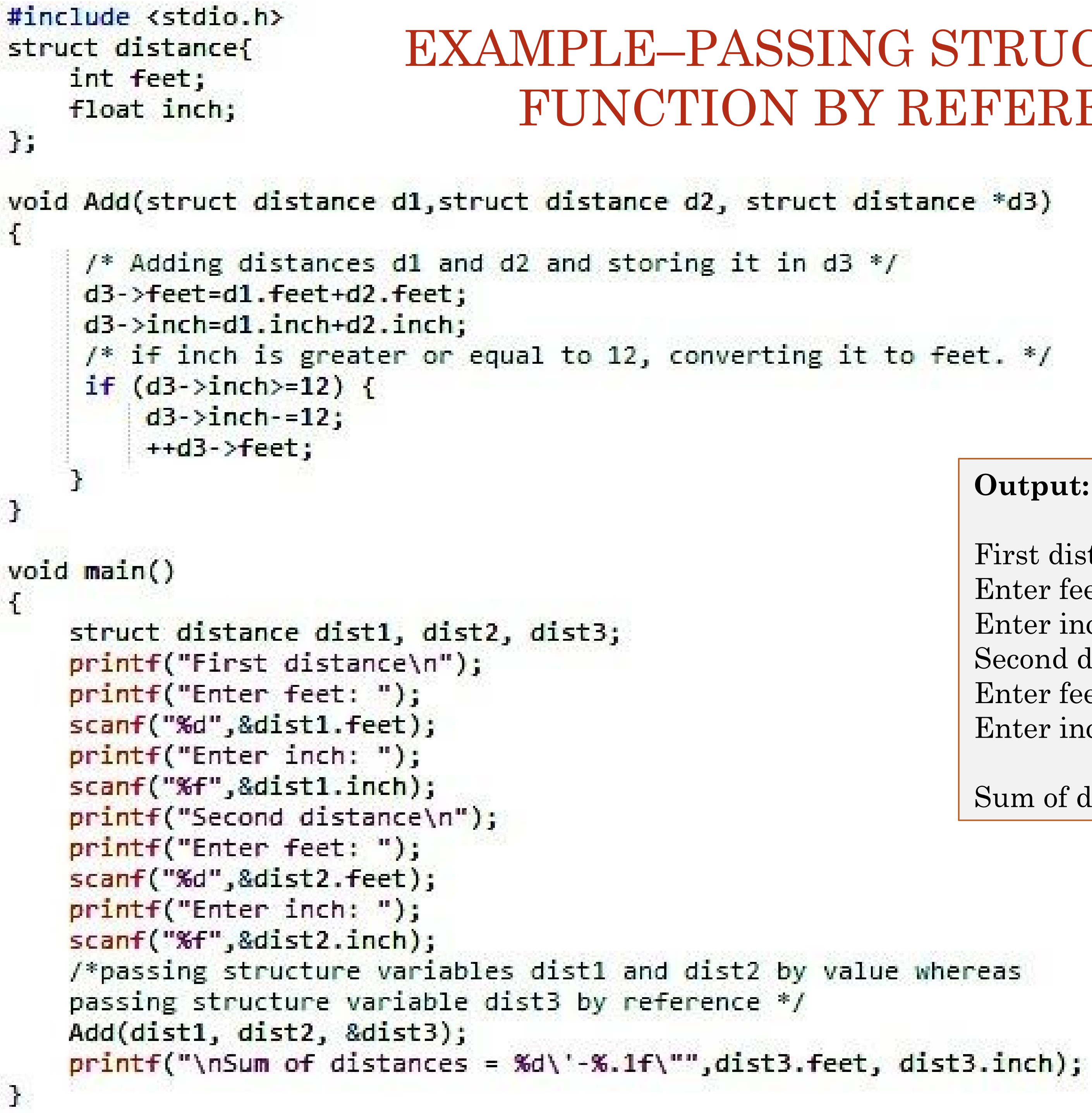


```
#include <stdio.h>
#include <string.h>

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, "C Programming");
   strcpy( Book1.author, "Nuha Ali");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 6495407;

   /* book 2 specification */
   strcpy( Book2.title, "Telecom Billing");
   strcpy( Book2.author, "Zara Ali");
   strcpy( Book2.subject, "Telecom Billing Tutorial");
   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);
}
```

PASSING STRUCTURE TO FUNCTION BY ADDRESS/REFERENCE

- The **address** location of **structure** variable is passed to function while passing it by reference.
- If structure is passed by reference, change made in structure variable in function definition reflects in original structure variable in the calling function.
- Exercise: Write a C program to add two distances(feet-inch system) entered by user. To solve this program, make a structure. Pass two structure variable (containing distance in feet and inch) to add function by reference and display the result in main function without returning it.

EXAMPLE–PASSING STRUCTURE TO FUNCTION BY REFERENCE



```c
#include <stdio.h>
struct distance{
    int feet;
    float inch;
};

void Add(struct distance d1,struct distance d2, struct distance *d3)
{
     /* Adding distances d1 and d2 and storing it in d3 */
     d3->feet=d1.feet+d2.feet;
     d3->inch=d1.inch+d2.inch;
     /* if inch is greater or equal to 12, converting it to feet. */
     if (d3->inch>=12) {
         d3->inch-=12;
         ++d3->feet;
    }
}

void main()
{
    struct distance dist1, dist2, dist3;
    printf("First distance\n");
    printf("Enter feet: ");
    scanf("%d",&dist1.feet);
    printf("Enter inch: ");
    scanf("%f",&dist1.inch);
    printf("Second distance\n");
    printf("Enter feet: ");
    scanf("%d",&dist2.feet);
    printf("Enter inch: ");
    scanf("%f",&dist2.inch);
    /*passing structure variables dist1 and dist2 by value whereas
    passing structure variable dist3 by reference */
    Add(dist1, dist2, &dist3);
    printf("\nSum of distances = %d\'-%.1f\"",dist3.feet, dist3.inch);
}
```

Output:

```
First distance
Enter feet: 12
Enter inch: 6.8
Second distance
Enter feet: 5
Enter inch: 7.5

Sum of distances = 18'-2.3"
```

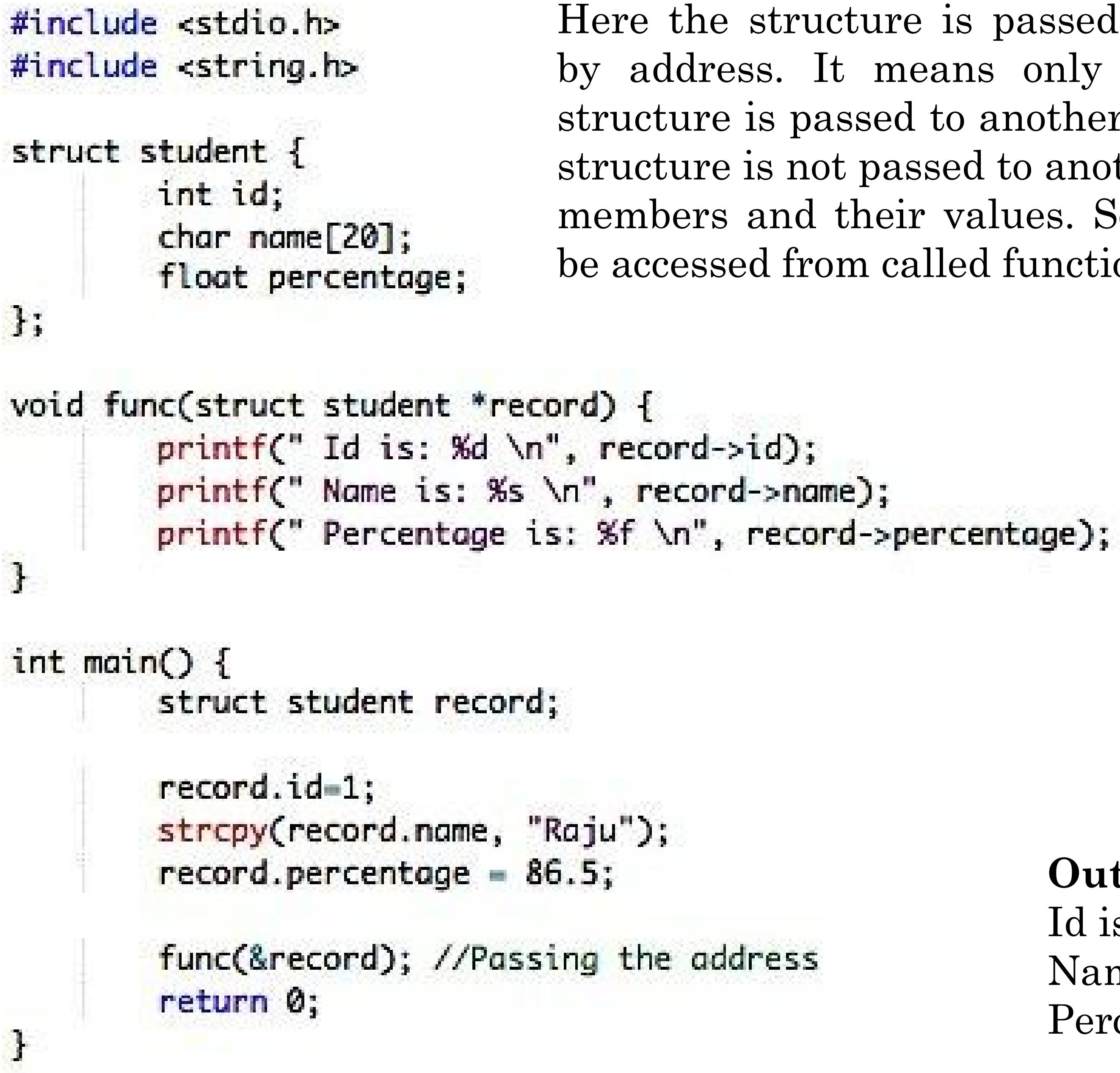
Explanation of previous example

In the previous program, structure variables dist1 and dist2 are passed by value (because value of dist1 and dist2 does not need to be displayed in main function) and dist3 is passed by reference ,i.e, address of dist3 (&dist3) is passed as an argument.

Thus, the structure pointer variable d3 points to the address of dist3. If any change is made in d3 variable, effect of it is seed in dist3 variable in main function.

EXAMPLE–PASSING STRUCTURE TO FUNCTION BY ADDRESS/REFERENCE

Here the structure is passed to another function by address. It means only the address of the structure is passed to another function. The whole structure is not passed to another function with all members and their values. So, this structure can be accessed from called function by its address.

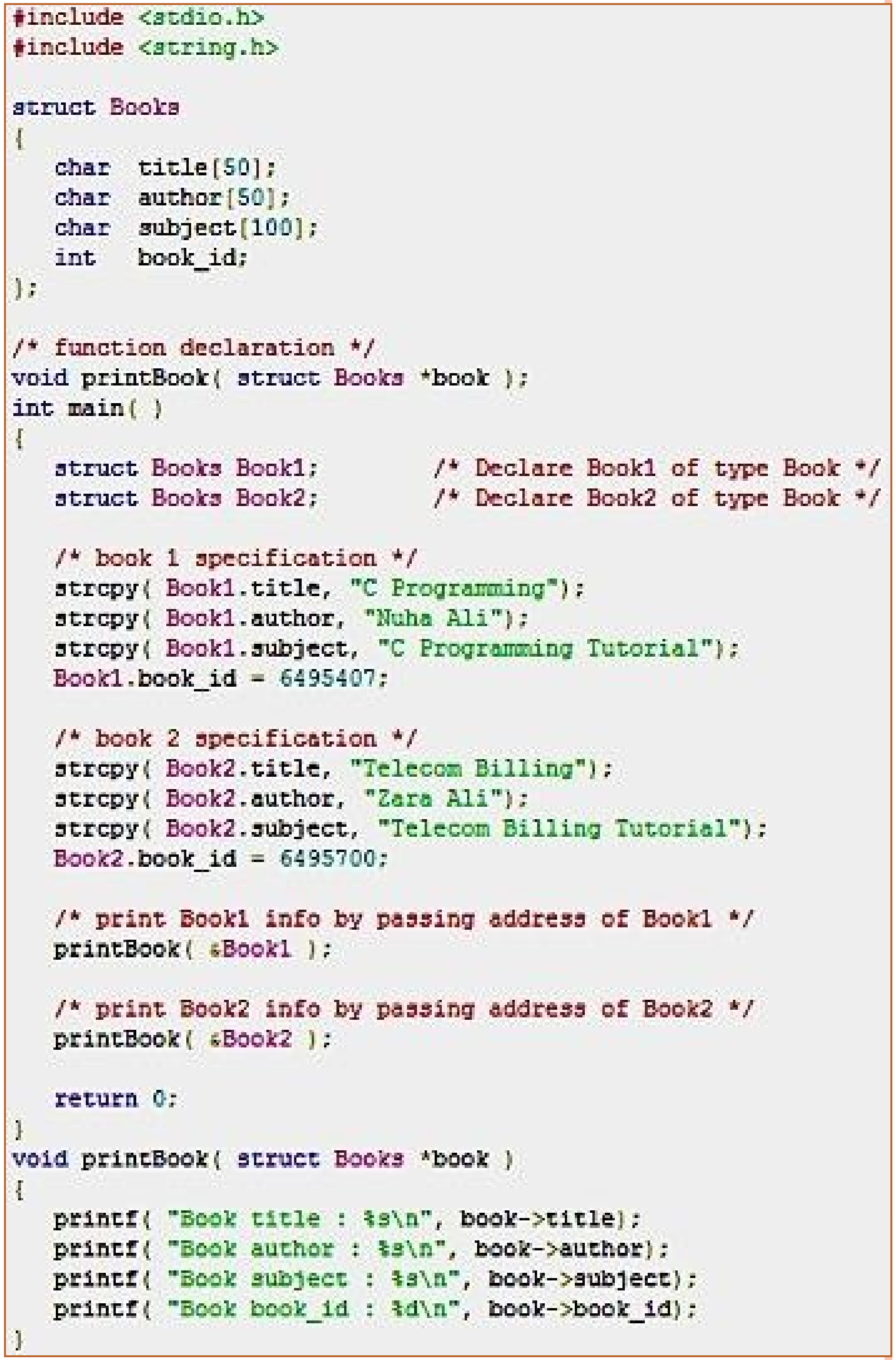


```
#include <stdio.h>
#include <string.h>

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

void func(struct student *record) {
        printf(" Id is: %d \n", record->id);
        printf(" Name is: %s \n", record->name);
        printf(" Percentage is: %f \n", record->percentage);
}

int main() {
        struct student record;

        record.id=1;
        strcpy(record.name, "Raju");
        record.percentage = 86.5;

        func(&record); //Passing the address
        return 0;
}
```

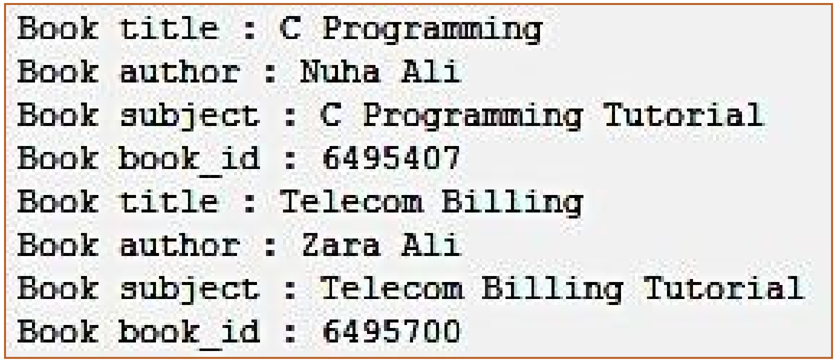
Output:

```
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

EXAMPLE–PASSING STRUCTURE TO FUNCTION BY ADDRESS/REFERENCE



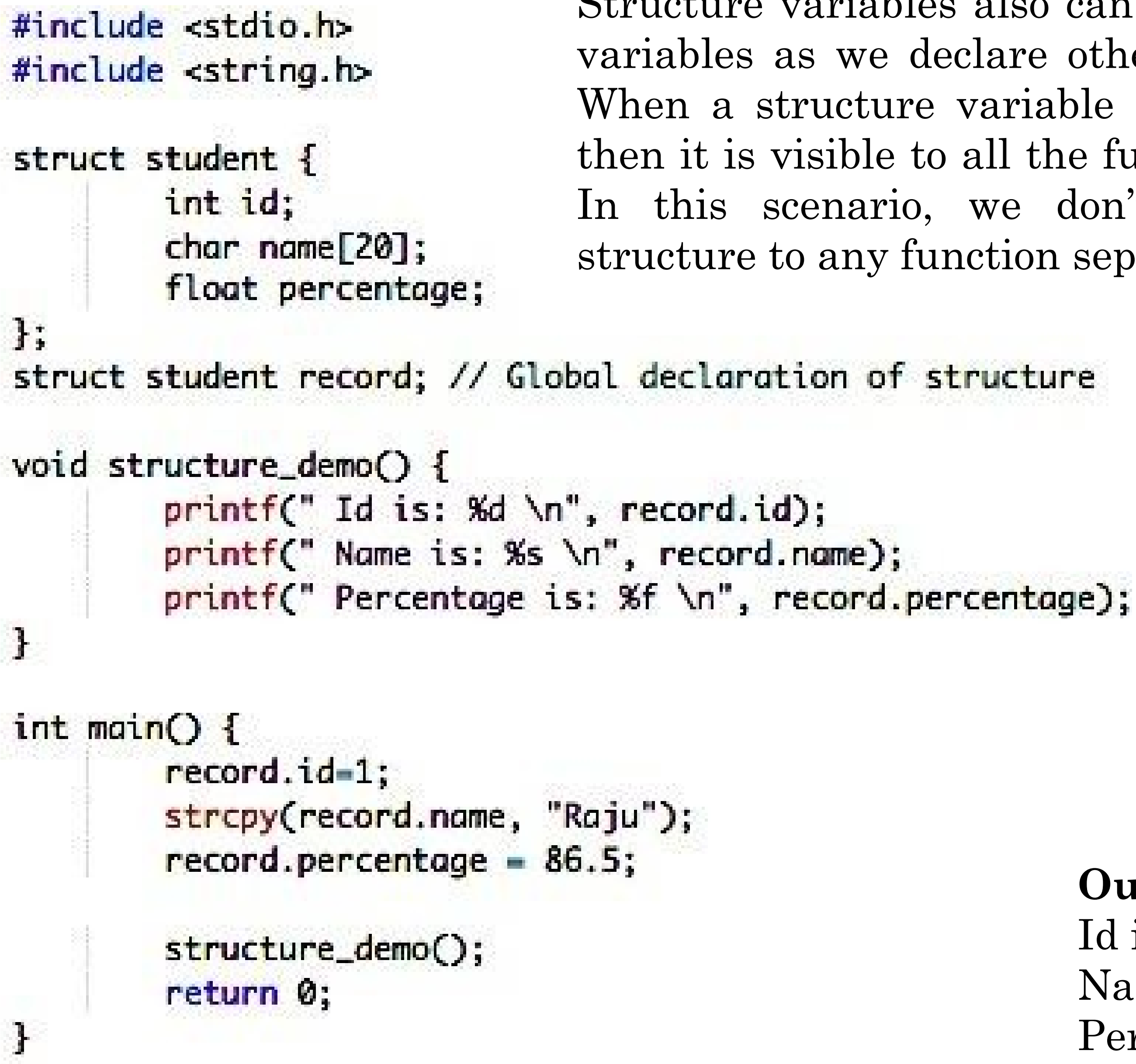
```
#include <stdio.h>
#include <string.h>

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, "C Programming");
   strcpy( Book1.author, "Nuha Ali");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 6495407;

   /* book 2 specification */
   strcpy( Book2.title, "Telecom Billing");
   strcpy( Book2.author, "Zara Ali");
   strcpy( Book2.subject, "Telecom Billing Tutorial");
   Book2.book_id = 6495700;

   /* print Book1 info by passing address of Book1 */
   printBook( &Book1 );

   /* print Book2 info by passing address of Book2 */
   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);
}
```



```
Book title : C Programming
Book author : Nuha Ali
Book subject : C Programming Tutorial
Book book_id : 6495407
Book title : Telecom Billing
Book author : Zara Ali
Book subject : Telecom Billing Tutorial
Book book_id : 6495700
```

EXAMPLE PROGRAM TO DECLARE A STRUCTURE VARIABLE AS GLOBAL

Structure variables also can be declared as global variables as we declare other variables in C. So, When a structure variable is declared as global, then it is visible to all the functions in a program. In this scenario, we don't need to pass the structure to any function separately.



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

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

void structure_demo() {
        printf(" Id is: %d \n", record.id);
        printf(" Name is: %s \n", record.name);
        printf(" Percentage is: %f \n", record.percentage);
}

int main() {
        record.id=1;
        strcpy(record.name, "Raju");
        record.percentage = 86.5;

        structure_demo();
        return 0;
}
```

Output:

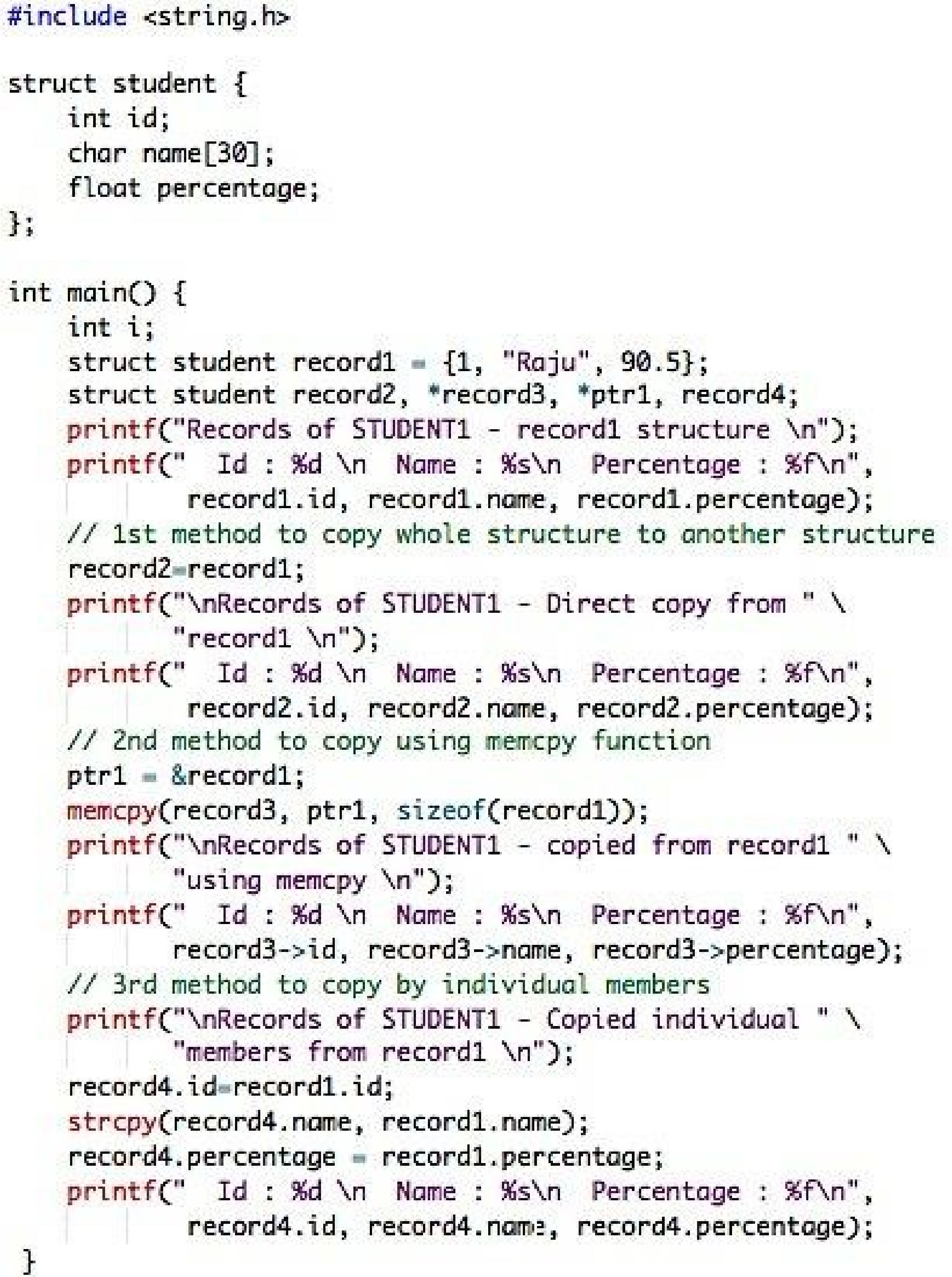
```
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

COPY A STRUCTURE

- There are many methods to copy one structure to another structure in C.
  - We can copy using direct assignment of one structure to another structure or
  - we can use C inbuilt function “memcpy()” or
  - we can copy by individual structure members.

Output:

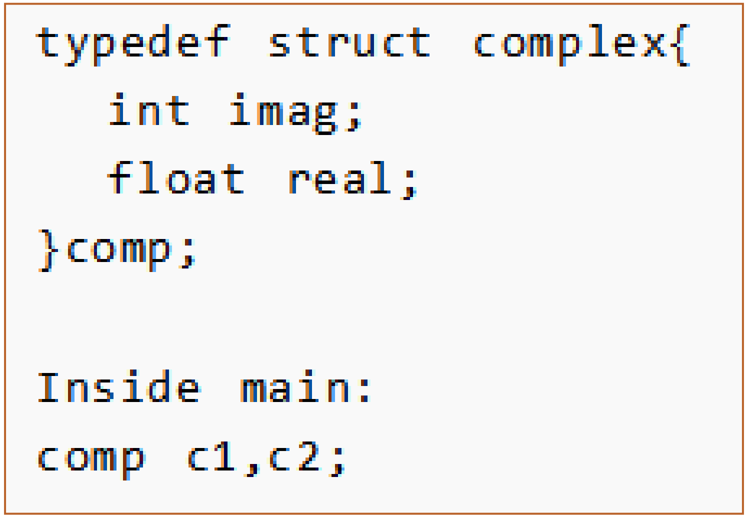
Records of STUDENT1 - **record1 structure**
Id : 1
Name : Raju
Percentage : 90.500000
Records of STUDENT1 – **Direct copy from record1**
Id : 1
Name : Raju
Percentage : 90.500000
Records of STUDENT1 – **copied from record1 using memcpy**
Id : 1
Name : Raju
Percentage : 90.500000
Records of STUDENT1 – **Copied individual members from record1**
Id : 1
Name : Raju
Percentage : 90.500000



```
#include <string.h>

struct student {
      int id;
      char name[30];
      float percentage;
};

int main() {
      int i;
      struct student record1 = {1, "Raju", 90.5};
      struct student record2, *record3, *ptr1, record4;
      printf("Records of STUDENT1 - record1 structure \n");
      printf("  Id : %d \n  Name : %s\n  Percentage : %f\n",
              record1.id, record1.name, record1.percentage);
      // 1st method to copy whole structure to another structure
      record2=record1;
      printf("\nRecords of STUDENT1 - Direct copy from " \
             "record1 \n");
      printf("  Id : %d \n  Name : %s\n  Percentage : %f\n",
              record2.id, record2.name, record2.percentage);
      // 2nd method to copy using memcpy function
      ptr1 = &record1;
      memcpy(record3, ptr1, sizeof(record1));
      printf("\nRecords of STUDENT1 - copied from record1 " \
             "using memcpy \n");
      printf("  Id : %d \n  Name : %s\n  Percentage : %f\n",
             record3->id, record3->name, record3->percentage);
      // 3rd method to copy by individual members
      printf("\nRecords of STUDENT1 - Copied individual " \
             "members from record1 \n");
      record4.id=record1.id;
      strcpy(record4.name, record1.name);
      record4.percentage = record1.percentage;
      printf("  Id : %d \n  Name : %s\n  Percentage : %f\n",
              record4.id, record4.name, record4.percentage);
 }
```

KEYWORD **TYPEDEF** WHILE USING STRUCTURE

- Programmer generally use *typedef* while using structure in C language. For example:

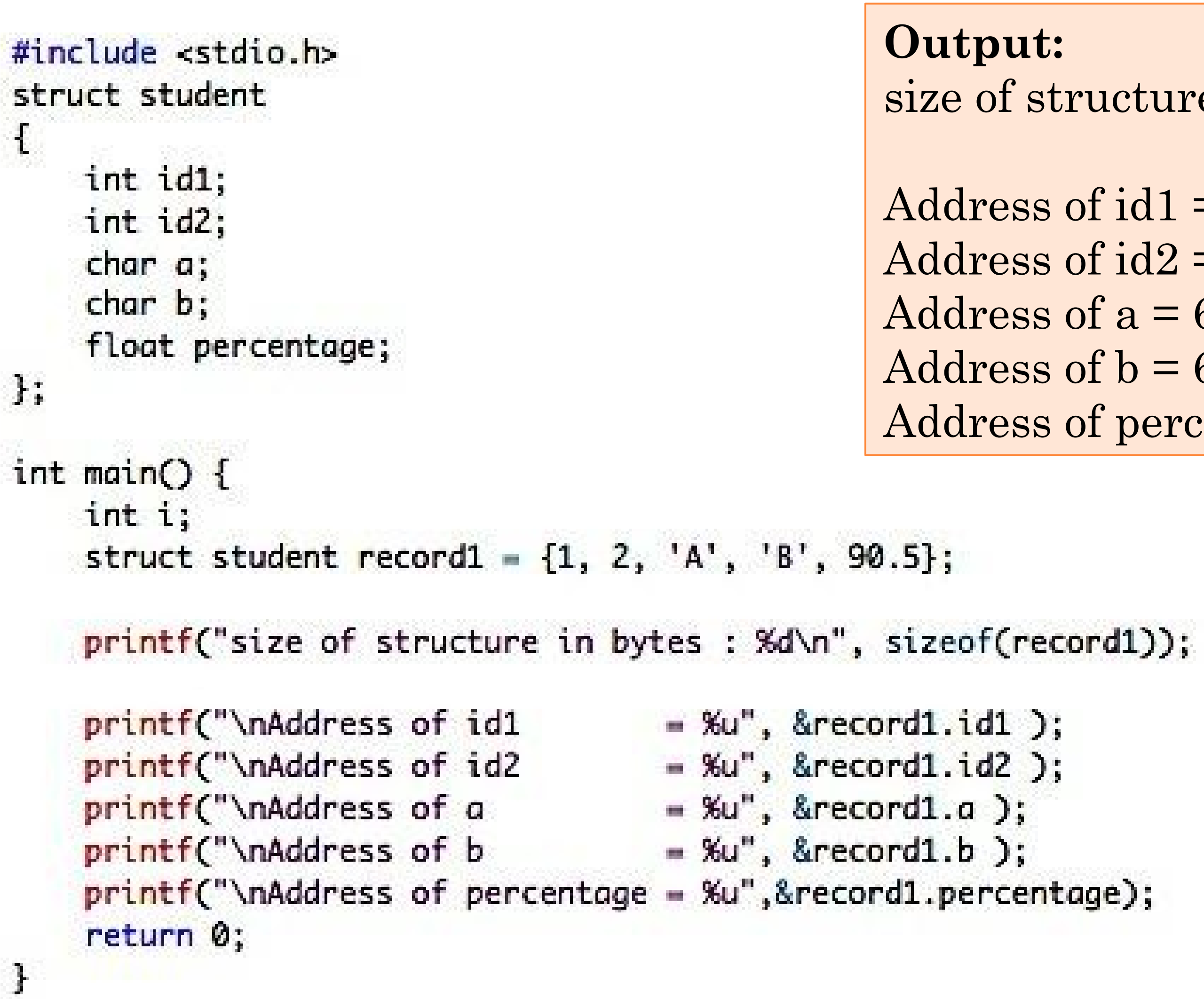


```
typedef struct complex{
   int imag;
   float real;
}comp;

Inside main:
comp c1,c2;
```

- Here, typedef keyword is used in creating a type *comp*(which is of type as **struct complex**). Then, two structure variables *c1* and *c2* are created by this *comp* type.

STRUCT MEMORY ALLOCATION

- How structure members are stored in memory?
  - Always, contiguous(adjacent) memory locations are used to store structure members in memory. Consider below example to understand how memory is allocated for structures.



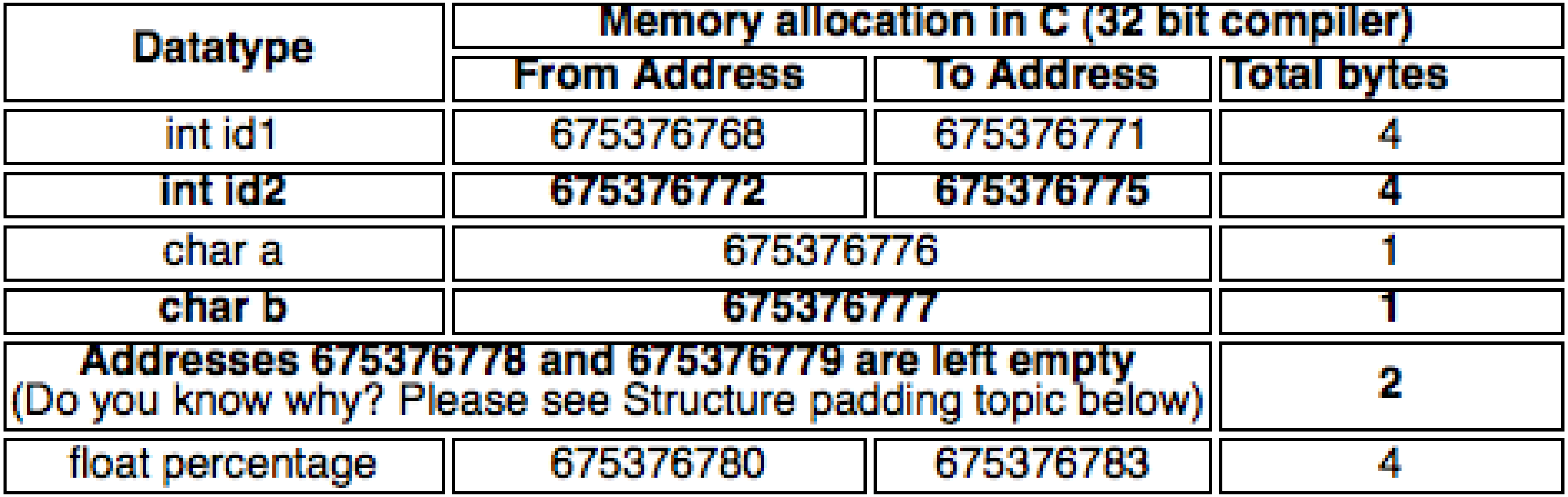
```c
#include <stdio.h>
struct student
{
    int id1;
    int id2;
    char a;
    char b;
    float percentage;
};

int main() {
    int i;
    struct student record1 = {1, 2, 'A', 'B', 90.5};

    printf("size of structure in bytes : %d\n", sizeof(record1));

    printf("\nAddress of id1        = %u", &record1.id1 );
    printf("\nAddress of id2        = %u", &record1.id2 );
    printf("\nAddress of a          = %u", &record1.a );
    printf("\nAddress of b          = %u", &record1.b );
    printf("\nAddress of percentage = %u",&record1.percentage);
    return 0;
}
```

Output:

```
size of structure in bytes : 16

Address of id1 = 675376768
Address of id2 = 675376772
Address of a = 675376776
Address of b = 675376777
Address of percentage = 675376780
```

STRUCT MEMORY ALLOCATION

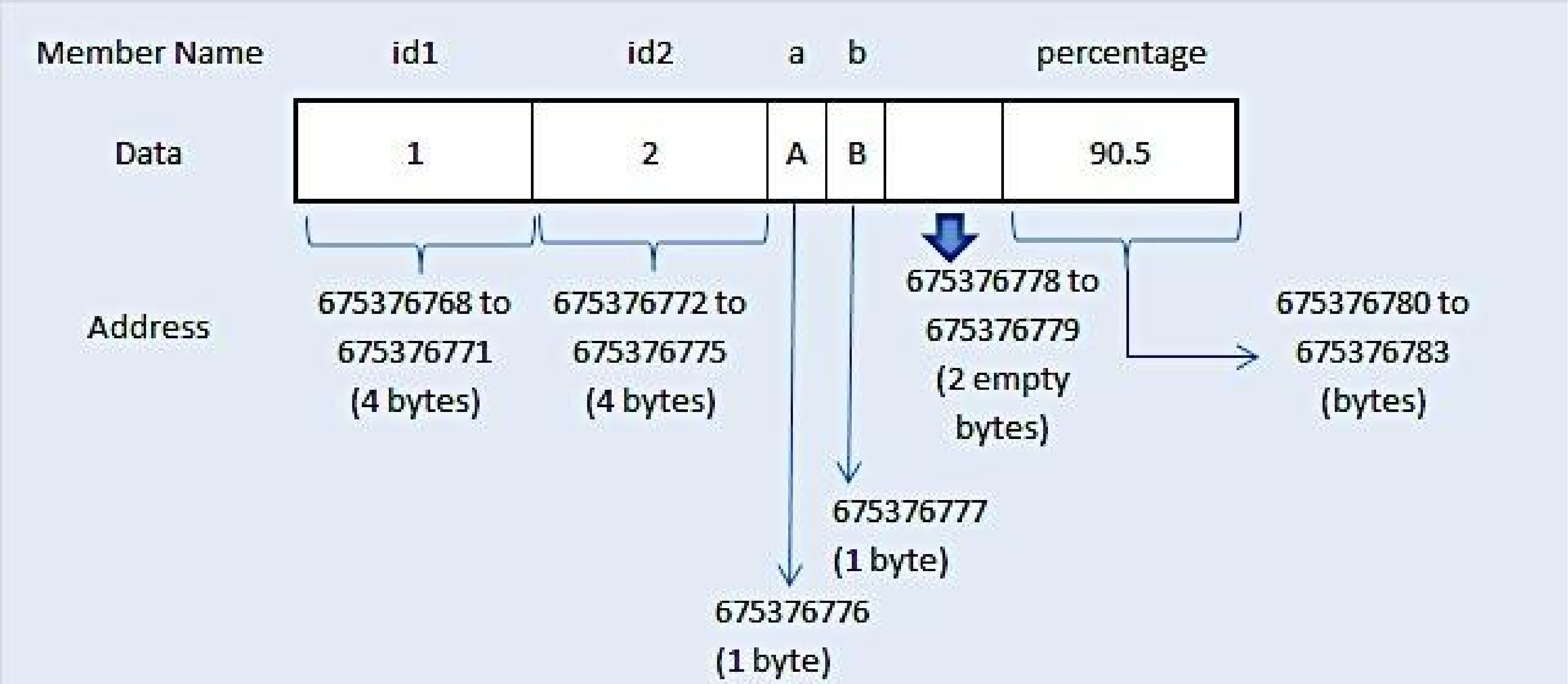
- There are 5 members declared for structure in above program. In 32 bit compiler,
  - 4 bytes of memory is occupied by *int* datatype.
  - 1 byte of memory is occupied by *char* datatype and
  - 4 bytes of memory is occupied by *float* datatype.
- Please refer below table to know from where to where memory is allocated for each datatype in contiguous (adjacent) location in memory.



Datatype	**Memory allocation in C (32 bit compiler)**		
	From Address	**To Address**	**Total bytes**
int id1	675376768	675376771	4
int id2	**675376772**	**675376775**	**4**
char a	675376776		1
char b	**675376777**		**1**
Addresses 675376778 and 675376779 are left empty (Do you know why? Please see Structure padding topic below)			**2**
float percentage	675376780	675376783	4

STRUCT MEMORY ALLOCATION

- The pictorial representation of above structure memory allocation is given below.
- This diagram will help you to understand the memory allocation concept in C very easily.


Member Name
id1
id2
a
b
percentage
Data
1
2
A
B
90.5
Address
675376768 to 675376771 (4 bytes)
675376772 to 675376775 (4 bytes)
675376776 (1 byte)
675376777 (1 byte)
675376778 to 675376779 (2 empty bytes)
675376780 to 675376783 (bytes)

STRUCTURE PADDING

- In order to align the data in memory, one or more empty bytes (addresses) are inserted (or left empty) between memory addresses which are allocated for other structure members while memory allocation. This concept is called *structure padding*.
- Architecture of a computer processor is such a way that it can read 1 word (4 byte in 32 bit processor) from memory at a time.
- To make use of this advantage of processor, data are always aligned as 4 bytes package which leads to insert empty addresses between other member's address.
- Because of this structure padding concept in C, size of the structure is always not same as what we think.

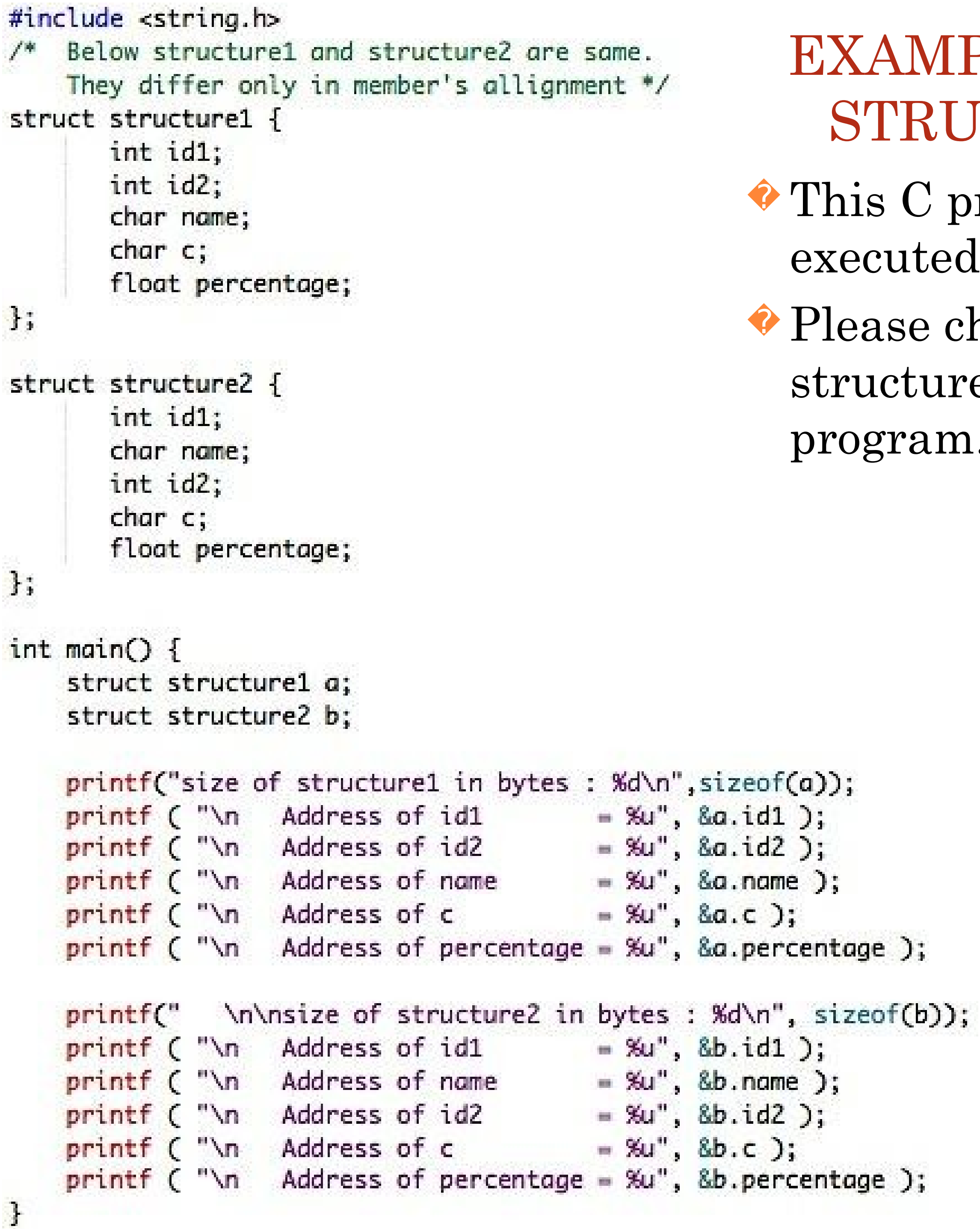
STRUCTURE PADDING

- For example, consider below structure that has 5 members.
- struct student

  ```
  {
      int id1;
      int id2;
      char a;
      char b;
      float percentage;
  };
  ```

- As per C concepts, int and float datatypes occupy 4 bytes each and char datatype occupies 1 byte for 32 bit processor. So, only 14 bytes (4+4+1+1+4) should be allocated for above structure.
- But, this is wrong. Do you know why?
  - Architecture of a computer processor is such a way that it can read 1 word from memory at a time.
  - 1 word is equal to 4 bytes for 32 bit processor and 8 bytes for 64 bit processor.
  - So, 32 bit processor always reads 4 bytes at a time and 64 bit processor always reads 8 bytes at a time.
  - This concept is very useful to increase the processor speed.
  - To make use of this advantage, memory is arranged as a group of 4 bytes in 32 bit processor and 8 bytes in 64 bit processor.

EXAMPLE PROGRAM FOR STRUCTURE PADDING



```c
#include <string.h>
/*  Below structure1 and structure2 are same.
    They differ only in member's allignment */
struct structure1 {
        int id1;
        int id2;
        char name;
        char c;
        float percentage;
};

struct structure2 {
        int id1;
        char name;
        int id2;
        char c;
        float percentage;
};

int main() {
    struct structure1 a;
    struct structure2 b;

    printf("size of structure1 in bytes : %d\n",sizeof(a));
    printf ( "\n   Address of id1        = %u", &a.id1 );
    printf ( "\n   Address of id2        = %u", &a.id2 );
    printf ( "\n   Address of name       = %u", &a.name );
    printf ( "\n   Address of c          = %u", &a.c );
    printf ( "\n   Address of percentage = %u", &a.percentage );

    printf("   \n\nsize of structure2 in bytes : %d\n", sizeof(b));
    printf ( "\n   Address of id1        = %u", &b.id1 );
    printf ( "\n   Address of name       = %u", &b.name );
    printf ( "\n   Address of id2        = %u", &b.id2 );
    printf ( "\n   Address of c          = %u", &b.c );
    printf ( "\n   Address of percentage = %u", &b.percentage );
}
```

- This C program is compiled and executed in 32 bit compiler.
- Please check memory allocated for structure1 and structure2 of this program.

Output:

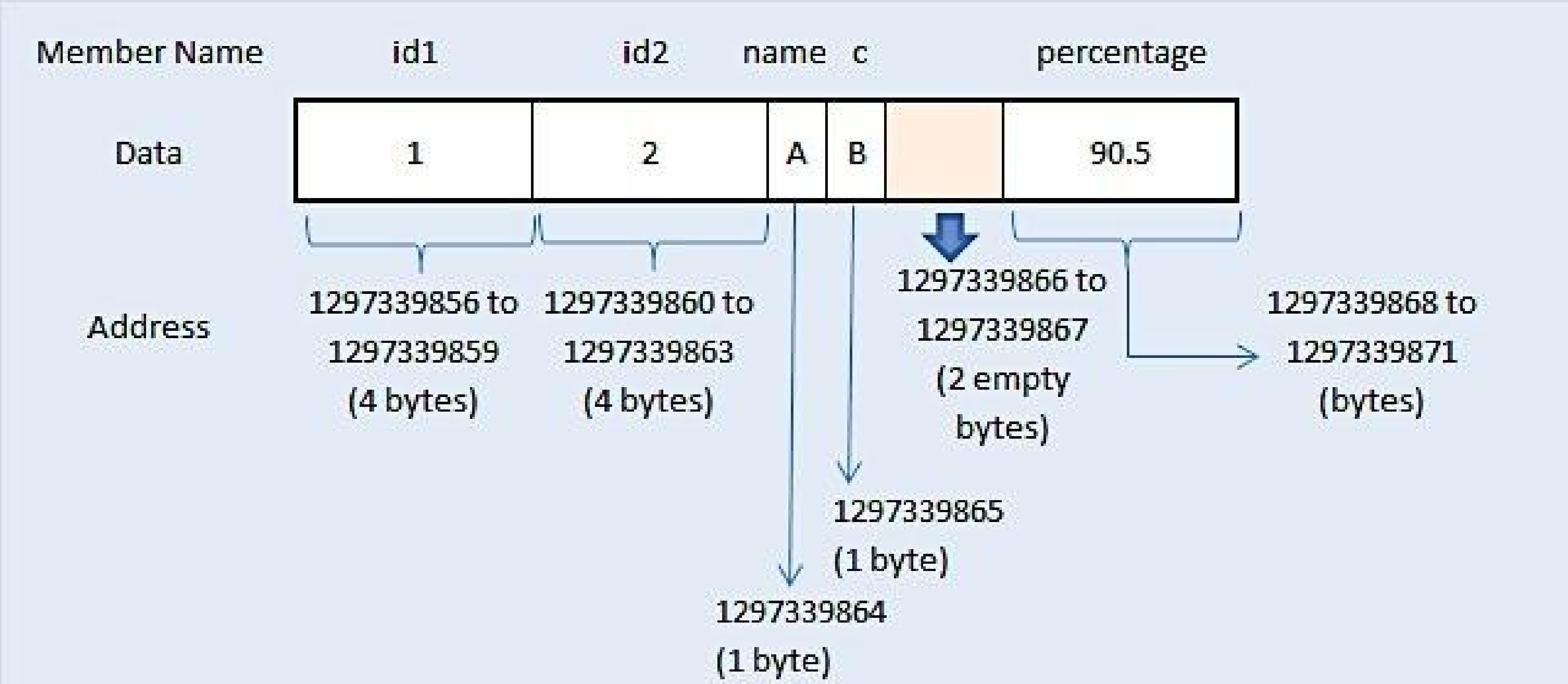
```
size of structure1 in bytes : 16

Address of id1 = 1297339856
Address of id2 = 1297339860
Address of name = 1297339864
Address of c = 1297339865
Address of percentage = 1297339868

size of structure2 in bytes : 20

Address of id1 = 1297339824
Address of name = 1297339828
Address of id2 = 1297339832
Address of c = 1297339836
Address of percentage = 1297339840
```

STRUCTURE PADDING ANALYSIS FOR PREVIOUS C PROGRAM

- Memory allocation for **structure1:**
  - In above program, memory for structure1 is allocated sequentially for first 4 members.
  - Whereas, memory for 5th member “percentage” is not allocated immediate next to the end of member “c”
  - There are only 2 bytes remaining in the package of 4 bytes after memory allocated to member “c”.
  - Range of this 4 byte package is from 1297339864 to 1297339867.
  - Addresses 1297339864 and 1297339865 are used for members “name and c”. Addresses 1297339866 and 1297339867 only is available in this package.
  - But, member “percentage” is datatype of float and requires 4 bytes. It can’t be stored in the same memory package as it requires 4 bytes. Only 2 bytes are free in that package.
  - So, next 4 byte of memory package is chosen to store percentage data which is from 1297339868 to 1297339871.
  - Because of this, memory 1297339866 and 1297339867 are not used by the program and those 2 bytes are left empty.
  - So, size of structure1 is 16 bytes which is 2 bytes extra than what we think. Because, 2 bytes are left empty.

STRUCTURE PADDING ANALYSIS FOR PREVIOUS C PROGRAM

Memory allocation for **structure1**



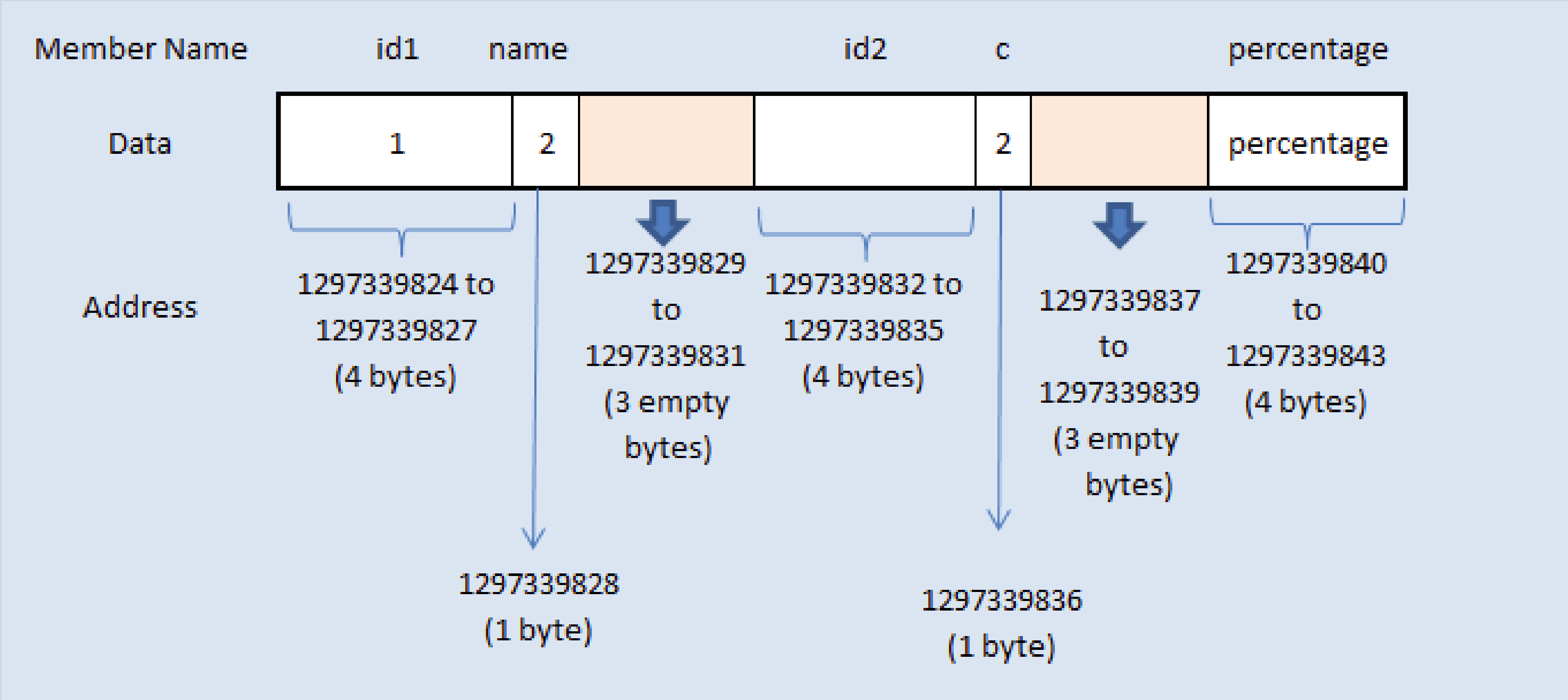
Member Name	id1	id2	name	c		percentage
Data	1	2	A	B		90.5
Address	1297339856 to 1297339859 (4 bytes)	1297339860 to 1297339863 (4 bytes)	1297339864 (1 byte)	1297339865 (1 byte)	1297339866 to 1297339867 (2 empty bytes)	1297339868 to 1297339871 (bytes)

STRUCTURE PADDING ANALYSIS FOR PREVIOUS C PROGRAM

- Memory allocation for **structure2:**
  - Memory for structure2 is also allocated as same as above concept. Please note that structure1 and structure2 are same. But, they differ only in the order of the members declared inside the structure.
  - 4 bytes of memory is allocated for 1st structure member “id1″ which occupies whole 4 byte of memory package.
  - Then, 2nd structure member “name” occupies only 1 byte of memory in next 4 byte package and remaining 3 bytes are left empty. Because,
    3rd structure member “id2″ of datatype integer requires whole 4 byte of memory in the package. But, this is not possible as only 3 bytes available in the package.
  - So, next whole 4 byte package is used for structure member “id2″.
  - Again, 4th structure member “c” occupies only 1 byte of memory in next 4 byte package and remaining 3 bytes are left empty.
  - Because, 5th structure member “percentage” of datatype float requires whole 4 byte of memory in the package.
  - But, this is also not possible as only 3 bytes available in the package. So, next whole 4 byte package is used for structure member “percentage”.
  - So, size of structure2 is 20 bytes which is 6 bytes extra than what we think. Because, 6 bytes are left empty.

STRUCTURE PADDING ANALYSIS FOR PREVIOUS C PROGRAM

Memory allocation for **structure2**



Member Name	id1	name		id2	c		percentage
Data	1	2			2		percentage
Address	1297339824 to 1297339827 (4 bytes)	1297339828 (1 byte)	1297339829 to 1297339831 (3 empty bytes)	1297339832 to 1297339835 (4 bytes)	1297339836 (1 byte)	1297339837 to 1297339839 (3 empty bytes)	1297339840 to 1297339843 (4 bytes)