CS101: Problem Solving through C Programming

Sachchida Nand Chaurasia
Assistant Professor

Department of Computer Science
Banaras Hindu University
Varanasi
Email id: snchaurasia@bhu.ac.in, sachchidanand.mca07@gmail.com

February 26, 2021

Structure Basics in C I

✓ Structures in C are used to create new data types. So why would we want need to create new data types?

✓ Suppose we are creating a program for storing records of the students.

✓ A student has many attributes like name, roll no, marks, attendance etc. Some items are strings and some are numbers.

✓ Here is the one way to approach this problem.

```
#include<stdio.h>
#include<string.h>
int main()
{
     char name[20];
     int roll_no, i;
     float marks[5];
     printf("Enter name: ");
     scanf("%s", name);
     printf("Enter roll no: ");
     scanf("%d", &roll_no);
     printf("\n");
     for(i = 0; i < 5; i++)
     {
```

Structure Basics in C II

```
15          printf("Enter marks for %d: subject: ", i+1);
16          scanf("%f", &marks[i]);
17      }
18      printf("\nYou entered: \n\n");
19      printf("Name: %s\n", name);
20      printf("roll no: %d\n", roll_no);
21      printf("\n");
22      for(i = 0; i < 5; i++)
23      {
24          printf("Marks in %d subject %f: l\n", i+1, marks[i]);
25      }
26      return 0;
27  }
```

Defining Structure:

```
struct tagname
{
     data_type member1;
     data_type member2;
     ...
     ...
     data_type memberN;
}
;
```

Let's define a simple structure called the student.

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

Creating Structure Variables:

We can't use structure definition in any way unless we declare structure variables.

There are two ways to declare structure variables:

1. With the structure definition
2. Using tagname

With the structure definition:

```
1 struct student
2 {
3       char name[20];
4       int roll_no;
5       float marks;
6 }
7  student1, student2;
```

Defining structure this way has several limitations:

1. As this structure has no name associated with it, we can't create structure variables of this structure type anywhere else in the program. If it is necessary for you to declare variables of this structure type then you have to write the same template again. We can't send these structure variables to other functions.
2. Due to the limitations mentioned this method is not widely used.

Using tagname:

To declare structure variable using tagname use the following syntax:

```
Syntax: struct tagname variable_name;

//where variable_name must be a valid identifier.

struct student student1;
struct student student1, student2, student3;
```

Structure Basics in C VI

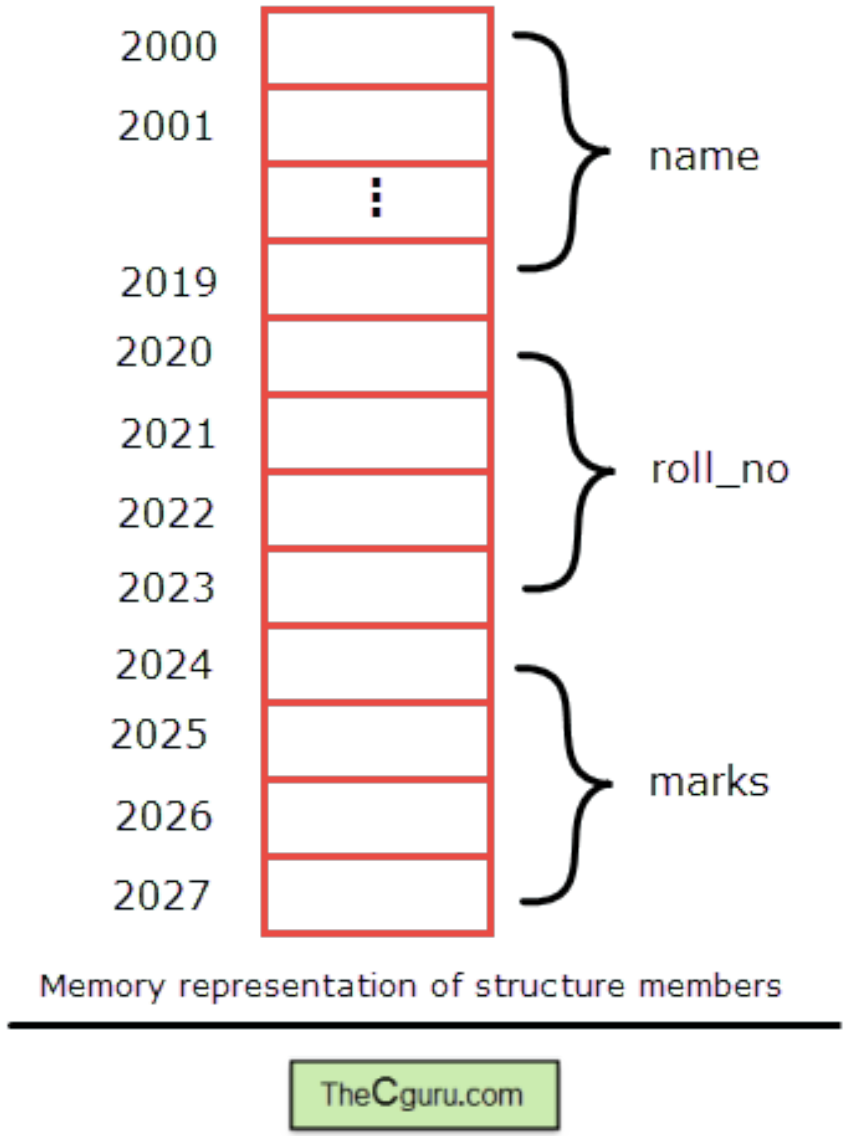
✓ When a variable is declared only then the compiler reserve space in memory. It is important to understand that the members of a structure are stored in the memory in the order in which they are defined.

✓ In this case, each structure variable of type student has 3 members namely: name, roll_no, marks.

✓ As a result, the compiler will allocate memory sufficient to hold all the members of the structure.

✓ So here each structure variable occupies 28 bytes (20+4+4) of memory.

Structure Basics in C VII



Memory representation of structure members

TheCguru.com

Initializing Structure Variables:

✓ To initialize the structure variables we use the same syntax as we used for initializing arrays.

```
struct student
{
     char name[20];
     int roll_no;
     float marks;
}
 student1 =
{
     "Jim", 14, 89
}
;

struct student student2 =
{
     "Tim", 10, 82
}
;
```

The value of members must be placed in the same order and of the same type as defined in the structure template.

✓ Another important thing to understand is that we are not allowed to initialize members at the time of defining structure.

```
1 struct student
2 {
3       char name[20] = "Phil";    // invalid
4       int roll_no = 10; // invalid
5       float marks = 3.14;    // invalid
6 }
7 ;
```

✓ Defining a structure only creates a template, no memory is allocated until structure variables are created. Hence at this point there no variables called name, roll_no and marks, so how we can store data in a variable which do not exist? We can't.

✓ f the number of initializers is less than the number of members then the remaining members are given a value of 0.

Operation on Structures:

✓ After creating structure definition and structure variables. Obviously, the next logical step is to learn how to access members of a structure.

✓ The dot (.) operator or membership operator is used to access members of a structure using a structure variable. Here is the syntax:

Syntax: structure_variable.member_name;

✓ We can refer to a member of a structure by writing structure variable followed by a dot (.) operator, followed by the member name.

```
printf("Name: %s", student_1.name);
printf("Name: %d", student_2.roll_no);
printf("Name: %f", student_1.marks);
```

Let's try changing the values of structure members.

```
student_1.roll_no = 10; // change roll no of student_1 from 44 to 10
student_1.marks++;  // increment marks of student_1 by 1
```

Structure Basics in C XI

✓ Recall from the chapter operator precedence and associativity that the precedence of dot(.) operator is greater than that of ++ operator and assignment operator (=). So in the above expression first dot (.) operator is applied in the expression followed by ++ operator.

✓ We can also assign a structure variable to another structure variable of the same type.

```
1  struct student
2  {
3       char name[20];
4       int roll_no;
5       float marks;
6  }
7  ;
8
9  struct student student1 =
10 {
11      "Jon", 44, 96
12 }
13 , student2;
14
15 student2 = student1;
```

It is important to note that we can't use arithmetic, relational and bitwise operators with structure variables.

```
student1 + student2;  // invalid
student1 == student2; // invalid
student1 & student2;  // invalid
```

✓ The following program demonstrates how we can define a structure and read values of structure members.

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

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

```

Structure Basics in C XIII

```
12 int main()
13 {
14      struct student student_1 =
15      {
16           "Jim", 10, 34.5
17      }
18      , student_2, student_3;
19
20      printf("Details of student 1\n\n");
21
22      printf("Name: %s\n", student_1.name);
23      printf("Roll no: %d\n", student_1.roll_no);
24      printf("Marks: %.2f\n", student_1.marks);
25
26      printf("\n");
27
28      printf("Enter name of student2: ");
29      scanf("%s", student_2.name);
30
31      printf("Enter roll no of student2: ");
32      scanf("%d", &student_2.roll_no);
```

Structure Basics in C XIV

```
33
34      printf("Enter marks of student2: ");
35      scanf("%f", &student_2.marks);
36
37      printf("\nDetails of student 2\n\n");
38
39      printf("Name: %s\n", student_2.name);
40      printf("Roll no: %d\n", student_2.roll_no);
41      printf("Marks: %.2f\n", student_2.marks);
42      strcpy(student_3.name, "King");
43      student_3.roll_no = ++student_2.roll_no;
44      student_3.marks = student_2.marks + 10;
45
46      printf("\nDetails of student 3\n\n");
47
48      printf("Name: %s\n", student_3.name);
49      printf("Roll no: %d\n", student_3.roll_no);
50      printf("Marks: %.2f\n", student_3.marks);
51
52      return 0;
53  }
```

How Structures are stored in Memory:

✓ Members of a structure are always stored in consecutive memory locations but the memory occupied by each member may vary.

```
#include<stdio.h>

struct book
{
     char title[5];
     int year;
     double price;
}
;

int main()
{
     struct book b1 =
     {
          "Book1", 1988, 4.51
     }
     ;
```

Structure Basics in C XVI

```
18
19      printf("Address of title = %u\n", b1.title);
20      printf("Address of year = %u\n", &b1.year);
21      printf("Address of price = %u\n", &b1.price);
22
23      printf("Size of b1 = %d\n", sizeof(b1));
24
25      // signal to operating system program ran fine
26      return 0;
27  }
```

Structure Basics in C XVII

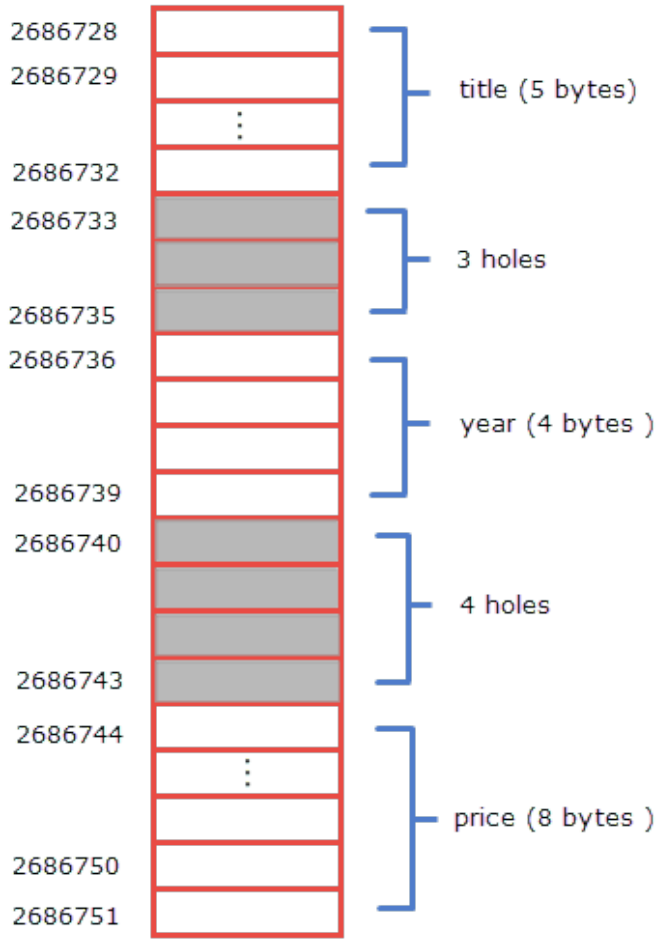
```
struct book
{
    char title[5];
    int year;
    double price;
};

struct book b1 =
{"Book1", 1988, 4.51};
```

Size of structure b1 =

name(5 bytes) + 3 holes + year(4 bytes) + 4 holes + price(8 bytes)

= 24 bytes



A structure variable in memory

TheCguru.com

Array of Structures in C I

✓ Declaring an array of structure is same as declaring an array of fundamental types.

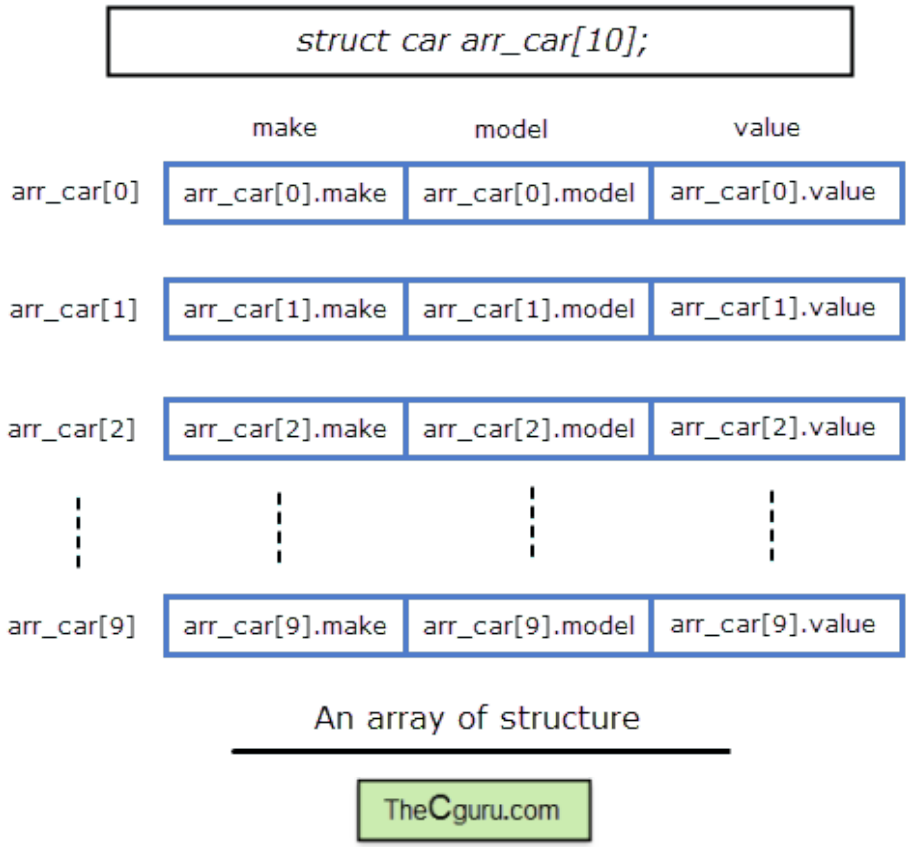
✓ Since an array is a collection of elements of the same type. In an array of structures, each element of an array is of the structure type.

```
struct car
{
     char make[20];
     char model[30];
     int year;
}
;
```

Here is how we can declare an array of structure car.

```
struct car arr_car[10];
```

Array of Structures in C II



Array of Structures in C III

```c
#include<stdio.h>
#include<string.h>
#define MAX 2

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

int main()
{
     struct student arr_student[MAX];
     int i;

     for(i = 0; i < MAX; i++ )
     {
          printf("\nEnter details of student %d\n\n", i+1);

```

Array of Structures in C IV

```
22          printf("Enter name: ");
23          scanf("%s", arr_student[i].name);
24
25          printf("Enter roll no: ");
26          scanf("%d", &arr_student[i].roll_no);
27
28          printf("Enter marks: ");
29          scanf("%f", &arr_student[i].marks);
30      }
31
32      printf("\n");
33
34      printf("Name\tRoll no\tMarks\n");
35
36      for(i = 0; i < MAX; i++ )
37      {
38          printf("%s\t%d\t%.2f\n",
39          arr_student[i].name, arr_student[i].roll_no, arr_student[i].marks);
40      }
41      return 0;
42  }
```

Initializing Array of Structures:

✓ We can also initialize the array of structures using the same syntax as that for initializing arrays. Let's take an example:

```
struct car
{
     char make[20];
     char model[30];
     int year;
}
;
struct car arr_car[2] =
{
     {
          "Audi", "TT", 2016
     }
     ,
     {
          "Bentley", "Azure", 2002
     }
}
```

Array of Structures in C VI

```
18 ;
```

Array as Member of Structure in C I

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

Let's create another structure called student to store name, roll no and marks of 5 subjects.

```
struct student
{
     char name[20];
     int roll_no;
     float marks[5];
}
;
```

Array as Member of Structure in C II

If student_1 is a variable of type struct student then:
student_1.marks[0] - refers to the marks in the first subject
student_1.marks[1] - refers to the marks in the second subject
and so on. Similarly, if arr_student[10] is an array of type struct student then:
arr_student[0].marks[0] - refers to the marks of first student in the first subject
arr_student[1].marks[2] - refers to the marks of the second student in the third subject
and so on.

```
#include<stdio.h>
#include<string.h>
#define MAX 2
#define SUBJECTS 2

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

Array as Member of Structure in C III

```
13
14 int main()
15 {
16      struct student arr_student[MAX];
17      int i, j;
18      float sum = 0;
19
20      for(i = 0; i < MAX; i++ )
21      {
22           printf("\nEnter details of student %d\n\n", i+1);
23
24           printf("Enter name: ");
25           scanf("%s", arr_student[i].name);
26
27           printf("Enter roll no: ");
28           scanf("%d", &arr_student[i].roll_no);
29
30           for(j = 0; j < SUBJECTS; j++)
31           {
32                printf("Enter marks: ");
33                scanf("%f", &arr_student[i].marks[j]);
```

Array as Member of Structure in C IV

```
34          }
35      }
36      printf("\n");
37      printf("Name\tRoll no\tAverage\n\n");
38
39      for(i = 0; i < MAX; i++ )
40      {
41          sum = 0;
42
43          for(j = 0; j < SUBJECTS; j++)
44          {
45              sum += arr_student[i].marks[j];
46          }
47          printf("%s\t%d\t%.2f\n",
48           arr_student[i].name, arr_student[i].roll_no, sum/SUBJECTS);
49      }
50
51      return 0;
52  }
```

Nested Structures in C I

✓ A structure can be nested inside another structure. In other words, the members of a structure can be of any other type including structure.

✓ Here is the syntax to create nested structures.

```
structure tagname_1
{
     member1;
     member2;
     ...
     membern;

     structure tagname_2
     {
          member_1;
          member_2;
          ...
          member_n;
     }
     , var1
}
 var2;
```

Nested Structures in C II

Let's take an example:

```
struct student
{
     struct person
     {
          char name[20];
          int age;
          char dob[10];
     }
      p ;

     int rollno;
     float marks;
}
 stu;
```

Nested Structures in C III

Here we have defined structure person as a member of structure student.
Here is how we can access the members of person structure.
stu.p.name - refers to the name of the person
stu.p.age - refers to the age of the person
stu.p.dob - refers to the date of birth of the person

✓ It is important to note that structure person doesn't exist on its own. We can't declare structure variable of type struct person anywhere else in the program.

✓ Instead of defining the structure inside another structure. We could have defined it outside and then declare it's variable inside the structure where we want to use it. For example:

```
1 struct person
2 {
3       char name[20];
4       int age;
5       char dob[10];
6 }
7 ;
```

Nested Structures in C IV

```
1 struct student
2 {
3       struct person info;
4       int rollno;
5       float marks;
6 }
```

✓ Here the first member is of type struct person. If we use this method of creating nested structures then you must first define the structures before creating variables of its types.

✓ So, it's mandatory for you to first define person structure before using it's variable as a member of the structure student.

Nesting of structure within itself is now allowed.

```c
struct citizen
{
     char name[50];
     char address[100];
     int age;
     int ssn;
     struct citizen relative; // invalid
}
```

Initializing nested Structures:

```c
struct person
{
     char name[20];
     int age;
     char dob[10];
}
;

struct student
{
```

Nested Structures in C VI

```
11      struct person info;
12      int rollno;
13      float marks[10];
14  }
15
16  struct student student_1 =
17  {
18
19      {
20          "Adam", 25, 1990
21      }
22      ,
23         101,
24         90
25
26  }
27  ;
```

Nested Structures in C VII

```c
#include<stdio.h>

struct person
{
     char name[20];
     int age;
     char dob[10];
}
;

struct student
{
     struct person info;
     int roll_no;
     float marks;
}
;

int main()
{
     struct student s1;
```

Nested Structures in C VIII

```
22
23      printf("Details of student: \n\n");
24
25      printf("Enter name: ");
26      scanf("%s", s1.info.name);
27
28      printf("Enter age: ");
29      scanf("%d", &s1.info.age);
30
31      printf("Enter dob: ");
32      scanf("%s", s1.info.dob);
33
34      printf("Enter roll no: ");
35      scanf("%d", &s1.roll_no);
36
37      printf("Enter marks: ");
38      scanf("%f", &s1.marks);
39
40      printf("\n*******************************\n\n");
41
42      printf("Name: %s\n", s1.info.name);
```

Nested Structures in C IX

```
43      printf("Age: %d\n", s1.info.age);
44      printf("DOB: %s\n", s1.info.dob);
45      printf("Roll no: %d\n", s1.roll_no);
46      printf("Marks: %.2f\n", s1.marks);
47
48      // signal to operating system program ran fine
49      return 0;
50  }
```

Pointer to a Structure in C I

```
struct dog
{
     char name[10];
     char breed[10];
     int age;
     char color[10];
}
;

struct dog spike;

// declaring a pointer to a structure of type struct dog
struct dog *ptr_dog
```

✓ This declares a pointer ptr_dog that can store the address of the variable of type struct dog. We can now assign the address of variable spike to ptr_dog using & operator.

```
ptr_dog = &spike;
```

Now ptr_dog points to the structure variable spike.

Pointer to a Structure in C II

Accessing members using Pointer:

There are two ways of accessing members of structure using pointer:

1. Using indirection (*) operator and dot (.) operator.
2. Using arrow (->) operator or membership operator.

Using Indirection (*) Operator and Dot (.) Operator:

✓ At this point ptr_dog points to the structure variable spike, so by dereferencing it we will get the contents of the spike.

✓ This means spike and *ptr_dog are functionally equivalent.

✓ To access a member of structure write *ptr_dog followed by a dot(.) operator, followed by the name of the member.

For example:

(*ptr_dog).name - refers to the name of dog

(*ptr_dog).breed - refers to the breed of dog

and so on.

✓ Parentheses around *ptr_dog are necessary because the precedence of dot(.) operator is greater than that of indirection (*) operator.

Pointer to a Structure in C III

Using arrow operator (->)

To access members using arrow (->) operator write pointer variable followed by -> operator, followed by name of the member.

```
ptr_dog->name   // refers to the name of dog
ptr_dog->breed  // refers to the breed of dog
```

We can also modify the value of members using pointer notation.

```
strcpy(ptr_dog->name, "new_name");
```

✓ Here we know that the name of the array (ptr_dog->name) is a constant pointer and points to the 0th element of the array. So we can't assign a new string to it using assignment operator (=), that's why strcpy() function is used.

```
--ptr_dog->age;
```

In the above expression precedence of arrow operator (->) is greater than that of prefix decrement operator (–), so first -> operator is applied in the expression then its value is decremented by 1.

Pointer to a Structure in C IV

```c
#include<stdio.h>

struct dog
{
     char name[10];
     char breed[10];
     int age;
     char color[10];
}
;

int main()
{
     struct dog my_dog =
     {
          "tyke", "Bulldog", 5, "white"
     }
     ;
     struct dog *ptr_dog;
     ptr_dog = &my_dog;

```

Pointer to a Structure in C V

```
22      printf("Dog's name: %s\n", ptr_dog->name);
23      printf("Dog's breed: %s\n", ptr_dog->breed);
24      printf("Dog's age: %d\n", ptr_dog->age);
25      printf("Dog's color: %s\n", ptr_dog->color);
26
27      // changing the name of dog from tyke to jack
28      strcpy(ptr_dog->name, "jack");
29
30      // increasing age of dog by 1 year
31      ptr_dog->age++;
32
33      printf("Dog's new name is: %s\n", ptr_dog->name);
34      printf("Dog's age is: %d\n", ptr_dog->age);
35
36      // signal to operating system program ran fine
37      return 0;
38  }
```

We can also have a pointer as a member of the structure. For example:

```
struct test
{
     char name[20];
     int *ptr_mem;
}
;

struct test t1, *str_ptr = &t1;
```

Here ptr_mem is a pointer to int and a member of structure test.
There are two ways in which we can access the value (i.e address) of ptr_mem:

1. Using structure variable - t1.ptr_mem
2. Using pointer variable - str_ptr->ptr_mem

Similarly, there are two ways in which we can access the value pointed to by ptr_mem.

1. Using structure variable - *t1.ptr_mem
2. Using pointer variable - *str_ptr->ptr_mem

✓ Since the precedence of dot(.) operator is greater than that of indirection(*) operator, so in the expression *t1.ptr_mem the dot(.) is applied before the indirection(*) operator. Similarly in the expression *str_ptr->ptr_mem, the arrow (->) operator is applied followed by indirection(*) operator.

```
#include<stdio.h>

struct student
{
      char *name;
      int age;
      char *program;
      char *subjects[5];
}
;

int main()
{
      struct student stu =
      {
            "Lucy",
            25,
            "CS",
```

```
19
20          {
21               "CS-01", "CS-02", "CS-03", "CS-04", "CS-05"
22          }
23
24     }
25     ;
26
27     struct student *ptr_stu = &stu;
28     int i;
29
30     printf("Accessing members using structure variable: \n\n");
31
32     printf("Name: %s\n", stu.name);
33     printf("Age: %d\n", stu.age);
34     printf("Program enrolled: %s\n", stu.program);
35
36     for(i = 0; i < 5; i++)
37     {
38          printf("Subject : %s \n", stu.subjects[i]);
39     }
40
41     printf("\n\nAccessing members using pointer variable: \n\n");
```

```
42
43      printf("Name: %s\n", ptr_stu->name);
44      printf("Age: %d\n", ptr_stu->age);
45      printf("Program enrolled: %s\n", ptr_stu->program);
46
47      for(i = 0; i < 5; i++)
48      {
49           printf("Subject : %s \n", ptr_stu->subjects[i]);
50      }
51
52      return 0;
53  }
```

Name	Type
name	a pointer to char
age	int
program	a pointer to char
subjects	an array of 5 pointers to char

Pointers as Structure Member in C I

✓ We can also have a pointer as a member of the structure. For example:

```
struct test
{
     char name[20];
     int *ptr_mem;
}
;

struct test t1, *str_ptr = &t1;
```

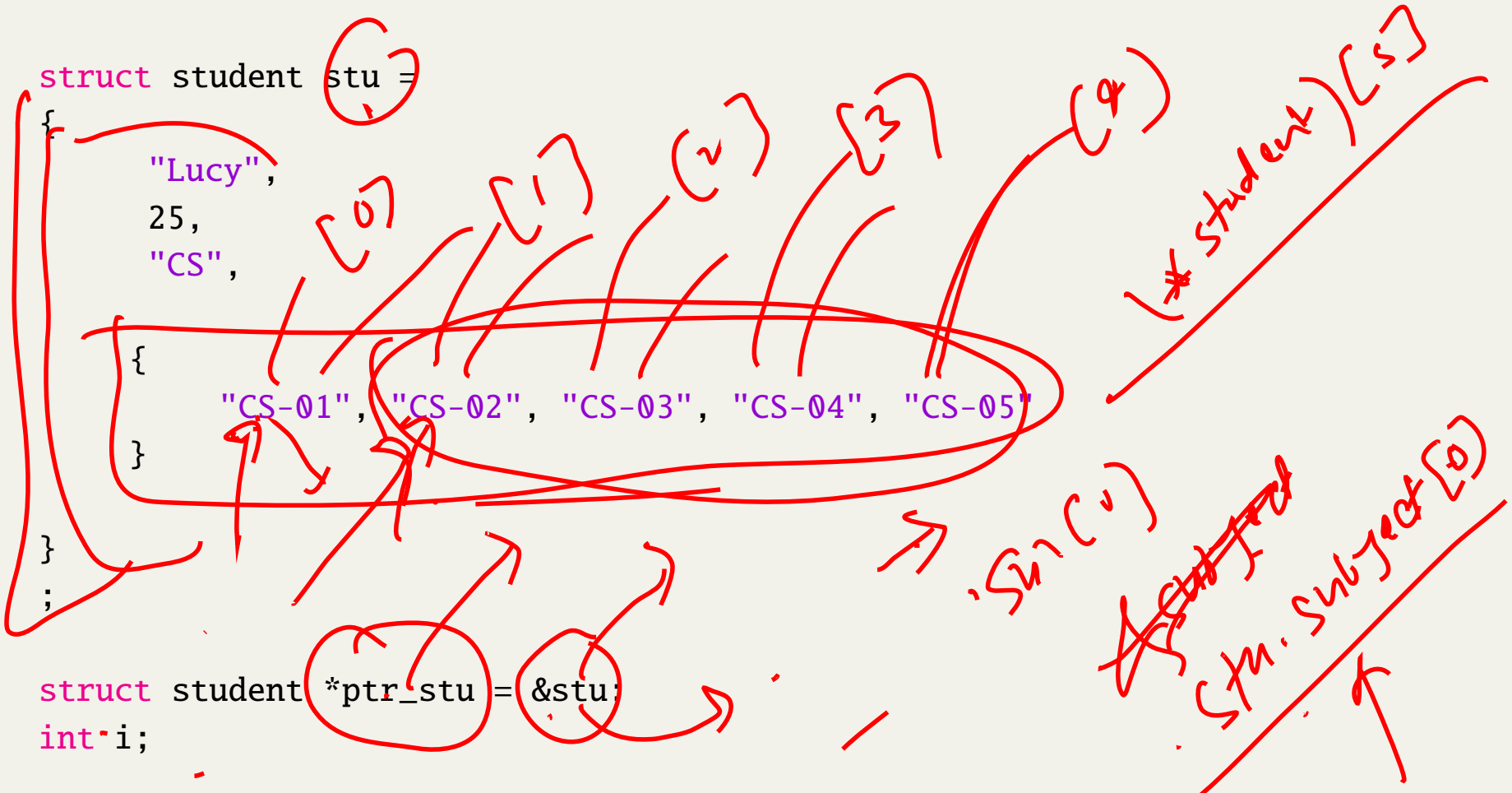
```
#include<stdio.h>

struct student
{
     char *name;
     int age;
     char *program;
     char *subjects[5];
}
;

```

Pointers as Structure Member in C II

```
12 int main()
13 {
14     struct student stu =
15     {
16         "Lucy",
17         25,
18         "CS",
19
20         {
21             "CS-01", "CS-02", "CS-03", "CS-04", "CS-05"
22         }
23
24     }
25     ;
26
27     struct student *ptr_stu = &stu;
28     int i;
29
30     printf("Accessing members using structure variable: \n\n");
31
32     printf("Name: %s\n", stu.name);
```



Pointers as Structure Member in C III

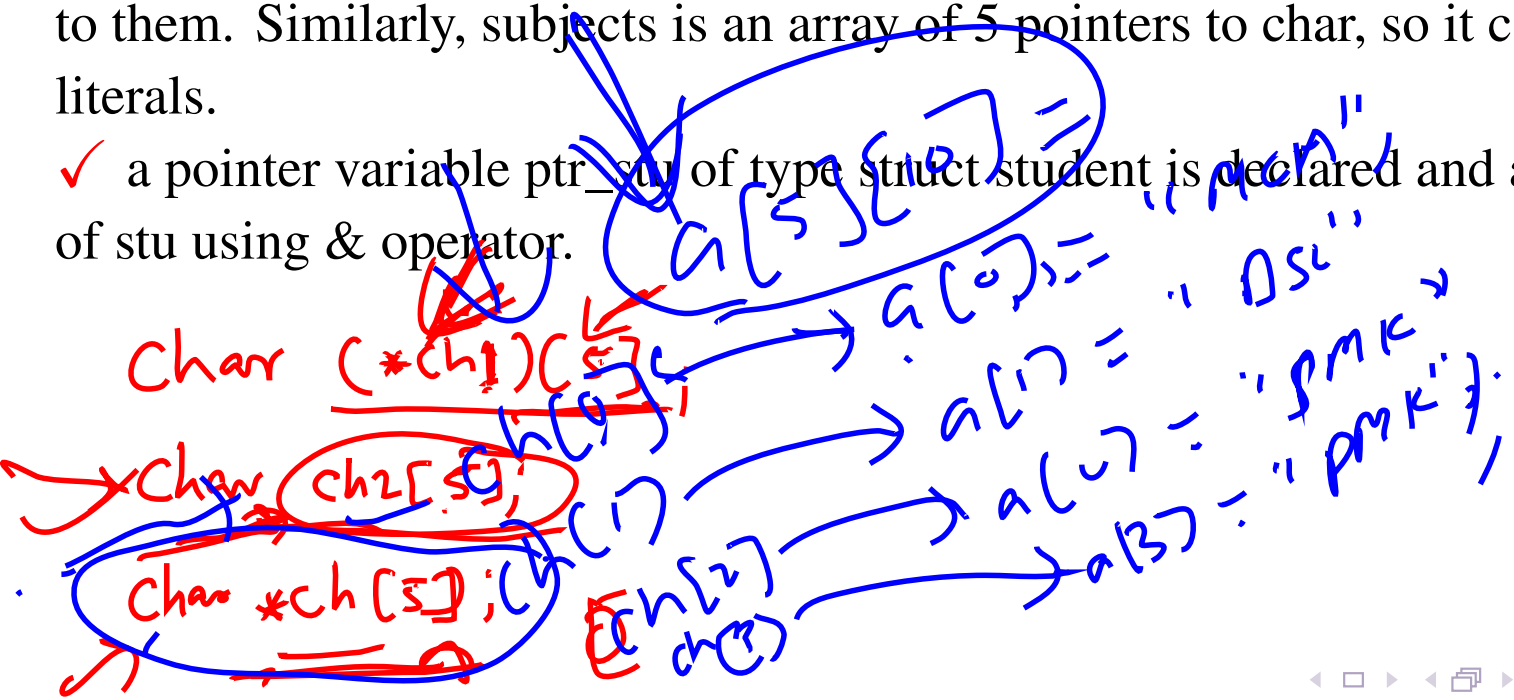
```
33      printf("Age: %d\n", stu.age);
34      printf("Program enrolled: %s\n", stu.program);
35
36      for(i = 0; i < 5; i++)
37      {
38           printf("Subject : %s \n", stu.subjects[i]);
39      }
40
41      printf("\n\nAccessing members using pointer variable: \n\n");
42
43      printf("Name: %s\n", ptr_stu->name);
44      printf("Age: %d\n", ptr_stu->age);
45      printf("Program enrolled: %s\n", ptr_stu->program);
46
47      for(i = 0; i < 5; i++)
48      {
49           printf("Subject : %s \n", ptr_stu->subjects[i]);
50      }
51
52      return 0;
53 }
```

Pointers as Structure Member in C IV

Name	Type
`name`	a pointer to `char`
`age`	`int`
`program`	a pointer to `char`
`subjects`	an array of `5` pointers to `char`

✓ Since name and program are pointers to char so we can directly assign string literals to them. Similarly, subjects is an array of 5 pointers to char, so it can hold 5 string literals.

✓ a pointer variable ptr_stu of type struct student is declared and assigned the address of stu using & operator.



✓ Like all other types, we can pass structures as arguments to a function.

✓ In fact, we can pass, individual members, structure variables, a pointer to structures etc to the function.

✓ Similarly, functions can return either an individual member or structures variable or pointer to the structure.

Passing Structure Members as arguments to Function:

✓ We can pass individual members to a function just like ordinary variables.

```
#include<stdio.h>
/*structure is defined above all functions so it is global.*/
struct student
{
     char name[20];
     int roll_no;
     int marks;
}
;
void print_struct(char name[], int roll_no, int marks);

int main()
```

Structures and Functions in C II

```
13 {
14      struct student stu =
15      {
16           "Tim", 1, 78
17      }
18      ;
19      print_struct(stu.name, stu.roll_no, stu.marks);
20      return 0;
21 }
22
23 void print_struct(char name[], int roll_no, int marks)
24 {
25      printf("Name: %s\n", name);
26      printf("Roll no: %d\n", roll_no);
27      printf("Marks: %d\n", marks);
28      printf("\n");
29 }
```

Passing Structure Variable as Argument to a Function:

✓ If a structure contains two-three members then we can easily pass them to function but what if there are 9-10 or more members ? Certainly passing 9-10 members is a tiresome and error-prone process.

✓ So in such cases instead of passing members individually, we can pass structure variable itself.

```
#include<stdio.h>

/*
structure is defined above all functions so it is global.
*/

struct student
{
     char name[20];
     int roll_no;
     int marks;
}
;

```

Structures and Functions in C IV

```
15 void print_struct(struct student stu);
16
17 int main()
18 {
19      struct student stu =
20      {
21           "George", 10, 69
22      }
23      ;
24      print_struct(stu);
25      return 0;
26 }
27
28 void print_struct(struct student stu)
29 {
30      printf("Name: %s\n", stu.name);
31      printf("Roll no: %d\n", stu.roll_no);
32      printf("Marks: %d\n", stu.marks);
33      printf("\n");
34 }
```

Passing Structure Pointers as Argument to a Function:

Although passing structure variable as an argument allows us to pass all the members of the structure to a function there are some downsides to this operation.

✓ A copy of the structure is passed to the formal argument. If the structure is large and you are passing structure variables frequently then it can take quite a bit of time which make the program inefficient.

✓ Additional memory is required to save every copy of the structure.

The following program demonstrates how to pass structure pointers as arguments to a function.

```
1  #include<stdio.h>
2
3  /*
4  structure is defined above all functions so it is global.
5  */
6
7  struct employee
8  {
9       char name[20];
10      int age;
```

Structures and Functions in C VI

```
11      char doj[10]; // date of joining
12      char designation[20];
13  }
14  ;
15
16  void print_struct(struct employee *);
17
18  int main()
19  {
20      struct employee dev =
21      {
22          "Jane", 25, "25/2/2015", "Developer"
23      }
24      ;
25      print_struct(&dev);
26
27      return 0;
28  }
29
30  void print_struct(struct employee *ptr)
31  {
```

```
32      printf("Name: %s\n", ptr->name);
33      printf("Age: %d\n", ptr->age);
34      printf("Date of joining: %s\n", ptr->doj);
35      printf("Age: %s\n", ptr->designation);
36      printf("\n");
37  }
```

The downside of passing structure pointer to a function is that the function can modify the original structure.

✓ If that is what you intentionally want that's fine. However, if don't want functions to modify original structure use the const keyword. Recall that const keyword when applied to a variable makes it read-only.

Structures and Functions in C VIII

```c
#include<stdio.h>

/*
structure is defined above all functions so it is global.
*/

struct employee
{
     char name[20];
     int age;
     char doj[10]; // date of joining
     char designation[20];
}
;

void print_struct(const struct employee *);

int main()
{
     struct employee dev =
     {
```

Structures and Functions in C IX

```
22              "Jane", 25, "25/2/2015", "Developer"
23      }
24      ;
25      print_struct(&dev);
26
27      return 0;
28  }
29
30  void print_struct(const struct employee *ptr)
31  {
32      printf("Name: %s\n", ptr->name);
33      printf("Age: %d\n", ptr->age);
34      printf("Date of joining: %s\n", ptr->doj);
35      printf("Age: %s\n", ptr->designation);
36
37      //ptr->age = 11;
38
39      printf("\n");
40  }
```

Array of Structures as Function Arguments:

✓ we can pass an array of structures to a function.

```
#include<stdio.h>

/*
structure is defined above all functions so it is global.
*/

struct company
{
     char name[20];
     char ceo[20];
     float revenue; // in $
     float pps; // price per stock in $
}
;

void print_struct(const struct company str_arr[]);

int main()
{
```

Structures and Functions in C XI

```
20
21      struct company companies[3] =
22      {
23
24          {
25              "Country Books", "Tim Green", 999999999, 1300
26          }
27          ,
28
29          {
30              "Country Cooks", "Jim Green", 9999999, 700
31          }
32          ,
33
34          {
35              "Country Hooks", "Sim Green", 99999, 300
36          }
37          ,
38
39      }
40      ;
```

Structures and Functions in C XII

```
41      print_struct(companies);
42
43      return 0;
44  }
45
46  void print_struct(struct company str_arr[])
47  {
48      int i;
49
50      for(i= 0; i<3; i++)
51      {
52          printf("Name: %s\n", str_arr[i].name);
53          printf("CEO: %d\n", str_arr[i].ceo);
54          printf("Revenue: %.2f\n", str_arr[i].revenue);
55          printf("Price per stock : %.2f\n", str_arr[i].pps);
56          printf("\n");
57      }
58  }
```

Returning Structure from Function:

✓ To return a structure from a function we must specify the appropriate return type in the function definition and declaration.

```
struct player check_health(struct player p);
{
     ...
}
```

```
#include<stdio.h>

/*
structure is defined above all functions so it is global.
*/

struct player
{
     char name[20];
     float height;
     float weight;
     float fees;
```

Structures and Functions in C XIV

```
13 }
14 ;
15
16 void print_struct(struct player p);
17 struct player deduct_fees(struct player p);
18
19 int main()
20 {
21      struct player p =
22      {
23           "Joe", 5.9, 59, 5000
24      }
25      ;
26      print_struct(p);
27      p = deduct_fees(p);
28      print_struct(p);
29
30      return 0;
31 }
32
33 struct player deduct_fees(struct player p)
```

```
34 {
35      p.fees -= 1000;
36      return p;
37 }
38
39 void print_struct(const struct player p)
40 {
41      printf("Name: %s\n", p.name);
42      printf("Height: %.2f\n", p.height);
43      printf("Weight: %.2f\n", p.weight);
44      printf("Fees: %.2f\n", p.fees);
45
46      printf("\n");
47 }
```

Returning a Structure Pointer from Function:

✓ To return structure pointer from a function all we need to do is to specify the appropriate return type in the function definition and function declaration.

```
struct *movie add_rating(struct movie *p);
{
     ...
}
```

✓ This function accepts an argument of type pointer to struct movie and returns and a pointer of type struct movie.

```
#include<stdio.h>

/*
structure is defined above all functions so it is global.
*/

struct movie
{
     char title[20];
     char language[20];
     char director[20];
     int year;
     int rating;
```

Structures and Functions in C XVII

```
14 }
15 ;
16
17 void print_struct(const struct movie *p);
18 struct movie *add_rating(struct movie *p);
19
20 int main()
21 {
22      struct movie m =
23      {
24           "The Accountant", "English" , "Gavin O'Connor", 2016, 1000
25      }
26      ;
27      struct movie *ptr_m1 = &m, *ptr_m2;
28
29      print_struct(ptr_m1);
30      ptr_m2 = add_rating(ptr_m1);
31      print_struct(ptr_m2);
32
33      return 0;
34 }
```

Structures and Functions in C XVIII

```
35
36 struct movie *add_rating(struct movie *p)
37 {
38      p->rating++; // increment rating by 1
39      return p;
40 }
41
42 void print_struct(const struct movie *p)
43 {
44      printf("Title: %s\n", p->title);
45      printf("Language: %s\n", p->language);
46      printf("Director: %s\n", p->director);
47      printf("Year: %d\n", p->year);
48      printf("Rating: %d\n", p->rating);
49
50      printf("\n");
51 }
```

typedef statement in C I

✓ The typedef is an advance feature in C language which allows us to create an alias or new name for an existing type or user defined type.

✓ The syntax of typedef is as follows:

Syntax: typedef data_type new_name;

typedef: It is a keyword.

data_type: It is the name of any existing type or user defined type created using structure/union.

new_name: alias or new name you want to give to any existing type or user defined type.

```
1 typedef int myint;
2
3 typedef int myint, integer;
4
5 typedef unsigned long int ulint;
6 typedef float real;
```

Example 1: Declaring a local alias using typedef

typedef statement in C II

```c
#include<stdio.h>
void foo(void);

int main()
{
     typedef unsigned char uchar;
     uchar ch = 'a';
     printf("ch inside main() : %c\n", ch);
     foo();
     return 0;
}

void foo(void)
{
     // uchar ch = 'a'; // Error
     unsigned char ch = 'z';
     printf("ch inside foo() : %c\n", ch);
}
```

Example 2: Declaring a global alias using typedef

typedef statement in C III

```c
#include<stdio.h>

typedef unsigned char uchar;
void foo(void);

int main()
{
     uchar ch = 'a';
     printf("ch inside main() : %c\n", ch);
     foo();
     return 0;
}

void foo(void)
{
     uchar ch = 'z';
     printf("ch inside foo() : %c\n", ch);
}
```

typedef statement in C IV

typedef with a pointer:

typedef int * iptr;

✓ After this statement iptr is an alias of a pointer to int or (int*). Here is how we can declare an integer pointer using iptr:

```
typedef int * iptr;
iptr p;
This declaration is same as:

int *p;

iptr a, *b; // same as int *a, **b;
iptr arr[10]; // same as int *arr[10];
```

typedef statement in C V

```c
#include<stdio.h>
typedef int * iptr;

int main()
{
     int a = 99;
     iptr p; // same as int *p
     p = &a;

     printf("%u\n", p);
     printf("%d\n", *p);

     return 0;
}
```

typedef with an Array:

typedef statement in C VI

```
typedef int iarr[10];
typedef int iarr[10];
After this declaration, iarr is an alias of array of 10 integer elements.

iarr a, b, c[5]; // same as int a[10], b[10], c[10][5];
```

✓ In this declaration, a and b are arrays of 10 integers and c is a 2-D array of dimension 10*5.

```
#include<stdio.h>
typedef int iarr[10];

int main()
{
     int i;

     // same as int a[10] =
     {
          12,43,45,65,67,87,89,91,14,19
     }
     iarr a =
```

```
13      {
14           12,43,45,65,67,87,89,91,14,19
15      }
16      ;
17
18      for(i = 0; i < 10; i++)
19      {
20           printf("a[%d] = %d\n",i ,a[i]);
21      }
22      return 0;
23 }
```

typedef with a Structure:

typedef statement in C VIII

```
struct book
{
     char title[20];
     char publisher[20];
     char author[20];
     int year;
     int pages;
}
;

typedef struct book Book;
Book b1 =
{
     "The Alchemist", "TDM Publication" , "Paulo Coelho", 1978, 331
}
;
```

We can also combine structure definition and typedef declaration. The syntax to so is:

typedef statement in C IX

```
typedef struct book
{
     char title[20];
     char publisher[20];
     char author[20];
     int year;
     int pages;
}
 Book;
```

```
#include<stdio.h>

typedef struct book
{
     char title[20];
     char publisher[20];
     char author[20];
     int year;
        int pages;
}
 Book;
```

typedef statement in C X

```
12
13 int main()
14 {
15
16      Book b1 =
17      {
18           "The Zahir",
19           "Harper Perennial" ,
20           "Paulo Coelho",
21            2005,
22            336
23
24      }
25      ;
26
27      printf("Title: %s\n", b1.title);
28      printf("Author: %s\n", b1.author);
29
30      return 0;
31 }
```