Structure

Why Use Structures

We have seen earlier how ordinary variables can hold one piece of information and how arrays can hold a number of pieces of information of the same data type. These two data types can handle a great variety of situations. But quite often we deal with entities that are collection of dissimilar data types.

For example, suppose you want to store data about a book. You might want to store its name (a string), its price (a float) and number of pages in it (an int). If data about say 3 such books is to be stored, then we can follow two approaches:

(a) Construct individual arrays, one for storing names, another for storing prices and still another for storing number of pages.
(b) Use a structure variable.

Let us examine these two approaches one by one. For the sake of programming convenience assume that the names of books would be single character long. Let us begin with a program that uses arrays.

```
main( )
{
 char  name[3] ;
 float  price[3] ;
 int  pages[3], i ;
 printf ( "\nEnter names, prices and no. of pages of 3 books\n" ) ;
 for ( i = 0 ; i <= 2 ; i++ )
     scanf ( "%c %f %d", &name[i], &price[i], &pages[i] );
 printf ( "\nAnd this is what you entered\n" ) ;
      for ( i = 0 ; i <= 2 ; i++ )
          printf ( "%c %f %d\n", name[i], price[i], pages[i] ); }
```

And here is the sample run...

Enter names, prices and no. of pages of 3 books A 100.00 354 C 256.50 682 F 233.70 512

And this is what you entered A 100.000000 354 C 256.500000 682 F 233.700000 512

This approach no doubt allows you to store names, prices and number of pages. But as you must have realized, it is an unwieldy approach that obscures the fact that you are dealing with a group of characteristics related to a single entity—the book.

The program becomes more difficult to handle as the number of items relating to the book go on increasing. For example, we would be required to use a number of arrays, if we also decide to store name of the publisher, date of purchase of book, etc. To solve this problem, C provides a special data type—the structure.

A structure contains a number of data types grouped together. These data types may or may not be of the same type. The following example illustrates the use of this data type.

```
main( ) {
 struct book  {
  char  name ;
  float  price ;
  int  pages ;
  } ;
 struct book  b1, b2, b3 ;
 printf ( "\nEnter names, prices & no. of pages of 3 books\n" ) ;
 scanf ( "%c %f %d", &b1.name, &b1.price, &b1.pages ) ;
  scanf ( "%c %f %d", &b2.name, &b2.price, &b2.pages ) ;
  scanf ( "%c %f %d", &b3.name, &b3.price, &b3.pages ) ;
  printf ( "\nAnd this is what you entered" ) ;
  printf ( "\n%c %f %d", b1.name, b1.price, b1.pages ) ;
  printf ( "\n%c %f %d", b2.name, b2.price, b2.pages ) ;
   printf ( "\n%c %f %d", b3.name, b3.price, b3.pages ) ; }
```

And here is the output...

Enter names, prices and no. of pages of 3 books A 100.00 354 C 256.50 682 F 233.70 512

And this is what you entered A 100.000000 354 C 256.500000 682 F 233.700000 512

Defining structure

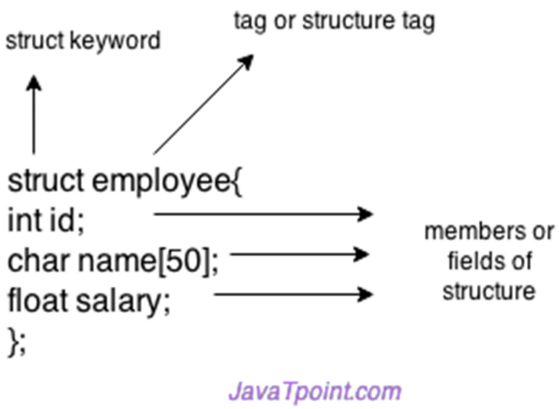
The **struct** keyword is used to define structure. Let's see the syntax to define structure in c.

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

Let's see the example to define structure for employee in c.

```
struct employee
{   int id;
   char name[50];
   float salary;
};
```

Here, **struct** is the keyword, **employee** is the tag name of structure; **id**, **name** and **salary** are the members or fields of the structure. Let's understand it by the diagram given below:



Let us now look at these concepts one by one.

Declaring structure variable

We can declare variable for the structure, so that we can access the member of structure easily. There are two ways to declare structure variable:

(a) By struct keyword within main() function

(b) By declaring variable at the time of defining structure.

1st way:

Let's see the example to declare structure variable by struct keyword. It should be declared within the main function.

```
struct employee
{   int id;
    char name[50];
    float salary;
};
```

Now write given code inside the main() function.

```
struct employee e1, e2;
```

2nd way:

Let's see another way to declare variable at the time of defining structure.

```
struct employee
{   int id;
    char name[50];
    float salary;
}e1,e2;
```

Which approach is good

But if no. of variable are not fixed, use 1st approach. It provides you flexibility to declare the structure variable many times.

If no. of variables are fixed, use 2nd approach. It saves your code to declare variable in main() fuction.

More Explanation

In our example program, the following statement declares the structure type:

```
struct book {
 char  name ;
 float  price ;
 int  pages ;
} ;
```

This statement defines a new data type called struct book. Each variable of this data type will consist of a character variable called name, a float variable called price and an integer variable called pages. The general form of a structure declaration statement is given below:

```
struct <structure name> {
structure element 1 ;
structure element 2 ;
structure element 3 ;
```

.....

.....

} ;

Once the new structure data type has been defined one or more variables can be declared to be of that type. For example the variables b1, b2, b3 can be declared to be of the type struct book, as,

struct book b1, b2, b3 ;

This statement sets aside space in memory. It makes available space to hold all the elements in the structure—in this case, 7 bytes—one for name, four for price and two for pages. These bytes are always in adjacent memory locations.

If we so desire, we can combine the declaration of the structure type and the structure variables in one statement.

For example,

struct book {

char name ;

float price ;

int pages ;

} ;

struct book b1, b2, b3 ;

is same as...

struct book {

char name ;

float price ;

int pages ;

} b1, b2, b3 ;

or even...

```
struct
{
char  name ;
float  price ;
int  pages ;
} b1, b2, b3 ;
```

Initialization

Like primary variables and arrays, structure variables can also be initialized where they are declared. The format used is quite similar to that used to initiate arrays.

```
struct book {
  char  name[10] ;
 float  price ;
int  pages ;
} ;
struct book  b1 = { "Basic", 130.00,  550 } ;
 struct book  b2 = { "Physics", 150.80, 800 } ;
```

Note the following points while declaring a structure type:

(a) The closing brace in the structure type declaration must be followed by a semicolon.
(b) It is important to understand that a structure type declaration does not tell the compiler to reserve any space in memory. All a structure declaration does is, it defines the 'form' of the structure.
(c) Usually structure type declaration appears at the top of the source code file, before any variables or functions are defined. In very large programs they are usually put in a separate header file, and the file is included (using the preprocessor directive #include) in whichever program we want to use this structure type.

Accessing Structure Elements

Having declared the structure type and the structure variables, let us see how the elements of the structure can be accessed.

In arrays we can access individual elements of an array using a subscript. Structures use a different scheme. They use a dot (.) operator. So to refer to pages of the structure defined in our sample program we have to use,

b1.pages

Similarly, to refer to price we would use,

b1.price

Note that before the dot there must always be a structure variable and after the dot there must always be a structure element.

How Structure Elements are Stored

Whatever be the elements of a structure, they are always stored in contiguous memory locations. The following program would illustrate this:

```
/* Memory map of structure elements */ main( ) {
 struct book  {    char  name ;
float  price ;
  int  pages ;  } ;
 struct book  b1 = { 'B', 130.00, 550 } ;
  printf ( "\nAddress of name = %u", &b1.name ) ;
  printf ( "\nAddress of price = %u", &b1.price ) ;
  printf ( "\nAddress of pages = %u", &b1.pages ) ;
 }
```

Here is the output of the program...

Address of name = 65518
Address of price = 65519
Address of pages = 65523

Actually the structure elements are stored in memory as shown in the Figure 10.1.

b1

b1.price	b1.name	b1.pages
'B'	550	130.00
65518	65519	65523

Array of Structures

In our sample program, to store data of 100 books we would be required to use 100 different structure variables from b1 to b100, which is definitely not very convenient. A better approach would be to use an array of structures. Following program shows how to use an array of structures.

```
/* Usage of an array of structures */

main( ) {

 struct book  {

 char  name ;

 float  price ;

 int  pages ;

 } ;


 struct book  b[100] ;  int  i ;
```

```
 for ( i = 0 ; i <= 99 ; i++ )  {
  printf ( "\nEnter name, price and pages " ) ;
  scanf ( "%c %f %d", &b[i].name, &b[i].price, &b[i].pages ) ;
  }
 for ( i = 0 ; i <= 99 ; i++ )
   printf ( "\n%c %f %d", b[i].name, b[i].price, b[i].pages ) ;
return 0;
 }
 linkfloat( ) {
float a = 0, *b ;  b = &a ;  /* cause emulator to be linked */
   a = *b ;    /* suppress the warning - variable not used */
}
```

Now a few comments about the program:

(a) Notice how the array of structures is declared...

```
 struct book b[100] ;
```

This provides space in memory for 100 structures of the type struct book.

(b) The syntax we use to reference each element of the array b is similar to the syntax used for arrays of ints and chars. For example, we refer to zeroth book's price as b[0].price. Similarly, we refer first book's pages as b[1].pages.

(c) It should be appreciated what careful thought Dennis Ritchie has put into C language. He first defined array as a collection of similar elements; then realized that dissimilar data types that are often found in real life cannot be handled using arrays, therefore created a new data type called structure. But even using structures programming convenience could not be achieved, because a lot of variables (b1 to b100 for storing data about hundred books) needed to be handled. Therefore he allowed us to create an array of structures; an array of similar data types which themselves are a collection of dissimilar data types. Hats off to the genius!

(d) In an array of structures all elements of the array are stored in adjacent memory locations. Since each element of this array is a structure, and since all structure elements are always stored in adjacent locations you can very well visualise the arrangement of array of structures in memory. In our example, b[0]'s name, price and pages in memory would be immediately followed by b[1]'s name, price and pages, and so on.

(e) What is the function linkfloat( ) doing here? If you don't define it you are bound to get the error "Floating Point Formats Not Linked" with majority of C Compilers. What causes this error to occur? When parsing our source file, if the compiler encounters a reference to the address of a float, it sets a flag to have the linker link in the floating-point emulator. A floating point emulator is used to manipulate floating point numbers in runtime library functions like

scanf( ) and atof( ). There are some cases in which the reference to the float is a bit obscure and the compiler does not detect the need for the emulator. The most common is using scanf( ) to read a float in an array of structures as shown in our program. How can we force the formats to be linked? That's where the linkfloat( ) function comes in. It forces linking of the floating-point emulator into an application. There is no need to call this function, just define it anywhere in your program.

One More Example

Let's see an example of structure with array that stores information of 5 students and prints it.

```
#include<stdio.h>

#include <string.h>

struct student{

int rollno;

char name[10];

};

int main(){

int i;

struct student st[5];

printf("Enter Records of 5 students");

for(i=0;i<5;i++){

printf("\nEnter Rollno:");

scanf("%d",&st[i].rollno);
```

```
printf("\nEnter Name:");
scanf("%s",&st[i].name);
}
printf("\nStudent Information List:");
for(i=0;i<5;i++){
printf("\nRollno:%d, Name:%s",st[i].rollno,st[i].name);
}
  return 0;
}
```

Output:

```
Enter Records of 5 students
Enter Rollno:1
Enter Name:Sonoo
Enter Rollno:2
Enter Name:Ratan
Enter Rollno:3
Enter Name:Vimal
Enter Rollno:4
Enter Name:James
Enter Rollno:5
Enter Name:Sarfraz

Student Information List:
Rollno:1, Name:Sonoo
Rollno:2, Name:Ratan
```

Rollno:3, Name:Vimal

Rollno:4, Name:James

Rollno:5, Name:Sarfraz

Copying of Structures

The values of a structure variable can be assigned to another structure variable of the same type using the assignment operator. It is not necessary to copy the structure elements piece-meal. Obviously, programmers prefer assignment to piece-meal copying. This is shown in the following example.

```
main( ) {
struct employee  {
  char  name[10] ;
  int  age ;
  float  salary ;
} ;
struct employee  e1 = { "Sanjay", 30, 5500.50 } ;
 struct employee  e2, e3 ;

/* piece-meal copying */
strcpy ( e2.name, e1.name ) ;
e2.age = e1.age ;
e2.salary = e1.salary ;

/* copying all elements at one go */
e3 = e2 ;
```

```
printf ( "\n%s %d %f", e1.name, e1.age, e1.salary ) ;

printf ( "\n%s %d %f", e2.name, e2.age, e2.salary ) ;

printf ( "\n%s %d %f", e3.name, e3.age, e3.salary ) ; }
```

The output of the program would be...

```
Sanjay 30 5500.500000

Sanjay 30 5500.500000

Sanjay 30 5500.500000
```

Nested Structure

Nested structure in c language can have another structure as a member. There are two ways to define nested structure in c language:

(1) By separate structure

(2) By Embedded structure

1) Separate structure

We can create 2 structures, but dependent structure should be used inside the main structure as a member. Let's see the code of nested structure.

```
struct Date

{

  int dd;

  int mm;

  int yyyy;
```

```
};
struct Employee
{
  int id;
  char name[20];
  struct Date doj;
}emp1;
```

As you can see, doj (date of joining) is the variable of type Date. Here doj is used as a member in Employee structure. In this way, we can use Date structure in many structures.

---

2) Embedded structure

We can define structure within the structure also. It requires less code than previous way. But it can't be used in many structures.

```
struct Employee
{
  int id;
  char name[20];
  struct Date
   {
     int dd;
     int mm;
     int yyyy;
   }doj;
}emp1;
```

C Nested Structure example 1

```
main( ) {
    struct address
       {
       char  phone[15] ;
      char  city[25] ;
      int  pin ;
    } ;
 struct emp  {
   char  name[25] ;
   struct address  a ;
      } ;
  struct emp  e = { "jeru", "531046", "nagpur", 10 };
 printf ( "\nname = %s phone = %s", e.name, e.a.phone ) ;
 printf ( "\ncity = %s pin = %d", e.a.city, e.a.pin ) ;
}
```

And here is the output...

name = jeru phone = 531046 city = nagpur pin = 10

Notice the method used to access the element of a structure that is part of another structure. For this the dot operator is used twice, as in the expression,

e.a.pin or e.a.city

C Nested Structure example 2

Let's see a simple example of nested structure in C language.

```
#include <stdio.h>
#include <string.h>
struct Employee
{
  int id;
  char name[20];
  struct Date
   {
     int dd;
     int mm;
     int yyyy;
   }doj;
}e1;
int main( )
{
  //storing employee information
  e1.id=101;
  strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array
  e1.doj.dd=10;
  e1.doj.mm=11;
  e1.doj.yyyy=2014;

  //printing first employee information
```

```
   printf( "employee id : %d\n", e1.id);
   printf( "employee name : %s\n", e1.name);
   printf( "employee date of joining (dd/mm/yyyy) : %d/%d/%d\n", e1.doj.dd,e1.doj.mm,e1.doj.y
yyy);
   return 0;
}
```

Output:

employee id : 101

employee name : Sonoo Jaiswal

employee date of joining (dd/mm/yyyy) : 10/11/2014

Pointer to Structure

The way we can have a pointer pointing to an int, or a pointer pointing to a char, similarly we can have a pointer pointing to a struct. Such pointers are known as 'structure pointers'. Let us look at a program that demonstrates the usage of a structure pointer.

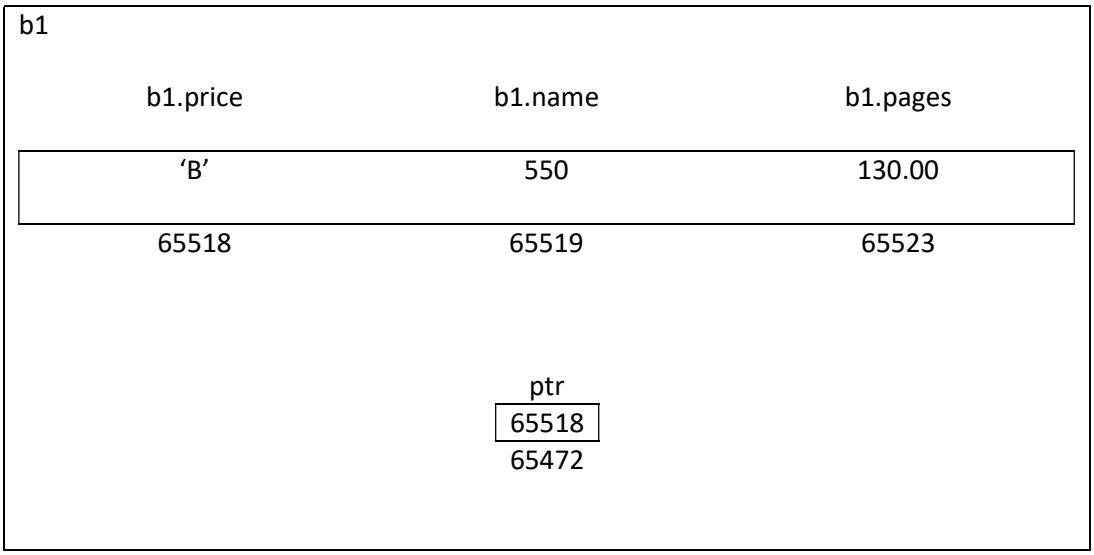
```
main( ) {
 struct book   {
  char  name[25] ;
  char  author[25] ;
  int  callno ;
 } ;
 struct book  b1 = { "Let us C", "YPK", 101 } ;
 struct book  *ptr ;
 ptr = &b1 ;
```

```
 printf ( "\n%s %s %d", b1.name, b1.author, b1.callno ) ;

 printf ( "\n%s %s %d", ptr->name, ptr->author, ptr->callno ) ;

}
```

The first printf( ) is as usual.

The second printf( ) however is peculiar. We can't use ptr.name or ptr.callno because ptr is not a structure variable but a pointer to a structure, and the dot

operator requires a structure variable on its left. In such cases C provides an operator ->, called an arrow operator to refer to the structure elements. Remember that on the left hand side of the '.' structure operator, there must always be a structure variable, whereas on the left hand side of the '->' operator there must always be a pointer to a structure. The arrangement of the structure variable and pointer to structure in memory is shown in the following Figure .



Passing Structure to function

Like an ordinary variable, a structure variable can also be passed to a function. We may either pass individual structure elements or the entire structure variable at one go. Let us examine both the approaches one by one using suitable programs.

```
/* Passing individual structure elements */

main( ) {

 struct book   {

 char  name[25] ;

  char  author[25] ;

  int  callno ;

 } ;

 struct book b1 = { "Let us C", "YPK", 101 } ;

 display ( b1.name, b1.author, b1.callno ) ;

}


display ( char  *s, char  *t, int  n )

{

 printf ( "\n%s %s %d", s, t, n ) ;

 }
```

And here is the output...

Observe that in the declaration of the structure, name and author have been declared as arrays. Therefore, when we call the function display( ) using,

```
display ( b1.name, b1.author, b1.callno ) ;
```

we are passing the base addresses of the arrays name and author, but the value stored in callno. Thus, this is a mixed call—a call by reference as well as a call by value.

It can be immediately realized that to pass individual elements would become more tedious as the number of structure elements go on increasing.

Call By Value

A better way would be to pass the entire structure variable at a time. This method is shown in the following program.

```
struct book  {
char  name[25] ;
 char  author[25] ;
  int  callno ;
 } ;
main( ) {
struct book  b1 = { "Let us C", "YPK", 101 } ;
display ( b1 ) ;
 }
display ( struct book  b )
 {
printf ( "\n%s %s %d", b.name, b.author, b.callno ) ;
 }
```

And here is the output...

Let us C YPK 101

Note that here the calling of function display( ) becomes quite compact,

```
display ( b1 ) ;
```

Having collected what is being passed to the display( ) function, the question comes, how do we define the formal arguments in the function. We cannot say,

```
struct book b1 ;
```

because the data type struct book is not known to the function display( ). Therefore, it becomes necessary to define the structure type struct book outside main( ), so that it becomes known to all functions in the program.

Call By Reference

```
/* Passing address of a structure variable */

struct book  {

 char  name[25] ;

  char  author[25] ;

   int  callno ;

  } ;


main( ) {

 struct book  b1 = { "Let us C", "YPK", 101 } ;

  display ( &b1 ) ;

}


display ( struct book  *b )

 {

  printf ( "\n%s %s %d", b->name, b->author, b->callno ) ;

 }
```

And here is the output...

Let us C YPK 101

Again note that to access the structure elements using pointer to a structure we have to use the '->' operator.

Also, the structure struct book should be declared outside main( ) such that this data type is available to display( ) while declaring pointer to the structure.

Uses of Structures

Where are structures useful? The immediate application that comes to the mind is Database Management. That is, to maintain data about employees in an organization, books in a library, items in a store, financial accounting transactions in a company etc. But mind you, use of structures stretches much beyond database management. They can be used for a variety of purposes like:

(a) Changing the size of the cursor

(b) Clearing the contents of the screen

(c)Placing the cursor at an appropriate position on screen

(d) Drawing any graphics shape on the screen
(e) Receiving a key from the keyboard
(f) Checking the memory size of the computer
(g) Finding out the list of equipment attached to the computer
(h) Formatting a floppy
(i) Hiding a file from the directory
(j) Displaying the directory of a disk
(k) Sending the output to printer
(l) Interacting with the mouse

C Program to Store Information of a Student Using Structure

```
#include <stdio.h>
struct student
{
```

```
    char name[50];
    int roll;
    float marks;
} s;

int main()
{
    printf("Enter information:\n");

    printf("Enter name: ");
    scanf("%s", s.name);

    printf("Enter roll number: ");
    scanf("%d", &s.roll);

    printf("Enter marks: ");
    scanf("%f", &s.marks);


    printf("Displaying Information:\n");

    printf("Name: ");
    puts(s.name);

    printf("Roll number: %d\n",s.roll);

    printf("Marks: %.1f\n", s.marks);

    return 0;
}
```

Output

```
Enter information:
Enter name: Jack
Enter roll number: 23
Enter marks: 34.5
Displaying Information:
Name: Jack
Roll number: 23
Marks: 34.5
```

Question

Create a structure to specify data on students given below:

Roll number, Name, Department, Course, Year of joining

Assume that there are not more than 450 students in the college.

(a) Write a function to print names of all students who joined in a particular year.
(b) Write a function to print the data of a student whose roll number is received by the function.

Solution

Program: /* create structure to hold student data */

```
#include <stdio.h>
struct stud {
int r_n; /* Roll Number */
char name[ 20]; /* Name */
char dep[ 15]; /* Department */
char course[ 10]; /* Course */
int y_o_j; /*Year Of Joining */
};
struct stud s[ 450]; /* array of structure */
void set_student_data();
void display();
void name_acc_year (int);
int data_acc_rollno (int);

int main() {
int i, r;
int y;

printf ("\nEnter the data for each Student:\n\n "); /* Initialize the values for the students structure */

set_student_data();

/* Display all the elements of the student structure */
display();
/* Search data on year of Joining */
printf ("\nEnter the Year of Joining of the Student ");
scanf ("%d", &y);
name_acc_year (y); /* year of joining passed to function */
/* Search data based on roll number */
printf ("\nEnter the Roll Number of the Student");
scanf ("%d", &r);
data_acc_rollno (r); /* roll number passed to function */
return 0;
}

void set_student_data() /* Enter student data */
{
int i;
for (i = 0; i < 450; i++) {
fflush (stdin); /* Flush the input buffer */
printf ("\nEnter the Roll NUmber of the student\n");
scanf ("%d", &s[ i ].r_n);
fflush (stdin);
```

```
 printf ("Enter the name of the student\n");
 scanf ("%s", s[ i ].name);
 fflush (stdin); printf ("Enter the name of the Department\n");
 scanf ("%s", s[ i ].dep);
 fflush (stdin);
 printf ("Enter the name of the Course\n");
scanf ("%s", s[ i ].course);
 fflush (stdin);
printf ("Enter the Year of Joining of the student\n");
 scanf ("%d", &s[ i ].y_o_j);

}

/* function to display data */
void display() {
int i;
 for (i = 0; i < 450; i++) {
 printf ("\n\tRoll Number of student %d = %d \n", i+1, s[ i ].r_n);
 printf ("\n\tName of student %d = %s \n", i+1, s[ i ].name);
 printf ("\n\tName of the Department = %s \n", s[ i ].dep);
 printf ("\n\tName of the Course = %s \n", s[ i ].course);
printf ("\n\tYear of Joining of student %d = %d \n\n", i+1, s[ i ].y_o_j);
 }
 }

/* function to get name based on year of joining */
void name_acc_year (int y)
 {
 int i, j = 0;
 for (i = 0; i < 450; i++) {
 if (y == s[ i ].y_o_j)
   {
      printf ("%s joined in the year %d\n", s[ i ].name, s[ i ].y_o_j);
      j = 1;
   }
}
if (j == 0)
   printf ("\nNo student joined in the year %d", y);
}

/* function to get student data based on roll number */
int data_acc_rollno (int r)
{
 int i, j = 0;
for (i = 0; i < 450; i++)
{
if (s[ i ].r_n == r)
  {
```

```
  printf ("\n\tRoll Number of student = %d \n", s[ i ].r_n);
  printf ("\n\tName of student = %s \n", s[ i ].name);
  printf ("\n\tName of the Department = %s \n", s[ i ].dep);
  printf ("\n\tName of the Course = %s \n", s[ i ].course);
  printf ("\n\tYear of Joining of student = %d \n\n", s[ i ].y_o_j);
  j = 1;
  }
  }
if (j == 0)
    printf ("\nNo such Roll Number present.");

}
```