Chapter 10

Structures, Unions & Enumerations

Dr. Niroj Kumar Pani

nirojpani@gmail.com

Department of Computer Science Engineering & Applications

Indira Gandhi Institute of Technology

Sarang, Odisha

Chapter Outline…

- Structures
  - Structures: Why & What?
  - Working with Structures
  - Structure Initialization
  - Copying & Comparing Structure Variables
  - Array of Structures
  - Arrays within Structures
  - Structures within Structures
  - Pointers to Structures
  - Assignments - I

- Unions
  - All About Unions
  - Programming Example
  - Assignments - II
- Enumerations
  - All About Enumerations
  - Programming Examples

Structures

Structures: Why & What?

- **Why Structures?:** Some programming situations demands to combine / group dissimilar data types to represent one entity.

  For Example, suppose we are developing a program for some University and we want to represent a student. A "student" can be thought of as a combination of the following elements:

  - Registration number (int)
  - Name (string)
  - CGPA (float)

  C provides a technique to group dissimilar data types as one entity: the structures.

- **What is a Structure?:** A structure is a finite, ordered, collection of dissimilar data types that share a common name (variable name).

Working with Structures

- Working with structures involves three basic activities:
    1. Defining a structure
    2. Declaring structure variables
    3. Accessing structure elements / members

  We discuss them one-by-one in detail next.

Defining Structures

- In a structure definition, we specify the structure name and its elements.
- Syntax:

```
struct structure_name
{
    data_type element_1;
    data_type element_2;
    …
};
```

e.g.,

```
struct Student
{
    int regdNo;
    char name[30];
    float cgpa;
};
```

- NOTES:
  1. **When a structure is defined, only a new data type** (user-defined data type) **is created, e.g., the above code creates a new user-defined data type** "struct student". **However, no memory is allocated to it** (memory is allocated to a variable, not to a data type).
  2. **The structure elements can be of any data type: primary** (except void, i.e., int, char, float, double), **derived** (arrays, pointers), **and even user-defined** (structure, unions, enumeration). (We shall see some examples later.)

3. **Normally, a structure should be defined at the beginning of the program, outside of all functions. This allows the structure definition to be global and hance can be used by all the functions in the program (which we often want).**

However, if we exclusively want to make a structure local to a particular function, then we can define it within that function.

```
#include <stdio.h>
struct Student /* Ideal place for defining a structure. */
{
    …
};
void main()
{
    …
}
```

Declaring Structure Variables

- **After a structure is defined, only a new data type** (user-defined data type) **is created. So, in order to use it, we need to declare variables of its type.**
- **Syntax:**

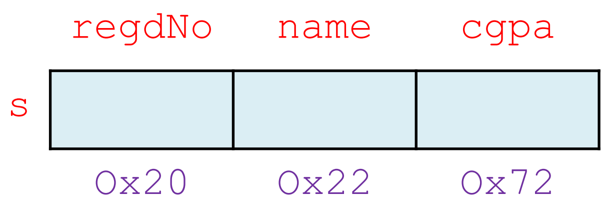
```
struct structure_name variable_name;
```

e.g.,

```
struct Student s;  /* A variable of type "struct Student" is created */
struct Student s1, s2;  /* Two variables of type "struct Student" are created */
```

- **NOTES:**
  1. **Memory allocation is done at the time of structure variable declaration. For Ex, the following code would result in the given memory allocation.**

```
struct Student
{
    int regdNo;
    char name[30];
    float cgpa;
};
struct Student s;
```



2. Normally, structure variables should be declared inside the function in which we need them. This makes the variables local to the function (which we often want).

However, if we exclusively want to make a structure variable global then we can declare it outside of all functions.

```
#include <stdio.h>
struct Student  /* Ideal place for defining a structure. */
{
    …
};
void main()
{
    struct Student s1, s2;  /* Ideal place for declaring structure variables. */
    …
}
```

3. **It is allowed to declare structure variables at the place of structure definition.**

Syntax 1:

```
struct Student
{
    int regdNo;
    char name[30];
    float cgpa;
} s1, s2, s3;  /* Three variables of type “struct student” are created */
```

Syntax 2: (If we declare structure variables at the place of structure definition, writing the “structure name” becomes optional.)

```
struct  /* No structure name */
{
    int regdNo;
    char name[30];
    float cgpa;
} s1, s2, s3;  /* Three variables of type “struct student” are created */
```

[NOTE]: Declaring structure variables at the place of structure definition should be avoided. for the following reasons.

- Since a structure is generally defined outside of all functions, declaring variable at the place of definition would make the structure variable global (which we may not want).
- If we follow “Syntax 2”, without a structure name, we can’t use it for future declarations.

4. **Type-Defined Structures: If we don't like to mention the keyword** "struct" **while we declare structure variables** (e.g., struct Student s;), **we can use** typedef **to define a structure as follows.**

```
typedef struct
{
    data_type element_1;
    data_type element_2;
    …
} structure_name;
```

e.g.,

```
typedef struct
{
    int regdNo;
    char name[30];
    float cgpa;
} Student;
```

The above code would mean that a new structure type **is created and is renamed to** "Student". **Therefore, the following declaration can be used to create structure variables.**

```
Student s;  /* A variable of type "Student" is created */
Student s1, s2, s3;  /* Three variables of type "Student" are created */
```

Accessing Structure Elements / Members

- **The structure elements (elements of a structure variable) can be accessed by using the** dot operator / period operator (“.”).

Example: For the following structure definition and declaration:

```
struct Student
{
    int regdNo;
    char name[30];
    float cgpa;
};
struct Student s;
```

The members of the structure variable ‘s’ **can be accessed as:**

```
s.redgNo;
s.name[];
s.cgpa;
```

Programming Example :

```
/* PR10_1.c: Define a structure "Student" having regd. no., name, and CGPA as attributes. Write a
program to read the information for one student from the keyboard and print the same on the screen. */

# include <stdio.h>
# include <conio.h>

struct Student  /* Structure definition */
{
   int regdNo;
   char name[30];
   float cgpa;
};

void main()
{
   struct Student s;  /* Structure declaration */

   printf("\nEnter the student\'s Regd. No., Name, and CGPA: ");
   scanf("%d %s %f", &s.regdNo, s.name, &s.cgpa); /* Accessing members*/

   printf("\nStudent Details\n---------------");
   printf("\nRegd. No.: %d\nName: %s\nCGPA: %f\n",
   s.regdNo, s.name, s.cgpa); /* Accessing members*/

   getch();
}
```

Output

```
Enter the student's Regd. No.,
Name, and CGPA: 12345 Kartik 9.62

Student Details
---------------
Regd. No.: 12345
Name: Kartik
CGPA: 9.620000
```

Structure Initialization

- Like any other variable, a structure variable can be initialized at the place of declaration or later in a separate statement.

  The following code demonstrates:

```
struct Person
{
    int id;
    char name[30];
    int age;
};

void main()
{
    struct Person per1 = {2391, "Ram", 25}; /* Declaration at initialization. */
    struct Person per2;
    ...
    per2 = {7895, "Kartik", 46}; /* Declaration in a separate statement. */
    ...
}
```

Copying & Comparing Structure Variables

- **Copying: Two variables of the same structure type can be copied the same way as ordinary variables by using the assignment operator ("=").**

  For Example: If per1, per2, **and** per3 **are variables of the same structure, then the following statements are valid.**

  ```
  per1 = per2;
  per2 = per3;
  ```

- **Comparing: C doesn't permit any logical operations on structure variables.**

  For Example: If per1, per2, **and** per3 **are some structure variables (of the same / different structures), the following statements are not permitted.**

  ```
  per1 == per2;
  per2 != emp3;
  ```

  [NOTE]: In case, we need to compare two structure variables of the same structure type, we may do so by comparing individual members.

■ Programming Example :

```
/* PR10_2.c: Copying and Comparing Structures. */

# include <stdio.h>
# include <conio.h>

typedef struct
{
   int id;
   char name[30];
   int age;
} Person;

void main()
{
   Person per1 = {12345, "Ravi", 32};
   Person per2;
   per2 = per1;  /* Copying */

   if(per1.id == per2.id)  /* Comparing */
   {
       printf("\nper1 and per2 are the same.\n\n");
       printf("Id: %d\nName: %s\nAge: %d\n",
       per1.id, per2.name, per1.age);
   }
   getch();
}
```

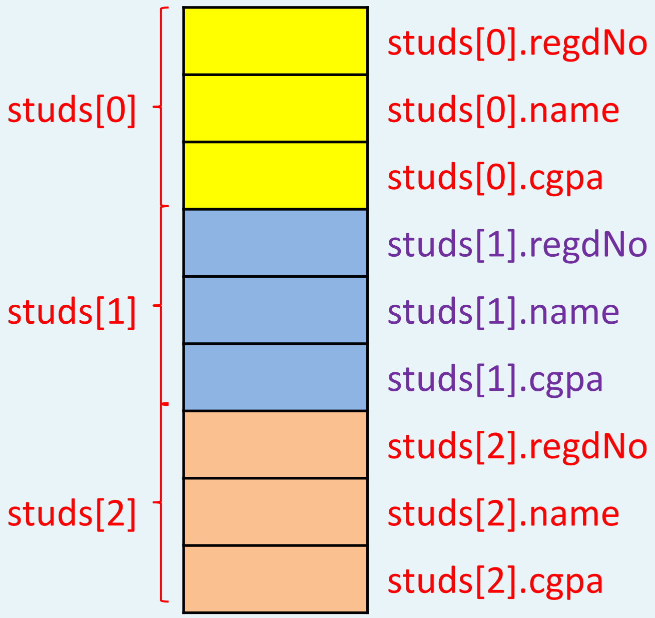
Output

```
per1 and per2 are the same.

Id: 12345
Name: Ravi
Age: 32
```

Array of Structures

- It is possible to declare an array of structures, each element of the array representing a structure variable.
- **Programming Example:**

```
/* PR10_3.c: PR10_3.c: An array of students. */

# include <stdio.h>
# include <conio.h>

struct Student
{
    int regdNo;
    char name[30];
    float cgpa;
};

void main()
{
    int i;

    struct Student studs[3]; /* An array consisting of 3 "struct Student" is created */
```



[Cont.]

```
    printf("\nEnter Students Information\n");
    printf("==========================\n");

    for (i=0;i<3;i++)
    {
        printf("\nStudent %d\n----------\n", i+1);
        printf("Regd. No.: ");
        scanf("%d", &studs[i].regdNo);
        printf("Name: ");
        scanf("%s", studs[i].name);
        printf("CGPA: ");
        scanf("%f", &studs[i].cgpa);
    }
    printf("\n\nStudents Details\n================\n");
    for (i=0;i<3;i++)
    {
        printf("\nDetails of Student %d\n--------------------", i+1);
        printf("\nRegd. No.: %d\nName: %s\nCGPA: %.2f\n",
        studs[i].regdNo, studs[i].name, studs[i].cgpa);
    }

    getch();
}
```

Output

```
Enter Students Information
==========================

Student 1
---------
Regd. No.: 12345
Name: Ravi
CGPA: 9.9

Student 2
---------
Regd. No.: 15698
Name: Krish
CGPA: 8.3

Student 3
---------
Regd. No.: 15623
Name: Kartik
CGPA: 9.2
```

[Cont.]

```
Students Details
==========================

Details of Student 1
--------------------
Regd. No.: 12345
Name: Ravi
CGPA: 9.90

Details of Student 2
--------------------
Regd. No.: 15698
Name: Krish
CGPA: 8.30

Details of Student 3
--------------------
Regd. No.: 15623
Name: Kartik
CGPA: 9.20
```

Arrays within Structures

- **As already pointed out, a structure element can be an array.**

  We have already done some programs having a string (a character array ending with '\0') "name" as a structure element. Let's do one more program containing an array as a structure element.

- **Programming Example:**

```
/* PR10_4.c: Array within Structure. */

# include <stdio.h>
# include <conio.h>
# define MAX 2
# define SEMESTERS 2

struct Student
{
   int regdNo;
   char name[30];
   float cgpa[SEMESTERS];
};
```

[Cont.]

```
void main()
{
    struct Student studs[MAX];
    int i, j;
    float sum = 0;

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

        printf("Regd. no: ");
        scanf("%d", &studs[i].regdNo);

        printf("Name: ");
        scanf("%s", studs[i].name);

        for(j = 0; j < SEMESTERS; j++)
        {
            printf("CGPA in Semester %d: ", j+1);
            scanf("%f", &studs[i].cgpa[j]);
        }
    }
```

[Cont.]

```
    printf("\nStudents Information");
    printf("\n====================\n\n");
    printf("Regd. No\tName\tAverage CGPA\n");
    printf("--------\t----\t------------\n");

    for(i = 0; i < MAX; i++ )
    {
        sum = 0;
        for(j = 0; j < SEMESTERS; j++)
        {
            sum = sum + studs[i].cgpa[j];
        }
        printf("%d\t\t%s\t\t%.2f\n",
        studs[i].regdNo, studs[i].name, sum/SEMESTERS);
    }

    getch();
}
```

Output

```
Enter details of student 1
--------------------------
Regd. no: 12345
Name: Rick
CGPA in Semester 1: 9.9
CGPA in Semester 2: 8.4

Enter details of student 2
--------------------------
Regd. no: 54898
Name: Tim
CGPA in Semester 1: 9.0
CGPA in Semester 2: 9.4

Students Information
=====================

Regd. No        Name     Average CGPA
--------        ----     ------------
12345           Ric              9.45
54898           Tim              8.90
```

Structures within Structures

- We can include a structure as a structure element (nesting of structures).
- Syntax 1:

```
struct Employee /* A structure "struct Employee" is defined. */
{
    char name[30];

    struct Date
    {
        int day;
        char month[10];
        int year;
    } joiningDate;  /* A structure "struct Date" is defined, and its variable is declared
                       within the structure "struct Employee. */

    float salary;
};
```

- **Syntax 2 (Preferred):**

```
struct Date  /* A structure “struct Date” is defined. */
{
    int day;
    char month[10];
    int year;
};

struct Employee  /* A structure “struct Employee” is defined. */
{
    char name[30];
    struct Date joiningDate; /* A variable of type “struct Date” is declared
                                within the structure “struct Employee. */
    float salary;
};
```

■ Programming Example:

```
/* PR10_5.c: Structure within structure. */

# include <stdio.h>
# include <conio.h>

typedef struct
{
   int day;
   char month[10];
   int year;
} Date;

struct Employee
{
   char name[30];
   Date joiningDate; /* Structure variable declared within another structure */
   float salary;
};
```

[Cont.]

```
void main()
{
   struct Employee e;

   printf("\nEnter employee information\n");
   printf("==========================\n\n");

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

   printf("Date of joining (dd mmm yyyy): ");
   scanf("%d %s %d",
   &e.joiningDate.day, e.joiningDate.month, &e.joiningDate.year);

   printf("Salary: ");
   scanf("%f", &e.salary);

   printf("\n\nEmployee Details\n----------------");
   printf("\nName: %s\nDate of Joining: %d %s %d\nSalary: %.2f ",
   e.name,
   e.joiningDate.day, e.joiningDate.month, e.joiningDate.year,
   e.salary);

   getch();
}
```

Output

```
Enter Employee Information
==========================
Name: Ramesh
Date of joining (dd mmm yyyy): 20 March 1999
Salary: 36000

Employee Details
----------------
Name: Remesh
Date of joining is: 20 March 1999
Salary: 36000.00
```

Pointers to Structures

- It is possible to declare a pointer to a structure and by using the pointer variable the structure members can be accessed.

  The following code demonstrates:

```
struct Student
{
    int regdNo;
    char name[30];
    float cgpa;
};

void main()
{
    struct Student *s;  /* s is a pointer to "struct Student", i.e.,
                           it can store the address of a variable of type "struct Student". */
    ...
}
```

- In such case, the structure members are accessed by either of the following ways:
  - indirection operator (*) along with a dot operator (.)
  - arrow operator (->) (most common)

For Example: For the structure definition and declaration given in the last slide, its members could be accessed as:

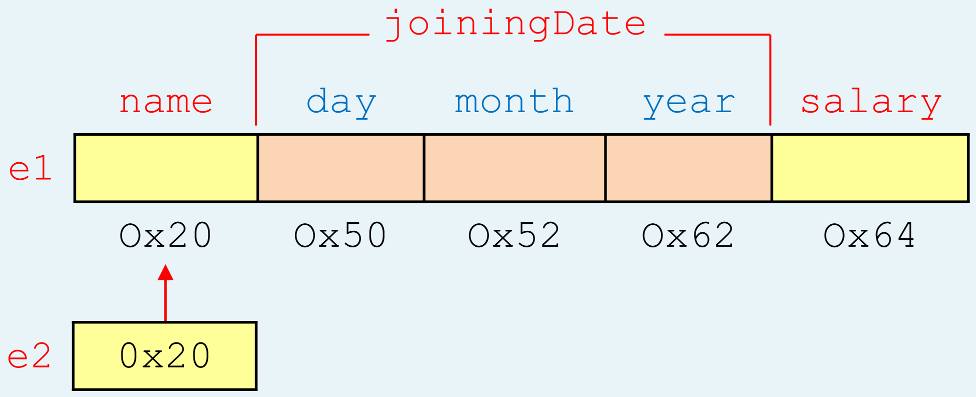
```
(*s).redgNo;
(*s).name[];
(*s).cgpa;
```

or

```
S->redgNo;
S->name[];
S->cgpa;
```

Programming Example:

```
/* PR10_6.c: Demonstration of pointers to structures. */

# include <stdio.h>

typedef struct
{
   int day;
   char month[10];
   int year;
} Date;

struct Employee
{
   char name[30];
   Date joiningDate;  /* Structure variable declared within another structure */
   float salary;
};

void main()
{
   struct Employee e1, *e2; /* e1 is a normal variable of type "struct Employee".
                               e2 is a pointer to "struct Employee", i.e., it can store the
                               address of a variable of type "struct Employee". */

   e2 = &e1;  /* A pointer variable must be assigned to an address of its type. */
```



[Cont.]

```
    printf("\nEnter employee information\n");
    printf("==========================\n\n");

    printf("Name: ");
    scanf("%s", e2->name); /* Access through pointer  */

    printf("Date of joining (dd mmm yyyy): ");
    scanf("%d %s %d", &e2->joiningDate.day, e2->joiningDate.month,
    &e2->joiningDate.year); /* Access through pointer  */

    printf("Salary: ");
    scanf("%f", &e2->salary); /* Access through pointer  */

    printf("\nEmployee Details\n----------------");

    printf("\nName: %s\nDate of Joining: %d %s %d\nSalary: %.2f ",
    e1.name, e1.joiningDate.day, e1.joiningDate.month,
    e1.joiningDate.year, e1.salary); /* Access through normal variable  */

    printf("\n\nName: %s\nDate of Joining: %d %s %d\nSalary: %.2f ",
    (*e2).name, (*e2).joiningDate.day, (*e2).joiningDate.month,
    (*e2).joiningDate.year, (*e2).salary); /* Access through pointer  */

    printf("\n\nName: %s\nDate of Joining: %d %s %d\nSalary: %.2f ",
    e2->name, e2->joiningDate.day, e2->joiningDate.month,
    e2->joiningDate.year, e2->salary); /* Access through pointer  */
}
```

Output

```
Enter Employee Information
==========================
Name: Ramesh
Date of joining (dd mmm yyyy): 20 March 1999
Salary: 36000

Employee Details
----------------
Name: Remesh
Date of joining is: 20 March 1999
Salary: 36000.00

Name: Remesh
Date of joining is: 20 March 1999
Salary: 36000.00

Name: Remesh
Date of joining is: 20 March 1999
Salary: 36000.00
```

Assignments - I

Complete the experiments given in “Lab Manual - Section 11”.

Unions

All About Unions

- Unions are a concept borrowed from structures.
  - Like a structure, a union is a finite, ordered, collection of dissimilar data types that share a common name.
  - All the concepts that we have discussed in structures are also applicable to unions.
- In terms of syntax, both structures and unions are almost the same (have minor differences). However, there is a major difference between them in terms of storage (memory allocation).

  We discuss the differences one-by-one.

■ Difference 1 - Regarding Definition & Variable Declaration:

Unlike a structure which is defined and whose variables are declared with the keyword struct, **a union is defined, and its variables are declared with the keyword** union.

```
struct Item
{
    int code;
    char name[10];
    float price;
};

void main()
{
    struct Item a;
    ...
}
```

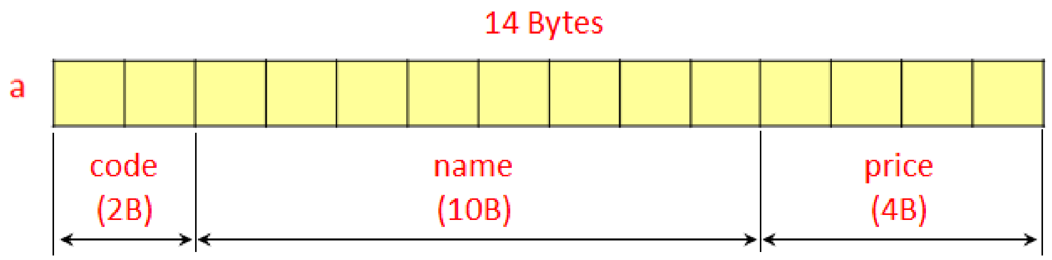
```
union Item
{
    int code;
    char name[10];
    float price;
};

void main()
{
    union Item b;
    ...
}
```

- **Difference 2 - Regarding Memory Allocation (The Key Difference):**

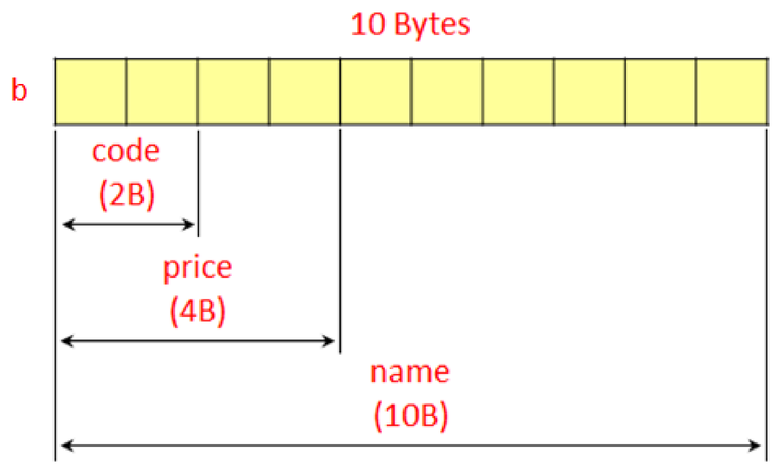
  When a union variable is declared, memory space is allocated to the variable. However, unlike structures, where each member has its own memory space, in unions, the members share the memory space.

  Therefore, the total memory space of a structure is equal to the sum of the memory space of its members, whereas the total memory space of a union is equal to the memory space of its largest member.

```
struct Item
{
    int code;
    char name[10];
    float price;
};
void main()
{
    struct Item a;
    ...
}
```



```
union Item
{
    int code;
    char name[10];
    float price;
};
void main()
{
    union Item b;
    ...
}
```



Then what is the use of unions?

Unions can be useful where any one of the members of the variable is sufficient to give complete information about the variable.

For example, suppose a "student" **is attributed by** "either a registration number or a roll number" **and a** "CGPA"**. In such condition, if we implement a student as given in** "Code 1"**, then every student requires** 38 bytes **of space. On the other hand, if we implement a student as given in** "Code 2"**, then we can save** 2 bytes **of space.**

```c
struct Student
{
    int rollNo;
    int regdNo;
    char name[30];
    float cgpa;
};
```

[Code 1]

```c
union StudentId
{
    int rollNo;
    int regdNo;
};
struct Student
{
    union StudentId id;
    char name[30];
    float cgpa;
};
```

[Code 2]

Difference 3 - Regarding Accessing the Members:

In structures, the members could be accessed in any order irrespective of which member’s value is input in which order. However, in unions, we must access the member whose value is currently stored (i.e., in LIFO order) otherwise we get the erroneous values (which is machine dependent).

```
struct Item
{
    int code;
    char name[10];
    float price;
};

void main()
{
    struct Item a;
    a.code = 12345;
    a.name = “Tea”;
    a.price = 30.5;
    printf(“%d”, a.code); /* Prints 12345 */
    printf(“%s”, a.name); /* Prints Tea */
    printf(“%f”, a.price); /* Prints 30.500000 */
}
```

```
union Item
{
    int code;
    char name[10];
    float price;
};

void main()
{
    union Item b;
    b.code = 12345;
    b.name = “Tea”;
    b.price = 30.5;
    printf(“%d”, b.code);  /* Prints erroneous value */
    printf(“%s”, b.name);  /* Prints erroneous value */
    printf(“%f”, b.price);  /* Prints 30.500000 */
}
```

Difference 4 - Regarding Union Initialization:

While initializing a structure at the place of declaration, we can initialize all its members. However, while initializing a union at the place of declaration, only its first member can be initialized.

```
struct Item
{
    int code;
    char name[10];
    float price;
};

void main()
{
    struct Item a = {123, "Tea", 30};
    ...
}
```

```
union Item
{
    int code;
    char name[10];
    float price;
};

void main()
{
    union Item b = {123};
    ...
}
```

Programming Example

```
/* PR10_7.c: Demonstration of Union. */

# include <stdio.h>
# include <conio.h>

union StudentId
{
   int rollNo;
   int regdNo;
};
struct Student
{
   union StudentId id;
   char name[30];
   float cgpa;
};
void main()
{
   int i;
   char choice;

   struct Student studs[3];
   printf("\nStudents Information\n");
   printf("====================");
```

[Cont.]

```
    for (i=0;i<3;i++)
    {
        printf("\n\nEnter Data for Student %d", i+1);
        printf("\n-------------------------\n");
        printf("Id (You can enter Roll No. [Press 1]
        or Regd. No.[Press any other key]): ");
        choice = getche();
        if(choice == '1')
        {
            printf("\nRoll No.: ");
            scanf("%d", &studs[i].id.rollNo);
        }
        else
        {
            printf("\nRegd. No.: ");
            scanf("%d", &studs[i].id.regdNo);
        }
        printf("Name: ");
        scanf("%s", studs[i].name);
        printf("CGPA: ");
        scanf("%f", &studs[i].cgpa);
```

[Cont.]

```
        printf("\nDetails of Student %d", i+1);
        printf("\n--------------------");
        if(choice == '1')
        {
            printf("\nRoll No.: %d", studs[i].id.rollNo);
        }
        else
        {
            printf("\nRegd. No.: %d", studs[i].id.regdNo);
        }
        printf("\nName: %s", studs[i].name);
        printf("\nCGPA: %.2f", studs[i].cgpa);
    }

    getch();
}
```

Output

```
Students Information
====================

Enter Data for Student 1
------------------------
Id (You can enter Roll No. [Press 1] or Regd. No.[Press any other key]): 1
Roll No.: 31
Name: Remesh
CGPA: 8.6

Details of Student 1
--------------------
Roll No.: 31
Name: Remesh
CGPA: 8.60
```

[Cont.]

```
Enter Data for Student 2
------------------------
Id (You can enter Roll No. [Press 1] or Regd. No.[Press any other key]): 2
Regd. No.: 35984
Name: Kartik
CGPA: 9.9

Details of Student 1
--------------------
Regd. No.: 35984
Name: Kartik
CGPA: 9.90

Enter Data for Student 3
------------------------
Id (You can enter Roll No. [Press 1] or Regd. No.[Press any other key]): 1
Roll No.: 32
Name: Shiv
CGPA: 9.1

Details of Student 1
--------------------
Roll No.: 32
Name: Shiv
CGPA: 9.1
```

Assignments - II

Complete the experiments given in “Lab Manual - Section 12”.

Enumerations

All About Enumerations

- **What are Enumerations?:** An enumeration (also called enum) is a user-defined datatype in C. It is used to specify a user-defined range for a user-defined data type (and hence for a variable of that data type).
- **Defining Enumerations:** An enumeration is defined as follows

```
enum enumeration_name
{
    constant_1,
    constant_2,
    …
    constant_n
};
```

e.g.,

```
enum Day
{
    Sunday,
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday
};
```

keyword

Called "enumeration constants"

NOTES:

- The above code creates a new user-defined data type "enum Day" that has the value range "Sunday, Monday, ... Saturday".

- **When an enumeration is defined, the compiler internally assigns integral digits beginning with 0 to all the enumerated constants, e.g., in the previous code, Sunday is automatically assigned to 0, Monday is automatically assigned to 1, …., Saturday is automatically assigned to 6.** **However, this automatic assignment can be overridden by explicitly assigning some integral values to the enumerated constants. In such case, the remaining enumerated constants are automatically assigned values that increases successively by 1.**

For Example:

```
enum Day
{
    Sunday,         /* Automatically assigned to 0. */
    Monday = 3,     /* Explicitly assigned to 3. */
    Tuesday,        /* Automatically assigned to 4. */
    Wednesday,      /* Automatically assigned to 5. */
    Thursday,       /* Automatically assigned to 6. */
    Friday = 27,    /* Explicitly assigned to 27. */
    Saturday        /* Automatically assigned to 28. */
};
```

- **Declaring Enumeration Variables:**

  An enumeration variable is declared as follows

```
enum enumeration_name variable_name;
```

e.g.,

```
enum Day weekStart; /* A variable of type "enum Day" is created */
enum Day weekEnd;
```

NOTE: An enumeration variable can only take one of the enumeration constants as its value. For example:

```
enum Day{Sun, Mon, Tue, Wed, Thu, Fri, Sat};
enum Day weekStart = Sun; /* OK. */
enum Day weekday1 = Mon; /* OK. */
enum Day weekday2 = 3; /* OK. Assigned to Thu. */
enum Day weekday3 = 6; /* OK. Assigned to Sat. */
enum Day weekday4 = 9; /* Illegal. */
enum Day weekday5 = Nothing; /* Illegal. */
```

- **Defining & Declaring in the Same Statement: It is allowed to define an enumeration data type and declare its variables in one statement. For example,**

```
enum Day{Sun, Mon, Tue, Wed, Thu, Fri, Sat} weekStart, weekEnd;
```

Programming Examples

- **Programming Example 1:**

```
/* PR10_8.c: Demonstration of Enumerations. */

# include <stdio.h>

enum Day{Sun, Mon, Tue, Wed, Thu, Fri, Sat};

void main()
{
    enum Day today;
    today = Wed;
    printf("Today is day %d of the week.\n", today);
}
```

Output

```
Today is day 3 of the week.
```

■ Programming Example 2:

```
/* PR10_9.c: Demonstration of Enumerations. */

# include <stdio.h>

enum month{Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct,
Nov, Dec};

void main()
{
    enum month i;
    for (i=Jan; i<=Dec; i++)
    {
       printf("%d ", i);
    }
}
```

Output

```
0 1 2 3 4 5 6 7 8 9 10 11
```

End of Chapter 10