Chapter - 9

STRUCTURES, UNIONS AND ENUMERATED DATA TYPES

LEARNING OBJECTIVES

After reading this chapter, the readers will be able to

- *understand the purpose of the user defined data types – Structures, Unions and Enumerated data types.*
- *declaration of structure data types and variables.*
- *access, initialize and process the structure members and the entire structures.*
- *use array of structures and nested structures.*
- *use structures as function arguments and return values.*
- *use pointers to access structures and their members.*
- *learn about union data types.*
- *learn the purpose of enumerated data types.*

9.0 typedef

The **typedef** keyword allows the programmer to create a new data type name for an existing data type. No new data type is created but an alternative name is given to any standard data type or derived data type. The general form of the declaration statement using the keyword **typedef** is

typedef <existing data type> <new data type>

We can redefine int to INTEGER with the statement shown below

```
typedef  int INTEGER;
```

- typedef statements can be placed anywhere in a C program as long as they precede to their first use in the program.

  The typedef identifier is traditionally coded in uppercase.

  Example 9.0

```
typedef  int  MARKS;
typedef  int  ID_NUMBER;
typedef  float HEIGHT;
typedef  char UPPER_CASE;
```

In the first two statements typedef used to give new data type name MARKS and ID_NUMBER respectively to the standard type **int**, while HEIGHT is the new name given to the data type **float** and UPPER_CASE is the new name given to the data type **char .**

The declarations of the variables with the new data type names would be as follows:

```
MARKS English, physics, chemistry;
ID_NUMBER student, employee;
HEIGHT building, tower, hill;
```

9.1. INTRODUCTION

We have seen that arrays can be used to represent logically related data items that belong to the same data type using a single name. However, array cannot be used to represent a collection of logically related items belonging to different data types using a single name. To tackle this suitably, C supports a user defined data type known as structure that can store related information of same data type or different data

types together under a single name. For example, it can be used to store the related items of different data types : employee number, name, basic pay, da, hra, gross pay and net pay together under a single name. Some other examples of structure are:

complex : real part, imaginary part
fraction : numerator, denominator
date : day, month, year
student : roll number, name, age, marks in three subjects
address : name, door number, street, locality, city, pin code

Each element in a structure is called a field. It has many of the characteristics of the variables that have been used in the programs so far. It has a type, and it exists in memory. It can be assigned values, which in turn can be accessed for selection or manipulation. The primary difference between a field and variable is field is part of a structure.

The difference between an array and a structure is that all elements in an array must be of the same type, while the elements in a structure can be of the same or different types.

9.2 STRUCTURE TYPE DECLARATION

There are two ways to declare a structure : tagged structure and type – defined structures.

Tagged structure

A tagged structure can be used to define variables, parameters, and return types. The general syntax is

```
struct TAG
{
  data type field1;
  data type field2;
  ….  …  ….;
  data type fieldn;
};
```

A tagged structure begins with the key word **struct** followed by TAG. The TAG is the identifier for the structure, and it allows us to declare structure variables, function parameters and return type.field1 ,field2, … fieldn can be variables ,arrays, pointers or other structure variables. After the closing brace, if semicolon is there no variables are declared.

Example 9.1

```
struct student
{
   long int id_no;
   char name[20];
   int age;
   float cgpa;
};
```

Type declaration with typedef

The more powerful way to declare a structure is to use a type definition typedef . The general syntax is

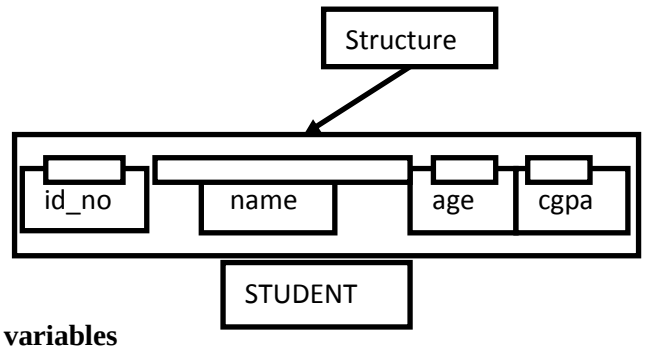
```
typedef  struct
   {
```

```
  data type field1;
 data type field2;
….   …    ….;
 data type fieldn;
}TYPE;
```

An identifier TYPE is required at the end of the block , this identifier is the type definition name.

Example 9.2

```
typedef  struct
{
   long int id_no;
   char name[20];
   int age;
   float cgpa;
}STUDENT;
```



9.3 Declaring structure variables

After a structure has been declared, we can declare variables using it. The structure variable declaration is similar to the declaration of any other data type. The following examples will demonstrate the declaration of structure type and the variables that use it.

Example 9.3

```
struct STUDENT
{
   long int id_no;
   char name[20];
   int age;
   float cgpa;
 };
struct STUDENT first, second , third;
```

```
typedef struct
{
   long int id_no;
   char name[20];
   int age;
   float cgpa;
 }STUDENT;
STUDENT first , second, third;
```

Note that in the second form of variable declaration, the keyword **struct** is not required because STUDENT is the name of the structure data type.

Remember that the fields of structure themselves are not variables. They do not occupy any memory until they are associated with structure variables such as first, second, third. When the compiler encounters declaration statement, it allocates memory space for the structure variables.

- The declaration of structure variables can be combined with the definition of tagged structure as shown in the following example.

Example 9.4

```
struct STUDENT
{
  long int id_no;
  char name[20];
  int age;
  float cgpa;
}first, second, third;
```

- If the variables are declared in the definition of tagged structure the use of tag is optional

Example9.5

```
struct
{
  long int id_no;
  char name[20];
  int age;
  float cgpa;
}first, second, third;
```

But the problem with this definition is without tag name we cannot use it for future declarations.

Initialization

We can initialize structure. The rules for structure initialization are similar to the rules for array initialization. The initializers are enclosed in braces and separated by commas. They must match with corresponding types in the structure definition.

Example 9.6

```
typedef struct
{
  long int id_no;
  int age;
  char gender;
  float cgpa;
}STUDENT;
STUDENT ex6={11000136,16,'M',7.8};
```

In this example there is an initial value for each structure member.

11000136	16	M	7.8
Id_no	age	gender	cgpa

If the number of initializers is less than the number of fields, like in arrays, the structure members will be assigned null values, zero for integers and floating point numbers, and null('\0') for characters and strings.

Example 9.7

```
STUDENT ex7= {11000136,16};
```

Since there are no initial values for gender and cgpa, they are initialized with null and zero respectively.

- Individual members cannot be initialized in the structure definition .The following initialization is not valid

```
struct student
{
    long int id_no=11000035;
    char name[20]="XXX";
    int age=16;
    float cgpa=7.2;
};
```

9.4 ACCESSING STRUCTURE MEMBERS

Any field of a structure cannot be accessed directly. It can be accessed through the structure variable by using the operator '.' Which is also known as 'dot operator' or period operator. The general syntax is

<structure variable name> **.** <fieldname>

In example 9.3, the fields of the structure id_no, name, age and cgpa through the structure variable **first** can be accessed by

```
first.id_no
first.name
first.age
first.cgpa
```

- Structure members can be used in expressions like ordinary variables or array elements.
- We can read data into and write data for structure members just as we can for individual variables. The value for the members of the structure in example 9.3 can be read from the keyboard and placed in the structure variable **first** using the following scanf() statement.

```
scanf("%d%[^\n]%d%f",&first.id_no,first.name,&first.age,&first.cgpa);
```

Note that the address operator at the beginning of the structure variable name.

- The structure member operator ' .' has highest priority along with the operators( ) and [ ], and the associatively is left to right.
- Increment and decrement operators can be applied to numeric type members

```
first.age ++;

++ first.age;   are allowed.
```

In both cases value of the field **age** is incremented by 1.

Program 9.1 program to read and display the data of a student consisting of id_number, name,age and cgpa.

```
typedef  struct
{
    long int id_no;
   char name[20];
   int age;
  float cgpa;
}STUDENT;
```

```
#include<stdio.h>
    void main()
    {
        STUDENT first;
        printf("Enter the Id number:");
        scanf("%ld",&first.id_no);
        printf("Enter the name :") ;
        flushall();
        gets(first.name);
        printf(""Enter the age:");
        scanf("%d",&first.age);
        printf("Enter the CGPA:");
        scanf("%f",&first.cgpa);
        printf("ID number=%ld\n",first.id_no);
        printf("Name=%s\n",first.name);
        printf("Age=%d\n",first.age);
        printf("CGPA=%.ef\n",first.cgpa);
     }
```

Operations on structures

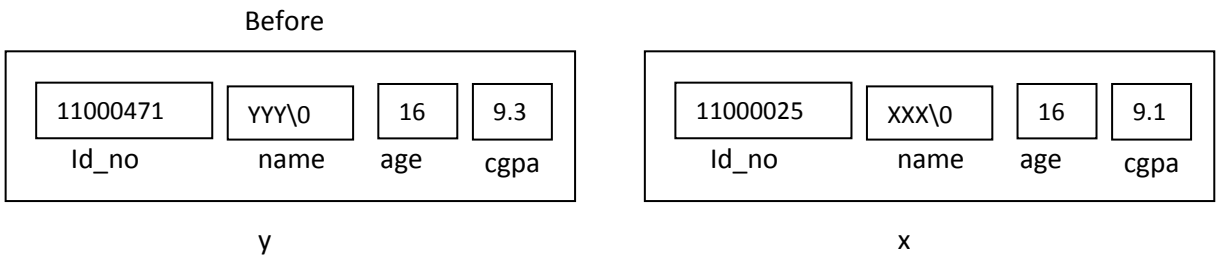
Only one operation , assignment is allowed on the structure itself. That is, a structure can only be copied to another structure of the same type using the assignment operator.

Example 9.8

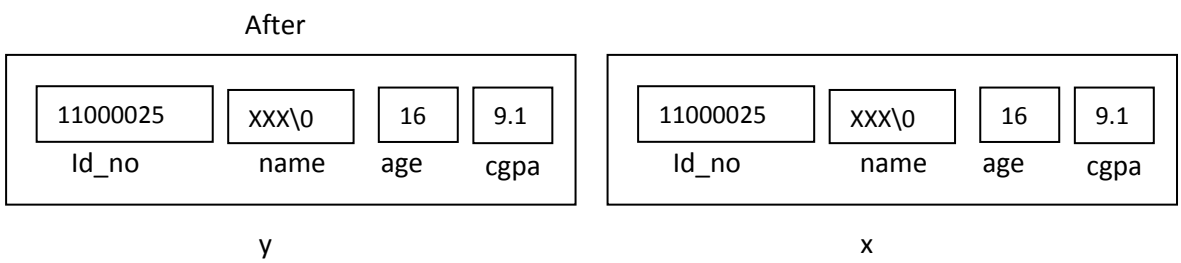
Consider the definition of the structure in program 9.7

STUDENT x={11000025,"xxx",16,9.1}, y={11000047,"yyy",16,9.3};

y=x; /* structure x is copied to the structure y*/



y=x



- C does not permit any relational or logical operations on structures. The expressions such as

x==y, x!=y, x&&y

are not allowed.

Program 9.2 Define a structure COMPLEX whose fields are the real and imaginary parts of a complex number. Write a program to read two complex numbers through keyboard and find their sum.

```
#include<stdio.h>
typedef struct
{
   float real;
   float imaginary;
}COMPLEX;
void main()
{
   COMPLEX z1,z2,z3;
   printf("Enter the real and imaginary part of first complex number:");
   scanf("%f%f",&z1.real,&z1.imaginary);
   prinf(Enter the real and imaginary parts of second complex number:");
   scanf("%f%f",&z2.real,&z2.imaginary);
   z3.real=z1.real+z2.real;
   z3.imaginary=z1.imaginary+ z2.imaginary;
   printf("Sum of given complex numbers=(%.2f,%.2f)\n",z3.real,z3.imaginary);
}
```

9.5 ARRAY OF STRUCTURES

We can create the array of structures like array of integers. For example, to store the data of a group of students, we can use an array of structures. By putting the data in an array,we can quickly and easily work with the data.

Example 9.9

```
struct STINFO
{
   int rno;
   float height;
   float weight;
};
strut STINFO stary[10];
```

stary is an array of structures of the type struct STINFO .To access the data for one student, we refer to the array name with the subscript.

stary[0].rno, stary[1].rno, stary[2].rno etc will refer the roll numbers of first, second and third students.

stary[3].height refer the height of the 4^{th} student

stary[4].weight refer the weight of the 5^{th} student

Program 9.3 Create a structure to store the following details:

Roll number ,name ,marks1,marks2,marks3 and total.

Write a program to read roll number, name and marks in three subjects of n students and find the total marks of each student. Display all the above student details sorted by total marks.

```
typedef struct
{
    int rno;
    char name[20];
    float marks1,marks2,marks3;
    float total;
}STUDENT;
#include<stdio.h>
void main()
{
   STUDENT stary[10],temp;
   int i,n,p;
   printf("Enter the number of students:");
   scanf("%d",&n);
   for(i=0;i<n;++i)
   {
      printf("Enter the roll number:");
      scanf("%d",&stary[i].rno);
      flushall();
      printf("Enter the name:");
      gets(stary[i].name);
      printf("Enter the marks in 3 subjects:");
      scanf("%f%f%f",&star[i].marks1,&stary[i].marks2,&stary[i].marks3);
   }
      /* To find the total marks of each students */

  for(i=0;i<n;++i)
          stary[i].total=stary[i].marks1+stary[i].marks2+stary[i].marks3;
      /*To sort the student details on total marks */
      for(p=1;p<n;++p)
         for(i=0;i<n-p;++i)
           if(stary[i].total<stary[i+1].total)
           {
              temp= stary[i];
               stary[i]=stary[i+1];
                stary[i+1]=temp;
           }
      printf("ROLL NO \t \t NAME \t \t MARKS1 \t MARKS2 \t MARKS3 \t TOTAL \n\n");
      for(i=0;i<n;++i)
        printf("%d\t%s\t\t%6.2f\t%6.2f\t%6.2f\t%6.2f\n",stary[i].rno,stary[i].name,
        stary[i].marks1,stary[i].marks2,stary[i].marks3,stray[i].total);
}
```

Note that when the total marks are not in order, the elements of the array, that is structures are interchanged. During the interchange one structure is copied into another structure as shown in the following program segment

```
{
     temp=stary[i];
    stray[i]=stary[i+1];
    stary[i+1]=temp;
 }
```

Structures containing arrays

Structures can have one or more arrays as members. We have already used arrays of characters inside a structure. Similarly we can use one-dimensional or multidimensional arrays of type int or float

Example 9.10:

```
struct strecord
{
    int rno;
    float sub[6];
    float total;
}stary[10];
```

In the above structure definition **sub[6]** is a field of the structure **strecord**.

The elements of the array sub[6] can be accessed using appropriate subscripts.

stary[0].sub[0], stary[0].sub[1], stary[0].sub[1], stary[0].sub[2] would refer to the marks of first student in first, second ,third subjects.

stary[4].sub[5] would refer the marks of 5^{th} student in sixth subject.

Program 9.4: Create a structure to store the following details:

Id.Number;
marks in six subjects;
total marks.

Write a program to read the Id.Number and marks in six subjects of a list of students, find the total marks of each student, and display all the above student details in tabular form.

```
typedef struct
{
    long int id_no;
    float sub[6];
    float total;
}STUDENT;
#include<stdio.h>
void main()
{
    STUDENT stary[10];
    int i,n;
    i=0;
```

```
    while(1)
    {
        printf("Enter the Id.Number at the end enter 0:");
        scanf("%ld",&stary[i].id_no);
        if(stary[i].id_no ==0)
        {
            n=i;
            break;
         }
         printf("Enter the marks in six subjects:");
         for(j=0;j<6;++j)
             scanf("%f",&stary[i].sub[j]);
         ++i;
      }
/* To find the total marks of each student */
    for(i=0;i<n;++i)
    {
        stary[i].total=0;
        for(j=0;j<6;++j)
             stary[i].total=stary[i].total+stary[i].sub[j];
    }
    printf("ID-NUMBER \t SUB1 \t SUB2 \t SUB3 \t SUB4 \t SUB5 \t SUB6 \t TOTAL\n");
    for(i=0;i<n;++i)
    {
        printf("%ld\t",stary[i].id_no);
        for(j=0;j<6;++j)
            printf("%6.2f \t",stary[i].sub[j]);
        printf("%6.2f\n",stary[i].total);
    }
}//end of main
```

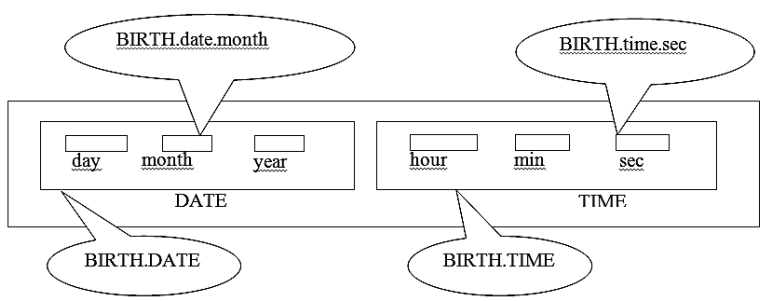
Marks in the six subjects are stored in a one dimensional array sub[6] . Marks of the $(i+1)^{th}$ student in $(j+1)^{th}$ subject is referred by stary[i].sub[j]

The following loop is used to read the marks i^{th} student and store in the array sub[6]

```
  for(j=0;j<6;++j)
    scanf("%f",stary[i].sub[j]);
```

9.6 NESTED STRUCTURES

Structure member can be another structure. When a structure includes another structure, it is a nested structure. For example, we can have a structure called birth that stores the date and time of birth of a baby. The DATE is a member structure that stores the day, month and year. The TIME is another member structure that stores the hour, minute and second . The structure design is shown below.



Although it is possible to declare a nested structure with one declaration it is not recommended. But it will be more clear and easier if each structure is declared separately and then grouped in the higher level structure

The nesting must be done from the lowest level to the most inclusive level.

The definition of the above design is given below

Example9.11:

```
typedef struct
   {
       int day;
        int month;
        int year;
    }DATE;

    typedef struct
    {
        int hour;
        int min;
        int sec;
    }TIME;
    typedef struct
    {
        DATE dob;
        TIME tob;
    }BIRTH;
   BIRTH   baby;
```

There is no limit to the number of structures that can be nested, normally we do not go beyond three

Referencing nested structures

The complete set of references for the structure baby are given below:

```
baby .dob
baby .dob.day
baby .dob.year
baby .tob
baby .tob.hour
baby .tob.min
baby .tob.sec
```

Nested structure Initialization

Initialization procedure for nested structure is same as the simple structure initialization. Each lower level structure member values are enclosed in as set of braces. Definition and initialization for the structure variable **baby** are shown below.

```
BIRTH baby={{05,06,2012},{15,30,20}};
```

First inner pair of braces contains values for day, month and year, and second inner pair of braces contains the values for hour, minute and second.

Program 9.6 Program to read the roll number and date births of a list of students and display the output sorted on date of birth.

```
typedef struct
{
    int day;
    int month;
    int year;
}DATE;
typedef struct
{
    int rno;
    DATE dob;
}STUDENT;
#include<stdio.h>
void main()
{
    STUDENT x[10],temp;
    int i,n,p;
    i=0;
    while(1)
    {
         printf("Enter the roll number, at the end enter 0:");
         scanf("%d",&x[i].rno);
         if(x[i].rno==0)
        {
            n=i;
            break;
        }
        printf("Enter the date of birth in the format dd-mm-yy:\n");
        scanf("%d-%d-%d",&x[i].dob.day,&x[i].dob.month,&x[i].dob.year);
    }


/* To sort the student records on date of birth*/

    for(p=1;p<n;++p)
    {
        for(i=0;i<n-p;++i)
        if(x[i].dob.year<x[i+1].dob.year)
        {
           temp=x[i];
           x[i]=x[i+1];
           x[i+1]=temp;
        }
       else if(x[i].dob.year==x[i+1].dob.year)
       {
           if(x[i].dob.month<x[i+1].dob.month)
           {
               temp=x[i];
               x[i]=x[i+1];
               x[i+1]=temp;
           }
           else if(x[i].dob.month==x[i+1].dob.month)
```

```
            {
                if(x[i].dob.day<x[i+1].dob.day)
                {
                   temp=x[i];
                   x[i]=x[i+1];
                   x[i+1]=temp;
                }
            }
          }
      }//end of outer for loop
printf("Rno\tDOB\n");
for(i=0;i<n;++i)
   printf("%d\t\t%d-%d-%d\n",x[i].rno,x[i].dob.day,x[i].dob.month,x[i].dob.year);
}
```

After sorting the student records are arranged in the order of youngest to oldest.

9.7 STRUTURES AND FUNCTIONS

C supports the passing of structure members and structures to functions and return them. A function can accesses the members of a structure in three ways:

1. Individual members of a structure can be passed through arguments in the function call and they are treated as ordinary variables in the called function.
2. The second method is a copy of the whole structure can be passed, that is the structure can be passed by value. Any changes to structure members within the function are not reflected in the original structure. Therefore it is necessary to return the whole structure back to the calling function.
3. The third method is to pass the address of a structure or member and the function can accesses the members through indirection operator.

The following program will demonstrate the passing of structure members to a function.

Program 9.7 Program to find the modulus of a complex number.

```
typedef struct
{
   float real;
   float imaginary;
}COMPLEX;
#include<stdio.h>
#include<math.h>
float  find_modulus(float x, float y);
void main()
{
   COMPLEX z;
   float modulus;
   printf("Enter the real and imaginary parts of the complex number:");
   scanf("%f%f",&z.real,&z.imaginary);
   modulus=find_modulus(z.real,z.imaginary);
   printf("Modulus of given complex number = %f\n",modulus);
}
```

```
float   find_modulus(float x,float y)
{
     return(sqrt(x*x+y*y));
}
```
In the above program the structure members z.real and z.imaginary are passed to the function find_modulus through arguments. They are copied to the dummy arguments x and y.

The following program demonstrate the passing of structures by value and the function returning a structure.

Program 9.8 Program to find the product of two given complex numbers, using a function which will find the product

```
typedef struct
{
    float real;
    float imaginary;
}COMPLEX;
#include<stdio.h>
COMPLEX  product(COMPLEX  z1 , COMPLEX  z2);
void main()
{
    COMPLEX z1,z2,z3;
    printf("Enter the real and imaginary parts of the first complex number:");
    scanf("%f%f",&z1.real,&z1.imaginary);
    printf("Enter the real and imaginary parts of the second complex number:");
    scanf("%f%f",&z2.real,&z2.imaginary);
    z3=product(z1,z2);
    printf("Product=(%.2f,%.2f)\n",z3.real,z3.imaginary);
}
COMPLEX  product(COMPLEX  z1 , COMPLEX  z2)
{
    COMPLEX z3;
    z3.real=z1.real*z2.real-z1.imaginary*z2.imaginary;
    z3.imaginay=z1.real*z2.imaginay+z1.imaginary*z2.real;
    return(z3);
}
```
In the above program the structure variables z1 and z2 are passed to the function by value. Within the function **product()**, the product of the complex numbers is obtained and is stored in local variable z3of the function product(). The value of z3 is returned to the called function and is assigned to the local variable z3of main().

Passing structures through pointers

Structure can be passed through pointers. When the structure is in dynamic memory, it is common to pass structure through pointer. Now let us rewrite Program9.8 by passing structures through pointers

Program 9.9 Passing structures through pointers.

```
#include<stdio.h>
typedef struct
{
   float   real;
   float imaginary;
```

```
}COMPLEX;
void  read_num(COMPLEX  *pz);
void product(COMPLEX  *pz1 , COMPLEX  *pz2 , COMPLEX  *pz3);
void print_num(COMPLEX  *pz);
void main()
{
    COMPLEX z1,z2,z3;
    printf("Enter the real and imaginary parts of first complex number:");
    read_num(&z1);
    printf("Enter the real and imaginary parts of second complex number:");
    read_num(&z2);
    product(&z1,&z2,&z3);
    print_num(&z3);
}
void  read_num(COMPLEX  *pz)
{
    scanf("%f%f",&pz→real,&pz→imaginary);
    return;
}
void product(COMPLEX  *pz1 , COMPLEX  *pz2 , COMPLEX  *pz3)
{
     pz3→real=pz1→real*pz2→real-pz1→imaginary*pz2→imaginary;
     pz3 →  imaginary=pz1 →real*pz2→imaginary+pz2→imaginary*pz2→real;
     return;
}
void print_num(COMPLEX  *pz)
{
     printf("product=(%f,%f)\n",pz→real , pz→imaginary);
     return;
}
```

The following points are to be noted in this program

1. The function **read_num()** is written to read the complex number. The pointers to the structure variables z1and z2 are passed in the first and second calls of the function.
2. In the header of the function read_num() the dummy argument pz is a pointer to the structure.
3. Pointers to the structure variables z1,z2 andz3 are passed to the function **product** which will return the pointer to the product of the two complex numbers z3.
4. The function **print_num()** is written to display a complex number.
5. To accesses the members of the structure through the pointer the indirection operator(→) is used, which has the highest precedence along with . , ( ) and [ ] operators.
6. The expressions pz → real and pz → imaginary are used to access the values of the structure members.
7. In the expressions &pz→ real and &pz → imaginary the operator '&' apply on the members **real** and **imaginary** but not on the pointer **pz**.

Size of structure

The operator **sizeof** can be used to find the size of a defined structure data type . The expression

sizeof(struct ex)

If **a** is a variable of the type **struct ex** then the expression

sizeof(a)

would also give the same answer.

9.8 UNIONS

The union is a user defined data type that allows the memory to be shared by different types of data. The union follows same syntax as the structure except the keywords **struct** and **union**, the formats are the same but there is a major difference in the storage. In structure, each member has its own storage location, where as all the members of a union share a common memory location.

Even though union contain many members in the program only one member is available at a time. The general syntax of union declaration is

```
union TAG
{
   data type member 1;
   data type member 2;
        …….
   data type member n;
};
```

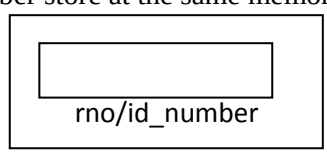
```
typedef union
  {
     data type member1;
     data type member1;
           …….
     data type member n;
  }TAG;
```

A student in a section may be referred by roll number or id.number. Therefore depending on the available data only one of them may be stored in the memory at any time. Thus a union can be defined to allocate a common memory location for them.

Example 9.12:

```
union student_id
{
    int rno;
    long int id_number;
}x;
```

Both rno and id_number store at the same memory address



student_id

- The rules for referencing a union are identical to those for structures. To reference individual fields within the union we use the **dot** operator. When a union is being referenced through a pointer, the **arrow** operator can be used.
  x.rno and x.id_number canbe used to refer the union members rno and id_number.

- Union may be initialized only with a value of the same type as the first union member. Other members can be initialized either assigning values or reading from the keyboard.

Example 9.13

```
union student_id
    {
        int rno;
        long int id_number;
```

```
    };
  union  student_id x={35};
  printf("rno=%d\n",x.rno);
  printf("id_number=%ld\n",x.id_number);
```

The output of the above program segment is

```
rno=35
id_number=0
```

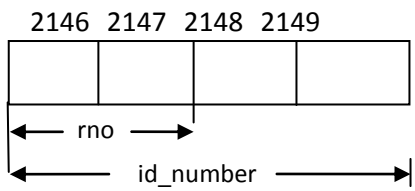
Example 9.14

```
 union ex
{
    int a ;
    float b;
};
union  ex  x={19.75};
```

is invalid declaration . Because the type of first member is int , but x is initialized with float value.

- When a union is defined, enough space is allocated to store the largest data type member in the definition. In the above example the size of rno is 2 bytes and the size of id_number is 4 bytes. Therefore 4 bytes of space is allocated regardless of what type of data is being stored.

Storage of 4 bytes



- During the accessing of union members we should make sure that we can access the member whose value is currently stored.

```
x.rno=50;
x.id_number=11000132;
printf("%d",x.rno);
```

would produce erroneous output(which is machine dependent)

9.9 ENUMERATED TYPES

The enumerated type is a user defined data type based on the standard integer type. In an enumerated type, each integer is given an identifier called an enumerated constant. We can thus use the enumerated constants as symbolic names, which make our programs more readable.

Like standard types have identifiers(int for integer),a set of values and a set of operations , enumerated type also will have a name, set of values and a set of operations defined on them.

Declaring an enumerated type

enum typename{ identifier 1, identifier 2,…,identifier n};

Where **enum** is a keyword, typename is the name given to the enumerated type and identifier 1, identifier2, ... are a set of enumerated. constants enclosed in a pair of braces.

Enumeration identifiers are assigned integer values beginning with 0. That is , the identifier 1 is assigned 0, identifier 2 is assigned 1 and so on.

Example 9.15

```
enum color {BLUE,GREEN,ORANGE,PURPLE,RED};
```

The enumerated type color has five possible values. The range of values BLUE representing the value 0,GREEN representing the value 1, ORANGE the value 2, PURPLE the value 3 and RED the value 4.

- The automatic assignments can be overridden by assigning values explicitly to the enumeration constant.

Example 9.16

enum color{ BLUE=1,GREEN,ORANGE,PURPLE,RED };

Here the identifier BLUE is assigned the value 1. The remaining identifiers are assigned values that increase successively by 1.

- The explicit assignment can be made to any one of the identifier in the list

Example 9.17

enum color{ BLUE,GREEN,ORANGE=3,PURPLE,RED };

Here the identifier ORANGE is assigned the value 3. The remaining identifiers are assigned values that increase successively by 1 in the forward direction and decrease successively by 1 in the backward direction.

Declaring variables of enumerated type

We can declare variable of enumerated data type. The general syntax is

enum type name v1,v2,v3,…vn;

The enumerated variables can have one of the values identifier 1,identifier 2, … identifier n;

Example 9.18

enum color{ BLUE,GREEN,ORANGE,PURPLE,RED };
enum color foreground, background;

9.9.1 Operations on enumerated type

We can store values in enumerated type variables and manipulate them.

- Assigning values to enumerated type.

Enumerated variables can be assigned values through assignment statement.

foreground=GREEN;
background=RED;

- The value of one enumerated variable can be assigned to another enumerated variable.
  foreground=background;

Comparing enumerated types

Enumerated types can be compared using the relational operators. Given two variables of enum type **color**, they can be compared for equal, lesser than, greater than, not equal to. We can also compare them to enumeration identifiers as shown follow:

```
if (foreground==background)
…..
if (foreground==ORANGE)
```

.....

- Since enumerated types are derived from integer types they can be used in switch statement.

Example 9.19

```
enum day {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY};
enum day birthday;
switch (birthday)
{
  case MONDAY: printf("The birthday is on MONDAY");
               break;
  case TUESDAY: printf("The birthday is on TUESDAY");
               break;
   ---
  }
```

- Enumerated variable can be used as index of for loop.

```
for (birthday=MONDAY; birthday<=SUNDAY; ++birthday)
{
    …
}
```

Input/output Operations

Since enumerated types are derived data types they cannot be read and written using the formatted input/output functions. An enumerated type must be read as an integer and written as an integer.

```
enum day holiday, birthday;
scanf("%d%d",&holiday,&birthday");
printf("%d\t%d", holiday, birthday");
```

if the input is

1 2

The output is

1 2

Note that "SUNDAY" is a string, where as SUNDAY is an enumerated type identifier.

SUMMARY

- Structure is used to pack a collection of logically related data items, possibly of different data types, having a common name.
- Each element of a structure is called a field or member.
- One difference between array and structure is that all elements of the array must be of same data type while the elements in a structure can be same or different data types.
- Unlike arrays, structure data type must be defined first and will be used later for declaring structure variables.
- Structure members cannot be initialized in the structure data type definition
- Structure variables can be initialized in the declaration. The rule for initialization is the same as that for array initialization.
- Structure members can be accessed using the **dot**(.) operator.
- Structure members can also be accessed through structure pointer. For this the **arrow** ($\rightarrow$) operator is used.
- Both the **dot(.)** and **arrow($\rightarrow$)** operators will have highest precedence along with () and [ ].
- Structure members can be used in expressions like ordinary variable names or array elements.
- The only operator allowed on entire structure is assignment.
- The individual members or the whole structure or the pointer to the structure can be passed to a function through arguments and a function can return a structure.
- An array of structures can be declared and structure members can be arrays.
- One structure can be embedded in another structure .This is called **nesting of structure**
- A union is a structure all of whose members share the same storage.
- A union can store only one of its members at a time.
- An enumerated data type is a user defined data type to improve the readability of the program. It is built on the integer data type.
- An enumerated data type consists of collection of enumeration constants which are represented by identifiers. Each identifier is given an integer value.

Suggested Reading:

1. Chapter-13 : C for Engineers and Scientists by Harry H.Cheng.
2. Chapters-12: Computer Science- A Structured Programming approach using C by B.A.Forouzan & Ritchard F.Gilberg.

EXERCISES

Review Questions

9.1 How is a structure different from array?
9.2 How is a union different from structure?
9.3 What is the purpose of tag in structure type declaration?
9.4 What is the keyword used to define a structure data type?
9.5 What is the keyword used to define union data type?
9.6 Can we declare structure variable in the structure definition?
9.7 Is the tag of a structure mandatory?
9.8 Can the structure contain members of the same data type?
9.9 Can we use pointers to access the members of a structure?
9.10 Can a function return a structure?
9.11 Can we copy one structure to another structure?
9.12 Can we compare two structures using '==' operator?
9.13 Will type definition create a new data type?
9.14 Can we declare array of structures?
9.15 An array cannot be a member of a structure(T/F)
9.16 A structure cannot be a member of another structure(T/F)
9.17 Structure variables can be initialized in the declaration(T/F)
9.18 _____ operator is used to access the members of a structure?
9.19 _____ operator is used to access the members of a structure through its pointer?
9.20 Can we initialize the members of a structure in the structure definition?
9.21 Can a structure be member of a union?
9.22 Can a union be member of a structure?
9.23 Is structure a fundamental data type?
9.24 Any member of union can be intialized in the declaration of structure variable(T?F)
9.25 What is the error in the following structure definition?

```
struct
{
   int a;
   float b;
}
```

9.26 What is the error in the following structure definition?

```
struct ex
{
    int a=5;
    float x=3.75;
    char ch='?';
};
```

9.27 Find the error in the following code?

```
struct ex
{
  int a;
  char b;
} ex x[10];
```

9.28 Find the error in the following code?

```
struct ex
```

```
{
   int a;
   char b;
};
struct ex x=10, '*';
```

9.29 Is there any error in the following code?

```
struct ex1
{
  int a;
  float b;
};
struct ex2
{
  double a;
  char ch;
}
struct ex1 p;
struct ex2 q;
```

9.30 Find the error in the following code?

```
struct ex
{
  int a;
  float b;
};
void main()
{
struct ex p = {10,3.5},
struct ex q = {8,3.5};
if (p==q)
  printf("The structures are equal\n");
else
  printf("The structures are not equal\n");
}
```

Multiple Choice Questions

1. What is the output when the following code is executed?

```
struct s
 {
   int i;
   float f;
   } x;
void main()
{
  x.i=10;
  x.f=14.5;
printf("i=%d; f=%f\n",x.i,x.f);
}
```

a) i=10, f=14.500000

b) error in the program as the structure members cannot be initialized through assignment statement
c) 10, 14.50000

2. What is the output when the following program is executed?
(Assume that the program is run on a 16 bit machine)

```
#include<stdio.h>
main()
{
  struct ex
    {
      int a;
      float b;
      char ch;
    };
printf("No of bytes allotted for the structure=%d", sizeof(struct ex));
}
```

a) No of bytes allotted for the structure = 3
b) No of bytes allotted for the structure = 7
c) No of bytes allotted for the structure = 8
d) No of bytes allotted for the structure = 12

3. Consider the following declaration

```
struct ex
{
 int rno;
 char name[20], gender;
 float height;
};
struct ex p;
```

How many bytes of space is allotted for the structure variable p if the program is run on a 16 bit machine.

a) 4 b) 24 c) 27 d) 28

ANSWERS:

9.1)a 9.2)b 9.3)c

Comprehensive Questions

9.1 Explain the structure data types in C?
9.2 Explain how structures are passed to a function through arguments?
9.3 Define a structure with the following address fields: street, block, area, country. Write a program to conduct search for an address based on any of these fields.
9.4 Create a structure to store the following employee details: employee number, designation, gender, basic pay, da, hra, deductions, gross pay, net pay. Given the employee number, designation, gender, basic pay, da rate, hra rate and deductions, write a program to calculate gross pay and net pay.
9.5 Define a structure POINT where members are the coordinates of a point in a plane. Write a function that receive the coordinates of two points from the calling function and return the distance between them. Write another function that return the coordinates of the midpoint. Write the function main which accepts the coordinates of two points from keyword and option to perform the specified operation by calling one of the two functions.

9.6 Define a structure TIME with the following fields: hours, minutes and seconds. Write a function that returns a time structure containing the time elapsed between two parameters. You must handle the case when the start time is in the previous day.

9.7 Define a structure COMPLEX whose fields are the real and imaginary parts of a complex numbers. Write a function that returns the quotient of two parameters. Call this function in main to divide one complex number by another complex number. You must handle the case when the denominator is zero.

9.8 Define a structure POINT whose members are the coordinates of a point in a plane. Write a function that accepts the structure representing a point and return an integer (1,2,3 or 4) that indicates in which quadrant the point is located.

9.9 Define a structure DATE with the fields: day, month and year. Write a function that will increment the date by one day and return the new date. If the date is the last day in the month, the month field must be incremented by one. If the month is December, the value of the year field must be changed when the day is 31. Also check for the leap year.

9.10 Define a structure DATE with the fields: day, month and year. Define another structure STUDENT whose fields are roll number, name, height and date of birth which is a structure of the type DATE. Write a program to read and display the information.

9.11 Define a structure COMPLEX whose fields are real and imaginary parts of a complex number. Write a function that receives the pointers to two complex numbers through arguments and return pointer to the sum of the two complex numbers. Write the corresponding calling function main() also.

9.12 Define a structure CRICKET whose fields are name of the player, number of innings played, total runs scored and bating average. Using CRICKET declare an array x with 50 elements and write a program to read the name, number of innings and total runs scored by each of the 50 players, and find the batting average. Display all the 50 players details sorted by batting average in tabular from.

9.13 Define a structure FRACTION whose fields are the numerator and denominator of a fraction. Write a function that will accept two fractions through arguments, return 0 if the two fractions are equal, 1 if the first fraction is less than the second, and 2 other wise.