CP MODULE 2(Part 2 of 2)

Structure, union and enumerated data type.

STRUCTURE

Consider a situation where we want to store details of students such as rollno,name and mark. For this we can declare three variables rollno of type int , name of type character array, mark of type float. These three variables are related to student. But problem here is that, we cannot group these variable into a single unit.

Array of size three cannot be used for storing these three variables because these variables (rollno,name and mark) are not of same type.

To solve this problem C has a special feature called **structure.**

- ***Structure*** *is a collection of one or more variables (elements), of same or different types, grouped together under a single name.*
- The individual structure elements are called **members**.
- The individual members can be ordinary variables, pointers, arrays, or other structures.
- Size of structure is equal to the size of all individual members in that structure.

A structure must be defined in terms of its individual members.

Syntax of structure definition/declaration (*composition of a structure*) is:

```
struct tag {
        member 1;
        member 2;
        . . . . .
        member m;
} ;
```

Here **struct** is a required keyword; tag is a name that identifies structures of this type ; and member 1 , member 2, . . . , member m are individual member declarations.

The *member names within a particular structure* must be *distinct* from one another, but a member name can be the same as the name of a variable that is defined outside of the structure.

Structure variable Declaration

- After defining/declaring the structure individual structure-type variables can be declared as follows:
    - **storage-class struct tag variable1 , variable2, . . . , variablen;**
    - Here storage -class(auto,extern,static) is an optional storage class specifier. variable1 , variable2 . . , variablen are structure variables of type tag.

- Structure declaration and structure variable declaration can be combined as follows:

```
struct tag {
        member 1;
        member 2;
```

. . . . .
member m;
} variable1 , variable2, . . . , variablen;

Example of structure student for storing details of student:-rollno,name and mark

```
struct student
{
        int rollno;
        char name[20];
        float mark;
};
```

Variables of this student structure can be declared as follows:

```
struct student s1,s2;
```

Here s1 and s2 are variables of type student structure.
It can also be written as

```
struct student
{
        int rollno;
        char name[20];
        float mark;
}s1,s2;
```

Size of structure is equal to the size of all individual members in that structure.
Here size of student structure=size of rollno(int, so 2 bytes) + size of name[20] (char, so 1 byte array size is 20, so 1* 20 =20bytes) + size of mark(float, so 4 bytes)= 2+20+4=26 bytes.

- **sizeof ()** function is used to find the size of an element
  sizeof(int) is 2 bytes

To find the size of the structure student defined above use the following printf statement
printf("%d",sizeof(student));
OUTPUT : 26

USER-DEFINED DATA TYPES (typedef)

The typedef feature allows users to define new data-types that are *equivalent to existing data types.*
In general terms, a *new data type new_type* is defined using existing data type ***type***
typedef type new_type;
Example: define a new type age which is same as int and declare variables m and f of type age:

```
typedef int  age;
age m,f;
```

This is equivalent to

int m,f;

E.g. Define height as a 100-element, floating-point array type-, men and women are 100-element, floating-point arrays.

typedef float height[1001;
height men, women;

Another way to express this is:-

typedef float height;
height men[100], women[100];

- ***Use of typedef in structure:*** The typedef can also used for *avoiding(eliminating) struct tag for declaring structure variables.*

typedef struct {
member I ;
member 2;
. . . . .
member m;
} new_type;

Variables of this structure can be declared as
new_type variable1,variable2,...variablen;
No need to write *struct tag variable1,...* for declaring variables

The first declaration defines *stud* as a user-defined data type. The second declaration defines *s1* and *s2* as structure variables of type *stud.*

Student structure definition ***Using typedef*** Variables s1 and s2 are also declared after that.	Student structure definition without typedef. Variables s1 and s2 are also declared after that.
typedef struct { int rollno; char name[20]; float mark; }**stud**; ***stud*** s1,s2;	***struct student*** *{* *int rollno;* *char name[20];* *float mark;* *};* ***struct student*** s1,s2;

NESTED STRUCTURE(Structure within a structure)

A structure can be written inside another structure. A structure variable may be defined as a member of another structure.
Example declare variable of data structure inside account structureture

```
struct account {
int  acct_no;
char acct_type;
char name[80];
float balance;
        struct date {
        int  month;
        int  day;
        int  year;
        } lastpayment ;
} oldcustomer, newcustomer;
```

This can also be written as below by creating variable of struct date inside struct account. In this situations, the declaration of the embedded structure must appear before the declaration of the outer structure.

```
struct date {
int  month;
int  day;
int  year;
} ;
struct account {
int  acct_no;
char acct_type;
char name[80];
float balance;
struct date lastpayment ;
} oldcustomer, newcustomer;
```

The second structure (account) contains another structure (date) as one of its members. So the declaration of date should precede the declaration of account.

- structure is a self-contained entity with respect to member definitions. Thus, the same member name can be used in different structures to represent different data.

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

```
char c;
};
struct second {
char a;
float b, c;
};
```

individual member names a, b and c appear in both structure declarations, but the associated data types are different. Within each structure the member names are distinct.

STRUCTURE INITIALIZATION

A structure variable, like an array, can be initialized only if its storage class is either external or static. The initial values must appear in the order in which they will be assigned to their corresponding structure members, enclosed in braces and separated by commas.

The general form of structure variable initialization is:

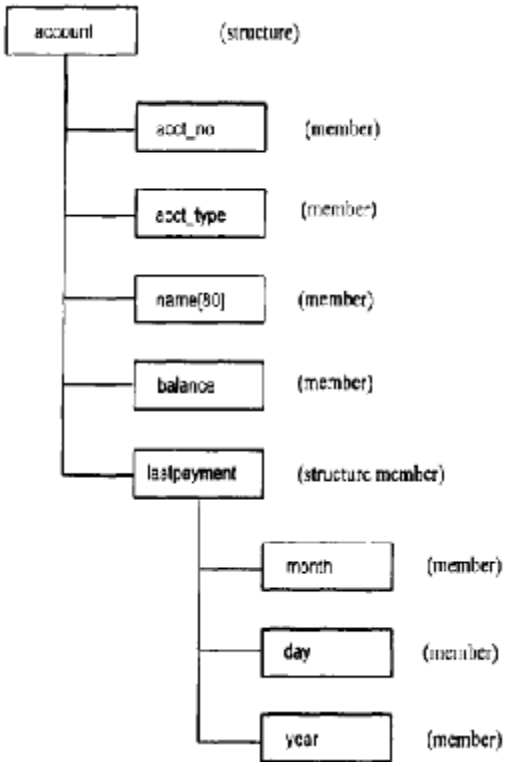
storage-class struct tag variable = {value 1, value 2, . . ., value tn);

Example:

```
struct student
{
        int rollno;
        char name[20];
        float mark;
};
static struct student s={12,"Anu",50.0};
```

Example:

```
struct date {
int month;
int day;
int year;
};
struct account (
int acct_no;
char acct_type;
char name[80];
float balance ;
struct date lastpayment;
};
static struct account customer = (12345, ' R I , "John W. Smith", 586.30, 5, 24, 1990);
```



ARRAY OF STRUCTURES

It is also possible to define an array of structures; i.e., an array in which each element is a structure.

```
struct student
{
        int rollno;
        char name[20];
        float mark;
};
struct student s[100];
```

Here s is a array of structure student of size 100. S is 100-element array of student structures.

Hence, each element of s is a separate structure of type student to store details of one student.

An array of structures can be assigned initial values just as any other array..

Initialization example:

```
struct date {
char name(80);
int month;
int day;
int year;
```

```
} ;
static struct date birthday [ ] = {"Amy", 12, 30, 73,
"Gail", 5, 13, 66,
"Marc", 7, 15, 72,
"Marla", 11, 29, 70,
"Megan', 2, 4, 77,
“Sharon”,12, 29, 63,
"Susan", 4, 12, 69};
```

Here birthday is an array of structures whose size is unspecified. The initial values will define the size of the array, and the amount of memory required to store the array.

PROCESSING A STRUCTURE(*ACCESSING ELEMENTS IN A STRUCTURE*)

Structure member can be accessed by writing

variable. member

where variable refers to-the name of a structure-type variable, and member refers to the name of a member within the structure. The period (.) that separates the variable name from the member name. This period is an operator.

If a structure member is itself a structure, then a member of the embedded structure can be accessed by writing

variable. member. submember

Similarly, if a structure member is an array, then an individual array element can be accessed by writing

variable. member[expression]

where expression is a nonnegative value that indicates the array element.

Consider the following structure:-

```
struct date {
int month;
int day;
int year;
};
struct account (
int acct_no;
char acct_type;
char name[80];
float balance ;
struct date lastpayment;
}customer;
```

If we wanted to access the customer’s account number, we would write

customer.acct_no

Expression	Meaning
++customer.balance	Increment the value of customer. Balance before accessing its value
customer.balance++	Increment the value of customer. balance after accessing its value
--customer.acctno	Decrement the value of customer. acct_no
&customer	Access the beginning address of customer
&customer.acct_no	Access the address of customer. acct_no

To access the month of the last payment, we would therefore write

customer.lastpayment.month

this value can be incremented by writing

++customer.lastpayment.month

To access third letter in customer's name, we can write

customer.name[2]

Address of third letter in customer's name can be obtained as

&customer.name[2]

In array of structures, member can be accessed using:

array[expression] .member

where *array* refers to the structure array name, and array[expression] is an individual array element (a structure variable). Therefore array[expression].member will refer to a specific member within a particular structure.

Consider the following structure:-

```
struct date {
int month;
int day;
int year;
};
struct account (
int acct_no;
char acct_type;
char name[80];
float balance ;
struct date lastpayment;
}customer[100];
```

if we wanted to access the account number for the 14th customer it can be written as

customer[13].acct_no
8th character within the 2nd customer's name can be accessed by writing
customer[1] .name[7].
The month of the 14th customer's last payment can be accessed by writing
customer[l3].lastpayment.month
++customer [13] .lastpayment. day increase the value of the day by 1.
EXAMPLE This program inputs and displays the details of a book.

```
#include<stdio.h>
struct book
{
char title[15];
char author[15];
int edition;
float price;
};
main()
{
struct book b;
printf("Enter the title of the book\n");
gets(b.title);
printf("Enter the author of the book\n");
gets(b.author);
printf("Enter the price of the book\n");
scanf("%f",&b.price);
printf("Enter the edition of the book\n");
scanf("%d",&b.edition);
printf("BOOK DETAILS\n");
printf("Title : %s\n",b.title);
printf("Author : %s\n",b.author);
printf("Edition : %d\n",b.edition);
printf("Price : %f\n",b.price);
}
```

EXAMPLE
Input and display the details of student using structure

```
#include <stdio.h>
struct student
{
   char name[12];
   int rollno;
```

```
};
struct student stud;
main()
{
    printf("Enter name" );
    scanf("%s",stud.name);
    printf("Enter rollno" );
     scanf("%d",&stud.rollno);
    printf("Name is %s",stud.name);
   printf("Rollno is%d",stud.rollno);

}
```

……

EXAMPLE

Input and display the details of n students using structure

```
#include <stdio.h>

struct student
{
   char name[12];
   int rollno;
};
struct student s[100];

main()
{
   printf("Hello World");
     int i,n;
 printf("Enter how many students?");
 scanf("%d",&n);

  for(i=0;i<n;i++)
  {
     printf("Enter name" );
   scanf(" %s",s[i].name);
   printf("Enter rollno" );

   scanf("%d",&s[i].rollno);
  }
  for(i=0;i<n;i++)
  {
     printf("\n name is %s",s[i].name);
   printf("\nrollno is %d",s[i].rollno);
  }
  }
```

UNION

Unions is like structures.

- Union contain members of same or different data types.
- The ***members*** within a union share the same storage area within the computer's memory,
    - But each *member within a structure* is assigned its own unique storage area.
- since all the union members share the same location, only one member can be accessed at a time.

Declaring a union is similar to declaring a structure. Its general form is:
union *tag*
{
member-1;
member-2;
.
.
member-m;
};
where **union** is a required keyword and the other terms have the same meaning as in a structure definition.
Once a union is defined, **union variables can then be declared** as:
union *tag variable-1 , variable-2, . . . , variable-n;*

where union is a required keyword, tag is the name that appeared in the union definition and variable-1 , variable-2, . . . , variable-n are variables of type tag.

- Comparison between Structure and Union

Structure	**Union**
Declared using **struct** keyword	Declared using **union** keyword
contain members whose individual data types may differ from one another (is a collection of data whose data type may be same or different)	contain members whose individual data types may differ from one another. (is a collection of data whose data type may be same or different)
Each **member** within a structure is assigned its own unique storage area.	All ***members*** within a union share the same storage area within the computer's memory. Union help to save memory
Total size of structure = size of *all the members* in the structure	Total size of structure = size of the **largest member** in the structure
Members can be accessed at any time	Members can be accessed soon after its value is stored(assigned). (Only the value of member last stored is found in the storage.(i.e.only one value

	can be stored in a union at a time))
E.g ***struct*** *student* *{* *int rollno;* *char name[20];* *float mark;* *};* Here sizeof(struct student) is 26 Size of rollno is 2bytes Sizeof name[20] is 1*20=20bytes Size of mark is 4bytes So size of struct student is 2+ (1*20) +4 =26 bytes	E.g. ***union*** *student* *{* *int rollno;* *char name[20];* *float mark;* *};* Here sizeof(union student) is 20 Size of rollno is 2bytes Sizeof name[20] is 1*20=**20bytes** Size of mark is 4bytes So size of union student is the size of the largest member=**20 bytes**

Write a program in C to Input and display the details of student using UNION

```
#include <stdio.h>
union student
{
   char name[12];
   int rollno;
};
union student stud;
main()
{
   printf("Enter name" );
  scanf("%s",stud.name);
  printf("Name is %s",stud.name);

   printf("Enter rollno" );
   scanf("%d",&stud.rollno);
   printf("Rollno is%d",stud.rollno);

}
```

If we write the codes as below:

```
 stud.name=”Anu”;
stud.rollno=3;
```

```
printf("Rollno is%d",stud.rollno);

printf("Name is%d",stud.name);
```

Last Value written i.e value of of rollno will overwrite the value of name.
Here only value of Rollno will be displayed correctly. Name cannot be accessed correctly.

Enumerations

- An enumeration is a data type, similar to a structure or a union.
- Its members are constants that are **written as identifiers**, but they have signed integer values.
- These constants represent values that can be assigned to corresponding enumeration variables.
- *An enumeration may be **defined***as:-
  enum identifier {value-1, value-2, ..., value-m};
  - where enum is a required keyword; identifier is a name that identifies the enumeration, and value-1, value-2, ..., value-m represent the possible values that any variable of the defined type can take. These possible values are known as **enumerators** or **enumeration constants**.
- Once the enumeration has been defined, corresponding **enumeration variables can declared** as:-
  enum identifier var-1, var-2, ..., var-n;
  - var-1, var-2,....., var-n are variables of the defined type.
  -
- E.g.
  enum colours {black, blue, cyan, green, magenta);
  enum colours foreground, background;
  - The first line defines an enumeration named colours (i.e., we define a new data type colours).
  - Any variable of colours type can take one of the possible values black, blue, cyan, green, magenta.
  - The second line declares two variables foreground and background to be enumeration variables of type colours. Thus, each variable can be assigned any one of the values black through magenta.
- The two declarations can be combined if desired, resulting in

enum colours{black, blue, cyan, green, magenta} foreground, background;

Enumeration constants are automatically assigned equivalent integer values, beginning with 0 for the first enumerator, and for each successive enumerator, the assigned value being increased

by 1.
In this example
black 0
blue 1
cyan 2
green 3
magenta 4

It is possible for the user to specify values of his choice for enumerators
enum colours {black=-1, blue, cyan, green=0, magenta);
This results in the assignment of values as follows:
black -1
blue 0
cyan 1
green 0
magenta 1

E.g.
enum colours {black, blue, cyan, green, magenta}foreground, background;
foreground = cyan;
printf("**%s**",foreground);
This will result in an error. This is because, the enumerators are not strings (Here %s is used). They are just names for integers. Internally they are treated as integers only.
Thus to print the equivalent integer value, we have to write the last statement as
printf("%d",foreground);
and this will print
2

Programming examples

1. Input the details (name, roll number and marks in 6 subjects) of n students and print the details along with the total marks scored.

```
#include<stdio.h>
typedef struct
{
char name[15];
int roll, marks[6], total;
}student;
```

```
main()
{
student s[10];
int n,i,j;
printf("How many students?");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter student name\n");
scanf("%s",s[i].name);
printf("Enter roll number\n");
scanf("%d",&s[i].roll);
printf("Enter marks in six subjects\n");
for(j=0;j<6;j++)
        scanf("%d",&s[i].marks[j]);
s[i].total=0;
for(j=0;j<6;j++)
        s[i].total+=s[i].marks[j];
}
for(i=0;i<n;i++)
{
printf("STUDENT %d DETAILS\n",i+1);
printf("Name:%s\n",s[i].name);
printf("Roll:%d\n",s[i].roll);
for(j=0;j<6;j++)
printf("Marks %d: %d\n",j+1,s[i].marks[j]);
printf("Total:%d\n",s[i].total);
}
}
```

Q.2. Input the details (name, employee ID and date of joining) of two employees.
Represent the date of joining as a nested structure. Print the details of the employee with more Experience

```
#include<stdio.h>
typedef struct
{
int date;
```

```
char month[15];
int year;
}doj;
typedef struct
{
char name[15];
int empID;
doj d;
}employee;

main()
{
employee e[2];
int i;
for(i=0;i<2;i++)
{
printf("Enter employee name\n");
scanf("%s",e[i].name);
printf("Enter employee ID\n");
scanf("%d",&e[i].empID);
printf("Enter day of joining\n");
scanf("%d",&e[i].d.date);
printf("Enter month of joining\n");
scanf("%s",e[i].d.month);
printf("Enter year of joining\n");
scanf("%d",&e[i].d.year);
}
printf("DETAILS OF THE SENIOR EMPLOYEE\n");
if(e[0].d.year<e[1].d.year)
{
printf("Name:%s\n",e[0].name);
printf("Employee ID:%d\n",e[0].empID);
printf("Date of joining:%d %s
%d\n",e[0].d.date,e[0].d.month,e[0].d.year);
}
else
{
printf("Name:%s\n",e[1].name);
printf("Employee ID:%d\n",e[1].empID);
printf("Date of joining:%s %d
```

```
%d\n",e[1].d.month,e[1].d.date,e[1].d.year);
}
}
```

Q. 3 In a company there are two type of employees – Type0 and Type1. Employees belonging to Type0 are paid on hourly basis at the rate of 750 per hour. Type1 employees are paid monthly, and their gross salary is calculated as the sum of basic pay, DA (125% of basic pay) and HRA (550). Input the number of hours for Type0 employees and basic pay of Type1 employees and print the gross salary. Implement salary details(hour and basicpay) using union. Implement employee structure to store name,department-name,type of employee, gross and variable of union salary.

```
#include<stdio.h>
typedef union
{
int hours;
float basicpay;
}salary;
typedef struct
{
char name[15],dept[15];
int type;
float gross;
salary s;
}employee;
main()
{
employee emp;
printf("Enter the name of employee");
scanf("%s",emp.name);
printf("Enter the department of employee");
scanf("%s",emp.dept);
printf("Enter the type of employee:0 or 1\n");
scanf("%d",&emp.type);
if(emp.type==0)
{
printf("Enter the number of hours\n");
scanf("%d",&emp.s.hours);
emp.gross=emp.s.hours*750;
}
else
{
```

```
printf("Enter the basic pay\n");
scanf("%f",&emp.s.basicpay);
emp.gross=emp.s.basicpay+emp.s.basicpay*1.25+550;
}
printf("Employee Details\n");
printf("Name:%s\n",emp.name);
printf("Department:%s\n",emp.dept);
printf("Salary:%f\n",emp.gross);
}
```