Structures and Unions

We know an array is used to store a collection of data of the same type. But if we want to deal with a collection of data of various type such as integer, string, float etc we use structures in C language. It is a method of packing data of different types. It is a convenient tool for handling logically related data items of bio-data people comprising of name, place, date etc. , salary details of staff comprising of name, pay da, hra etc.

Defining a structure.

In general it is defined with the syntax name **struct** as follows

```
Struct  structure_name
{
        Data type variable1;
        Data type  variable2;
        …
}
```

For example

1
```
Struct account
{
      Int accountno
      Char name[50];
      Float balance;
}customer[20]
```

Note :1here accountno, name and balance are called members of the tructure

2
```
struct date
{
      Int month;
      Int day;
      Int year;
}dateofbirth;
```

In these examples customer is a structure array of type account and dateofbirth is a structural type of date.
Within a structure members can be structures. In the following example of biodata structure date which is a structure is a member.

For example

```
struct date
{
      Int day;
      Int month;
      Int year;
```

```
}
Struct  biodata
{
     Name char[30];
     Int age ;
     Date birthdate;
}staff[30];
```

Here staff is an array of structure of type biodata

Note: we can declare other variables also of biodata type structure as follows.
Struct biodata customer[20]; , Struct biodata student; etc

Processing a structure

The members of a structure are themselves not variable. They should be linked to the structure variable to make them meaningful members. The linking is done by period (.)

If staff[] is structure array then the details of first staff say staff[1] is got by staff[1].name, staff[1].age, staff[1].birthdate.day, staff[1].birthdate.month, staff[1].birthdate.year . we can assign name, age and birthdate of staff[1] by
Staff[1].name=”Jayachandran”
staff[1].age=26
staff[1].birthdate.day=11
staff[1].birthdate.month=6
staff[1].birthdate.year=1980

If ‘employee’ is a structure variable of type biodata as mentioned above then the details of ‘employee’ is got by declaring ‘employee as biodata type by the statement
biodata employee;
The details of employee are got by employee.name, employee.age, employee.birthdate.year etc.

Note:

Structure initialisation

Like any other variable or array a structure variable can also be initalised.by using syntax static

```
Struct record
{
     Char name[30];
     Int age;
     Int weight;
}
```

Static struct record student1={“rajan”, 18, 62}

Here student1 is of record structure and the name,age and weight are initialised as “rajan”, 18 and 62 respectively.

1 Write a c program to read biodata of students showing name, place, pin, phone and grade

Solution

```
#include<stdio.h>
Main()
{
      Struct biodata
      {
            Char name[30];
            Char Place[40]
            Int pin;
            Long Int phone;
            Char grade;
      };
Struct biodata student[50];
Int n;
Prinf("\n no of students");
Scanf("%d",n);
For(i=1;i<=n;++i)
{
      Scanf("%s",student[i].name);
      Scanf("%s",student[i].place);
      Scanf("%d",student[i].pin);
      Scanf("%ld",student[i].phone);
      Scanf("%c",student[i].grade);
}
}
```

User Defiined Data Type

This is to define new data type equivalent to existing data types. Once defined a user-defined data type then new variables can be declared in terms of this new data type. For defining new data type we use the syntax typedef as flollows

typedef type new-type.
Here type refers to existing data type

For example
Ex1:

```
Typedef int integer;
```

Now integer is a new type and using this type variable, array etc can be defined as

```
Integer x;
Integer mark[100];
```

Ex2:

```
Typedef struct
{
     Int accno;
     Char name[30];
     Float balance;
}record;
```

Now record is structure type using this type declare customer, staff as record type

Record customer;
Record staff[100];

Passing structures to functions

Mainly there are two methods by which structures can be transferred to and from a function.

1 Transfer structure members individually
2 Passing structures as pointers (ie by reference)

Example 1

```
#include<stdio.h>
Typedef struct
     {
          Int accno;
          Char name[30];
          Float balance;
     }record;
Main()
{
     …..
     Record customer;
     . . . . .
     Customer.balance=adjust(customer.name,customer.accn
o,balance)
     . . . . .
}

Float adjust(char name[], int accnumber, float bal)
{
     Float x;
     . . . . .
     X=
     . . . . .
     Return(x);
}
```

Example 2

```
#include<stdio.h>
Typedef struct
      {
            Int accno;
            Char name[30];
            Float balance;
      }record;

Main()
{
      Record customer;

      Void adjust(record *cust)
      . . . . . .
      Adjust(&customer);
      Printf("\n %s\t%f",coustomer.name,customer.balance)
}

Void adjust(record *cust)
{
      Float x;
      . . . .
      Cust->balance=...
      . . . .
      Return;
}
```

In the first example structure members are passed individually where as in the second case customer is passed entirely as a pointer named cust. The values of structure members are accessed by using -> symbol like cust->.name, cust->balance etc.

Unions

Union is a concept similar to a structure with the major difference in terms of storage. In the case of structures each member has its own storage location, but a union may contain many members of different types but can handle only one at a time. Union is also defined as a structure is done but using the syntax union.

```
Union  var
{
      Int m;
      Char c;
      Float a;
}
```

Union var x;
Now x is a union containing three members m,c,a. But only one value can be stored either in x.m, x.c or x.a