I B.Tech_CSE_PPSC

STRUCTURES & UNIONS

****

Enumerated, Structure, and Union: The Type Definition (typedef), Enumerated Types, Structure, Unions, and Programming Application.

****

We have seen that arrays can be used to represent a group of data items that belong to the same type, such as int or float. However if we want to represent a collection of data items of different types using a single name, then we can not use array.

C supports a constructed data type known as structure, which is a method of packing data of different data types.

A structure is a convenient tool for handling a group of logically related data items.

Consider a book database consisting of book name, author, and number of pages and price we can define a structure to hold this information as follows.

```
struct book_bank
 {
     char title[20];
     char author[15];
     int  pages;
     float price;
 };
```

The keyword struct declares a structure to hold the details of four fields, namely title, author, pages and price. These fields are called structure elements or members. Each member may belong to a different type of data.

book_bank is the name of the structure and is called the structure tag. The tag name may be used subsequently to declare variables that have the tag's structure.

Note that the above declaration has not declared any variables. It simply describes a format.

The general format of a structure:

```
 struct tag_name
{
   data_type    member1;
   data_type    member2;
         -
         -
         -
};
```

We can declare structure variables using the tag name any where in the program for example the statement

```
struct book_bank  book1, book2;
```

declares book1 and book2 as variables of type "struct book_bank". Each one of these variables has four members. Remember that the members of a structure themselves are not variables. They do not occupy any memory until they are associated with the structure variables such as book1.

Normally structure definitions appear at the beginning of the program file, before any variables or functions are defined. They may also appear before main.

GIVING VALUES TO MEMBERS:

We can assign values to the members of a structure in a number of ways. As mentioned earlier the members themselves are not variables. They should be linked to the structure variables in order to make them meaningful members. The link between a number and a variable is established using the member operator '.' which is also known as 'dot operator' or 'period operator'.

```
book1.price
```

is the variable representing the price of book1 and can be treated like any other ordinary variables.

Assigning values to members using assignment operator.

```
book1.pages = 250;
book1.price = 28.50;
```

Assigning values to structure members using scanf() statement:

```
scanf("%s",book1.title);
scanf("%s",book1.author);
scanf("%d",&book1.pages);
scanf("%f",&book1.price);
```

STRUCTURE INITIALIZATION:

Like any other data type, a structure variable can be initialized. (However a structure must be declared as static if it is to be initialized inside a function).

```
struct student
{
  int stno;
  int m1, m2, m3;
};
struct student st1={100,50,60,70};
```

The above declaration assigns values to the members of structure variable st1 as follows

```
st1.stno=100
st1.m1=50
st1.m2=60
st1.m3=70
```

Ex 1) Program to create a structure, to read members and to print members

```
#include <stdio.h>
#include <conio.h>
struct student
{
  int stno;
  char stname[10];
  int m1,m2,m3;
};

void  main()
{
  struct student s;

  clrscr();
  printf("enter student no : ");
  scanf("%d",&s.stno);
  printf("enter student name : ");
  scanf("%s",s.stname);
  printf("enter m1 m2 m3 : ");
  scanf("%d%d%d",&s.m1,&s.m2,&s.m3);

   printf("student number = %d\n",s.stno);
   printf("student name   = %s\n",s.stname);
   printf("m1 = %d m2 = %d m3 = %d\n",s.m1,s.m2,s.m3);
   getch();
}
```

OUTPUT

```
Enter student no : 501
Enter student Name : Ramya
Enter m1 m2 m3 : 50 60 70
Student no=501
Student name = Ramya
m1=50 m2=60 m3=70
```

EX 2) Program to read a student structure and to find total, average and result

```
#include <stdio.h>
#include <conio.h>
struct student
{
  int stno;
  char stname[10];
  int m1,m2,m3;
};

void main()
{
 struct student s;
 int tot;
 float avg;
 char result[15];

 clrscr();

 printf("enter student no :");
 scanf("%d",&s.stno);
 printf("enter student name :");
 scanf("%s",s.stname);
 printf("enter m1 m2 m3 :");
 scanf("%d%d%d",&s.m1,&s.m2,&s.m3);
 tot=s.m1+s.m2+s.m3;
 avg=(float)tot/3.0;

  if (s.m1<35||s.m2<35||s.m3<35)
       strcpy(result,"fail");
  else
   strcpy(result,"Pass");

  printf("student number = %d\n",s.stno);
  printf("student name   = %s\n",s.stname);
  printf("m1 = %d m2 = %d m3 = %d\n",s.m1,s.m2,s.m3);
  printf("total marks    = %d\n",tot);
  printf("Average marks  = %f\n",avg);
  printf("result         = %s\n",result);

  getch();
}
```

OUTPUT

```
Enter student no : 501
Enter student Name : Ramya
Enter m1 m2 m3 : 50 60 70
Student no=501
Student name = Ramya
m1=50 m2=60 m3=70
total marks = 180
Average marks =  60.00
Result = Pass
```

In this program stno, stname, m1, m2, m3 are structure members and tot, avg and result are local variables declared in main(). We evaluate tot, average and results based on the structure members.

In the next program let us declare tot avg and result as structure members along with stno, stname, m1,m2, m3 and find contents of tot, avg, result.

EX 3) write a program to declare a student structure, read members of structure variable find total average and result. Print members of structure.

```
#include <stdio.h>
#include <conio.h>
struct student
{
  int stno;
  char stname[10];
  int m1,m2,m3;
  int tot;
  float avg;
  char result[10];
};

void main()
{
    struct student s;

    clrscr();

    printf("enter student no");
    scanf("%d",&s.stno);
    printf("enter student name");
    scanf("%s",s.stname);
    printf("enter m1 m2 m3");
    scanf("%d%d%d",&s.m1,&s.m2,&s.m3);

    s.tot=s.m1+s.m2+s.m3;
    s.avg=(float)s.tot/3.0;

    if (s.m1<35||s.m2<35||s.m3<35)
          strcpy(s.result,"Fail");
    else
          strcpy(s.result,"Pass");

     printf("student number = %d\n",s.stno);
     printf("student name   = %s\n",s.stname);
     printf("m1 = %d m2 = %d m3 = %d\n",s.m1,s.m2,s.m3);
     printf("Total marks    = %d\n",s.tot);
     printf("Average marks  = %f\n",s.avg);
     printf("Result         = %s\n",s.result);
     getch();
}
```

ARRAYS OF STRUCTURES

We use structures to describe the format of a number of related variables. For example, in analyzing the marks obtained by a class of students, we may use a template to describe student name, marks obtained in various subjects and then declare all the students as structure variables. In such cases, we may declare an array of structures, each element of the array representing a structure variable.

```
struct   student s[100];
```

Defines an array called s that consists of 100 elements. Each element is defined to be of the type struct student. The individual members of the structure are:

s[0].stno, s[0].stname, s[0].m1, s[0].m2, s[0].m3
s[1].stno, s[1].stname, s[1].m1, s[1].m2, s[1].m3
.
.
s[99].stno, s[99].stname, s[99].m1, s[99].m2, s[99].m3

ARRAYS WITH IN STRUCTURES:

C permits the use of arrays as structure members.

```
struct marks
 {
    int stno;
    float sub[3];
 };
struct marks student[10];
```

here the member sub contains three elements sub[0], sub[1] and sub[2]. These elements can be accessed using appropriate subscripts. For example

```
student[1].sub[0];
student[1].sub[1];
student[1].sub[2];
```

would refer to the marks obtained in the first, second and third subjects by the second student.

Ex 4) Program to create an array of structures. Use a student structure to represent a class of students.

```
#include<stdio.h>
#include <conio.h>
struct student
{
   int stno;
   char stname[10];
   int m1,m2,m3;
 };

 void main()
 {
  struct student s[60];
  clrscr();
  printf("enter no of students :");
  scanf("%d",&n);

  for (i=0;i<n;i++)
  {
        printf("enter student no :");
        scanf("%d",&s[i].stno);
        printf("enter student name :");
        scanf("%s",s[i].stname);
        printf("enter m1 m2 m3 :");
        scanf("%d%d%d",&s[i].m1,&s[i].m2,&s[i].m3);
  }
   printf("students information\n");
   for(i=0;i<n;i++)
        printf("%d %s %d %d %d\n",s[i].stno,s[i].stname,s[i].m1,s[i].m2,s[i].m3);
 getch();
 }
```

OUTPUT:

```
Enter no of students 3

Enter student no : 501
Enter student Name : Ramya
Enter m1 m2 m3 : 50 60 70

Enter student no : 502
Enter student Name : saraswathy
Enter m1 m2 m3 : 80 70 60

Enter student no : 503
Enter student Name : padmavathy
Enter m1 m2 m3 : 40 35 80

Students information
501 Ramya        50 60 70
502 saraswathy   80 70 60
503 padmavathy 40 35 80
```

Ex 5) Program to access array of structures. Read each student's information as a structure and find total, average and result for each student

```
 #include<stdio.h>
#include <conio.h>
struct student
{
  int stno;
  char stname[10];
  int m1,m2,m3;
  int tot;
  float avg;
  char result[10];
};
void main()
{
  struct student s[60];
  int i,n;
  clrscr();
  printf("enter no of students");
  scanf("%d",&n);

  for (i=0;i<n;i++)
   {
      printf("enter student no");
      scanf("%d",&s[i].stno);
      printf("enter student name");
      scanf("%s",s[i].stname);
      printf("enter m1 m2 m3");
      scanf("%d%d%d",&s[i].m1,&s[i].m2,&s[i].m3);
   }


  for (i=0;i<n;i++)
    {
       S[i].tot=s[i].m1+s[i].m2+s[i].m3;
       S[i].avg=(float)s[i].tot/3.0;
       if (s[i].m1<35||s[i].m2<35||s[i].m3<35)
         strcpy(s[i].result,"Fail");
       else
         strcpy(s[i].result,"Pass");
    }

  for(i=0;i<n;i++)
    printf("%d %s %d %d %d %d %f %s\n",s[i].stno,s[i].stname,s[i].m1,
                s[i].m2, s[i].m3, s[i].tot,s[i].avg,s[i].result);

  getch();
}
```

OUTPUT:

```
Enter no of students 3

Enter student no : 501
Enter student Name : Ramya
Enter m1 m2 m3 : 50 60 70

Enter student no : 502
Enter student Name : saraswathy
Enter m1 m2 m3 : 80 70 60

Enter student no : 503
Enter student Name : padmavathy
Enter m1 m2 m3 : 40 35 80

501 Ramya      50 60 70  180  60.00 Pass
502 saraswathy 80 70 60  210  70.00 Pass
503 padmavathy 40 34 80  154  51.33 Fail
```

STRUCTURES WITH IN STRUCTURES

Structures within structures means nesting of structures.

```
struct salary
{
    char name[10];
    char dept[10];
    int  basic;
    int  da;
    int  hra;
    int  city_allowance;
 }employee;
```

This structure defines name, department, basic and three kinds of allowances together and declares them under a sub-structure as shown below.

```
struct allow
{
     int  da;
     int  hra;
     int  city;
};

struct employee
{
    char name[20];
    char dept[10];
    float basic
    struct allow salary;
    float it,net;
};
struct employee emp;
```

The employee structure contains a member named salary which itself is a structure with three members. The members contained in the inner structure namely da, hra and city can be referred as

```
emp.name
emp.dept
emp.basic
emp.salary.da
emp.salary.hra
emp.salary.city
emp.gross
emp.it
emp.net
```

typedef statement

The function of this statement is to redefine the name of an existing variable type. For example, consider the following structure declaration

```
struct student
{
    int stno;
    char stname[10];
    int m1,m2,m3;
};
struct student s1;
```

This structure declaration can be made more easy to use when renamed using typedef as shown below:

```
struct student
{
    int stno;
    char stname[10];
    int m1,m2,m3;
};
typedef struct student stype;
stype s1;
```

typedef function renames structure student as stype. From here onwards we can use stype instead of struct student. i.e. single word instead of two words.

STRUCTURES AND FUNCTIONS

We know that the main philosophy of C language is the use of functions. Therefore it is natural that C supports the passing of structure values as arguments to functions. There are 3 methods by which the values of a structure can be transferred from one function to another function.

The first method is to pass each member of the structure as an actual argument of the function call. The actual arguments are then treated independently like ordinary variables. This is the most elementary method and becomes unmanageable and inefficient when the structure size is large.

The second method involves passing of a copy of the entire structure to the called function. Since the function is working on a copy of the structure, any changes to structure members within the function are not reflected in the original structure. It is therefore, necessary for the function to return the entire structure back to the calling function.

The third approach employs a concept called pointers to pass the structure as an argument. In this case, the address location of the structure is passed to the called function. The function can access indirectly the entire structure and work on it. This is similar to the way arrays are passed to functions. This method is more efficient as compared to the second.

EX 6)Program to pass a structure variable to a function and to access the members of structure in the function

```
#include <stdio.h>
#include <conio.h>
struct student
{
  int stno;
  char stname[20];
  int m1,m2,m3;
};

void modify(s)
struct student s;
{
   s.m1=s.m1+10;
   s.m2=s.m2+10;
   s.m3=s.m3+10;
   printf("structure within the function \n");
   printf("%d %s %d %d %d\n",s.stno,s.stname,s.m1,s.m2,s.m3);
}

void main()
{
   struct student st;

   clrscr();

   printf("enter student no, name m1,m2,m3\n");
   scanf("%d%s%d%d%d",&st.stno,st.stname,&st.m1,&st.m2,&st.m3);
   printf("structure before passing to the function\n");
   printf("%d %s %d %d %d\n",st.stno,st.stname,st.m1,st.m2,st.m3);
   modify(st);
   printf("structure after passing to the function\n");
   printf("%d %s %d %d %d\n",st.stno,st.stname,st.m1,st.m2,st.m3);
   getch();
}
```

Output

```
Enter student no name m1 m2 m3 : 501 rama 50 60 70
Before function call : 501 rama 50 60 70
Inside function      : 501 rama 60 70 80
After function call  : 501 rama 50 60 70
```

EX 7) Program to modify a structure using a function and return modified structure variable to main function

```
#include <stdio.h>
#include <conio.h>
struct student
{
  int stno;
  char stname[20];
  int m1,m2,m3;
};
typedef struct student stype;

stype modify(s)
stype s;
{
   s.m1=s.m1+10;
   s.m2=s.m2+10;
   s.m3=s.m3+10;
   printf("structure within the function \n");
   printf("%d %s %d %d %d\n",s.stno,s.stname,s.m1,s.m2,s.m3);
   return(s);
}

void main()
{
    stype st, mst;

    clrscr();

    printf("enter student no, name m1,m2,m3\n");
    scanf("%d%s%d%d%d",&st.stno,st.stname,&st.m1,&st.m2,&st.m3);
    printf("structure before passing to the function\n");
    printf("%d %s %d %d %d\n",st.stno,st.stname,st.m1,st.m2,st.m3);
    mst=modify(st);
    printf("modified structure after passing to the function\n");
    printf("%d %s %d %d %d\n",mst.stno,mst.stname,mst.m1,mst.m2,mst.m3);
    getch();

}
```

output

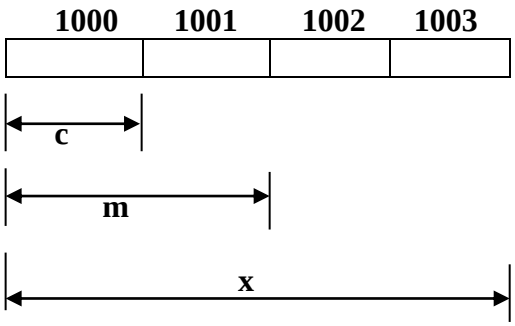
```
Enter student no name m1 m2 m3 : 501 rama 50 60 70
Before function call : 501 rama 50 60 70
Inside function      : 501 rama 60 70 80
After function call  : 501 rama 60 70 80
```

UNIONS

Unions are a concept borrowed from structures and therefore follow the same syntax as structures. However there is a major distinction between them in terms of storage. In structure each member has its own storage location, where as all the members of a union use the same location. This implies that, although a union may contain many members of different types, it can handle only one member at a time. Like structures a union can be declared using the key word union as follows.

```
union   item
 {
        int m;
        float x;
        char c;
 }code;
```

This declares a variable code of type union item. The union contains three members each with a different data type. We can use only one of them at a time. This is due to the fact that only one location is allocated for a union variable, irrespective of its size.



Storage of 4 bytes.

The compiler allocates a piece of storage that is large enough to hold the largest variable type in the union. In the declaration above, the member x requires 4 bytes, which is the largest among the members. Fig shows how all the three variables share the same address.

```
#include <stdio.h>
#include <conio.h>
union data
{
    char ch;
    int num;
    float avg;
};
void main( )
{
  union data d;
  int n;
  clrscr();
  n=sizeof(d);
  printf("size of the union = %d bytes",n);
  getch()
}
```

size of the union = 4 bytes

```
#include <stdio.h>
#include <conio.h>
union data
{
    char ch;
    int num;
    float avg;
};
void main( )
{
    union data d;

    clrscr();

    d.ch='a';
    printf("d.ch = %c\n",d.ch);

    d.num=250;
    printf("d.num = %d\n",d.num);

    d.avg=56.78;
    printf("d.avg = %f\n",d.avg);

    printf("d.ch = %c\n",d.ch);
    printf("d.num = %d\n",d.num);
    printf("d.avg = %f\n",d.avg);

    getch();
}
```

OUTPUT

```
d.ch=a
d.num=250
d.avg=56.78
d.ch=xxxxxxxxxxx
d.num=xxxxxxxxx
d.avg=56.78
```

```
/*-------------------------------------------------------------------------------
    Ex 8) Write a program to find tomorrow's date using today's date
     For example        Today          Tomorrow
                        12-04-2002     13-04-2002
                        30-04-2002     01-05-2002
                        31-12-2002     01-01-2003
-------------------------------------------------------------------------------*/


#include<stdio.h>
#include <conio.h>
struct date
{
  int day;
  int month;
  int year;
};
                /* Function to find whether the given year is leap year or not  */
 int leapyear(int y)
 {
      int rem4,rem100,rem400;

      rem4   = y%4;
      rem100 = y%100;
      rem400 = y%400;

      if (((rem4==0) &&(rem100 !=0)) || (rem400==0))
      {
           printf("%d is a leapyear \n",y);
           return(29);
      }
     else
        {
           printf("%d is not a leap year\n",y);
           return(28);
        }
  }
```

```
void main()
{
    struct date today, tomorrow;
    static int mdays[12]={31,28,31,30,31,30,31,31,30,31,30,31};
    int maxdays;

     clrscr();

     printf("enter  today dd mm yyyy:");
     scanf("%d%d%d",&today.day,&today.month,&today.year);

     if (today.month==2)
          maxdays=leapyear(today.year);
     else
         maxdays=mdays[today.month-1];

      if (today.day<maxdays)
       {
            tomorrow.day=today.day+1;
            tomorrow.month=today.month;
            tomorrow.year=today.year;
        }
        else
           {
                tomorrow.day=1;
                tomorrow.month=today.month+1;
                tomorrow.year=today.year;

                if (tomorrow.month>12)
                {
                  tomorrow.month=1;   tomorrow.year++;
                }
         }

   printf("tomorrow = %d-%d-%d\n",tomorrow.day,tomorrow.month,tomorrow.year);

   getch();
}
```

OUTPUT

```
enter  today dd mm yyyy: 30 12 2010
tomorrow = 31-12-2010

enter today dd mm yyyy: 31 12 2010
tomorrow = 01-01-2011
```

POINTERS AND STRUCTURES

We can pass a structure to a function as a pointer to a function. We use Arrow operator to access members of structure using a pointer.

Ex 9)Program to pass a student structure as a pointer to a function to find total , avg, result

```
#include <stdio.h>
#include <conio.h>
struct student
{
    int stno;
    char stname[20];
    int m1,m2,m3;
    int tot;
    float avg;
    char result[10];
 };
 typedef struct student stype;

void process(s)
 stype *s;
 {
    s->tot= s->m1 + s->m2 + s->m3;
    s->avg = (float)s->tot / 3;
    if ((s->m1 <40)|| (s->m2<40)|| (s->m3<40))
          strcpy (s->result,"fail");
    else
          strcpy (s->result,"pass");
 }
void main()
 {
   stype st;

  clrscr();

  printf("enter stno name, m1,m2 m3");
  scanf("%d%s%d%d%d",&st.stno,st.stname,&st.m1,&st.m2,&st.m3);
   process(&st);

   printf("Total   = %d\n",st.tot);
   printf("Average = %f\n",st.avg);
   printf("Result  = %s\n",st.result);
   getch();
 }
```

Enumerated Data Types

Enumeration (enum) is a user-defined data type (same as structure). It consists of various elements of that type. There is no such specific use of enum, we use it just to make our codes neat and more readable. We can write C programs without using enumerations also.

For example, Summer, Spring, Winter and Autumn are the names of four seasons. Thus, we can say that these are of types season. Therefore, this becomes an enumeration with name season and Summer, Spring, Winter and Autumn as its elements.

So, you are clear with the basic idea of enum. Now let's see how to define it.

Defining an Enum

An enum is defined in the same way as structure with the keyword struct replaced by the keyword enum and the elements separated by 'comma' as follows.

```
enum enum_name
{
    element1,
    element2,
    element3,
    element4,
};
```

Now let's define an enum of the above example of seasons.

```
enum season {summer, spring, winter, autumn };
```

Here, we have defined an enum with name 'season' and 'Summer, Spring, Winter and Autumn' as its elements.

Declaration of Enum Variable
We also declare an enum variable in the same way as that of structures. We create an enum variable as follows.

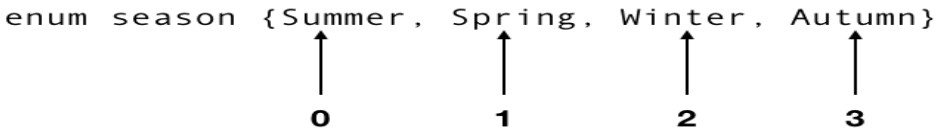
```
enum season {summer, spring, winter, autumn };
void main()
{
  enum season s;
}
```

So, here 's' is the variable of the enum named season. This variable will represent a season. We can also declare an enum variable as follows.

```
enum season
{
    summer,  spring,  winter,  autumn
}s;
```

Values of the Members of Enum :

All the elements of an enum have a value. By default, the value of the first element is 0, that of the second element is 1 and so on.




```
#include <stdio.h>
enum season { summer, spring, winter, autumn};
void main()
{
   enum season s;
   s=spring;
   printf("%d\n",s);
  getch();
}
```

Here, first we defined an enum named 'season' and declared its variable 's' in the main function as we have seen before. The values of Summer, Spring, Winter and Autumn are 0, 1, 2 and 3 respectively. So, by writing s=spring, we assigned a value '1' to the variable 's' since the value of 'Spring' is 1.

We can also change the default value and assign any value of our choice to an element of enum. Once we change the default value of any enum element, then the values of all the elements after it will also be changed accordingly. An example will make this point clearer.

```
#include <stdio.h>
enum days { sun, mon, tue, wed, thu, fri, sat};
void main()
{
   enum days day;
   day=thu;
   printf("%d\n",day);
  getch();
}
```

The default value of 'sun' will be 0, 'mon' will be 1, 'tue' will be 2 and so on. In the above example, we defined the **value of tue** as **5**. So the values of 'wed', 'thurs', 'fri' and 'sat' will become 6, 7, 8 and 9 respectively. There will be no effect on the values of sun and mon which will remain 0 and 1 respectively. Thus the value of thurs i.e. 7 will get printed.
Let's see one more example of enum.

```
#include <stdio.h>
enum days { sun, mon, tue, wed, thu, fri, sat};
void main()
{
   enum days day;
   day=thu;
   printf("%d\n",day+2);
  getch();
}
```

In this example, the value of 'thu' i.e. 4 is assigned to the variable day. Since we are printing 'day+2' i.e. 6 (=4+2), so the output will be 6.

```
#include <stdio.h>
#include <conio.h>
void main()
{
 enum months {JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC};
 enum months datemonth;
 int n,days;

 clrscr();
 printf("enter  month number");
 scanf("%d",&n);
 datemonth=n-1;

 switch(datemonth)
 {
  case JAN:  days=31;
             break;
  case FEB:  days=28;
             break;
  case MAR: days=31;
             break;
  case APR:  days=30;
             break;
  case MAY: days=31;
             break;
  case JUN:  days=30;
             break;
  case JUL: days=31;
             break;
  case AUG: days=31;
             break;
  case SEP: days=30;
             break;
  case OCT: days=31;
             break;
  case NOV: days=30;
             break;
  case DEC: days=31;
             break;
 }
  printf("days =%d\n",days);
  getch();
}
```

OUTPUT

```
enter month number 1
days=31
enter month number 2
days=28
enter month number 6
days=30
```

```
//-----------------------------------------------
// enumerated data type
//-----------------------------------------------
#include <stdio.h>
#include <conio.h>
void main()
{
	enum week { Sun, Mon, Tue, Wed, Thu, Fri, Sat};

	clrscr();

	printf("Sun = %d\n", Sun);
	printf("Mon = %d\n", Mon);
	printf("Tue = %d\n", Tue);
	printf("Wed = %d\n", Wed);
	printf("Thu = %d\n", Thu);
	printf("Fri = %d\n", Fri);
	printf("Sat = %d\n", Sat);

	getch();

}
```

OUTPUT

```
Sun=0
Mon=1
Tue=2
Wed=3
Thu=4
Fri=5
Sat=6
```