typedef

- By using **typedef** we can redefine the name of an existing variable type.
- The main advantage of **typedef** is that we can create meaningful data type names for increasing the readability of the program.

```
# define H 60
main()
{
        typedef int hours;
        hours h;
        printf("Enter number of hours:");
        scanf("%d",&h);
        printf("\nNumber of minutes=%d",h*H);
        printf("\nNumber of seconds=%d",h*H*H);
}
```

1

```
main()
{
        typedef struct
        {
                char name[20];
                int age;
                float salary;
        } info;
        info employee = {"abc",30,12000.50};
        printf("\nName\tAge\tSalary\n\n");
        printf("%s\t%d\t%f",employee.name,employee.age,emp
loyee.salary);
}
```

2

Enumerated data type

- Through the keyword **enum**, one can create its own data type and define what values the variables of these data types can hold.

```
main()
{
        enum days { Mon, Tue, Wed, Thu, Fri, Sat, Sun };
        printf("\nMon=%d",Mon); //0
        printf("\nSun=%d",Sun); //6
}
```

• The statement **enum days** declares the new data type **days** and specifies its possible values as **Mon, Tue, Wed, Thu, Fri, Sat, Sun**. These values are called **enumerators**.

• Internally the compiler treats the enumerators as integers. Each value on the list of permissible values corresponds to an integer, starting with 0. Thus, **Mon is stored as 0**, **Tue as 1** and so on.

3

```
main()
{
        enum days {Mon=1,Tue,Wed,Thu,Fri,Sat,Sun=10};
        printf("\nMon=%d",Mon); //1
        printf("\nSun=%d",Sun); //10
}
```

```
main()
{
        enum days {Mon=100,Tue,Wed,Thu,Fri,Sat,Sun=100};
        printf("\nMon=%d",Mon); //100
        printf("\nSun=%d",Sun); //100
}
```

4

```
main()
{
        enum capital
        {
                Mumbai,Delhi,Lucknow
        };
        struct state
        {
                char name[20];
                enum capital c;
        } s;
        strcpy(s.name,"UP");
        s.c=Lucknow;
        printf("\nState =%s",s.name);     // UP
        printf("\nCapital =%d",s.c);      // 2

}
```

5

//Example using enum and typedef

```
main()
{
        enum capital
        {
                Mumbai,Delhi,Lucknow
        };
        typedef struct
        {
                char name[20];
                enum capital c;
        } state;
        state s;
        strcpy(s.name,"UP");
        s.c=Lucknow;
        printf("\nState =%s",s.name); // UP
        printf("\nCapital =%d",s.c); //2

}
```

6

Limitation of **enum** variables

- There is no way to use the enumerated values directly in I/O functions like printf() and scanf().
- Of course we can write a function to print the correct enumerated values using **switch** statement.

7

```
main()
{
        enum capital
        {
                Mumbai,Delhi,Lucknow
        };
        typedef struct
        {
                char name[20];
                enum capital c;
        } state;
        state s;
        strcpy(s.name,"UP");
        s.c=Lucknow;
        printf("\nCapital =%d",s.c); //2
```

8

```
switch(s.c)
        {
        case 0:
                printf("\nCapital =Mumbai");
                break;
        case 1:
                printf("\nCapital =Delhi");
                break;
        case 2:
                printf("\nCapital =Lucknow");
                break;
        }
        printf("\nState =%s",s.name); // UP
}
```

9

Unions

- Unions are also derived data types like structures.
- Both unions and structures are used to group a number of different variables together.
- But, there is a major distinction between them in terms of storage.
- In structures, each member has its own storage location, whereas all the members of a union share 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.
- The compiler allocates a piece of storage that is large enough to hold the largest variable type in the union.

10

- That is, a union offers a way for a section of memory to be treated as a variable of one type on one occasion, and as a different variable of a different type on another occasion.
- Union is a frequent requirement while interacting with hardware. i.e. sometimes we are required to access all bytes simultaneously and sometimes each byte individually.

11

```
main()
{
        struct data
        {
                char c; int i; float f;
        }a;
        printf("\nsizeof(a)=%d bytes\n",sizeof(a));//9(vc)
}

main()
{
        union data
        {
                char c; int i; float f;
        }a;
        printf("\nsizeof(a)=%d bytes\n",sizeof(a)); //4
}
```

12

- During accessing a union member, we should make sure that we are accessing the member whose value is currently stored.
- A union creates a storage location that can be used by **any one** of its members **at a time**.
- When a different member is assigned a new value, the new value supercedes the previous member's value.

13

```
main()
{
        union data
        {
                char c; int i; float f;
        }a;
        printf("\nsizeof(a)=%d bytes\n",sizeof(a));

        a.i=33;
        printf("\na.i=%d",a.i);
        a.f=23.4;
        printf("\na.i=%d",a.i);
        printf("\na.f=%f",a.f);
}
```

Output:

```
sizeof(a)=4 bytes
a.i=33
a.i=1102787379
a.f=23.400000Press any key to continue
```

14

- Just as one structure can be nested within another, a union too can be nested in another union.
- Not only that, there can be a union in a structure, or a structure in a union.

15