Initialization of Structure members:

A structure variable, like an array can be initialized in two ways:

I. struct student

{

char name [80]; int roll_no; float marks ;

} s1={“Saravana”,34,469};

II. struct student

{

char name [80];

int roll_no;

float marks ;

} s1;

struct student s1= {“Saravana”,34,469};

[OR]

s1.name={“Saravana”};

s1.roll_no=34; s1.marks=500;

It is also possible to define an array of structure, that is an array in which each element is a structure. The procedure is shown in the following example: struct student

{

char name [80];

int roll_no ;

float marks ;

} st [100];

In this declaration st is a 100 element array of structures. It means each element of st represents an individual student record.

Accessing the Structure Members

The members of a structure are usually processed individually, as separate entities.

Therefore, we must be able to access the individual structure members. A structure member can

be accessed by writing:

structure_variable.member_name //[normal variable]

Eg:

struct student

{

char name [80]; int roll_no ; float marks ; }st;

e.g. if we want to get the detail of a member of a structure then we can write as scanf("%s",st.name); or scanf("%d", &st.roll_no) and so on.

if we want to print the detail of a member of a structure then we can write as

printf("%s",st.name); or printf("%d", st.roll_no) and so on.

The use of the period operator can be extended to arrays of structure by writing:

array [expression].member_name /(array variable]

Eg:

struct student

{

char name [80];

int roll_no ;

float marks ; }st[10];

```
#include"conio.h"

struct student

{

char name[25],grade;

int reg_no[15];

int s1,s2,s3,s4,s5,total;

float avg;
}sa[20];


void main()

{

int i,n,total; float avg;

printf("\n Enter the count of Students need marksheet::");
scanf("%d",&n);

printf("\n Enter the name of students:,reg_no, 5 sub marks::");
for(i=0;i<n;i++)

{
```

```
char grade; float marks; struct date d1; }st[100]; main()

{

int i,n;

void readinput (int i); void writeoutput(int i); printf(“Student system”);

printf(“How many students are there ?”); scanf(“%d” &n);

for (i=□ ; i<n; ++i){ readinput (i);

if( st[i].marks <80) st[i].grade=’A’; else st[i].grade='A'+;

}

for (i=□ ; i<n; ++i) writeoutput(i);

}

void readinput (int i)

{

printf(“\n student no % \n”, i+1); printf(“Name:”);

scanf(“%[^\n]”, st[i].name); printf(“Address:”); scanf(“%[^\n]",
st[i].address);

printf(“Roll number”);
```

void adj(record stud) /*function definition */

{

stud.name=”Tanishq”; stud.roll_no=3; stud.marks=98.0; return;

}

8 UNIONS

Union, like structures, contain members whose individual data types may differ from one another. However, the members that compose a union all share the same storage area within the computer’s memory

, whereas each member within a structure is assigned its own unique storage area. Thus, unions are used to conserve memory.

In general terms, the composition of a union may be defined as union tag{

member1; member 2;

- - -

member m };

Where union is a required keyword and the other terms have the same meaning as in a structure definition. Individual union vari-ables can then be declared as storage-class union tag variable1, variable2, -----, variable