```
{
   struct student s1={”sona”,16,101 };
   struct student s2={”rupa”,17,102 };
display(s1);
display(s2);
}
display(struct student s)
{
printf(“\n name=%s, \n age=%d ,\n roll=%d”, s.name, s.age, s.roll);
}
```

```
Output: name=sona
        roll=16
```

Lecture Note: 26

UNION

Union is derived data type contains collection of different data type or dissimilar elements. All definition declaration of union variable and accessing member is similar to structure, but instead of keyword struct the keyword union is used, the main difference between union and structure is

Each member of structure occupy the memory location, but in the unions members share memory. Union is used for saving memory and concept is useful when it is not necessary to use all members of union at a time.

Where union offers a memory treated as variable of one type on one occasion where (struct), it read number of different variables stored at different place of memory.

Syntax of union:

```
union student

{

datatype member1;

datatype member2;

};
```

Like structure variable, union variable can be declared with definition or separately such as

```
union union name

{

Datatype member1;

}var1;
```

Example:-   union student s;

Union members can also be accessed by the dot operator with union variable and if we have pointer to union then member can be accessed by using (arrow) operator as with structure.

Example:- struct student

```
struct student
{
int i;
char  ch[10];
};struct student s;
```

Here datatype/member structure occupy 12 byte of location is memory, where as in the union side it occupy only 10 byte.

Lecture Note:27

Nested of Union

When one union is inside the another union it is called nested of union.

Example:-

```
union a
{
int i;
int age;
};
union b
```

```
{
char name[10];
union a aa;
}; union b bb;
```

There can also be union inside structure or structure in union.

Example:-

```
  void  main()
    {
   struct a
  {
int i;
char ch[20];
};
struct  b
{
int i;
char d[10];
};
union z
{
struct a a1;
struct b b1;
}; union z z1;
z1.b1.j=20;
z1.a1.i=10;
z1.a1.ch[10]= " i";
z1.b1.d[0]="j ";
printf(" ");
```

Dynamic memory Allocation

The process of allocating memory at the time of execution or at the runtime, is called dynamic memory location.

Two types of problem may occur in static memory allocation.

If number of values to be stored is less than the size of memory, there would be wastage of memory.

If we would want to store more values by increase in size during the execution on assigned size then it fails.

Allocation and release of memory space can be done with the help of some library function called dynamic memory allocation function. These library function are called as **dynamic memory allocation function.** These library function prototype are found in the header file, "alloc.h" where it has defined.

Function take memory from memory area is called heap and release when not required.

Pointer has important role in the dynamic memory allocation to allocate memory.

malloc():