6.0 Introduction

A simple variable can store one value at a time. An array can store a number of variables of the same type. For example, marks obtained by a student in five subjects can be stored in an array marks[5]. marks[0], marks[1], etc, will store student information like the marks obtained in various subjects. Suppose we want to store student information in array name, htno, address, marks obtained. We cannot store this information in an array because name, htno, address are of character type and marks obtained is of integer type. So arrays are used to store homogeneous data. To store heterogeneous data elements in a single group, C language provides a facility called the ***structure***.

6.1 Definition of structure

A structure is a collection of variables of different types that are logically grouped together and referred under a single name.

- ✓ Structure is a user-defined data type.
- ✓ User-defined data type also called as derived data type why because we derived it from the primary/basic data types.

In C language, a structure can be defined as follows:

```
struct structure_name
{
        data_type member_variable1;
        data_type member_variable2;
                … …
        data_type member_variableN;
};
```

The variables declared inside the structure are known as *members* of the structure.

For Example

Struct student $\longrightarrow$ Datatype

```
{
    char name[20];
    char ht_no[10];
    char gender;
    float marks;
    long int phone_no;
};
```

Points remember

- ✓ The members of the structure may be any of the common data type, pointers, arrays or even the other structures.
- ✓ Member names with in a structure must be different.
- ✓ Structure definition starts with the open brace(**{**) and ends with closing brace(**}**) followed by a semicolon.

- ✓ The compiler does not reserve memory any memory when structure is defined. We have to define the variable to store its information.

6.2 Structure variable

As stated earlier, the compiler does not reserve any memory when structure is defined. To store members of structures in memory, we have to define the structure variables. The structure variables may be declared in the following ways:

- ✓ In the structure declaration
- ✓ Using the structure tag // like variable declaration.

In the structure declaration

The structure variable can be declared after the closing brace. Following example shows this.

```
struct date
{
      int day;
      int month;
      int year;
}dob, doj;
```

Here date is the structure tag, while *dob* and *doj* are variables type date.

Using the structure tag

The variables of structure may also be declared separately by using the structure tag as shown below:

```
struct date
{
      int dat;
      int month;
      int year;
};

struct date dob,doj;
```

6.3 Initialization of structure

We can initialize the values to the members of the structure as:

```
struct structure_name structure_variable={value1, value2, … , valueN};
```

- ✓ There is a one-to-one correspondence between the members and their initializing values.
- ✓ C does not allow the initialization of individual structure members within the structure definition template.

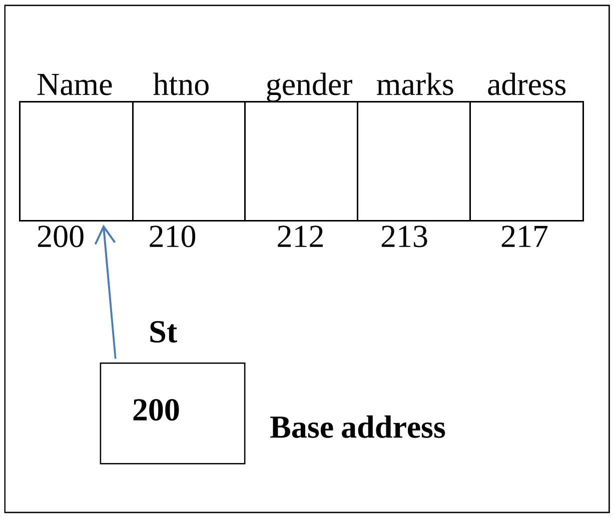
```
struct date
{
      int dat;
      int month;
```

```
    int year;
};
```

struct date dob={25,12,1998}; **or** ={.month=9,.year=2019,.dat=19};

Following example shows how to initialize structure in the above method.

```
#include<stdio.h>
struct student
{
        char name[30];
        int htno;
        char gender;
        int marks;
        char address[30];
};
int main()
{
        struct student
        st={"XYZ",581,'M',79.5,"Nuzvid"};
        printf("Student Name:%s\n",st.name);
        printf("RollNo:%d\n",st.htno);
        printf("Gender:%c\n",st.gender);
        printf("Marks obtained:%d\n",st.marks);
        printf("Address:%s\n",st.address);
        return 0;
}
```



Output

```
Student Name: XYZ
Roll No: 581
Gender: M
Marks obtained:79.5
Address: Nuzvid
```

6.4 Accessing members of structure

For accessing any member of the structure, we use dot (.) operator or *period* operator. We can also use → operator also. when we are accessing structure members through pointers then we use arrow operator.

Syntax:

structure_variable**.**membername

Here, *structure_variable* refers to the name of a *structure* type variable and *member* refers to the name of a member within the structure.

Following example shows how to access structure members

```
#include<stdio.h>
```

```
struct date
{
      int day;
      int month;
      int year;
};

int main()
{
          struct date dob;
          printf("Enter your birthday details\n");
          printf("Enter the day:");
          scanf("%d",&dob.day);
          printf("Enter the month:");
          scanf("%d",&dob.month);
          printf("Enter the year:");
          scanf("%d",&dob.year);
          printf("Your date of birth is:");
          printf("%d-%d-%d\n",dob.day,dob.month,dob.year); return
          0;
}
```

Output

```
Enter your birthday details
Enter the day:12
Enter the month:7
Enter the year:1998
Your date of birth is: 12-7-1998
```

Note: structure is also define local and global..

Local structure	Global structure

```
#include<stdio.h>
void  check();
void main()
{
          struct student
          {
                    int regid;
                    char name[100];
                    float cgpa;
          }st;
}
void check()
{
          st.name="chikku";
          printf("%s",st.name);
```
	```
	#include<stdio.h>
	void check();
	struct student
	          {
	                    int regid;
	                    char name[100];
	                    float cgpa;
	          }st;
	void main()
	{
	          check();
	}
	void check()
	{
	          st.cgpa=9.9;
	```

}
Output:-
ERROR
 	
Note:-It will give error because here structure is local.so we can access it only in main function.
	printf("%.1f",st.cgpa);
	}
	Output:-
	9.9

6.5 Nested Structure

We can take any structure as a member of structure. i.e. called structure with in structure or nested structure.
For example:

```
struct
{
        member 1;
        member 2;
        … …
        … …
        struct
        {
                member 1;
                member 2;
        }s_var2;
        … …
        member m;
}s_var1;
```

For accessing the member 1 of inner structure we write as:

```
        s_var1.s_var2.member1;
```

Following example illustrates the nested structure

```
#include<stdio.h>Struct dob
        {
                int day;
                int month;
                int year;
        };


struct student
{
        char name[30];
        int htno;
        struct dob d1;
        };

int main()
{
        struct student st;
        printf("Enter student details\n");
        printf("Enter student name:");
        scanf("%s",st.name);
        printf("Enter rollno:");
        scanf("%d",&st.htno);
         printf("Enter the day:");
        scanf("%d",&st.d1.day);
        printf("Enter the month:");
        scanf("%d",&st.d1.month);
         printf("Enter the year:"); scanf("%d",&st.d1.year);
        printf("Student Details\n------------.............\n");
        printf("Student Name:%s\n",st.name);
        printf("Roll No:%d\n",st.htno);
        printf("Date of birth:%d-%d-%d\n",st.d1.day,st.d1.month,st.d1.year);
        return 0;
}
```

Output

```
Enter student details
Enter student name: Chikku
Enter roll no: 1224
Enter address: vizag
Enter the day:15
Enter the month:8
Enter the year:1991
Student Details
-------------------
Student Name: Chikku
Roll No: 1224
Date of birth is: 15-8-1991
Address: vizag
```

6.6 Array of structures :

- ✓ An array of structures is declared in the same way as we declare an array of a predefined data type.

  Syntax :

```
 struct struct name
{
 data_type member;
…………………
…………………..
…………………..
}
```

```
struct struct_name struct_var[index];
```

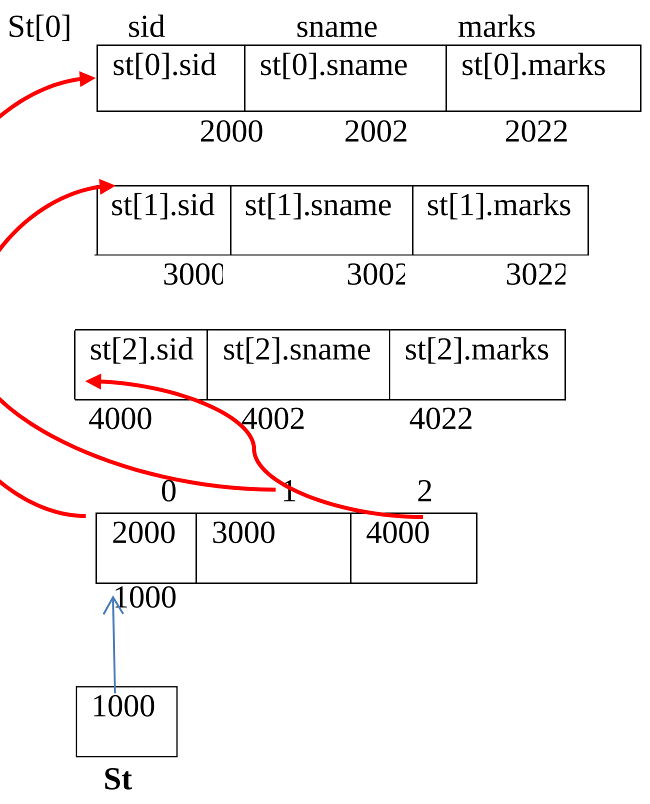
Example:

```
#include<stdio.h>
struct student
{
        int sid;
        char name[100];
        float marks;
};
int main( )
{
   int i;
  struct student st[3];
  printf("enter three students details:");
  for(i=0;i<3;i++)
{
printf("\n enter the students roolno:");
scanf("%d",&st[i].sid);
printf("\n enter the students Name:");
scanf("%s",st[i].name);
 printf("\n enter the students marks:");
scanf("%f",&st[i].marks);
}
  for(i=0;i<3;i++)
{
printf("Details Of students :");
printf("Roll no=%d",st[i].sid);
printf("Name=%s",st[i].name);
printf("marks =%f",st[i].marks);
}
```

OUTPUT:

```
enter the students roolno: 1224
enter the students name: Ram
enter the students marks: 78.9
enter the students roolno: 1256
enter the students name:  Syam
enter the students marks: 45.7
enter the students roolno: 7654
enter the students name: Lava
enter the students marks: 90.9
Details Of students
Roll no=1224
Name=Ram
marks = 78.9
Details  Of students
Roll no=Syam
Name=1256
```

```
marks = 45.7
Details  Of students
Roll no=7654
Name=Lava
marks = 90.9
```

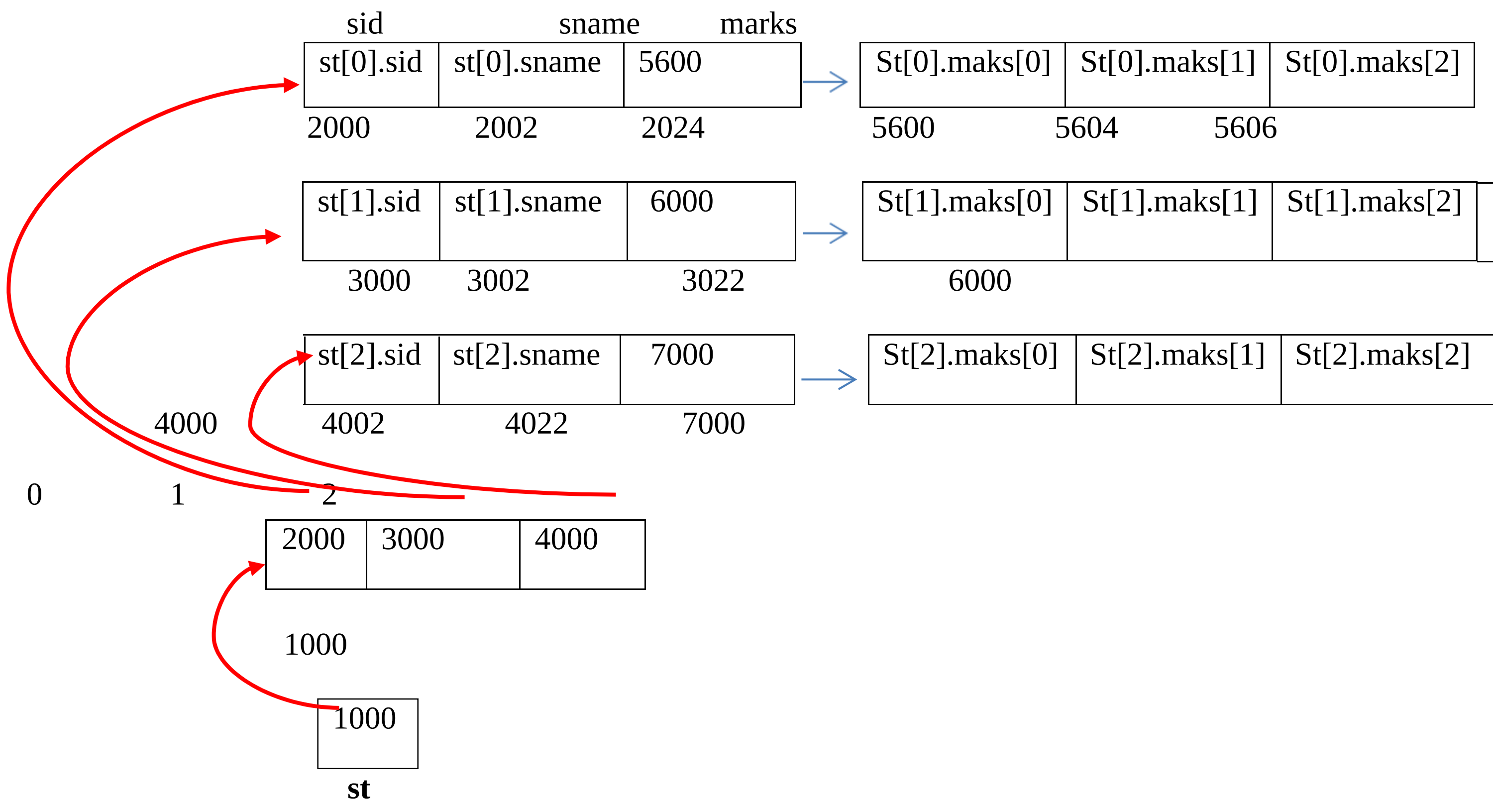


6.7 Arrays in structure:

- ✓ We are also declare arrays as structure members , for example, I need three subject marks for each student then we declare a marks array variable inside the structure.

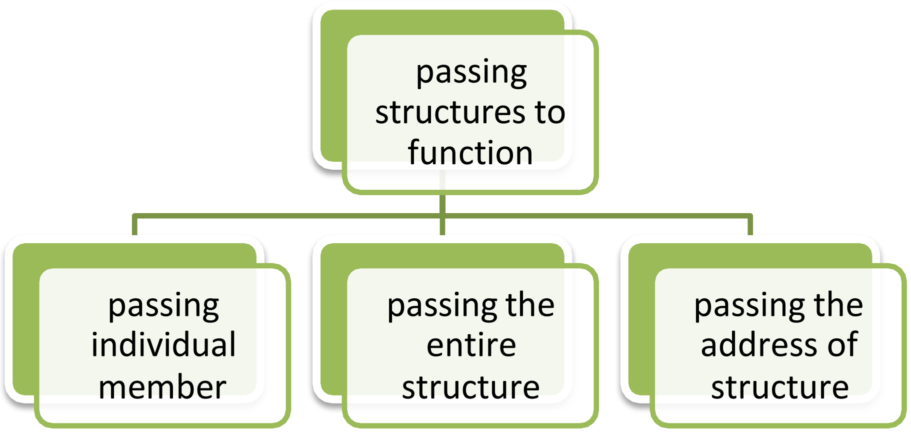
Example:

```
struct student
{
        int regid;
        char name[20];
        float marks[3];
};
struct student st[3];
```



6.8 Structures to Functions:-

→ A function may access the members of a structure in three ways


passing structures to function
passing individual member
passing the entire structure
passing the address of structure

6.8.1 Passing individual member:

→ To pass any individual members of the structure to a function we must use the direct selection operator to refer to the individual member for the actual parameters.

→ The called function does not know it the two variable are ordinary variable or structure member

Ex:-

```
#include<stdio.h>
struct prgm
{
        int x;
           int y;
};
void add(int,int);
void main()
{
        struct prgm p={24,7};
        add(p.x,p.y);
}
void add(int a,int b)
{
        printf("%d+%d=%d",a,b,a+b);
}
```

Output:-

```
24+7=31
```

```
#include<stdio.h>
typedef struct
{
        int x;
           int y;
}student;
void add(int,int);
void main()
{
        student p={24,7};
        add(p.x,p.y);
}
void add(int a,int b)
{
        printf("%d+%d=%d",a,b,a+b);
}
```

Output:-

```
24+7=31
```

6.8.2 Passing entire structure :

- → We can pass an entire structure as a function argument when a structure is passed as an argument. It is using call by value method.
- → A copy of each member of structure is made. This is a very complicated especially when structure is very big or the function is called frequently.
- → In such a situation passing and working with pointers may be more efficient.

Syntax:-

```
struct struct_name fun_name(struct struct_name struct var);
```

```
#include<stdio.h>
struct prgm
{
        int x;
         int y;
};
void add(struct prgm );
void main()
{
        struct prgm p={24,7};
        add(p);
}
void add(struct prgm pr)
{
      printf("%d+%d=%d",pr.x,pr.y,
pr.x+pr.y);
}
```

Output:-

```
24+7=31
```

```
#include<stdio.h>
typedef struct
{
        int x;
         int y;
}student;
void add(student);
void main()
{
        student p={24,7};
        add(p);
}
void add(student s)
{
        printf("%d+%d=%d",s.x,s.y,s.x+s.y);
}
```

Output:-

```
24+7=31
```

Typedef in structures:-

```
#include<stdio.h>
typedef struct student
{
        int x;
         int y;
};
void main()
{
        typedef struct student a;
        a s={27,3};
        printf("%d" ,s.x+s.y);
}
```

```
#include<stdio.h>
typedef struct student
{
        int x;
    int y;
}a;
void main()
{
        a s={27,3};
        printf("%d",s.x+s.y);
}
```

6.8.3 Passing structures through pointers (pointers to structures):-

- → Passing large structures to function using the call by value method is very inefficient.
- → It is professional to run structure through pointer
- → It is possible to create a pointer to almost any type in C ,including user defined types

Syntax:-

```
struct struct_name
{
      datatype member_name;
      .............;
      ..............;
}*ptr;
                  Or
struct struct_name *ptr;
```

Ex:-

```
#include<stdio.h>
typedef struct student
{
        int sid;
        char name[20];
        float marks;
};
void main()
{
        typedef struct student st;
        st *ptr1,*ptr2,st1,st2;
        ptr1=&st1;
        ptr2=&st2;
        printf("Enter sid1=");
        scanf("%d",&ptr1->sid);
        printf("Enter name1=");
        scanf("%s",&(ptr1->name));
        printf("Enter marks1=");
        scanf("%f",&(ptr1->marks));
        printf("Enter sid2=");
        scanf("%d",&ptr2->sid);
        printf("Enter name2=");
        scanf("%s",&(ptr2->name));
        printf("Enter marks2=");
        scanf("%f",&(ptr2->marks));
        printf("\nStudent details=\n");
        printf("sid=%d\nsname=%s\nmarks=%.1f\n\n",ptr1->sid,ptr1->name,ptr1->marks);
        printf("sid=%d\nsname=%s\nmarks=%.1f\n",ptr2->sid,ptr2->name,ptr2->marks);  }
```

Output

```
enter sid=123
enter name1=Lakki
enter marks1=9.8
enter sid2=456
entername2=aba
enter marks2=9.5

Student details=
sid1=123
name1=lakki
marks1=9.8

sid2=456
name2=aba
marks2=9.5
```

6.9 Self-referential structure

A self-referential is one which contains a pointer to its own type. This type of structure is also known as linked list. For example

```
struct node
{
	int data;
	struct node *nextptr;
};
```

Defines a data type, struct node. A structure type struct node has two members-integer member data and pointer member nextptr. Member nextptr points to a structure of type struct node – a structure of the same type as one being declared here, hence the term “self-referential structure”. Member nextptr referred to as a link i.e. nextptr can be used to tie a structure of type struct node to another structure of the same type. self referential structures can be linked together to form a usual data structures such as lists etc.

```
#include<stdio.h>
struct student
{
	char name[30];
	int age;
	char address[20];
	struct student *next;
};
int main()
{
    struct student st1={"Ramesh",24,"Vizag"};
    struct student st2={"Rajesh",21,"Nuzvid "};
    struct student st3={"Mahesh",22,"Hyd"};
    st1.next = &st2;
    st2.next = &st3;
    st3.next = NULL;
    printf("Student Name:%s\tAge:%d\tAddress:%s\n",st1.name,st1.age,st1.address);
    printf("Student 1 stored at %x \n",st1.next);
    printf("Student Name:%s\tAge:%d\tAddress:%s\n",st2.name,st2.age,st2.address);
    printf("Student 2 stored at %x \n",st2.next);
    printf("Student Name:%s\tAge:%d\tAddress:%s\n",st3.name,st3.age,st3.address);
    printf("Student 3 stored at %x \n",st3.next); return 0;
}
```

Output

```
Student Name:Ramesh        Age:24        Address: Vizag
Student 1 stored at 88f0
Student Name:Rajesh        Age:21        Address:Nuzvid
Student 2 stored at 8854
Student Name:Mahesh        Age:22        Address: Hyd
Student 3 stored at 0
```

To Create Structure using malloc( ) :

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
        int n,i;
        printf("Enter no of students=");
        scanf("%d",&n);
        struct student
        {
                int regid;
                char name[100];
                float cgpa;
        };
        struct student*ptr;
        ptr=(struct student*)malloc(n*sizeof(struct student));
        if(ptr==NULL)
        {
                printf("no storage left on your device");
                exit(1);
        }
        else
        {
                for(i=0;i<n;i++)
                {
                        printf("enter REGID=");
                        scanf("%d",&(ptr+i)->regid);
                        printf("enter name=");
                        scanf("%s",&(ptr+i)->name);
                        printf("enter cgpa=");
                        scanf("%f",&(ptr+i)->cgpa);
                }
                Printf("\nStudent details=");
                for(i=0;i<n;i++)
                {
                printf("regid=%d\n",(ptr+i)->regid);
                printf("name=%s\n",(ptr+i)->name);
                printf("cgpa=%.1f\n\n",(ptr+i)->cgpa);
                }
        }
}
```

Output:-

```
Enter no of students=2
enter  REGID=123
enter name=Ramesh
enter cgpa=9.8
enter REGID=456
enter name=abc
enter cgpa=9.5

Student details=
REGID=123
name=Ramesh
cgpa=9.8

 REGID=456
name=abc
cgpa=9.5
```

Note:-

If pointer variable is pointing to the structure we are using ->operator to read the structure members.