STRUCTURES

Difference between arrays and structures:

i. Data Collection

Array is a collection of homogeneous data.
Stucture is a collection of heterogeneous data.

ii. Element Reference

Array elements are referred by subscript.
Structure elements are referred by its unique name.

iii. Access Method

Array elements are accessed by it's position or subscript.
Stucture elements are accessed by its object as '.' operator.

iv. Data type

Array is a derived data type.
Structure is user defined data type.

Structure definition:

•A structure is a collection of variables under a single name.

•These variables can be of different types.

•A structure is a convenient way of grouping several pieces of related information together.

Declaration:

Struct struct_name{
structure_members
}instance1,instance2….;

(Or) *Struct struct_name{*
structure_members
};
struct struct_name instance1,instance2…..;

Ex:

```
struct address{
      unsigned int house_number;
      char street_name[50];
      int zip_code;
      char country[50];
    };
Struct address ad;
```

ad is object to struct address.

We can access structure elements by using '.' operator.

object.structure_element;

Ex ad.house_number

Sample program:

```
main()
{
        struct student
        {
                int no;
                char name[20];
                float fee;
        };
        struct student stu;
        printf("\n enter student name, no, fee::");
        scanf("%s %d %f ",stu.name, &stu.no, &stu.fee);
        printf("\n entered student details are as follows::\n");
        printf("\n Number::%d\n Name is::%s\n Fee::%f “, stu.no, stu.name, stu.fee);
         printf("\n address are as follows::");
         printf("\n number::%u \n name::%u\n fee::%u ",&stu.no,&stu.name,&stu.fee);
        printf("\n size of atructure is::%d ",sizeof(stu));
        printf("\n");
}
```

Output:

```
enter student name, no, fee::ism_student
23
30000

 entered student details are as follows::

 Number::23
 Name is::ism_student
 Fee::30000.000000
 address are as follows::
 number::3218583400
 name::3218583404
 fee::3218583424
 size of structure is::28
```

Pointers with structures

```
main()
{
        struct student { int no; char name[20]; float fee; };
        struct student stu;
        struct student *ptr;
        ptr=&stu;
        printf("\n enter student name, no, fee::");
        scanf("%s%d%f",ptr->name,&ptr->no,&ptr->fee);
        printf("\n entered student details are as follows::\n");
        printf("\n Number::%d\n Name is::%s\n Fee::%f",
            ptr->no,ptr->name,ptr->fee);
        printf("\n");
}
```

Array of structures:

```
main()
{
	int n,i;
	struct student { int no; char name[20]; float fee; };
	struct student stu[100];
	printf("\n enter how many records you are going to enter::");
	scanf("%d",&n);
	for(i=0;i<n;i++){
		printf("\n enter %d record::",i+1);
		printf("\n enter student name, no, fee::");
		scanf("%s%d%f",stu[i].name,&stu[i].no,&stu[i].fee);
	}
	printf("\n entered student details are as follows::\n");
	for(i=0;i<n;i++){
		printf("\n %d record is",i+1);
		printf("\n Number::%d\n Name is::%s\n Fee::%f",stu[i].no,stu[i].name,stu[i].fee);
		printf("\n address are as follows::");
		printf("\n number::%u\n name::%u\n fee::%u",&stu[i].no,&stu[i].name,&stu[i].fee);
	}
	printf("\n size of atructure is::%d",sizeof(stu[0]));
	printf("\n");
}
```

Out put:

```
enter how many records you are going to enter::2
 enter 1 record::
enter student name, no, fee::ism_stu1 23 30000
enter 2 record::
enter student name, no, fee::ism_stu2 25 30000
entered student details are as follows::
1 record is
Number::23        Name is::ism_stu1          Fee::30000.000000
address are as follows::
number::3220795044        name::3220795048          fee::3220795068
2 record is
Number::25        Name is::ism_stu2          Fee::30000.000000
address are as follows::
number::3220795072        name::3220795076          fee::3220795096

size of structure is::28
```

Passing structure to function:

```
struct student { int no; char name[20]; float fee; };
void disp(struct student *);
main(){
        struct student stu;
        printf("\n enter student name, no, fee::");
        scanf("%s%d%f",stu.name,&stu.no,&stu.fee);
        disp(&stu);
}
void disp(struct student *stu1){
     printf("\n entered student details are as follows::\n");
  printf("\n Number::%d\n Name is::%s\n
Fee::%f",stu1>no,stu1>name,
  stu1->fee);
        printf("\n");
}
```

Output:

```
enter student name, no, fee::
ism_student
23
30000

 entered student details are as follows::

 Number::23
 Name is::ism_student
 Fee::30000.000000
```

Returning a structure pointer to the function:

```
struct student { int no; char name[20]; float fee; }stu;
void disp(struct student *);
        struct student *read() {
      printf("\n enter student name, no, fee::");
      scanf("%s%d%f",stu.name,&stu.no,&stu.fee);
      return &stu;       }
main() {
      struct student *stu;
      stu=read();
      disp(stu);          }
void disp(struct student *stu1) {
printf("\n entered student details are as follows::\n");
printf("\n Number::%d\n Name is::%s\n Fee::%f",stu1->no,
stu1->name,stu1->fee);
      printf("\n");
}
```

Output:

```
enter student name, no, fee::
ism_student
43
30000

 entered student details are as follows::

 Number::43
 Name is::ism_student
 Fee::30000.000000
```

Structure alignment and padding:

•Char variables can be byte aligned and appear at any byte boundary .

•Short (2 byte) variables must be 2 byte aligned, they can appear at any even byte boundary. This means that 0x10004567 is not a valid location for a short variable but 0x10004566 is.

•Long (4 byte) variables must be 4 byte aligned, they can only appear at byte boundaries that are a multiple of 4 bytes. This means that 0x10004566 is not a valid location for a long variable but 0x10004568 is.

•Structure padding occurs because the members of the structure must appear at the correct byte boundary, to achieve this the compiler puts in padding bytes (or bits if bit fields are in use) so that the structure members appear in the correct location.

Before Structure Packing:

```
#include<stdio.h>
int main()
{
        struct sample
        {
                int a;
                char b;
                int c;
        };
        struct sample s;
        printf("\n%d\n",sizeof(s));
        return 0;
}
```

After Structure Packing:

```
#include<stdio.h>
//#pragma pack(1)
int main()
{
        struct sample
        {
                int a;
                char b;
                int c;
        }__attribute__((packed));
        struct sample s;
        printf("\n%d\n",sizeof(s));
        return 0;
}
```

Self-reference Structures:

Creating a structure pointer within the same structure, which is for referring itself.

```
struct samp
 {
        int data;
        struct samp *node;
 }s;
```

Sizeof(s) = 8 bytes.

•Node is a pointer having the capability of holding the address of same structure type.

•Finally this structure having the capability holding one integer type value and one address location of same structure type.

Sample program:

```
main() {
 struct samp { int data; struct samp *node; }s1,s2;
 s1.data=10;      s1.node=&s2;    s2.data=20;  s2.node=NULL;
 printf("\n sizeof s1 is::%d\n sizeof s2 is::%d",sizeof(s1),sizeof(s2));
 printf("\n address of s1 is::%p",&s1);
 printf("\n address of s1 data is::%p",&s1.data);
 printf("\n asddress of s1 node is::%p",&s1.node);
 printf("\n value at s1 data is::%d",s1.data);
 printf("\n value at s1 node is::%p",s1.node);
 printf("\n address of s2 is::%p",&s2);
 printf("\n address of s2 data is::%p",&s2.data);
 printf("\n asddress of s2 node is::%p",&s2.node);
 printf("\n value at s2 data is::%d",s2.data);
 printf("\n value at s2 node is::%p",s2.node);
 }
```

Output:

sizeof s1 is::8
sizeof s2 is::8
address of s1 is::0xbf8b5acc
address of s1 data is::0xbf8b5acc
asddress of s1 node is::0xbf8b5ad0
value at s1 data is::10
value at s1 node is::**0xbf8b5ac4**
address of s2 is::**0xbf8b5ac4**
address of s2 data is::0xbf8b5ac4
asddress of s2 node is::0xbf8b5ac8
value at s2 data is::20
value at s2 node is::(nil)