Structures

1

Structure

- A structure is a derived data type that contains a number of data types grouped together.
- These data types may or may not be of the same type.

2

```
void main()
{
	struct account
	{
		int acc_no;
		char acc_name[20];
		float acc_balance;
	};
	struct account a1,a2;
	printf("\nsize=%d",sizeof(struct account));
	printf("\nEnter account no, Name and Balances:\n");
	scanf("%d %s %f",&a1.acc_no,a1.acc_name,&a1.acc_balance);
	scanf("%d %s %f",&a2.acc_no,a2.acc_name,&a2.acc_balance);

	printf("\n%d %s %f",a1.acc_no,a1.acc_name,a1.acc_balance);
	printf("\n%d %s %f",a2.acc_no,a2.acc_name,a2.acc_balance);
}
```

3

OUTPUT

```
size=28
Enter account no, Name and Balances:
123 ABCD 2345.50
345 DFGR 4523.50

123 ABCD 2345.500000
345 DFGR 4523.500000Press any key to continue
```

4

Structures can be declared by the following means:

```
struct account
          {
                    int acc_no;
                    char acc_name[20];
                    float acc_balance;
          }; //semicolon is a must
          struct account a1, a2={123,”Naveen”,12345.50};

struct account
          {
                    int acc_no;
                    char acc_name[20];
                    float acc_balance;
          } a1, a2={123,”Naveen”,12345.50};
```

5

//Array of Structures

```
void main()
{
          struct account
          {
                    int acc_no;
                    char acc_name[20];
                    float acc_balance;
          };
          struct account a[10];
//This provides space in memory for 10 structures of the type struct account
          int i;
          for(i=0;i<10;i++)
          {
          printf("\nEnter account no, Name and Balance:\n");
          scanf("%d %s %f",&a[i].acc_no,a[i].acc_name,&a[i].acc_balance);
          }
          for(i=0;i<10;i++)
            printf("\n%d %s %f",a[i].acc_no,a[i].acc_name,a[i].acc_balance);
}
```

6

Point to Remember

- Structure elements are always stored in adjacent memory locations.
- In an array of structures all elements of the array are stored in adjacent memory locations.
- In the previous example, a[0]'s account number, name and balance in memory would be immediately followed by a[1]'s account number, name and balance and so on.

7

- The values of a structure variable can be assigned to another structure variable of the **same** type using the assignment operator.
- It is not necessary to copy the structure elements piece-meal (one-by-one).

8

```
void main()
{
        struct employee
        {
                char name[10];
                int age;
                float salary;
        }e1={"ABC",30,15000.50},e2,e3;

        //piece-meal copying
        strcpy(e2.name,e1.name);
        e2.age=e1.age;
        e2.salary=e1.salary;

        //copying all elements at one go
        e3=e2;

        printf("\nThrough e1: %s,%d,%f",e1.name,e1.age,e1.salary);
        printf("\nThrough e2: %s,%d,%f",e2.name,e2.age,e2.salary);
        printf("\nThrough e3: %s,%d,%f\n",e3.name,e3.age,e3.salary);
}
```

OUTPUT:

```
Through e1: ABC,30,15000.500000
Through e2: ABC,30,15000.500000
Through e3: ABC,30,15000.500000
Press any key to continue
```

```
//Nesting of structures
void main()
{
        struct address
        {
                char phone[15];
                char city[25];
                int pin;
        };
        struct emp
        {
                char name[25];
                struct address a;
        }e={"Naveen","1234567","Lucknow",226022};

        printf("\nname=%s,phone=%s",e.name,e.a.phone);
        printf("\ncity=%s,pin=%d\n",e.a.city,e.a.pin);
}
```

11

OUTPUT:

```
name=Naveen, phone=1234567
city=Lucknow, pin=226022
Press any key to continue
```

- **maruti.engine.bolt.large.quantity**
- This means we are referring to the quantity of large bolts which fit on an engine of a maruti car.

12

//Passing individual structure elements to a function

```
void display(char*,char*,int);
void main()
{
        struct book
        {
                char name[25];
                char author[25];
                int pages;
        } b={"Let Us C","YPK",700};
        display(b.name,b.author,b.pages);
```

/*Here we are passing the base addresses of the arrays **name** and **author**, but the value stored in **pages**. Thus, this is a mixed call: a call by reference as well as a call by value
*/

```
}
void display(char *s,char *t,int n)
{
        printf("\nname=%s",s);
        printf("\nauthor=%s",t);
        printf("\nNumber of pages=%d\n",n);
}
```

OUTPUT:

```
name=Let Us C
author=YPK
Number of pages=700
Press any key to continue
```

13

//Passing entire structure variable at a time to a function

```
void display(struct book);
struct book
{
        char name[25];
        char author[25];
        int pages;
};
void main()
{
        struct book b1={"Let Us C","YPK",700};
        display(b1);
}
void display(struct book b)
{
        printf("\nname=%s",b.name);
        printf("\nauthor=%s",b.author);
        printf("\nNumber of pages=%d\n",b.pages);
}
```

14

Point to Remember

- Note that **struct book** has been defined outside **main()**.
- It is so because the data type **struct book** is not known to the function **display()**.
- Defining **struct book** before any function makes it available to all the functions in the program.

```
/* FUNCTION RETURNING A STRUCTURE */

#include<stdio.h>
struct data
{
        int quantity;
        double price;

};
struct data fun(int,double);
```

```
main()
{
	struct data a;
	int q;
	double p;
	printf("\nEnter quantity and price:");
	scanf("%d %lf",&q,&p);
	a = fun(q,p);
	printf("\nUpdated quantity = %d",a.quantity);
	printf("\nUpdated price = %lf",a.price);
}
```

17

```
struct data fun(int q, double p)
{
	struct data item={0,0};
	item.quantity += q;
	item.price += p;
	return(item);
}
```

OUTPUT:

```
Enter quantity and price:100 25.50

Updated quantity = 100
Updated price = 25.500000
Press any key to continue
```

18

```
//Usage of a Structure pointer
void main()
{
        struct book
        {
                char name[25];
                char author[25];
                int pages;
        } b={"Let Us C","YPK",700};
  struct book *ptr;
  ptr=&b;
 //ptr will contain the base address of b i.e. the address of the character 'L'

 //printing using structure variable
 printf("\nname=%s,author=%s,pages=%d",b.name,b.author,b.pages);

 //printing using structure pointer
 printf("\nname=%s,author=%s,pages=%d",ptr->name,ptr->author,ptr->pages);

}
```

19

```
//Passing address of a structure variable
void display(struct book *);
struct book
{
        char name[25];
        char author[25];
        int pages;
};
void main()
{
        struct book b1={"Let Us C","YPK",700};
        display(&b1);
}
void display(struct book *b)
{
        printf("\nname=%s",b->name);
        printf("\nauthor=%s",b->author);
        printf("\nNumber of pages=%d\n",b->pages);
}
```

20

Point to Remember

- Remember that on the left hand side of the **‘.’** structure operator, there must always be a structure variable.
- Whereas on the left hand side of the **‘->’** operator, there must always be a pointer to a structure.