UNIT - 5

STRUCTURES & UNIONS

Structures

We know that arrays can be used to represent a collection of elements of the same data type like int, float etc. They cannot be used to hold a collection of different types of elements. C supports a structured data type known as a **structure**, a way for packing data of different data types. A **structure** is a convenient tool for handling a group of logically related data items. For example, it can be used to represent the details of a student or the details of a text book etc.
Definition: A structure is a collection of heterogeneous data elements referred by the same name.

Defining a Structure

Unlike arrays, structures must be first defined first for their later use. The syntax for defining a structure is as shown below:

```
struct  sturcturename/tagname
{
        datatype var1;
        datatype var2;
        ----
        ----
};
```

The keyword **struct** declares a structure. The **structurename / tagname** represents the name of the structure. The structure definition is always terminated with a semicolon. Let us see an example for defining a structure to store the details of a student:

```
struct student
{
        char name[20];
        char rollno[15];
        int age;
        char grade;
};
```

In the above example, **student** is the name of the structure. The members of the student structure are: name, rollno, age and grade. A structure itself does not occupy any memory in the RAM. Memory is allocated only when we create variables using the structure.

Declaring Structure Variables

After defining a structure, we can create variables of that type. A structure variable declaration is similar to the declaration of variables of any other data types. It includes the following elements:

1. The keyword **struct.**
2. The structure name or tagname.
3. List of variable names separated with commas.
4. A terminating semicolon.

The syntax for declaring structure variables is as shown below:

```
struct  structurename  var1, var2, …., varN;
```

Example for creating structure variables is shown below:

```
struct student student1, student2, student3;
```

We can also combine the structure definition and the declaration of structure variables into a single line as shown below:

```
struct student
{
                char name[20];
                char rollno[15];
                int age;
                char grade;
}student1, student2, student3;
```

Type-Defined Structures

We can use the keyword **typedef** to define a structure as follows:

```
typedef  struct
{
        char name[20];
        char rollno[15];
        int age;
        char grade;
}student;
```

The name **student** represents the structure definition associated with it and therefore can be used to declare structure variables as shown below:

```
student student1, student2, student3;
```

Accessing Structure Members

We can access and assign values to the members of a structure in a number of ways. As mentioned earlier, the members themselves are not variables. They should be linked to the structure variables in order to make them meaningful members. The link between a member and a variable is established using the member operator ' . ' which is also known as dot operator or period operator. The syntax for accessing a structure member is as shown below:

```
structure-varaible.membername
```

Examples of accessing the members of the student structure are shown below:

```
student1.name

student1.age
```

Structure Initialization

Like any other data type, a structure variable can be initialized at compile time. An example of compile time initialization of the student structure is shown below:

```
struct student
{
                char name[20];
                char rollno[15];
                int age;
                char grade;
};
struct student student1 = {“teja”,”10A1”,26,’A’};
struct student student2 = {“raju”,”10A2”,24,’B’};
```

In the above example: teja, 10A1, 26 and A are assigned to student1’s name, rollno, age and grade. Whereas raju, 10A2, 24, B are assigned to student2’s name, rollno, age and grade.
Note: *Partial initialization of a structure variable is supported. For example we can assign values to name and rollno and leave out age and grade. In case of partial initialization, the default values for int type is zero. For float and double it is 0.0. For characters and strings it is ‘\0’.*

An example for dynamic initialization is shown below:

```
gets(student1.name);
gets(student1.rollno);
scanf(%d”, &student1.age);
scanf(“%c”, &grade);
```

Arrays of Structures

We use structures to describe the format of a number of related variables. For example, we want to store details of 100 textbooks it will become difficult to declare and maintain 100 variables and store the data. Instead, we can declare and array of structure variables as shown below:

```
struct textbook
{
                char name[40];
                char author[20];
                int pages;
}
struct textbook book[10];
```

In the above example, we are declaring an array book with 10 elements of the type **textbook** which is a structure.

Structures within Structure

In C, structures can be nested. A structure can be defined within in another structure. The members of the inner structure can be accessed using the variable of the outer structure as shown below:

```
outer-struct-variable.inner-struct-variable.membername
```

An example for a nested structure is shown below:

```
struct student
{
                struct
                {
                        char fname[20];
                        char mname[20];
                        char lname[20];
                }name;
                char grade;
};
struct student s1;
strcpy(s1.name.fname, “satish”);
```

In the above example, **student** is the outer structure and the inner structure **name** consists of three members: fname, mname and lname.

Structures and Functions

We know that the functions are the basic building blocks of a C program. So, it is natural for C language to support passing structures as parameters in functions. We pass a copy of the entire structure to the called function. Since the function is working on a copy of the structure, any changes to structure members within the function are not reflected in the original structure. The syntax for passing the structure variable as a parameter is shown below:

```
return-type function-name(struct structname var)
{
                ----
                ----
                ----
                return expression;
}
```

Programs

```
/* C program to create a structure */
#include<stdio.h>
#include<conio.h>
main()
{
   struct student
   {
      char name[20];
      char rollno[15];
      int age;
      char grade;
   };
   struct student student1;
   clrscr();
   getch();
}
```

```
/* C program to create a structure using typedef*/
#include<stdio.h>
#include<conio.h>
main()
{
   typedef struct
   {
      char name[20];
      char rollno[15];
      int age;
      char grade;
   }student;
   student student1;
   clrscr();
   getch();
}
```

```
/* C program to create a structure */
#include<stdio.h>
#include<conio.h>
main()
{
   struct student
   {
      char name[20];
      char rollno[15];
      int age;
      char grade;
   }student1, student2;
   clrscr();
```

```
   getch();
}


/* C program to demostrate static initialization of a structure */
#include<stdio.h>
#include<conio.h>
main()
{
   struct student
   {
      char name[20];
      char rollno[15];
      int age;
      char grade;
   };
   struct student student1 = {"teja","10A1",26,'A'};
   clrscr();
   printf("Name of student1 is: %s\n",student1.name);
   printf("Rollno of student1 is: %s\n",student1.rollno);
   printf("Age of student1 is: %d\n",student1.age);
   printf("Grade of student1 is: %c\n",student1.grade);
   getch();
}


/* C program to demostrate dynamic initialization of a structure */
#include<stdio.h>
#include<conio.h>
main()
{
   struct student
   {
      char name[20];
      char rollno[15];
      int age;
      char grade;
   };
   struct student student1;
   clrscr();
   printf("Enter the name of student1: ");
   gets(student1.name);
   printf("Enter the rollno of student1: ");
   gets(student1.rollno);
   printf("Enter the age of student1: ");
   scanf("%d",&student1.age);
   printf("Enter the grade of student1: ");
   fflush(stdin);
   scanf("%c",&student1.grade);
   printf("\n\nName of student1 is: %s\n",student1.name);
   printf("Rollno of student1 is: %s\n",student1.rollno);
```

```
   printf("Age of student1 is: %d\n",student1.age);
   printf("Grade of student1 is: %c\n",student1.grade);
   getch();
}
```

```
/* C program for copying one structure variable into another */
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
   struct textbook
   {
      char name[40];
      char author[20];
      int pages;
   };
   struct textbook t1,t2;
   clrscr();
   strcpy(t1.name, "C Programming");
   strcpy(t1.author, "E Balaguruswamy");
   t1.pages = 100;
   t2 = t1;
   printf("Name of text book is: %s",t2.name);
   printf("\nAuthor of text book is: %s",t2.author);
   printf("\nPages in text book are: %d",t2.pages);
   getch();
}
```

```
/* C program for comparing structure variables */
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
   struct textbook
   {
      char name[40];
      char author[20];
      int pages;
   };
   struct textbook t1,t2;
   clrscr();
   strcpy(t1.name, "C Programming");
   strcpy(t1.author, "E Balaguruswamy");
   t1.pages = 100;
   t2 = t1;
   if(t1.name == t2.name && t1.author == t2.author && t1.pages == t2.pages)
   {
```

```
      printf("t1 and t2 are equal");
   }
   else
   {
      printf("t1 and t2 are not equal");
   }
   getch();
}
```

```
/* C program to demonstrate arrays of structures */
#include<stdio.h>
#include<conio.h>
main()
{
   struct textbook
   {
      char name[40];
      char author[20];
      int pages;
   };
   struct textbook book[10];
   int i;
   clrscr();
   for(i = 0; i < 3; i++)
   {
      fflush(stdin);
      printf("Enter the name of book %d: ",i+1);
      gets(book[i].name);
      printf("Enter the author of book %d: ",i+1);
      gets(book[i].author);
      printf("Enter the pages of book %d: ",i+1);
      scanf("%d",&book[i].pages);
      printf("\n");
   }
   for(i = 0; i < 3; i++)
   {
      printf("Name of book %d is: %s \n",i+1,book[i].name);
      printf("Author of book %d is: %s \n",i+1,book[i].author);
      printf("Pages in book %d is: %d \n",i+1,book[i].pages);
      printf("\n");
   }
   getch();
}
```

```
/* C program to demonstrate partial initialization of a structure */
#include<stdio.h>
#include<conio.h>
main()
{
   struct textbook
   {
      char name[40];
      char author[20];
      int pages;
      float price;
   };
   struct textbook t1 = {"C Programming","E Balaguruswamy"};
   clrscr();
   printf("Name is: %s\n",t1.name);
   printf("Author is: %s\n",t1.author);
   printf("Pages are: %d\n",t1.pages);
   printf("Price is: %f",t1.price);
   getch();
}
```

```
/* C program to demostrate structure within structure */
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
   struct student
   {
      struct
      {
         char fname[20];
         char mname[20];
         char lname[20];
      }name;
      char grade;
   }s1;
   clrscr();
   strcpy(s1.name.fname, "Pericherla");
   strcpy(s1.name.mname, "Satya");
   strcpy(s1.name.lname, "Suryateja");
   s1.grade = 'A';
   printf("Name is: %s %s %s \n",s1.name.fname,s1.name.mname,s1.name.lname);
   printf("Grade is: %c",s1.grade);
   getch();
}
```

```
/* C program to demonstrate passing structure to a function */
#include<stdio.h>
#include<conio.h>
struct textbook
{
   char name[40];
   char author[20];
   int pages;
   float price;
};
void dispbook(struct textbook t);
main()
{
   struct textbook t1={"C Programming","E Balaguruswamy",300,200};
   clrscr();
   dispbook(t1);
   getch();
}
void dispbook(struct textbook t)
{
   printf("Name is: %s \n",t.name);
   printf("Author is: %s \n",t.author);
   printf("Pages are: %d \n",t.pages);
   printf("Price is: %f",t.price);
}
```

```
/* C program to calculate size of a structure */
#include<stdio.h>
#include<conio.h>
main()
{
   struct textbook
   {
      char name[40];
      char author[20];
      int pages;
      float price;
   };
   struct textbook t1;
   clrscr();
   printf("Size of structure is: %d",sizeof(t1));
   getch();
}
```

Unions

Unions have the same syntax as that of a structure since both of them are similar. However, there is a major difference between them in terms of memory allocation. A structure is allocated memory for all the members of the structure whereas a union is allocated memory only for largest member of the union. This implies that, although a union may contain many members of different types, it can handle only one member at a time. Like structure, a union can be declared using the **union** keyword as shown below:

```
union student
{
        char name[20];
        char grade;
};
union student s1,s2;
```

In the above code **student** is the name of the union. Also **s1** and **s2** are union variables. Memory is allocated only for **name** member of the union. So, the limitation on unions is: *only one member can be used at a time.* Unions can be used in all places where a structure is allowed.

Program

```
/* C program to demostrate union */
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
   union student
   {
      char name[20];
      char grade;
      int marks;
   };
   union student s1;
   clrscr();
   strcpy(s1.name,"siva kumar");
   s1.grade = 'A';
   s1.marks = 98;
   printf("Name is: %s \n",s1.name);
   printf("Grade is: %c \n",s1.grade);
   printf("Marks are: %d",s1.marks);
   getch();
}
```

In the above program we will only get the correct value **98** for the member s1.marks as in a union only one value can b e used and stored at a time. We will get unexpected values for other members: name and grade.