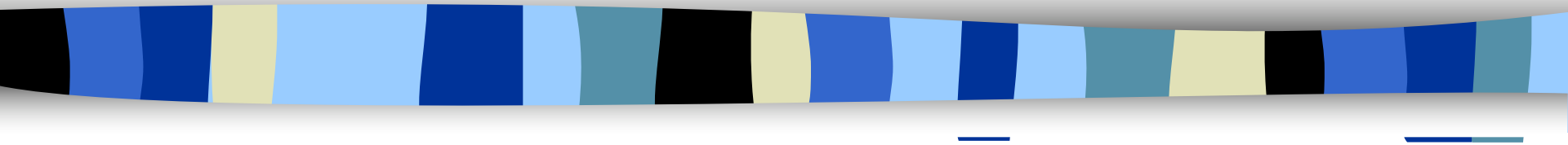
Structures



Why Use Structures

- We have seen earlier how ordinary variables can hold one piece of information and how arrays can hold a number of pieces of information of the same data type.
- These two data types can handle a great variety of situations. But quite often we deal with entities that are collection of dissimilar data types.
- For example, suppose you want to store data about a book. You might want to store
  - its name (a string),
  - its price (a float) and
  - number of pages in it (an int).

  If data about say 3 such books is to be stored, then we can follow **two** approaches:

(a) Construct individual arrays, one for storing names, another for storing prices and still another for storing number of pages.

(b) Use a structure variable.

What is a Structure?

- Arrays allow to define type of variables that can hold several data items of the same kind.

- Similarly structure is another user defined data type available in C that allows to combine data items of different kinds.

- Structure is a **collection of different types of variables under single name**.

- It is a convenient tool for handling a group of logically related data items.
  - Examples:
    - Student name, roll number and marks.

Defining a Structure

- To define a structure, you must use the **struct** statement. **The struct statement defines a new data type, with more than one member.**
- The format of the struct statement is as follows −

```
struct tag_name {
        data _type member 1;
        data _type member 2;
        :
        data _type member m;
    };
```

  - ***struct*** **is the required keyword.**
  - **tag is the name of the structure.**
  - **member 1, member 2, … are individual member declarations.**

Contd.

- The individual structure elements (logically related data items) are called *members*.

- The individual members can be ordinary variables, pointers, arrays, or other structures.
  - The member names within a particular structure must be distinct from one another.
  - A member name can be the same as the name of a variable defined outside of the structure.

Defining a structure

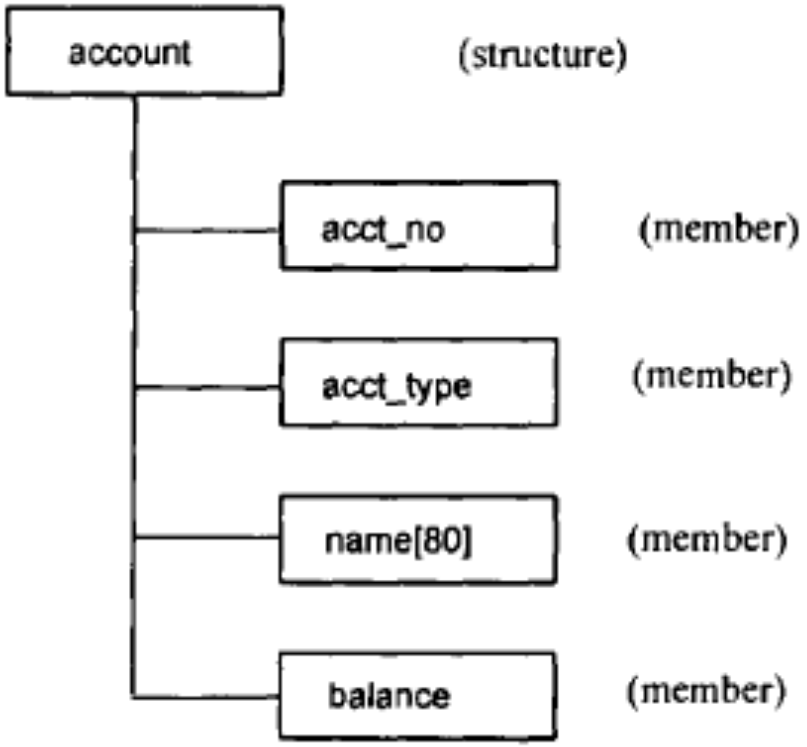
- **A structure definition:**

```
struct account {
                    int acct_no;
                    char acct_type;
                    char name[80];
                    float balance;
                    };
```

- This structure is named **account** (i.e., the tag is **account).**
- It contains four members: an integer quantity **(acct_no),** a single character **(acct_type),** an 80-element character array **(name [80]),**and a floating-point quantity **(balance).**

Defining a structure

- **Note- The above definition has not declared any variables. It simply describes a format called template to represent information.**



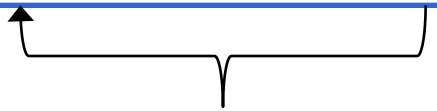
Declaring structure variable

- Once a structure has been defined, the individual **structure-type variables** can be declared as:

```
struct tag_name var_1, var_2, …, var_n;
```

- **Example:**

```
struct account a1, a2, a3;
```



A new data-type

It declares a1, a2, a3 as variables of type struct account.

Each one these variables has four members as specified by the template.

A Compact Form

- It is possible to combine the declaration of the structure with that of the structure variables:

```
struct tag {
            member 1;
            member 2;
            :
            member m;
          } var_1, var_2,…, var_n;
```

- In this form, “tag” is optional.

Equivalent Declarations

```
struct student {
                    char name[30];
                    int roll_number;
                    int total_marks;
                    char dob[10];
                  } a1, a2, a3;
```

```
struct  {
                    char name[30];
                    int roll_number;
                    int total_marks;
                    char dob[10];
          } a1, a2, a3;
```

Structure variable Declarations

```
struct Student
{
        int id;
        char name[32];
};
struct Student S1,S2,S3;
```

```
struct Student
{
      int id;
      char
name[32];
}S1,S2,S3;
```

```
struct
{
      int id;
      char
name[32];
}S1,S2,S3;
```

Important Points

- **Note- The members of structure themselves are not variables. They do not occupy any memory until they are associated with the structure variables.**

- **When the compiler come across structure variable declaration statement, it reserves memory space for structure variables.**

Structure variable Declarations

```
struct Student {
        int id;
        char name[32];
};      // until now no memory allocation
struct student S1,S2,S3; /*now memory is reserved for variables*/
```

Structure Initialization

- C language does not permit initialization of individual structure members within the template.

```
struct Point
{
   int x = 0;  // COMPILER ERROR:  cannot initialize members here
   int y = 0;  // COMPILER ERROR:  cannot initialize members here
};
```

- The initialization must be done only in the declaration of structure variables.

- Like primary variables and arrays, structure variables can also be initialized where they are declared. The format used is quite similar to that used to initiate arrays.

Structure Initialization

```
struct student
{
   char name[20];
   int roll;
   float marks;
} std1 = { "Pritesh",67,78.3 };
```

The order of values enclosed in braces must match the order of members in structure definition.

Structure Initialization

Declaring and Initializing Multiple Variables

```
struct book
    {
        char name[10] ;
        float price ;
        int pages ;
    } ;
    struct book b1 = { "Basic", 130.00, 550 } ;
    struct book b2 = { "Physics", 150.80, 800 } ;
```

There is one-to-one correspondence between the members and their initializing values.

Structure Initialization

Initializing inside main

```
struct student
{
    int mark1;
    int mark2;
    int mark3;
};
void main()
{
struct student s1 = {89,54,65};
- - - - --
- - - - --
};
```

Structure Initialization

Partial Initialization

```
struct student
{
   int mark1;
   int mark2;
   int mark3;
} sub1={67};
```

Data Type	Default value if not initialized
integer	0
float	0.00
char and string	'\0'

Though there are three members of structure, only one is initialized , Then remaining two members are initialized with **Zero**.

The uninitialized members should be only at the end of the list.

Accessing Structure Members

- The members of a structure are processed individually, as separate entities.
- To access any member of a structure, we use the **member access operator '.'** or also known as **'dot operator'**

- A structure member can be accessed by writing

  StructureVariable.member

  where StructureVariable *refers to the name of a structure-type* variable, and *member refers to the name of a member* within the structure.

Accessing Structure Members

Example:

```
struct Student
{
      int id;
      char name[32];
}S1,S2,S3;

S1.id=1234;
S1.name= “C Programming”;
scanf (“%d %s”, &S2.id, S2.name);
```

Example

```
struct Point
{
   int x, y;
};                    // structure template or definition
int main()
{
   struct Point p1; // structure variable declaration


    p1.x = 20; // Accesing members of point p1
    p1.y= 1;
   printf ("x = %d, y = %d", p1.x, p1.y);
   return 0;
}
```

Example: Complex number addition

```
#include <stdio.h>
void main()
{
        struct complex
        {
                float real;
                float cmplex;
        } a, b, c;
        scanf (“%f %f”, &a.real, &a.cmplex);
        scanf (“%f %f”, &b.real, &b.cmplex);
        c.real = a.real + b.real;
        c.cmplex = a.cmplex + b.cmplex;
        printf (“\n %f + %f ”, c.real, c.cmplex);
}
```

Structure within Structure : Nested Structure

- Structure written inside another structure is called as nesting of two structures.
- Nested Structures are allowed in C Programming Language.
- We can write one structure inside another structure as **member** of another structure.

Structure within Structure : Nested Structure

Way1: Declare two separate structures

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

struct Employee {
  char ename[20];
  int ssn;
  float salary;
  struct date doj;
 }emp1;
```

Accessing Nested Elements :

- Structure members are accessed using dot operator.
- 'date' structure is nested within Employee Structure.
- Members of the 'date' can be accessed using 'Employee'
- emp1 & doj are two structure names (Variables)

Structure within Structure : Nested Structure

Way1: Declare two separate structures

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

struct Employee {
 char ename[20];
 int ssn;
 float salary;
struct date doj;
 }emp1;
```

Accessing Nested Members :

- An inner-most member in a nested structure can be accessed by chaining all the concerned structure variables (***from outer-most to inner-most***) with the member using dot operator.
- Accessing month Member : **emp1.doj.month**
- Accessing day Member : **emp1.doj.day**
- Accessing year Member : **emp1.doj.year**

Structure within Structure : Nested Structure

Way 2 : Declare Embedded structures

```
struct Employee
 {
 char ename[20];
 int ssn;
 float salary;
struct date {
              int day;
              int month;
              int year;
               } doj;
 }emp1;
```

Accessing Nested Members :

- An inner-most member in a nested structure can be accessed by chaining all the concerned structure variables (***from outer-most to inner-most***) with the member using dot operator.
- Accessing month Member : **emp1.doj.month**
- Accessing day Member : **emp1.doj.day**
- Accessing year Member : **emp1.doj.year**

Nested Structure: Example

```
#include <stdio.h>
struct Employee
{
   char ename[20];
   int ssn;
   float salary;
         struct date
               {
                int day;
                int month;
                int year;
               } doj;

} emp = {  "Pritesh",  1000,
1000.50 , {  22,6,1990 }  };
```

```
void main()
{
printf("\n Employee Name   : %s",
emp.ename);

printf("\n Employee SSN    : %d",
emp.ssn);

printf("\n Employee Salary : %f",
emp.salary);

printf("\n Employee DOJ:  %d/ %d/
%d",  emp.doj.day,
emp.doj.month,  emp.doj.year);
}
```

Arrays of Structures

- Structure is used to store the information of one particular object but if we need to store such 100 objects then Array of Structure is used.
- **Each element of the array represents a structure variable**
- Once a structure has been defined, we can declare an array of structures.

```
struct student
{
   char name[20];
   int roll;
   float marks;
};
```

Arrays of Structures

```
struct student class[50];
```

defines an array called **class** that consists of **50 elements**. Each element is of type **struct student.**

Each element has three members.

– The individual members can be accessed as:

```
class[0].name
class[0].roll
class[0].marks
……..
class[49].name
class[49].roll
class[49].marks
```

Example

```
struct Point
{
   int x, y;
};
void main()
{
   struct Point arr[10]; // Create an array of structures

   arr[0].x = 10;  //   Access array members
   arr[0].y = 20;

    printf("%d %d", arr[0].x, arr[0].y);
}
```