SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with ‘A’ Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

COURSE NAME : 23ITT101- PROBLEM SOLVING & C PROGRAMMING

I YEAR /I SEMESTER

Unit V – STRUCTURE AND UNION

Topic : Structure

Topics Covered

- Structures and Unions
  - Introduction
  - Defining a Structure
  - Declaring Structure Variables
  - Accessing Structure Members

Structures and Unions - Introduction

- C supports a **constructed data type** known as **structures,** a mechanism for **packing data of different types.**
- It is a convenient tool for **handling a group of logically related data item.**

Structures and Unions - Introduction

Example:

Time - Seconds, minutes and hours

Date - Day, month and year

Book - Author, title, book and year

It is used to **organize complex data in a more meaningful way.**

Defining a Structure

- Unlike arrays, Structures must be defined first for their format that may be used later to declare structure variables.
- Consider a book database consisting of book name, author, number of pages and price.
- We can define a structure to hold this information as follows.

Defining a Structure

```
struct book_bank
{
char title[20];
char author[15];
int pages;
float price;
};
```

Defining a Structure

- The **keyword** **struct** declares a structure to hold the details of four data fields, namely title, author, pages and price are called **structure elements or members**.

book_bank – name of the structure or **structure tag**.

Here tag name used to declare variables that have the tag’s structure.

Defining 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.

Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book −

Title
Author
Subject
Book ID
Pages
Price

Defining a Structure

Each members may belong to a different type of data. The tag name may be used subsequently to declare variables that have the tag's structure

Title ------------→ Char array

Author ------------→ Char array

Subject -------------→ Char array

Book ID ------------→ Integer

Pages ----------------→ Integer

Price ------------------→ Float

Defining a Structure

General format of a structure:

```
struct  tag_name
{
data_type member 1;
data_type member 2;
---------
};
```

Example of a structure

```
struct Student
{
char name[25];
int age;
char branch[10];
 // F for female and M for male
char gender;
 };
```

- Here struct Student declares a structure to hold the details of a student which consists of 4 data fields, namely name, age, branch and gender. These fields are called **structure elements or members**.
- Each member can have different datatype, like in this case, name is an array of char type and age is of int type etc. **Student** is the name of the structure and is called as the **structure tag**.

Defining a Structure

- In defining a structure, note the following syntax:
- The template is **terminated with a semicolon**.
- While the entire definition is considered as a statement, **each member is declared independently** for its name and type in a separate statement inside the template.
- The tag name book-bank can be used to declare structure variables of its type.

Defining a Structure

Arrays

- Array is a collection of related data elements of same type.
- Array is a derived data type
- Array behaves like built-in data type. Here we have to declare an array variable and use it.

Structures

- Structure can have elements of different types.
- Structure is a programmer defined one.
- Here we have to design and declare a data structure before the variables of the type are declared and used.

Defining a Structure

Arrays Vs Structures

- Both the arrays and structure are classified as structured data types as they provide a mechanism that **enable us to access and manipulate the data** in easy manner, but they differ in the following manner.
- After defining a structure format we can declare variables of that type.
- A structure variable declaration is **similar to the declaration of variables of any other data type**.

Declaring Structure with variables

It includes the following elements.

- The **keyword** struct
- The structure **tag name**
- List of **variable names** separated by commas
- A terminating **semicolon**.

EXAMPLE:

```
struct book_bank book1, book2, book3;
```

Book1, book2 and book3 are variables of type struct book_bank.

Declaring Structure with variables

```
struct book_bank
{
 char title[20];
 char author[15];
 int pages;
 float price;
}
struct book_bank, book1, book2, book3;
```

Declaring Structure variables separately

```
struct Student
 {
char name[25];
int age;
char branch[10];
//F for female and M for male char gender;
};
struct Student S1, S2; //declaring variables of struct Student
```

Declaring Structure variables with structure definition

```
struct student
{
char name[25];
int age;
char branch[10];
//F for female and M for male char gender;
}S1, S2;
```

Here S1 and S2 are variables of structure Student. However this approach is not much recommended.

Declaring Structure with variables

- **Members** of a structure themselves are **not variables**.
- They **do not occupy any memory until they are associated with the structure variables**.
- When the **compiler comes across declaration** statement, it **reserves memory space** for the structure variables.
- It is also **allowed to combine both the structure definition and variables declaration** in one statement.

Declaring Structure with variables

- The use of **tag name is optional**.
- **Without a tag name** we **cannot use it for future declarations.**
- **Structure definitions** appear at a **beginning of the program** file before any variable or functions are defined.
- In such cases the **definition** is **global and can be used by other functions as well.**

Declaring Structure with variables

Type - Defined Structure:

Use keyword typedef to define a structure,

```
typedef struct
{ …...
type number 1;
type number 2;
} type_name;
```

Declaring Structure with variables

type_name represents structure definition associated with it and it is used to declare structure variables.

type_name variable1, variable2, ..;

1. Type_name is a the type definition name, not a variable
2. We cannot define a variable with typedef declaration.

Accessing structure members

- We can access and assign values to the members of a structure in a number of ways.
- Members are not variables
- Structure members have no meaning individually without the structure.
- Here the link between member and a variable is established using a **member operator '.'** which is also known as '**dot operator**' or **period operator**.

Accessing structure members

Example:

```
book1.price;
book1.title;
book1.author;
book1.pages ;
```

Accessing structure members

```
#include<stdio.h>
struct Point
{
int x, y;
};
int main()
{
struct Point p1 = {0, 1};
// Accessing members of point p1
p1.x = 20;
printf ("x = %d, y = %d", p1.x, p1.y);
return 0;
 }
```

Accessing structure members

```
#include<stdio.h>
#include<string.h>
struct Student
{
char name[25];
int age;
char branch[10];
//F for female and M for male
char gender;
};

int main()
{
struct Student s1;
/*s1 is a variable of Student type and
age is a member of Student */
s1.age = 18;
/* using string function to add name */
strcpy(s1.name, "Arun");
/* displaying the stored values */
printf("Name of Student 1: %s\n", s1.name);
printf("Age of Student 1: %d\n", s1.age);
return 0;
}
```

Accessing structure members

```
#include <stdio.h>
#include <string.h>
struct Books {
char title[50];
char author[50];
char subject[100];
int book_id;
};
int main( ) {

struct Books Book1;
/* Declare Book1 of type Book */
struct Books Book2;
 /* Declare Book2 of type Book */

/* book 1 specification */
strcpy( Book1.title, "C Programming");
strcpy( Book1.author, "Nuha Ali");
strcpy( Book1.subject, "C Programming Tutorial");
Book1.book_id = 6495407;

/* book 2 specification */
strcpy( Book2.title, "Telecom Billing");
strcpy( Book2.author, "Zara Ali");
strcpy( Book2.subject, "Telecom Billing Tutorial");
Book2.book_id = 6495700;
```

```
/* print Book1 info */
printf( "Book 1 title : %s\n", Book1.title);
printf( "Book 1 author : %s\n", Book1.author);
printf( "Book 1 subject : %s\n", Book1.subject);
printf( "Book 1 book_id : %d\n", Book1.book_id);

/* print Book2 info */
printf( "Book 2 title : %s\n", Book2.title);
printf( "Book 2 author : %s\n", Book2.author);
printf( "Book 2 subject : %s\n", Book2.subject);
printf( "Book 2 book_id : %d\n", Book2.book_id);

return 0;
}
```

Output::

```
Book 1 title : C Programming
Book 1 author : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id : 6495407
Book 2 title : Telecom Billing
Book 2 author : Zara Ali
Book 2 subject : Telecom Billing Tutorial
Book 2 book_id : 6495700
```

Summary

- C supports a constructed data type known as structures, a mechanism for packing data of different types.
- Structures must be defined first for their format that may be used later to declare structure variables.
- A structure variable declaration is similar to the declaration of variables of any other data type.

Thank you