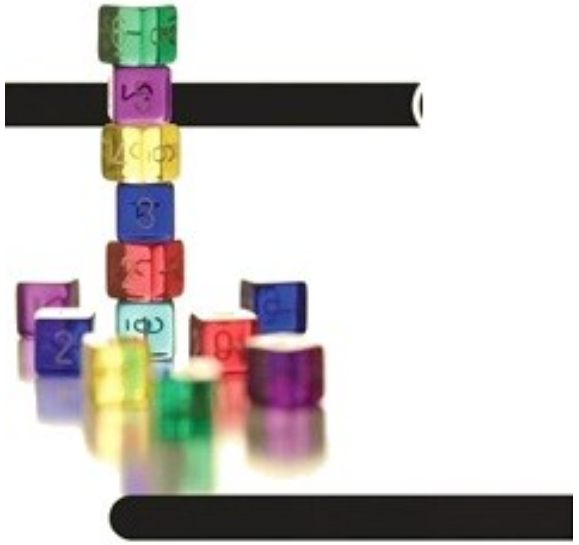


S21_1 Structures

Objectives

To learn and appreciate the following concepts

- Basic operations and programs using structures
- Advantages of structures over array

Session outcome

At the end of session one will be able to

1. Understand the overall ideology of structures
2. Write programs using structures

Introduction

- ***W***e've seen variables of simple data types, such as float, char, and int.

➢ ***W***e saw **derived data** type, **arrays** to store group of related data.

- Variables of such types represent one item of information: a height, an amount, a count, or group of item with same data type: list[10] and so on.

- But, these basic types does not support the storage of compound data.

  Eg. **Student** {name, address, age, sem, branch}

Introduction

- C provides facility to define one's own type (user-defined) that may be a composite of basic types (**int, char, double**, etc) and other user-defined types.

  ✓**Structures**

Introduction

- Definition:
  - collection of *one or more* variables, *possibly of different types*, grouped together under a *single name* for convenient handling

- A structure type in C is called struct.

Structures

- Structures hold data that belong together.
- Examples:
  - ✓Student record: student id, name, branch, gender, start year, …
  - ✓Bank account: account number, name, address, balance, …
  - ✓Address book: name, address, telephone number, …
- In database applications, structures are called records.

Structure *versus* Array

- A struct is heterogeneous, that means it can be composed of data of different types.

- In contrast, array is homogeneous since it can contain only data of the same type.

Structure Definition - Syntax

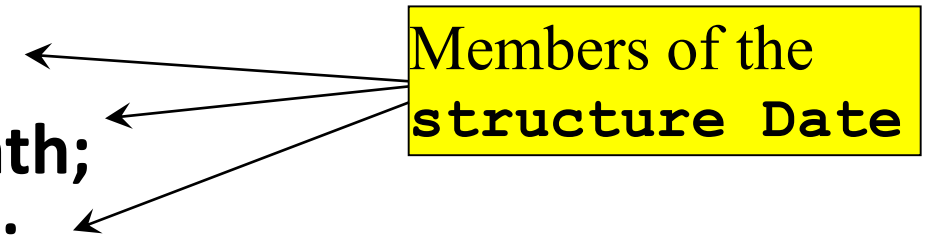
The general format of a structure definition

```
struct  structure_name
{
  data_type member1;
  data_type member2;
  …
};
```

Structure Definition - Examples

- Example:

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



Structure examples

More examples:

i)
```
struct StudentInfo{
        int Id;
        int age;
        char Gender;
        double CGA;
  };
```

The “StudentInfo” structure has 4 members of different types.

ii)
```
struct Employee{
        char Name[15];
        char Address[30];
        int Age;
        float Basic;
        float DA;
        float NetSalary;
  };
```

The “Employee” structure has 6 members

Important Points Regarding Structures

- **Definition** of Structure reserves **no space**.
- It is nothing but the " **Template / Map / Shape** " of the structure .
- Memory is created, very first time when a **variable of structure type is created** / **Instance** is created.

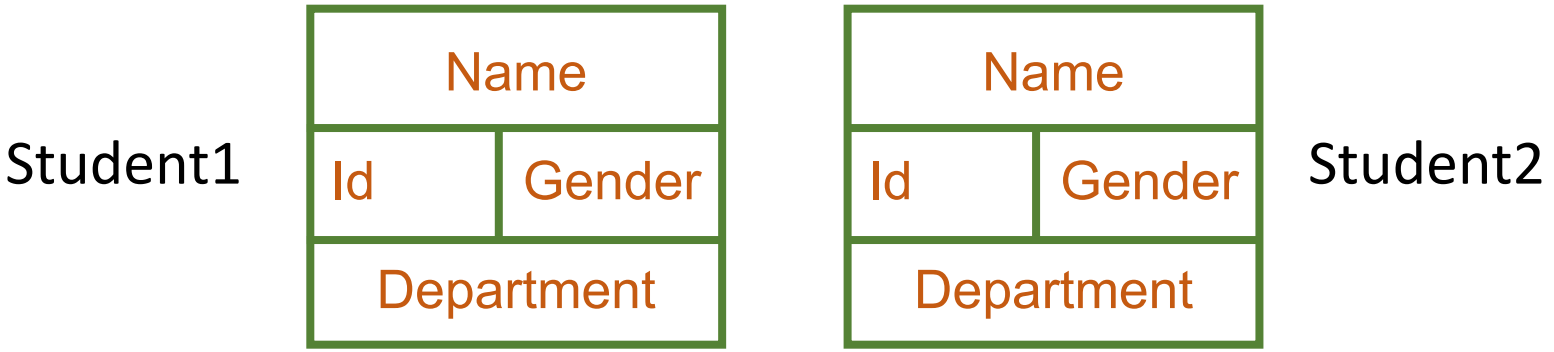
Declaring Structure Variables

Declaration of a variable of **struct** type using **struct** tag name, after structure definition:

<*struct-type*> <identifier_list>;

- Example:

StudentInfo **Student1, Student2;**



Student1 and Student2 are variables of *StudentInfo* type.

Declaring Structure Variables

Declare them at the time of structure definition:

```
struct student
{
    int rollno;
    int age;
    char name[10];
    float height;
}s1, s2, s3;  /* Defines 3 variables  of type student */
```

Members of a structure themselves are not variables. i.e. **rollno** alone does not have any value or meaning.

Member or dot operator

- The link between member and a structure variable is established using the **member operator '.'** which is also known as '**dot operator**'

  <struct-variable>.<member_name>

e.g.: student s1; // s1 is a variable of type student structure.

s1. rollno;

s1. age;

s1. name;

s1.height;

```
struct student
{
    int rollno;
    int age;
    char name[10];
    float height;
}
```

Example of Same Member Names in Different structures

```
struct fruit {
   char  name[15];
   int   calories;
};
```

```
struct vegetable {
      char name[15];
      int calories;
   }
```

```
struct fruit a;
```

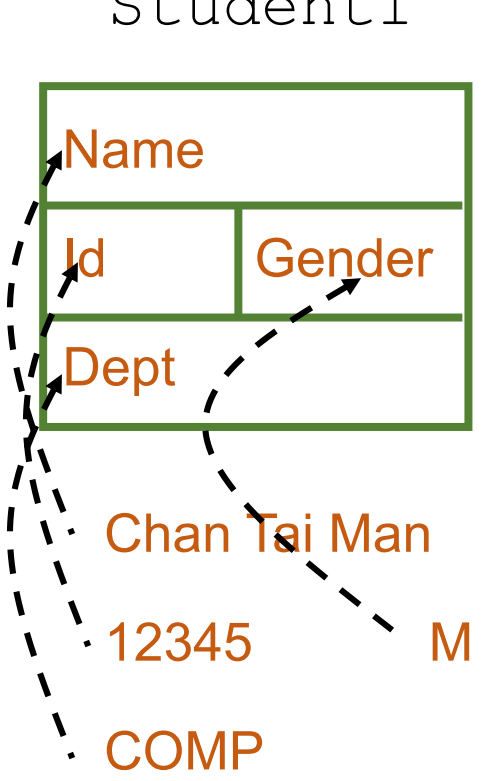
```
struct vegetable  b;
```

We can access a.calories and b.calories without ambiguity.

Ex: Member accessing using dot operator

- Example:

```
StudentRecord Student1; //Student1 is a variable of type StudentRecord
strcpy(Student1.Name, "Chan Tai Man");
Student1.Id = 12345;
strcpy(Student1.Dept, "COMP");
Student1.gender = 'M';
printf("The student is “);
switch (Student1.gender){
      case 'F': cout << "Ms. "; break;
      case 'M': cout << "Mr. "; break;
}
printf(“%s \n”, Student1.Name);
```



Assigning values to members

Different ways to assign values to the members of a structure:

Assigning string:

strcpy(s1.name, "Rama");

Assignment statement:

s1.rollno = 1335;

s1.age = 18;

s1.height = 5.8;

```
struct student
  {
  int rollno;
  int age;
  char name[20];
  float height;
  }s1;
```

Reading the values into members:

scanf("%s %d %f %f", s1.name, &s1.age, &s1.rollno, &s1.height);

Omission of the Tag Name

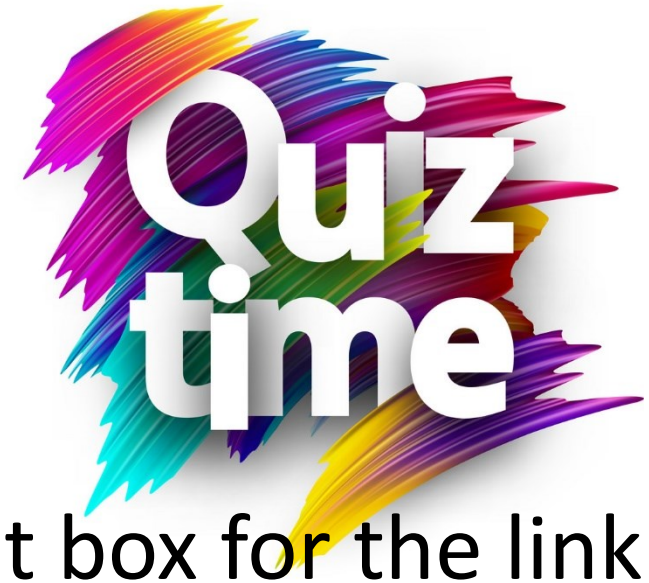
```
struct {                    /* Since no tag name is   */
  char *last_name;                /* used, no variables can */
  int  student_id;                /* be declared later in   */
  char grade;               /* the program.           */
} s1, s2, s3;

struct student {                    /* Variables can now be   */
  char *last_name;                  /*  declared later in   */
  int  student_id;                  /* the program as shown   */
  char grade;               /* below. */
};

struct student   temp, class[100];
```

Summary

- Structure Basics
- Member accessing using dot operator
- Simple problems using structures



Go to posts/chat box for the link to the question

submit your solution in next 2 minutes

The session will resume in 3 minutes