Lecture 08: Structures

Today’s Lecture

- ❑ Need for Structure
- ❑ Declaring a Structure in c++
- ❑ Creating structure variables
- ❑ Initializing structure variables
- ❑ Operations on structures (accessing structure’s members)
- ❑ Nested Structures
- ❑ Array as Member of Structure
- ❑ Array of structures

Structures

- So far we used built-in data types such as Int, char, double, arrays etc.,
- However many times we need more than one variable to represent something of interest.
- **Struct** is an **aggregate data type** (also called an **aggregate**) that can contain multiple data members which allows members to have different types
- Array is another example of aggregate data type which require that all members must be of a single type.

Problems with independent data types

- We want to represent date as year/month/day:

```
int year;
int month;
int day;
```

- There are a number of problems with this approach:
  - The problem is that there is no logical connection between them. • There are now three members to manage (which could be more in case of Employee, Student etc.)
  - If we want to pass time’s information to a function then we have to pass 3 variables and that too in order which will create a mess.
  - Returning more than one values from the function is also not possible. • And if we want to create several ‘dates’, we need to create 3 variables for every one of it. Scalability problem.

Structures

- C++ allows us to create our own user-defined aggregate data types, ***struct***.

- ***Struct*** (short for structure) is a collection of different data types under the same name and single type.

- Structs make the management of related sets of variables much simpler.

Example of usage of structures

- For some applications, we need data structures to store record, for example, of a student, teacher, employee or a product etc.

- we can define a data structure to describe a group of related data, such as a "record" in a file. e.g.

- *Student record (definition)*

ID Number Family Name Given Names Date of Birth

- Example (content of such a record)

11112222 “Kennedy" "John Andrew" "12/04/1989"

Declaring Structures in C++

- We can declare the structure by using the **struct** keyword •

The general syntax of a struct in C++ is given below:

```
struct <structName>
{
        <datatype> <memberName1>;
        <datatype> <memberName2>;
        <datatype> <memberName3>;
        ......
};
```

Example: Declaring a C++ struct

- Structure is defined before the main program and the structure variable is then defined in main program

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

structure **name**

members *of the structure*

(sometimes called "fields")

- This merely *declares a new data type* called ***Date***. You can then use it to create further variables of type Date.

Important:- Date is not a variable. There is no memory allocated for it. It is merely a *type* (like int, float, etc).

Defining a Structure Variable

Syntax :-

<structName> <variableName>;

Examples:

Date birthday;

- creates a variable called *birthday* of type *Date*. This variable has 3 *components* (members) : **day**, **month**, and **year**.
- We can also create multiple variables of type date. For example,

Date admissiondate;

Date graduationdate;

- creates another variable of type **Date**, also with component parts called **day**, **month** and **year**.

Defining a Structure Variable Vs Defining a "normal" Variable

```
int number;
Date birthday;
```

NAME **of the variables**

TYPE **of the variables**

note the consistent format :

```
<type> <variableName>;
```

Employee struct: Complete Example

```
struct Employee
```

```
  {
    int id;
    int age;
    double                    }
  salary; };
  int main() {
  Employee asif;


  Return 0;
```

the name ***asif*** refers to the entire struct (which contains the member variables).

Employee struct: Accessing Specific member variables

- To access a specific member variable, we use the member selection operator (operator.) in between the struct variable name and the member name, e.g., asif.age or asif.id.

```
struct Employee
 {
   int id;
   int age;
   double salary;
 };
int main() {
Employee asif;
asif.age = 30;

cout<<asif.age;
Return 0;
 }
```

Creating structure of Library Database

Data warehousing	Stephen Brobst

Network Security	Martin
Data mining	Muhammad Za

1293 1998 9382 2003 9993 2003 3423 C Programming M. Kamber Waley 4
1996

Creating structure of Library Database

Network Security	Martin
Data mining	Muhammad Za
Data warehousing	Stephen Brob

1293 1998 9382 2003 9993 2003 3423 C Programming M. Kamber Waley 4
1996

```
struct library
{
   int ISBN;
   int copies;
   int PYear;
   char bookName[30];
   char AuthorName[30];
   char PublisherName[30];
};
```

OPERATIONS ON STRUCTURES (ACCESSING STRUCTURE'S MEMBERS)

Accessing Structure Members

- Struct member variables work just like normal variables, so it is possible to do normal operations on them, including assignment, arithmetic, comparison, etc…

```
Library libraryVariable;
cin >> libraryVariable.ISBN;
cin >> libraryVariable.bookName;
cin >> libraryVariable.AuthorName;

cout << libraryVariable.ISBN << libraryVariable.bookName <<
libraryVariable.AuthorName;

int tempISBN = libraryVariable.ISBN + 1;
```

Common Errors in Accessing

Structures

Library libraryVariable; //define a struct variable.

Okay.

```
cout << bookName;
```

Error! // bookName is not a variable. It is only the name of a member in a structure

```
cout << Library.bookName;
```

Error! // Library is not the name of a variable. It is the name of a type

Common Errors in Accessing Structures

(contd.)

Library libraryVariable; //define a struct variable. Okay.

```
cout << libraryVariable;

//cout does not know how to handle the variable libraryVariable, as it is not
one of the built-in types. You have to give it individual bits of
libraryVariable that it can recognize and handle.


cout << libraryVariable.ISBN <<
 libraryVariable.bookName; //this is OK
```

Accessing Structure Variables (Example 1)

```
void main (void)
{
      struct Library
      {
             int ISBN, copies, PYear;
             char bookName[30], AuthorName[30], PublisherName[30];
      };

Library CSlibrary;
```

```
 CSlibrary.ISBN = 1293;
strcpy (CSlibrary.bookName, “Network Security”);
strcpy (CSlibrary.AuthorName, “Martin”);
strcpy (CSlibrary.PublisherName, “Waley”);
 CSlibrary.copies = 4;
 CSlibrary.PYear = 1998;

    cout << CSlibrary.ISBN << CSlibrary.bookName << CSlibrary.AuthorName <<
CSlibrary.PublisherName << CSlibrary.copies << CSlibrary.PYear; }
```

Assignment to Structure Variable

- The value of a structure variable can be assigned to another structure variable *of the same type*, e.g :

```
Library CSlibrary, EElibrary;
strcpy (CSlibrary.bookName , “CPP Programming”);
CSlibrary.ISBN = 1293;
```

- Now assign one struct variable to another using ‘=‘ operator.

```
EElibrary = CSlibrary;
cout << EElibrary.bookName << EElibrary.ISBN;
```

- Assignment is the only operation permitted on a structure. We can not add, subtract, multiply or divide structures.

Struct Aggregate Initialization

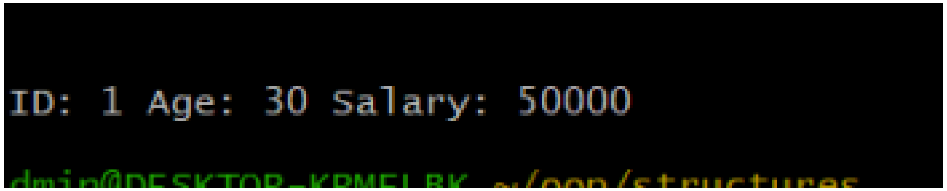
- Struct members are not initialized by default.
- When we create a variable ‘***asif***’ and do not initialize it then printing asif.age will give undefined behavior.

```
struct Employee
 {
    int id;
    int age;
    double salary;
 };
int main() {
Employee asif;
cout<<asif.age;
Return 0;
 }
```

Struct Aggregate Initialization

- Aggregate initialization allows us to directly initialize the members of struct variable.
- initialization does a memberwise initialization, which means each member in the struct is initialized in the order of declaration.

```
struct Employee
 {
    int id;
    int age;
   double salary;
 };
int main() {
Employee asif = {1, 30, 50000};
```



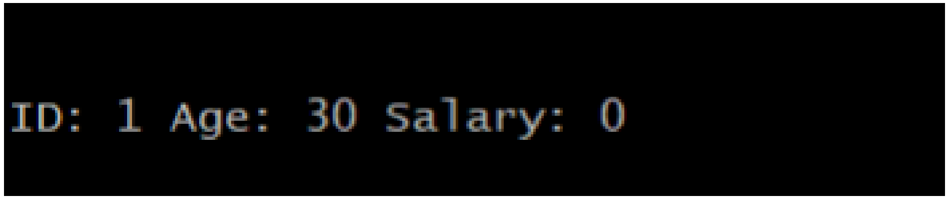
```
ID: 1 Age: 30 Salary: 50000

dmin@DESKTOP-KPMELBK ~/oop/structures
```

```
cout<<"ID: "<<asif.id<<"Age: "<<asif.age<<"Salary:
 "<<asif.salary<<endl;
return 0;
}
```

Missing initializers in an initializer list

- If the number of initialization values is fewer than the number of members, then all remaining members will be initialized to 0.
- In the below example, asif.id is initialized to 1, asif.age is initialized to 30 and because asif.salary is not given an explicit initializer, it will be initialized to 0.

```
struct Employee
{
  int id;
  int age;
  double salary;
};
int main() {
Employee asif = {1, 30};
```



```
ID: 1 Age: 30 Salary: 0
```

```
cout<<“ID: “<<asif.id<<“Age: “<<asif.age<<“Salary:
 “<<asif.salary<<endl;
return 0;
}
```

NESTED STRUCTURES (EMBEDDED STRUCTS)

Structures within Structures

```
struct Library {
       int ISBN, copies, PYear;
       char bookName[30], AuthorName[30],
   PublisherName[30]; };
struct University {
  char Name [30];
  char city [30];
  Library CSlibrary;
   };
```

```
void main () {
  University FAST;
   strcpy (FAST.Name, “CFD”);
   strcpy (FAST.city, “Chiniot”);
   FAST.CSlibrary.ISBN = 1293;
   strcpy (FAST.CSlibrary.bookName, “CPP programming”);
 }
```

Accessing Structure in Structure

```
cin >> FAST.CSlibrary.bookName;
cin >> FAST.CSlibrary.ISBN;

cout << FAST.CSlibrary.bookName << FAST.CSlibrary.ISBN;
```

Nested structures: Another Example

- A nested structure occurs when a structure is a member of a structure.

In the following example the structure `addr` is nested inside `emp`:

```
struct addr {
char name[40];
char street[40];
char city[40];
char zip[7];
};

struct emp { //outer struct
addr address; //inner struct
float wage;
} worker;

worker.address.zip = "T2N3F4";
```

USE OF TYPEDEF WITH A STRUCTURE (IN C)

`typedef`

- `typedef` (stands for type definition) declaration introduces new name or creates synonym (or alias) for existing type.
- To construct shorter or more meaningful names for types already defined by the language or for types that you have declared.
- The syntax is,

```
typedef type-declaration the_synonym;
```

- The following paragraphs illustrate other `typedef` declaration examples,

```
// a char type using capital letter
typedef char CHAR;
// a pointer to a string (char *)
typedef CHAR * PTRCHAR;
// then use it as function parameter
PTRCHAR strchr(PTRCHAR source, CHAR destination);

typedef unsigned int uint;
// equivalent to 'unsigned int ui';
uint ui;
```

Use of typedef with Structure

- When we use `typedef` (in C) with structure then it creates the alias of the structure.

▪ There is no need to write struct keyword every time with a variable declaration ▪ Names for structure types are often defined with `typedef` to create shorter and simpler type name.

▪ For example, the following definition.

```
Typedef struct {
            char Name [30];
            char city [30];
 }sUniversity;
```

creates the structure type sUniversity.

▪ Now, the following declaration,

```
  sUniversity FAST;
```

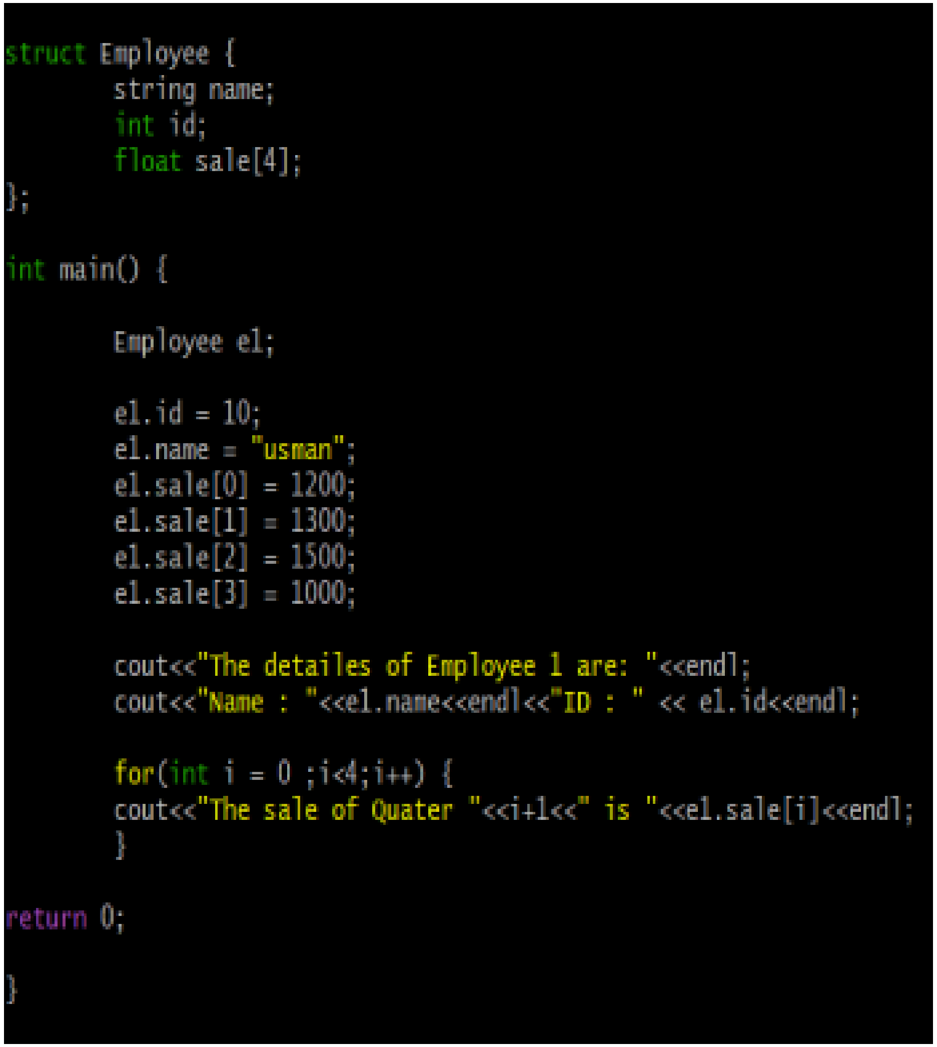
declares a variable of type `University` structure.

ARRAY AS MEMBER OF STRUCTURE

Arrays in Structure

- Structures can have arrays as there members
- In the given example there is an array of size 4 which will store the sale of the employee •

The array members of structures are accessed in the same manner as we access the elements of the array



```
struct Employee {
        string name;
        int id;
        float sale[4];
};

int main() {

        Employee e1;

        e1.id = 10;
        e1.name = "usman";
        e1.sale[0] = 1200;
        e1.sale[1] = 1300;
        e1.sale[2] = 1500;
        e1.sale[3] = 1000;

        cout<<"The detailes of Employee 1 are: "<<endl;
        cout<<"Name : "<<e1.name<<endl<<"ID : " << e1.id<<endl;

        for(int i = 0 ;i<4;i++) {
        cout<<"The sale of Quater "<<i+1<<" is "<<e1.sale[i]<<endl;
        }

return 0;

}
```

ARRAY OF STRUCTURES

Arrays of Structures

- Declaring an array of structure is same as declaring an array of fundamental types.
- Since an array is a collection of elements of the same type, in an array of structures, each element of an array is of the structure type. Let’s revisit the student’s structure example:

```
struct student
{
    char name[20];
    int roll_no;
    float marks[5];
};
```

- we can declare an array of **struct student**:

```
struct student arr_student[10]
```

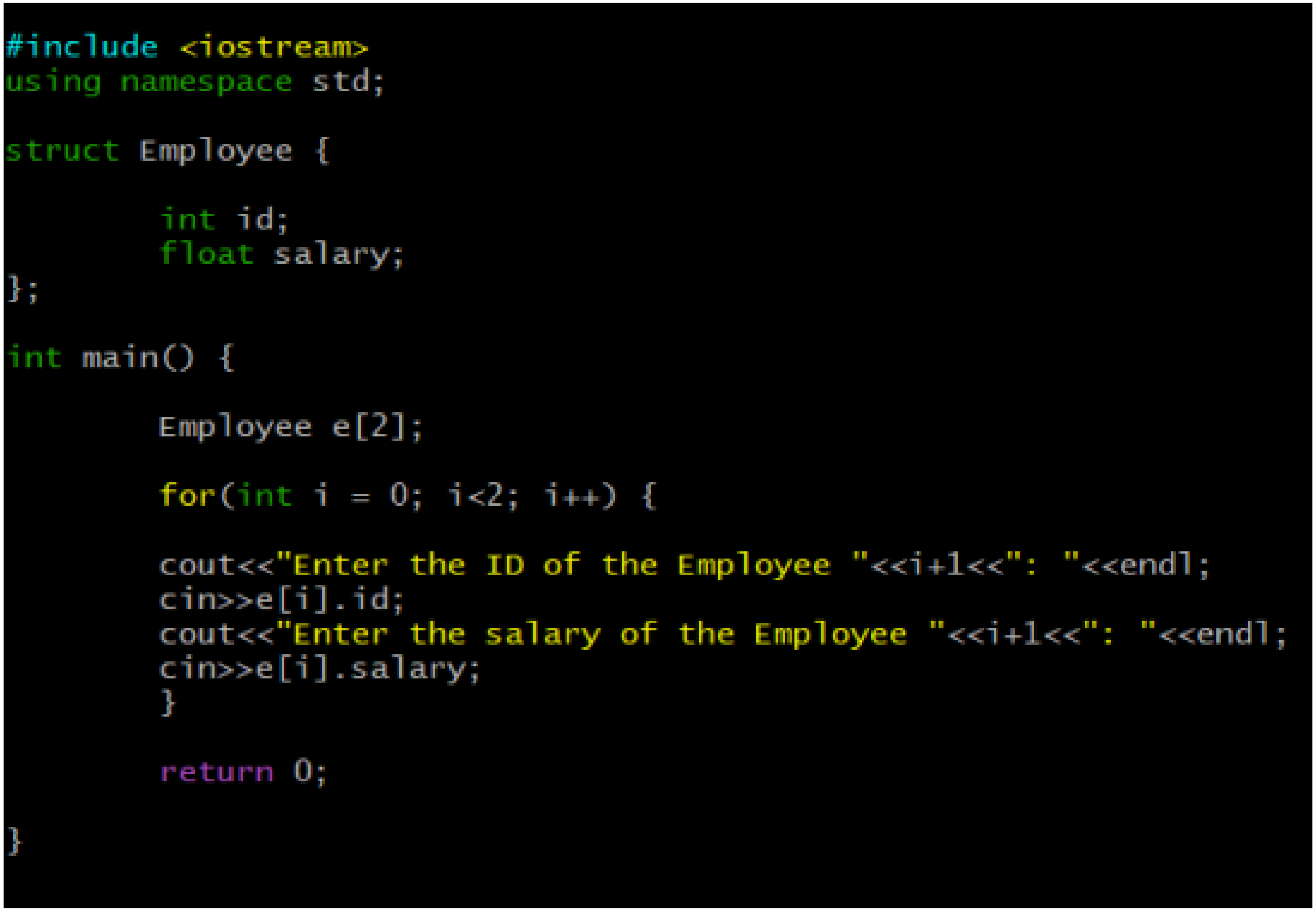
Accessing members of the Arrays of type struct student

arr_student[0].marks[0] - refers to the marks of first student in the first subject

arr_student[1].marks[2] - refers to the marks of the second student in the third subject and so on.

Arrays of type structures: another example

- We create an array of structures of type employee.
- We then assign values to each element of the array .



```
#include <iostream>
using namespace std;

struct Employee {

        int id;
        float salary;
};

int main() {

        Employee e[2];

        for(int i = 0; i<2; i++) {

        cout<<"Enter the ID of the Employee "<<i+1<<": "<<endl;
        cin>>e[i].id;
        cout<<"Enter the salary of the Employee "<<i+1<<": "<<endl;
        cin>>e[i].salary;
        }

        return 0;

}
```