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.,
- Structures are used to create new data types.
- But why would we need to create new data types?
  - We will find that in a moment.

Example

```
// We want to represent time as year/month/date:
int f ()
{
  int year1, year2, month, date;

  year1 = 2050;
  year2 = 2020;
  month = 12;
  date = 30;
  date++; // Should we increase year1 or year2?
}

// The problem is that there is no logical
// connection between them. We need “structure”!
```

Need for new user types

- ❑ For some applications, we need data structures to store record, for example, of a student, teacher, 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	**"Citizen"**	**"John Andrew"**	**"12/04/1989"**

Declaring Structures in C++

Syntax :-

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

Example: Declaring a C++ struct

```
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 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:

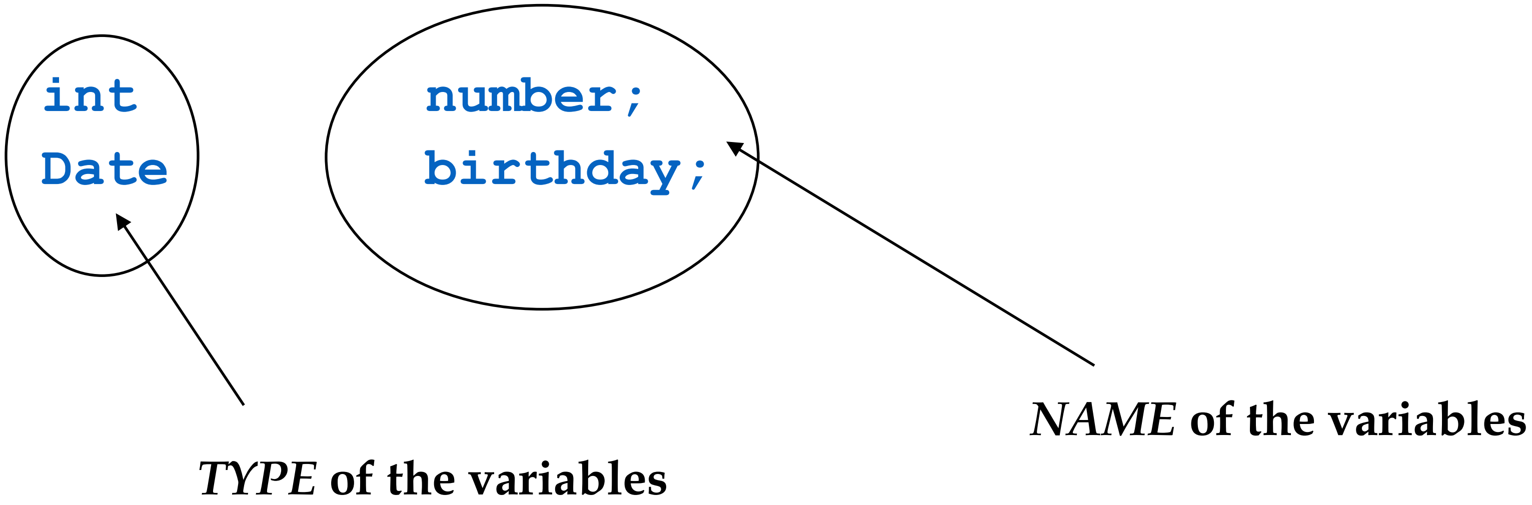
```
Date birthday;
```

- creates a variable called **birthday** of type **Date**. This variable has 3 *components* (members) : **day**, **month**, and **year**.

```
Date today;
```

- 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;
```

note the consistent format :

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

Initializing Structure Type Variables

```
struct Name
{
    char first[30];
    Char last[30];
};

Name poet_name; //create a variable of type Name
strcpy(poet_name.first,”Mirza”);
strcpy(poet_name.last,”Ghalib”);
```

Note :
Values of the members need to be copied individually, AFTER the variable is created.

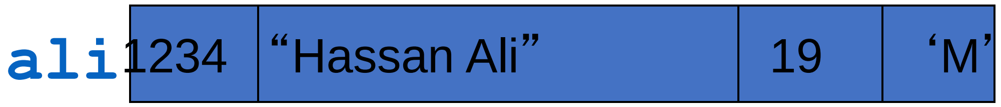
"Mirza"	**"Ghalib"**

Members of Different Types

```
struct Student
{
    ... id;
    ... name;
    ... age;
    ... gender;
};
student std;
std.id = 1234;
strcpy(std.name,”Hassan Ali”);
std.age = 19;
std.gender = 'M';
```

***The members of a* struct**
***need not be of the same type*.**

What should be the types of these members?



Creating structure of Library Database

ISBN	Book Name	Author Name	Publisher	Number of Copies	Year of Publish
1293	Network Security	Martin	Waley	4	1998
9382	Data mining	Muhammad Zaki	Wrox	6	2003
9993	Data warehousing	Stephen Brobst	MIT	8	2003
3423	C Programming	M. Kamber	Waley	4	1996

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

OPERATIONS ON STRUCTURES (ACCESSING STRUCTURE'S MEMBERS)

Accessing Structure Members

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

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

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

The dot is called the "member" operator

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.)

```
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.

NESTED STRUCTURES

Structures within Structures

```
void main ()
{
    struct University
    {                       char Name [30];
                            char city [30];
                            Library CSlibrary;
    };
    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;
```

Array as member of structure

Arrays inside Structures

- we have already been using arrays as members inside structures. Consider the student struct;

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

- The student structure defined above has a member name which is an array of 20 characters.

Arrays inside Structures

- Let's create another structure called student to store name, roll no and marks of 5 subjects.

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

- If student_1 is a variable of type struct student then:
  - student_1.marks[0] - refers to the marks in the first subject
  - student_1.marks[1] - refers to the marks in the second subject

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.

Homework#1 (submit by 1st April)

- Create a student structure, whose members are Name (a char array), roll_number, marks (an array of type float having size 5), major (a char array, to show the major of the student),
    - There shall also be a nested structure of type ***date*** struct inside the student structure, for the birthdate and registration date.
- Now first create a student variable named CSStudent. Fill up all the fields (members) with some random values from the console using "**cin**"
- Secondly create another student variable named EEStudent.
- Assign CSStudent to EEStudent.
- Show the values of the members of both struct variables using cout.