Chapter 2

Structure, Unions and Enumerations

Fundamentals of Programming in C++ (CoSc 2031)

Topics

Topics	Subtopics
2: Structure, Unions and Enumerations	2.1. Defining and Referencing Structure Members 2.2. Initializing Structure Members 2.3. Array of Structures 2.4. Nesting Structures 2.5. Defining and Referencing Unions 2.6. Defining and Referencing Enumerations 2.7. Defining User Defined Data Types using typedef

Structures

- A Structure is a collection of related data items, possibly of different types.
- A structure allows you to wrap one or more variables that may be in different data types into one.
- It can contain any valid data type like int ,char, float, array, pointer or even structures.
- A structure type in C++ is called struct.
- A struct is heterogeneous in that it can be composed of data of different types.
- In contrast, array is homogeneous since it can contain only data of the same type.

Structures

- A Structure is a collection of related data items, possibly of different types.
- A structure allows you to wrap one or more variables that may be in different data types into one.
- It can contain any valid data type like int ,char, float, array, pointer or even structures.
- A structure type in C++ is called struct.
- A struct is heterogeneous in that it can be composed of data of different types.
- In contrast, array is homogeneous since it can contain only data of the same type.

Structures

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

Structures

- Individual components of a struct type are called members (or fields).
- Members can be of different types (simple, array or struct).
- A struct is named as a whole while individual members are named using field identifiers.
- Complex data structures can be formed by defining arrays of structs.

struct basics

- Definition of a structure:
- To define a structure, you use the struct keyword.
- struct struct_name{ structure_member }; or

```
struct <struct-type>{
        <type> <identifier_list>;
        <type> <identifier_list>;
        ...
} ;
```

Each identifier defines a member of the structure.

- Example:

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

The "Date" structure has 3 members, day, month & year.

struct examples

- Example:

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

The "StudentInfo" structure has 4 members of different types.

- Example:

```
struct StudentGrade{
    char Name[15];
    char Course[9];
    int Lab[5];
    int Homework[3];
    int Exam[2];
};
```

The "StudentGrade" structure has 5 members of different array types.

struct examples

- Example:

```
struct BankAccount{
     char Name[15];
     int AcountNo[10];
     double balance;
     Date Birthday;
};
```

$\}$ The “BankAcount” structure has simple, array and structure types as members.

- Example:

```
struct StudentRecord{
     char Name[15];
     int Id;
     char Dept[5];
     char Gender;
};
```

$\}$ The “StudentRecord” structure has 4 members.

struct basics

- Declaring a structure
- You can declare structure variables together with the structure defintion:
- struct struct_name {
  structure_member;
  ... } instance_1,instance_2 instance_n;

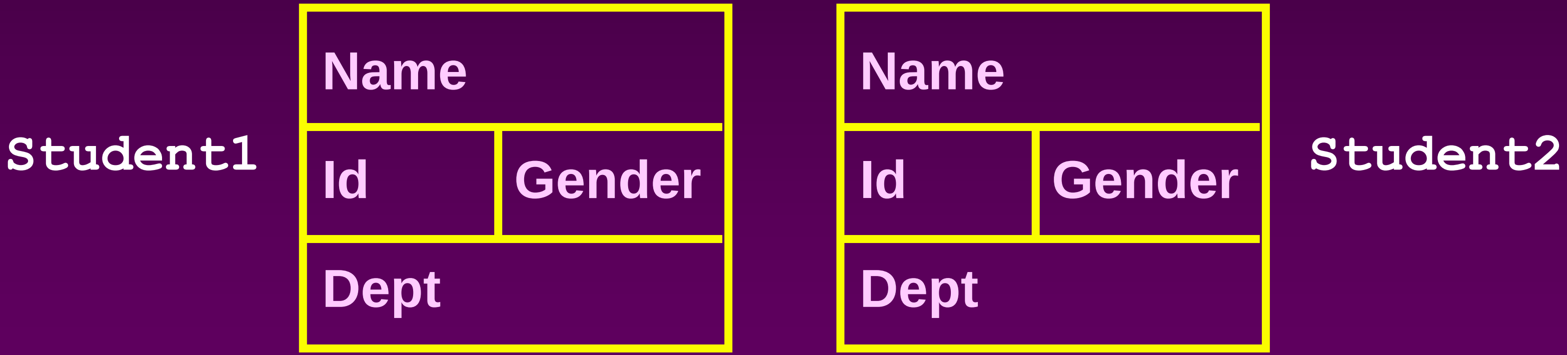
Or, you can declare the structure variable after you define the structure:

struct struct_name instance_1,instance_2 instance_n;

struct basics

- Example:

```
StudentRecord Student1, Student2;
```



`Student1` and **`Student2`** are variables of **`StudentRecord`** type.

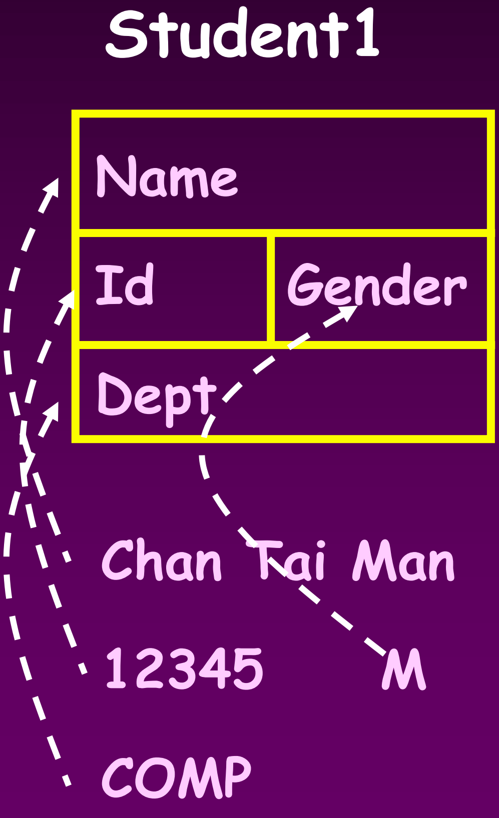
Accessing a structure member

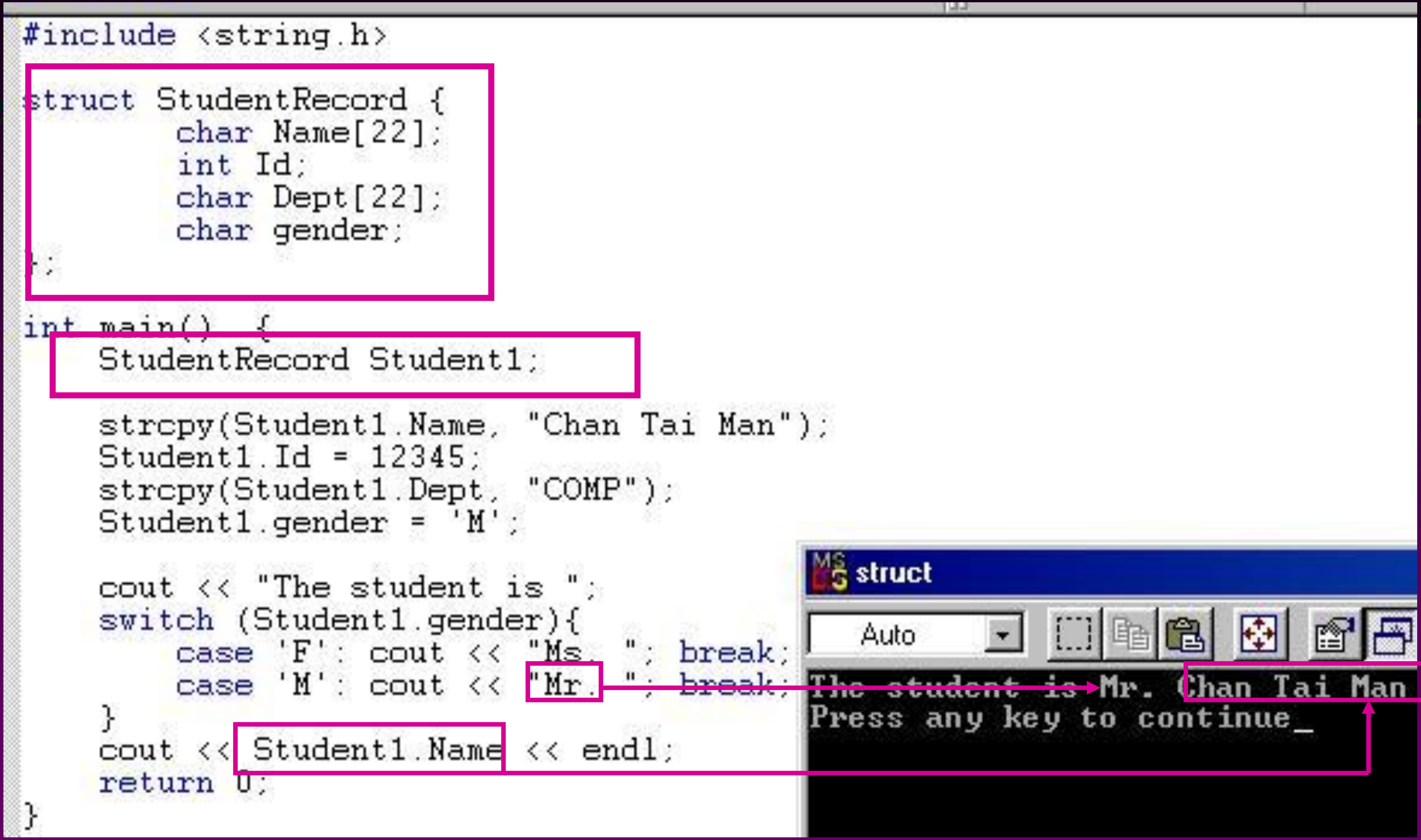
- The members of a struct type variable are accessed with the dot (.) operator:

```
<struct-variable>.<member_name>;
```

- Example:

```
strcpy(Student1.Name, "Chan Tai Man");
Student1.Id = 12345;
strcpy(Student1.Dept, "COMP");
Student1.gender = 'M';
cout << "The student is ";
switch (Student1.gender){
      case 'F': cout << "Ms. "; break;
      case 'M': cout << "Mr. "; break;
}
cout << Student1.Name << endl;
```





```
#include <string.h>

struct StudentRecord {
        char Name[22];
        int Id;
        char Dept[22];
        char gender;
};

int main()  {
    StudentRecord Student1;

    strcpy(Student1.Name, "Chan Tai Man");
    Student1.Id = 12345;
    strcpy(Student1.Dept, "COMP");
    Student1.gender = 'M';

    cout << "The student is ";
    switch (Student1.gender){
        case 'F': cout << "Ms. "; break;
        case 'M': cout << "Mr. "; break;
    }
    cout << Student1.Name << endl;
    return 0;
}
```

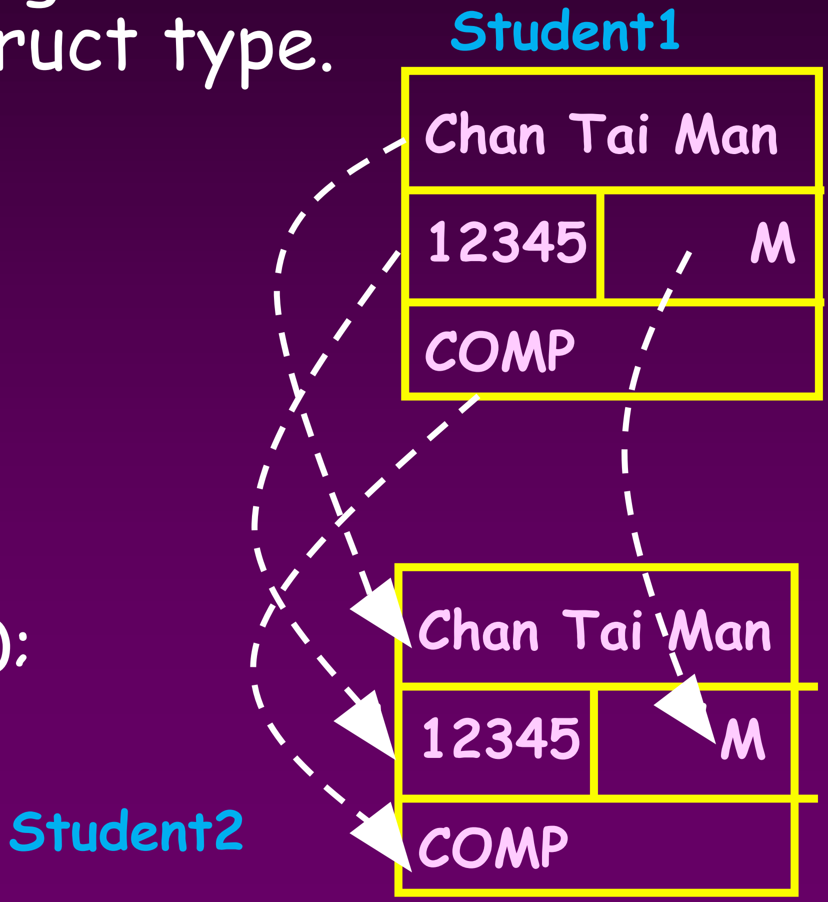
```
The student is Mr. Chan Tai Man
Press any key to continue_
```

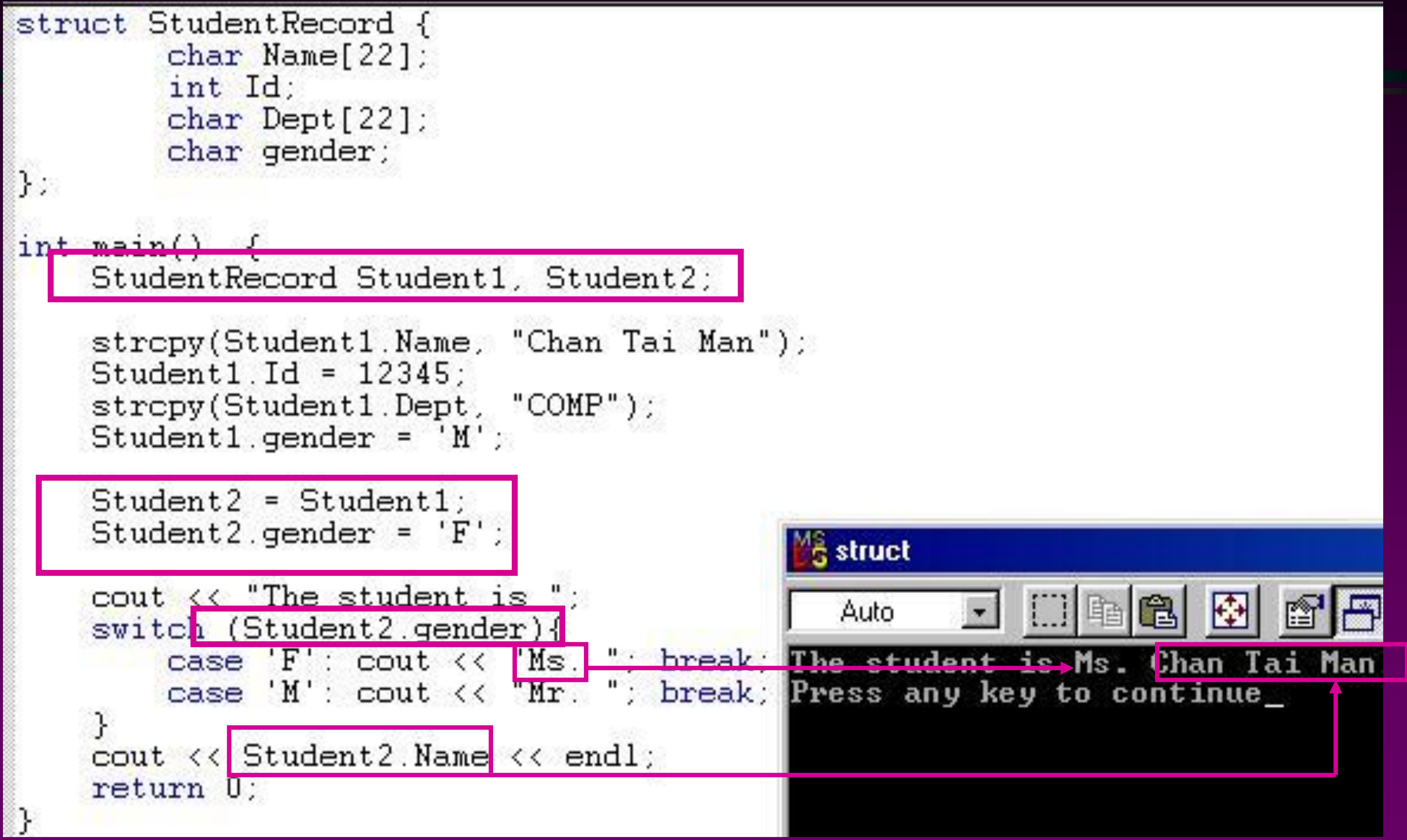
Ex. struct-to-struct assignment

- The values contained in one struct type variable can be assigned to another variable of the same struct type.

- Example:

```
strcpy(Student1.Name,
        "Chan Tai Man");
Student1.Id = 12345;
strcpy(Student1.Dept, "COMP");
Student1.gender = 'M';
Student2 = Student1;
```





```
struct StudentRecord {
        char Name[22];
        int Id;
        char Dept[22];
        char gender;
};

int main()  {
    StudentRecord Student1, Student2;

    strcpy(Student1.Name, "Chan Tai Man");
    Student1.Id = 12345;
    strcpy(Student1.Dept, "COMP");
    Student1.gender = 'M';

    Student2 = Student1;
    Student2.gender = 'F';

    cout << "The student is ";
    switch (Student2.gender){
        case 'F': cout << "Ms. "; break;
        case 'M': cout << "Mr. "; break;
    }
    cout << Student2.Name << endl;
    return 0;
}
```

```
The student is Ms. Chan Tai Man
Press any key to continue_
```

Initializing Structure Data

- You can initialize members when you declare a structure, or you can initialize a structure in the body of the program. Here is a complete program.

```
struct cd_collection
{
        char title[25];
        char artist[20];
        int num_songs;
        float price;
        char date_purchased[9];
} cd1 = {"Red Moon Men","Sams and the Sneeds", 12, 11.95,"08/13/93"};
cout<<"\nhere is the info about cd1"<<endl;
cout<<cd1.title<<endl;
cout<<cd1.artist<<endl;
cout<<cd1.num_songs<<endl;
cout<<cd1.price<<endl;
cout<<cd1.date_purchased<<endl
```

Cont'd

- A better approach to initialize structures is to use the dot operator(.). the dot operator is one way to initialize individual members of a structure variable in the body of your program. The syntax of the dot operator is :

  structureVariableName.memberName

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

  struct student
  {
      char name[30];
      int id;
      string department;
      string semester;
      float marks;
  };
  ```

Cont'd

```
int main()
{
    cout << "*-----Please Enter  Valid Student Information-----*" << endl;
    student stu;
    cout << "\nEnter Full Name: ";
    cin.get(stu.name, 30);
    cout << "Enter Registration: ";
    cin >> stu.id;
    cout << "Enter Department: ";
    cin >> stu.department;
    cout << "Enter Semester: ";
    cin >> stu.semester;
    cout << "Enter Total Semester Marks: ";
    cin >> stu.marks;
    cout << "\n\n*-----Academic Information of " << stu.name << "-----*"<< endl;
    cout << "\nName: " << stu.name << endl;
    cout <<"Reg No: " << stu.id << endl;
    cout <<"Department: " << stu.department << endl;
    cout << "Semester: " << stu.semester << endl;
    cout << "Total Marks: " << stu.marks;
    return 0;
}
```

Array of structure

- It is common to use arrays of structures. However, the structure has to be defined first, before any array declarations that refer to this particular structure.
- Example:

```
struct employee {
char name[80];
float hours;
float wage;
};
  employee staff[100];
```

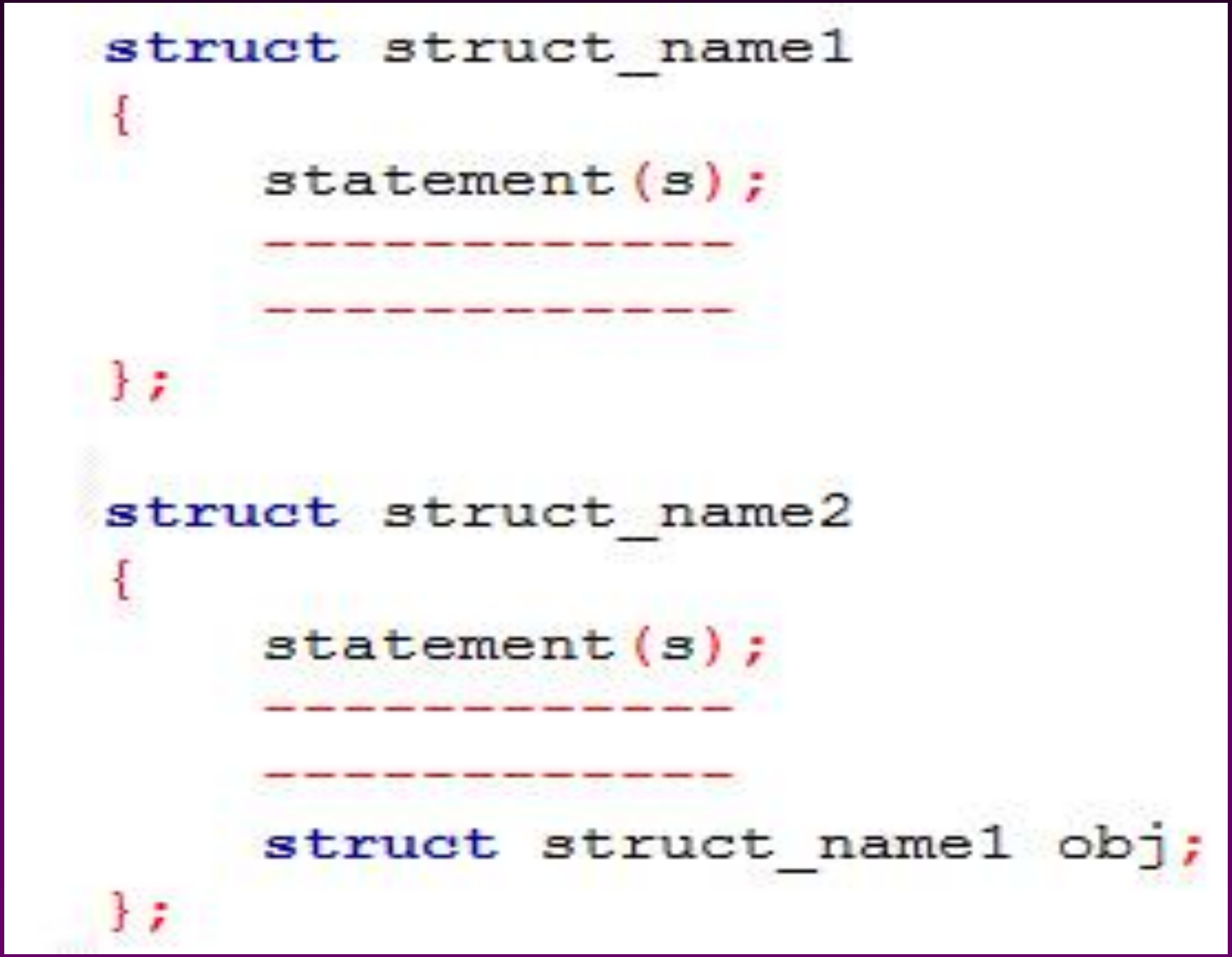
- Any entry in the database can be referred to by using the dot operator:

```
cout << staff[81].name;
staff[3].hours = 38.5;
```

Nesting Structures

- When a structure contains another structure, it is called **nested structure.**
- For example, we have two structures named Address and Employee. To make Address nested to Employee, we have to define Address structure before and outside Employee structure and create an object of Address structure inside Employee structure.

Syntax for structure within structure or nested structure



```
struct struct_name1
{
        statement(s);
        ---------------
        ---------------
};

struct struct_name2
{
        statement(s);
        ---------------
        ---------------
        struct struct_name1 obj;
};
```

Example

```
struct addr // structure tag
{
          int houseno ;
          char area[26] ;
          char city[26] ;
          char state[26] ;
};
struct emp // structure tag
{
           addr ad;
          int empno ;
          char name[26] ;
          char desig[16] ;
          float basic ;
};  emp worker; // create structure variable
```

C++ Accessing Nested Structure Member

- The members of structures are accessed using dot operator. To access the city member of address structure which is an element of another structure worker, we shall write :
-       worker.ad.city
- To initialize houseno member of address structure, element of worker structure, we can write as follows:
-       worker.ad.houseno = 1693
- As you can see, the elements of each structure are referenced from outermost to innermost.

Example

```
#include <iostream>

using namespace std;

struct date_of_birth
{
        int dd, mm, yy;
};

struct student

{

        char name[30];
        int rollNumber;
        date_of_birth dob;

} student st;
```

Cont'd

```
int main()
{
        student s;
        cout<<"Enter name : ";
        cin.getline(s.name,25);
        cout<<"Enter roll number : ";
        cin>>s.rollNumber;
        cout<<"Enter date of birth (dd mm yy) : " ;
        cin>>s.dob.dd>>s.dob.mm>>s.dob.yy;
        cout<<"Name:"<<s.name<<",Roll
        Number:"<<s.rollNumber<<endl; cout<<"Date of
        birth:"<<s.dob.dd<<"/"<<s.dob.mm<<"/"<<s.dob.yy<<e
ndl;    return 0;
}
```

Union

- Union is a user-defined datatype. All the members of union share same memory location.
- Size of union is decided by the size of largest member of union. If you want to use same memory location for two or more members, union is the best for that.
- Unions are similar to structures. Union variables are created in same manner as structure variables. The keyword “union” is used to define unions in C++ language.

Union

Defining a union

- Syntax

```
union unionName
{
        //member definitions
}; union_variables;
```

Example

```
union Courses
{
        char WebSite[50];
        char Subject[50];
        int Price;
};
```

Enumerations

- **Enumeration** is a user defined datatype in C/C++ language. It is used to assign names to the integral constants which makes a program easy to read and maintain. The keyword "enum" is used to declare an enumeration.
- The following is the syntax of enums.

```
    enum enum_name{const1, const2, ....... };
```

- Here,
  - enum_name − Any name given by user.
  - const1, const2 − These are values of type flag.

Enumerations

- Example

```
 #include <iostream>
 using namespace std;
enum colors{red=5, black};
enum suit{heart, diamond=8, spade=3, club};
int main()
{
    cout <<"The value of enum color : "<<red<<","<<black;
    cout <<"\nThe default value of enum suit :
      "<<heart<<","<<diamond<<","<<spade<<","<<club;
    return 0;
}
```

Enumerations

- Output
  - The value of enum color : 5,6
  - The default value of enum suit : 0,8,3,4

End of Topic 2