Fundamentals of Computer Programming

Chapter 7 User Defined Data types

(union, enum, typedef and class & object)

Chere L. (M.Tech)
Lecturer, SWEG, AASTU

g

Outline

- Introduction to User defined Data type
- Defining structure (with tag and without tag)
- Declaring structure variable
- Initializing structure elements
- Accessing structure elements
- Nested structure
  - ✓ Defining structure within structure
  - ✓ Structure variable within other structure definition
- Array of structure
- Structure pointer
- Structure with function
  - ✓ Structure variable as parameter
  - ✓ Structure as a return type

Outline

- Other User defined Data types
  - ✓ Anonymous unions (union)
  - ✓ Enumerated types (enum)
  - ✓ Type definition (typedef)
- Class and Object as user defined data type
  - ✓ Class and Object Basics
  - ✓ Class and Object Creation

7.1) Union

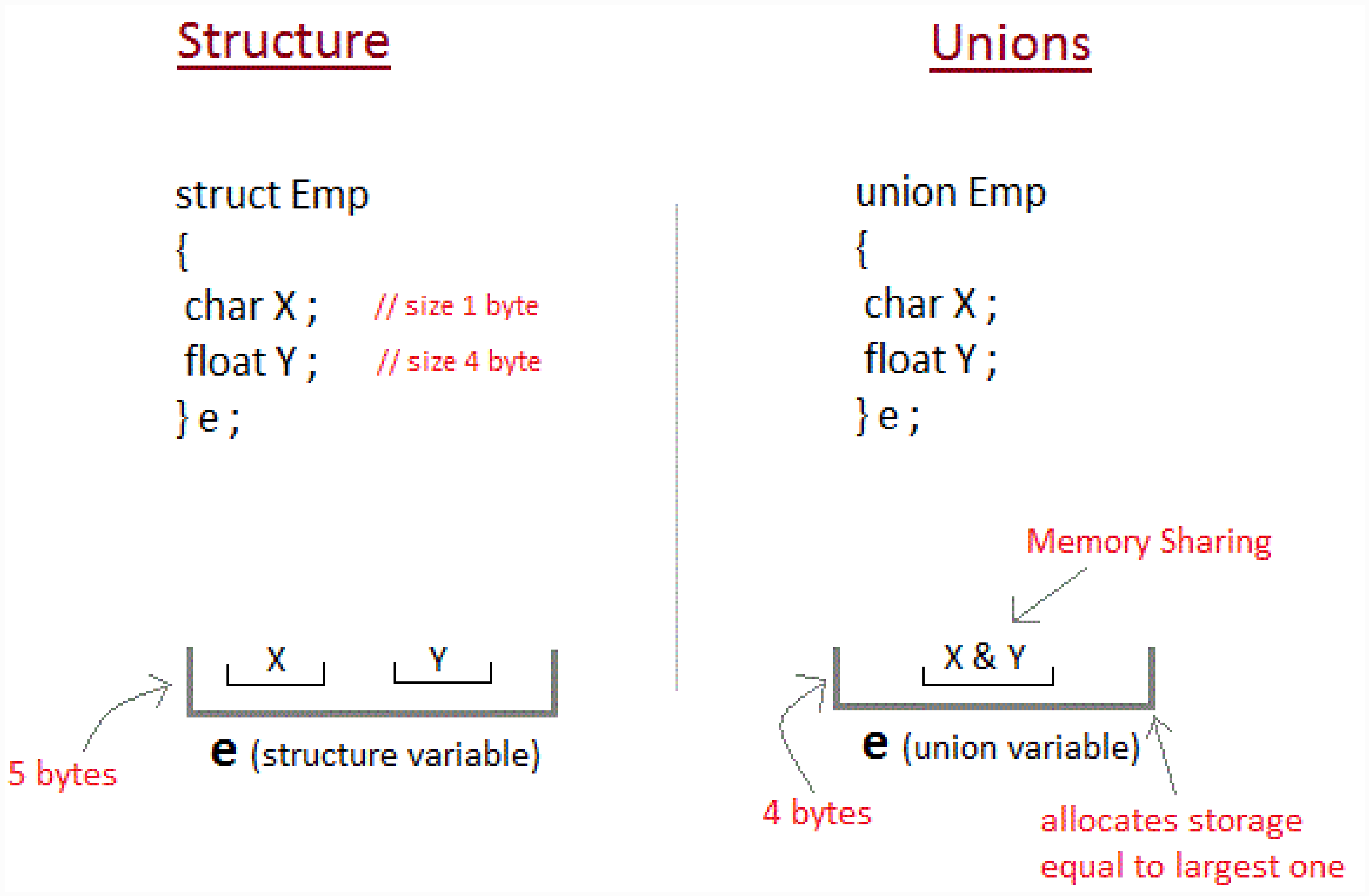
- A union is also a user defined data type that declares a set of multiple variable sharing the same memory area
- Unions are similar to structures in certain aspects.
  - ✓ *It used to group together variables of different data types.*
  - ✓ *It declared and used in the same way as structures.*
  - ✓ *The individual members can be accessed using dot operator.*
- **Syntax:**

```
    union union_tag{
            type1 member_namel;
            type2 member_name2;
                  ...        ...
            typeN member_nameN;
        };
```

7.1) Union (cont'd)

CONTENTS

Difference between structure and Union in terms of storage.



Structure

```
struct Emp
{
 char X ;     // size 1 byte
 float Y ;    // size 4 byte
} e ;
```

Unions

```
union Emp
{
 char X ;
 float Y ;
} e ;
```

7.1) Union (cont'd)

Example of comparing size of union and structure

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

// defining a struct
struct student1 {
            int roll_no;
            char name[40];
 };

// defining a union
union student2 {
            int roll_no;
            char name[40];
 };
```

```
int main()
{
    struct student1 s1;
    union student2 u1;
     cout << "size of structure : ";
     cout << sizeof(s1) << endl;
     cout << "size of union : ";
     cout<< sizeof(u1) << endl;
return 0;
}
```

7.1) Union (cont'd)

Summary of structure & union Distinction

No	Structure	Union
1	Every member has its own memory	All members use the same memory
2	Keyword *strcut* is used	Keyword *union* is used
3	All members can be initialized	Only its first members can be initialized
4	Consume more space	Memory conversion

When/Where it can be used?

- Allow data members which are mutually exclusive to share the same memory. important when memory is more scarce.
- Useful in ***Embedded programming*** or in situations where direct access to the hardware/memory is needed.

7.1) Union (cont’d)

Example: creating object & accessing union members

```
#include <iostream>
using namespace std;
union Employee{
        int Id, Age;
        char Name[25];
       long Salary;
    };

int main(){
     Employee E;
     cout << "\nEnter Employee Id : ";               cin >> E.Id;
     cout << "Employee Id : " << E.Id;
     cout << "\n\nEnter Employee Name : ";       cin >> E.Name;
     cout << "Employee Name : " << E.Name;
     cout << "\n\nEnter Employee Age : ";         cin >> E.Age;
     cout << "Employee Age : " << E.Age;
     cout << "\n\nEnter Employee Salary : ";     cin >> E.Salary;
     cout << "Employee Salary : " << E.Salary;
     }
```

7.2) Enumeration

- Enumerations are used to create new data types of multiple integer constants.
- An enumeration consists of a list of values.
- *enum* types can be used to set up collections of **named integer constants.**
  - ✓ Each value has a unique number i.e. Integer CONSTANT starting from 0.
  - ✓ Cannot take any other data type than integer
- **enum** declaration is only a blueprint for the variable is created and does not contain any fundamental data type
- **Syntax**

```
enum  identifier { variable1, variable2, . . .};
```

7.2) Enumeration (cont'd)

- **Example 1:**

  enum days_of_week {Monday, Tuesday,
  Wednesday, Thursday, Friday,
  Saturday, Sunday};

- **Enumerations**
  - ✓ *Each value in an enumeration is always assigned an integer value.*
  - ✓ *By default, the value starts from 0 and so on.* E.g. On the above **enum declaration**

    VALUES: Monday = 0, Tuesday = 1 ,… , Sunday = 6

  - ✓ *However, the values can be assigned in different way also*
  - ✓ **Example:**

    enum colors{ white, Green, Blue, Red=7, Black, Pink**};**

7.2) Enumeration (cont'd)

- **Example 1: Enumeration Type**

```
#include <iostream>
using namespace std;
enum week {
                Monday, Tuesday, Wednesday,
                Thursday, Friday, Saturday, Sunday
      };
int main() {
     week today;
     today = Wednesday;
     cout << "Day " << today+1;
     cout<<"\nSize: "<<sizeof(week);
return 0;
 }
```

7.2) Enumeration (cont'd)

- **Example 2: enum declaration and changing default value**

```
#include <iostream>
using namespace std;
enum colors {  green=1, red, blue, black=0, white,
                    brown=11, pink, yellow, aqua, gray
       }flag_color;   //definition at the time declaration
int main() {
      colors painting;
      painting = brown;
      flag_color = green;

      cout<<"Painting Colors: "<<endl;
      for(int i=painting; i<= 15; ++i ) {
                 cout<<painting+i<<"  ";  }
  return 0;
 }
```

7.2) Enumeration (cont'd)

Application

Enables programmers to define variables and restrict its value to a set of possible values which are integer (*4 bytes) and* also makes to take only one value out of many possible values

- When it expected that the variable to have one of the possible set of values.
  - ✓ E.g., assume a variable that holds the direction. Since we have four directions, this variable can take any one of the four values
- switch case statements, where all the values that case blocks expect can be defined in an enum.
- A good choice to work with flags

```
enum styleFlags {  ITALICS = 1, BOLD,
                          REGULAR, UNDERLINE} button;
```

7.3) typedef

- The keyword **typedef** provides a mechanism for creating *synonyms (or aliases)* for previously defined data types.
- In simple words, a **typedef** is a way of *renaming* a type
- **Syntax:**

  typesdef existing_type_name new_typeName;

- Once the **typedef synonym (new_typeName)** is defined, you can declare your variables using the **synonym** as a new type.

7.3) typedef (cont'd)

- **Example:**

  typedef *unsigned short int USHI;* *USHI age;*

  typedef struct Student Tstud; *Tstud class[50[];*

- In this example, both ***USHI*** and ***Tstud*** are a ***synonym***

Note:

- **Typedef** does not really create a *new* type
- It is good practice to
  - ✓ Capitalize the first letter of typedef synonym.
  - ✓ Preface it with a capital "T" to indicate it's one of your new types e.g. TRobot, TStudent, etc.

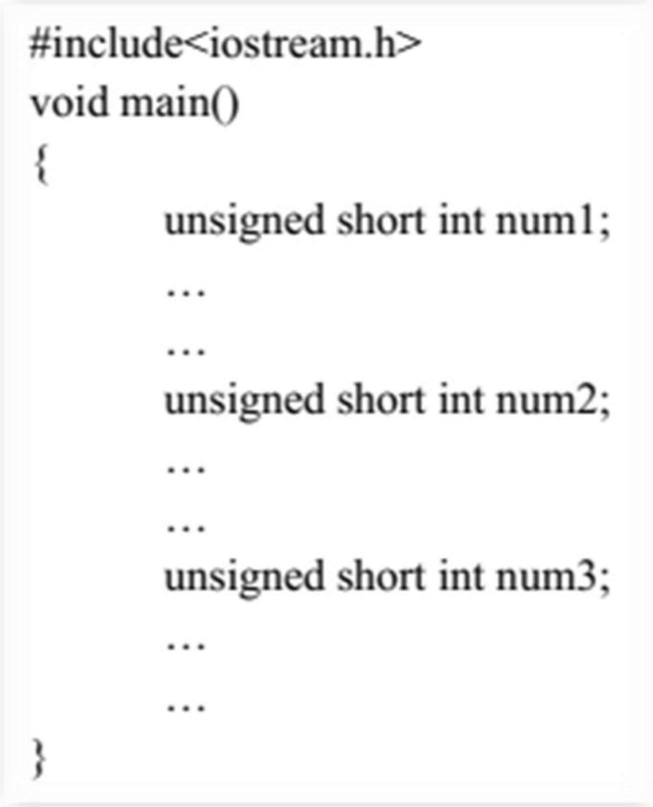
7.3) typedef (cont'd)

Example: complete program

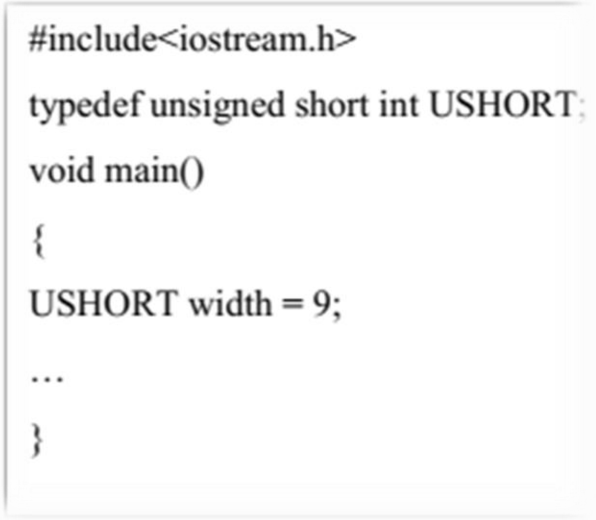
```
#include<iostream.h>
usning namespace std;

struct student_records{
      char sId[10];
      char sName[20];
      float mark;
};

typedef unsingned short int ME;
typedef student_records studRec;

int main(){
    ME a=1, b;
    cout<<"Enter a number: ";
    cin>>b;
    cout<<"the Numbers are \n"
    cout<<a<<"\n"<<b;
    cout<<"\n"<<sizeof(ME);

  studRec S1 = {"ETS215/12",
                 "John", 80};
   cout<<"Student Info:\n";
    cout<<"ID: "<<S1.sId;
    cout<<"Name: "<<S1.sName;
    cout<<"Mark: "<<S1.mark;

    return 0;
}
```

7.3) typedef (cont'd)

Application



```
#include<iostream.h>
void main()
{
          unsigned short int num1;
          …
          …
          unsigned short int num2;
          …
          …
          unsigned short int num3;
          …
          …
}
```



```
#include<iostream.h>
typedef unsigned short int USHORT;
void main()
{
USHORT width = 9;
…
}
```

(8) Class and Object

- Classes and Objects are basic concepts of OOP which revolve around the real life entities

Class

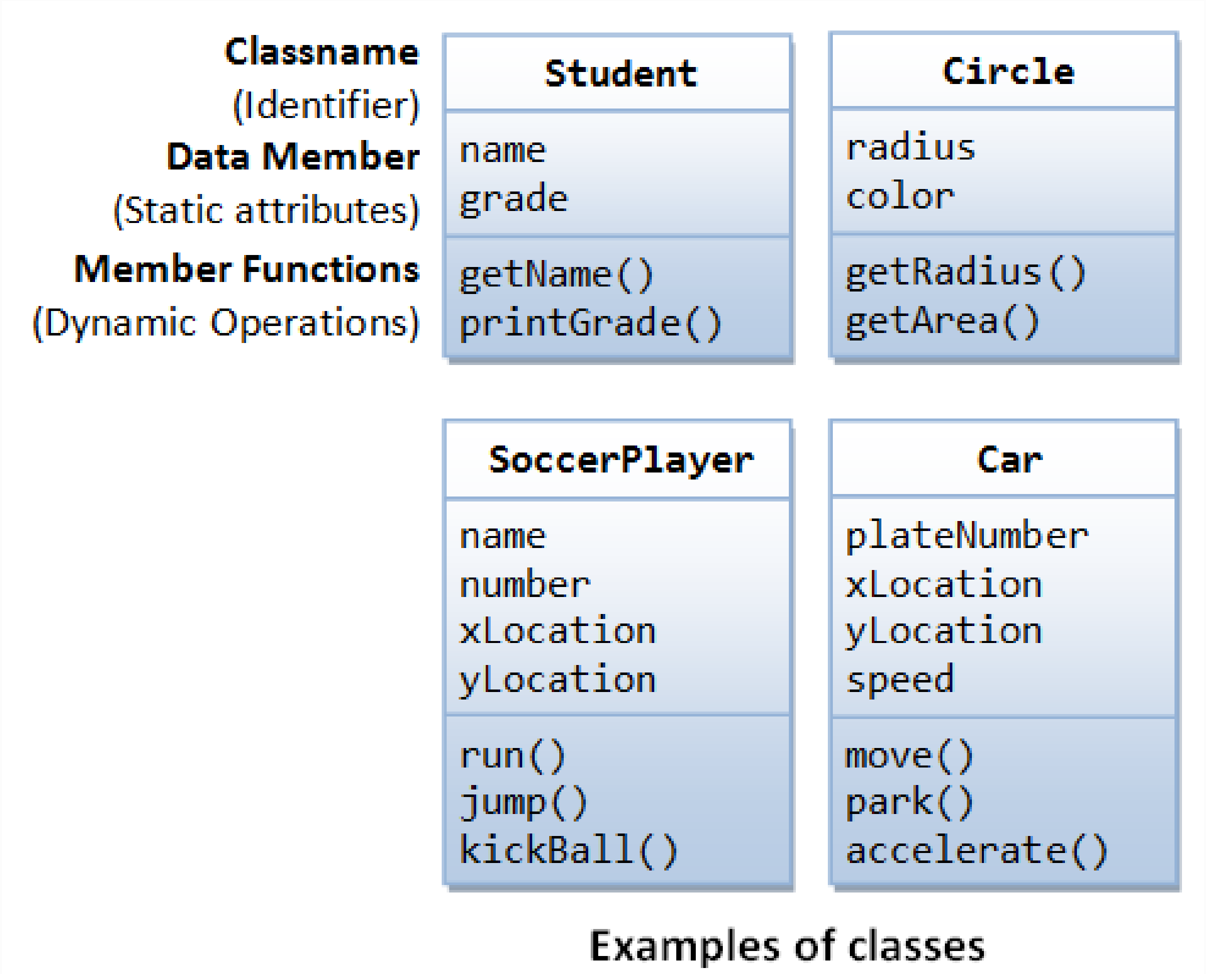
- A **user defined blueprint (prototype)** from which objects are created.
- It represents the *set of properties* or *methods* that are common to all objects of one type.

Object

- Represents the real life entities.
- An object consists of:
  - ✓ **state –** represented by **attributes (data member)** of an object
  - ✓ **behavior -** represented by **methods (functions)** of an object

(8) Class and Object (cont'd)

- Class model objects that have attributes (data members) and behaviors (member functions)



Examples of classes

(8) Class and Object (cont'd)

Class and Object creation

- Class is defined using keyword **class**
- Have a body delineated with braces (**{** and **};**)
- Member visibility is determined by access modifier:
  - ✓ private
  - ✓ protected
  - ✓ public

Syntax

```
class className {
        // some data
        // some functions
};
```

(8) Class and Object (cont'd)

Example

```
class Room {
    public:
        double length;
        double breadth;
        double height;

        double calculateArea(){
            return length * breadth;
        }

        double calculateVolume(){
            return length * breadth * height;
        }

};
```

(8) Class and Object (cont'd)

Example (cont. . .)

```
int main() {

  Room room1;  // create object of Room class

    // assign values to data members
    room1.length = 42.5;
    room1.breadth = 30.8;
    room1.height = 19.2;

    // calculate and display the area and volume of the room
    cout << "Area of Room =  " << room1.calculateArea() <<
      endl;
    cout<< "Volume of Room =  "<<room1.calculateVolume();

    return 0;
}
```

(8) Class and Object (cont'd)

Some of the differences between class Vs. structure

Structure	Class
A grouping of variables of various data types referenced by the same name.	A collection of related variables and functions contained within a single structure.
By default, all the members of the structure are **public** unless access specifier is specified	By default, all the members of the structure are **private** unless access specifier is specified
It's instance is called the '***structure variable***'.	A class instance is called 'object'.
It does not support inheritance.	It supports inheritance.
Memory is allocated on the stack.	Memory is allocated on the heap.

(8) Class and Object (cont'd)

Some of the differences between class Vs. structure

Structure	Class
Value Type	Reference Type
Purpose - grouping of data	Purpose - data abstraction and further inheritance.
It is used for smaller amounts of data.	It is used for a huge amount of data.
NULL value is Not possible	It may have null values.
It may have only parameterized constructor.	It may have all the types of constructors and destructors.
automatically initialize its members.	constructors are used to initialize the class members.

(8) Class and Object (cont'd)

Example program - Class Vs. struct

```
// classes example
#include <iostream>
using namespace std;

class Rectangle1 {
   //private by default
     int width, height;
  public:
    ~Rectangle1 (int x, int y){
            this.width =x ;
            this.height = y;
     }
    int area() {
      return width*height;
     }
};
```

```
struct Rectangle2 {
  //public by default
     int width, height;
int area() {
       return width*height;
      }  };

int main () {
  Rectangle1 rect1 = Reactangle1
(3,4);   //constructor
  cout << "area 1: " << rect.area();
   Rectangle2 rect2 = {6, 5};
  cout << "area 2: " << rect2.area();
  return 0;
}
```

Practical Exercise

1. Create a class named 'Student' with a string variable 'name' and an integer variable 'roll_no'. Assign the value of roll_no as '2' and that of name as "John" by creating an object of the class Student.

2. Write a program to print the area and perimeter of a triangle having sides of 3, 4 and 5 units by creating a class named 'Triangle' with the constructor having the three sides as its parameters.

3. Write a program to print the area of a rectangle by creating a class named 'Area' having two functions. First function named as 'setDim' takes the length and breadth of the rectangle as parameters and the second function named as 'getArea' returns the area of the rectangle. Length and breadth of the rectangle are entered through keyboard.

4. Print the sum, difference and product of two complex numbers by creating a class named 'Complex' with separate functions for each operation whose real and imaginary parts are entered by the user.

Practical Exercise

5. Show how to declare **Real** as a synonym for float (float and Real can be used interchangeably).

6. Show how to declare the following constants using an **enum statement**:
   WINTER =1, SPRING = 2, SUMMER = 3 and FALL = 4.

7. Provide an example that demonstrate the use of **enum** in defining flags.

8. Which user defined data type is appropriate to design self driving car gear? Justify your answer by providing demonstration example.

9. Assume there are a medical data set stored on cloud (server computer). A client computer access and process each data groups independently. Which user defined data type is appropriate to write the client side program? Justify your answer with demonstration.

MCQ

1. What does your class can hold?

   A. data B. functions C. both a & b D. none

2. The union data type differ from structure both in storage allocation and member initialization.

   A. True B. False

3. Unlike that of union by default the members of class is public.

   A. True B. False

4. Members of one of the following user defined data type is not associated with fundamental data types.

   A. enumeration B. class

   C. structure D. union E) B&C

5. A user defined data types that is useful for embedded programming

   A. union B. typedef C. class D. enum.

MCQ

6. An appropriate user defined data types to work with FLAGS;
   A. strcut B. union C. class D. enum.
7. An appropriate user defined data types to work with FLAGS;
   A. strcut B. union C. class D. enum.
8. Which of the following is odd in their declaration style?
   A. strcut B. union
   C. typedef D. class E. enum
9. There is no programming philosophy between structure and class.
   A. True B. False
10. Which of the following is not used to create a new Data Type?
    A. strcut B. union C. typedef E. enum
11. Is it mandatory that the size of all elements in a union should be same?
    A. True B. False

Short Answer

1. Compare and contrast structure data type with the following:
   a) Class
   b) Union
   c) Array
2. Explain the importance of enumeration and union types.
3. What is the size of the following user defined data types
   a) Structure c) Union
   b) Class d) enum
4. Does structure replace class completely? Justify your answer with examples.

Reading Resources/Materials

Chapter 10 & 11:

- ✓ Herbert Schildt; C++ from the Ground Up (3rd Edition), 2003 McGraw-Hill, California 94710 U.S.A.

Chapter 8:

- ✓ Bjarne Stroustrup;The C++ Programming Language [4th Edition], Pearson Education, Inc , 2013

Chapter 15:

- ✓ Diane Zak; An Introduction to Programming with C++ (8th Edition), 2016 Cengage Learning

Chapter 10:

- ✓ Walter Savitch; Problem Solving With C++ [10th edition, University of California, San Diego, 2018

Thank You

For Your Attention!!