

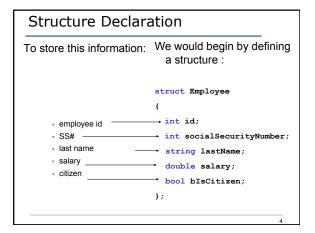
structs

- Arrays are useful for storing a collection of data elements of the **same** data type
- What about storing a collection of data elements of different data types?
- Related information can be placed in a *structure*, which has a general format as follows:
- struct StructName
 - // variable declarations
- };

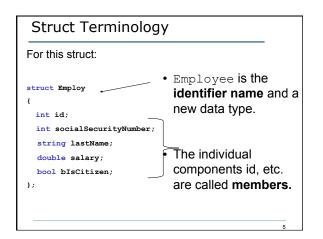
{

struct Definition

- structs store a collection of data elements of different data types
- For example, what if we wanted to keep the following information on a particular employee:
 - o employee id
 - o SS#
 - last name
 - o salary
 - o citizen
- The elements have different data types, so we can't conveniently use an array. Instead we will use a struct









Notes on structs

- A semicolon is required after the closing brace of the struct declaration
- The struct declaration does not create a variable
- It just tells the compiler what that struct is made of
- The **struct** declaration is usually placed above the main

Variable Declaration

 As with all data types, in order to use our new data type Employee we must allocate storage space by declaring variables of this data type:

Employee sEngineer, sTech;

 This will allocate space for two variables called sEngineer and sTech, each containing the previously described members id, socialSecurityNumber, etc. Each of these variables is a separate instance of Employee

Dot Operator

- To access a struct member, we use the *dot operator* (period between struct variable name and member name).
- In the variable sEngineer of data type Employee we can make the assignments:
 - sEngineer.id = 12345;

sEngineer.socialSecurityNumber = 534334343;

sEngineer.lastName = "Doe"; sEngineer.salary = 45443.34;

sEngineer.bIsCitizen = true;

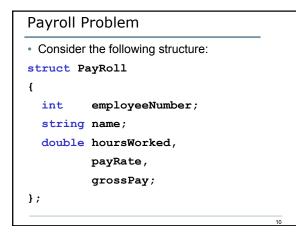
Notes on Structures

 You cannot output the entire contents of a struct variable by simply using its name

```
o cout << sEngineer; // ERROR!</pre>
```

• Similarly, you cannot compare two struct variables by using their name

o if(sEngineer == sTech) // ERROR!

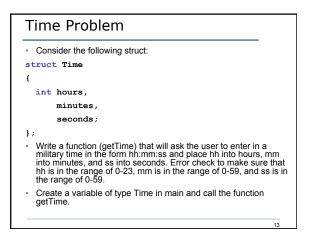


Payroll Problem

• Declare a PayRoll variable sDeptHead and assign the employeeNumber, name, and payRate with the values 123, Joe Smith, and 10.00.

Passing structs to Functions

- structs can be passed to functions by reference or value in the same manner that other data types have been passed
- Generally, passing structs by reference is preferred since passing by value requires a local copy of the struct to be created within the function's variables



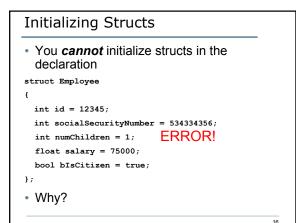


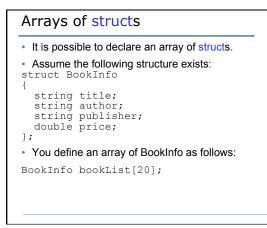
Displaying/Comparing structs

```
• Which of the following C++ statements are legal
given variables sTime1 and sTime2 of type Time
exist?
a) cout << sTime1 << sTime2;
b) if (sTime1 == sTime2)
    {
        cout << "times are equal";
    }
c) cout << sTime1.hours;
d) cin >> sTime1;
e) cin >> sTime1.hours;
```

Initializing structs

- Use an initializer list
 - o Employee sManager(12345, 534334356, 1, 76899, true);
- You can also initialize only some of the members in a struct:
 - o Employee sManager(12345, 534334356, 1);





Arrays of structs

- To access a specific member in the array, you would use the index of the array and the dot operator.
- For example:

bookList[0].title = "Jane Eyre";

bookList[2].author = "John Grisham";

CS250 Introduction to Computer Science II

17

Program

- A datafile called athletes.txt exists which contains an unknown amount of information where each line of the file contains an id, age, and weight of a specific athlete. The last line contains the sentinals 9999 0 0
- The program will contain two functions:
 - void readAthleteData This function reads in up to 100 lines of data into an array of structs and returns the number of athletes in the datafile.
 - **int whatAge** This function returns the age of the athlete with a given idNumber.
- Declare a struct for the athlete's data
- Create an array of structs to hold all athlete's data
- Write each function described above

CS250 Introduction to Computer Science II