Lesson 5
Records (`structs`)

Records (`structs`)

- `struct`: collection of a fixed number of components (members), accessed by name
  - Members may be of different types
- Syntax:

```
struct structName
{
    dataType1 identifier1;
    dataType2 identifier2;
      .
      .
      .
    dataTypen identifiern;
};
```

Records (`structs`) (cont'd.)

- A `struct` is a definition, not a declaration
  - Must declare a variable of that type to use it

```
struct houseType
{
    string style;
    int numOfBedrooms;
    int numOfBathrooms;
    int numOfCarsGarage;
    int yearBuilt;
    int finishedSquareFootage;
    double price;
    double tax;
};

  //variable declaration
houseType newHouse;
```

Records (`structs`) (cont'd.)

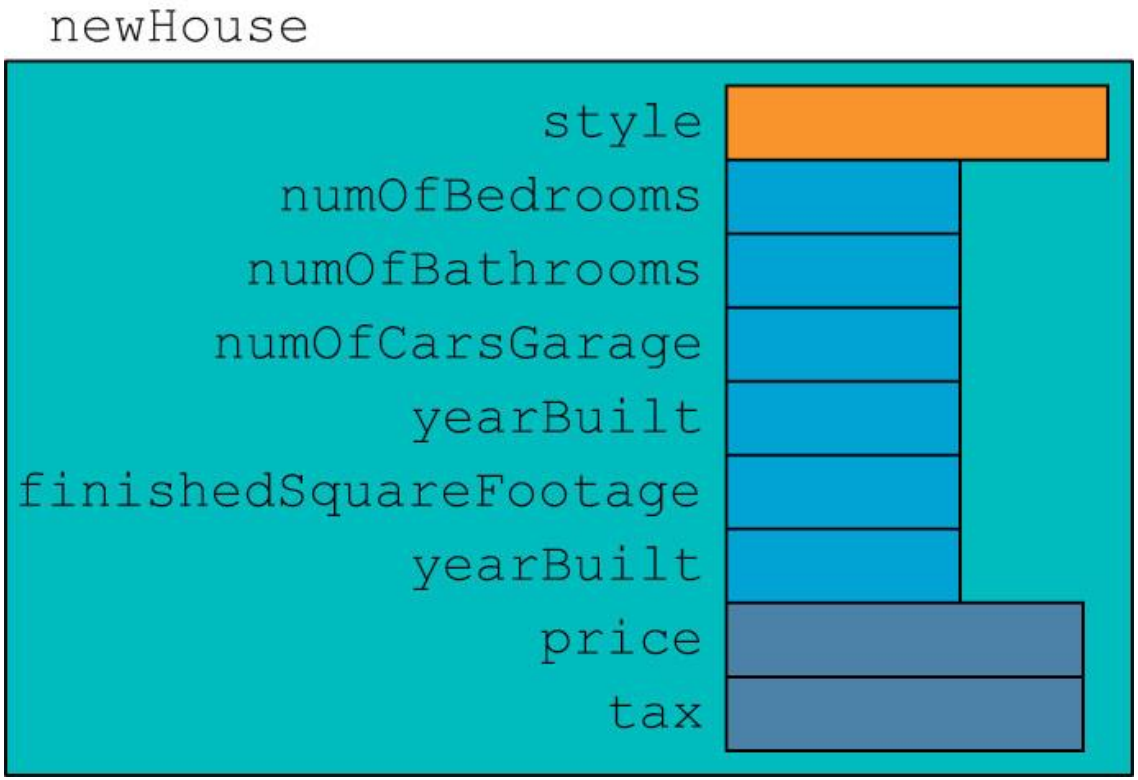

FIGURE 9-1 `struct` newHouse

Accessing `struct` Members

- Syntax to access a `struct` member:

```
structVariableName.memberName
```

- The dot (.) is called the <u>member access operator</u>

Accessing `struct` Members (cont'd.)

- To initialize the members of `newStudent`:

```
newStudent.GPA = 0.0;
newStudent.firstName = "John";
newStudent.lastName = "Brown";
```

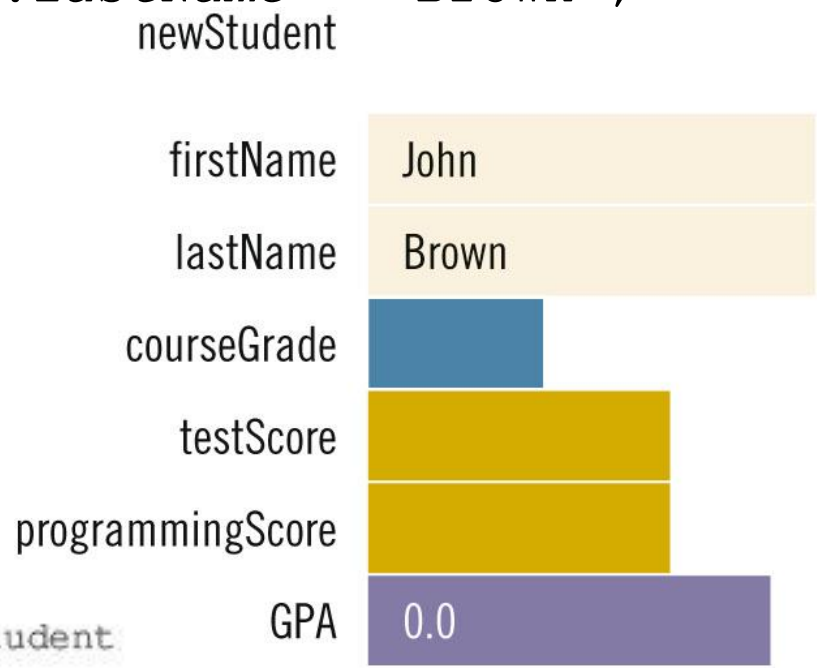

FIGURE 9-2 `struct newStudent`

Assignment

- Value of one `struct` variable can be assigned to another `struct` variable of the same type using an assignment statement
- The statement:

```
student = newStudent;
```

copies the contents of `newStudent` into `student`

Assignment (cont'd.)

- The assignment statement:

```
student = newStudent;
```

is equivalent to the following statements:

```
student.firstName = newStudent.firstName;
student.lastName = newStudent.lastName;
student.courseGrade = newStudent.courseGrade;
student.testScore = newStudent.testScore;
student.programmingScore = newStudent.programmingScore;
student.GPA = newStudent.GPA;
```

Comparison (Relational Operators)

- Compare `struct` variables member-wise
  - No aggregate relational operations allowed
- To compare the values of `student` and `newStudent`:

```
if (student.firstName == newStudent.firstName &&
    student.lastName == newStudent.lastName)
.
.
.
```

Input/Output

- No aggregate input/output operations on a `struct` variable
- Data in a `struct` variable must be read or written one member at a time
- Example: output `newStudent` contents

```
cout << newStudent.firstName << " " << newStudent.lastName
     << " " << newStudent.courseGrade
     << " " << newStudent.testScore
     << " " << newStudent.programmingScore
     << " " << newStudent.GPA << endl;
```

`struct` Variables and Functions

- A `struct` variable can be passed as a parameter by value or by reference
- A function can return a value of type `struct`

```
void printStudent(studentType student)
{
    cout << student.firstName << " " << student.lastName
         << " " << student.courseGrade
         << " " << student.testScore
         << " " << student.programmingScore
         << " " << student.GPA << endl;
}
```

Arrays versus `structs`

TABLE 9-1 Arrays vs. `structs`

Aggregate Operation	Array	`struct`
Arithmetic	No	No
Assignment	No	Yes
Input/output	No (except strings)	No
Comparison	No	No
Parameter passing	By reference only	By value or by reference
Function returning a value	No	Yes

`structs` in Arrays

- Example:

```
struct employeeType
{
    string firstName;
    string lastName;
    int    personID;
    string deptID;
    double yearlySalary;
    double monthlySalary;
    double yearToDatePaid;
    double monthlyBonus;
};
```

`structs` in Arrays (cont'd.)

```
employeeType employees[50];
```

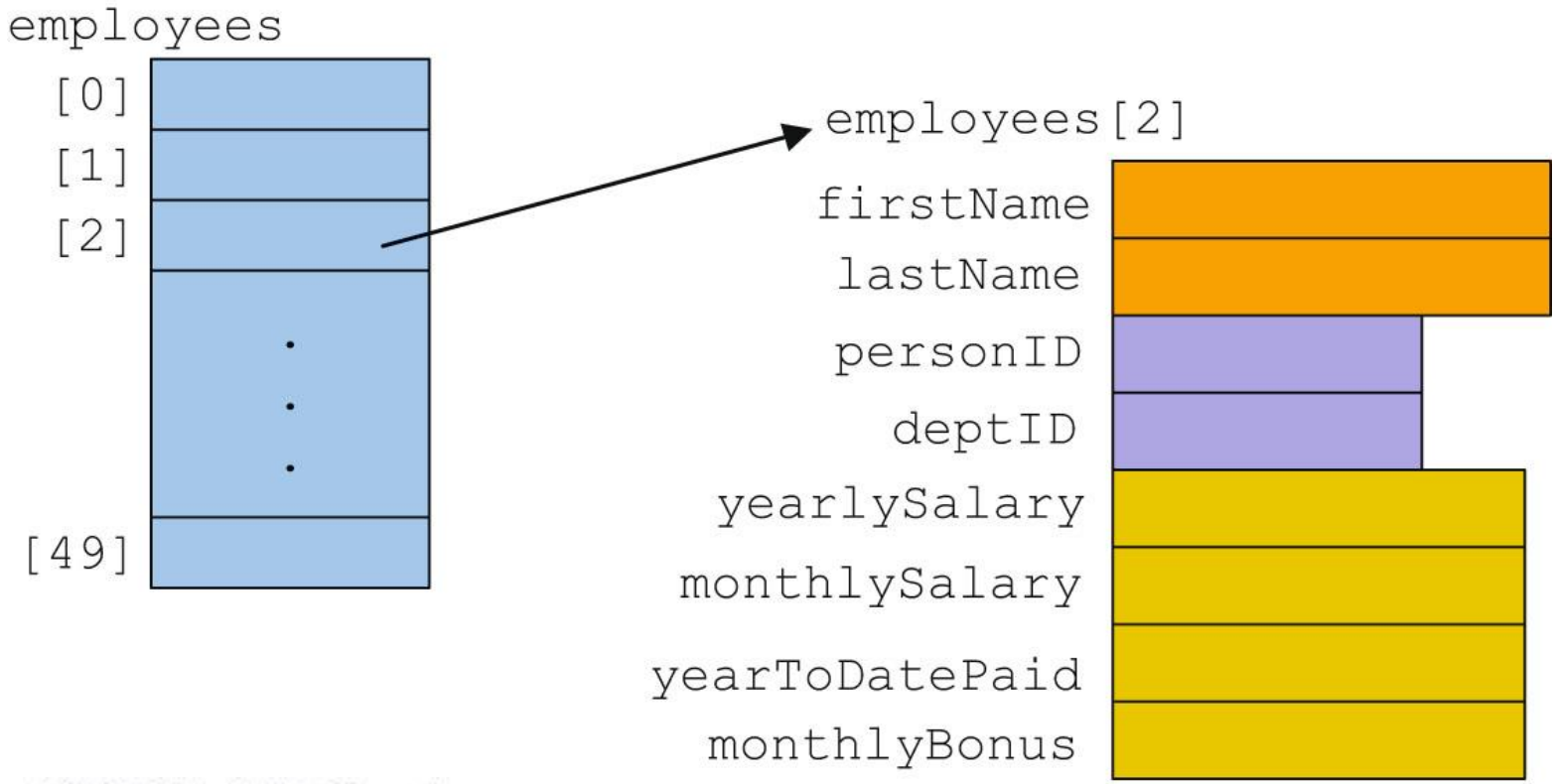

FIGURE 9-7 Array of employees

`structs` within a `struct`

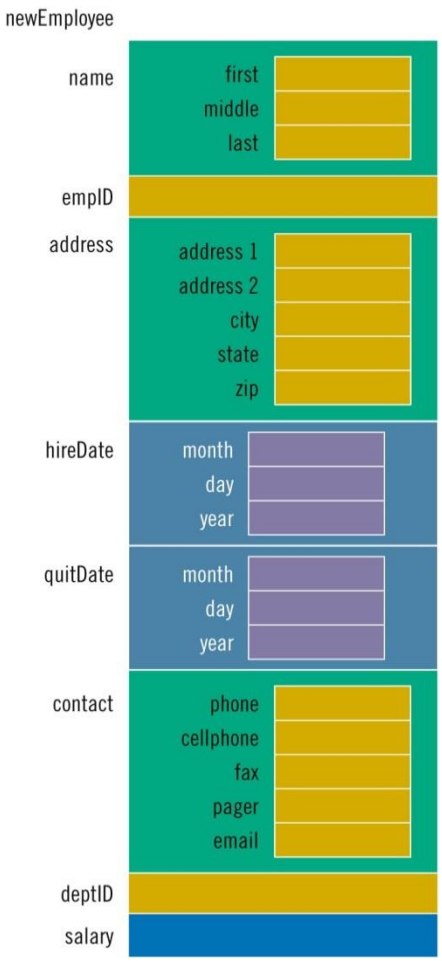

FIGURE 9-8 `struct` variable `newEmployee`