Lecture Notes

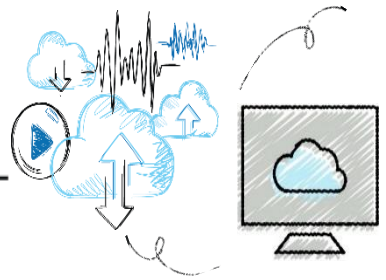
Chapter 9

Records (`structs`)

ECE 111: Introduction to C and C++ Programming

Instructor: Dr. Shayan (Sean) Taheri

Gannon University (GU)



Personal Information

- Name: Shayan (Sean) Taheri.
- Date of Birth: July/28/1991.
- Current Position: Assistant Professor at Gannon University
- Previous Position: Postdoctoral Fellow at University of Florida.
- Ph.D. Degree: Electrical Engineering from the University of Central Florida.
- M.S. Degree: Computer Engineering from the Utah State University.
- University Profile: https://www.gannon.edu/FacultyProfiles.aspx?profile=taheri001

Objectives (1 of 2)

- In this chapter, you will:
  - Learn about records (**`structs`**)
  - Examine various operations on a **`struct`**
  - Explore ways to manipulate data using a **`struct`**
  - Learn about the relationship between a **`struct`** and functions
  - Examine the difference between arrays and **`structs`**

CENGAGE Learning

Objectives (2 of 2)

- Discover how arrays are used in a **`struct`**
- Learn how to create an array of **`struct`** items
- Learn how to create **`struct`**s within a **`struct`**s

Records (`structs`) (1 of 3)

- **`struct`**: a collection of a fixed number of components in which the components are accessed by name
  - The components may be of different types and are called the members of the **`struct`**
- Syntax

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

Records (structs) (2 of 3)

- 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) (3 of 3)

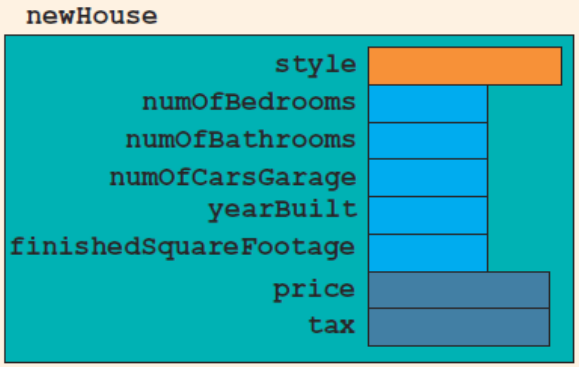


FIGURE 9-1 struct newHouse

Accessing `struct` Members (1 of 2)

- Syntax to access a **`struct`** member:

```
structVariableName.memberName
```

- The dot (**`.`**) is called the member access operator

Accessing `struct` Members (2 of 2)

- To initialize the members of **`newStudent`**:

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

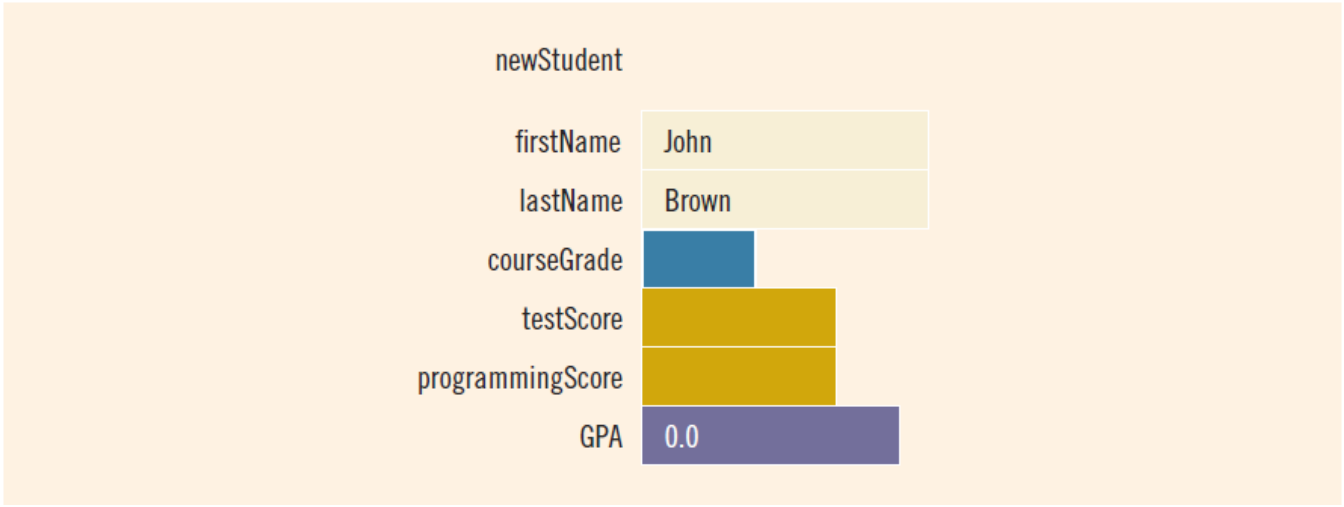


FIGURE 9-2 `struct` newStudent

Assignment (1 of 2)

- 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 (2 of 2)

- 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;
```

CENGAGE Learning

Comparison (Relational Operators)

- Compare **`struct`** variables member-wise
  - No aggregate relational operations are 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 are allowed on a **`struct`** variable
- Data in a **`struct`** variable must be read or written one member at a time
- The following code would output **`newStudent`** contents:

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

CENGAGE Learning

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**
- The following function displays the contents a **struct** variable of type **studentType**:

```
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`

Data Type	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

Arrays in `structs` (1 of 3)

- Two items are associated with a list:
  - Values (elements)
  - Length of the list
- Define a **`struct`** containing both items:

```
const int ARRAY_SIZE = 1000;
struct listType
{
    int listElem[ARRAY_SIZE];   //array containing the list
    int listLength;             //length of the list
};
```

Arrays in `structs` (2 of 3)

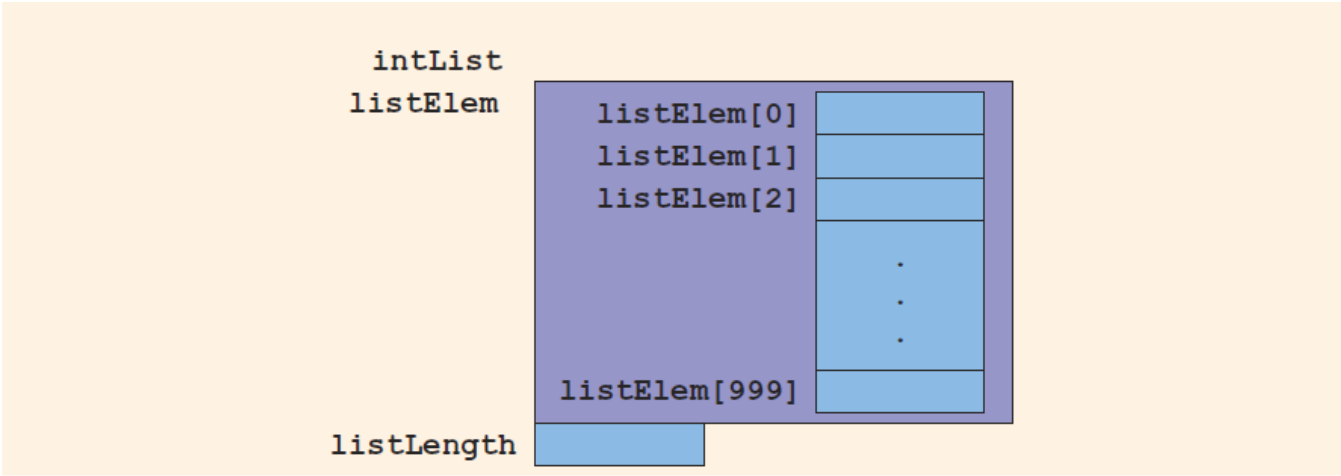


FIGURE 9-5 `struct` variable `intList`

Arrays in `structs` (3 of 3)

- Consider these statements and refer to the figure below showing the results following execution of the statements:

```
intList.listLength = 0;     //Line 1
intList.listElem[0] = 12;   //Line 2
intList.listLength++;       //Line 3
intList.listElem[1] = 37;   //Line 4
intList.listLength++;       //Line 5
```

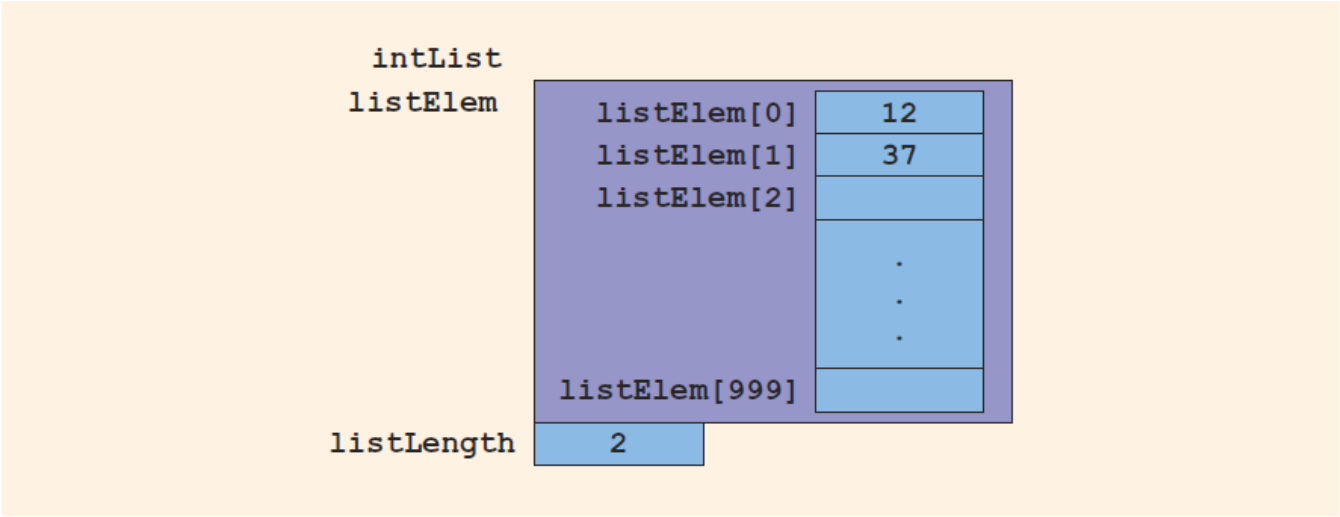


FIGURE 9-6 `intList` after the statements in Lines 1 through 5 execute

structs in Arrays (1 of 2)

- Example

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

structs in Arrays (2 of 2)

employeeType employees[50]

- Declares the array **employees** of **50** components of type **employeeType**

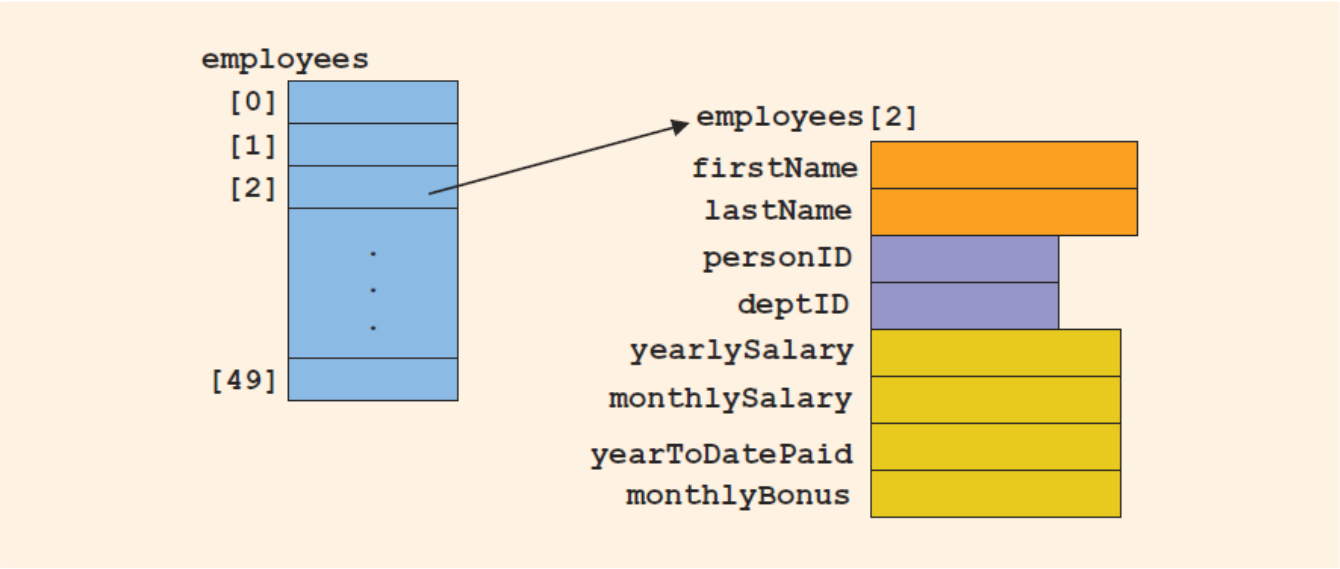


FIGURE 9-7 Array of **employees**

structs within a struct

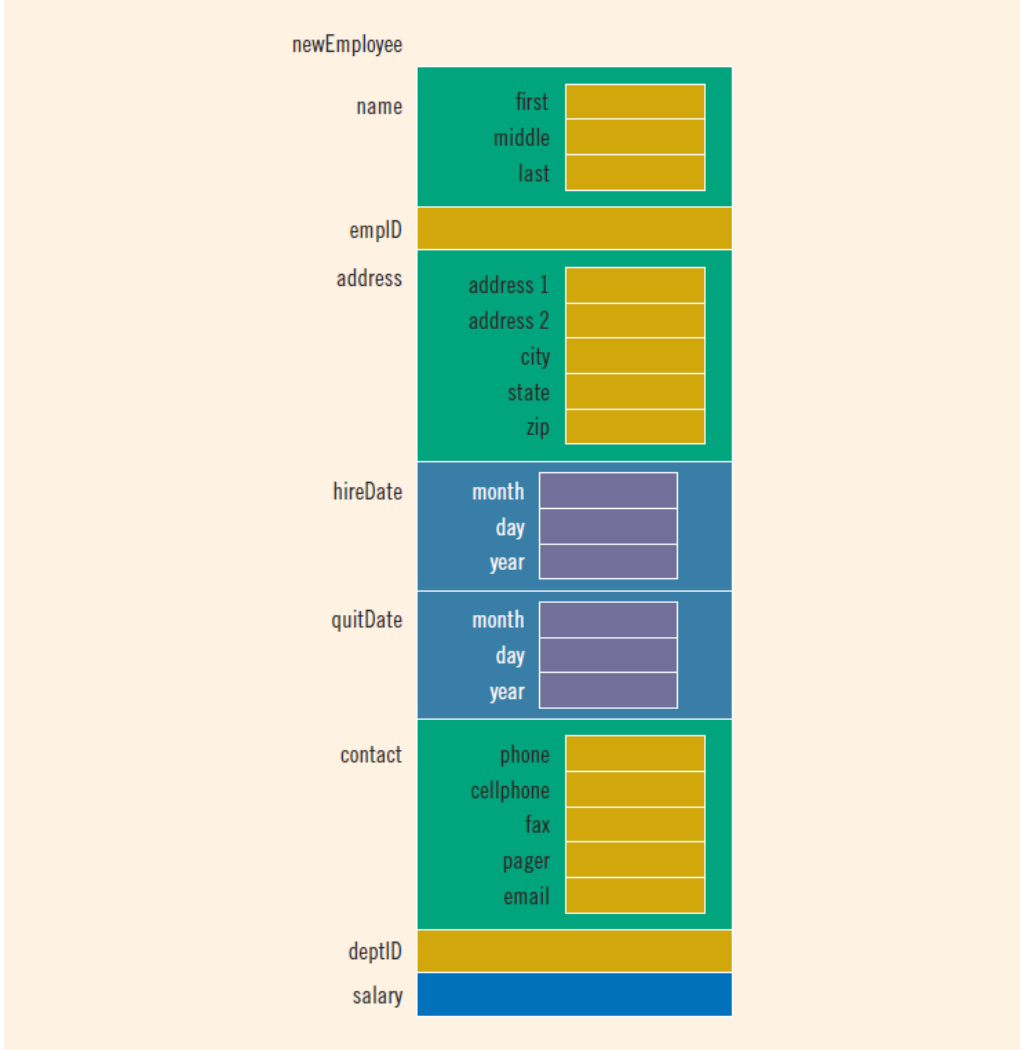


FIGURE 9-8 `struct` variable `newEmployee`

Quick Review (1 of 2)

- A `struct` is a collection of a fixed number of components
- Components of a `struct` can be of different types
  - Called members
  - Accessed by name
- `struct` is a reserved word
- No memory is allocated for a `struct`
  - Memory is allocated only when variables are declared

Quick Review (2 of 2)

- In C++, the dot (.) operator is called the member access operator
  - Used to access members of a `struct`
- The only built-in operations on a `struct` are the assignment and member access operations
- Neither arithmetic nor relational operations are allowed on `struct`s
- A `struct` can be passed by value or reference
- A function can return a value of type `struct`
- A `struct` can be a member of another `struct`

Reading Assignment – Very Important for "GU – ECE 111"

- Malik, D.S., 2014. **C++ programming: Program design including data structures**. Cengage Learning.
  - "**Chapter 9**: **User-Defined Functions**".

CENGAGE Learning