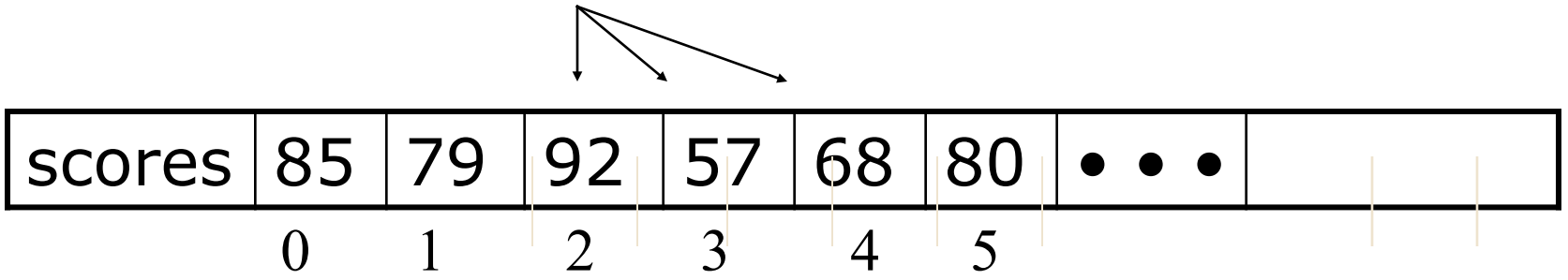
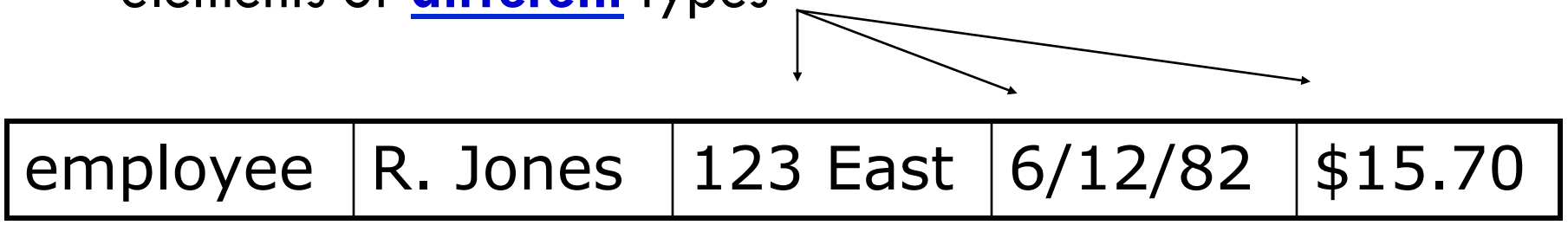
C++ STRUCTURES

C++ structures

- Recall that elements of arrays must all be of the **same** type



- In some situations, we wish to group elements of **different** types



C++ structures

- C++ structures are used to group related components of different types

Example data

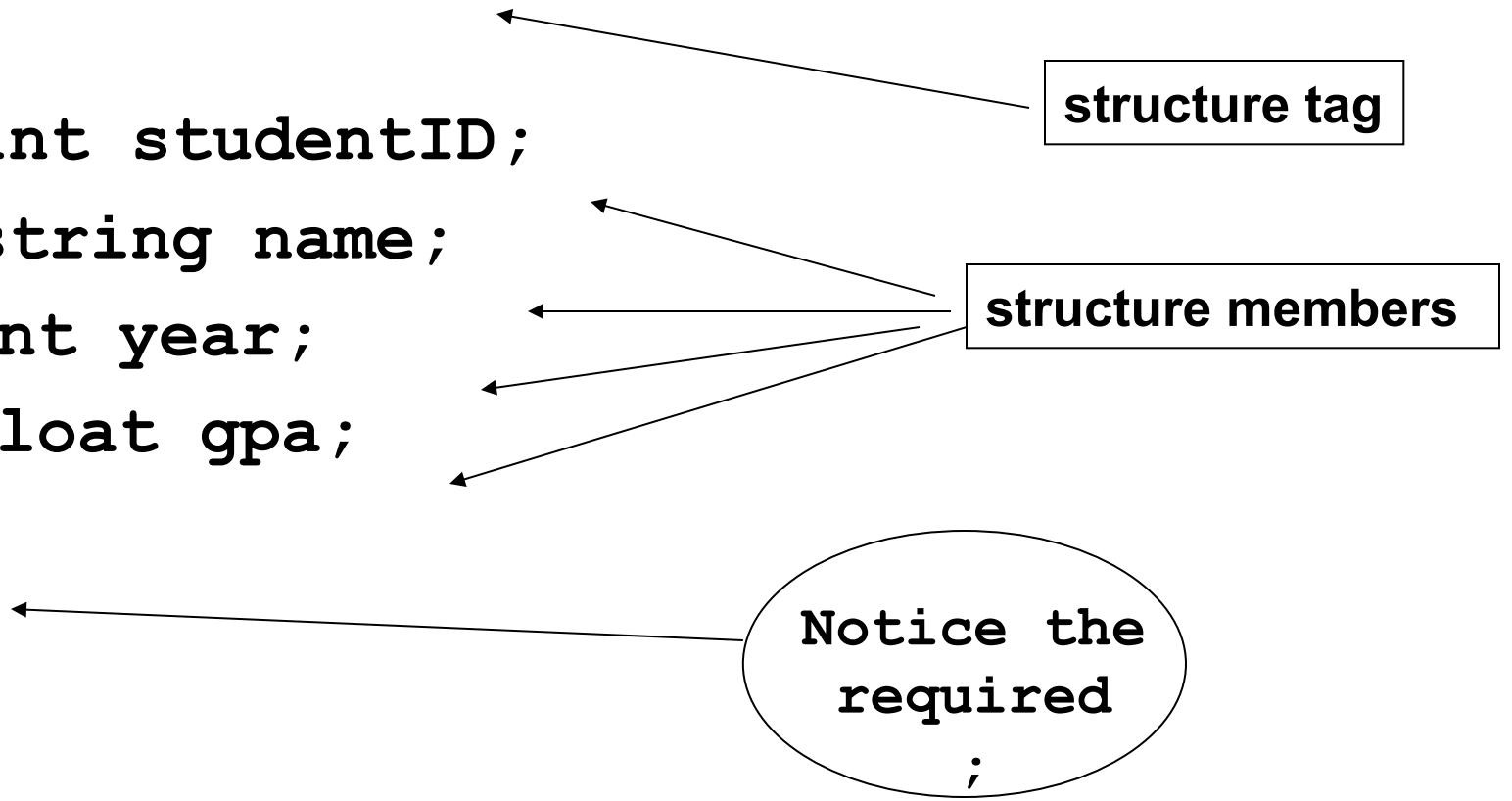
employee	R. Jones	123 East	6/12/82	$15.70

Structure

- A structure is a container, it can hold lots of *different things* unlike an array in which every element must be of the same type.
- Structures are used to organize related data into a single package.

Example `struct` Declaration

```
struct StudentType
{
   int studentID;
   string name;
   int year;
   float gpa;
};
```



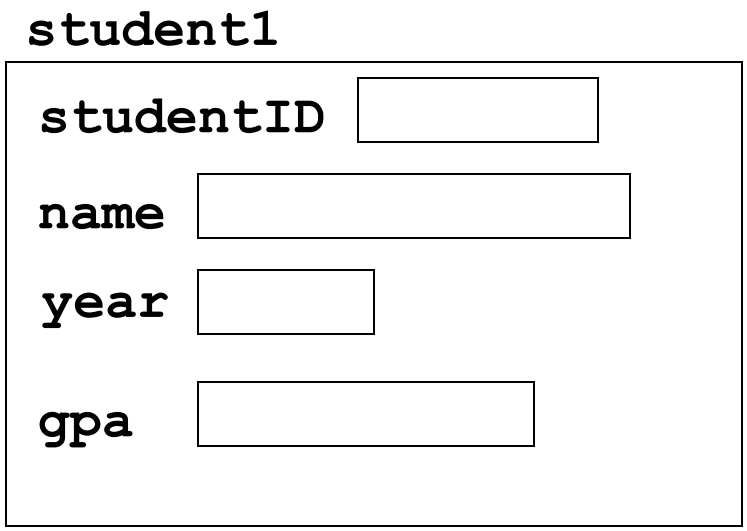
Important note about order of structure members

- Even though in the previous StudentType structure example, we declared the members studentID, name, year, and gpa in that order. When the compiler creates storage for the structure, the ORDER IS UNKNOWN and may be rearranged by the compiler!!

Defining Structure Variables

- **`struct`** declaration does NOT allocate memory or create variables
- To define variables, use structure tag as type name

```
Student student1;
```



Example - Student Record

- Student Record:
  - Name — a string
  - HW Grades — an array of 3 floats
  - Test Grades — an array of 2 floats
  - Final Average — a floats

Example Structure Definition

```
struct StudentRecord {
  string name;      // student name
  float hw[3];      // homework grades
  float test[2];    // test grades
  float average;    // final average
};
```

Accessing Members

- You can treat the members of a struct just like variables.
- You need to use the *member access operator* '.' (pronounced "dot"):

StudentRecord studentA;

```
cout << studentA.name << endl;
studentA.hw[2] = 82.3;
studentA.average = total/100;
```

Structure Assignment

- You can use structures just like variables:

```
StudentRecord student1,student2;
student1.name = "Joe Student";
Student1.average = 88.5;
…
student2 = student1;
```

Copies the entire structure

Displaying `struct` Members

- To display the contents of a **`struct`** variable, you must display each field separately, using the dot operator

Wrong:

```
cout << student1; // won't work!
```

Correct:

```
cout << student1.studentID << endl;
cout << student1.name << endl;
cout << student1.year << endl;
cout << student1.gpa;
```

Initializing a Structure

- Cannot initialize members in the structure declaration, because no memory has been allocated yet

```
struct Student      // Illegal
{                   // initialization
  int studentID = 1145;
  string name = "Alex";
  short year = 1;
  float gpa = 2.95;
};
```

NO NO NO NO NO

Aggregate Operations with Structures

- Recall that arrays had none

- Structures DO have some aggregate operators
  - assignment statement =
  - parameter (value or reference)
  - return a structure as a function type

Aggregate Operations with Structures

- Limitations on aggregate operations
  - no **I/O**

```
cout << old_part;
cin >> new_part;
```

  - no **arithmetic** operations

```
old_part = new_part + old_part;
```

  - no **comparisons**

```
if (old_part < new_part)
    cout << ...;
```

Aggregate Operations with Structures

- **`struct`** variables must be compared member-wise.
- To compare the values of **`student`** and **`newStudent`**, you must compare them member-wise, as follows:

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

Initializing a Structure (cont.)

- Structure members are initialized at the time a structure **variable** is created
- Can initialize a structure variable's members with C++ initialization code

```
void Initialize(StudentRecord  &oneStudent)
{
        oneStudent.ID=0;
        oneStudent.name="";
        oneStudent.credit = 0;
        …
}
```

Input/Output

- There are no aggregate input/output operations on `struct`.
  - Data in a `struct` variable must be read one member at a time.
  - Contents of a `struct` must be written one member at a time.

`struct` Variables and Functions

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

Arrays of structures

- First declare a struct (such as **`StudentType`**)
- Then specify an array of that type

  StudentType list[50];
- Access elements of the array, elements of the struct

  list[3].name =“bob”;

Testing and Debugging Hints

- Declaration of a **struct** type must end with a semicolon **;**
- Be sure to specify the full member selector when referencing a component of a struct variable
  - don't leave out the struct name

Testing and Debugging

- When using an array in a struct, the index goes at the end

```
student_rec.scores[x]
```

- When using an array of struct, the index goes after the struct name

```
list[x].name
```

Testing and Debugging

- Process struct members separately … the only aggregate operations will be

  Assignment =

- Parameter passing---use like any other type