Topic 7 – Structs (records)

Chapter 11 in the shrinkwrap

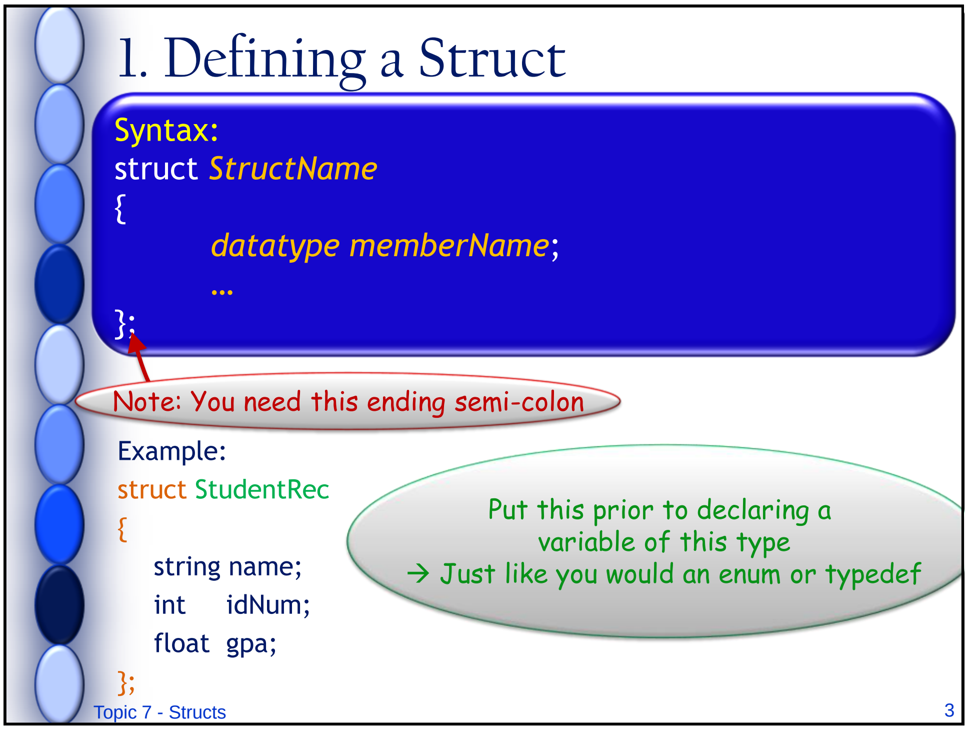
Related items - different datatypes

Before we used parallel arrays

→ there is a better way

- Structs or *Struct*ured Variables*(a.k.a.* records)
  - Allow us to store related variables in one structure – even if they have different datatypes.

- Basic Process
  1. Define the struct
     - Define the members of the structure
     - Each member has a datatype and a name
  2. Declare an identifier of your new struct type
  3. Use the struct



1. Defining a Struct

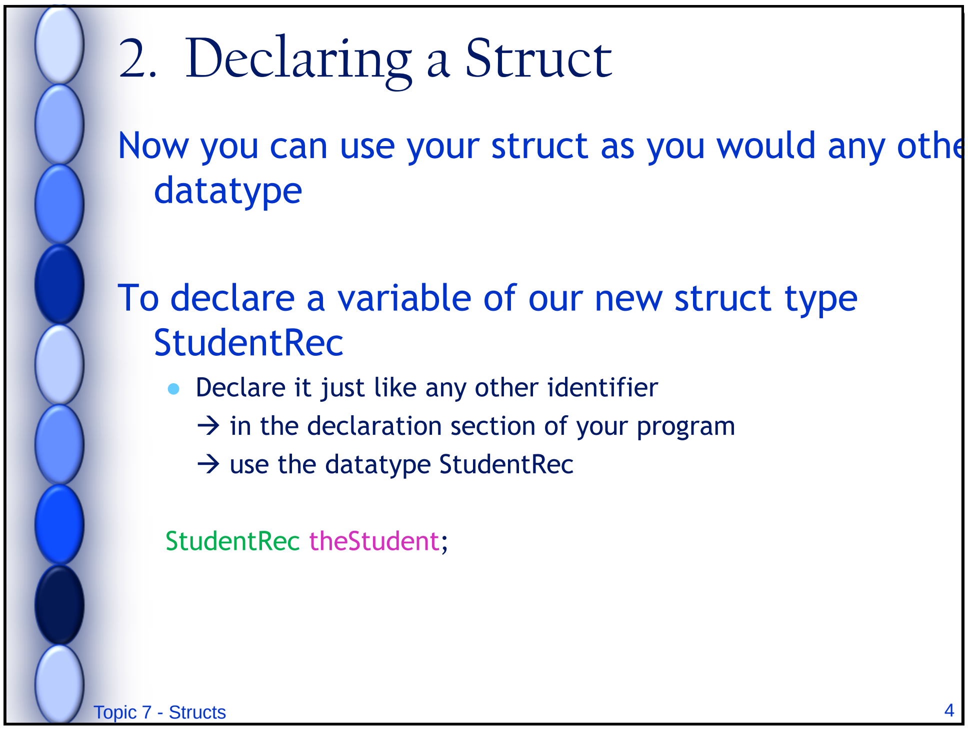
```
Syntax:
struct StructName
{
        datatype memberName;
        …
};
```

Note: You need this ending semi-colon

Example:

```
struct StudentRec
{
    string name;
    int    idNum;
    float  gpa;
};
```

Put this prior to declaring a variable of this type
→ Just like you would an enum or typedef



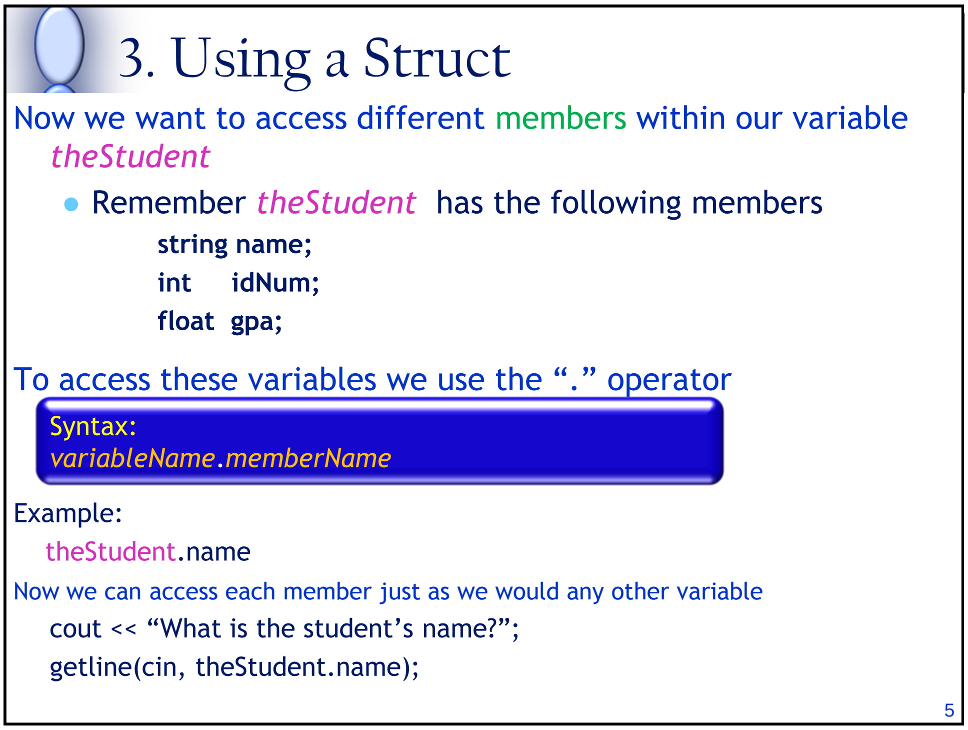
2. Declaring a Struct

Now you can use your struct as you would any othe
datatype

To declare a variable of our new struct type StudentRec

- Declare it just like any other identifier
  → in the declaration section of your program
  → use the datatype StudentRec

```
StudentRec theStudent;
```



3. Using a Struct

Now we want to access different members within our variable *theStudent*

- Remember *theStudent* has the following members

```
string name;
int    idNum;
float  gpa;
```

To access these variables we use the "." operator

Syntax:
variableName.*memberName*

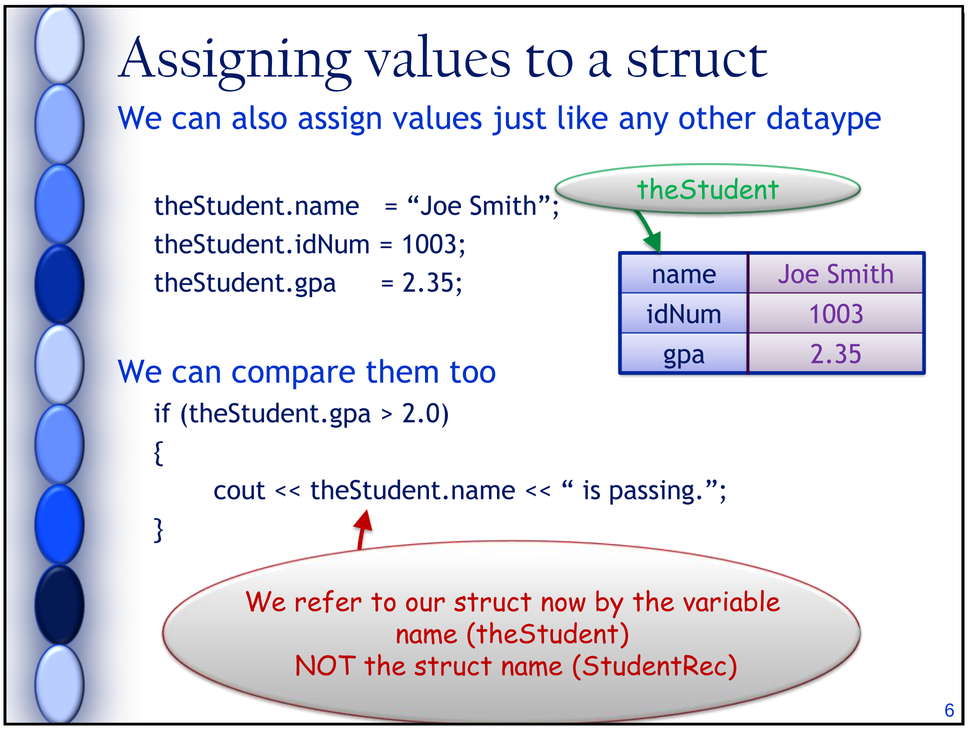
Example:

theStudent.name

Now we can access each member just as we would any other variable

```
cout << "What is the student's name?";
getline(cin, theStudent.name);
```

5



Assigning values to a struct

We can also assign values just like any other dataype

```
theStudent.name   = "Joe Smith";
theStudent.idNum = 1003;
theStudent.gpa     = 2.35;
```

theStudent

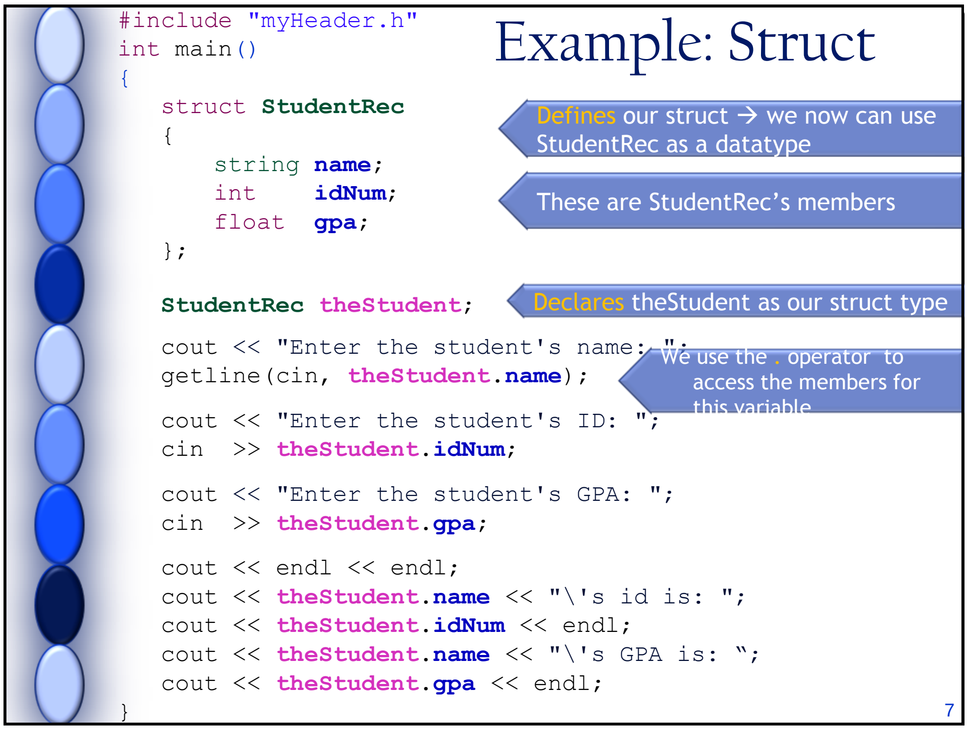
name	Joe Smith
idNum	1003
gpa	2.35

We can compare them too

```
if (theStudent.gpa > 2.0)
{
      cout << theStudent.name << " is passing.";
}
```

We refer to our struct now by the variable name (theStudent)
NOT the struct name (StudentRec)

6



Example: Struct

```
#include "myHeader.h"
int main()
{
    struct StudentRec
    {
        string name;
        int    idNum;
        float  gpa;
    };

    StudentRec theStudent;

    cout << "Enter the student's name: ";
    getline(cin, theStudent.name);

    cout << "Enter the student's ID: ";
    cin  >> theStudent.idNum;

    cout << "Enter the student's GPA: ";
    cin  >> theStudent.gpa;

    cout << endl << endl;
    cout << theStudent.name << "\'s id is: ";
    cout << theStudent.idNum << endl;
    cout << theStudent.name << "\'s GPA is: ";
    cout << theStudent.gpa << endl;
}
```

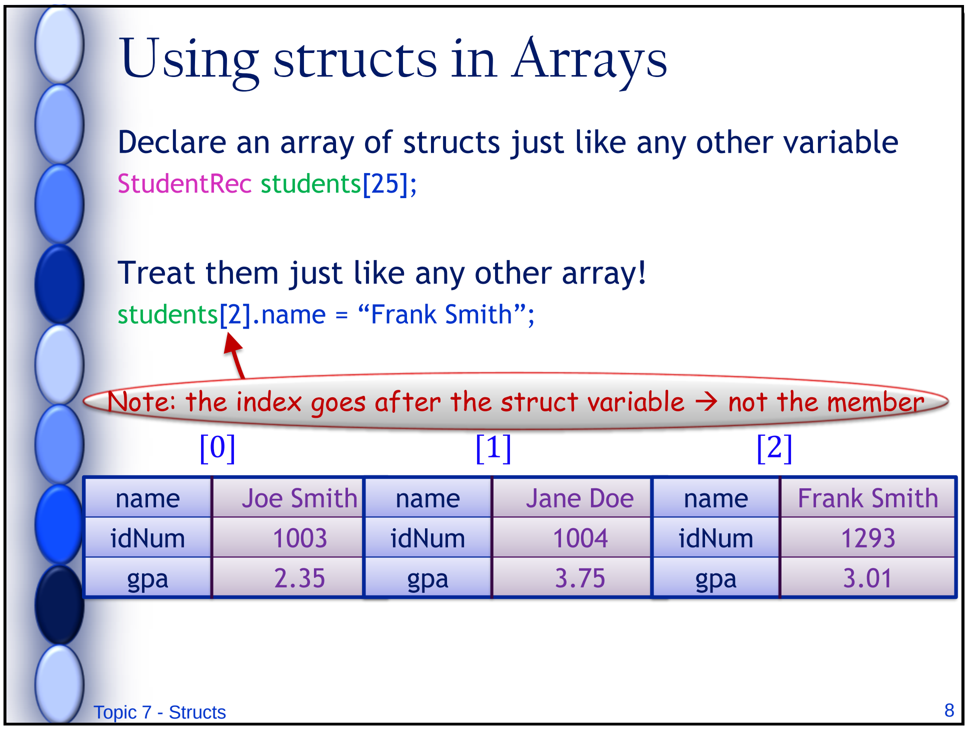
Defines our struct → we now can use StudentRec as a datatype

These are StudentRec's members

Declares theStudent as our struct type

We use the . operator to access the members for this variable

7



Using structs in Arrays

Declare an array of structs just like any other variable

StudentRec students[25];

Treat them just like any other array!

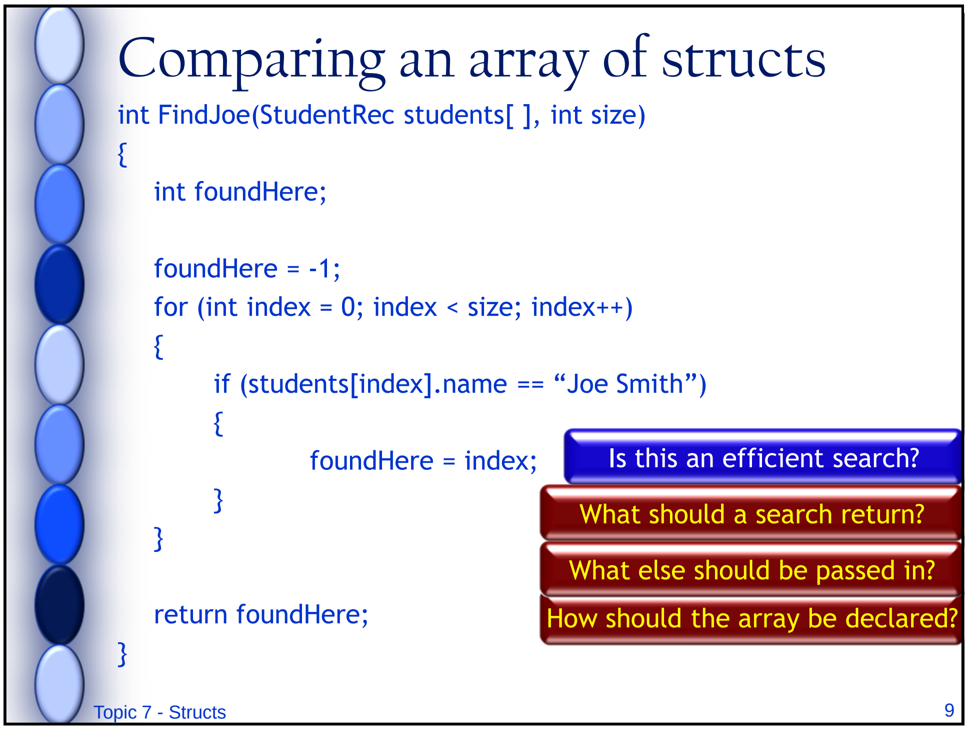
students[2].name = "Frank Smith";

Note: the index goes after the struct variable → not the member

[0]		[1]		[2]	
name	Joe Smith	name	Jane Doe	name	Frank Smith
idNum	1003	idNum	1004	idNum	1293
gpa	2.35	gpa	3.75	gpa	3.01

Topic 7 - Structs

8



Comparing an array of structs

```
int FindJoe(StudentRec students[ ], int size)
{
    int foundHere;

    foundHere = -1;
    for (int index = 0; index < size; index++)
    {
          if (students[index].name == “Joe Smith”)
          {
                    foundHere = index;
          }
    }

    return foundHere;
}
```

Is this an efficient search?

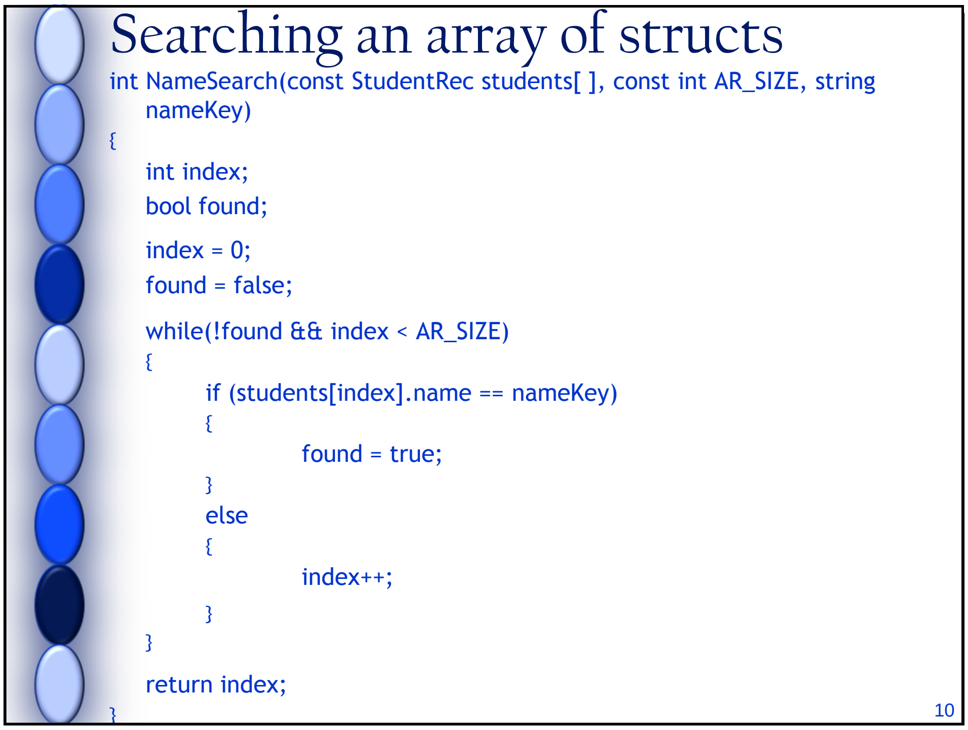
What should a search return?

What else should be passed in?

How should the array be declared?

Topic 7 - Structs

9



Searching an array of structs

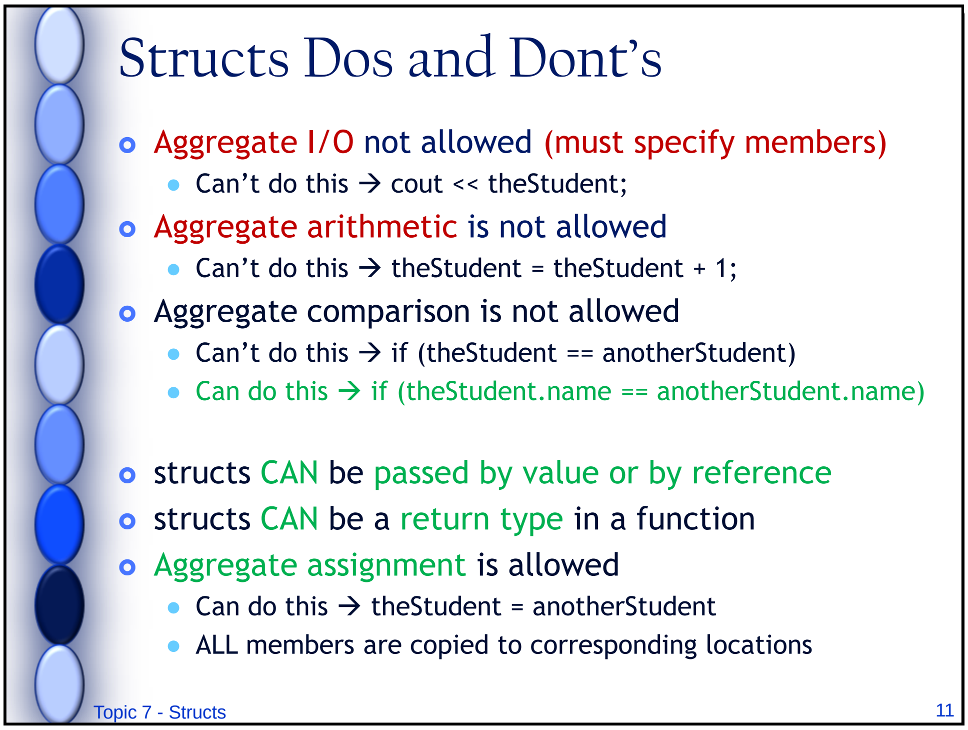
```
int NameSearch(const StudentRec students[ ], const int AR_SIZE, string
    nameKey)
{
    int index;
    bool found;

    index = 0;
    found = false;

    while(!found && index < AR_SIZE)
    {
          if (students[index].name == nameKey)
          {
                    found = true;
          }
          else
          {
                    index++;
          }
    }

    return index;
}
```

10



Structs Dos and Dont's

- Aggregate I/O not allowed (must specify members)
  - Can't do this → cout << theStudent;
- Aggregate arithmetic is not allowed
  - Can't do this → theStudent = theStudent + 1;
- Aggregate comparison is not allowed
  - Can't do this → if (theStudent == anotherStudent)
  - Can do this → if (theStudent.name == anotherStudent.name)
- structs CAN be passed by value or by reference
- structs CAN be a return type in a function
- Aggregate assignment is allowed
  - Can do this → theStudent = anotherStudent
  - ALL members are copied to corresponding locations

Topic 7 - Structs 11