Structures (Part 2)

Structures and the assignment operator

- With regular primitive types we have a wide variety of operations available, including assignment, comparisons, arithmetic, etc.
- Most of these operations would **NOT** make sense on structures. Arithmetic and comparisons, for example:

```
Student s1, s2;

s1 = s1 + s2;          // ILLEGAL!
                       // How would we add two students together, anyways?

if (s1 < s2)           // ILLEGAL.  What would this mean, anyways?
   // yadda yadda
```

- HOWEVER, using the assignment operator on structures **IS** legal, as long as they are the same type. Example (using previous struct definitions):

```
Student s1, s2;
Fraction f1, f2;

s1 = s2;               // LEGAL.  Copies contents of s2 into s1
f1 = f2;               // LEGAL.  Copies f2 into f1
```

- Note that in the above example, the two assignment statements are equivalent to doing the following:

```
strcpy(s1.fName, s2.fName);           // these 4 lines are equivalent to
strcpy(s1.lName, s2.lName);           //    s1 = s2;
s1.socSecNumber = s2.socSecNumber;
s1.gpa = s2.gpa;

f1.num = f2.num;                      // these 2 lines are equivalent to
f1.denom = f2.denom;                  //    f1 = f2;
```

  Clearly, direct assignment between entire structures is **easier**, if a full copy of the whole thing is the desired result!

Passing structures into and out of functions

- Just like a variable of a basic type, a structure can be passed into functions, and a structure can be returned from a function.
- To use structures in functions, use *`structname`* as the parameter type, or as a return type, on a function declaration
- Examples (assuming struct definition examples from previous page):

```
// function that passes a structure variable as a parameter
void PrintStudent(Student s);

// function that passes in structure variables and returns a struct
Fraction Add(Fraction f1, Fraction f2);
```

- structfunc1.cpp -- Here's an example using a function that takes a structure as a parameter

Pass by value, reference, address

- Just like with regular varaibles, structures can be passed by value or by reference, or a pointer to a structure can be passed (i.e. pass by address)
- If just a plain structure variable is passed, as in the above examples, it's pass by value. A **copy** of the structure is made
- To pass by reference, use the `&` on the structure type, just as with regular data types
- To pass by address, use *pointers to structures* as the parameters and/or return
- Examples (using the previously seen structure definitions):

```
// function that passes a pointer-to-student-structure as a parameter
void GetStudentData(Student* s);

// function that passes in structures by const reference, and returns a
//   struct by value
Fraction Add(const Fraction& f1, const Fraction& f2);

// function that uses const on a structure pointer parameter
//  this function could take in an array of Students, as well as the
//  address of a single student.
void PrintStudents(const Student* s);

// or, this prototype is equivalent to the one above
void PrintStudents(const Student s[]);
```

- As with pointers to the regular built-in types, you can use `const` to ensure a function cannot change the target of a pointer
- **It's often a GOOD idea to pass structures to and from functions by address or by reference**
  - structures are compound data, usually larger than plain atomic variables
  - Pass-by-value means copying a structure. **NOT** copying is desirable for efficiency, especially if the structure is very large
- students.cpp -- an example of structures passed by address into functions
- frac.cpp -- Example of fraction structure, along with some functions