Ch 11. Structured Data
Part 1 (11.2 to 11.8)

CS 2308
Fall 2011

Jill Seaman

Lecture 3

Data Types

- Data Type:
  - set of values
  - set of operations over those values
- example: Integer
  - whole numbers, -32768 to 32767
  - +, -, *, /, %, ==, !=, <, >, <=, >=, ...
- Which operation is not valid for float?

Data Types (C/C++)

- Scalar (or Basic) Data Types (atomic values)
  - Arithmetic types
    - Integers
      - short, int, long
      - char
    - Floating points
      - float, double, long double
- Composite (or Aggregate) Types:
  - Arrays: ordered sequence of values of the same type
  - Structs: named components of various types

Structures

- Used to represent a relationship between values of different types
- Example: student
  - ID Number
  - Name
  - Age
  - Major
  - Address

Structures

- Define this as a struct in C++:

```
struct Student {
   int idNumber;
   char name[25];
   int age;
   char major[25];
};
```

- NOTE: semicolon after last brace!
- A struct is a data type, by convention the name is capitalized.
- To define a variable of type Student:

```
Student csStudent;
```

Structures

- Can define multiple variables of type Student:

```
Student student1, student2, gradStudent;
```

- Each on has its own set of member variables

Accessing Structure Members

- Use dot notation to access members of a struct variable:

```
student1.age = 18;
student2.idNumber = 123456;
cin >> gradStudent.name;
```

- You can use member variables just like regular variables (of the same type).

```
student1.age++;
myFunc(student2.idNumber);
if (student1.age==student2.age) {
   ...
}
```

Structure: operations

- Valid operations over structs:
  - assignment: student1 = student2;
  - function call: myFunc(gradStudent,x);
- Invalid operations over structs:
  - comparison: student1 == student2
  - output: cout << student1;
  - input: cin >> student2;
  - Must do these member by member

Initializing structures

- Can initialize when variable is defined:

```
Student student1 = {123456,”John Smith”,22,”Mathematics”};
```

- Must give values in order of the struct declaration.
- Can NOT initialize members in struct declaration:

```
struct Student {
   int id = 123456;              //ILLEGAL
   char name[15] = “John Smith”; //ILLEGAL
}
```

- Why not?

Arrays of Structures

- You can store values of structured types in arrays.

```
Student roster[40];

//input a name
cout << “Enter the first student's name: “;
cin >> roster[0].name;

//...

//output all the id numbers and names
for (int i=0; i<40; i++) {
   cout << roster[i].idNumber << endl;
   cout << roster[i].name << endl;
}
```

Nested Structures

- You can nest one structure inside another.

```
struct Address {
   char street[25];
   char city[15];
   char state[2];
   int zip;
};

struct Student {
   int idNumber;
   char name[25];
   Address homeAddress;
};

Student student1;

cout << student1.homeAddress.state << endl;
```

Structures as function arguments

- Structure variables may be passed as arguments to functions.

```
void showStudent(Student x) {
   cout << x.idNumber << endl;
   cout << x.name << endl;
   cout << x.major << endl;
}


Student student1;

//input information about student1 here

showStudent(student1);
```

Structures as function arguments

- By default, structure variables are passed by value.
- If the function needs to change the value of a member, the structure variable should be passed by reference.

```
void happyBirthday(Student &s) {
   s.age++;
}
```

Returning Structure from Function

- A function may return a structure.

```
Student inputStudent() {
   Student result;

   ifile inFile;
   inFile.open(“students.dat”);

   inFile >> result.idNumber;
   inFile >> result.name;
   inFile >> result.age;
   inFile >> result.major;
   inFile.close();

   return result;
}

Student student1 = inputStudent();
```