

Structures

Unit 7

Gaddis: 11.2-8

CS 1428
Fall 2017

Jill Seaman

1

Data Types

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

2

Data Types (C/C++)

- Scalar (or Basic, Primitive) Data Types
 - ▶ atomic values, such as:
 - ▶ Integers:
 - ➔ short, int, long, char, bool
 - ▶ Floating Points:
 - ➔ float, double, long double
- Composite (or Aggregate) Types:
 - ▶ values of these types are composed from other values.
 - ▶ Arrays: ordered sequence of values of the same type
 - ▶ Structures: named components of various types

3

11.2 Structures

- Composite data type used to group multiple variables together into a unit.
- Example: student
 - ▶ ID Number
 - ▶ Name
 - ▶ Age
 - ▶ Major
- Each student has a value for each of these variables (or attributes).

4

Structures in C++

- Define the student as a struct in C++:

```
struct Student {  
    int idNumber;  
    string name;  
    int age;  
    string major;  
};
```

- NOTE: semicolon after last curly bracket!
- A struct is a data type, and by convention the name is capitalized.
- The components are called “members” (or “fields”).

5

Defining structure variables

- So far we have defined a new data type, but we haven't defined any variables of that type.
- To define a variable of type Student:

```
Student myStudent;
```

- Can define multiple variables of type Student:

```
Student student1, student2, aGradStudent;
```

- Each one has its own set of the member variables in the Student data type

6

Defining structure variables

- Each variable of type Student has its own set of the member variables from the Student data type

```
Student student1, student2;
```

student1	student2
idNumber <input type="text"/>	idNumber <input type="text"/>
name <input type="text"/>	name <input type="text"/>
age <input type="text"/>	age <input type="text"/>
major <input type="text"/>	major <input type="text"/>

7

11.3 Accessing Structure Members

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

```
student1.age = 18;  
student2.idNumber = 123456;  
cin >> aGradStudent.name;  
aGradStudent.major = "Rocket Science";
```

- Member variables of structures can be used just like regular variables of the same type.

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

8

Operations over structures:

- Valid operations over entire structs:
 - assignment: `student1 = student2;`
 - function call: `myFunc(gradStudent, x);`
- Invalid operations over entire structs:
 - comparison: `student1 == student2`
 - output: `cout << student1;`
 - input: `cin >> student2;`
 - Must do these member by member!
- How is this different from Arrays?

9

Outputting & comparing structure variables

- Output the members one at a time:

```
cout << student1.idNumber << " ";
cout << student1.name << " ";
cout << student1.age << " ";
cout << student1.major << endl;
```

Output: 11122 Chris Johnson 19 Chemistry

- Comparing two structs:

```
if (student1.idNumber == student2.idNumber &&
    student1.name == student2.name &&
    student1.age == student2.age &&
    student1.major == student2.major)
    ...
```

10

11.4 Initializing a Structure

- Struct variable can be initialized when it is defined:
`Student student1 = {123456, "John Smith", 22, "Math"};`
- Must give values of members in order of the struct declaration.
- Can NOT initialize members in structure declaration, only variable definition:

```
struct StudentA {
    int id = 123456;           //ILLEGAL
    string name = "John Smith"; //ILLEGAL
}
```

11

```
struct EmployeePay {
    string name;           // Employee name
    int empNum;           // Employee number
    double payRate;       // Hourly pay rate
    double hours;         // Hours worked
    double grossPay;      // Gross pay
};

int main() {
    EmployeePay employee1 = {"Betty Ross", 141, 18.75};
    EmployeePay employee2 = {"Jill Sandburg", 142, 17.50};

    cout << fixed << setprecision(2);

    // Calculate pay for employee1
    cout << "Name: " << employee1.name << endl;
    cout << "Employee Number: " << employee1.empNum << endl;
    cout << "Enter the hours worked by this employee: ";
    cin >> employee1.hours;
    employee1.grossPay = employee1.hours * employee1.payRate;
    cout << "Gross Pay: " << employee1.grossPay << endl << endl;

    // Calculate pay for employee2
    cout << "Name: " << employee2.name << endl;
    cout << "Employee Number: " << employee2.empNum << endl;
    cout << "Enter the hours worked by this employee: ";
    cin >> employee2.hours;
    employee2.grossPay = employee2.hours * employee2.payRate;
    cout << "Gross Pay: " << employee2.grossPay << endl;
}
```

Initializes only name, empNum, and payRate

12

Sample output from previous program:

```
Name: Betty Ross
Employee Number: 141
Enter the hours worked by this employee: 40 [Enter]
Gross Pay: 750.00

Name: Jill Sandburg
Employee Number: 142
Enter the hours worked by this employee: 20 [Enter]
Gross Pay: 350.00
```

13

11.5 Arrays of Structures

- You can store values of structure types in arrays.

```
Student roster[40]; //holds 40 Student structs
```

- Each student structure is accessible via the subscript notation:

```
roster[0] = student1; //copies student1 to first elem.
```

- Members of structure accessible via dot operator

```
cout << roster[0].name << endl;
```

14

Arrays of Structures

- Arrays of structures processed in loops:

```
Student roster[40];

//input
for (int i=0; i<40; i++) {
    cout << "Enter the name, age, idNumber and "
         << "major of the next student: \n";
    cin >> roster[i].name >> roster[i].age
        >> roster[i].idNumber >> roster[i].major;
}

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

15

11.6 Nested Structures

- You can nest one structure inside another.

```
struct Address {
    string street;
    string city;
    string state;
    int zip;
};

struct Student {
    int idNumber;
    string name;
    Address homeAddress;
};
```

16

Nested Structures

- Use dot operator multiple times to get into the nested structure:

```
Student student1;
student1.name = "Bob Lambert";
student1.homeAddress.city = "San Angelo";
student1.homeAddress.state = "TX";
```

- Or set up address structure separately:

```
Address a1;
a1.street = "101 Main St.";
a1.city = "San Angelo";
a1.state = "TX";
a1.zip = 76903;

student1.name = "Bob Lambert";
student1.homeAddress = a1;
```

17

11.7 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.age << endl;
    cout << x.major << endl;
}

int main() {
    Student student1;

    //input information about student1 here

    showStudent(student1);
}
```

Note: Student declaration must be global!!

18

Structures as function arguments

- By default, structure variables are passed by value (like most variables).
- 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++;          //or s.age = s.age+1;
}
```

19

11.8 Returning a Structure from a Function

- A function may return a structure.

```
Student inputStudent(ifstream &fin) {
    Student result;
    fin >> result.idNumber;
    fin >> result.name;
    fin >> result.age;
    fin >> result.major;
    return result;
}

int main() {
    ifstream inFile;
    inFile.open("students.dat");
    Student student1 = inputStudent(inFile);
    for (int i=0; i<40; i++)
        roster[i] = inputStudent(inFile);
    inFile.close();
}
```

Note: always pass iostreams by reference!!

20