**Lecture 25 - Chapter 10: C Structures, Unions, Bit Manipulation – Wed Nov 15 or Thurs Nov 16**

**Announcements**

Reading:

* Chapter 10

Assignments:

* Assign: Assignment #10 - due on **Nov 20** (MW class) or **Nov 21** (TR class) **(no late assignments accepted)**

**Today’s Goals**

1. Accessing Structures Members
2. Using Structures with Functions
3. Example

**Today’s Terminology**

**Terminology**

* Derived Data Type
  + Created from one or more types of basic data types
  + Most common are pointers, arrays, structures and unions
* Structures
  + Collection of related variables under one name
  + Derived data type
* Self-referential structure
  + When a structure contains as a member that is a pointer to its same structure type
* Union
  + Collection of related variables under one name, but only one at a time contains a value
  + Derived data type
* Bitwise Operators
  + & - AND
  + | - OR
  + ^ - Exclusive OR
* Bit Fields
  + Number of bits in which an unsigned or signed integral member of a structure or union is stored
* Enumeration
  + Set of integer constants represented by identifiers

**Accessing Structure Members**

**Accessing Structure Members**

* To access the individual members within a structure use
  + Structure member operator – dot operator (.) and
  + Structure pointer operator – arrow operator (->)

**Example: Structure Containing Another Structure**

* Can a structure contain a member that is another structure?
  + Yes, if that member is not itself!

**struct** student {

**char** name[30];

**double** examGrades[3];

}; These two structures have member

with the same name. This is legal!

**struct** class {

**char** name[20];

**int** numStudents;

**struct** student studentList[5]; Member that is a structure!

};

**void** **main**() { Create some students!

**struct** student csStudent1 = {"Ken Roczen", {65, 92, 72}};

**struct** student csStudent2 = {"Ryan Dungey", {98, 100, 78}};

**struct** student csStudent3 = {"Eli Tomac", {99, 95, 64}}; Create a class!

**struct** class csClass = {"CS2060", 3, {csStudent1, csStudent2, csStudent3}};

**printf** ("csClass.className = %s\n", csClass.name);

**printf** ("csClass.maxNumStduents = %d\n", csClass.numStudents);

Display students and their grades

**for** (**int** j = 0; j < csClass.numStudents; j++) {

**printf** ("csClass.studentList[%d].name = %s\n", j, csClass.studentList[j].name);

**for** (**int** k = 0; k < 3; k++) {

**printf** ("csClass.studentList[%d].grades = %.2f\n",

k, csClass.studentList[j].grades[k]);

}

**puts** ("");

}

} // main

**Displays**

csClass.className = CS2060

csClass.maxNumStduents = 3

csClass.studentList[0].name = Ken Roczen

csClass.studentList[0].grades = 65.00

csClass.studentList[1].grades = 92.00

csClass.studentList[2].grades = 72.00

csClass.studentList[1].name = Ryan Dungey

csClass.studentList[0].grades = 98.00

csClass.studentList[1].grades = 100.00

csClass.studentList[2].grades = 78.00

csClass.studentList[2].name = Eli Tomac

csClass.studentList[0].grades = 99.00

csClass.studentList[1].grades = 95.00

csClass.studentList[2].grades = 64.00

**Example: Array of Structures that Contain Another Structure**

**void** **main**() { **First we create several students**

**struct** student kenRoczen = {"Ken Roczen", {65, 92, 72}};

**struct** student ryanDungey = {"Ryan Dungey", {98, 100, 78}};

**struct** student eliTomac = {"Eli Tomac", {99, 95, 64}};

**struct** student andrewShort = {"Andrew Short", {99, 95, 64}};

**Create a class and add students**

**struct** class cs2060 = {"CS2060", 2, {kenRoczen, eliTomac}};

**struct** class cs1150 = {"CS1150", 3, {kenRoczen, ryanDungey, andrewShort}};

**struct** class cs4200 = {"CS4200", 2, {ryanDungey, eliTomac}};

**Now create an array of classes**

**struct** class csClasses[3] = {cs2060, cs1150, cs4200};

**Print the CS1150 students and**

**their grades**

**puts** ("Students in CS1150");

**for** (**int** k = 0; k < 3; k++) {

**printf** ("%s\n", csClasses[1].studentList[k].name);

**for** (**int** j = 0; j < 3; j++) {

**printf** ("%.2f\n", csClasses[1].studentList[k].grades[j]);

}

}

} // main

**Displays**

Students in CS1150

Ken Roczen

65.00

92.00

72.00

Ryan Dungey

98.00

100.00

78.00

Andrew Short

99.00

95.00

64.00

**Using Structures with Functions**

**Pass by Value vs Pass by Reference Review**

* Pass by value
  + Means a copy ***of the value/variable*** is sent to the function
* Pass by reference
  + Means the ***address to the array*** is sent to the function

**Passing Scalars vs. Arrays vs. Structures**

* Scalars - C uses ***pass by value***
* Arrays - C uses ***pass by reference***
* Structures – C uses ***pass by value***

**Passing Structures**

* The following can be done with structures when passing to functions
  + Pass individual members of structure
  + Pass entire structure
  + Pass pointer to structure – in this case ***pass by reference*** is used!
* What happens when passing an **array of structures** to a function?
  + The array is passed by reference!
* What happens when passing an **array that is a member of a structure**?
  + Structure is passed by value so array is passed by value!
* Note
  + Passing structures by reference is more efficient than passing structures by value
  + Why? Because when passing by value a copy of the entire structure must be created

**Passing Structure Example**

**#include** <stdio.h>

**struct** employee {

**char** firstName[20];

**char** lastName[20];

**unsigned** **int** age;

**char** gender;

**double** hourlySalary;

**struct** assistant \*employee;

};

**void** **passStructByValue** (**struct** employee aEmployee);

**void** **main**() {

**struct** employee aEmployee = { "Joe", "Smith" };

**puts** ("");

**puts** ("Passing struct to functions - pass by value");

**puts** ("Before function call");

**printf** ("Employee's age = %d\n", aEmployee.age);

**printf** ("Employee's hourly salary = %3.2f\n", aEmployee.hourlySalary);

passStructByValue (aEmployee);

**puts** ("After function call - show that nothing was changed");

**printf** ("Employee's age = %d\n", aEmployee.age);

**printf** ("Employee's hourly salary = %3.2f\n", aEmployee.hourlySalary);

**struct** employee \*employeePtr; // employeePtr will point to an employee type

employeePtr = &aEmployee; // employeePtr given address of aEmployee

**puts** ("");

**puts** ("Passing struct to functions - pass by reference");

**puts** ("Before function call");

**printf** ("Employee's age = %d\n", employeePtr->age);

**printf** ("Employee's hourly salary = %3.2f\n\n", employeePtr->hourlySalary);

passStructByReference (employeePtr);

**puts** ("After function call - show that nothing was changed");

**printf** ("Employee's age = %d\n", employeePtr->age);

**printf** ("Employee's hourly salary = %3.2f\n\n", employeePtr->hourlySalary);

};

**void** **passStructByValue** (**struct** employee aEmployee) {

**puts** ("In function call - change values in the structure");

aEmployee.age = 25;

aEmployee.hourlySalary = 30.00;

**printf** ("Employee's age = %d\n", aEmployee.age);

**printf** ("Employee's hourly salary = %3.2f\n\n", aEmployee.hourlySalary);

} // passStructByValue

**void** **passStructByReference** (**struct** employee \*employeePtr) {

**puts** ("In function call - change values in the structure");

employeePtr->age = 25;

employeePtr->hourlySalary = 30.00;

**printf** ("Employee's age = %d\n", employeePtr->age);

**printf** ("Employee's hourly salary = %3.2f\n\n", employeePtr->hourlySalary);

} // passStructByReference

**Displays**

**Passing struct to functions - pass by value**

Before function call

Employee's age = 0

Employee's hourly salary = 0.00

In function call - change values in the structure

Employee's age = 25

Employee's hourly salary = 30.00

After function call - show that nothing was changed

Employee's age = 0

Employee's hourly salary = 0.00

**Passing struct to functions - pass by reference**

Before function call

Employee's age = 0

Employee's hourly salary = 0.00

In function call - change values in the structure

Employee's age = 25

Employee's hourly salary = 30.00

After function call - show that nothing was changed

Employee's age = 25

Employee's hourly salary = 30.00

**Passing Array of Structures Example**

* How are arrays passed to functions - arrays are passed by reference
* So an array of structures is passed by reference

**void** **passArrayOfStructByReference** (**struct** employee employeeList[]);

**void** **main**() { Create array of structures

using initializer list

**struct** employee employeeA = { "Joe", "Smith", 23, 'm', 15.0 };

**struct** employee employeeB = { "Sally", "Lee", 33, 'f', 17.0 };

**struct** employee employeeC = { "Jeff", "Jones", 45, 'm', 37.0 };

**struct** employee myEmployees[3] = { employeeA, employeeB, employeeC };

Could also create this way

myEmployees[0] = employeeA;

myEmployees[1] = employeeB; Assignment is allowed with

myEmployees[2] = employeeC; structures of same type

**for** (**int** i = 0; i < 3; i ++) {

**printf** ("myEmployees[%d].firstName = %s\n", i, myEmployees[i].firstName);

**printf** ("myEmployees[%d].hourlySalary = %5.2f\n", i, myEmployees[i].hourlySalary);

}

passArrayOfStructByReference (myEmployees); Passing array by reference

**QUESTION:** Will this change

Values inside the structure

**for** (**int** i = 0; i < 3; i ++) {

**printf** ("myEmployees[%d].firstName = %s\n", i, myEmployees[i].firstName);

**printf** ("myEmployees[%d].hourlySalary = %5.2f\n", i, myEmployees[i].hourlySalary);

}

}

**void** **passArrayOfStructByReference** (**struct** employee employeeList[]) {

**puts** ("In function - change everyone’s pay to $25");

**for** (**int** i = 0; i < 3; i++) {

employeeList[i].hourlySalary = 25.00;

}

**for** (**int** i = 0; i < 3; i++) {

**printf** ("First name = %s\n", employeeList[i].firstName);

**printf** ("Hourly salary = %3.2f\n\n", employeeList[i].hourlySalary);

}

} // passArrayOfStructByReference

**Displays**

myEmployees[0].firstName = Joe

myEmployees[0].hourlySalary = 15.00

myEmployees[1].firstName = Sally

myEmployees[1].hourlySalary = 17.00

myEmployees[2].firstName = Jeff

myEmployees[2].hourlySalary = 37.00

------------------------------------------

In function - change everyone’s pay to $25

First name = Joe

Hourly salary = 25.00

First name = Sally Printed in main

Hourly salary = 25.00

First name = Jeff

Hourly salary = 25.00

------------------------------------------

**Passing array by reference**

myEmployees[0].firstName = Joe **caused changes to structure**

myEmployees[0].hourlySalary = 25.00 **values to be seen in main**

myEmployees[1].firstName = Sally

myEmployees[1].hourlySalary = 25.00

myEmployees[2].firstName = Jeff

myEmployees[2].hourlySalary = 25.00 No different than passing

array of ints by reference!

**Passing a Structure with an Array Example**

* How are structures passed to functions?
  + Structure is passed by value
* If a structure contains an array member, how is it passed?
  + Array is now passed by value!

**struct** student {

**char** name[50];

**double** grades[3];

}; **Initialize structure and array member!**

**void** **main**() {

**struct** student mathStudent = {"Mike Collins", {98, 95, 78}};

**printf** ("Name = %s\n", mathStudent.name);

**for** (**int** i = 0; i < 3; i++) {

**printf** ("Grades[%d] = %.2f\n", i, mathStudent.grades[i]);

}

passStructWithArrayMember (mathStudent);

**printf** ("Name = %s\n", mathStudent.name);

**for** (**int** i = 0; i < 3; i++) {

**printf** ("ExamGrades[%d] = %.2f\n", i, mathStudent.grades[i]);

}

} // main **Structure is passed**

**by value, but it**

**void** **passStructWithArrayMember** (**struct** student aStudent) { **contains an array**

**puts** ("In function - change the structure's array member");

**for** (**int** i = 0; i < 3; i++) { **What about this**

aStudent.grades[i] = 100.00; **array**?

}

**printf** ("Name = %s\n", aStudent.name);

**for** (**int** i = 0; i < 3; i++) {

**printf** ("Grades[%d] = %.2f\n", i, aStudent.grades[i]);

}

} // passArrayOfStructByReference

**Displays**

Name = Mike Collins

Grades[0] = 98.00

Grades[1] = 95.00

Grades[2] = 78.00

---------------------------------------------------

In function - change the structure's array member

Name = Mike Collins

Grades[0] = 100.00 Printed in main

Grades[1] = 100.00

Grades[2] = 100.00

---------------------------------------------------

Name = Mike Collins **Passing structure by**

Grades[0] = 98.00 **value caused array**

Grades[1] = 95.00 **to be passed by**

Grades[2] = 78.00 **value!**

**typedef**

**typedef**

* Used with structs to create aliases for the struct data type

**Example**

* Here we define a new data type called employee

**struct** student {

**char** name[50];

**double** grades[3];

};

And we declare variable like this:

**struct** student csStudent1 = {"Ken Roczen", {65, 92, 72}};

* With typedef we can create an alias like this

**typedef** **struct** {

**char** name[50];

**double** grades[3]; Here we’ve now created an **alias** for this data

} Student; type. This data type is called “Student”

Now we can define variables of this type (Student)

Eliminated **struct student** and now can

just say **Student**! A lot cleaner!

Student aStudent = {"Tim Smith", {74, 65, 89}};

**printf** ("Name = %s\n", aStudent.name);

**for** (**int** i = 0; i < 3; i++) {

**printf** ("Grade[%d] = %.2f\n", i, aStudent.grades[i]);

}

**Notes**

* Capitalize the first letter of typedef names – tells us they are aliases for another type name
* Typedef **DOES NOT** create a new type
* Typedef **DOES** create a new *type name*
  + A lot easier to say ***Student*** than ***struct student***
  + Adds to readability and maintainability