**Chapter 11**

**Structures**

**Overview**

* Structure: C++ construct that allows multiple variables to be grouped together
* General Format:

struct *structName*

{

*type****1*** *field****1***;

*type****2*** *field****2***;

. . .

};

**Example:**

struct Student

{

int studentID;

char name[50];

double gpa;

};

struct names (i.e., *declarations*) commonly begin with uppercase letter. All subsequent letters are lowercase. struct names consisting of multiple words have the 1st letter of the *next* word as uppercase:

**Example:**

StudentAccounts

**Declaring Structure Variables**

Student stu1;

Student stu2 = {1,”Joe Smith”,3.5}; //Use braces only when declaring/initializing

//structure variable

**Accessing Structure Members**

Use the dot (.) operator to refer to members of struct variables:

cout << “Enter Student ID”;

cin >> stu1.studentID;

//Assume next statements **out of context** (not followed by cin >>)

cout << “Enter Student Name: “;

cin.getline(stu1.name, sizeof(stu1.name));

//Initialize gpa to 3.75

stu1.gpa = 3.75;

* To display the contents of a struct variable, must display each field separately, using the dot operator:

cout << stu1; **// won’t compile!**

cout << stu1.studentID << endl;

cout << stu1.name << endl;

cout << " " << stu1.gpa << endl;

* Cannot compare struct variables directly:

if (stu1 == stu2) // **won’t compile**

* Instead, must compare on a field basis:

if (stu1.studentID == stu1.studentID) ...

**Array of Structures**

* Can be used in place of parallel arrays  
  const int NUM\_STUDENTS = 20;  
  Student stuList[NUM\_STUDENTS];
* Individual structures accessible using subscript notation
* Fields within structures accessible using dot notation:

cout << stuList[5].studentID;

**Nested Structures**

struct PersonInfo

{

char name[50];

char address[75],

city[25];

};

struct Student

{

int studentID;

PersonInfo pData;

double gpa;

};

Student s;

cout << Enter Student Name: “;

cin.getline(s.pData.name,sizeof(s.pData.name));

cout << “Enter Student Address: “;

cin.getline(s.pData.address,sizeof(s.pData.address));

cout << “Enter Student City: “;

cin.getline(s.pData.city,sizeof(s.pData.city));

//Declare/Initialize Nested Structure

Student s2 = {2,{“Joe Smith,”1215 Main”,”Conway”},3.5};

**Structures as Function Arguments**

* May pass members of struct variables to functions:

computeGPA(stu.gpa);

* May pass entire struct variables to functions:

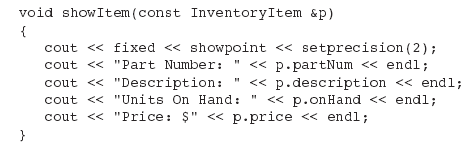
showData(stu);

* Can use reference parameter if function needs to modify contents of structure variable

**Pass by Value Example:**

****

**Pass by Reference Example**

****

**Returning a structure from a Function**

* Function can return a struct:

Student getStudentData(); // prototype

stu1 = getStudentData(); // call

* Function must define a local structure
  + for internal use

for use with return statement

Student getStudentData()

{

Student tempStu;

char buffer[255];

cout << “Enter name: “;

cin.getline(tempStu.pData.name, sizeof(tempStu.pData.name));

cout << “Enter address: “;

cin.getline(tempStu.pData.address, sizeof(tempStu.pData.address));

cout << “Enter city: “;

cin.getline(tempStu.pData.city, sizeof(tempStu.pData.city));

//Read id as an array of characters

cout << “Enter id: “;

cin.getline(buffer,sizeof(buffer));

tempStu.studentID = atoi(buffer);

return tempStu;

}

**Pointers to Structures**

* A structure variable has an address
* Pointers to structures are variables that can hold the address of a structure:

Student \*stuPtr;

* Can use & operator to assign address:

stuPtr = & stu1;

* Can use new to dynamically allocate an array of structures

stuPtr = new Student[no\_students];

**Accessing Fields of Pointer Variables to Structures**

* can use \* to dereference pointer variable:

cout << (\*stuPtr).studentID;

* Can use structure pointer operator (->) to use clearer notation:

cout << stuPtr->studentID;