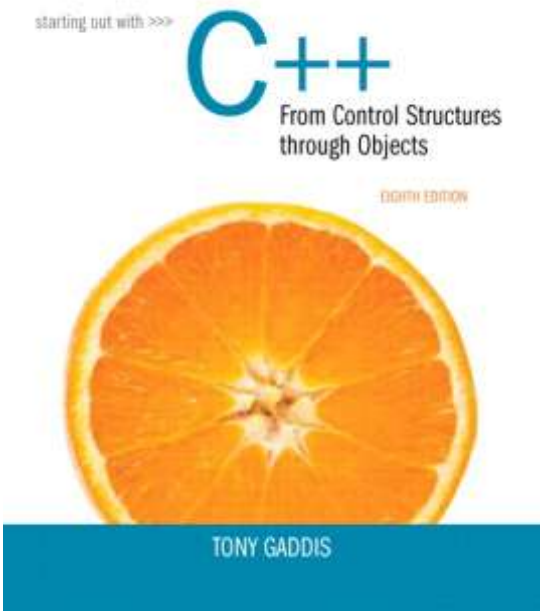
# Chapter 11:

## Structured Data






## 11.2

## Combining Data into Structures

Addison-Wesley is an imprint of PEARSON

Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

## Combining Data into Structures

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

```
struct <structName>
{
   type1 field1;
   type2 field2;
   . . .
};
```



## Example `struct` Declaration

```
struct Student
{
   int studentID;
   string name;
   short yearInSchool;
   double gpa;
};
```

structure tag

structure members

## `struct` Declaration Notes

- Must have `;` after closing `}`
- `struct` names commonly begin with uppercase letter
- Multiple fields of same type can be in comma-separated list:

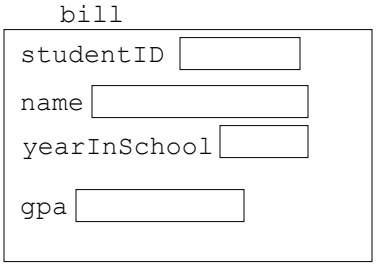
```
string name,
       address;
```



## Defining Variables

- `struct` declaration does not allocate memory or create variables
- To define variables, use structure tag as type name:

```
Student bill;
```





# 11.3

# Accessing Structure Members



## Accessing Structure Members

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

```
cin >> stu1.studentID;
getline(cin, stu1.name);
stu1.gpa = 3.75;
```

- Member variables can be used in any manner appropriate for their data type

Addison-Wesley is an imprint of PEARSON

Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

**Program 11-1**

```
// This program demonstrates the use of structures.
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

struct PayRoll
{
   int empNumber;     // Employee number
   string name;       // Employee's name
   double hours;      // Hours worked
   double payRate;    // Hourly payRate
   double grossPay;   // Gross pay
};

int main()
{
   PayRoll employee; // employee is a PayRoll structure.

   // Get the employee's number.
   cout << "Enter the employee's number: ";
   cin >> employee.empNumber;

   // Get the employee's name.
   cout << "Enter the employee's name: ";
```



```
   cin.ignore();  // To skip the remaining '\n' character
   getline(cin, employee.name);

   // Get the hours worked by the employee.
   cout << "How many hours did the employee work? ";
   cin >> employee.hours;

   // Get the employee's hourly pay rate.
   cout << "What is the employee's hourly payRate? ";
   cin >> employee.payRate;

   // Calculate the employee's gross pay.
   employee.grossPay = employee.hours * employee.payRate;

   // Display the employee data.
   cout << "Here is the employee's payroll data:\n";
   cout << "Name: " << employee.name << endl;
   cout << "Number: " << employee.empNumber << endl;
   cout << "Hours worked: " << employee.hours << endl;
   cout << "Hourly payRate: " << employee.payRate << endl;
   cout << fixed << showpoint << setprecision(2);
   cout << "Gross Pay: $" << employee.grossPay << endl;
   return 0;
}
```

**Program Output with Example Input Shown in Bold**

```
Enter the employee's number: 489 [Enter]
Enter the employee's name: Jill Smith [Enter]
How many hours did the employee work? 40 [Enter]
What is the employee's hourly pay rate? 20 [Enter]
Here is the employee's payroll data:
Name: Jill Smith
Number: 489
Hours worked: 40
Hourly pay rate: 20
Gross pay: $800.00
```



## Displaying a `struct` Variable

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

```
cout << bill; // won't work
cout << bill.studentID << endl;
cout << bill.name << endl;
cout << bill.yearInSchool;
cout << " " << bill.gpa;
```

## Comparing `struct` Variables

- Cannot compare `struct` variables directly:

```
if (bill == william) // won't work
```

- Instead, must compare on a field basis:

```
if (bill.studentID ==
                william.studentID) ...
```





# 11.4

# Initializing a Structure

## Initializing a Structure

- `struct` variable can be initialized when defined:

```
Student s = {11465, "Joan", 2, 3.75};
```

- Can also be initialized member-by-member after definition:

```
s.name = "Joan";
s.gpa = 3.75;
```



## More on Initializing a Structure

- May initialize only some members:

```
Student bill = {14579};
```

- Cannot skip over members:

```
Student s = {1234, "John", ,
                    2.83}; // illegal
```

- Cannot initialize in the structure declaration, since this does not allocate memory

## Excerpts From Program 11-3

```
 8  struct EmployeePay
 9  {
10     string name;          // Employee name
11     int empNum;           // Employee number
12     double payRate;       // Hourly pay rate
13     double hours;         // Hours worked
14     double grossPay;      // Gross pay
15  };

19     EmployeePay employee1 = {"Betty Ross", 141, 18.75};
20     EmployeePay employee2 = {"Jill Sandburg", 142, 17.50};
```




# 11.5

# Arrays of Structures

Addison-Wesley is an imprint of PEARSON

Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

## Arrays of Structures

- Structures can be defined in arrays
- 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;
```



**Program 11-4**

```
1  // This program uses an array of structures.
2  #include <iostream>
3  #include <iomanip>
4  using namespace std;
5
6  struct PayInfo
7  {
8     int hours;        // Hours worked
9     double payRate;   // Hourly pay rate
10 };
11
12 int main()
13 {
14    const int NUM_WORKERS = 3;    // Number of workers
15    PayInfo workers[NUM_WORKERS]; // Array of structures
16    int index;                    // Loop counter
17
```

```
18      // Get employee pay data.
19      cout << "Enter the hours worked by " << NUM_WORKERS
20           << " employees and their hourly rates.\n";
21
22      for (index = 0; index < NUM_WORKERS; index++)
23      {
24         // Get the hours worked by an employee.
25         cout << "Hours worked by employee #" << (index + 1);
26         cout << ": ";
27         cin >> workers[index].hours;
28
29         // Get the employee's hourly pay rate.
30         cout << "Hourly pay rate for employee #";
31         cout << (index + 1) << ": ";
32         cin >> workers[index].payRate;
33         cout << endl;
34      }
35
36      // Display each employee's gross pay.
37      cout << "Here is the gross pay for each employee:\n";
38      cout << fixed << showpoint << setprecision(2);
39      for (index = 0; index < NUM_WORKERS; index++)
40      {
41         double gross;
42         gross = workers[index].hours * workers[index].payRate;
43         cout << "Employee #" << (index + 1);
44         cout << ": $" << gross << endl;
45      }
46      return 0;
47   }
```



**Program Output with Example Input Shown in Bold**

```
Enter the hours worked by 3 employees and their hourly rates.
Hours worked by employee #1: 10 [Enter]
Hourly pay rate for employee #1: 9.75 [Enter]

Hours worked by employee #2: 20 [Enter]
Hourly pay rate for employee #2: 10.00 [Enter]

Hours worked by employee #3: 40 [Enter]
Hourly pay rate for employee #3: 20.00 [Enter]

Here is the gross pay for each employee:
Employee #1: $97.50
Employee #2: $200.00
Employee #3: $800.00
```

Addison-Wesley is an imprint of PEARSON

Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.


# 11.6

# Nested Structures



## Nested Structures

A structure can contain another structure as a member:

```
struct PersonInfo
{       string name,
        address,
        city;
};

struct Student
{       int studentID;
        PersonInfo pData;
        short yearInSchool;
        double gpa;
};
```

Addison-Wesley is an imprint of PEARSON
Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

## Members of Nested Structures

- Use the dot operator multiple times to refer to fields of nested structures:

```
Student s;
s.pData.name = "Joanne";
s.pData.city = "Tulsa";
```





# 11.7

# Structures as Function Arguments

## 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



## Excerpts from Program 11-6

```
 8  struct InventoryItem
 9  {
10     int partNum;                 // Part number
11     string description;          // Item description
12     int onHand;                  // Units on hand
13     double price;                // Unit price
14  };
```

```
61  void showItem(InventoryItem p)
62  {
63     cout << fixed << showpoint << setprecision(2);
64     cout << "Part Number: " << p.partNum << endl;
65     cout << "Description: " << p.description << endl;
66     cout << "Units On Hand: " << p.onHand << endl;
67     cout << "Price: $" << p.price << endl;
68  }
```

## Structures as Function Arguments - Notes

- Using value parameter for structure can slow down a program, waste space
- Using a reference parameter will speed up program, but function may change data in structure
- Using a `const` reference parameter allows read-only access to reference parameter, does not waste space, speed



## Revised `showItem` Function

```
void showItem(const InventoryItem &p)
{
   cout << fixed << showpoint << setprecision(2);
   cout << "Part Number: " << p.partNum << endl;
   cout << "Description: " << p.description << endl;
   cout << "Units On Hand: " << p.onHand << endl;
   cout << "Price: $" << p.price << endl;
}
```


# 11.8

## Returning a Structure from a Function



### 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

Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

## Returning a Structure from a Function - Example

```
Student getStudentData()
{
    Student tempStu;
    cin >> tempStu.studentID;
    getline(cin, tempStu.pData.name);
    getline(cin, tempStu.pData.address);
    getline(cin, tempStu.pData.city);
    cin >> tempStu.yearInSchool;
    cin >> tempStu.gpa;
    return tempStu;
}
```



**Program 11-7**

```
1  // This program uses a function to return a structure. This
2  // is a modification of Program 11-2.
3  #include <iostream>
4  #include <iomanip>
5  #include <cmath>  // For the pow function
6  using namespace std;
7
8  // Constant for pi.
9  const double PI = 3.14159;
10
11 // Structure declaration
12 struct Circle
13 {
14    double radius;      // A circle's radius
15    double diameter;    // A circle's diameter
16    double area;        // A circle's area
17 };
18
19 // Function prototype
20 Circle getInfo();
21
22 int main()
23 {
24    Circle c;      // Define a structure variable
```

```
25
26      // Get data about the circle.
27      c = getInfo();
28
29      // Calculate the circle's area.
30      c.area = PI * pow(c.radius, 2.0);
31
32      // Display the circle data.
33      cout << "The radius and area of the circle are:\n";
34      cout << fixed << setprecision(2);
35      cout << "Radius: " << c.radius << endl;
36      cout << "Area: " << c.area << endl;
37      return 0;
38  }
39
```



```
40  //*************************************************************
41  // Definition of function getInfo. This function uses a local   *
42  // variable, tempCircle, which is a circle structure. The user  *
43  // enters the diameter of the circle, which is stored in        *
44  // tempCircle.diameter. The function then calculates the radius *
45  // which is stored in tempCircle.radius. tempCircle is then     *
46  // returned from the function.                                  *
47  //*************************************************************
48
49  Circle getInfo()
50  {
51     Circle tempCircle;  // Temporary structure variable
52
53     // Store circle data in the temporary variable.
54     cout << "Enter the diameter of a circle: ";
55     cin >> tempCircle.diameter;
56     tempCircle.radius = tempCircle.diameter / 2.0;
57
58     // Return the temporary variable.
59     return tempCircle;
60  }
```

**Program Output with Example Input Shown in Bold**

```
Enter the diameter of a circle: 10 [Enter]
The radius and area of the circle are:
Radius: 5.00
Area: 78.54
```