Chapter 11 Structured Data

11.1 Abstract Data Types

Concept:

Abstract data types (ADTs) are data types created by the programmer. ADTs have their own range (or domain) of data and their own sets of operations that may be performed on them.

- The term **abstract data type (ADT)** is a foundational concept in computer science, particularly in object-oriented programming.
- This chapter introduces the C++ `struct`, a tool for creating your own ADTs.

Abstraction

- An **abstraction** is a simplified, general model of a concept, focusing only on its essential characteristics.
- For example, the word "dog" is an abstraction that captures the general idea of a dog without detailing any specific breed.
- In reality, dogs are concrete—they are specific breeds like poodles or retrievers, not just the abstract idea of a "dog."

Data Types

- C++ includes built-in **primitive data types** that are a fundamental part of the language.

Table 11-1 Primitive Data Types

`BOOL`	`SHORT INT`	`LONG LONG INT`
`char`	`unsigned short int`	`unsigned long long int`
`char_16_t`	`int`	
`char_32_t`	`unsigned int`	`float`
`wchar_t`	`long int`	`double`

`BOOL`	`SHORT INT`	`LONG LONG INT`
`unsigned char`	`unsigned long int`	`long double`

- A data type serves two main purposes:
    - It defines the **range of values** a variable can hold (e.g., an `int` cannot hold a fraction).
    - It defines the **operations** that can be performed on those values (e.g., the modulus operator `%` works only on integers).
- Primitive data types are abstract. For instance, `int` is the abstraction, while the variables defined from it are concrete instances.

```
int x = 1, y = 2, z = 3;
```

- In this code, `x`, `y`, and `z` are three distinct, concrete instances of the abstract data type `int`.

Abstract Data Types

- An **Abstract Data Type (ADT)** is a custom data type created by a programmer.
- It is typically composed of one or more primitive data types.
- The programmer defines the acceptable values and the specific operations for the ADT.
- For example, a `Clock` program could have ADTs for `Hours`, `Minutes`, and `Seconds`, each with its own specific range (1-12 for `Hours`, 0-59 for `Minutes` and `Seconds`) and rules for operations like incrementing.

11.2 Structures

Concept:

C++ allows you to group several variables together into a single item known as a structure.

- While arrays group items of the same data type, a **structure** can group related variables of *different* data types.
- This is useful for bundling data that belongs together, such as an employee's records.

Table 11-2 Variables in a Payroll System

VARIABLE DEFINITION	DATA HELD
`int empNumber;`	Employee number
`string name;`	Employee's name
`double hours;`	Hours worked
`double payRate;`	Hourly pay rate
`double grossPay;`	Gross pay

Creating a Structure

- To use a structure, you must first declare it using the following format:

```
struct tag
{
   variable declaration;
};
```

- The `tag` becomes the name of the new data type.
- The variables declared inside the braces are called **members**.
- Here is an example of a `PayRoll` structure declaration:

```
struct PayRoll
{
   int empNumber;
   string name;
   double hours;
   double payRate;
   double grossPay;
};
```

Warning!

Notice a semicolon is required after the closing brace of the structure declaration.

Note:

In this text, we begin the names of structure tags with an uppercase letter. This visually differentiates these names from the names of variables.

Note:

The structure declaration shown contains three `double` members, each declared on a separate line. The three could also have been declared on the same line, as

```
struct PayRoll
{
   int empNumber;
   string name;
   double hours, payRate, grossPay;
};
```

Many programmers prefer to place each member declaration on a separate line, however, for increased readability.

- A structure declaration creates a new data type but does not define a variable.
- You can define variables (or instances) of the structure type just like any other variable.

```
PayRoll deptHead;
```

- The `deptHead` variable is of type `PayRoll` and contains all the members defined in the structure.
- You can define multiple variables of a structure type. Each is a separate **instance** in memory, containing its own set of members.

```
PayRoll deptHead, foreman, associate;
```

- In summary, using structures involves two steps:
  - **Declare the structure:** This defines its name (tag) and its members.
  - **Define variables:** Create instances of the structure to hold data in your program.

11.3 Accessing Structure Members

Concept:

The *dot operator* (.) allows you to access structure members in a program.

- To access an individual member of a structure variable, use the **dot operator (`.`)**.
- The dot operator connects the structure variable's name with the member's name.

```
deptHead.empNumber = 475;
```

- Member variables can be used just like regular variables for assignments, calculations, and input/output.

```
cout << deptHead.empNumber << endl;
cout << deptHead.name << endl;
```

- **Program 11-1** demonstrates creating a structure, defining a variable of that structure type, and accessing its members to get user input, perform a calculation, and display results.

Program 11-1

```
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

struct PayRoll
{
   int empNumber;
   string name;
   double hours;
   double payRate;
   double grossPay;
};

int main()
{
    PayRoll employee;

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

    cout << "Enter the employee's name: ";
    cin.ignore();
    getline(cin, employee.name);

    cout << "How many hours did the employee work? ";
    cin >> employee.hours;

```

```
29        cout << "What is the employee's hourly payRate? ";
30        cin >> employee.payRate;
31
32        employee.grossPay = employee.hours * employee.payRate;
33
34        cout << "Here is the employee's payroll data:\n";
35        cout << "Name: " << employee.name << endl;
36        cout << "Number: " << employee.empNumber << endl;
37        cout << "Hours worked: " << employee.hours << endl;
38        cout << "Hourly payRate: " << employee.payRate << endl;
39        cout << fixed << showpoint << setprecision(2);
40        cout << "Gross Pay: $" << employee.grossPay << endl;
41        return 0;
42    }
```

Program Output

Note:

Program 11-1 has the following call, in line 26, to `cin`'s `ignore` member function:

```
cin.ignore();
```

Recall that the `ignore` function causes `cin` to ignore the next character in the input buffer. This is necessary for the `getline` function to work properly in the program.

Note:

The contents of a structure variable cannot be displayed by passing the entire variable to `cout`. For example, assuming `employee` is a `PayRoll` structure variable, the following statement will not work:

```
cout << employee << endl;
```

Instead, each member must be separately passed to `cout`

.

- **Program 11-2** further illustrates using a structure member in a mathematical operation by passing it to the `pow` function.

Program 11-2

```
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;

const double PI = 3.14159;

struct Circle
{
    double radius;
    double diameter;
    double area;
};

int main()
{
    Circle c;

    cout << "Enter the diameter of a circle: ";
    cin >> c.diameter;

    c.radius = c.diameter / 2;

    c.area = PI * pow(c.radius, 2.0);

    cout << fixed << showpoint << setprecision(2);
    cout << "The radius and area of the circle are:\n";
    cout << "Radius: " << c.radius << endl;
    cout << "Area: " << c.area << endl;
    return 0;
}
```

💻 **Program Output**

Comparing Structure Variables

- You cannot directly compare two structure variables using relational operators like `==`.

```
if (circle1 == circle2)
```

- To compare two structures, you must compare their members individually.

```
if (circle1.radius == circle2.radius &&
    circle1.diameter == circle2.diameter &&
    circle1.area == circle2.area)
```

11.4 Initializing a Structure

Concept:

The members of a structure variable may be initialized with starting values when the structure variable is defined.

- A structure variable can be initialized with an initialization list when it is defined, similar to an array.

```
struct CityInfo
{
   string cityName;
   string state;
   long population;
   int distance;
};
```

- The values in the list are assigned to the members in the order they are declared.

```
CityInfo location = {"Asheville", "NC", 80000, 28};
```

- You can provide initializers for only the first few members. The remaining members will be left uninitialized.

```
CityInfo location = {"Atlanta", "GA"};
```

- You cannot skip members during initialization. The following code is illegal:

```
CityInfo location = {"Knoxville", "TN", , 90};
```

- **Program 11-3** demonstrates how to work with partially initialized structure variables.

Program 11-3

```
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

struct EmployeePay
{
    string name;
    int empNum;
    double payRate;
    double hours;
    double grossPay;
};

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

    cout << fixed << showpoint << setprecision(2);

    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;

    cout << "Name: " << employee2.name << endl;
    cout << "Employee Number: " << employee2.empNum << endl;
```

```
31        cout << "Enter the hours worked by this employee: ";
32        cin >> employee2.hours;
33        employee2.grossPay = employee2.hours * employee2.payRate;
34        cout << "Gross Pay: " << employee2.grossPay << endl;
35        return 0;
36    }
```

Program Output

- Starting with C++11, the equals sign (`=`) is optional when initializing a structure.

```
CityInfo location {"Asheville", "NC", 80000, 28};
```

Checkpoint

11.1 Write a structure declaration to hold the following data about a savings account:

- Account Number (`string` object)
- Account Balance (`double`)
- Interest Rate (`double`)
- Average Monthly Balance (`double`)

11.2 Write a definition statement for a variable of the structure you declared in Question 11.1. Initialize the members with the following data:

- Account Number: ACZ42137-B12-7
- Account Balance: $4512.59
- Interest Rate: 4%
- Average Monthly Balance: $4217.07

11.3 The following program skeleton, when complete, asks the user to enter these data about his or her favorite movie:

- Name of movie
- Name of the movie's director
- Name of the movie's producer
- The year the movie was released

Complete the following program by declaring the structure that holds this data, defining a structure variable, and writing the individual statements necessary.

```
#include <iostream>
using namespace std;
int main()
{
   cout << "Enter the following data about your\n";
   cout << "favorite movie.\n";
   cout << "name: ";
   cout << "Director: ";
   cout << "Producer: ";
   cout << "Year of release: ";
   cout << "Here is data on your favorite movie:\n";
   return 0;
}
```

11.5 Arrays of Structures

Concept:

Arrays of structures can simplify some programming tasks.

- Instead of using multiple parallel arrays to store related data, you can use a single **array of structures**.
- An array of structures is defined like any other array. For example, using this structure declaration:

```
struct BookInfo
{
   string title;
```

```
    string author;
    string publisher;
    double price;
};
```

- You can define an array of 20 `BookInfo` structures as follows:

```
BookInfo bookList[20];
```

- Each element in the array is a complete structure variable (e.g., `bookList[0]`, `bookList[1]`).
- To access a member of a structure within the array, combine the subscript and the dot operator.

```
bookList[5].title
```

- You can use a loop to iterate through the array and process each structure element.

```
for (int index = 0; index < 20; index++)
{
    cout << bookList[index].title << endl;
    cout << bookList[index].author << endl;
    cout << bookList[index].publisher << endl;
    cout << bookList[index].price << endl << endl;
}
```

- **Program 11-4** uses an array of structures to manage payroll data for multiple employees.

Program 11-4

```
1   #include <iostream>
2   #include <iomanip>
3   using namespace std;
4
5   struct PayInfo
6   {
7       int hours;
8       double payRate;
9   };
10
11  int main()
```

```
12  {
13      const int NUM_WORKERS = 3;
14      PayInfo workers[NUM_WORKERS];
15      int index;
16
17      cout << "Enter the hours worked by " << NUM_WORKERS
18          << " employees and their hourly rates.\n";
19
20      for (index = 0; index < NUM_WORKERS; index++)
21      {
22          cout << "Hours worked by employee #" << (index + 1);
23          cout << ": ";
24          cin >> workers[index].hours;
25
26          cout << "Hourly pay rate for employee #";
27          cout << (index + 1) << ": ";
28          cin >> workers[index].payRate;
29          cout << endl;
30      }
31
32      cout << "Here is the gross pay for each employee:\n";
33      cout << fixed << showpoint << setprecision(2);
34      for (index = 0; index < NUM_WORKERS; index++)
35      {
36          double gross;
37          gross = workers[index].hours * workers[index].payRate;
38          cout << "Employee #" << (index + 1);
39          cout << ": $" << gross << endl;
40      }
41      return 0;
42  }
```

Program Output

Initializing a Structure Array

- You can initialize an array of structures by providing an initialization list for each element.
- Each element's initializers are enclosed in their own set of braces.

```
PayInfo workers[NUM_WORKERS] = {
                                    {10, 9.75 },
                                    {15, 8.62 },
                                    {40, 15.65}
                                  };
```

11.6 Focus on Software Engineering: Nested Structures

Concept:

It's possible for a structure variable to be a member of another structure variable.

- C++ allows you to nest a structure inside another structure.
- This is useful when a logical relationship exists between groups of data.
- Consider these two structure declarations:

```
struct Costs
{
   double wholesale;
   double retail;
};
struct Item
{
   string partNum;
   string description;
   Costs pricing;
};
```

- In the `Item` structure, the `pricing` member is itself a `Costs` structure.
- To access the members of the nested structure, you use two dot operators.

```
Item widget;

widget.pricing.wholesale = 100.0;
widget.pricing.retail = 150.0;
```

- **Program 11-5** provides a detailed example of using nested structures to store employee information, including birth date and address data.

Program 11-5

```
#include <iostream>
#include <string>
using namespace std;

struct Date
{
    int month;
    int day;
    int year;
};

struct Place
{
    string address;
    string city;
    string state;
    string zip;
};

struct EmployeeInfo
{
    string name;
    int employeeNumber;
    Date birthDate;
    Place residence;
};

int main()
{
    EmployeeInfo manager;

    cout << "Enter the manager's name: ";
    getline(cin, manager.name);
```

```
34        cout << "Enter the manager's employee number: ";
35        cin >> manager.employeeNumber;
36
37        cout << "Now enter the manager's date of birth.\n";
38        cout << "Month (up to 2 digits): ";
39        cin >> manager.birthDate.month;
40        cout << "Day (up to 2 digits): ";
41        cin >> manager.birthDate.day;
42        cout << "Year: ";
43        cin >> manager.birthDate.year;
44        cin.ignore();
45
46        cout << "Enter the manager's street address: ";
47        getline(cin, manager.residence.address);
48        cout << "City: ";
49        getline(cin, manager.residence.city);
50        cout << "State: ";
51        getline(cin, manager.residence.state);
52        cout << "ZIP Code: ";
53        getline(cin, manager.residence.zip);
54
55        cout << "\nHere is the manager's information:\n";
56        cout << manager.name << endl;
57        cout << "Employee number " << manager.employeeNumber << endl;
58        cout << "Date of birth: ";
59        cout << manager.birthDate.month << "-";
60        cout << manager.birthDate.day << "-";
61        cout << manager.birthDate.year << endl;
62        cout << "Place of residence:\n";
63        cout << manager.residence.address << endl;
64        cout << manager.residence.city << ", ";
65        cout << manager.residence.state << " ";
66        cout << manager.residence.zip << endl;
67        return 0;
68    }
```

Program Output

Checkpoint

For Questions 11.4–11.7 below, assume the `Product` structure is declared as follows:

```
struct Product
{
    string description;
    int partNum;
    double cost;
};
```

11.4 Write a definition for an array of 100 `Product` structures. Do not initialize the array.

11.5 Write a loop that will step through the entire array you defined in Question 11.4, setting all the product descriptions to an empty string, all part numbers to zero, and all costs to zero.

11.6 Write the statements that will store the following data in the first element of the array you defined in Question 11.4:

- Description: Claw hammer
- Part Number: 547
- Part Cost: $8.29

11.7 Write a loop that will display the contents of the entire array you created in Question 11.4.

11.8 Write a structure declaration named `Measurement`, with the following members:

- `miles`, an integer

- `meters`, a `long` integer

11.9 Write a structure declaration named `Destination`, with the following members:

- `city`, a `string` object
- `distance`, a `Measurement` structure (declared in Question 11.8)

Also define a variable of this structure type.

11.10 Write statements that store the following data in the variable you defined in Question 11.9:

- City: Tupelo
- Miles: 375
- Meters: 603,375

11.7 Structures as Function Arguments

Concept:

Structure variables may be passed as arguments to functions.

Passing a Structure to a Function

- You can pass individual members of a structure to a function, just like regular variables.

```
struct Rectangle
{
   double length;
   double width;
   double area;
};
```

- For example, if `box` is a `Rectangle` variable, you can pass its members to a function:

```
box.area = multiply(box.length, box.width);
```

- It is often more convenient to pass the entire structure variable to a function.

```
void showRect(Rectangle r)
{
   cout << r.length << endl;
   cout << r.width << endl;
   cout << r.area << endl;
}
```

- To call this function, you pass the structure variable:

```
showRect(box);
```

- By default, structures are **passed by value**. This means the function receives a copy of the structure, and any changes made inside the function do not affect the original variable.
- To allow a function to modify the original structure, you can **pass it by reference** by using a reference variable as the parameter.
- **Program 11-6** demonstrates passing a structure by reference to the `getItem` function (to modify it) and by value to the `showItem` function (to display it).

Program 11-6

```
 1  #include <iostream>
 2  #include <string>
 3  #include <iomanip>
 4  using namespace std;
 5
 6  struct InventoryItem
 7  {
 8      int partNum;
 9      string description;
10      int onHand;
11      double price;
12  };
13
14  void getItem(InventoryItem&);
15  void showItem(InventoryItem);
16
17  int main()
18  {
19      InventoryItem part;
```

```
20
21      getItem(part);
22      showItem(part);
23      return 0;
24  }
25
26
27  void getItem(InventoryItem &p)
28  {
29      cout << "Enter the part number: ";
30      cin >> p.partNum;
31
32      cout << "Enter the part description: ";
33      cin.ignore();
34      getline(cin, p.description);
35
36      cout << "Enter the quantity on hand: ";
37      cin >> p.onHand;
38
39      cout << "Enter the unit price: ";
40      cin >> p.price;
41  }
42
43
44  void showItem(InventoryItem p)
45  {
46      cout << fixed << showpoint << setprecision(2);
47      cout << "Part Number: " << p.partNum << endl;
48      cout << "Description: " << p.description << endl;
49      cout << "Units On Hand: " << p.onHand << endl;
50      cout << "Price: $" << p.price << endl;
51  }
```

Program Output

- A structure must be declared before it is used in a function prototype or definition.

Constant Reference Parameters

- Passing large structures by value can be inefficient because the system has to create a complete copy of the structure.
- Passing by reference is more efficient because only an address is passed, avoiding the copy operation.
- If a function does not need to change the original structure, it's best to pass it as a **constant reference**. This provides the efficiency of passing by reference while preventing the function from modifying the original argument.

```
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;
}
```

- The function prototype would also need to be updated:

```
void showItem(const InventoryItem&);
```

11.8 Returning a Structure from a Function

Concept:

A function may return a structure.

- A function can be designed to return an entire structure, just as it can return an `int` or a `double`.
- To do this, specify the structure type as the function's return type.

```
Circle getCircleData()
{
   Circle temp;

   temp.radius = 10.0;
   temp.diameter = 20.0;
   temp.area = 314.159;
```

```
    return temp;
}
```

- The returned structure can then be assigned to a variable of the same structure type.

```
myCircle = getCircleData();
```

- A local structure variable inside the function is used to hold the member values before it is returned.
- **Program 11-7** modifies a previous program to use a function that returns a structure.

Program 11-7

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

const double PI = 3.14159;

struct Circle
{
    double radius;
    double diameter;
    double area;
};

Circle getInfo();

int main()
{
    Circle c;

    c = getInfo();

    c.area = PI * pow(c.radius, 2.0);

    cout << "The radius and area of the circle are:\n";
    cout << fixed << setprecision(2);
    cout << "Radius: " << c.radius << endl;
    cout << "Area: " << c.area << endl;
    return 0;
}

```

```
32
33  Circle getInfo()
34  {
35      Circle tempCircle;
36
37      cout << "Enter the diameter of a circle: ";
38      cin >> tempCircle.diameter;
39      tempCircle.radius = tempCircle.diameter / 2.0;
40
41      return tempCircle;
42  }
```

Program Output

Note:

In Chapter 6, you learned that C++ only allows you to return a single value from a function. Structures, however, provide a way around this limitation. Even though a structure may have several members, a structure variable is a single value. By packaging multiple values inside a structure, you can return as many variables as you need from a function.

11.9 Pointers to Structures

Concept:

You may take the address of a structure variable and create variables that are pointers to structures.

- You can define a pointer to a structure just like any other pointer variable.

```
Circle *cirPtr = nullptr;
```

- You can assign the address of a structure variable to the pointer.

```
Circle myCircle = { 10.0, 20.0, 314.159 };
Circle *cirPtr = nullptr;
```

```
cirPtr = &myCircle;
```

- Using the indirection operator (`*`) to access members through a pointer can be awkward due to operator precedence. The dot operator (`.`) has higher precedence, so parentheses are required.

```
(*cirPtr).radius = 10;
```

- C++ provides the **structure pointer operator (`->`)** as a more convenient way to dereference a structure pointer and access a member.

```
cirPtr->radius = 10;
```

> **Note:**
> The structure pointer operator is supposed to look like an arrow, thus visually indicating that a "pointer" is being used.

- **Program 11-8** demonstrates passing a pointer to a structure as a function parameter, allowing the function to modify the original argument.

Program 11-8

```
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

struct Student
{
    string name;
    int idNum;
    int creditHours;
    double gpa;
};

void getData(Student *);

int main()
{
    Student freshman;

    cout << "Enter the following student data:\n";
```

```
21        getData(&freshman);
22        cout << "\nHere is the student data you entered:\n";
23
24        cout << setprecision(3);
25        cout << "Name: " << freshman.name << endl;
26        cout << "ID Number: " << freshman.idNum << endl;
27        cout << "Credit Hours: " << freshman.creditHours << endl;
28        cout << "GPA: " << freshman.gpa << endl;
29        return 0;
30    }
31
32
33    void getData(Student *s)
34    {
35        cout << "Student name: ";
36        getline(cin, s->name);
37
38        cout << "Student ID Number: ";
39        cin >> s->idNum;
40
41        cout << "Credit Hours Enrolled: ";
42        cin >> s->creditHours;
43
44        cout << "Current GPA: ";
45        cin >> s->gpa;
46    }
```

Program Output

Dynamically Allocating a Structure

- You can use the `new` operator with a structure pointer to dynamically allocate a structure in memory.

```
Circle *cirPtr = nullptr;
cirPtr = new Circle;
cirPtr->radius = 10;
cirPtr->diameter = 20;
cirPtr->area = 314.159;
```

- You can also dynamically allocate an array of structures.

```
Circle *circles = nullptr;
circles = new Circle[5];
for (int count = 0; count < 5; count++)
{
   cout << "Enter the radius for circle "
        << (count + 1) << ": ";
   cin >> circles[count].radius;
}
```

11.10 Focus on Software Engineering: When to Use ., When to Use ->, and When to Use *

- Differentiating between operators is crucial, especially when a structure contains a pointer as a member.
- The structure pointer operator (`->`) is used to dereference a **pointer to a structure**.
- The indirection operator (`*`) is used to dereference a **pointer that is a member** of a structure.
- For example, if `stPtr` is a pointer to a `GradeInfo` structure, and `testScores` is a pointer member within that structure:

```
cout << *stPtr->testScores;
```

- In this statement:
  - `stPtr->testScores` uses the `->` operator to access the `testScores` member of the structure that `stPtr` points to.
  - The `*` operator then dereferences the `testScores` pointer to get the value it points to.

Table 11-3 Structure Pointer Expressions

EXPRESSION	DESCRIPTION

EXPRESSION	DESCRIPTION
`s->m`	`s` is a structure pointer and `m` is a member. This expression accesses the `m` member of the structure pointed to by `s`.
`*a.p`	`a` is a structure variable and `p`, a pointer, is a member. This expression dereferences the value pointed to by `p`.
`(*s).m`	`s` is a structure pointer and `m` is a member. The `*` operator dereferences `s`, causing the expression to access the `m` member of the structure pointed to by `s`. This expression is the same as `s->m`.
`*s->p`	`s` is a structure pointer and `p`, a pointer, is a member of the structure pointed to by `s`. This expression accesses the value pointed to by `p`. (The `->` operator dereferences `s` and the `*` operator dereferences `p`.)
`*(*s).p`	`s` is a structure pointer and `p`, a pointer, is a member of the structure pointed to by `s`. This expression accesses the value pointed to by `p`. `(*s)` dereferences `s` and the outermost `*` operator dereferences `p`. The expression `*s->p` is equivalent.

Checkpoint

Assume the following structure declaration exists for Questions 11.11–11.15:

```
struct Rectangle
{
   int length;
   int width;
};
```

11.11 Write a function that accepts a `Rectangle` structure as its argument and displays the structure's contents on the screen.

11.12 Write a function that uses a `Rectangle` structure reference variable as its parameter and stores the user's input in the structure's members.

11.13 Write a function that returns a `Rectangle` structure. The function should store the user's input in the members of the structure before returning it.

11.14 Write the definition of a pointer to a `Rectangle` structure.

11.15 Assume `rptr` is a pointer to a `Rectangle` structure. Which of the expressions, A, B, or C, is equivalent to the following expression:

```
   rptr->width
```

`*rptr.width`

`(*rptr).width`

`rptr.(*width)`

11.11 Enumerated Data Types

Concept:

An enumerated data type is a programmer-defined data type. It consists of values known as enumerators, which represent integer constants.

- An **enumerated data type** (`enum`) is a custom data type that you create by specifying a set of values, known as **enumerators**.
- Enumerators are identifiers that represent integer constants.

```
enum Day { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY };
```

- This creates a new type named `Day`. You can then define variables of this type:

```
Day workDay = WEDNESDAY;
```

- Internally, the compiler assigns integer values to the enumerators, starting from 0. So, `MONDAY` is 0, `TUESDAY` is 1, `WEDNESDAY` is 2, and so on.
- You cannot directly assign an integer to an `enum` variable without an explicit cast.

```
workDay = 3;
workDay = static_cast<Day>(3);
```

- You can, however, assign an enumerator to an `int` variable.

```
int x = THURSDAY;
```

- Enumerator values can be compared using relational operators since they are integers internally.
- **Program 11-9** demonstrates using enumerators to make a `for` loop more readable.

Program 11-9

```
#include <iostream>
#include <iomanip>
using namespace std;

enum Day { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY };

int main()
{
    const int NUM_DAYS = 5;
    double sales[NUM_DAYS];
    double total = 0.0;
    int index;

    for (index = MONDAY; index <= FRIDAY; index++)
    {
        cout << "Enter the sales for day "
             << index << ": ";
        cin >> sales[index];
    }

    for (index = MONDAY; index <= FRIDAY; index++)
       total += sales[index];

    cout << "The total sales are $" << setprecision(2)
         << fixed << total << endl;

    return 0;
}
```

💻 **Program Output**

- You cannot use increment (`++`) or decrement (`--`) operators directly on `enum` variables. You must use a cast.

```
for (workDay = MONDAY; workDay <= FRIDAY; workDay++)
for (workDay = MONDAY; workDay <= FRIDAY; workDay = static_cast<Day>(workDay + 1))
```

- To display a string (like "Monday") instead of the integer value (0), you must write code, such as a `switch` statement, to map the enumerator to the string.
- You can specify your own integer values for enumerators. Any enumerator without an assigned value will be assigned one more than the previous enumerator.

```
enum Water { FREEZING = 32, BOILING = 212 };
enum Colors { RED, ORANGE, YELLOW = 9, GREEN, BLUE };
```

Using Strongly Typed `enum`s in C++ 11

- Standard `enum`s can cause naming conflicts because all enumerators share the same scope.
- C++11 introduced the **strongly typed enum** (or `enum class`) to solve this problem. It makes enumerators local to the `enum`.

```
enum class Presidents { MCKINLEY, ROOSEVELT, TAFT };
enum class VicePresidents { ROOSEVELT, FAIRBANKS, SHERMAN };
```

- To use a strongly typed enumerator, you must prefix it with the `enum` name and the scope resolution operator (`::`).

```
Presidents prez = Presidents::ROOSEVELT;
if (prez == Presidents::ROOSEVELT) { /* ... */ }
```

- To get the underlying integer value of a strongly typed enumerator, you must use a cast.

```
int x = static_cast<int>(Presidents::ROOSEVELT);
```

- You can also specify the underlying integer type for a strongly typed `enum` (e.g., `char`, `unsigned int`).

```
enum class Day : char { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY };
```

Checkpoint

11.16 Look at the following declaration:

```
enum Flower { ROSE, DAISY, PETUNIA };
```

In memory, what value will be stored for the enumerator `ROSE`? For `DAISY`? For `PETUNIA`?

11.17 What will the following code display?

```
enum { HOBBIT, ELF = 7, DRAGON };
cout << HOBBIT << " " << ELF << " " << DRAGON << endl;
```

11.18 Does the enumerated data type declared in Checkpoint Question 11.17 have a name, or is it anonymous?

11.19 What will the following code display?

```
enum Letters { Z, Y, X };
if (Z > X)
   cout << "Z is greater than X. \n";
else
   cout << "Z is not greater than X. \n";
```

11.20 Will the following code cause an error, or will it compile without any errors? If it causes an error, rewrite it so it compiles.

```
enum Color { RED, GREEN, BLUE };
Color c;
c = 0;
```

11.21 Will the following code cause an error, or will it compile without any errors? If it causes an error, rewrite it so it compiles.

```
enum Color { RED, GREEN, BLUE };
Color c = RED;
c++;
```

Note:

For an additional example of this chapter's topics, see the High Adventure Travel Part 2 Case Study on the Computer Science Portal at pearsonhighered.com/gaddis.

Review Questions and Exercises

Short Answer

1. What is a primitive data type?
2. Does a structure declaration cause a structure variable to be created?
3. Both arrays and structures are capable of storing multiple values. What is the difference between an array and a structure?
4. Look at the following structure declaration:

```
struct Point
{
   int x;
   int y;
};
```

   Write statements that

   1. define a `Point` structure variable named `center`.
   2. assign 12 to the `x` member of `center`.
   3. assign 7 to the `y` member of `center`.
   4. display the contents of the `x` and `y` members of `center`.

5. Look at the following structure declaration:

```
struct FullName
```

`{`

`string lastName;`

`string middleName;`

`string firstName;`

`};`

Write statements that

1. define a `FullName` structure variable named `info`.
2. assign your last, middle, and first name to the members of the `info` variable.
3. display the contents of the members of the `info` variable.

6. Look at the following code:

```
struct PartData
{
   string partName;
   int idNumber;
};
PartData inventory[100];
```

Write a statement that displays the contents of the `partName` member of element 49 of the `inventory` array.

7. Look at the following code:

```
struct Town
{
   string townName;
   string countyName;
   double population;
   double elevation;
};
Town t = { "Canton", "Haywood", 9478 };
```

1. What value is stored in `t.townName`?
2. What value is stored in `t.countyName`?

3. What value is stored in `t.population`?
4. What value is stored in `t.elevation`?

8. Look at the following code:

```
structure Rectangle
{
   int length;
   int width;
};
Rectangle *r = nullptr
```

Write statements that

1. dynamically allocate a `Rectangle` structure variable and use `r` to point to it.
2. assign 10 to the structure's `length` member and 14 to the structure's `width` member.

9. What will the following code display?

```
enum { POODLE, BOXER, TERRIER };
cout << POODLE << " " << BOXER << " " << TERRIER << endl;
```

10. Look at the following declaration:

```
enum Person { BILL, JOHN, CLAIRE, BOB };
Person p;
```

Indicate whether each of the following statements or expressions is valid or invalid.

1. `p = BOB;`
2. `p++;`
3. `BILL > BOB`
4. `p = 0;`
5. `int x = BILL;`
6. `p = static_cast<Person>(3);`

7. `cout << CLAIRE << endl;`

Fill-in-the-Blank

1. Before a structure variable can be created, the structure must be _____.
2. The _____ is the name of the structure type.
3. The variables declared inside a structure declaration are called _____.
4. A(n) _____ is required after the closing brace of a structure declaration.
5. In the definition of a structure variable, the _____ is placed before the variable name, just like the data type of a regular variable is placed before its name.
6. The _____ operator allows you to access structure members.

Algorithm Workbench

1. The structure `Car` is declared as follows:

```
struct Car
{
   string carMake;
   string carModel;
   int yearModel;
   double cost;
};
```

   Write a definition statement that defines a `Car` structure variable initialized with the following data:

   - Make: Ford
   - Model: Mustang
   - Year Model: 1968
   - Cost: $20,000

2. Define an array of 25 of the `Car` structure variables (the structure is declared in Question 17).

3. Define an array of 35 of the `Car` structure variables. Initialize the first three elements with the following data:

MAKE	MODEL	YEAR	COST
Ford	Taurus	1997	$21,000
Honda	Accord	1992	$11,000
Lamborghini	Countach	1997	$200,000

4. Write a loop that will step through the array you defined in Question 19, displaying the contents of each element.

5. Declare a structure named `TempScale`, with the following members:

```
fahrenheit: a double
centigrade: a double
```

   Next, declare a structure named `Reading`, with the following members:

```
windSpeed: an int
humidity: a double
temperature: a TempScale structure variable
```

   Next, define a `Reading` structure variable.

6. Write statements that will store the following data in the variable you defined in Problem 21:

   Wind Speed: 37 mph

   Humidity: 32%

   Fahrenheit temperature: 32 degrees

   Centigrade temperature: 0 degrees

7. Write a function called `showReading`. It should accept a `Reading` structure variable (see Problem 21) as its argument. The function should display the contents of the variable on the screen.

8. Write a function called `findReading`. It should use a `Reading` structure reference variable (see Problem 21) as its parameter. The function should ask the user to enter values for each member of the structure.

9. Write a function called `getReading`, which returns a `Reading` structure (see Problem 21). The function should ask the user to enter values for each member of a `Reading` structure, then return the structure.

10. Write a function called `recordReading`. It should use a `Reading` structure pointer variable (see Problem 21) as its parameter. The function should ask the user to enter values for each member of the structure pointed to by the parameter.

11. Rewrite the following statement using the structure pointer operator:

    ```
    (*rptr).windSpeed = 50;
    ```

12. Look at the following statement:

    ```
    enum Color { RED, ORANGE, GREEN, BLUE };
    ```

    1. What is the name of the data type declared by this statement?
    2. What are the enumerators for this type?
    3. Write a statement that defines a variable of this type and initializes it with a valid value.

13. A pet store sells dogs, cats, birds, and hamsters. Write a declaration for an anonymous enumerated data type that can represent the types of pets the store sells.

True or False

1. T F A semicolon is required after the closing brace of a structure declaration.
2. T F A structure declaration does not define a variable.
3. T F The contents of a structure variable can be displayed by passing the structure variable to the `cout` object.
4. T F Structure variables may not be initialized.
5. T F In a structure variable's initialization list, you do not have to provide initializers for all the members.

6. T F You may skip members in a structure's initialization list.
7. T F The following expression refers to element 5 in the array `carInfo`: `carInfo.model[5]`
8. T F An array of structures may be initialized.
9. T F A structure variable may not be a member of another structure.
10. T F A structure member variable may be passed to a function as an argument.
11. T F An entire structure may not be passed to a function as an argument.
12. T F A function may return a structure.
13. T F When a function returns a structure, it is always necessary for the function to have a local structure variable to hold the member values that are to be returned.
14. T F The indirection operator has higher precedence than the dot operator.
15. T F The structure pointer operator does not automatically dereference the structure pointer on its left.

Find the Errors

Each of the following declarations, programs, and program segments has errors. Locate as many as you can.

1.
```
struct
{
   int x;
   float y;
};
```

2.
```
struct Values
{
   string name;
   int age;
}
```

3.
```
struct TwoVals
{
   int a, b;
};
```

```
int main ()
{
   TwoVals.a = 10;
   TwoVals.b = 20;
   return 0;
}
```

4.
```
#include <iostream>
using namespace std;
struct ThreeVals
{
   int a, b, c;
};
int main()
{
   ThreeVals vals = {1, 2, 3};
   cout << vals << endl;
   return 0;
}
```

5.
```
#include <iostream>
#include <string>
using namespace std;
struct names
{
   string first;
   string last;
};
int main ()
{
   names customer = "Smith", "Orley";
   cout << names.first << endl;
   cout << names.last << endl;
   return 0;
}
```

6.
```
struct FourVals
{
   int a, b, c, d;
};
int main ()
{
   FourVals nums = {1, 2, , 4};
```

```
   return 0;
}
```

7.
```
struct TwoVals
{
   int a;
   int b;
};
TwoVals getVals()
{
   TwoVals.a = TwoVals.b = 0;
}
```

8.
```
struct ThreeVals
{
   int a, b, c;
};
int main ()
{
   TwoVals s, *sptr = nullptr;
   sptr = &s;
   *sptr.a = 1;
   return 0;
}
```

Programming Challenges

1. Movie Data

   Write a program that uses a structure named `MovieData` to store the following information about a movie:

   - Title
   - Director
   - Year Released
   - Running Time (in minutes)

   The program should create two `MovieData` variables, store values in their members, and pass each one, in turn, to a function that displays the information about the movie in a

clearly formatted manner.

2. Movie Profit

   Modify the program written for Programming Challenge 1 (Movie Data) to include two additional members that hold the movie's production costs and first-year revenues. Modify the function that displays the movie data to display the title, director, release year, running time, and first year's profit or loss.

3. Corporate Sales Data

   Write a program that uses a structure to store the following data on a company division:

   - Division Name (such as East, West, North, or South)
   - First-Quarter Sales
   - Second-Quarter Sales
   - Third-Quarter Sales
   - Fourth-Quarter Sales
   - Total Annual Sales
   - Average Quarterly Sales

   The program should use four variables of this structure. Each variable should represent one of the following corporate divisions: East, West, North, and South. The user should be asked for the four quarters' sales figures for each division. Each division's total and average sales should be calculated and stored in the appropriate member of each structure variable. These figures should then be displayed on the screen.

   Input Validation: Do not accept negative numbers for any sales figures.

4. Weather Statistics

   

   Solving the Weather Statistics Problem

   Write a program that uses a structure to store the following weather data for a particular month:

- Total Rainfall
- High Temperature
- Low Temperature
- Average Temperature

The program should have an array of 12 structures to hold weather data for an entire year. When the program runs, it should ask the user to enter data for each month. (The average temperature should be calculated.) Once the data are entered for all the months, the program should calculate and display the average monthly rainfall, the total rainfall for the year, the highest and lowest temperatures for the year (and the months they occurred in), and the average of all the monthly average temperatures.

Input Validation: Only accept temperatures within the range between –100 and +140 degrees Fahrenheit.

5. Weather Statistics Modification

   Modify the program that you wrote for Programming Challenge 4 (weather statistics) so it defines an enumerated data type with enumerators for the months (`JANUARY`, `FEBRUARY`, so on). The program should use the enumerated type to step through the elements of the array.

6. Soccer Scores

   Write a program that stores the following data about a soccer player in a structure:

   - Player's Name
   - Player's Number
   - Points Scored by Player

   The program should keep an array of 12 of these structures. Each element is for a different player on a team. When the program runs, it should ask the user to enter the data for each player. It should then show a table that lists each player's number, name, and points scored. The program should also calculate and display the total points earned by the team. The number and name of the player who has earned the most points should also be displayed.

   Input Validation: Do not accept negative values for players' numbers or points scored.

7. Customer Accounts

   Write a program that uses a structure to store the following data about a customer account:

   - Name
   - Address
   - City, State, and ZIP
   - Telephone Number
   - Account Balance
   - Date of Last Payment

   The program should use an array of at least 10 structures. It should let the user enter data into the array, change the contents of any element, and display all the data stored in the array. The program should have a menu-driven user interface.

   Input Validation: When the data for a new account is entered, be sure the user enters data for all the fields. No negative account balances should be entered.

8. Search Function for Customer Accounts Program

   Add a function to Programming Challenge 7 (Customer Accounts) that allows the user to search the structure array for a particular customer's account. It should accept part of the customer's name as an argument then search for an account with a name that matches it. All accounts that match should be displayed. If no account matches, a message saying so should be displayed.

9. Speakers' Bureau

   Write a program that keeps track of a speakers' bureau. The program should use a structure to store the following data about a speaker:

   - Name
   - Telephone Number
   - Speaking Topic
   - Fee Required

The program should use an array of at least 10 structures. It should let the user enter data into the array, change the contents of any element, and display all the data stored in the array. The program should have a menu-driven user interface.

Input Validation: When the data for a new speaker is entered, be sure the user enters data for all the fields. No negative amounts should be entered for a speaker's fee.

10. Search Function for the Speakers' Bureau Program

    Add a function to Programming Challenge 9 (Speakers' Bureau) that allows the user to search for a speaker on a particular topic. It should accept a key word as an argument then search the array for a structure with that key word in the Speaking Topic field. All structures that match should be displayed. If no structure matches, a message saying so should be displayed.

11. Monthly Budget

    A student has established the following monthly budget:

Housing	$500.00
Utilities	$150.00
Household Expenses	$65.00
Transportation	$50.00
Food	$250.00
Medical	$30.00
Insurance	$100.00
Entertainment	$150.00
Clothing	$75.00
Miscellaneous	$50.00

    Write a program that has a `MonthlyBudget` structure designed to hold each of these expense categories. The program should pass the structure to a function that asks the user to enter the amounts spent in each budget category during a month. The program should then pass the structure to a function that displays a report indicating the amount

over or under in each category, as well as the amount over or under for the entire monthly budget.

12. Course Grade

    Write a program that uses a structure to store the following data:

MEMBER NAME	DESCRIPTION
Name	Student name
Idnum	Student ID number
Tests	Pointer to an array of test scores
Average	Average test score
Grade	Course grade

The program should keep a list of test scores for a group of students. It should ask the user how many test scores there are to be and how many students there are. It should then dynamically allocate an array of structures. Each structure's `Tests` member should point to a dynamically allocated array that will hold the test scores.

After the arrays have been dynamically allocated, the program should ask for the ID number and all the test scores for each student. The average test score should be calculated and stored in the `average` member of each structure. The course grade should be computed on the basis of the following grading scale:

AVERAGE TEST GRADE	COURSE GRADE
91–100	A
81–90	B
71–80	C
61–70	D
60 or below	F

The course grade should then be stored in the `Grade` member of each structure. Once all this data is calculated, a table should be displayed on the screen listing each student's name, ID number, average test score, and course grade.

Input Validation: Be sure all the data for each student is entered. Do not accept negative numbers for any test score.

13. Drink Machine Simulator

    Write a program that simulates a soft drink machine. The program should use a structure that stores the following data:

    - Drink Name
    - Drink Cost
    - Number of Drinks in Machine

    The program should create an array of five structures. The elements should be initialized with the following data:

DRINK NAME	COST	NUMBER IN MACHINE
Cola	.75	20
Root Beer	.75	20
Lemon-Lime	.75	20
Grape Soda	.80	20
Cream Soda	.80	20

    Each time the program runs, it should enter a loop that performs the following steps: A list of drinks is displayed on the screen. The user should be allowed to either quit the program or pick a drink. If the user selects a drink, he or she will next enter the amount of money that is to be inserted into the drink machine. The program should display the amount of change that would be returned, and subtract one from the number of that drink left in the machine. If the user selects a drink that has sold out, a message should be displayed. The loop then repeats. When the user chooses to quit the program, it should display the total amount of money the machine earned.

    Input Validation: When the user enters an amount of money, do not accept negative values or values greater than $1.00.

14. Inventory Bins

Write a program that simulates inventory bins in a warehouse. Each bin holds a number of the same type of parts. The program should use a structure that keeps the following data:

- Description of the part kept in the bin
- Number of parts in the bin

The program should have an array of 10 bins, initialized with the following data:

PART DESCRIPTION	NUMBER OF PARTS IN THE BIN
Valve	10
Bearing	5
Bushing	15
Coupling	21
Flange	7
Gear	5
Gear Housing	5
Vacuum Gripper	25
Cable	18
Rod	12

The program should have the following functions:

AddParts—increases a specific bin's part count by a specified number.
RemoveParts—decreases a specific bin's part count by a specified number.

When the program runs, it should repeat a loop that performs the following steps: The user should see a list of what each bin holds and how many parts are in each bin. The user can choose to either quit the program or select a bin. When a bin is selected, the user can either add parts to it or remove parts from it. The loop then repeats, showing the updated bin data on the screen.

Input Validation: No bin can hold more than 30 parts, so don't let the user add more than a bin can hold. Also, don't accept negative values for the number of parts being added or removed.