Chapter 11

Structured Data

Abstract Data Types (ADT)

- Data types created by the programmer
- ADTs have their own range (or domain) of data and their own set of operations that can be performed on them
What is abstraction?

- An abstraction is a general model of something

Data Types

- C++ has several primitive data types:

<table>
<thead>
<tr>
<th>bool</th>
<th>int</th>
<th>unsigned long int</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>long int</td>
<td>double</td>
</tr>
<tr>
<td>unsigned char</td>
<td>unsigned short int</td>
<td>double</td>
</tr>
<tr>
<td>short int</td>
<td>unsigned int</td>
<td>long double</td>
</tr>
</tbody>
</table>

- What is defined by a data type:
  - Values the variable can hold
  - The operations that can be applied to those values
Abstract Data Types

- A data type created by the programmer:
  - The programmer decides what values are acceptable for the data type
  - The programmer decides what operations can be performed on the data type

Combining Data into Structures

- C++ allows you to group several variables together into a single item known as a structure
- C calls this a struct
Payroll System

<table>
<thead>
<tr>
<th>Variable Declaration</th>
<th>Information Held</th>
</tr>
</thead>
<tbody>
<tr>
<td>int empNumber;</td>
<td>Employee number</td>
</tr>
<tr>
<td>char name [SIZE];</td>
<td>Employee name</td>
</tr>
<tr>
<td>double hours;</td>
<td>Hours worked</td>
</tr>
<tr>
<td>double payRate;</td>
<td>Hourly pay rate</td>
</tr>
<tr>
<td>double grossPay;</td>
<td>Gross pay</td>
</tr>
</tbody>
</table>

Payroll as a C++ struct

```cpp
struct PayRoll
{
    int empNumber;
    char name[SIZE];
    double hours;
    double payRate;
    double grossPay;
}; // PayRoll
```
**Declaration**

Payroll deptHead;

Members

Structure Variable Name

depthHead

- empNumber
- name
- hours
- payRate
- grossPay

**Another Declaration**

PayRoll deptHead, foreman, associate

deptHead

- empNumber
- name
- hours
- payRate
- grossPay

foreman

- empNumber
- name
- hours
- payRate
- grossPay

associate

- empNumber
- name
- hours
- payRate
- grossPay
How Do You Create a Structure

- Create the structure declaration
  - This establishes the tag (or name) of the structure and a list of items that are members
- Declare variables (or instances) of the structure and use them in the program to hold data

Declaration Notes

- Must have a semicolon after the closing right brace

```c
struct PayRoll
{
    int empNumber;
    char name[25];
    double hours;
    double payRate;
    double grossPay;
}; // PayRoll
```

Notice the ;
How Do I Access Structure Members?

- The dot operator (.) allows access to structure members in a program
- For example:
  ```cpp
  PayRoll employee;
  ...
  cin >> employee.hours;
  cout << employee.payRate;
  ```
- Member variables can be used in any manner appropriate for their data type

---

Using Structures – Program

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

const int SIZE = 25;

struct PayRoll
{
  int empNumber; // Employee number
  char name[SIZE]; // Employee's name
  double hours; // Hours worked
  double payRate; // Hourly Payrate
  double grossPay; // Gross Pay
}; // PayRoll
```
int main ( void )
{
    PayRoll employee;

    cout << "Enter the employee's number: ";
    cin >> employee.empNumber;
    cout << "Enter the employee's name: ";
    cin.ignore(); // Skip the remaining 'n' character
    cin.getline(employee.name, 25);
    cout << "How many hours did the employee work? ";
    cin >> employee.hours;
    cout << "What is the employee's hourly payrate? ";
    cin >> employee.payRate;
    employee.grossPay = employee.hours * employee.payRate;
    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;
} // main
Using Structures – Output

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 payrate? 20 [Enter]
Here is the employee's payroll data:
Name: Jill Smith
Number: 489
Hours worked: 40
Hourly Payrate: 20
Gross Pay: $800.00

Displaying a Structure

- 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;
  ```
Comparing **struct** Variables

- **You cannot** compare struct variables directly:
  ```
  if ( employee1 == employee2 ) // won't work
  ```

- **Instead, you must** compare on a field basis:
  ```
  if ( employee1.grossPay < employee2.grossPay ) ...
  ```

---

**Circle Structure – Program**

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

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

const double PI = 3.14159;
```
int main ( void )
{
    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 << "The radius and area of the circle are: \n";
    cout << fixed << showpoint << setprecision(2);
    cout << "Radius: " << c.radius << endl;
    cout << "Area: " << c.area << endl;
    return 0;
} // main

Circle Structure – Program (cont)

Circle Structure – Output

Enter the diameter of a circle: 10 [Enter]
The radius and area of the circle are:
Radius: 5
Area: 78.54
C-Strings as Structure Members

- When a character array is a structure member, use the same sting manipulation techniques with it as you would with any other character array.

C-String in a Structure – Program

```c
#include <iostream>
#include <cstring>
using namespace std;

const int LENGTH = 15;
const int FULLLENGTH = 45;

struct Name
{
    char first[LENGTH];
    char middle[LENGTH];
    char last[LENGTH];
    char full[FULLLENGTH];
}; // Name
```
C-String in a Structure – Program (cont)

```c
int main ( void )
{
    Name person;
    cout << "Enter your first name: ";
    cin >> person.first;
    cout << "Enter your middle name: ";
    cin >> person.middle;
    cout << "Enter your last name: ";
    cin >> person.last;
    strcpy(person.full, person.first);
    strcat(person.full, " ");
    strcat(person.full, person.middle);
    strcat(person.full, person.last);
    cout << "Your full name is " << person.full << endl;
    return 0;
} // main
```

C-String in a Structure – Output

Enter your first name: **Josephine** [Enter]
Enter your middle name: **Yvonne** [Enter]
Enter your last name: **Smith** [Enter]

Your full name is Josephine Yvonne Smith
Initializing a Structure

- The members of a structure variable may be initialized with starting values when the structure variable is declared.
- Consider the following example:

```c
struct GeoInfo
{
    char cityName[30];
    char state[3];
    long population;
    int distance;
}; // GeoInfo
GeoInfo location = {"Ashville", "NC", 50000, 28};
```

More on Initializing a Structure

- May initialize only some members:
  GeoInfo city = {"Providence" };
- Cannot skip over members:
  GeoInfo city = {"Logan", "UT", 55000 };
- Cannot initialize the structure declaration, since it does not allocate memory.
Arrays of Structures

- Arrays of structures can simplify some programming tasks

```c
struct BookInfo
{
  char title[50];
  char author[30];
  char publisher[25];
  double price;
}; // BookInfo
BookInfo bookList[20];
```

Using Arrays of Structures – Program

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

struct PayInfo
{
  int hours;       // Hours Worked
  double payRate; // Hourly Pay Rate
}; // PayInfo
```
int main ( void )
{
    const int NUMWORKERS = 5;
    PayInfo workers[NUMWORKERS];
    cout << "Enter the hours worked by " << NUMWORKERS << " employees and their hourly rates.\n";
    for ( int index = 0; index < NUMWORKERS; index++ )
    {
        cout << "Hours worked by employee #" << (index + 1);
        cout << ": ";
        cin >> workers[index].hours;
        cout << "Hourly pay rate for employee #";
        cout << (index + 1) << ": ";
        cin >> workers[index].payRate;
    } // for
    cout << "Here is the gross pay for each employee:\n";
    cout << fixed << showpoint << setprecision(2);
    for ( int index = 0; index < NUMWORKERS; index++ )
    {
        double gross;
        gross = workers[index].hours * workers[index].payRate;
        cout << "Employee #" << (index + 1);
        cout << ": ") << gross << endl;
    } // for
    return 0;
} // main
Using Arrays of Structures – Output

Enter the hours worked by 5 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: 15 [Enter]
Hourly pay rate for employee #2: 8.62 [Enter]
Hours worked by employee #3: 20 [Enter]
Hourly pay rate for employee #3: 10.50 [Enter]
Hours worked by employee #4: 40 [Enter]
Hourly pay rate for employee #4: 18.75 [Enter]
Hours worked by employee #5: 40 [Enter]
Hourly pay rate for employee #5: 15.65 [Enter]

Here is the gross pay for each employee:

Employee #1: $97.50
Employee #2: $129.30
Employee #3: $210.00
Employee #4: $750.00
Employee #5: $626.00

Initializing a Structure Array

                     {15, 8.62},
                     {20, 10.50},
                     {40, 18.75},
                     {40, 15.65}};
Nested Structures

- It’s possible for a structure variable to be a member of another structure variable
- For example, consider the following:
  ```c
  struct Costs {
    double wholesale;
    double retail;
  }; // Costs
  struct Item {
    char partNum[10];
    char description[25];
    Costs pricing;
  }; // Item
  ```

Members of Nested Structures

- Use the dot operator multiple times to refer to fields of nested structures:
  ```c
  Item game;
  game.pricing.retail = 24.95;
  game.pricing.wholesale = 17.54;
  ```
Nested Structures – Program

#include <iostream>
using namespace std;

const int ADDRLENGTH = 50;
const int CITYLENGTH = 20;
const int STATELENGTH = 20;
const int ZIPLENGTH = 11;
const int NAMELENGTH = 50;

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

struct Place
{
    char address[ADDRLENGTH];
    char city[CITYLENGTH];
    char state[STATELENGTH];
    char zip[ZIPLENGTH];
}; // Place

struct EmpInfo
{
    char name[NAMELENGTH];
    int empNumber;
    Date birthDate;
    Place residence;
}; // EmpInfo
void main ( void )
{
    EmpInfo manager;
    cout << "Enter the manager's name: ";
    cin.getline(manager.name, NAMELENGTH);
    cout << "Enter the manager's employee number: ";
    cin >> manager.empNumber;
    cout << "Now enter the manager's date-of-birth\n";
    cout << "Month (up to 2 digits): ";
    cin >> manager.birthDate.month;
    cout << "Day (up to 2 digits): ";
    cin >> manager.birthDate.day;
    cout << "Year (2 digits): ";
    cin >> manager.birthDate.year;
    cout << "Place of residence:\n";
    cout << manager.residence.address << endl;
    cout << manager.residence.city << " ";
    cout << manager.residence.state << "  ";
    cout << manager.residence.zip << endl;
} // main
Nested Structures – Output

Enter the manager's name: John Smith [Enter]
Enter the manager's employee number: 789 [Enter]
Now enter the manager's date-of-birth
Month (up to 2 digits): 10 [Enter]
Day (up to 2 digits): 14 [Enter]
Year (2 digits): 65 [Enter]
Enter the manager's street address: 190 Disk Drive [Enter]
City: Redmond [Enter]
State: WA [Enter]
Zip Code: 98052 [Enter]
Here is the manager's information:
John Smith
Employee number 789
Date of birth: 10-14-65
Place of residence:
190 Disk Drive
Redmond, WA 98052

Structures as Function Arguments

- Structures may be passed as arguments:
  showItem( item );
- May pass an members of a structure variable to a function:
  totalPrice( item.pricing.retail );
- Can use a reference parameter if the function needs to modify the contents of the structure variable
Structures as Arguments – Program

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

const int SIZE = 50;

struct InvItem
{
    int partNum; // Part number
    char description[SIZE]; // Item description
    int onHand; // Units on hand
    double price; // Unit price
}; // InvItem

// Function Prototypes
void GetItem ( InvItem & );
void ShowItem ( InvItem );

int main ( void )
{
    InvItem part;
    GetItem(part);
    ShowItem(part);
    return 0;
} // main

void GetItem ( InvItem &piece )
{
    cout << "Enter the part number: ";
    cin >> piece.partNum;
    cout << "Enter the part description: ";
    cin.get(); // Eat the remaining newline
    cin.getline( piece.description, 50 );
    cout << "Enter the quantity on hand: ";
    cin >> piece.onHand;
    cout << "Enter the unit price: ";
    cin >> piece.price;
} // GetItem
```
structures as arguments – program (cont)

```cpp
void ShowItem ( InvItem piece )
{
    cout << fixed << showpoint << setprecision(2);
    cout << "Part Number: " << piece.partNum << endl;
    cout << "Description: " << piece.description << endl;
    cout << "Units On Hand: " << piece.onHand << endl;
    cout << "Price: $" << piece.price << endl;
} // ShowItem
```

structures as arguments – output

Enter the part number: 800 [Enter]
Enter the part description: Screwdriver [Enter]
Enter the quantity on hand: 135 [Enter]
Enter the unit price: 1.25 [Enter]
Part Number: 800
Description: Screwdriver
Units On Hand: 135
Price: $1.25
Constant Reference Parameters

- Sometimes structures can be quite large
- Therefore, passing by value can decrease a program’s performance because it has to be copied
- But passing by reference can cause problems because function can modify the structure
- Instead, pass by constant reference:
  ```cpp
  void ShowItem ( const InvItem &piece )
  {
    cout << fixed << showpoint << setprecision(2);
    cout << "Part Number: " << piece.partNum << endl;
    cout << "Description: " << piece.description << endl;
    cout << "Price: $" << piece.price << endl;
  } // ShowItem
  ```

Returning a Structure from a Function

- A function can return a struct:
  ```cpp
  Circle getInfo ( void ); // prototype
  c = getInfo(); // call
  ```
- The function must define a local structure
  - For internal use
  - For use with the return statement
#include <iostream>
#include <iomanip>
#include <cmath>  // Math functions

using namespace std;

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

Circle getInfo ( void );

const double PI = 3.14159;

int main ( void )
{
    Circle c;
    c = getInfo();
    c.area = PI * pow(c.radius, 2.0);
    cout << "The radius and area of the circle are:\n";
    cout << fixed << showpoint << setprecision(2);
    cout << "Radius: " << c.radius << endl;
    cout << "Area: " << c.area << endl;
    return 0;
} // main

Circle getInfo ( void )
{
    Circle round;
    cout << "Enter the diameter of a circle: ";
    cin >> round.diameter;
    round.radius = round.diameter / 2;
    return round;
} // getInfo
Pointers to Structures – Initialize Radius

- You may take the address of a structure variable and create variables that are pointers to structures
  - Circle *cirPtr;
    cirPtr = &piePlate;
    *cirPtr.Radius = 10;  // incorrect
    // -. is higher precedence than *
    (*cirPtr).Radius = 10;  // correct
    cirPtr->Radius = 10;  // correct

Pointers and Structures – Program

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

const int NAMESIZE = 25;

struct PayRoll
{
    int empNumber;  // Employee number
    char name[NAMESIZE];  // Employee's name
    double hours;  // Hours worked
    double payRate;  // Hourly Payrate
    double grossPay;  // Gross Pay
};// PayRoll
```
int main ( void )
{
    PayRoll *employee;
    employee = new PayRoll;
    cout << "Enter the employee's number: ";
    cin >> employee->empNumber;
    cout << "Enter the employee's name: ";
    cin.ignore();
    cin.getline(employee->name, NAMESIZE);
    cout << "How many hours did the employee work? ";
    cin >> employee->hours;
    cout << "What is the employee's hourly payrate? ";
    cin >> employee->payRate;
    employee->grossPay = employee->hours * employee->payRate;
    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;
    delete employee;
    return 0;
} // main
## Pointers and Structures – Output

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 payrate? **20** [Enter]
Here is the employee's payroll data:
   - Name: Jill Smith
   - Number: 489
   - Hours worked: 40
   - Hourly Payrate: 20
   - Gross Pay: $800.00

---

## Dynamically Allocating a Structure

circle *cirPtr;
cirPtr = new Circle;
cirPtr->radius = 10;
cirPtr->diameter = 20;
cirPtr->area = 314.159;
The Use of ., ->, and *

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s-&gt;m</td>
<td>s is a structure pointer and m is a member. This expression accesses the m member of the structure pointer to by s.</td>
</tr>
<tr>
<td>*a.p</td>
<td>a is a structure variable and p, a pointer, is a member. This expression dereferences the value pointed to by p.</td>
</tr>
<tr>
<td>(*s).m</td>
<td>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-&gt;m.</td>
</tr>
<tr>
<td>*s-&gt;p</td>
<td>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.</td>
</tr>
<tr>
<td>*(*s).p</td>
<td>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. This expression *s-&gt;p is equivalent.</td>
</tr>
</tbody>
</table>

Enumerate Data Types

- An enumerated data type is a programmer-defined data type
- It consists of value known as enumerators, that represent integer constants
Enumerate Data Type

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

- The identifiers MONDAY, TUESDAY, WEDNESDAY, THURSDAY, and FRIDAY, that are listed inside the braces, are enumerators
  - They represent the values that belong to the Day data type

- Note that the enumerators are not strings, so they aren’t enclosed in quotes
  - They are identifiers

Enumerated Data Types

- Once you have created an enumerated data type in your program, you can define variables of that type

- Example
  
  ```
  Day workDay;
  ```

- This statement defines workDay as a variable of type Day
Enumerated Data Types

- What is an enumerator?
- Think of it as an integer named constant
- Internally, the compiler assigns integer values to the enumerators, beginning at 0

```c
enum Day { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, Friday };
```

In memory:
```
MONDAY = 0
TUESDAY = 1
WEDNESDAY = 2
THURSDAY = 3
FRIDAY = 4
```
Enumerated Data Types

- Using the Day declaration, the following code...
  ```cpp
cout << MONDAY << " " << WEDNESDAY << " "
<< FRIDAY << endl;
```
... produces the output
0 2 4

Assigning to a `enum` Variable

- You cannot directly assign an integer value to an enum variable
  ```cpp
  Day workDay = Thursday;
  Day workDay = 3; // Error
  ```
- Instead, you must cast the integer
  ```cpp
  workday = static_cast<Day>(3);
  workday = (Day)3;
  ```
Comparing Enumerator Value

- Since enumerators are essentially integers, they can be compared like integers
  
  ```
  if ( FRIDAY < workDay ) ... 
  ```

- Enumerator can be used to control a loop:
  ```
  for ( index = MONDAY, index <= Friday; index++ )
  {
    cout << "Enter the sale for day " << index << " : ";
    cin >> sales[index];
  } // for
  ```