Chapter 8
(Part 3)

High Level Programming Languages
Chapter Goals

- Define the concepts of a data type and strong typing
- Explain the concept of a parameter and distinguish between value and reference parameters
- Describe two composite data-structuring mechanisms
- Name, describe, and give examples of the three essential ingredients of an object-oriented language
- ... Some Hands-On
A Little Hands On
Hello World

<html>
<body>
<script type="text/javascript">
document.write("Hello World!")
</script>
</body>
</html>
Looping Statements

• The **while** statement is used to repeat a course of action.

• Let’s look at two distinct types of repetitions.
Looping Statements

- **Count-controlled loops**
  - Repeat a specified number of times
  - Use of a special variable called a loop control variable

Figure 8.4
Flow of control of *while* statement
## Looping Statements

- **Count-controlled** loops

<table>
<thead>
<tr>
<th>Language</th>
<th>Count-Controlled Loop with a while Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ada</td>
<td>Count := 1;</td>
</tr>
<tr>
<td></td>
<td>while Count &lt;= Limit loop</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>Count := Count + 1;</td>
</tr>
<tr>
<td></td>
<td>end loop;</td>
</tr>
<tr>
<td>VB.NET</td>
<td>Count = 1</td>
</tr>
<tr>
<td></td>
<td>While (count &lt;= limit)</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>count = count + 1</td>
</tr>
<tr>
<td></td>
<td>End While</td>
</tr>
<tr>
<td>C++/Java</td>
<td>count = 1;</td>
</tr>
<tr>
<td></td>
<td>while (count &lt;= limit)</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>count = count + 1;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>
Looping Statements

- **Event-controlled loops**
  - The number of repetitions is controlled by an event that occurs within the body of the loop itself

<table>
<thead>
<tr>
<th>Read a value</th>
<th>Initialize event</th>
</tr>
</thead>
<tbody>
<tr>
<td>While (value &gt;= 0)</td>
<td>Test event</td>
</tr>
<tr>
<td>...</td>
<td>Body of loop</td>
</tr>
<tr>
<td>Read a value</td>
<td>Update event</td>
</tr>
<tr>
<td>...</td>
<td>Statement(s) following loop</td>
</tr>
</tbody>
</table>
Looping Statements

- Event-controlled loops

Set sum to 0
Set posCount to 0
While (posCount <= 10)
    Read a value
    If (value > 0)
        Set posCount to posCount + 1
        Update event
        Add value into sum
    
    Set sum to sum + value
    Statement(s) following loop

Initialize sum to zero
Initialize event
Test event
Test to see if event should be updated

Page 249
<html>
<body>
<script type="text/javascript">
var i=0
while (i<=10)
{
    document.write("The number is " + i)
    document.write("<br />")
    i=i+1
}
</script>
</body>
</html>
Looping Statement

The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
The number is 10
Subprogram Statements

- We can give a section of code a name and use that name as a statement in another part of the program.
- When the name is encountered, the processing in the other part of the program halts while the named code is executed.
Subprogram Statements

• There are times when the calling unit needs to give information to the subprogram to use in its processing.

• A parameter list is a list of the identifiers with which the subprogram is to work, along with the types of each identifier placed in parentheses beside the subprogram name.
Subprogram Statements

(a) Subprogram A does its task and calling unit continues with next statement

Figure 8.5 Subprogram flow of control
Subprogram Statements

(b) Subprogram B does its task and returns a value that is added to 5 and stored in \( x \)

\[
x = 5 + \text{SubprogramB}()
\]

**Figure 8.5 Subprogram flow of control**
Subprogram Statements

• **Parameters**  Identifiers listed in parentheses beside the subprogram declaration; sometimes they are called **formal parameters**

• **Arguments**  Identifiers listed in parentheses on the subprogram call; sometimes they are called **actual parameters**
Subprogram Statements

- **Value parameter**  A parameter that expects a **copy** of its argument to be passed by the calling unit (put on the message board)

- **Reference parameter**  A parameter that expects the **address** of its argument to be passed by the calling unit (put on the message board)
## Subprogram Statements

<table>
<thead>
<tr>
<th>Language</th>
<th>Subprogram Declaration</th>
</tr>
</thead>
</table>
| **VB.NET** | Public Sub Example(ByVal one As Integer, ByVal two As Integer, ByRef three As Single)  
...  
End Sub |
| **C++/Java** | void Example(int one; int two; float& three)  
{  
...  
} |
Functions

```html
<html>
<head>
<script type="text/javascript">
function displaymessage()
{
alert("Hello World!"
}
</script>
</head>

<body>
<form>
<input type="button" value="Click me!" onclick="displaymessage()" >
</form>
</body>
</html>
```
Recursion

- **Recursion**  The ability of a subprogram to call itself

- Each recursive solution has **at least two cases**
  - **Base case**  The case to which we have an answer
  - **General case**  The case that expresses the solution in terms of a call to itself with a smaller version of the problem

- For example, the factorial of a number is defined as the number times the product of all the numbers between itself and 0:

  \[ N! = N \times (N - 1)! \]
Asynchronous Processing

- **Asynchronous processing** The concept that input and output can be accomplished through windows on the screen
  - *Clicking* has become a major form of input to the computer
  - Mouse clicking is not within the sequence of the program
  - A user can click a mouse at any time during the execution of a program
  - This type of processing is called **asynchronous**
Composite Data Types

- **Records**
  - A record is a named *heterogeneous* collection of items in which individual items are accessed by name
  - The *elements* in the collection *can be of various types*
## Composite Data Types

<table>
<thead>
<tr>
<th>Language</th>
<th>Record Type Declaration</th>
</tr>
</thead>
</table>
| Ada      | type Name_String is String (1..10);  
           | type Employee_Type is  
           |     record  
           |     Name : Name_String;  
           |     Age : Integer range 0..100;  
           |     Hourly_Wage : Float range 1.0..5000.0;  
           |     end record;           |
| VB.NET   | Structure Employee  
           |     Dim Name As String  
           |     Dim Age As Integer  
           |     Dim HourlyWage As Single  
           |     End Structure          |
| C++      | struct EmployeeType  
           |     {  
           |     string name;  
           |     int age;  
           |     float hourlyWage;  
           |     };  

## Composite Data Types

<table>
<thead>
<tr>
<th>Language</th>
<th>Record Variable Declaration and Usage</th>
</tr>
</thead>
</table>
| **Ada**  | An_Employee : Employee_Type;
          | ...                                    |
|          |   An_Employee.Name = "Sarah Gale";    |
|          |   An_Employee.Age = 32;               |
|          |   An_Employee.Hourly_Wage = 95.00;    |
| **VB.NET** | Dim AnEmployee As EmployeeType       |
|          |   ...                                 |
|          |   AnEmployee.Name = "Sarah Gale"     |
|          |   AnEmployee.Age = 32                 |
|          |   AnEmployee.HourlyWage 95.00        |
| **C++**  | EmployeeType anEmployee;             |
|          |   ...                                 |
|          |   anEmployee.name = "Sarah Gale";    |
|          |   anEmployee.age = 32;               |
|          |   anEmployee.hourlyWage = 95.00;     |
Arrays

- An **array** is a named collection of **homogeneous** items in which individual items are **accessed by their place within the collection**
  - The **place** within the collection is called an **index**

<table>
<thead>
<tr>
<th>Language</th>
<th>Array Declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ada</td>
<td>type Index_Range is range 1..10;</td>
</tr>
<tr>
<td></td>
<td>type Ten_Things is array (Index_Range) of Integer;</td>
</tr>
<tr>
<td>VB.NET</td>
<td>Dim TenThings(10) As Integer</td>
</tr>
<tr>
<td>C++/Java</td>
<td>int tenThings[10];</td>
</tr>
</tbody>
</table>
### Arrays

**Figure 8.8**

Array variable `tenThings` accessed from 0..9

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>1066</td>
</tr>
<tr>
<td>[1]</td>
<td>1492</td>
</tr>
<tr>
<td>[2]</td>
<td>1668</td>
</tr>
<tr>
<td>[3]</td>
<td>1945</td>
</tr>
<tr>
<td>[5]</td>
<td>1510</td>
</tr>
<tr>
<td>[6]</td>
<td>999</td>
</tr>
<tr>
<td>[7]</td>
<td>1001</td>
</tr>
<tr>
<td>[8]</td>
<td>21</td>
</tr>
</tbody>
</table>
Functionality of Object-Oriented Languages

- Encapsulation
- Inheritance
- Polymorphism
Encapsulation

• **Encapsulation**  A language feature that enforces information hiding

• **Class**  A language construct that is a pattern for an object and provides a mechanism for encapsulating the properties and actions of the object class

• **Instantiate**  Create an object from a class
Inheritance

- **Inheritance**  A construct that fosters reuse by allowing an application to take an already-tested class and derive a class from it that inherits the properties the application needs.

- **Polymorphism**  The ability of a language to have duplicate method names in an inheritance hierarchy and to apply the method that is appropriate for the object to which the method is applied.
Inheritance and polymorphism combined allow the programmer to build useful hierarchies of classes that can be reused in different applications.

Figure 8.9
Mapping of problem into solution
Homework

• Read Chapter Eight, Sections 8.3 – 8.4

• “PLAY” with JavaScript
http://www.w3schools.com/js/js_howto.asp

• Do some of the hands-on examples in class

• Program Assignment (#2) Next Class
Mid-Term

- Good Results
- Some incompletes
- Will review next class
Have A Great Weekend