Procedures -- Abstracting Common Operations

CSE 100

Algorithms are easier to formulate with a rich repertoire of operations, but for computers to execute the resulting programs, the operations must be simple operations. Procedures allow one to build powerful operations from simpler parts.

Functional Abstraction

- As noted in Lecture 4, Slide 5, “functional abstraction” is a powerful tool for algorithmic thinking*
- “Functional abstraction” means formulating the basic operations of a solution to a task in a more abstract form, i.e. independent of specific details, so that they may be reused, i.e. applied in many situations
- The important points --
  - Identifying the key operations of a task -- core logic of solution
  - Generalizing from the specific details -- parameterizing
  - Formulating the package for reuse -- assigning name, coding
- That’s very conceptual; consider some examples

* These concepts are covered in Chapter 4 of *Great Ideas in CS*

CSE 100 Procedures Are Everywhere

- The result of applying “functional abstraction” to problems is to create *procedures* or *functions*
- Examples …
  - Billing procedure for a company
  - Appeal procedure for capital crimes
  - UW Registration Procedure
  - Qualifying Dependent Test for the IRS
  - …
- Not Examples
  - Assembly instructions for a toy -- specific to the toy
  - π -- though a ratio, is just a number, but computing π could be
  - “Wake-up!” -- a single, immediate command with no reuse

CSE 100 Abstraction leads to parameterization

- Extracting the general process from a solution implies separating from particular instances …

```
❑ currentTemp = (5 / 9)*(reading - 32) ‘temp in C
❑ change = (5/9)*(midNiteTemp - 32)
          - (5/9)*(noonTemp - 32)   ‘figure diff in C
```

- The essential process in converting Fahrenheit to Celsius is the differencing and the product
- The parameter -- the component that changes from situation to situation -- is the Fahrenheit temperature

```
Function ConvertToC (tempF) As Integer
  ConvertToC = (5/9)*(tempF - 32)
End Function
…
currentTemp = ConvertToC(reading)
```

CSE 100 Functions and Procedures

- In computing, procedures perform an operation and have an effect; functions perform an operation and return a value
- Visual Basic has both, but we concentrate on procedures because of our programming style

	Procedure	Functions
Header key words	**`Sub Name()`**	**`Function Name() As Type`**
Trailer key words	**`End Sub`**	**`End Function`**

- Parameters are listed in parentheses
- The main difference is a function can return a value (whatever is assigned to its name), and so must have a type for the result given in the header

CSE 100 Parameters

- The names listed in the procedure header are the formal names used to program the computation in the body, i.e. procedure definition … they are local to the procedure, i.e. not known outside of it
- It is advisable to give the types of the parameters

Procedure name

Parameter name

Parameter type

```
Sub setReply (sign As String)
  lblAnnounce.Caption = “You are a ” & sign & “!”
End Sub
```

CSE 100 Using Procedures

- Procedures are defined in your form code, at the top after the “Option Explicit” line
- Procedures are invoked, I.e. executed, by being called using “Call”
- The parameter values given in the call are the *actual* parameters

```
n = 1000
x = 4
y = 28
Call sampleProc(n, x + y)
```

CSE 100 More On Parameters

- Parameters are a channel for passing data to a procedure … but it can also be a channel for passing data out

```
Sub switch (first, second As Integer)
Dim temp As Integer
  temp = first
  first = second
  second = temp
End Sub
…
Call switch (x,y)
```

- This is both beneficial and dangerous

CSE 100 Keeping Values In

- Parameters are normally “called by reference”
- Parameters can also be “called by value” which means that assignments to the (formal) parameters in the procedure to not affect the (actual) parameters

```
Sub cvt(degreesC As Integer, ByVal fahren As Integer)
    fahren = fahren - 32
    degreesC = (5/9)*fahren
End Sub
```