

J.P. Cohoon and I.W. Davidson
© 1999 McGraw-Hill, Inc.

Advanced Parameter Passing

Reference parameters, const parameters, and default parameters

Ch 7 / Foil 2

Consider

```
int main()
{ int Number1;
  int Number2;
  Number1 = PromptAndRead();
  Number2 = PromptAndRead();
  if (Number1 > Number2) {
    Swap(Number1, Number2);
  }
  cout << "The numbers in sorted order:"
        << Number1 << " " << Number2 << endl;
  return 0;
}
```

Ch 7 / Foil 3

Using

```
void Swap(int Value1, int Value2)
{ int Temp = Value1;
  Value1 = Value2;
  Value2 = Temp;
  return;
}
```

Ch 7 / Foil 4

Call by Reference

- If the formal argument declaration is of the form `pitypei &pnamei`, then
 - formal argument `pnamei` is an *alias* for the *i*th actual argument of the function call
 - `pnamei` is a *reference* parameter
 - Changes to the formal parameter change the actual parameter

Ch 7 / Foil 5

Reconsider

```

int main()
{
  int Number1;
  int Number2;
  Number1 = PromptAndRead();
  Number2 = PromptAndRead();
  if (Number1 > Number2)
  {
    Swap(Number1, Number2);
  }
  cout << "The numbers in sorted order: "
        << Number1 << ", " << Number2 << endl;
  return 0;
}

```

Ch 7 / Foil 6

Using

```

void Swap(int &Value1, int &Value2)
{
  int Temp = Value1;
  Value1 = Value2;
  Value2 = Temp;
  return;
}

```

Return statement not necessary for void functions

Passed by reference -- in an invocation the actual parameter is given rather than a copy

Ch 7 / Foil 7

Extraction

- Function to extract a value from a given stream

```
void GetNumber(istream &sin, int &MyNumber)
{
  sin >> MyNumber;
  return;
}
```

Why is the stream a reference parameter?
Why is the number a reference parameter?

Ch 7 / Foil 8

Getnum.cpp

```
int main()
{
  int Number1;
  int Number2;
  GetNumber(cin, Number1);
  GetNumber(cin, Number2);
  if (Number1 > Number2)
  {
    Swap(Number1, Number2);
  }
  cout << "The numbers in sorted order: "
        << Number1 << ", " << Number2 << endl;
  return 0;
}
```

Ch 7 / Foil 9

Constant Parameters

- The const modifier can be applied to formal parameter declarations
 - Const indicates that the function may not modify the parameter


```
void PromptAndGet(int &n, const string &s)
{
  cout << s ;
  cin >> n ;
  s = "Got it"; // illegal assignment
}
```
 - Sample invocation


```
int x;
PromptAndGet(x, "Enter number (n): ");
```

Ch 7 / Foil 10

Constant Parameters

- Usefulness
 - When we want to pass an object by reference, but we do not want to let the called function modify the object
- Question
 - Why not just pass the object by value?
- Answer
 - For large objects, making a copy of the object can be very inefficient

Ch 7 / Foil 11

Passing Constant Rectangles

```

void DrawBoxes(const RectangleShape &R1,
               const RectangleShape &R2)
{
  R1.Draw();
  R2.Draw();
}

int ApiMain()
{
  SimpleWindow Demo("Demo Program");
  Demo.Open();
  RectangleShape Rect1(Demo, 3, 2, Blue);
  RectangleShape Rect2(Demo, 6, 5, Yellow);
  DrawBoxes(Rect1, Rect2);
  return 0;
}

```

Ch 7 / Foil 12

Default Parameters

- Observations
 - Our functions up to this point required that we explicitly pass a value for each of the function parameters
 - It would be convenient to define functions that accept a varying number of parameters
- Default parameters
 - Allows programmer to define a default behavior
 - A value for a parameter can be implicitly passed
 - Reduces need for similar functions that differ only in the number of parameters accepted

Ch 7 / Foil 13

Default Parameters

- If the formal argument declaration is of the form


```
pitypei pinamei = divaluei
```

 then
 - If there is no *i*th argument in the function invocation, *p_iname_i* is initialized to *d_ivalue_i*
 - The parameter *p_iname_i* is an optional value parameter
 - Optional reference parameters are also permitted

Ch 7 / Foil 14

Consider

```
void PrintChar(char c = '=', int n = 80)
{ for (int i = 0; i < n; ++i)
  cout << c;
}
```

- What happens in the following invocations?


```
PrintChar('*', 20);
PrintChar('-');
PrintChar();
```

Ch 7 / Foil 15

Default Parameters

- Default parameters must appear after any mandatory parameters
- Example


```
void Trouble(int x = 5, double z, double y)
{
  ...
}
```

Ch 7 / Foil 16

Default Parameters

- Consider


```
bool GetNumber(int &n, istream &sin = cin)
{ return sin >> n ;
}
```
- Some possible invocations


```
int x, y, z;
ifstream fin("Data.txt");
GetNumber(x, cin);
GetNumber(y);
GetNumber(z, fin);
```
- Design your functions for ease and reuse!

Ch 7 / Foil 17

Casting of Function Parameters

- Calling a function is much like applying an operator to operands
- When a function is called
 - Compiler attempts to convert (coerce) the actual parameters to the types required by the formal parameters of the function

```
// ComputeInterest(): compute interest
double ComputeInterest(double Principle,
double InterestRate, double CompoundRate,
double Years);
...
double MyInterest =
    ComputeInterest(100, .075, 1, 10);
```

Ch 7 / Foil 18

Interest.cpp

```
int main()
{ cout << "Interest is "
  << ComputeInterest(100, .10, 1, 10) << endl;
  return 0;
}

double ComputeInterest(double Principle,
double InterestRate, double CompoundRate,
double Years)
{ double Interest = Principle
  * pow(1 + InterestRate, CompoundRate * Years);
  return Interest;
}
```

Ch 7 / Foil 19

Casting Parameters

The screenshot shows a C++ IDE with the following code:

```

cout << "Interest is " << computeInterest(100, .08, 1, 10) << endl;
return 0;
// computeInterest() computes interest on principle compounded
// Compounded times a year, for years years
double computeInterest(double Principle, double InterestRate,
double Compounded, double Years) {
    return 0;
}

```

The output window shows: `Interest is 110.8`

Ch 7 / Foil 20

Function Overloading

- A function name can be overloaded
 - Two functions with the same name but with different interfaces (i.e. parameter lists)
 - Typically this means different formal parameter lists
 - Difference in number of parameters
 - Min(a, b, c)
 - Min(a, b)
 - Difference in types of parameters
 - Min(10, 20)
 - Min(4.4, 9.2)

Ch 7 / Foil 21

Function Overloading

```

int Min(int a, int b)
{ cout << "Using int min()" << endl;
  if (a > b)
    return b;
  else
    return a;
}
double Min(double a, double b)
{ cout << "Using double min()" << endl;
  if (a > b)
    return b;
  else
    return a;
}

```

Ch 7 / Foil 22

Min.cpp

```

int main()
{ int a = 10;
  int b = 20;
  double x = 4.4;
  double y = 9.2;
  int c = Min(a, b);
  cout << "c is " << c << endl;
  int z = Min(x, y);
  cout << "z is " << z << endl;
  return 0;
}
    
```

Ch 7 / Foil 23


Function Overloading

- Compiler uses function overload resolution to call the most appropriate function
 - If a function definition exists where the type of the formal parameters exactly match the types of the actual parameters, then that function definition is invoked
 - If there is no exact match, the compiler will attempt to cast the actual parameters so that an appropriate function definition (if any) can be invoked
- The rules for function definition overloading are very complicated
 - Advice
 - Be very careful when using this feature

Ch 7 / Foil 24

Random Numbers

- Generating a sequence of random numbers is often useful
 - In a game, it ensures that a player does not see the same behavior each time
 - In a simulation of a complex system, random numbers can be used to help generate random events
 - Car crash in a simulation of a highway system
 - Likelihood of a gene in cell mutation
 - Weather simulation





Ch 7 / Foil 25

Uniform Random Numbers

- Uniform random number sequence
 - A sequence of random numbers where
 - Each value in the sequence is drawn from the same range of numbers
 - In each position of the sequence, any value in the number range is equally likely to occur


Ch 7 / Foil 26

Random Numbers

- Examples
 - Generate a uniform random number sequence in the range 1 to 6
 - Use a fair six-sided dice 
 - Each roll represents a new random number
 - Do two die produce uniform random numbers in the range 1 ... 12?
 - Generate a uniform random number sequence in the range 1 to 2
 - Use a fair coin 
 - Heads: 1, Tails: 2

Ch 7 / Foil 27

Random Numbers

- We can write an algorithm for generating what looks like random numbers  30 21 9 28 29 ...
- Because it's an algorithm, we know the rules for generating the next number
 - The generated numbers are not really random
 - They are properly called pseudorandom numbers

Ch 7 / Foil 28

Stdlib Library

- Provides in part an interface to functions that generate pseudorandom numbers
 - `rand()` -- returns a uniform pseudorandom number (unsigned int) from the inclusive interval 0 to `RAND_MAX`
- Consider `rand.cpp`

```

#include <iostream>
#include <string>
using namespace std;
#include <stdlib.h>
int main()
{ for (int i = 1; i <= 5; ++i)
  cout << rand() << endl;
  return 0;
}

```

Ch 7 / Foil 29

Different Sequences

- To produce a different sequence, invoke `void srand(unsigned int);`
- Consider `seed.cpp`

```

int main()
{ cout << "Enter a seed: ";
  unsigned int Seed;
  cin >> Seed;
  srand(Seed);
  for (int i = 1; i <= 5; ++i)
    cout << rand() << endl;
  return 0;
}

```

Ch 7 / Foil 30

Different Sequences

- To get a different sequence each time
 - Need a method of setting the seed to a random value
 - The standard method is to use the computer's clock as the value of the seed
 - The function invocation `time()` can be used
 - Returns an integral value of type `time_t`
 - Invocation `time(0)` returns a suitable value for generating a random sequence

Ch 7 / Foil 31

Randseed.cpp

```

#include <iostream>
#include <string>
using namespace std;
#include <stdlib.h>
#include <time.h>
int main()
{
  srand((unsigned int) time(0));
  for (int i = 1; i <= 5; ++i)
    cout << rand() << endl;
  return 0;
}

```

Ch 7 / Foil 32

A Factory Automation Trainer

- Problem statement
 - Design and implement a program to evaluate and train quality control inspectors

Ch 7 / Foil 33

A Factory Automation Trainer

- Object decomposition
 - Video camera
 - Display unit
 - Control unit
 - Parts
 - Simulator control

Ch 7 / Foil 34

A Factory Automation Trainer

- Because we do not have program-defined classes yet, we make functions to produce object behaviors

```

graph TD
    GetPartsImages[GetPartsImages()] --> ApiMain[ApiMain()]
    ApiMain --> DisplayPartsImages[DisplayPartsImages()]
    ApiMain --> CheckResponse[CheckResponse()]
    ApiMain --> DisplaySessionResults[DisplaySessionResults()]
    DisplayPartsImages --> CheckResponse
  
```

Ch 7 / Foil 35

A Factory Automation Trainer

- Simulation controller
 - Get parts image
 - Display parts images in the display window
 - Read and record the response of the trainee
 - Score the response
 - Check to see if training session should end
 - If time is not up, go back to step 1
 - If time is up, compute and print the statistics

Ch 7 / Foil 36

A Factory Automation Trainer

```

#include <iostream>           // Program 7.10
#include <string>
using namespace std;
#include <assert.h>
#include "uniform.h"
#include "rect.h"
// Size of display window
const float DisplayWidth = 14.0;
const float DisplayHeight = 6.0;
// Vertical position of squares
const float YPosition = DisplayHeight / 2.0;
// Length of the session (60 seconds)
const long TestTime = 60 * 1000L;
  
```

Ch 7 / Foil 37

A Factory Automation Trainer

```

// Randomly pick one of the six colors
color GenerateRandomColor()
{
  switch (Uniform(0, 5))
  {
    case 0: return Red;
    case 1: return Blue;
    case 2: return Green;
    case 3: return Yellow;
    case 4: return Cyan;
    case 5: return Magenta;
  }
}
    
```

Ch 7 / Foil 38

A Factory Automation Trainer

```

// Generate the test parts by randomly
// setting the color of the three parts
void GetPartsImages(RectangleShape &P1,
  RectangleShape &P2, RectangleShape &P3)
{
  P1.SetColor(GenerateRandomColor());
  P2.SetColor(GenerateRandomColor());
  P3.SetColor(GenerateRandomColor());
  return;
}
    
```

Ch 7 / Foil 39

A Factory Automation Trainer

```

// Display the shapes
void DisplayPartsImages(const RectangleShape &P1,
  const RectangleShape &P2,
  const RectangleShape &P3)
{
  P1.Draw();
  P2.Draw();
  P3.Draw();
  return;
}
    
```

Ch 7 / Foil 40

A Factory Automation Trainer

```

// Print the results from the training session
void PrintSessionResults(long Time, int Attempts,
int Correct, int Wrong)
{ cout << "\n\nFor a training period of "
  << Time / 1000L << " seconds" << endl;
  cout << "Groups viewed: " << Attempts << endl;
  cout << "Correct responses: " << Correct << endl;
  cout << "Incorrect responses: " << Wrong << endl;
  cout << "Percent correct: "
    << Correct / (float) Attempts * 100 << endl;
  return;
}
    
```

Ch 7 / Foil 41

A Factory Automation Trainer

```

// Determine if the appropriate response was given
bool CheckResponse(char Response,
const RectangleShape &P1,
const RectangleShape &P2,
const RectangleShape &P3)
{
  if (P1.GetColor() == P2.GetColor()
    || P1.GetColor() == P3.GetColor()
    || P2.GetColor() == P3.GetColor())
  { return Response == 'a';
  }
  else
  { return Response == 'r';
  }
}
    
```

Ch 7 / Foil 42

A Factory Automation Trainer

```

int ApiMain()
{
  InitializeSeed();
  // Window for displaying the objects
  SimpleWindow DisplayWindow("Training Window",
  DisplayWidth, DisplayHeight);
  DisplayWindow.Open();
  assert(DisplayWindow.GetStatus() == WindowOpen);
}
    
```

Ch 7 / Foil 43

A Factory Automation Trainer

```

// Print message telling user to arrange windows
cout << "Please resize this window so that\n"
<< "that both windows are visible,\n"
<< "and they do not overlap.\n"
<< "Type any character followed by a return\n"
<< "when you are ready to proceed" << endl;
char Response;
cin >> Response;
    
```

Ch 7 / Foil 44

A Factory Automation Trainer

```

cout << "\n\n\n";
// Create three rectangles for the three parts
RectangleShape Part1(DisplayWindow,
3.0, YPosition, Blue, 2.0, 2.0);
RectangleShape Part2(DisplayWindow,
7.0, YPosition, Blue, 2.0, 2.0);
RectangleShape Part3(DisplayWindow,
11.0, YPosition, Blue, 2.0, 2.0);
// Define objects for scoring the trainee
int Attempts = 0; // Number of tests done
int CorrectResponses = 0;
int IncorrectResponses = 0;
    
```

Ch 7 / Foil 45

A Factory Automation Trainer

```

// Record starting time
const long StartTime = GetMilliseconds();
long ElapsedTime;
do {
    GetPartsImages(Part1, Part2, Part3);
    DisplayPartsImages(Part1, Part2, Part3);

    cout << "Accept or reject (a or r)? ";
    char Response;
    cin >> Response;
}
    
```

Ch 7 / Foli 46

A Factory Automation Trainer

```
++Attempts;
if (CheckResponse(Response, Part1, Part2,
    Part3))
    ++CorrectResponses;
else
    ++IncorrectResponses;
ElapsedTime = GetMilliseconds();
} while ((ElapsedTime - StartTime) < TestTime);
PrintSessionResults(TestTime, Attempts,
    CorrectResponses, IncorrectResponses);
return 0;
}
```
