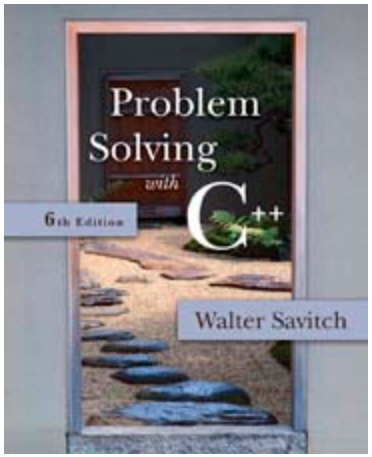
APS105: Lecture 11

Wael Aboelsaadat

wael@cs.toronto.edu

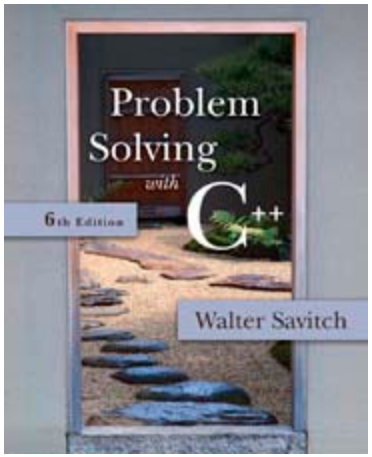
http://ccnet3.utoronto.ca/20079/aps105h1f/

Acknowledgement: These slides are a modified version of the text book slides as supplied by Addison Wesley



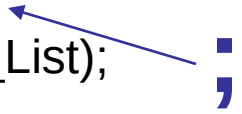
4.3

Programmer-Defined Functions



Programmer-Defined Functions

- Two components of a function definition
  - Function declaration (or function prototype)
    - Shows how the function is called
    - Must appear in the code before the function can be called
    - Syntax:
      Type_returned Function_Name(Parameter_List);
      //Comment describing what function does



  - Function definition
    - Describes how the function does its task
    - Can appear before or after the function is called
    - Syntax:
      Type_returned Function_Name(Parameter_List)
      {
      //code to make the function work
      }

Function Declaration

- Tells the return type
- Tells the name of the function
- Tells how many arguments are needed
- Tells the types of the arguments
- Tells the formal parameter names
  - Formal parameters are like placeholders for the actual arguments used when the function is called
  - Formal parameter names can be any valid identifier
- Example:

```
double total_cost(int number_par, double price_par);
// Compute total cost including 5% sales tax on
// number_par items at cost of price_par each
```

Function Definition

- Provides the same information as the declaration
- Describes how the function does its task
- Example:


```
double total_cost(int number_par, double price_par)
{
    const double TAX_RATE = 0.05; //5% tax
    double subtotal;
     subtotal = price_par * number_par;
    return (subtotal + subtotal * TAX_RATE);
}
```

function body

The Return Statement

- Ends the function call
- Returns the value calculated by the function
- Syntax:

  return expression;

  - expression performs the calculation
    or
  - expression is a variable containing the calculated value
- Example:

  return subtotal + subtotal * TAX_RATE;

The Function Call

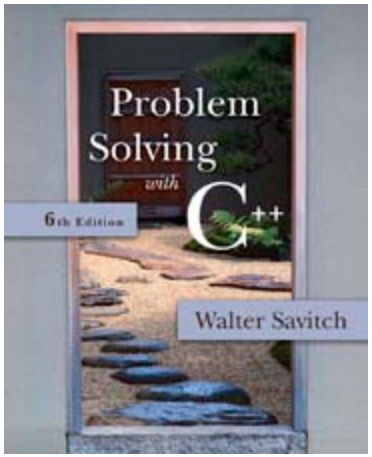
- Tells the name of the function to use
- Lists the arguments
- Is used in a statement where the returned value makes sense
- Example:

```
double bill = total_cost(number, price);
```

Display 4.3

4.1

Top-Down Design



Top Down Design

- To write a program
  - Develop the algorithm that the program will use
  - Translate the algorithm into the programming language

- Top Down Design
  (also called stepwise refinement)
  - Break the algorithm into subtasks
  - Break each subtask into smaller subtasks
  - Eventually the smaller subtasks are trivial to implement in the programming language

Benefits of Top Down Design

- Subtasks, or functions in C++, make programs
  - Easier to understand
  - Easier to change
  - Easier to write
  - Easier to test
  - Easier to debug
  - Easier for teams to develop

Type Casting

- Recall the problem with integer division:
  int total_candy = 9, number_of_people = 4;
  double candy_per_person;
  candy_per_person = total_candy / number_of_people;
  - candy_per_person = 2, not 2.25!
- A Type Cast produces a value of one type from another type
  - static_cast<double>(total_candy) produces a double representing the integer value of total_candy

Type Cast Example

- int total_candy = 9, number_of_people = 4;
  double candy_per_person;
  candy_per_person = static_cast<double>(total_candy)
  / number_of_people;
  - candy_per_person now is 2.25!
  - This would also work:
    candy_per_person = total_candy /
    static_cast<double>(number_of_people);
  - This would not!
    candy_per_person = static_cast<double>(total_candy /
    number_of_people);

Integer division occurs before type cast

Old Style Type Cast

- C++ is an evolving language
- This older method of type casting may be discontinued in future versions of C++

```
candy_per_person =
double(total_candy)/number_of_people;
```

Section 4.2 Conclusion

- Can you
  - Determine the value of d?

    double d = 11 / 2;

  - Determine the value of

pow(2,3)	fabs(-3.5)	sqrt(pow(3,2))
7 / abs(-2)	ceil(5.8)	floor(5.8)

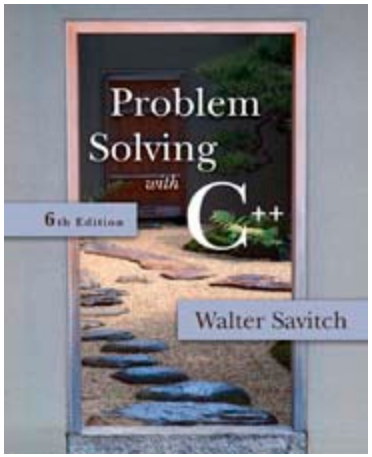
  - Convert the following to C++

$$\sqrt{x+y}$$

$$\frac{-b+\sqrt{b^2-4ac}}{2a}x+7$$

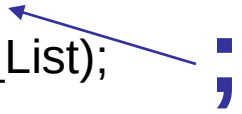
4.3

Programmer-Defined Functions



Programmer-Defined Functions

- Two components of a function definition
  - Function declaration (or function prototype)
    - Shows how the function is called
    - Must appear in the code before the function can be called
    - Syntax:
      Type_returned Function_Name(Parameter_List);
      //Comment describing what function does



  - Function definition
    - Describes how the function does its task
    - Can appear before or after the function is called
    - Syntax:
      Type_returned Function_Name(Parameter_List)
      {
      //code to make the function work
      }

Function Declaration

- Tells the return type
- Tells the name of the function
- Tells how many arguments are needed
- Tells the types of the arguments
- Tells the formal parameter names
  - Formal parameters are like placeholders for the actual arguments used when the function is called
  - Formal parameter names can be any valid identifier
- Example:

```
double total_cost(int number_par, double price_par);
// Compute total cost including 5% sales tax on
// number_par items at cost of price_par each
```

Function Definition

- Provides the same information as the declaration
- Describes how the function does its task
- Example:


```
double total_cost(int number_par, double price_par)
{
    const double TAX_RATE = 0.05; //5% tax
    double subtotal;
     subtotal = price_par * number_par;
    return (subtotal + subtotal * TAX_RATE);
}
```

function body

The Return Statement

- Ends the function call
- Returns the value calculated by the function
- Syntax:

  return expression;

  - expression performs the calculation
    or
  - expression is a variable containing the calculated value
- Example:

  return subtotal + subtotal * TAX_RATE;

The Function Call

- Tells the name of the function to use
- Lists the arguments
- Is used in a statement where the returned value makes sense
- Example:

```
double bill = total_cost(number, price);
```

Display 4.3

Function Call Details

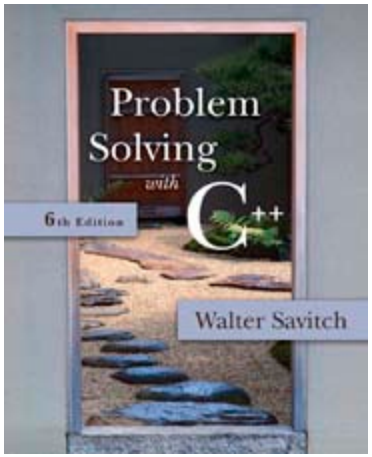
- The values of the arguments are plugged into the formal parameters (Call-by-value mechanism with call-by-value parameters)
  - The first argument is used for the first formal parameter, the second argument for the second formal parameter, and so forth.
  - The value plugged into the formal parameter is used in all instances of the formal parameter in the function body

Display 4.4 (1)

Display 4.4 (2)

4.5

Local Variables



Local Variables

- Variables declared in a function:
  - Are local to that function, they cannot be used from outside the function
  - Have the function as their scope
- Variables declared in the main part of a program:
  - Are local to the main part of the program, they cannot be used from outside the main part
  - Have the main part as their scope

Display 4.11 (1)

Display 4.11 (2)

Global Constants

- Global Named Constant
  - Available to more than one function as well as the main part of the program
  - Declared outside any function body
  - Declared outside the main function body
  - Declared before any function that uses it
- Example:

```
const double PI = 3.14159;
double volume(double);
 int main()
{…}
```

  - PI is available to the main function and to function volume

Display 4.12 (1)

Display 4.12 (2)

Global Variables

- Global Variable -- rarely used when more than one function must use a common variable
  - Declared just like a global constant except const is not used
  - Generally make programs more difficult to understand and maintain

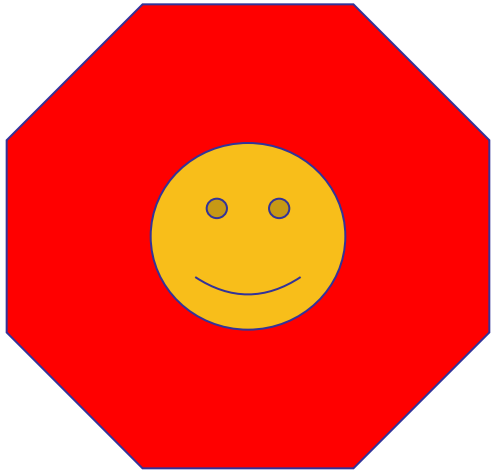
Formal Parameters are Local Variables

- Formal Parameters are actually variables that are local to the function definition
  - They are used just as if they were declared in the function body
  - Do NOT re-declare the formal parameters in the function body, they are declared in the function declaration
- The call-by-value mechanism
  - When a function is called the formal parameters are initialized to the values of the arguments in the function call

Display 4.13 (1)

Display 4.13 (2)

Chapter 4 -- End



Display 4.3 (1/2)

Back Next

A Function Definition (*part 1 of 2*)

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

double total_cost(int number_par, double price_par);
//Computes the total cost, including 5% sales tax,
//on number_par items at a cost of price_par each.

int main( )
{
    double price, bill;
    int number;

    cout << "Enter the number of items purchased: ";
    cin >> number;
    cout << "Enter the price per item $";
    cin >> price;

    bill = total_cost(number, price);

    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(2);
    cout << number << " items at "
         << "$" << price << " each.\n"
         << "Final bill, including tax, is $" << bill
         << endl;

    return 0;
}

double total_cost(int number_par, double price_par)
{
    const double TAX_RATE = 0.05; //5% sales tax
    double subtotal;

    subtotal = price_par * number_par;
    return (subtotal + subtotal*TAX_RATE);
}
```

function declaration

function call

function heading

function body

function definition

Display 4.3 (2/2)

A Function Definition (*part 2 of 2*)

Sample Dialogue

```
Enter the number of items purchased: 2
Enter the price per item: $10.10
2 items at  $10.10 each.
Final bill, including tax, is $21.21
```

Display 4.4 (1/2)

DISPLAY 4.4 Details of a Function Call *(part 1 of 2)*

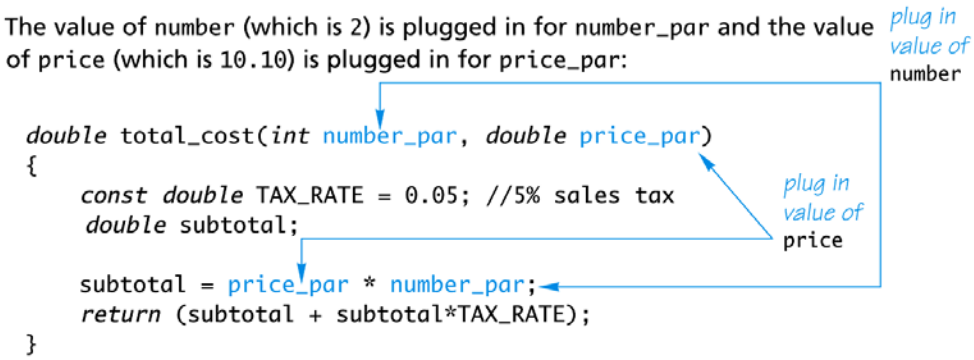
Anatomy of the Function Call in Display 4.3

0 Before the function is called, the values of the variables `number` and `price` are set to 2 and 10.10, by `cin` statements (as you can see in the Sample Dialogue in Display 4.3).

1 The following statement, which includes a function call, begins executing:

```
bill = total_cost(number, price);
```

2 The value of `number` (which is 2) is plugged in for `number_par` and the value of `price` (which is 10.10) is plugged in for `price_par`:




```
double total_cost(int number_par, double price_par)
{
    const double TAX_RATE = 0.05; //5% sales tax
    double subtotal;

    subtotal = price_par * number_par;
    return (subtotal + subtotal*TAX_RATE);
}
```

producing the following:

```
double total_cost(int 2, double 10.10)
{
    const double TAX_RATE = 0.05; //5% sales tax
    double subtotal;

    subtotal = 10.10 * 2;
    return (subtotal + subtotal*TAX_RATE);
}
```

Display 4.4 (2/2)

Back Next

DISPLAY 4.4 Details of a Function Call *(part 2 of 2)*

Anatomy of the Function Call in Display 4.3 (concluded)

3 The body of the function is executed, that is, the following is executed:

```
{
    const double TAX_RATE = 0.05; //5% sales tax
    double subtotal;

    subtotal = 10.10 * 2;
     return (subtotal + subtotal*TAX_RATE);
}
```

4 When the *return* statement is executed, the value of the expression after *return* is the value returned by the function. In this case, when

```
return (subtotal + subtotal*TAX_RATE);
```

is executed, the value of `(subtotal + subtotal*TAX_RATE)`, which is `21.21`, is returned by the function call

```
total_cost(number, price)
```

and so the value of `bill` (on the left-hand side of the equal sign) is set equal to `21.21` when the following statement finally ends:

```
bill = total_cost(number, price);
```

Display 4.5 (1/2)

Back Next

Incorrectly Ordered Arguments (*part 1 of 2*)

```
//Determines user's grade. Grades are Pass or Fail.
#include <iostream>
using namespace std;

char grade(int received_par, int min_score_par);
//Returns 'P' for passing, if received_par is
//min_score_par or higher. Otherwise returns 'F' for failing.

int main()
{
    int score, need_to_pass;
    char letter_grade;

    cout << "Enter your score"
         << " and the minimum needed to pass:\n";
    cin >> score >> need_to_pass;

    letter_grade = grade(need_to_pass, score);

    cout << "You received a score of " << score << endl
         << "Minimum to pass is " << need_to_pass << endl;

    if (letter_grade == 'P')
        cout << "You Passed. Congratulations!\n";
    else
        cout << "Sorry. You failed.\n";

    cout << letter_grade
         << " will be entered in your record.\n";

    return 0;
}

char grade(int received_par, int min_score_par)
{
    if (received_par >= min_score_par)
        return 'P';
    else
        return 'F';
}
```

Display 4.5 (2/2)

Back Next

Incorrectly Ordered Arguments (*part 2 of 2*)

Sample Dialogue

```
Enter your score and the minimum needed to pass:
98 60
You received a score of 98
Minimum to pass is 60
Sorry. You failed.
F will be entered in your record.
```

Display 4.6

Back Next

Syntax for a Function That Returns a Value

Function Declaration

Type_Returned Function_Name (Parameter_List) ;
Function_Declaration_Comment

Function Definition

Type_Returned Function_Name (Parameter_List) ← *function header*

body

```
{
    Declaration_1
    Declaration_2
        . . .
    Declaration_Last
    Executable_Statement_1
    Executable_Statement_2
        . . .
    Executable_Statement_Last
}
```

Must include one or more `return` *statements.*

Display 4.7

Back Next

Definitions That Are Black-Box Equivalent

Function Declaration

```
double new_balance(double balance_par, double rate_par);
//Returns the balance in a bank account after
//posting simple interest. The formal parameter balance_par is
//the old balance. The formal parameter rate_par is the interest rate.
//For example, if rate_par is 5.0, then the interest rate is 5%
//and so new_balance(100, 5.0) returns 105.00.
```

Definition 1

```
double new_balance(double balance_par, double rate_par)
{
    double interest_fraction, interest;

    interest_fraction = rate_par/100;
    interest = interest_fraction*balance_par;
    return (balance_par + interest);
}
```

Definition 2

```
double new_balance(double balance_par, double rate_par)
{
    double interest_fraction, updated_balance;

    interest_fraction = rate_par/100;
    updated_balance = balance_par*(1 + interest_fraction);
    return updated_balance;
}
```

Display 4.8

Simpler Formal Parameter Names

Function Declaration

```
double total_cost(int number, double price);
//Computes the total cost, including 5% sales tax, on
//number items at a cost of price each.
```

Function Definition

```
double total_cost(int number, double price)
{
    const double TAX_RATE = 0.05; //5% sales tax
    double subtotal;

    subtotal = price * number;
    return (subtotal + subtotal*TAX_RATE);
}
```

Display 4.9 (1/3)

Nicely Nested Loops (*part 1 of 3*)

```
//Determines the total number of green-necked vulture eggs
//counted by all conservationists in the conservation district.
#include <iostream>
using namespace std;

void instructions( );

void get_one_total(int& total);
//Precondition: User will enter a list of egg counts
//followed by a negative number.
//Postcondition: total is equal to the sum of all the egg counts.

int main( )
{
    instructions( );

    int number_of_reports;
    cout << "How many conservationist reports are there? ";
    cin >> number_of_reports;

    int grand_total = 0, subtotal, count;
    for (count = 1; count <= number_of_reports; count++)
    {
        cout << endl << "Enter the report of "
             << "conservationist number " << count << endl;
        get_one_total(subtotal);
        cout << "Total egg count for conservationist "
             << " number " << count << " is "
             << subtotal << endl;
        grand_total = grand_total + subtotal;
    }

    cout << endl << "Total egg count for all reports = "
         << grand_total << endl;

    return 0;
}
```

Display 4.9 (2/3)

Nicely Nested Loops (*part 2 of 3*)

```
//Uses iostream:
void instructions( )
{
    cout << "This program tallies conservationist reports\n"
         << "on the green-necked vulture.\n"
         << "Each conservationist's report consists of\n"
         << "a list of numbers. Each number is the count of\n"
         << "the eggs observed in one"
         << " green-necked vulture nest.\n"
         << "This program then tallies"
         << " the total number of eggs.\n";
}

//Uses iostream:
void get_one_total(int& total)
{
    cout << "Enter the number of eggs in each nest.\n"
         << "Place a negative integer"
         << " at the end of your list.\n";

    total = 0;
    int next;
    cin >> next;
    while (next >= 0)
    {
        total = total + next;
        cin >> next;
    }
}
```

Display 4.9 (3/3)

Nicely Nested Loops (*part 3 of 3*)

Sample Dialogue

```
This program tallies conservationist reports
on the green-necked vulture.
Each conservationist's report consists of
a list of numbers. Each number is the count of
the eggs observed in one green-necked vulture nest.
This program then tallies the total number of eggs.
How many conservationist reports are there? 3

Enter the report of conservationist number 1
Enter the number of eggs in each nest.
Place a negative integer at the end of your list.
1 0 0 2 -1
Total egg count for conservationist number 1 is 3

Enter the report of conservationist number 2
Enter the number of eggs in each nest.
Place a negative integer at the end of your list.
0 3 1 -1
Total egg count for conservationist number 2 is 4

Enter the report of conservationist number 3
Enter the number of eggs in each nest.
Place a negative integer at the end of your list.
-1
Total egg count for conservationist number 3 is 0

Total egg count for all reports = 7
```

Display 4.10 (1/2)

Back Next

Buying Pizza (*part 1 of 2*)

```
//Determines which of two pizza sizes is the best buy.
#include <iostream>
using namespace std;

double unitprice(int diameter, double price);
//Returns the price per square inch of a pizza. The formal
//parameter named diameter is the diameter of the pizza in inches.
//The formal parameter named price is the price of the pizza.

int main()
{
    int diameter_small, diameter_large;
    double price_small, unitprice_small,
           price_large, unitprice_large;

    cout << "Welcome to the Pizza Consumers Union.\n";
    cout << "Enter diameter of a small pizza (in inches): ";
    cin >> diameter_small;
    cout << "Enter the price of a small pizza: $";
    cin >> price_small;
    cout << "Enter diameter of a large pizza (in inches): ";
    cin >> diameter_large;
    cout << "Enter the price of a large pizza: $";
    cin >> price_large;

    unitprice_small = unitprice(diameter_small, price_small);
    unitprice_large = unitprice(diameter_large, price_large);

    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(2);
    cout << "Small pizza:\n"
         << "Diameter = " << diameter_small << " inches\n"
         << "Price = $" << price_small
         << " Per square inch = $" << unitprice_small << endl
         << "Large pizza:\n"
         << "Diameter = " << diameter_large << " inches\n"
         << "Price = $" << price_large
         << " Per square inch = $" << unitprice_large << endl;
```

Display 4.10 (2/2)

Back Next

Buying Pizza (*part 2 of 2*)

```
    if (unitprice_large < unitprice_small)
        cout << "The large one is the better buy.\n";
    else
        cout << "The small one is the better buy.\n";
    cout << "Buon Appetito!\n";

    return 0;
}

double unitprice(int diameter, double price)
{
    const double PI = 3.14159;
    double radius, area;

    radius = diameter/static_cast<double>(2);
    area = PI * radius * radius;
    return (price/area);
}
```

Sample Dialogue

```
Welcome to the Pizza Consumers Union.
Enter diameter of a small pizza (in inches): 10
Enter the price of a small pizza: $7.50
Enter diameter of a large pizza (in inches): 13
Enter the price of a large pizza: $14.75
Small pizza:
Diameter = 10 inches
Price = $7.50 Per square inch = $0.10
Large pizza:
Diameter = 13 inches
Price = $14.75 Per square inch = $0.11
The small one is the better buy.
Buon Appetito!
```

Display 4.11 (1/2)

Local Variables (*part 1 of 2*)

```
//Computes the average yield on an experimental pea growing patch.
#include <iostream>
using namespace std;

double est_total(int min_peas, int max_peas, int pod_count);
//Returns an estimate of the total number of peas harvested.
//The formal parameter pod_count is the number of pods.
//The formal parameters min_peas and max_peas are the minimum
//and maximum number of peas in a pod.

int main()
{
    int max_count, min_count, pod_count;
    double average_pea, yield;

    cout << "Enter minimum and maximum number of peas in a pod: ";
    cin >> min_count >> max_count;
    cout << "Enter the number of pods: ";
    cin >> pod_count;
    cout << "Enter the weight of an average pea (in ounces): ";
    cin >> average_pea;

    yield =
          est_total(min_count, max_count, pod_count) * average_pea;

    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(3);
    cout << "Min number of peas per pod = " << min_count << endl
         << "Max number of peas per pod = " << max_count << endl
         << "Pod count = " << pod_count << endl
         << "Average pea weight = "
         << average_pea << " ounces" << endl
         << "Estimated average yield = " << yield << " ounces"
         << endl;

    return 0;
}
```

This variable named average_pea *is local to the* main *part of the program.*

Display 4.11 (2/2)

Local Variables (*part 2 of 2*)

```
double est_total(int min_peas, int max_peas, int pod_count)
{
    double average_pea;

    average_pea = (max_peas + min_peas)/2.0;
    return (pod_count * average_pea);
}
```

This variable named average_pea *is local to the function est_total.*

Sample Dialogue

```
Enter minimum and maximum number of peas in a pod: 4 6
Enter the number of pods: 10
Enter the weight of an average pea (in ounces): 0.5
Min number of peas per pod = 4
Max number of peas per pod = 6
Pod count = 10
Average pea weight = 0.500 ounces
Estimated average yield = 25.000 ounces
```

A Global Named Constant (*part 1 of 2*)

Display 4.12 (1/2)

Back Next

```
//Computes the area of a circle and the volume of a sphere.
//Uses the same radius for both calculations.
#include <iostream>
#include <cmath>
using namespace std;

const double PI = 3.14159;

double area(double radius);
//Returns the area of a circle with the specified radius.

double volume(double radius);
//Returns the volume of a sphere with the specified radius.

int main()
{
    double radius_of_both, area_of_circle, volume_of_sphere;

    cout << "Enter a radius to use for both a circle\n"
         << "and a sphere (in inches): ";
    cin >> radius_of_both;

    area_of_circle = area(radius_of_both);
    volume_of_sphere = volume(radius_of_both);

    cout << "Radius = " << radius_of_both << " inches\n"
         << "Area of circle = " << area_of_circle
         << " square inches\n"
         << "Volume of sphere = " << volume_of_sphere
         << " cubic inches\n";

    return 0;
}
```

Display 4.12 (2/2)

A Global Named Constant (*part 2 of 2*)

```
double area(double radius)
{
    return (PI * pow(radius, 2));
}

double volume(double radius)
{
    return ((4.0/3.0) * PI * pow(radius, 3));
}
```

Sample Dialogue

```
Enter a radius to use for both a circle
and a sphere (in inches): 2
Radius = 2 inches
Area of circle = 12.5664 square inches
Volume of sphere = 33.5103 cubic inches
```

Display 4.13 (1/2)

Formal Parameter Used as a Local Variable (*part 1 of 2*)

```
//Law office billing program.
#include <iostream>
using namespace std;

const double RATE = 150.00; //Dollars per quarter hour.

double fee(int hours_worked, int minutes_worked);
//Returns the charges for hours_worked hours and
//minutes_worked minutes of legal services.

int main( )
{
    int hours, minutes;
    double bill;

    cout << "Welcome to the offices of\n"
         << "Dewey, Cheatham, and Howe.\n"
         << "The law office with a heart.\n"
         << "Enter the hours and minutes"
         << " of your consultation:\n";
    cin >> hours >> minutes;

    bill = fee(hours, minutes);

    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(2);
    cout << "For " << hours << " hours and " << minutes
         << " minutes, your bill is $" << bill << endl;

    return 0;
}

double fee(int hours_worked, int minutes_worked)
{
    int quarter_hours;

    minutes_worked = hours_worked*60 + minutes_worked;
    quarter_hours = minutes_worked/15;
    return (quarter_hours*RATE);
}
```

The value of minutes *is not changed by the call to* fee.

minutes_worked *is a local variable initialized to the value of* minutes.

Display 4.13 (2/2)

Formal Parameter Used as a Local Variable (*part 2 of 2*)

Sample Dialogue

```
Welcome to the offices of
Dewey, Cheatham, and Howe.
The law office with a heart.
Enter the hours and minutes of your consultation:
2 45
For 2 hours and 45 minutes, your bill is $1650.00
```

Display 4.14 (1/2)

Using Namespaces (*part 1 of 2*)

```
//Computes the area of a circle and the volume of a sphere.
//Uses the same radius for both calculations.
#include <iostream>
#include <cmath>

const double PI = 3.14159;

double area(double radius);
//Returns the area of a circle with the specified radius.

double volume(double radius);
//Returns the volume of a sphere with the specified radius.

int main( )
{
    using namespace std;

    double radius_of_both, area_of_circle, volume_of_sphere;

    cout << "Enter a radius to use for both a circle\n"
         << "and a sphere (in inches): ";
    cin >> radius_of_both;

    area_of_circle = area(radius_of_both);
    volume_of_sphere = volume(radius_of_both);

    cout << "Radius = " << radius_of_both << " inches\n"
         << "Area of circle = " << area_of_circle
         << " square inches\n"
         << "Volume of sphere = " << volume_of_sphere
         << " cubic inches\n";

    return 0;
}
```