Return Values

Today

- Parameter passing
- Return values
- Function prototype

Implementing functions

Example: Calculate the area of a circle

1) Pick a good descriptive name for the function

2) Give a type and name for each parameter

    There will be one parameter for each piece of information the function needs to do its job

3) Specify the type of the return value:

    double computeCircleArea(double radius);

4) Then write the body of the function, as statements enclosed in curly braces { … }

Implementing functions

Example: Calculate the area of a circle

Note: Useful comments at the top: description, parameters, return, algorithm

```
/*
      Computes the area of a circle
      @param radius -- the radius of the circle
      @return the area of the circle
*/
double computeCircleArea(int radius)
{
      const double PI = 3.14;
      double area = PI * radius * radius;
      return area;
}
```

Implementing functions

- How do you know your function works as intended??
  - You should always test the function
  - Write a main() function to do this
  - Let's test a couple different radii for our computeCircleArea function and see if it outputs the correct volumes

```
int main()
{
    double result1 = computeCircleArea(2);
    double result2 = computeCircleArea(10);
    cout << "A circle with a radius of 2 has area of " << result1 <<
    endl;
    cout << "A circle with a radius of 10 has area of " << result2
    << endl;
    return 0;
}
```

Parameter passing

Parameter Passing

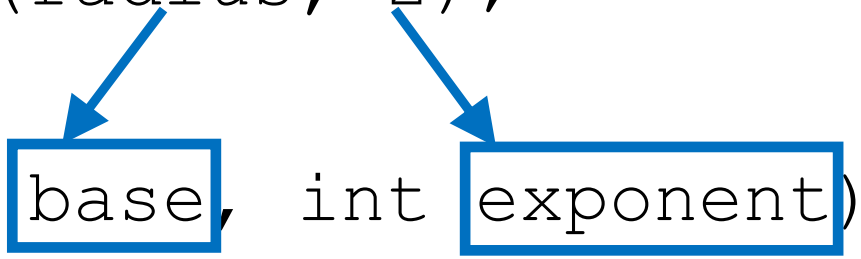
- When a function is called, a *parameter variable* is created for each value passed in.
- Each parameter variable is *initialized* with the corresponding parameter value from the call.

```
Area = 3.14 * ourPow(radius, 2);
…
double ourPow(double base, int exponent);
```

Parameter Passing

- When a function is called, a *parameter variable* is created for each value passed in.
- Each parameter variable is *initialized* with the corresponding parameter value from the call.

```
area = 3.14 * ourPow(radius, 2);
…
double ourPow(double base, int exponent);
```



Parameter Passing

- The Caller(computeCircleArea( ) function) calls the Callee(ourPow( ))
- When the **`ourPow( )`** function is called, the parameter variable **`base`** and **`exponent`** is created & initialized with the value that was passed in the function call.
- After the return statement, the local variables `base` and `exponent` disappear from memory.
- The calculated volume is stored in the variable, `area`

Return values

Return Values

The `return` statement ends the function execution. This behavior can be used to handle unusual cases.

What should we do if the side length is negative?

We choose to return a zero and not do any calculation:

```
double computeCircleArea(int radius)
{
     if (radius < 0)
                  return 1;
    const double PI = 3.14;
    double area = PI * radius * radius ;
    return area;
}
```

- Nothing is executed after a return statement !!!
- Execution returns to main()

Return Values: Shortcut

The **`return`** statement can return the value of any expression.

Instead of saving the return value in a variable and returning the variable, it is often possible to eliminate the variable and return a more complex expression:

```
double computeCircleArea(double radius)
{
   return 3.14 * radius * radius;
}
```

Common Error – Missing Return Value

Your non-void-function always needs to return something.

The code below: what is returned if the call passes in a negative value?

You need to ensure all paths of execution include a `return` statement.

```
double areaOfCircle(double radius)
{
  if (radius >= 0)
  {
     return 3.14 * radius * radius;
  }
}
```

Functions without return values: Void Function

- Consider the task of writing/printing a string with the following format around it
- Any string could be used
- For example, the string “Hello” would produce:

```
-------
!Hello!
-------
```

Functions without return values: Void Function

Definition: This kind of function is called a void function

- `void` is a type, just like `int` or `double`
- Use a return type of void to indicate that a function does not return a value
- `void` functions are used to simply perform a sequence of instructions, but not return any particular values to the caller
- Example: `void boxString( )`

Functions without return values – the *void* type

```
void box_string( )
{
      cout << "-------" << endl;
      cout << "!Hello!" << endl;
      cout << "-------" << endl;
}
```

- Note that this function doesn't compute any value.
- It performs some actions and then returns to the caller without returning a value
- There is no return statement

Calling void functions

- A void function has no return value, so we cannot call it with assignment like this:

```
result = boxString( );
// Error: boxString does not return a result
```

- Instead, we call it like this, without assignment:

```
boxString( );
```