

# Functions

# Today

---

- What are functions?
- Implementing functions
- Function parameters and arguments
- Return values

# Functions

# What is a function?

---

- A function
  - is a sequence of instructions with a name
  - packages a computation into a form that can be easily understood and reused
- example:

```
int main()  
{  
    double z = pow(2, 3);  
    ...  
}
```

# Functions as Black Box

---

- You can think of a function as a “black box”
  - Know what the box does, but can't see what's inside
  - Like a pressure cooker -- can't see inside, know what it does



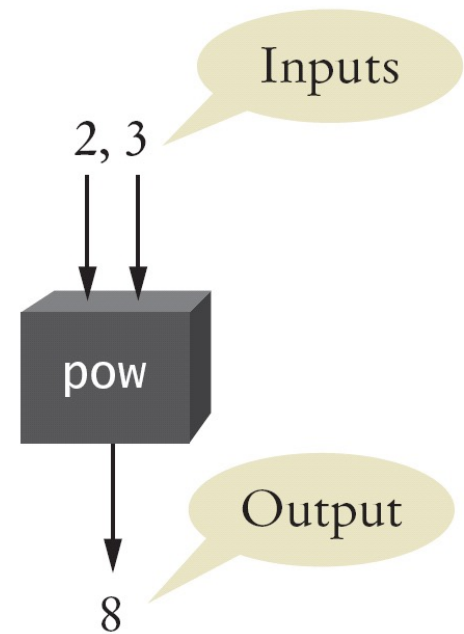
# Functions as Black Box

---

**Example:** How did the pow function do its job?

→ You didn't need to know in order to use it

→ You only need to know its specification (inputs/outputs, syntax)



# Calling a function

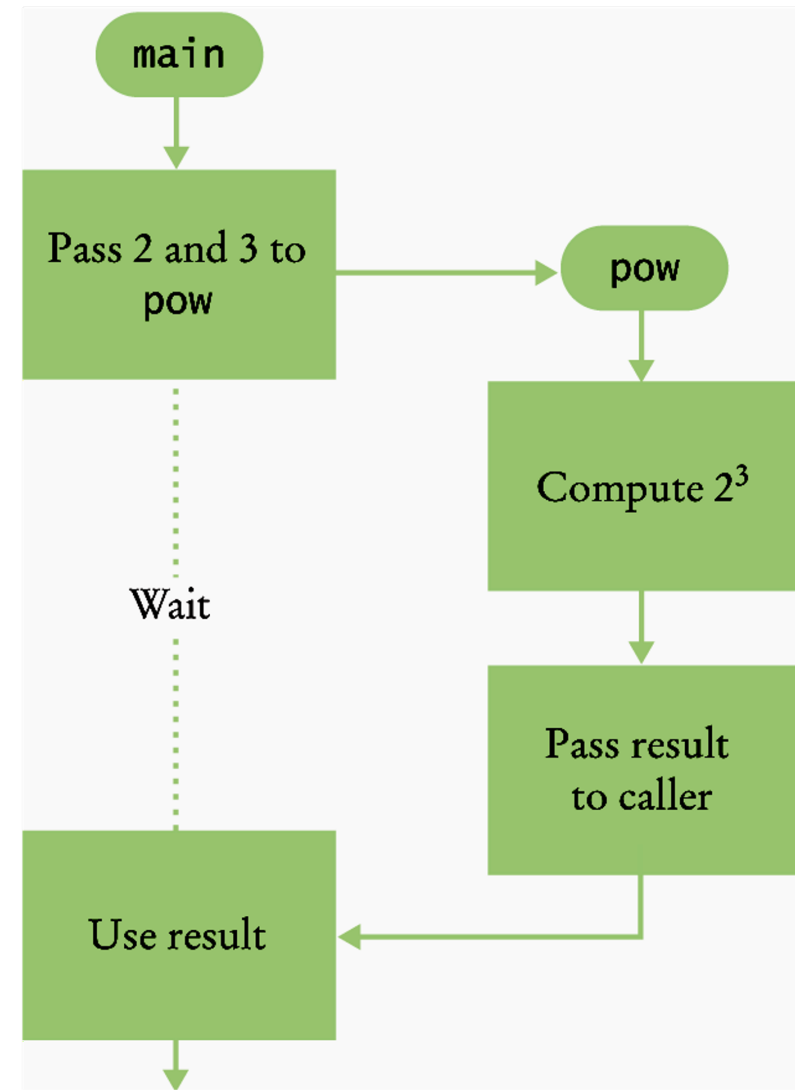
---

- main is a function, and so is pow
- main calls the pow function, asking it to compute  $2^3$
- The main function is temporarily suspended while pow does its thing
- The instructions of the pow function execute and compute the result
- The pow function returns its result back to main
- main resumes execution

```
int main()  
{  
    double z = pow(2, 3);  
    ...  
}
```

# Flowchart: Calling a function

Execution flow during a function call





# Actual parameters/arguments

---

- When another function calls the `pow` function, it provides inputs
  - (e.g., the 2 and 3 in the call `pow(2, 3)`)
- In order to avoid confusion with user-provided inputs (`cin >>` ), these values are called **function arguments**
- The output that the `pow` functions computes is called the return value
  - (as opposed to output using `cout <<` )

```
int main()  
{  
    double z = pow(2, 3);  
    ...  
}
```

# Parameters

---

**Note:** An output statement (cout) does not return a value and the return statement does not display output

- output  $\neq$  return
- return statement ends the called function and resumes execution of the program that called that function
  - Can also pass a value back to the calling program (e.g., return 0; )
- A cout << statement communicates only with the user running the program
  - Just spits things out to the screen. That's it.

# 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 areaOfCircle(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 areaOfCircle(double 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 values for our areaOfCircle function and see if it outputs the correct areas

```
int main()
{
    double result1 = areaOfCircle(2);
    double result2 = areaOfCircle(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;
}
```