Input Stream

1. Read in from the keyboard
2. Read in from file (future)

Input Stream

1. Read in from the keyboard
2. Read in from file (future)

Input Stream – from keyboard

```
string x;
cin >> x;
getline(cin, x);
```

The user types…	Value of x is…	Left on the stream is…
78 94 42		
901.23ab%!@h29ks		
The rain in Spain	The	
jk		

Functions

- Pre-defined functions:
  - *main() (only have one main function in each project)*

- *User-defined functions*

Declare Function

- Like variables, functions must be declared before they are used
  - That's why we've been putting them at the top of the file, before `main`
  - This simultaneously *declares* and *defines* the function
  - In practice, we like to put `main` first, so we separate the function *declaration* from the function *definition*

Function Prototypes

- Functions are ***declared*** with a prototype
  - Looks just like the function heading as a statement (with ;)

```
double pow( double x, double y );
```

- As long as the ***declaration*** is before the function is used, you can put the *definition* anywhere
  - The definition is unchanged, still has heading and body
  - Convention is to put all function declarations together, followed by all function definitions (with `main` first)

```
// declare a function with type (int)
int fun1();

// define a function
int fun1(){
        cout<<"hi";
}
```

```
// allocate memory for a variable
int x;

x = 10;
```

Prototypes and Organization

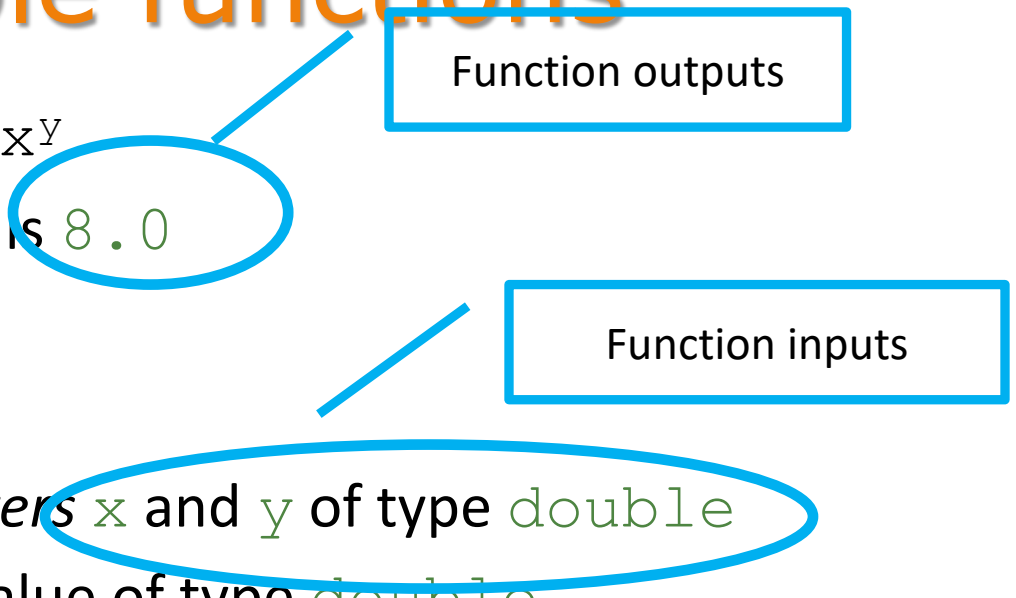
```
// declare a function
double get_side();

// define a function without input parameters
double get_side()
{
   double input;
   cout << "Please enter a side of the triangle: ";
   cin >> input;
   return input; // return value
}

int main()
{
   double x;
   // call
   x = get_side();
   return 0;
}
```

Function parameters and return values

- A *function* is a set of instructions
  - When *executed*, it accomplishes a task
- Most functions require *input parameters in a function*
  - Pieces of data that the function needs to do its job
  - This is **not the same** as stream input from the user/keyboard
- Many functions *output* a *return value*
  - A piece of data that is the result of that job
  - This is **not the same** as print out to the screen

Example functions

- `pow(x,y)` calculates x^y
  - `pow(2.0, 3.0)` is `8.0`



  - Input: two *parameters* `x` and `y` of type `double`
  - Output: returns a value of type `double`

Example functions

- `floor(x)` calculates the largest whole number less than `x`
  - `floor(48.79)` is `48.0`
  - Input: one parameter `x` of type `double`
  - Output: returns a value of type `double`

Predefined functions

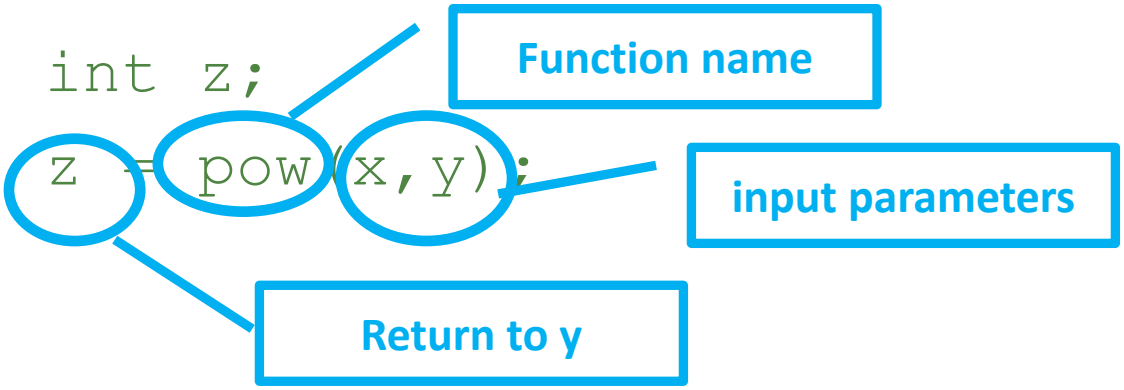
- Functions someone else wrote that you can use
- Predefined functions are organized into separate libraries
  - Stream I/O functions are in `iostream` library
  - Math functions are in `cmath` library
- Each library has a *header* file
- To use a predefined function, you `#include` the appropriate header file

Calling Functions

- Every function has 3 parts you need to know in order to use it (besides what it does, of course):
  - A name
    - Follows the same rules as variables names
    - Can't be the same as the name of a variable or a reserved word
  - A *parameter list*
    - These are the input values that the function needs in order to do its job
    - Each parameter is a specific data type (int, double, char, string, etc)
  - A *return type*
    - This is the data type that the function returns when it is done

Calling Functions

- Functions are *called* by name:



- When you call a function, you have to provide it with appropriate parameter values
  - Same number it expects to get
  - Same order
  - Same types

Using the Return Value

- The functions we've looked at all *return* a value
  - Sometimes we call this the *output* of the function
    - This is different than output to the screen!
  - Return values aren't printed to the screen, they are returned to the calling statement
    - You could also say the function evaluates to its return value
- Some examples:

```
x = 3 + 4;
```

evaluates to:

```
x = 7;
```

```
y = sqrt( 16.0 );
```

evaluates to:

```
y = 4;
```

Function Calls and Return Values

- When calling a function, you typically:
  - Save the return value for further calculation
  - Use the return value in some calculation
  - Print the return value

- In other words, functions are called:
  - In an assignment statement
  - In an expression
  - As an actual parameter to another function

Example Predefined Functions

TABLE 6-1 Predefined Functions

Function	Header File	Purpose	Parameter(s) Type	Result
`abs(x)`	`<cstdlib>`	Returns the absolute value of its argument: `abs(-7) = 7`	`int`	`int`
`ceil(x)`	`<cmath>`	Returns the smallest whole number that is not less than `x`: `ceil(56.34) = 57.0`	`double`	`double`
`cos(x)`	`<cmath>`	Returns the cosine of angle `x`: `cos(0.0) = 1.0`	`double` (radians)	`double`
`exp(x)`	`<cmath>`	Returns e^x, where `e = 2.718`: `exp(1.0) = 2.71828`	`double`	`double`
`fabs(x)`	`<cmath>`	Returns the absolute value of its argument: `fabs(-5.67) = 5.67`	`double`	`double`

Example Predefined Functions

TABLE 6-1 Predefined Functions (continued)

Function	Header File	Purpose	Parameter(s) Type	Result
`floor(x)`	`<cmath>`	Returns the largest whole number that is not greater than x: `floor(45.67) = 45.00`	`double`	`double`
`pow(x, y)`	`<cmath>`	Returns x^y; If `x` is negative, `y` must be a whole number: `pow(0.16, 0.5) = 0.4`	`double`	`double`
`tolower(x)`	`<cctype>`	Returns the lowercase value of `x` if `x` is uppercase; otherwise, returns `x`	`int`	`int`
`toupper(x)`	`<cctype>`	Returns the uppercase value of `x` if `x` is lowercase; otherwise, returns `x`	`int`	`int`

Exercises

1. Given three integer variables, `a`, `b` and `c`, write a C++ statement to find the greatest common denominator of `a` and `b`, and store it in `c`
   - How? Using a function!
   - The name of the function to compute the greatest common denominator is `gcd`
   - This function takes two integers as parameters
   - It returns a single integer

2. Given two string variables, `s1` and `s2`, write a C++ statement to assign s2 the characters in s1 in reverse order
   - How? Using a function!
   - The name of the function to reverse a string is `sreverse`
   - This function takes a string as its parameter
   - It returns a string

Exercises

3. Given three integer variables, `num1`, `num2` and `num3`, write a C++ statement to print the largest one
   - How? Using a function!
   - The name of the function to find the largest of three integers is `largest`
   - This function takes three integers as parameters
   - It returns a single integer

4. Given three integer variables, `num1`, `num2` and `num3`, write a C++ statement to print the largest one
   - This time using a different function
   - The name of the function to find the *larger* of *two* integers is `larger`
   - This function takes *two* integers as parameters
   - It returns a single integer

Side-effects

- Besides returning a value, functions can also have *side-effects*
  - For now, we'll focus on I/O related side-effects
- Examples:
  - Reading user input from a stream
  - Printing output to a stream
  - Drawing on the screen