CS135; User-Defined Functions

Gael Zarco

March 29, 2025

Functions are often called **modules** in C++.

1 Pre-Defined Functions

Examples of pre-defined functions include (each are of type `double`):

- `pow(x, y)` calculates x^y.
- `sqrt(x)` calculates nonnegative square root of x for $x >= 0$.
- `floor(x)` calculates the largest whole number that is $<= x$.

The x and y values in these functions are called **parameters** or **arguments**. A function's type is that of the returned value.

More Function Examples:

Function	Header File	Purpose	Parameter(s) Type	Result
`abs(x)`	`<cmath>`	Returns the absolute value of its argument: `abs(-7) = 7`	`int (double)`	`int (double)`
`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`

Function	Header File	Purpose	Parameter(s) Type	Result
`fabs(x)`	`<cmath>`	Returns the absolute value of its argument: `fabs(-5.67) = 5.67`	`double`	`double`
`floor(x)`	`<cmath>`	Returns the largest whole number that is not greater than `x`: `floor(45.67) = 45.00`	`double`	`double`
`islower(x)`	`<cctype>`	Returns `true` if `x` is a lowercase letter; otherwise it returns `false`; `islower('h')` is `true`	`int`	`int`
`isupper(x)`	`<cctype>`	Returns `true` if `x` is a uppercase letter; otherwise it returns `false`; `isupper('K')` is `true`	`int`	`int`
`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`
`sqrt(x)`	`<cmath>`	Returns the nonnegative square root of `x`, `x` must be nonnegative: `sqrt(4.0) = 2.0`	`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`

2 User-Defined Functions

User-defined functions in C++ are classified into two categories:

- **Value-Returning Functions** use `return` statement to return a value (has a data type).
- **Void-Returning Functions** do not have a return type; do not use `return` statement.

2.1 Value-Returning Functions

C++ functions such as pow, abs, islower, and toupper are examples of value-returning functions. Use these functions by including the *header file* in your

program and knowing the following:

1. The name of the function.
2. The *parameters* (if applicable)
3. The data type of each parameter.
4. The type of the function
5. The code required to accomplish the task (both value-returning and void).

To elaborate on *5*, the `abs` function may have the following definition:

Listing 1: `abs` Function

```
int abs(int number) {
  if (number < 0)
    number = -number;

  return number;
}
```

The function declared in the heading of the `abs` function defintion is known as the **Formal Parameter** (`number`). The heading within a function definition defines all formal parameters.

The **Actual Parameters** of a function are the variables passed into the function and copied into the formal parameters.

- **Formal Parameter**: A variable declared in the function heading.
- **Actual Parameter**: A variable or expression listed in a call to a function.

For example:

Listing 2: `pow` Function Header

```
  double pow(double base, double exponent)
```

In the code below, the values of `u` *and* `v` *are copied into* `base` *and* `exponent`*:*

Listing 3: `pow` Actual Parameter Example

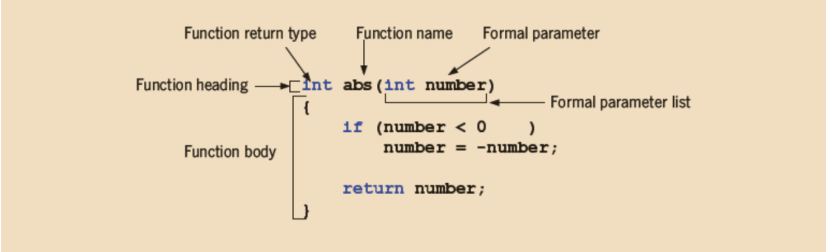
```
  double u = 2.5;
  double v = 3.0;
  double x, y;

  x = pow(u, v);
  y = pow(2.0, 3.2) + 5.1;
  cout << u << " to the power of 7 = " << pow(u, 7) << endl;
```

The values of `2.5` *and* `3.5` *are also copied into* `base` *and* `exponent`*; as are* `u` *and* `7`*.*

A value-returning function can be used:

- As an assignment statement.
- As a parameter in a function call.
- In an output statement.

Syntax Breakdown:



2.2 Formal Parameter List

Listing 4: Formal Parameter List Syntax

```
dataType identifier, dataType, identifier...
```

2.3 Function Call

Listing 5: Value-Returning Function Syntax & Example

```
functionName(actual parameter list)

// Example
x = abs (-5);
```

2.4 Actual Parameter List Syntax

Listing 6: Syntax For Actual Parameter List

```
expression or variable, expression or variable
```

A function's formal parameter list may be empty, however parentheses are still needed.

Listing 7: Empty Formal Parameter List Function Call

```
functionType functionName()
```

A value-returning function is also called an expression. Calling a function causes the function body to execute.

Function calls can be:

- part of an assignment or output statement.
- parameter in a function call.

2.5 `return` Statement

Functions return values using a `return` statement; passes the values outside the functions scope.

Listing 8: `return` Statement Syntax

```
return expr;
```

`expr` is a variable, constant value, or expression. `expr` is evaluated and its value is returned.

- The data type must match function type.

When a `return` statement executes in a function:

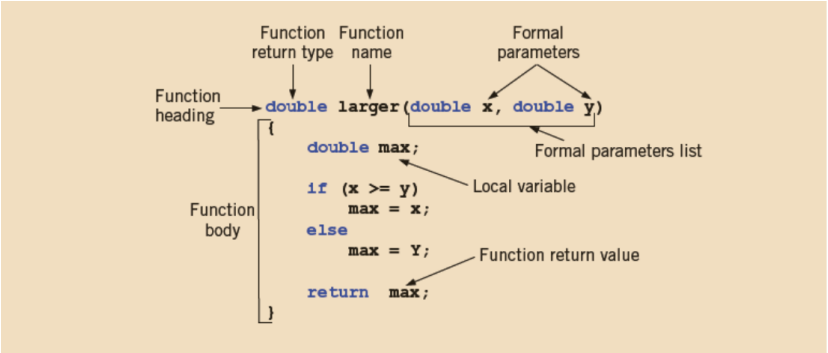
- The function is immediately terminated.
- Function call statement is replaced by the value of the `return` statement.
- Terminates the program if called within the `main` function.

Listing 9: Example program using `return`

```
double larger(double x, double y) {
  double max;

  if (x >= y)
    max = x;
  else
    max = y;

  return max;
}
```

The variable `max` is called a **Local Variable Declaration**, in which `max` is a variable local to the function `larger`.

Syntax breakdown:



2.6 Function Prototype

To work around the problem of undeclared identifiers regarding where to place function definitions in a program, place **Function Prototypes** before function definitions including `main`.

- Function prototype is *not* a definition.
- Gives the program the name of the function, number and data types of parameters, and data type of returned values.
- It is just enough information to let C++ use the function.
- Serves as a promise that the full definition will appear elsewhere in the program (will compile but *not* execute if missing defintion).

The **Function Prototype** is a function heading, terminated by a semicolon, `;`, without the function body.

Listing 10: Function Prototype Syntax

```
functionType functionName(parameter list);
```

3 Flow of Execution & Compilation

Execution always begins at the first statement in the `main` function.

- Other functions executed only when called.
- Function prototypes appear before any function definition (translated by compiler first).
- Function call transfers control to the first statement in the body of the called function.
- End of called function returns control to the point immediately before the function call (returned value replaces the function call statement).