Functions

2020 Spring: Introduction to C

July 1st, 2020

Today

- Functions
- Scope of Variables
- Parameterization
- Return Values
- Call by Value

Functions

- **function: A named group of statements**
  - Eliminates *redundancy* by code reuse
  - Procedural decomposition
    - Dividing a problem into functions
  - Writing a function is like adding a new command

- **Steps**
  1. Design the commands
  2. Declare (write down) the function
  3. Call (run) the function

Functions

- **Declare:** Give your function a name so it can be executed
- **Syntax**

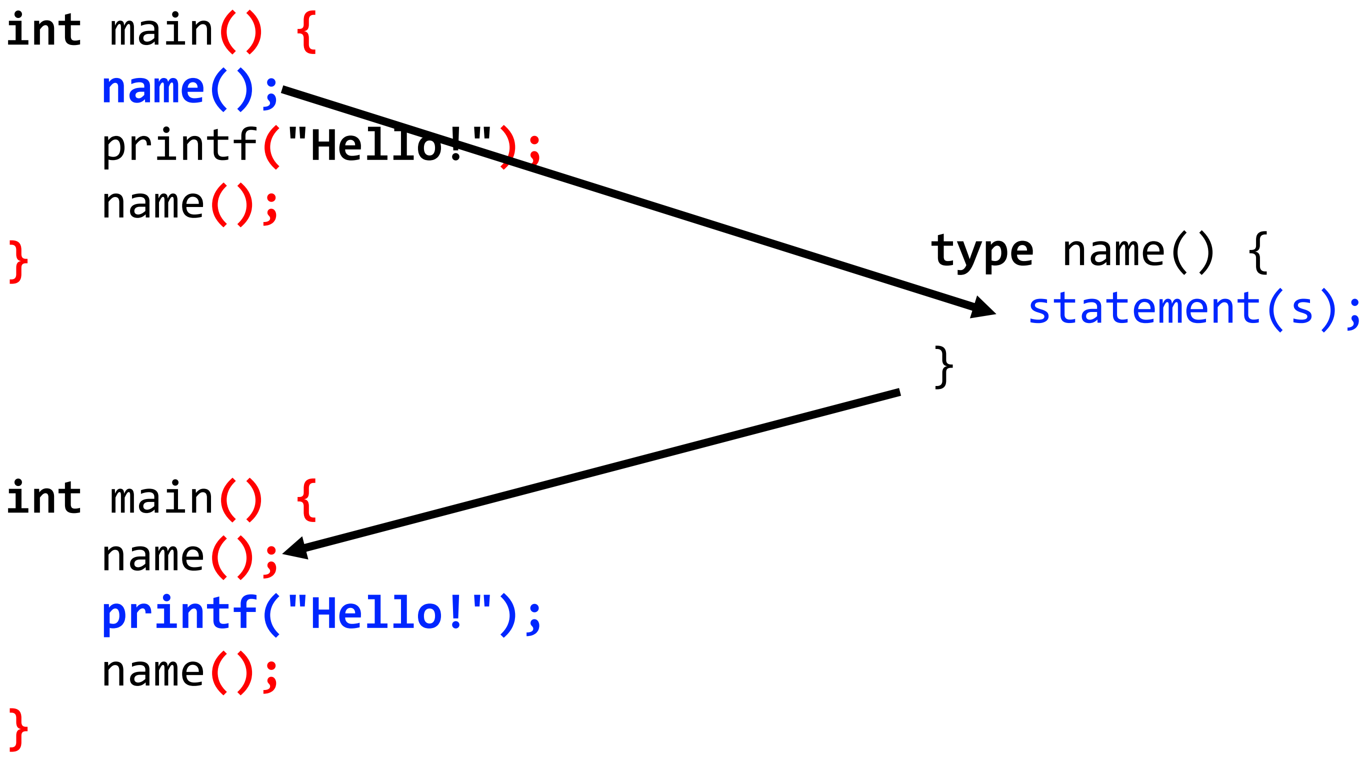
```
type name() {
    statement(s);
}
```

- **Call:** Execute the function's code by calling
  - You can call as many times as you want

```
int main() {
    name();
    printf("Hello!");
    name();
}
```

Control Flow

- **When a function is called, the program's execution**
  - "jumps" into that function
  - executes the function's statements
  - "jumps" back to the point where the function was called



```
int main() {
    name();
    printf("Hello!");
    name();
}
```

```
type name() {
    statement(s);
}
```

```
int main() {
    name();
    printf("Hello!");
    name();
}
```

Scope of Variables

- **scope: The part of a program where a variable exists**
  - Usually ***from its declaration to the end of the { } braces***
    - A variable declared in a `for` loop exists only in that loop
    - A variable declared in a function exists only in that function

- **Example**

```
void example() {
    int x = 3;
    for (int i = 1; i <= 10; ++i) {
        printf("%d\n", x);
    }
    // i no longer exists here
}
// x no longer exists here
```

Scope of Variables

- **Variables without overlapping scope can have same names**

```c
for (int i = 1; i <= 100; ++i) {
printf("/");
}
for (int i = 1; i <= 100; ++i) {
    printf("\\");
}
int i = 5;                    // OK: outside of loop
```

- **A variable can't be declared twice or used out of its scope**

```c
for (int i = 1; i <= 100; ++i) {
    int i = 2;                    // Error: duplicate local variable
    printf("/");
}
i = 4;              // Error: i cannot be resolved to a variable
```

Scope of Variables

- **You can use variables that can be accessed anywhere in the program**
  - *Global variable*: Its **scope** is the whole program

```c
int x = 5;

int main() {
    printf("%d", x);        // OK
}

void foo() {
    printf("%d", x + 1);    // OK
}
```

- Global variables will always be initialized to zero-equivalent values
- Generally not recommended – Best to keep scopes *small as possible*

Parameterization

- ***parameter*: A value passed to a function by its caller**
- **Syntax**
  - Declaration

```
type name(type param1) {
    statement(s);
}
```

  - Passing a parameter when calling the function
    - The value of expression must match the type in declaration

```
name(expression);
```

Parameterization – Multiple Parameters

- **Can pass multiple parameters to a function**

```c
void line(char c, int len) {
    for (int i = 0; i < len; ++i)
        printf("%c", c);
}

line('?', 5);    // prints ?????
```

- **Syntax**
  - Declaration

```
type name(type param1, ..., type paramn) {
    statement(s);
}
```

  - Passing parameters

```
name(expr1, ..., exprn);
```

Parameterization – Common Errors

- **If a function accepts a parameter, it is illegal to call it without passing any value for that parameter**

```
line();     // Error: parameter required
```

- **The value passed to a function must be of the correct type**

```
line('a', 3.2);       // Error: must be of type int
```

Exercise

- main 이외의 function을 사용할 것
- #2440 별 찍기 – 3
- #2442 별 찍기 – 5

Return

- **return: To send out a value as the result of a function**
  - Parameters send info ***in*** from the caller to the function
  - Return values send information ***out*** of a function to its caller
  - The function call will be evaluated to its return value

- **Syntax**
  - Specify the return type in declaration
    - **void** does not return anything
  - Function must return a value according to the type in declaration

```
type name(parameters) {
    statement(s);
    return expression;
}
```

Return Example

- **Absolute value function**

```
double abs(double x) {
    if (x >= 0)
        return x;
    else
        return -x;
}
```

  - `abs(1.2)` will be evaluated to 1.2, with type `double`
  - Can store the return value of the function by
    - `double y = abs(x);`
    - Now, this y can be used in other expressions

Call by Value

- **When a function is called:**
  - The value is stored into the parameter variable
  - The function's code executes, using that value (inside variable)

- **call by value: When values are passed as parameters, *their values are copied***
  - Modifying the parameter **will not affect the variable passed in**

```c
void strange(int x) {
    x = x + 1;
    printf("%d", x);    // 24
}

int main() {
    int x = 23;
    strange(x);
    printf("%d", x);    // 23
}
```

Call by Value

- **Example**

```
void swap(int x, int y) {
    int tmp = y;
    y = x;
    x = tmp;
    printf("%d, %d\n", x, y);   // 5, 10
}

int main() {
    int x = 10, y = 5;
    swap(x, y);
    printf("%d, %d\n", x, y);   // 10, 5 (not swapped)
}
```