Lecture 5: parameters; returns; math; if / else
Complex figure that scales

- Finish modifying the `mirror` code to use parameters so it can be scaled successfully.

A mirror of size 4:
```
#================#
|      <><>      |
|    <>....<>    |
|  <>........<>  |
|<>............<>|
|<>............<>|
|  <>........<>  |
|    <>....<>    |
|      <><>      |
#================#
```

A mirror of size 3:
```
#============#
|    <><>    |
|  <>....<>  |
|<>........<>|
|<>........<>|
|  <>....<>  |
|    <><>    |
#============#
```
Parameterization

• **parameter**: A value passed to a function by its caller.

• Instead of `line_of_4`, `line_of_3`, write `line` to draw any length.
  
  • When *declaring* the function, we will state that it requires a parameter for the number of equals.
  
  • When *calling* the function, we will specify how many equals signs to draw.

```plaintext
#================#
#============#
```

```plaintext
main

4

line

3

line

#=================#
#================#
```
Let's modify our loop table to use a user input size.

This can change the amount added in the loop expression.

<table>
<thead>
<tr>
<th>size</th>
<th>line</th>
<th>spaces</th>
<th>dots</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1,2,3,4</td>
<td>6,4,2,0</td>
<td>0,4,8,12</td>
</tr>
<tr>
<td>3</td>
<td>1,2,3</td>
<td>4,2,0</td>
<td>0,4,8</td>
</tr>
</tbody>
</table>

```
#================#
|      <><>      |      |    <><>    |
|    <>....<>    |      |  <>....<>  |
|  <>........<>  |      |<>........<>|
|<>............<>|      |<>........<>|
|<>............<>|      |  <>....<>  |
|  <>........<>  |      |    <><>    |
|    <>....<>    |      #============#
|      <><>      |
#================#
```
// Prints the expanding pattern of <> for the top half of the figure.
void top_half(int size) {
    for (int line = 1; line <= size; line++) {
        printf("|");
        for (int space = 1; space <= (line * -2 + (2*size)); space++) {
            printf(" ");
        }
        printf("<>");
        for (int dot = 1; dot <= (line * 4 - 4); dot++) {
            printf(".");
        }
        printf("<>");
        for (int space = 1; space <= (line * -2 + (2*size)); space++) {
            printf(" ");
        }
        printf("|
");
    }
}
Observations about scale factors

• The size can change the "intercept" in an expression.
  • Usually the "slope" is unchanged.

```c
void example(int size) {
    for (int space = 1; space <= (line * -2 + (2 * size)); space++) {
        printf(" ");
    }
    ...
}

• It doesn't replace every occurrence of the original value.

```c
for (int dot = 1; dot <= (line * 4 - 4); dot++) {
    printf(".");
}
Double border

- Alter the `mirror` code so that a double border can be drawn at the top and bottom.

- Does this new border row remind you of another part of our program?
  
  - How can we keep this new code from being redundant?
Multiple parameters

- A function can accept multiple parameters. (separate by , )
  - When calling it, you must pass values for each parameter.

- Declaration:
  ```
  void <name>(<type> <name>, ..., <type> <name>) {
      <statement>(s);
  }
  ```

- Call:
  ```
  <name>(<exp>, <exp>, ..., <exp>);
  ```
```c
void print_number(int number, int count) {
    for (int i = 1; i <= count; i++) {
        printf("%d", number);
    }
    printf("\n");
}

void main() {
    print_number(4, 9);
    print_number(17, 6);
    print_number(8, 0);
    print_number(0, 8);
}
```

Output:

```
444444444
171717171717
00000000
```
Strings

• **string**: A sequence of text characters.

  ```
  char[] <name> = "<text>";
  ```

• Examples:

  ```
  char[] name = "Marla Singer";
  char[] point = "(1, 2)";
  ```

• Note: You will sometimes see strings declared as `char*`
  • This will work too but is slightly different
  • More later on...
Strings as parameters

```c
void say_hello(char[] name) {
    printf("Welcome, %s\n", name);
}

int main() {
    say_hello("Merlin");
    char[] cat = "Sir Purrcival Cat";
    say_hello(cat);
}
```

Output:
Welcome, Merlin
Welcome, Sir Purrcival Cat
Value semantics

- **value semantics**: by default when variables (int, double) 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("1. x = %d\n", x);
}

int main() {
    int x = 23;
    strange(x);
    printf("2. x = %d\n", x);
    ...
}
```

Output:
1. x = 24
2. x = 23
void mystery(int x, int z, int y) {
    printf("%d and %d", z, (y - x));
}

int main() {
    int x = 9;
    int y = 2;
    int z = 5;

    mystery(z, y, x);

    mystery(y, x, z);
}
Math
C's math functions

- To access the following functions you must include
  ```c
  #include <math.h>
  ```

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sqrt(value)</code></td>
<td>square root</td>
</tr>
<tr>
<td><code>cbrt(value)</code></td>
<td>cube root</td>
</tr>
<tr>
<td><code>exp(value)</code></td>
<td>exponential function $e^{value}$</td>
</tr>
<tr>
<td><code>log(value)</code></td>
<td>natural logarithm, base $e$</td>
</tr>
<tr>
<td><code>log10(value)</code></td>
<td>logarithm, base 10</td>
</tr>
<tr>
<td><code>fabs(value)</code></td>
<td>absolute value as a floating point number</td>
</tr>
<tr>
<td><code>ceil(value)</code></td>
<td>rounds up to nearest larger whole number</td>
</tr>
<tr>
<td><code>floor(value)</code></td>
<td>rounds down to nearest smaller whole number</td>
</tr>
<tr>
<td><code>round(value)</code></td>
<td>rounds to the nearest whole number</td>
</tr>
<tr>
<td><code>pow(value1, value2)</code></td>
<td>value1 raised to the power of value2</td>
</tr>
<tr>
<td><code>fmod(value1, value2)</code></td>
<td>remainder of value1/value2 as a floating point number</td>
</tr>
<tr>
<td><code>sin(value)</code></td>
<td>sine/cosine/tangent of an angle in radians</td>
</tr>
<tr>
<td><code>cos(value)</code></td>
<td></td>
</tr>
<tr>
<td><code>tan(value)</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_E</td>
<td>2.7182818...</td>
</tr>
<tr>
<td>M_PI</td>
<td>3.1415926...</td>
</tr>
</tbody>
</table>
Math questions

- Evaluate the following expressions:
  - \( \text{fabs}(-1.23) \)
  - \( \text{pow}(3, 2) \)
  - \( \text{pow}(10, -2) \)
  - \( \sqrt{121.0} - \sqrt{256.0} \)
  - \( \text{round}(\text{M_PI}) + \text{round}(\text{M_E}) \)
  - \( \text{ceil}(6.022) + \text{floor}(15.9994) \)
  - \( \text{fabs}(-3, -5) \)
Quirks of real numbers

- Some math function return `double` or other non-int types.
  ```c
  pow(10, 3)  // 1000.0
  ```

- Some `double` values print poorly (too many digits).
  ```c
  double result = 1.0 / 3.0;
  printf("%f", result);  // 0.333333
  ```

- The computer represents `doubles` in an imprecise way.
Type casting

- **type cast**: A conversion from one type to another.
  - To promote an `int` into a `double` to get exact division from `/`
  - To truncate a `double` from a real number to an integer

**Syntax:**

```
(type) expression
```

**Examples:**

```
double result = (double) 19 / 5; // 3.8
int result2 = (int) result; // 3
int x = (int) pow(10, 3); // 1000
```
More about type casting

- Type casting has high precedence and only casts the item immediately next to it.
  - `double x = (double) 1 + 1 / 2;`  // 1.0
  - `double y = 1 + (double) 1 / 2;`  // 1.5

- You can use parentheses to force evaluation order.
  - `double average = (double) (a + b + c) / 3;`

- A conversion to `double` can be achieved in other ways.
  - `double average = 1.0 * (a + b + c) / 3;`
Why return and not print?

- It might seem more useful for the math functions to print their results rather than returning them. Why don't they?

- Answer: Returning is more flexible than printing.
  - We can compute several things before printing:
    ```c
    double pow1 = pow(3, 4);
    double pow2 = pow(10, 6);
    printf("Powers are %d and %d\n", pow1, pow2);
    
    double k = 13 * pow(3, 4) + 5 - sqrt(17.8);
    ```
Returning a value

def type name(parameters) {
  statements;
  ...
  return expression;
}

- When C reaches a return statement:
  - it evaluates the expression
  - it substitutes the return value in place of the call
  - it goes back to the caller and continues after the function call
Return examples

// Converts degrees Fahrenheit to Celsius.
double f_to_c(double degreesF) {
    double degreesC = 5.0 / 9.0 * (degreesF - 32);
    return degreesC;
}

// Computes triangle hypotenuse length given its side lengths.
double hypotenuse(int a, int b) {
    double c = sqrt(a * a + b * b);
    return c;
}

• You can shorten the examples by returning an expression:

    double fToC(double degreesF) {
      return 5.0 / 9.0 * (degreesF - 32);
    }
Common error: Not storing

- Many students incorrectly think that a `return` statement sends a variable's name back to the calling function.

```c
int main() {
    slope(0, 0, 6, 3);
    printf("The slope is %f\n", result); // ERROR:
    // cannot find symbol: result
}

double slope(int x1, int x2, int y1, int y2) {
    double dy = y2 - y1;
    double dx = x2 - x1;
    double result = dy / dx;
    return result;
}
```
Fixing the common error

• Returning sends the variable's value back. Store the returned value into a variable or use it in an expression.

```c
int main() {
    double s = slope(0, 0, 6, 3);
    printf("The slope is %f\n", s);
}

double slope(int x1, int x2, int y1, int y2) {
    double dy = y2 - y1;
    double dx = x2 - x1;
    double result = dy / dx;
    return result;
}
```
Exercise

- In physics, the displacement of a moving body represents its change in position over time while accelerating.
  - Given initial velocity \( v_0 \) in m/s, acceleration \( a \) in m/s\(^2\), and elapsed time \( t \) in s, the displacement of the body is:
    - Displacement = \( v_0 t + \frac{1}{2} a t^2 \)

- Write a function `displacement` that accepts \( v_0 \), \( a \), and \( t \) and computes and returns the change in position.
  - example: `displacement(3.0, 4.0, 5.0)` returns 65.0
Exercise solution

double displacement(double v0, double a, double t) {
    double d = v0 * t + 0.5 * a * pow(t, 2);
    return d;
}
The \texttt{if/else statement}
The **if** statement

Executes a block of statements only if a test is true

```c
if (test) {
    statement;
    ...
    statement;
}
```

- **Example:**
  ```c
double gpa;
scanf("%lf", &gpa);
if (gpa >= 2.0) {
    printf("Application accepted.");
}
```
The if/else statement

Executes one block if a test is true, another if false

if (test) {
    statement(s);
} else {
    statement(s);
}

• Example:
  
  double gpa;
  scanf("%lf", &gpa);
  if (gpa >= 2.0) {
      printf("Welcome to Mars University!");
  } else {
      printf("Application denied.");
  }  
  
  execute statement after if/else statement
Relational expressions

- *if* statements and *for* loops both use logical tests.

  ```java
  for (int i = 1; i <= 10; i++) { ... }
  if (i <= 10) { ... }
  ```

- These are *boolean* expressions

- Tests use *relational operators*:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>equals</td>
<td><code>1 + 1 == 2</code></td>
<td>true</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>does not equal</td>
<td><code>3.2 != 2.5</code></td>
<td>true</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>less than</td>
<td><code>10 &lt; 5</code></td>
<td>false</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than</td>
<td><code>10 &gt; 5</code></td>
<td>true</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less than or equal to</td>
<td><code>126 &lt;= 100</code></td>
<td>false</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>greater than or equal to</td>
<td><code>5.0 &gt;= 5.0</code></td>
<td>true</td>
</tr>
</tbody>
</table>
Nested if/else

Chooses between outcomes using many tests

```c
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else {
    statement(s);
}
```

- Example:

```c
if (x > 0) {
    printf("Positive");
} else if (x < 0) {
    printf("Negative");
} else {
    printf("Zero");
}
```
Nested if/else/if

- If it ends with `else`, exactly one path must be taken.
- If it ends with `if`, the code might not execute any path.

```java
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else if (test) {
    statement(s);
}
```

- Example:

```java
if (place == 1) {
    printf("Gold medal!"); // No code is executed if place is not 1
} else if (place == 2) {
    printf("Silver medal!"); // No code is executed if place is not 1 or 2
} else if (place == 3) {
    printf("Bronze medal."); // No code is executed if place is not 1, 2, or 3
}
```
Misuse of if

• What's wrong with the following code?

```c
printf("What percentage did you earn? ");
int percent;
scanf("%d", &percent);
if (percent >= 90) {
    printf("You got an A!\n");
}
if (percent >= 80) {
    printf("You got a B!\n");
}
if (percent >= 70) {
    printf("You got a C!\n");
}
if (percent >= 60) {
    printf("You got a D!\n");
}
if (percent < 60) {
    printf("You got an F!\n");
}
...```
Nested if structures

- **exactly 1 path** (mutually exclusive)
  ```java
  if (test) {
      statement(s);
  } else if (test) {
      statement(s);
  } else {
      statement(s);
  }
  ```

- **0 or 1 path** (mutually exclusive)
  ```java
  if (test) {
      statement(s);
  } else if (test) {
      statement(s);
  } else if (test) {
      statement(s);
  }
  ```

- **0, 1, or many paths** (independent tests; not exclusive)
  ```java
  if (test) {
      statement(s);
  } else if (test) {
      statement(s);
  } else if (test) {
      statement(s);
  }
  ```
Which nested if/else?

- (1) if/if/if   (2) nested if/else   (3) nested if/else/if
  - Whether a user is lower, middle, or upper-class based on income.
    - (2) nested if / else if / else
  - Whether you made the dean's list (GPA ≥ 3.8) or honor roll (3.5-3.8).
    - (3) nested if / else if
  - Whether a number is divisible by 2, 3, and/or 5.
    - (1) sequential if / if / if
  - Computing a grade of A, B, C, D, or F based on a percentage.
    - (2) nested if / else if / else if / else if / else
Nested if/else question

Write a program that produces output like the following:

This program reads data for two people and computes their basal metabolic rate and burn rate.

Enter next person's information:
height (in inches)? 73.5
weight (in pounds)? 230
age (in years)? 35
gender (enter 1 for male or 2 for female)? 1

Enter next person's information:
height (in inches)? 71
weight (in pounds)? 220.5
age (in years)? 20
gender (enter 1 for male or 2 for female)? 2

Person #1 basal metabolic rate = 2042.3
high resting burn rate
Person #2 basal metabolic rate = 1868.4
moderate resting burn rate

- Basal Metabolic Rate Formula:

  male BMR = 4.54545 x (weight in lb) + 15.875 x (height in inches) - 5 x (age in years) + 5

  female BMR = 4.54545 x (weight in lb) + 15.875 x (height in inches) - 5 x (age in years) - 161

<table>
<thead>
<tr>
<th>BMR</th>
<th>Burn Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>below 12000</td>
<td>low</td>
</tr>
<tr>
<td>1200 to 2000</td>
<td>moderate</td>
</tr>
<tr>
<td>above 2000</td>
<td>high</td>
</tr>
</tbody>
</table>
/ **Nested if/else**

// This program finds the basal metabolic rate (BMR) for two individuals.

// introduces the program to the user
void give_intro() { 
  printf("This program reads data for two\n");
  printf("people and computes their basal\n");
  printf("metabolic rate and burn rate.\n\n");
}
// this function contains the basal metabolic rate formula for
// converting the given height (in inches), weight
// (in pounds), age (in years) and gender (male or female) into a BMR
double bmr_for(double height, double weight, double age, int gender) {
  double bmr = 4.54545 * weight + 15.875 * height - 5 * age;
  if(gender == 1) {
    bmr += 5;
  } else {
    bmr -= 161;
  }
  return bmr;
}
// prompts for one person's statistics, returning the BMR
double get_bmr(int person) {
  double height;
  double weight;
  double age;
  int gender;
  printf("Enter person %d information: \n", person);
  printf("height (in inches)? ");
  scanf("%lf", &height);
  printf("weight (in pounds)? ");
  scanf("%lf", &weight);
  printf("age (in years)? ");
  scanf("%lf", &age);
  printf("gender (enter 1 for male or 2 for female)? ");
  scanf("%d", &gender);
  printf("\n");
  double bmr = bmr_for(height, weight, age, gender);
  return bmr;
}
Nested if/else, cont'd.

// reports the burn rate for the given BMR value
void report_status(double bmr) {
    if(bmr < 1200) {
        printf("low resting burn rate\n");
    } else if(bmr <= 2000) {
        printf("moderate resting burn rate\n");
    } else { // bmr > 2000
        printf("high resting burn rate\n");
    }
}

// reports the overall bmr values and status
void report_results(double bmr1, double bmr2) {
    printf("Person #1 basal metabolic rate = %.1f\n", bmr1);
    report_status(bmr1);
    printf("Person #2 basal metabolic rate = %.1f\n", bmr2);
    report_status(bmr2);
}

int main() {
    give_intro();
    double bmr1 = get_bmr(1);
    double bmr2 = get_bmr(2);
    //print(bmr1, bmr2);
    report_results(bmr1, bmr2);
}