Introduction to Haptics

Arduino Programming Language

(optional material for beginning programmers)

Allison M. Okamura
Stanford University
Programming Guidance

Potential resources:
• Online courses (e.g., EdX, Udacity)
• Web tutorials (Java or C programming languages are most appropriate)
• Arduino-specific tutorials

In this class:
• You will start from existing programs (sketches) and modify them
• The complexity of the programming you will do is low
• Debugging can be difficult because of the real-time nature of haptic interaction
• You should learn by doing. There is little you can do to damage your HapKit through programming mistakes!
We will start by going through some examples at

http://www.learn-c.org/
Arduino Programming Language Components

**Structure**
- Basic syntax
- Arithmetic operators
- Control structures
- Comparison Operators
- Boolean Operators

**Variables**
- Constants
- Data types
- Scope

**Functions**
- Digital I/O
- Analog I/O
- Math
- Serial communication
- Defining your own

Arduino reference materials obtained from http://arduino.cc under a Commons Attribution-ShareAlike 3.0 License.
Arduino Programming Language Components

Structure
- Basic syntax
- Arithmetic operators
- Control structures
- Comparison Operators
- Boolean Operators

Variables
- Constants
- Data types
- Scope

Functions
- Digital I/O
- Analog I/O
- Math
- Serial communication
- Defining your own

Arduino reference materials obtained from http://arduino.cc under a Commons Attribution-ShareAlike 3.0 License.

Stanford University
Introduction to Haptics
© Allison M. Okamura, 2014
Structure: Basic Syntax

; Each statement ends in a semicolon. For example: int a = 13;

{} Curly braces always come in pairs; they are used to define the start and end of functions, loops, and conditional statements. For example:
while (boolean expression)
{
    statement(s)
}

// Single line comment
/* */ Multi-line comment

#define Used to give a name to a constant value. For example: #define ledPin 3
Assignment operator stores the value to the right of the equal sign in the variable to the left of the equal sign: 

```c
sensorVal = analogRead(FSRPin);
```

Addition, subtraction, multiplication, and division. For example:

```c
result = value1 + value2;
result = value1 - value2;
result = value1 * value2;
result = value1 / value2;
```

where `value1` and `value2` are any variables or constants

Tips:
- Choose variable sizes that are large enough to hold the largest calculated result
- For math that requires fractions, use float variables (but there are drawbacks)
- Check for order of operations; use parentheses to enforce order
if

Tests whether a certain condition has been reached. Used in conjunction with a comparison operator. For example:

```cpp
if (someVariable > 50)
{
    // do something here
}
```

if...else

Allows you to do multiple tests. For example:

```cpp
if (force > 1)
{
    // action A
}
else
{
    // action B
}
```
for Creates a loop for repetitive operations.

for (initialization; condition; increment) {
    //statement(s);
}

for (int x = 0; x < 100; x++) {
    println(x);  // prints 0 to 99
}
switch case

Allows you to specify different code that should be executed in various conditions. For example:

```
switch (var) {
  case 1:
    //do something when var equals 1
    break;
  case 2:
    //do something when var equals 2
    break;
  default:
    // if nothing else matches, do the default
    // default is optional
}
```
The result of a statement with a comparison operator is either TRUE (1) or FALSE (2)

- \(x == y\) (\(x\) is equal to \(y\))
- \(x != y\) (\(x\) is not equal to \(y\))
- \(x < y\) (\(x\) is less than \(y\))
- \(x > y\) (\(x\) is greater than \(y\))
- \(x <= y\) (\(x\) is less than or equal to \(y\))
- \(x >= y\) (\(x\) is greater than or equal to \(y\))

Tips:
- Be careful not to accidentally use the assignment operator = instead of ==.
- Cannot use statements such as \(0 < x < 1\); need to do each comparison separately.
Logical AND. True only if both operands are true, e.g.

```cpp
if (digitalRead(2) == HIGH && digitalRead(3) == HIGH) {
    // do this only if both inputs are high
}
```

Logical OR. True if either operand is true, e.g.

```cpp
if (x > 0 || y > 0) {
    // do this if either x or y is greater than 0
}
```

NOT. True if the operand is false, e.g.

```cpp
if (!x) {
    // do this if x is false (0)
}
```
Arduino Programming Language Components

Structure
- Basic syntax
- Arithmetic operators
- Control structures
- Comparison Operators
- Boolean Operators

Variables
- Constants
- Data types
- Scope

Functions
- Digital I/O
- Analog I/O
- Math
- Serial communication
- Defining your own

Arduino reference materials obtained from http://arduino.cc under a Commons Attribution-ShareAlike 3.0 License.
Variables: Constants

**HIGH** When reading or writing to a digital pin, there are only two possible values a pin can take (or be set to): HIGH and LOW

**LOW**

**true** Logical levels (result of a comparison):
false is defined as 0
true is defined as 1 (but more broadly, anything but 0)

In addition, integer and floating-point constants can be used:

<table>
<thead>
<tr>
<th>Decimal integers</th>
<th>Floating point</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10.0</td>
</tr>
<tr>
<td>101</td>
<td>2.34E5</td>
</tr>
<tr>
<td>3200</td>
<td>67e-12</td>
</tr>
</tbody>
</table>

Arduino reference materials obtained from http://arduino.cc under a Commons Attribution-ShareAlike 3.0 License.
void  Used in function declarations to indicate that the function returns no information. For example:

```c
void setup() {
    // ...
}
```

```c
void loop() {
    // ...
}
```

boolean  A boolean holds one of two values, true or false. For example:

```c
boolean running = false;
if (running) {
    // do something
}
```
Variables: Data types

**char**
A data type that stores a character value. For example:

```c
char myChar = 'A';
char myChar = 65;  // both are equivalent
```

Coding is in this ASCII chart: http://arduino.cc/en/Reference/ASCIIchart

**float**
Datatype for floating-point numbers, a number that has a decimal point.

Floating-point numbers are often used to approximate analog and continuous values because they have greater resolution than integers. Floats have 6-7 decimal digits of precision. On the Hapkit board, `double` is the same as `float.`
Variables: Scope

Global vs. Local:
- A **global** variable is one that can be seen by every function in a program. Define it outside a function.
- A **local** variable is only visible to the function in which it is declared. Define it inside a function.
- For complex programs, local variables can help prevent programming errors. However, global variables are an easy way to share information across functions.

The **static** keyword is used to create variables that are visible to only one function. However unlike local variables that get created and destroyed every time a function is called, static variables persist beyond the function call, preserving their data between function calls. For example: `static int a;`
## Arduino Programming Language Components

### Structure
- Basic syntax
- Arithmetic operators
- Control structures
- Comparison Operators
- Boolean Operators

### Variables
- Constants
- Data types
- Scope

### Functions
- Digital I/O
- Analog I/O
- Math
- Serial communication
- Defining your own

Arduino reference materials obtained from http://arduino.cc under a Commons Attribution-ShareAlike 3.0 License.

Introduction to Haptics

© Allison M. Okamura, 2014
Functions: Digital I/O

For a description of the roles of different pins on the Arduino/Hapkit, see http://arduino.cc/en/Tutorial/DigitalPins

`pinMode(pin, mode)` Configures the specified pin to behave either as an input or an output. `pin` is the pin number.

`digitalWrite(pin, value)` Write a HIGH or a LOW value to a digital pin.

`digitalRead(pin)` Reads the value from a specified digital pin. The result will be either HIGH or LOW.
Functions: Analog I/O

analogReference(type)  The default reference voltage is 5V. This can be changed to a different type and different resolution using this function.

analogRead(pin)  Reads the value from the specified analog pin and returns a value between 0 and 1023 to represent a voltage between 0 and 5 volts (for default). It takes about 0.0001 seconds to read an analog pin.

analogWrite(pin, value)  Writes an analog value (PWM wave) to a pin. value is the duty cycle: between 0 (always off) and 255 (always on). Works on pins 3, 5, 6, 9, 10, and 11.
Functions: Math

min(x, y) Calculates the minimum of two numbers
max(x, y) Calculates the maximum of two numbers
abs(x) Computes the absolute value of a number
pow(base, exponent) Calculates the value of a number raised to a power
sqrt(x) Calculates the square root of a number
map(value, fromLow, fromHigh, toLow, toHigh) Re-maps a number from one range to another. That is, a value of fromLow would get mapped to toLow, a value of fromHigh to toHigh, values in between to values in between.

Trigonometric functions such as sin, cos, and tan are also available.

Arduino reference materials obtained from http://arduino.cc under a Commons Attribution-ShareAlike 3.0 License.

Stanford University

Introduction to Haptics

© Allison M. Okamura, 2014
Functions: Serial communication

Typically used for communication between an Arduino board and a computer via the USB port. Use the **serial monitor** to communicate with the board.

```
Serial.begin(9600);  // Used to begin serial communications, typically at a 9600 baud rate (bits per second)
Serial.print(val, format);  // Prints data to the serial port as human-readable ASCII text. Examples:
Serial.print(78) gives "78"
Serial.print(1.23456) gives "1.23"
Serial.println(1.23456, 4) gives "1.2346"
Serial.print("Hello world.") gives "Hello world."
Serial.println(val);  // Prints val followed by carriage return
```

Arduino reference materials obtained from http://arduino.cc under a Commons Attribution-ShareAlike 3.0 License.

© Allison M. Okamura, 2014
Many other functions have been created by Arduino users; some are posted at http://playground.arduino.cc/Main/GeneralCodeLibrary

You can also define your own function. This could be used to make your code more organized and efficient.

```java
int find_text(String needle, String haystack) {
    int foundpos = -1;
    for (int i = 0; (i < haystack.length() - needle.length()); i++) {
        if (haystack.substring(i,needle.length()+i) == needle) {
            foundpos = i;
        }
    }
    return foundpos;
}
```

This is a function that searches for a given string within another string. If the search string is found its position is returned, otherwise -1 is returned.