Introduction to C Programming

MIT 6.270 2012

What does a C program look like?

```
#include <joyos.h>

int usetup (void) {
        return 0;
}

int umain (void) {

        // Your code here...

        return 0;
}
```

What does a C program look like?

```
#include <joyos.h>

int usetup (void) {
        return 0;
}

int umain (void) {

        // Your code here...

        return 0;
}
```

Statements end with a semicolon

This is a comment. Doesn't get turned into machine instructions.

What do you do with the code?

- Edit it... you write code
  - Programmer's Notepad, vim, emacs...
- Compile it... turn it into processor instructions.
- Upload the code to the HappyBoard.
- Run and interact.

Simple Program

```
int umain() {
  // turn on motor 0, speed 200
  motor_set_vel(0, 200);

  // wait 3 seconds
  pause(3000);

  // turn off motor 0
  motor_set_vel(0, 0);
}
```

Variables

```
int umain (void) {

        uint8_t x = 5;
        uint8_t y = 15;
        uint8_t z = 25;

        z = x + y;

        x = 10;

        return 0;
}
```

How we store data on the microcontroller.

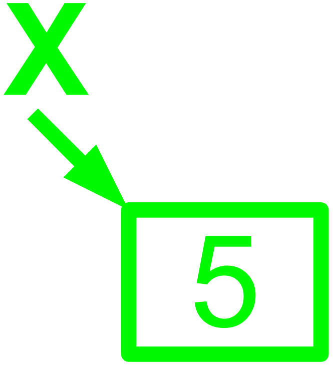
Data might come from sensors, user, communication link...

Variables give a name all of this data so we manipulate them.

Variables

```
int umain (void) {

        uint8_t x = 5;
        uint8_t y = 15;
        uint8_t z = 25;

        z = x + y;

        x = 10;

        return 0;
}
```

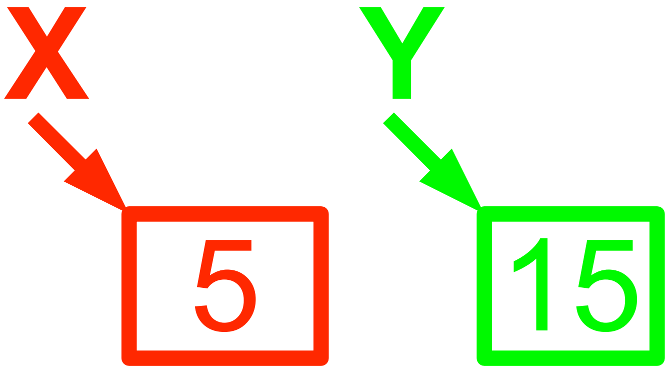
Here we assign names to a bunch of constants:



Variables

```
int umain (void) {

        uint8_t x = 5;
        uint8_t y = 15;
        uint8_t z = 25;

        z = x + y;

        x = 10;

        return 0;
}
```

Here we assign names to a bunch of constants:



X

5

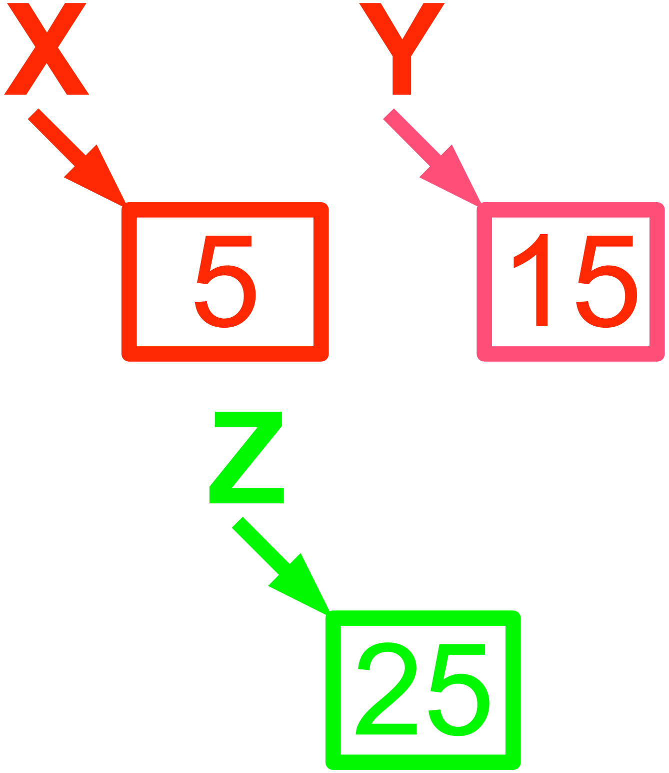
Y

15

Variables

```
int umain (void) {

        uint8_t x = 5;
        uint8_t y = 15;
        uint8_t z = 25;

        z = x + y;

        x = 10;

        return 0;
}
```

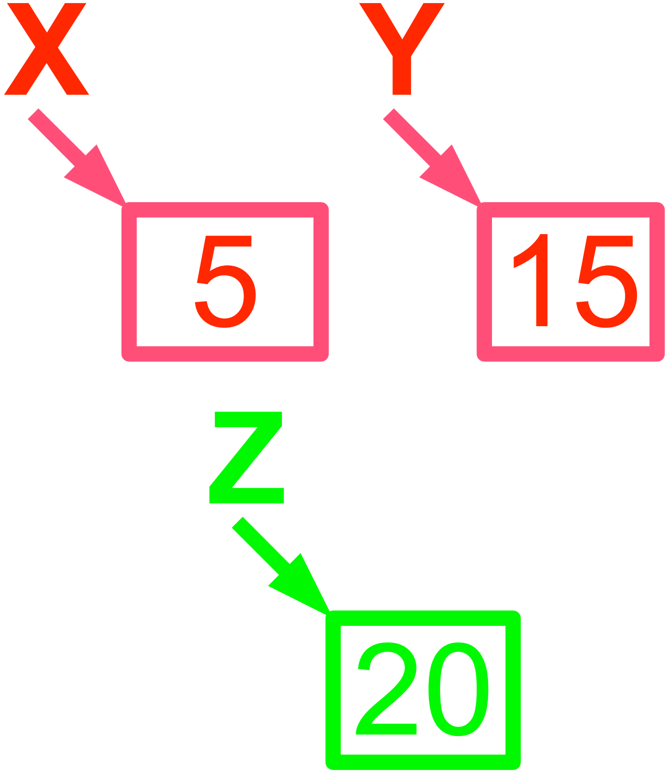
Here we assign names to a bunch of constants:



Variables

```
int umain (void) {

        uint8_t x = 5;
        uint8_t y = 15;
        uint8_t z = 25;

        z = x + y;

        x = 10;

        return 0;
}
```

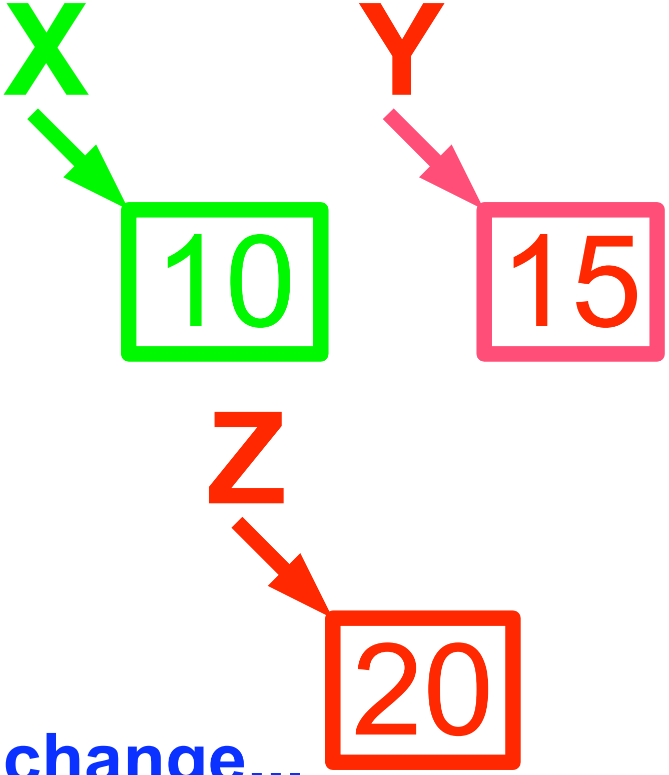
And then redefine Z to be equal to the sum of X and Y



Variables

```
int umain (void) {

        uint8_t x = 5;
        uint8_t y = 15;
        uint8_t z = 25;

        z = x + y;

        x = 10;

        return 0;
}
```

Instructions are executed as they are encountered.



Z doesn't change...

Variables

```
int umain (void) {

        uint8_t x = 5;
        uint8_t y = 15;
        uint8_t z = 25;

        z = x + y;

        x = 10;

        return 0;
}
```

Have to specify what kind of data a variable will hold the first time it's used.

Reserves a spot in memory.

“uint8_t”
unsigned **(positive)**
integer
8-bit **0 <= x <= 255**

Integers

- Unsigned:
  - uint8_t 0 <= x <= 255
  - uint16_t 0 <= x <= 65,535
  - uint32_t 0 <= x <= 4,294,967,295
- Signed:
  - int8_t -128 <= x <= 127
  - int16_t -32,768 <= x <= -32,767
  - int32_t −2,147,483,648 <= x <= 2,147,483,647

Real Numbers

- Float (32-bit)
  - float 1 E -38 <= x <= 3 E +38
  - About 7 significant figures
  - (same as double on the AVR)
- Examples
  - `float x = 1.618;`
  - `float y = -6.022e23;`
  - `float z = 1.6e-19;`

Basic output

printf writes to the USB serial port.
View this message using Termite on
windows or screen on Linux/Mac

```
int umain() {
    printf(“Hello world!\n”);
    uint8_t x = 42;
    printf(“Here's a number: %d\n”, x);
    return 0;
}
```

Basic output

Need to add newline character after each message.

```
int umain() {

    printf(“Hello world!\n”);

    uint8_t x = 42;

    printf(“Here's a number: %d\n”, x);

    return 0;
}
```

Basic output

```
int umain() {
    printf(“Hello world!\n”);
    uint8_t x = 42;
    printf(“Here's a number: %d\n”, x);
    return 0;
}
```

Special formatters, like %d, are replaced by the value of variables before being sent to the computer.

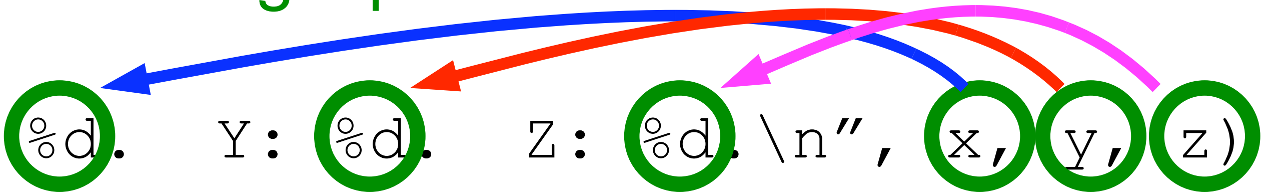
More printf

```
int umain() {

    x = 5;
    y = 15;
    Z = 25;

    z = x + y;

    x = 10;

    printf(“X: %d.  Y: %d.  Z: %d.\n”, x, y, z);

    return 0;
}
```

More printf

```
int umain() {

    x = 5;
    y = 15;
    Z = 25;

    z = x + y;

    x = 10;

    printf(“X: %d.  Y: %d.  Z: %d.\n”, x, y, z);

    return 0;
}
```

You can output multiple variables in a single printf

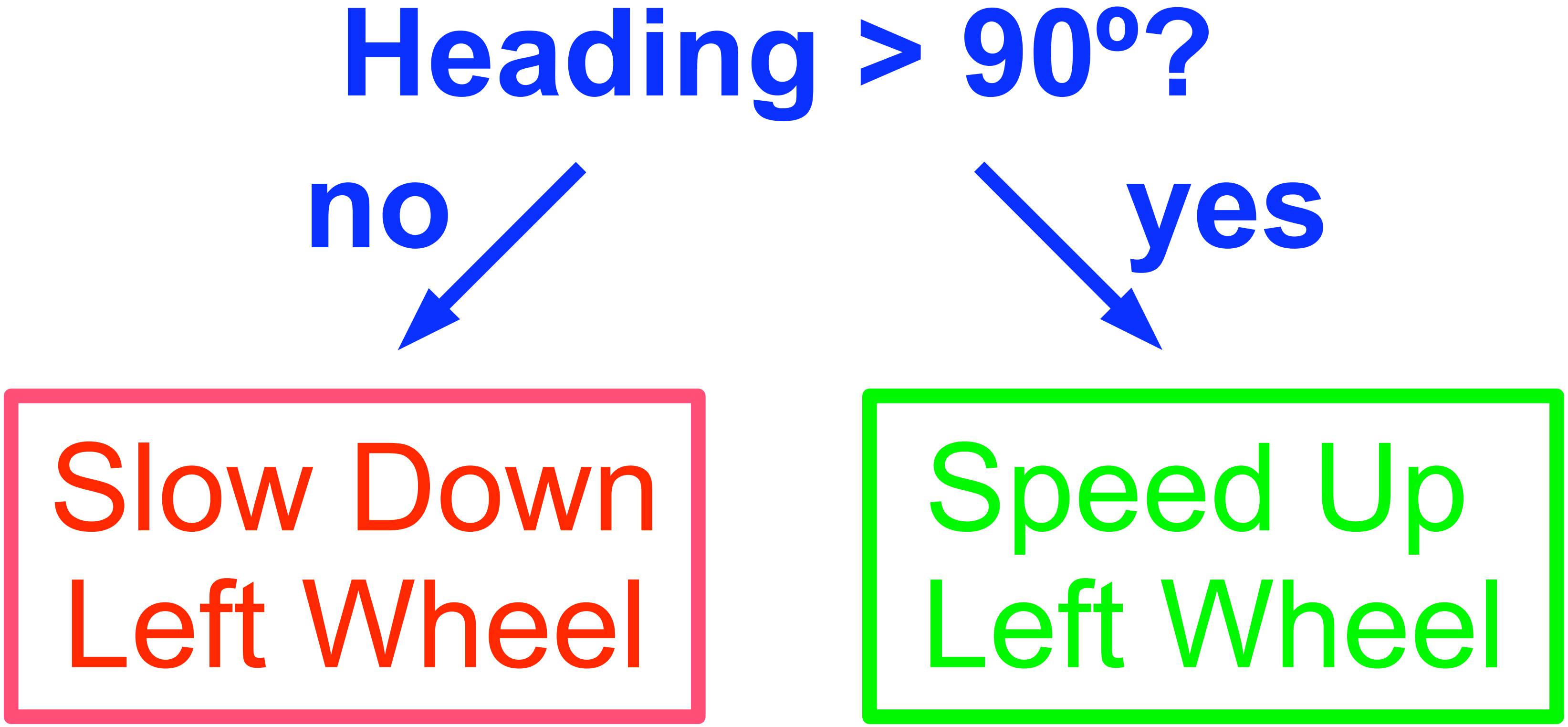


printf formatters

- printf("This is a variable: %d\n", x);
- %d - signed integer
- %03d - signed integer, padded with 0's to take up 3 digits (i.e. 003)
- %u - unsigned integer
- %f - floating point number
- %.2f - floating point number, to 2 decimal places
- %x - hex number
- \n - new line
- \t - tab

Conditionals

- Making decisions based on data.



Heading > 90º?

no

yes

Slow Down Left Wheel

Speed Up Left Wheel

Conditionals

```
int umain (void) {

        // ...

        if (heading > 90.0){
                left_weel_vel++;
        }
        else {
                left_wheel_vel--;
        }

        // ...
}
```

Conditionals

```
int umain (void) {

        // ...

        if (heading > 90.0){
                left_weel_vel++;
        }
        else {
                left_wheel_vel--;
        }

        // ...
}
```

```
left_wheel_vel = left_wheel_vel+1;
```

```
left_wheel_vel = left_wheel_vel-1;
```

Conditionals

```
int umain (void) {

        // ...

        if (heading > 90.0) {
                left_weel_vel++;
        }
        else {
                left_wheel_vel--;
        }

        // ...
}
```

Note that both actions are enclosed in curly braces

Conditionals

```
// ...

if (heading > 135.0){
        uart_printf(“Whoa.\n”);
        left_weel_vel += 2;
} else if (heading > 90.0){
        left_wheel_vel++;
} else {
        left_wheel_vel--;
}

// ...
```

Conditionals

```
// ...

if (heading > 135.0){
    uart_printf(“Whoa.\n”);
    left_wheel_vel += 2;
} else if (heading > 90.0){
    left_wheel_vel++;
} else {
    left_wheel_vel--;
}

// ...
```

```
left_wheel_vel = left_wheel_vel+2;
```

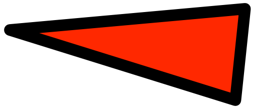
C is insensitive to whitespace

Conditionals

```
// ...

if (heading > 135.0){
        uart_printf(“Whoa.\n”);
        left_weel_vel += 2;
} else if (heading > 90.0){
        left_wheel_vel++;
} else {
        left_wheel_vel--;
}

// ...
```

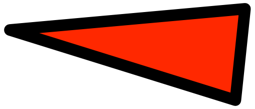
Conditionals

```
// ...

if (heading > 135.0){
        uart_printf(“Whoa.\n”);
        left_weel_vel += 2;
} else if (heading > 90.0){
        left_wheel_vel++;
} else {
        left_wheel_vel--;
}

// ...
```



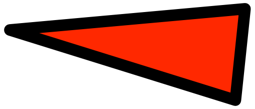
Conditionals

```
// ...

if (heading > 135.0){
        uart_printf(“Whoa.\n”);
        left_weel_vel += 2;
} else if (heading > 90.0){
        left_wheel_vel++;
} else {
        left_wheel_vel--;
}

// ...
```



Conditionals

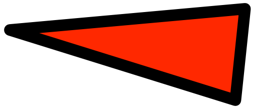
```
// ...

if (heading > 135.0){
        uart_printf(“Whoa.\n”);
        left_weel_vel += 2;
} else if (heading > 90.0){
        left_wheel_vel++;
} else {
        left_wheel_vel--;
}

// ...
```



Whoa.

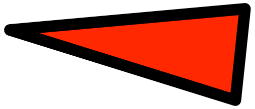
Conditionals

```
// ...

if (heading > 135.0){
        uart_printf(“Whoa.\n”);
        left_weel_vel += 2;
} else if (heading > 90.0){
        left_wheel_vel++;
} else {
        left_wheel_vel--;
}

// ...
```



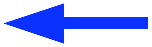
Conditionals

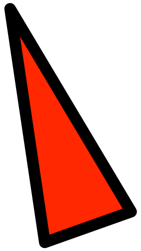
```
// ...

if (heading > 135.0){
        uart_printf(“Whoa.\n”);
        left_weel_vel += 2;
} else if (heading > 90.0){
        left_wheel_vel++;
} else {
        left_wheel_vel--;
}

// ...
```



Conditionals

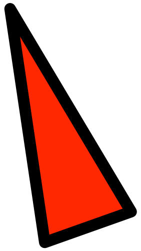
```
// ...

if (heading > 135.0){
        uart_printf(“Whoa.\n”);
        left_weel_vel += 2;
} else if (heading > 90.0){
        left_wheel_vel++;
} else {
        left_wheel_vel--;
}

// ...
```





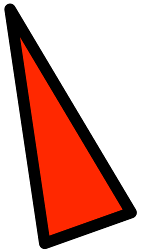
Conditionals

```
// ...

if (heading > 135.0){
        uart_printf(“Whoa.\n”);
        left_weel_vel += 2;
} else if (heading > 90.0){
        left_wheel_vel++;
} else {
        left_wheel_vel--;
}

// ...
```



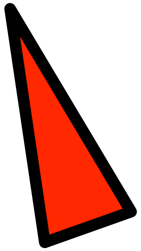
Conditionals

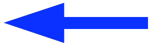
```
// ...

if (heading > 135.0){
        uart_printf(“Whoa.\n”);
        left_weel_vel += 2;
} else if (heading > 90.0){
        left_wheel_vel++;
} else {
        left_wheel_vel--;
}

// ...
```



Conditionals

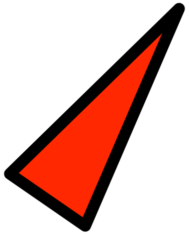
```
// ...

if (heading > 135.0){
        uart_printf(“Whoa.\n”);
        left_weel_vel += 2;
} else if (heading > 90.0){
        left_wheel_vel++;
} else {
        left_wheel_vel--;
}

// ...
```





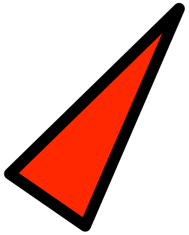
Conditionals

```
// ...

if (heading > 135.0){
        uart_printf(“Whoa.\n”);
        left_weel_vel += 2;
} else if (heading > 90.0){
        left_wheel_vel++;
} else {
        left_wheel_vel--;
}

// ...
```



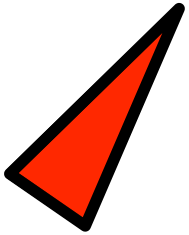
Conditionals

```
// ...

if (heading > 135.0){
        uart_printf(“Whoa.\n”);
        left_weel_vel += 2;
} else if (heading > 90.0){          ←
        left_wheel_vel++;
} else {
        left_wheel_vel--;
}

// ...
```



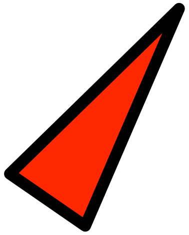
Conditionals

```
// ...

if (heading > 135.0){
        uart_printf(“Whoa.\n”);
        left_weel_vel += 2;
} else if (heading > 90.0){
        left_wheel_vel++;
} else {
        left_wheel_vel--;
}

// ...
```



Conditionals

```
// ...

if (heading > 135.0){
        uart_printf(“Whoa.\n”);
        left_weel_vel += 2;
} else if (heading > 90.0){
        left_wheel_vel++;
} else {
        left_wheel_vel--;
}

// ...
```



Conditionals

```
// ...

if (heading > 135.0){
        uart_printf(“Whoa.\n”);
}

// ...
```

Legal. No “else” or “else if” required.

Conditionals

- comparators

==	equals
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal
!=	not equal

- boolean operators
  - || OR (true if either operand is true)
  - && AND (true if both operand are true)
  - ! NOT (negates operand)

if (x >= 6 && x < 10)   - if x is between 6 and 10...

Loops - while

- Repeat code multiple times
- “while” loops run as long as the condition is true

```
while (condition) {
    do_something();
}
```

```
while (1) {      // loop forever
    int i = frob_read_range(0, 100);
    printf("The frob is at: %d\n", i);
    pause(200);
}
```

Conditional & Loop Full Example

```
int usetup() {
  gyro_init(11, 1400000, 1000);
  return 0;
}

int umain() {
  while (1) {
    float deg = gyro_get_degrees();
    if (deg < 0) {
      motor_set_vel(0,40);
      motor_set_vel(1,90);
    } else {
      motor_set_vel(0,90);
      motor_set_vel(1,40);
    }
  }
  return 0;
}
```

Loops - for

- Repeat code *n* times

```
for (initializer; condition to continue; step operation) {
    do_something();
}
```

```
int i;

for (i = 1; i <= 10; i++) {
   printf("%d \n", i);
}
```

Another example – ball dispenser

```
uint8_t last_lever = false;
while(1) {
     uint8_t cur_lever = (analog_read(8) < 500);

     if (cur_lever && !last_lever) {
         servo_set_pos(0, 341);
         pause(300);

         servo_set_pos(0, 220);
         pause(400);
     }

     last_lever = cur_lever;
}
```

Functions

```
// ...

float d2, d;

d2 = (myX-mouseX)*(myX-mouseX) +
     (myY-mouseY)*(myY-mouseY);

d = sqrt(d2);

if (d < 5.0){
        stop();
}

// ...
```

Functions

```
// ...

float d2, d;

d2 = (myX-mouseX)*(myX-mouseX) +
     (myY-mouseY)*(myY-mouseY);

d = sqrt(d2);

if (d < 10.0){  // mouse within 10cm?
        stop();
}

// ...
```

You can declare multiple variables at once.

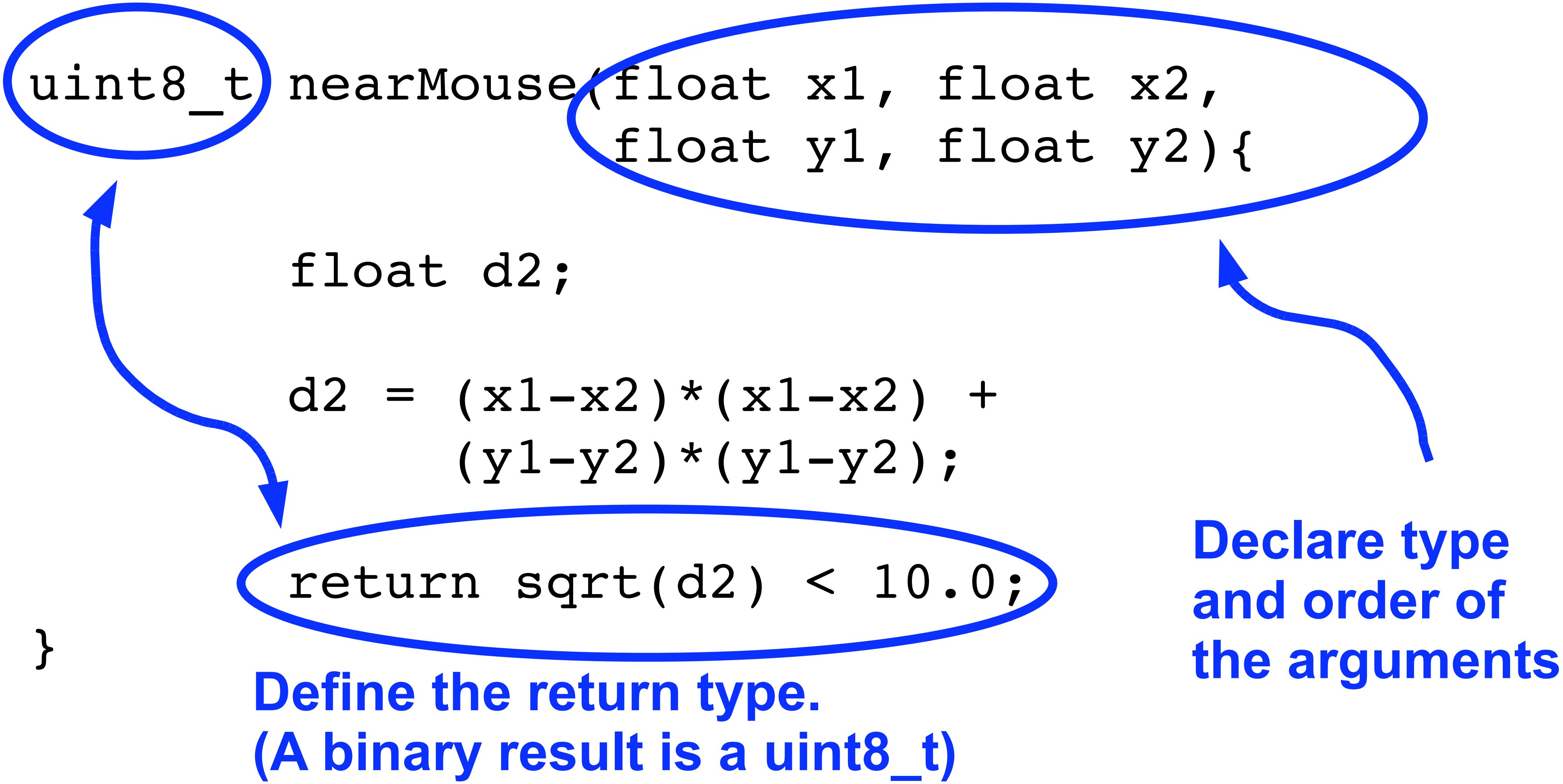
Functions

```
int umain (void) {
        // ...

        if (nearMouse(myX, mouseX,
                      myY, mouseY)){
                stop();
        }

        // ...
}
```

Functions

```
uint8_t nearMouse(float x1, float x2,
                  float y1, float y2){

        float d2;

        d2 = (x1-x2)*(x1-x2) +
             (y1-y2)*(y1-y2);

        return sqrt(d2) < 10.0;
}
```

Functions



```
uint8_t nearMouse(float x1, float x2,
                  float y1, float y2){

        float d2;

        d2 = (x1-x2)*(x1-x2) +
             (y1-y2)*(y1-y2);

        return sqrt(d2) < 10.0;
}
```

Define the return type.
(A binary result is a uint8_t)

Declare type
and order of
the arguments

Functions

```
uint8_t nearMouse(float,float,float,float);

int umain (void) {
        // ...
        if (nearMouse(myX, mouseX,
                      myY, mouseY)){
                stop();
        }
        // ...
}

uint8_t nearMouse(float x1, float x2,
                  float y1, float y2){
        // body of nearMouse() ...
}
```

Functions

```
uint8_t nearMouse(float,float,float,float);
```

Declare the function at the top.

```
int umain (void) {
        // ...
        if (nearMouse(myX, mouseX,
                      myY, mouseY)){
                stop();
        }
        // ...
}
```

Use it anywhere

```
uint8_t nearMouse(float x1, float x2,
                  float y1, float y2){
        // body of nearMouse() ...
}
```

But you have to actually implement it somewhere the file, of course...

Functions

```
uint8_t nearMouse(float,float,float,float);

int umain (void) {
        // ...
        if (nearMouse(myX, mouseX,
                      myY, mouseY)){
                stop();
        }
        // ...
}

uint8_t nearMouse(float x1, float x2,
                  float y1, float y2){
        // body of nearMouse() ...
}
```

Functions

```
uint8_t nearMouse(float,float,float,float);

int umain (void) {
        // ...
        if (nearMouse(myX, mouseX,
                      myY, mouseY)){
                stop();
        }
        // ...
}

uint8_t nearMouse(float x1, float x2,
                  float y1, float y2){    ←
        // body of nearMouse() ...
}
```

Functions

```
uint8_t nearMouse(float x1, float x2,
                  float y1, float y2){

        float d2;

        d2 = (x1-x2)*(x1-x2) +
             (y1-y2)*(y1-y2);

        return sqrt(d2) < 10.0;
}
```

Functions

```
uint8_t nearMouse(float x1, float x2,
                  float y1, float y2){

        float d2;

        d2 = (x1-x2)*(x1-x2) +
             (y1-y2)*(y1-y2);

        return sqrt(d2) < 10.0;
}
```

Functions

```
uint8_t nearMouse(float x1, float x2,
                  float y1, float y2){

        float d2;

        d2 = (x1-x2)*(x1-x2) +
             (y1-y2)*(y1-y2);

        return sqrt(d2) < 10.0;      ←
}
```

Functions

```
uint8_t nearMouse(float,float,float,float);

int umain (void) {
        // ...
        if (nearMouse(myX, mouseX,
                      myY, mouseY)){
                stop();
        }
        // ...
}

uint8_t nearMouse(float x1, float x2,
                  float y1, float y2){
        // body of nearMouse() ...
}
```

Functions

```
uint8_t nearMouse(float,float,float,float);

int umain (void) {
        // ...
        if (1){          ←
                stop();
        }
        // ...
}

uint8_t nearMouse(float x1, float x2,
                  float y1, float y2){
        // body of nearMouse() ...
}
```

Functions

```
uint8_t nearMouse(float,float,float,float);

int umain (void) {
        // ...
        if (1){
                stop();  ←
        }
        // ...
}

uint8_t nearMouse(float x1, float x2,
                  float y1, float y2){
        // body of nearMouse() ...
}
```

Functions

```
void driveForward(int16_t vel){
        motor_set_vel(0, vel);
        motor_set_vel(1, vel);
}
```

Functions

```
void driveForward(int16_t vel){
        motor_set_vel(0, vel);
        motor_set_vel(1, vel);
}
```

Functions don't have to return anything.

This function turns on a motor with velocity that ranges from -256 to 256.

Functions

```
void driveForward(){
        motor_set_vel(0, 100);
        motor_set_vel(1, 100);
}
```

Functions don't have to have arguments, either...

```
// ...
float x = doStuffWithNumbers(5.6,7);
if (x == 42){
        driveForward();
}
// ...
```

Common Mistakes

```
int x = 4;
if (x = 5) {
  printf("WTF?!");
}

float x = 3.9;
if (x + 0.1 != 4) {
  printf("MATH FAIL!");
}

uint8_t i;
for (i = 0; i < 300; i++) {
  printf(“%d\n”, i);
}
```

Happylab

- Soldering tutorial at the beginning
- Read entire lab
- Only need to actually wire up the following:
  - DC Motor
  - Gyro
  - Breakbeam
- You can test gyro and breakbeam on your own happyboard
- We have rental bots if you want to try out dc motors, servos
- If you're not sure which pin is which – see the example sensors

Function Reference

digital_read(pin) - read the input on pin, returns 0 or 1
analog_read(pin) - read the analog voltage on pin, returns 0-1023 (0-5V)
motor_set_vel(motor, vel) - set motor velocity (-255 to 255)
motor_brake(motor) - "brake" motor
servo_set_pos(servo, pos) - sets the servo to a specified position (0-511)
servo_disable(servo) - turns off control signals to servo - useful to stop continuous rotation servos
frob_read_range(low, high) - reads the frob - returns a number from low to high.
pause(millis) - pause the program
printf(params...) - write output to the USB port
go_click() / stop_click() - pause execution until go/stop pressed
go_press() / stop_press() - returns 1 if go/stop is currently pressed
encoder_read(pin) - read encoder clicks
encoder_reset(pin) - reset encoder clicks to 0
get_time() - get # of millis since Happyboard was turned on
get_time_us() - get # of microseconds since Happyboard was turned on