Writing Your First Program

© KISS Institute for Practical Robotics 1993-2026

Table of Contents

Table of Contents I

1 Table of Contents

2 Wombat Controller Guide
- The Hardware
- BotUI (Wombat OS)
- Activity 8.0: Connecting to Your Wombat with WiFi
- Activity 8.1: Accessing the KIPR Software Suite
- Activity 8.2: Connecting to Your Wombat with Ethernet

3 The KIPR Software Suite
- Activity 8.3: Welcome to the Software Suite
- Activity 8.4: Organizing and Creating Users and Projects
- Activity 8.5: Creating a User Folder
- Activity 8.6: Adding a Project
- Activity 8.7: A Tour of the KIPR Editor
- Activity 8.8: Compiling Your First Project
- Activity 8.9: Running Your Program on Your Robot

Table of Contents II

4 Learning About The C Template

- Activity 8.10: The KIPR Library
- Activity 8.11: Functions
- Activity 8.12: Quick Reference
- Activity 8.13: A Block of Code
- Activity 8.14: Programming Statements
- Activity 8.15: Terminating Statements
- Activity 8.16: Ending with return
- Activity 8.17: Program Speed
- Activity 8.18: Curly Braces
- Activity 8.19: Program Colors

5 Programming Basics

- Activity 8.20: Comments and Pseudocode
- Activity 8.21: Adding comments

Table of Contents III

- Activity 8.22: Running the Program with Comments
- Activity 8.23: The Importance of Commenting
- Activity 8.24: Printing to the Screen
- Activity 8.25: The msleep() function

6 Activity 8.26: Debugging

- Common Errors
- More Common Errors
- Closing Notes
- Reminder: Powering Off Your Wombat

Wombat Controller Guide

The Hardware

The Hardware

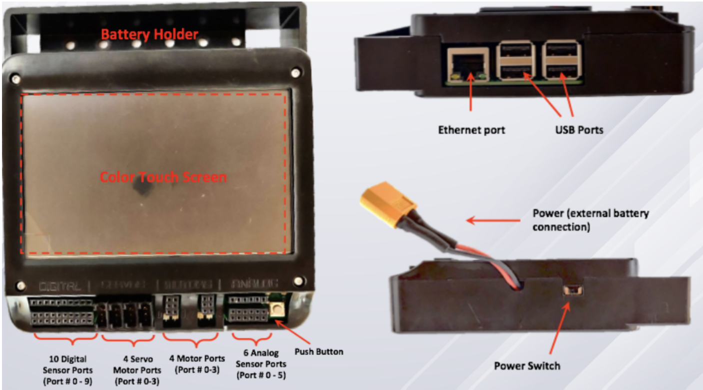

Battery Holder
Color Touch Screen
DIGITAL | SERVOS | MOTORS | ANALOG
10 Digital Sensor Ports (Port # 0 - 9)
4 Servo Motor Ports (Port # 0-3)
4 Motor Ports (Port # 0-3)
6 Analog Sensor Ports (Port # 0 - 5)
Push Button
Ethernet port
USB Ports
Power (external battery connection)
Power Switch

Figure 1: Wombat Controller Diagram

Making the Connection

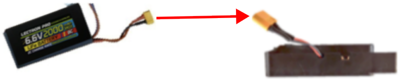


Figure 2: Yellow to yellow (Battery to controller)

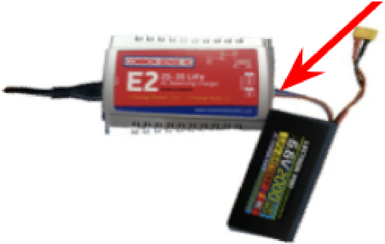


Figure 3: Small white to small white (Controller to charger)

Wombat Power

The KIPR Robotics Controller – Wombat, uses an external battery pack for power. It will void your warranty to use a battery pack with the Wombat that hasn't been approved by KIPR.

When your Wombat is not in use please be sure to do the following:

- TURN YOUR Wombat OFF
- UNPLUG THE BATTERY FROM THE Wombat

Leaving your battery plugged in and your Wombat turned on will drain your battery to the point where it can no longer be charged. If you plug your battery into the charger and the blue lights continue to flash, then you have probably drained your battery to the point where it cannot be charged again. If this happens you can call the KIPR office to help troubleshoot and/or purchase a replacement - 405-579-4609.

© KISS Institute for Practical Robotics 1993-2026

Charging the Controller's Battery

For charging the controller's battery, **use only the power supply which came with your controller**.

- *It is possible to damage to the battery from using the wrong charger or from too deep a discharge!*

The standard power pack is a lithium iron phosphate (LiFe) battery, a safer alternative to lithium polymer batteries. The safety rules applicable for re-charging any battery still apply:

- Do NOT leave the unattended while charging.
- Turn the Wombat off or unplug it from the battery while charging
- Charge in a cool, open area away from flammable materials.

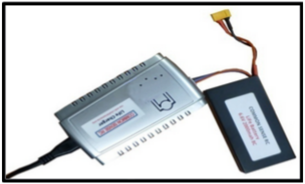


Figure 4: Plugging the battery into the charger

Powering on Your Wombat

The power switch is located on the side of the Wombat controller next to the external battery cable.

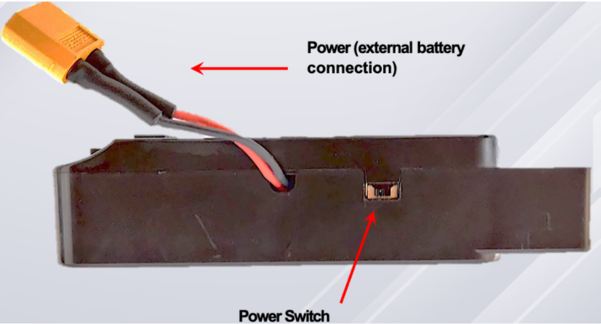
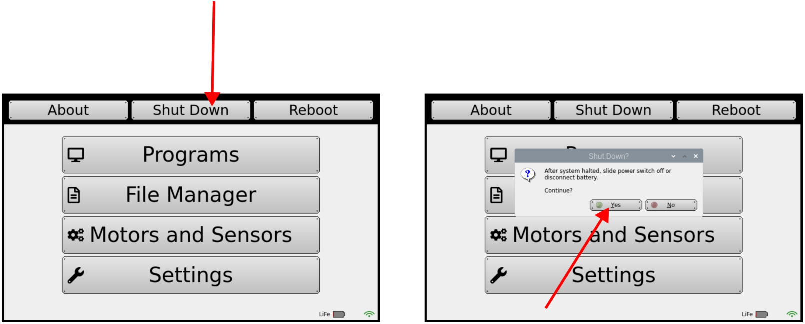


Figure 5: Wombat power switch

Powering Off Your Wombat

1. From the Home Screen, press "Shutdown."
2. Press "Yes" to confirm.



After system halted, slide power switch off or disconnect battery.

Continue?

Figure 6: Wombat shutdown procedure

Powering Off Your Wombat

3. After shutting down from the Home Screen, flip the power switch to off.
4. Unplug the battery, being careful to only grab the yellow connectors, **not** the wires.

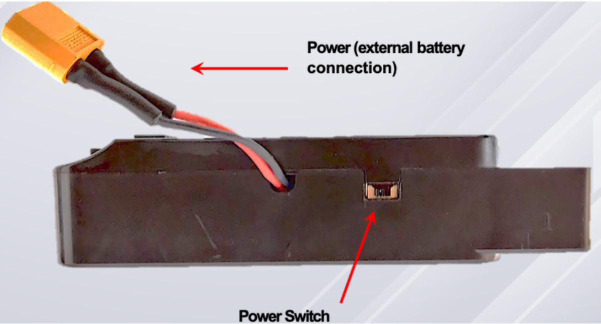


Figure 7: Wombat power switch

Battery Indicator

The Wombat has a battery indicator in the bottom right. **As of August 2025, this is broken and does not reflect the actual battery life**.

However, the is a yellow LED next to the red power indicator, which will only light up when the battery gets low. If the battery gets too low you will return to the rainbow screen.

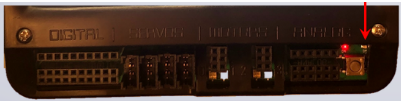


Figure 8: Yellow LED visible only when battery is critically low

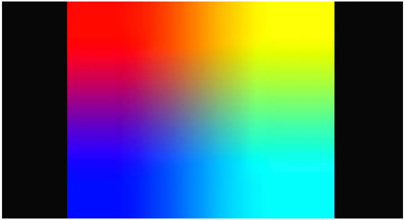


Figure 9: Rainbow screen

BotUI (Wombat OS)

Backing Up Your Programs

It is important to save your code somewhere in addition to on your robot. You have several options to back it up:

1. You can simply copy your code out of the IDE and paste it into a google document (or similar).
2. You can use a USB flash drive to back it up.
3. You can download your code from the IDE.

Backing Up With a USB

1. Boot up and insert USB drive into the ports on the side of the Wombat.
2. Select "Settings".
3. Select "Backup".
4. Select "Backup".

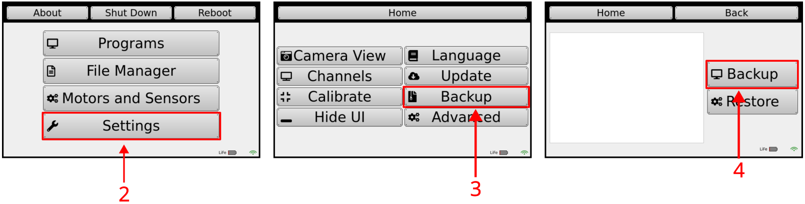


Figure 10: BotUI Backup Guide

Restoring Programs With a USB

1. Boot up and insert USB drive into the ports on the side of the Wombat.
2. Select “Settings”.
3. Select “Backup”.
4. Select “Restore”.

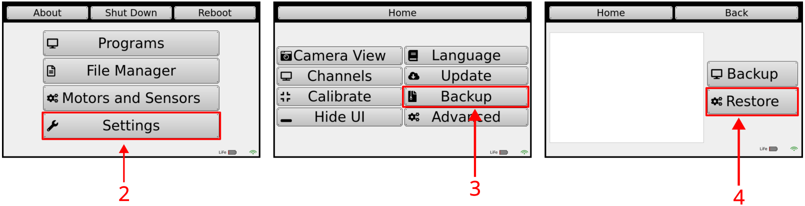


Figure 11: BotUI Restore Guide

Help! Where is My Home Screen?

Students may accidentally (or on purpose) hide BotUI, which will go the the desktop. To return to BotUI, they should select the Botguy icon on the top row.

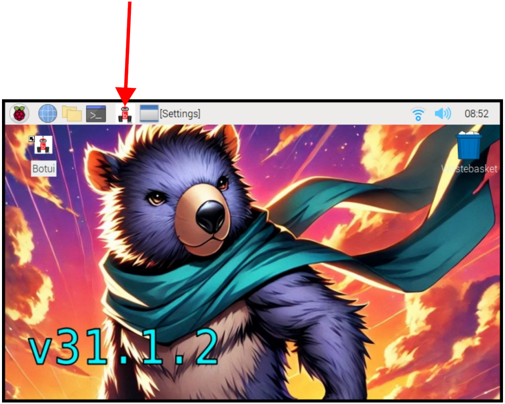


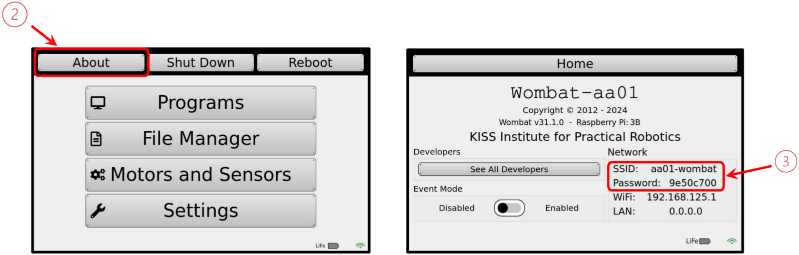
Figure 12: BotUI Return Guide

Activity 8.0: Connecting to Your Wombat with WiFi

© KISS Institute for Practical Robotics 1993-2026

Getting Network Info

1. Turn the Wombat with the black switch on the side
2. Select "About" on the main menu
3. Note the rows that say "SSID" and "Password"
   - If this section is blank for you, see the next section for troubleshooting steps.



Wombat-aa01
Copyright © 2012 - 2024
Wombat v31.1.0 - Raspberry Pi: 3B
KISS Institute for Practical Robotics

Developers

Network
SSID: aa01-wombat
Password: 9e50c700
WiFi: 192.168.125.1
LAN: 0.0.0.0

Event Mode
Disabled ● Enabled

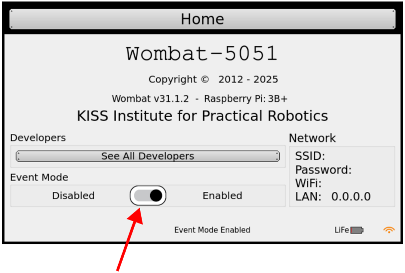
Figure 13: Getting Your Network Info

What If My Wifi Line is Empty?

This is a known issue which affects some older WombatOS versions. The best way to fix it is to update your Wombat! Instructions can be found on our site: kipr.org.

If you can't update immediately, here is a quick fix:

1. Flip the "Event Mode" switch to "Enabled."
2. Return to the Home Screen, wait **at least 5 seconds**, then return to "About".
3. Flip the "Event Mode" switch to "Disabled."
4. Return to the Home Screen, wait **at least 5 seconds**, then return to "About".
5. You should see numbers on the Wifi line.


Home

Wombat-5051

Copyright © 2012 - 2025

Wombat v31.1.2 - Raspberry Pi: 3B+

KISS Institute for Practical Robotics

Developers

See All Developers

Event Mode

Disabled ● Enabled

Network

SSID:
Password:
WiFi:
LAN: 0.0.0.0

Event Mode Enabled

LiFe

Figure 14: Enabling Event Mode

Connecting to the Wombat on a Chromebook

1. Enter your Wifi settings. The location may be different for some brands, but is usually on the bottom left.
2. Select your Wombat's Wi-Fi network from the list.
3. Enter the password from the Wombat's About page (previous slide).

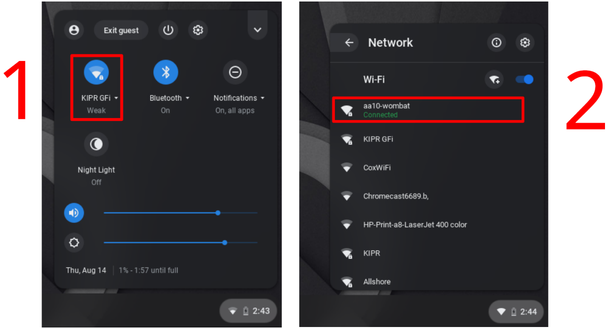


Figure 15: Connecting to the Wombat's Network

Connecting to the Wombat on Windows

1. Find your Wombat Wi-Fi signal in your Wi-Fi settings. The menu is usually located in the bottom left of your screen.
2. If you see "Enter the PIN from the router label", click "Connect using a security key instead."
3. Enter the password from the Wombat's About page (previous slide).

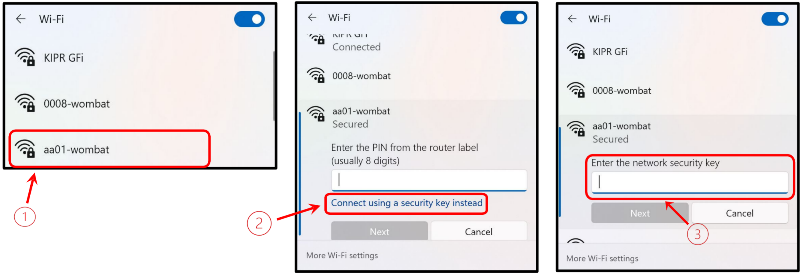


Figure 16: Connecting to the Wombat's Network

Connecting to the Wombat on Mac

1. Find your Wombat Wi-Fi signal in your Wi-Fi settings. The menu is located on the top bar of your screen (left image) or in your system settings app (right image).
2. Enter the password from the Wombat's About page (previous slide).

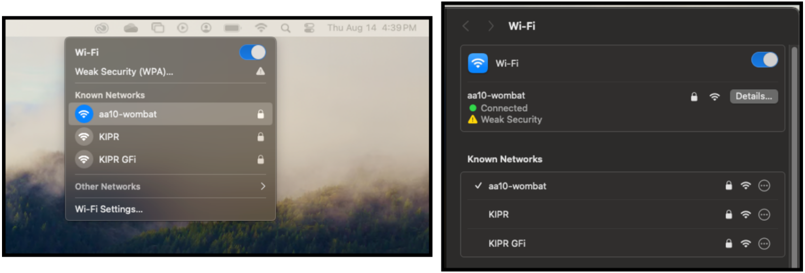


Figure 17: Connecting to the Wombat's Network

Important Note!

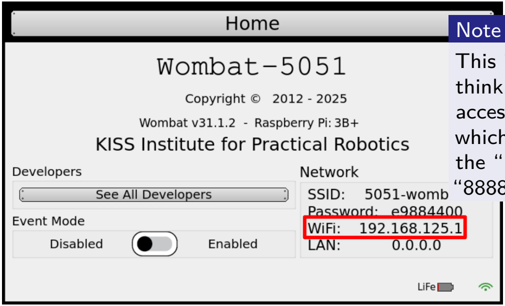
When you connect, you will probably see a warning like “no internet connection” or “connected with limited access”. This is normal, proceed to the next section.

Activity 8.1: Accessing the KIPR Software Suite

© KISS Institute for Practical Robotics 1993-2026

Activity 8.1: Accessing the KIPR Software Suite

1. After connecting, launch a web browser (such as Safari, Chrome or Firefox).
2. Note the set of numbers on the “WiFi” line.



Home

Wombat−5051

Copyright © 2012 - 2025

Wombat v31.1.2 - Raspberry Pi: 3B+

KISS Institute for Practical Robotics

Developers

See All Developers

Event Mode

Disabled ● Enabled

Network

SSID: 5051-womb

Password: e9884400

WiFi: 192.168.125.1

LAN: 0.0.0.0

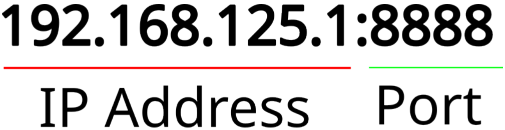
Note

This is called an “IP Address.” You can think of it as the building address, but to access the IDE, we also need to know which door to knock on. That is called the “port number,” and by default it is “8888.”

Figure 18: Finding Your Wombat’s IP Address

Activity 8.1: Accessing the KIPR Software Suite

To access the KIPR IDE, we need to combine the IP address with the port number, like this:



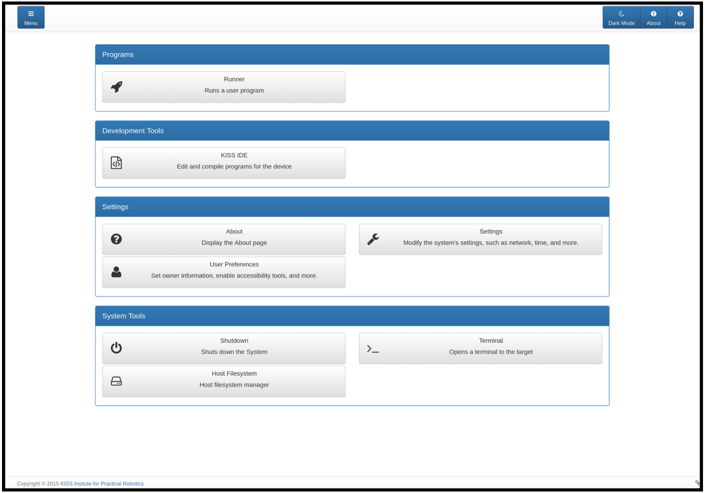
192.168.125.1:8888

IP Address

Port

Enter this into your browser’s URL bar, making sure to match the punctuation *exactly*. You should see the KIPR IDE (reference image next slide). If you need to use an ethernet cable to connect refer to the next section, “Connecting to Your Wombat with Ethernet.

KIPR IDE Reference Image



Programs

Runner
Runs a user program

Development Tools

KISS IDE
Edit and compile programs for the device

Settings

About
Display the About page

Settings
Modify the system's settings, such as network, time, and more.

User Preferences
Set owner information, enable accessibility tools, and more.

System Tools

Shutdown
Shuts down the System

Terminal
Opens a terminal to the target

Host Filesystem
Host filesystem manager

Copyright © 2015 KISS Insitute for Practical Robotics

Figure 19: KIPR IDE Homepage

Activity 8.2: Connecting to Your Wombat with Ethernet

Activity 8.2: Connecting to Your Wombat with Ethernet

You can also connect to your Wombat over a wired connection (ethernet). Ethernet can be more stable that WiFi, but only one person can be connected at a time. If you successfully connected in the previous section, you may proceed to the next section The KIPR Software Suite.

Activity 8.2: Connecting to Your Wombat with Ethernet

1. Connect your device that has an Ethernet port to the Wombat with an ethernet cable.
2. If you have no Ethernet port you need a dongle to convert USB to Ethernet and an Ethernet cable (refer to next slide).

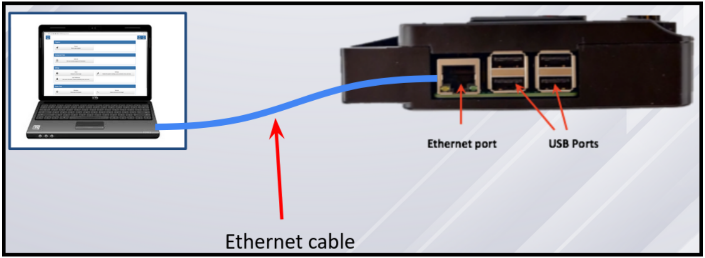


Figure 20: Wombat Ethernet Connection Graphic

Using a Dongle

- If your device does not have an ethernet port, you will need a dongle to convert it to USB.
- Make sure the dongle you purchase is compatible with your operating system (Windows, Mac, or Chrome)

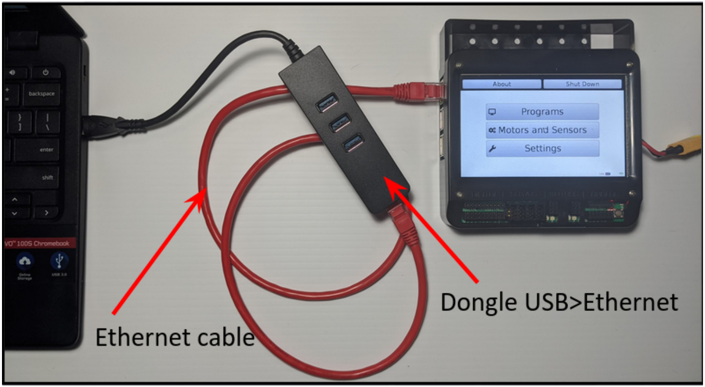
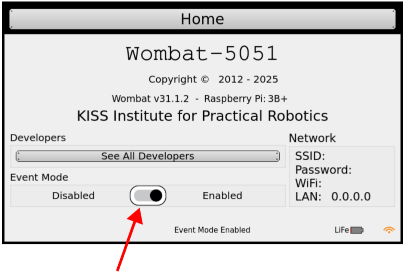


Figure 21: Ethernet Dongle

Enabling Event Mode

1. Make sure you have everything plugged into your wombat (Ethernet or Ethernet + Dongle).
2. This will allow only one person at a time to program the robot but it will ensure that you can connect if there is a lot of WiFi Interference.
3. FIRST you must use the UI on your robot by going to the "About" screen, then toggle to **Event Mode Enabled**.



Wombat-5051

Copyright © 2012 - 2025

Wombat v31.1.2 - Raspberry Pi: 3B+

KISS Institute for Practical Robotics

Developers

Event Mode

Disabled ● Enabled

Network

SSID:
Password:
WiFi:
LAN: 0.0.0.0

Event Mode Enabled

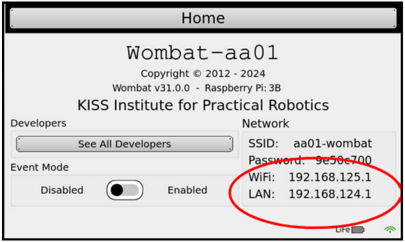
Figure 22: Enabling Event Mode

Hint

If LAN says "0.0.0.0", simply return to the Home Screen, the come back to "About."

Getting Network Info

Once plugged into the Wombat, with Event Mode Enabled, note the IP address on the "LAN" line. This is slightly different from the WiFi IP address, so be careful.



Home

Wombat-aa01

Copyright © 2012 - 2024

Wombat v31.0.0 - Raspberry Pi: 3B

KISS Institute for Practical Robotics

Developers

Event Mode

Network

SSID: aa01-wombat

Password: 9e50c700

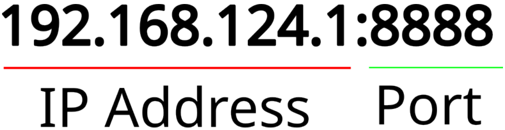
WiFi: 192.168.125.1

LAN: 192.168.124.1

Figure 23: Ethernet IP Address

Activity 8.1: Accessing the KIPR Software Suite

1. Launch a web browser (such as Safari, Chrome or Firefox).
2. Enter the address into the URL bar.



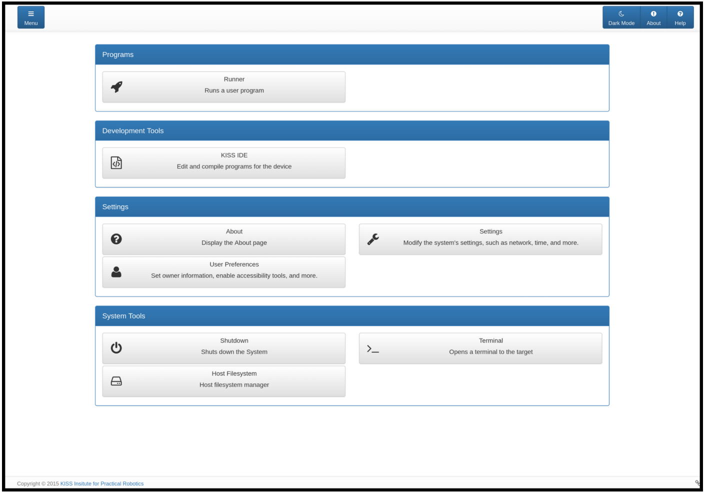
192.168.124.1:8888

IP Address

Port

This IP address and port tells the browser where to find the KIPR IDE. See the next slide for an sample image.

KIPR IDE Reference Image



Programs

Runner
Runs a user program

Development Tools

KISS IDE
Edit and compile programs for the device

Settings

About
Display the About page

Settings
Modify the system's settings, such as network, time, and more.

User Preferences
Set owner information, enable accessibility tools, and more.

System Tools

Shutdown
Shuts down the System

Terminal
Opens a terminal to the target

Host Filesystem
Host filesystem manager

Copyright © 2015 KISS Insitute for Practical Robotics

Figure 24: KIPR IDE Homepage

The KIPR Software Suite

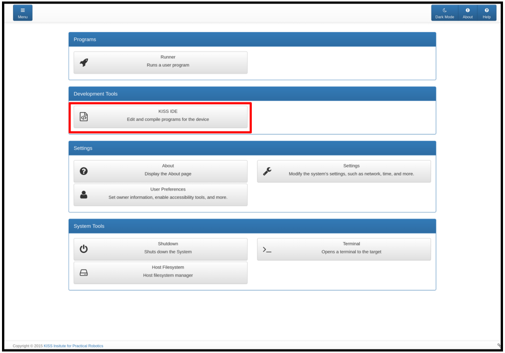
Activity 8.3: Welcome to the Software Suite

© KISS Institute for Practical Robotics 1993-2026

Activity 8.3: Welcome to the Software Suite

To make it easier for you to learn and use a programming language, KIPR provides a web-based Software Suite, which will allow you to write and compile source code using C, C++, Python, and block coding (v. 32+). The development package will work with almost any web browser **except Internet Explorer**.

Click on “KISS IDE” and proceed to the next slide.



Programs

Runner
Runs a user program

Development Tools

KISS IDE
Edit and compile programs for the device

Settings

About
Display the About page

Settings
Modify the system's settings, such as network, time, and more.

User Preferences
Set owner information, enable accessibility tools, and more.

System Tools

Shutdown
Shuts down the System

Terminal
Opens a terminal to the target

Host Filesystem
Host filesystem manager

Copyright © 2015 KISS Insitute for Practical Robotics

Activity 8.4: Organizing and Creating Users and Projects

Activity 8.4: Organizing and Creating Users and Projects

Create a folder for each student. This will make is easy for them to find their projects. Do not use the default user.

When creating a new user (folder) or a new project do not:

- Put any special characters or periods, etc. on it.
- This will eventually interfere with your project and later you will have problems.

Examples of good user(folder) names:

- Botguy folder
- sarah folder
- Sarah Projects

Examples of good project names:

- Activity 1
- Hello World
- Functions Introduction

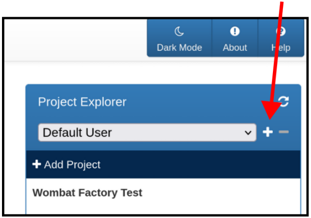
Bad examples:

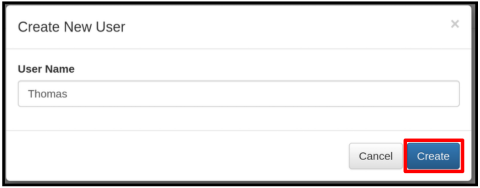
- m.j.c.
- my amazing project!
- Mrs Davis's project.
- :)

Activity 8.5: Creating a User Folder

Activity 8.5: Creating a User Folder

1. Click on KISS IDE.
2. Under **Project Explorer** click the + sign to add a user.
3. Name you new user (use your name, not mine!).
4. Click create.




Create New User

User Name: Thomas

Figure 25: Creating Your User

Activity 8.6: Adding a Project

© KISS Institute for Practical Robotics 1993-2026

Activity 8.6: Adding a Project

1. Proceed back to "Project Explorer" and select the user name you created from the drop down. You should see the folder you created.
2. Click "+ Add Project."
3. Continue to the next section for naming you project.

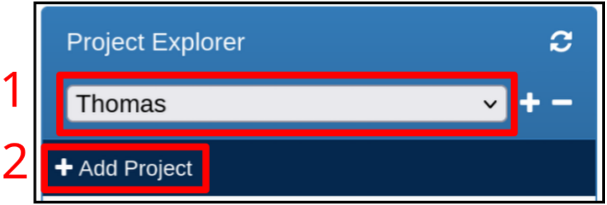
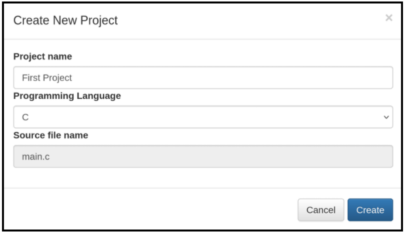


Figure 26: Adding a project

Naming Your Project

1. Enter the name of your project (e.g. “First Project”).
2. Leave the “Programming Language” as “C” and the “Source file name” as `main.c`.
3. Click “Create.”



Create New Project

Project name: First Project

Programming Language: C

Source file name: main.c

Figure 27: Naming your project

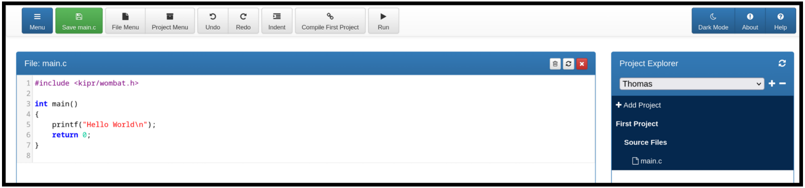
Warning

Make sure you don’t use any special characters! This includes “.”, “@”, “!”, or any emojis!

Activity 8.7: A Tour of the KIPR Editor

Activity 8.7: A Tour of the KIPR Editor

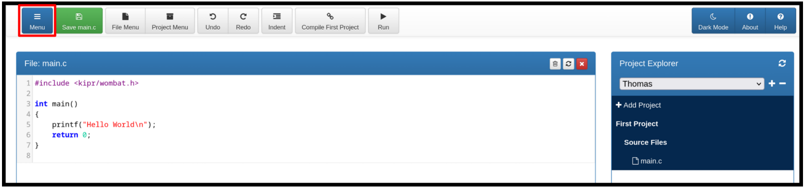
This is how every project will look when you first start. Proceed to the next section for more information.



```
#include <kipr/wombat.h>

int main()
{
    printf("Hello World\n");
    return 0;
}

```

Menu

“Menu” takes you right back to the Main Menu of the KIPR Software Suite.

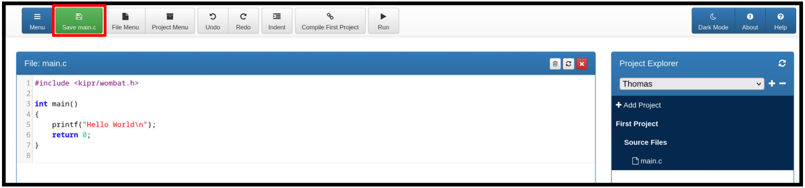


```
#include <kipr/wombat.h>

int main()
{
    printf("Hello World\n");
    return 0;
}
```

Figure 28: Menu button

Save

“Save main.c” saves your project code. A successful Compile also saves your code eliminating the need to use save main.c.

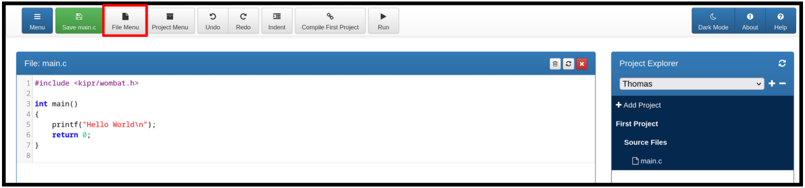


```
#include <kipr/wombat.h>

int main()
{
    printf("Hello World\n");
    return 0;
}

```

Figure 29: Save button

File Menu

"File Menu" gives you the option to delete or download `main.c` directly to your computer. This is another way to back up your code.

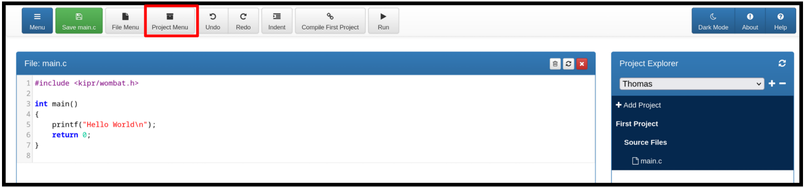


```
#include <kipr/wombat.h>

int main()
{
    printf("Hello World\n");
    return 0;
}

```

Figure 30: File menu button

Project Menu

“Project Menu” gives you the option to delete or download the entire project directly to your computer. This is another way to back up your code.

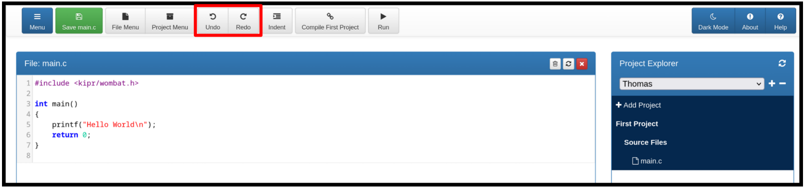


```
#include <kipr/wombat.h>

int main()
{
    printf("Hello World\n");
    return 0;
}
```

Figure 31: Project menu button

Undo and Redo

“Undo” undoes the last keystrokes (exactly like Control + Z). “Redo” puts the undone keystrokes back in.

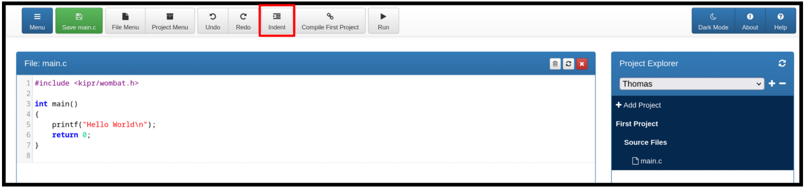


```
#include <kipr/wombat.h>

int main()
{
    printf("Hello World\n");
    return 0;
}

```

Figure 32: Undo and Redo buttons

Indent

“Indent” will format all of the code so it is easier to read. You should use this frequently!



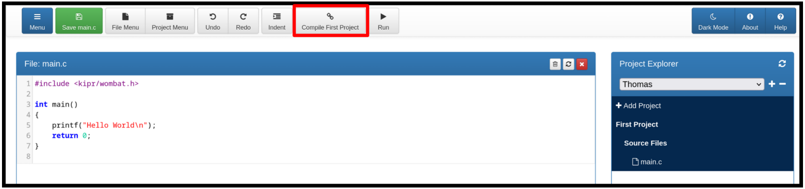
```
#include <kipr/wombat.h>

int main()
{
    printf("Hello World\n");
    return 0;
}

```

Figure 33: Indent button

Activity 8.8: Compiling Your First Project

Activity 8.8: Compiling Your First Project

“Compile” converts the source code (what you see in the editor below) to machine code that the robot can understand. If it compiles successfully, it also automatically saves the code. If it fails, it will give debugging information to help you figure out what went wrong. There is more explanation on debugging later in this document.



```
#include <kipr/wombat.h>

int main()
{
    printf("Hello World\n");
    return 0;
}
```

Figure 34: Compile button

Activity 8.9: Running Your Program on Your Robot

Activity 8.9: Running Your Program on Your Robot

“Run” executes (runs) the code that was successfully compiled.

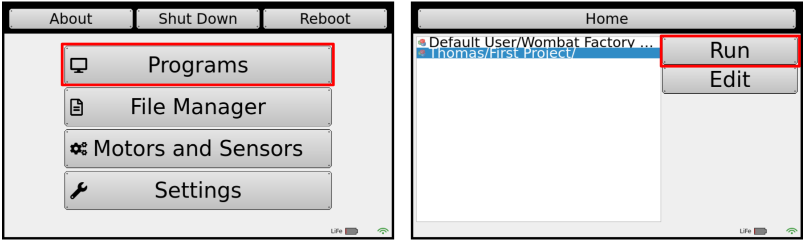


Figure 35: Run button

Learning About The C Template

Activity 8.10: The KIPR Library

Activity 8.10: The KIPR Library

Line 1 includes the KIPR library. All programs must include the KIPR library, as it contains all the functions you need to control your robot.

```
1  #include <kipr/wombat.h>
2
3  int main ()
4  {
5      printf("Hello World!\n");
6      return 0;
7  }
```

Comments

Throughout this section, we will be annotating the example code with comments. A comment starts with “//”, and when the computer sees this pattern, it will ignore everything else **on that line.** Programmers use comments to explain their code.

```c
#include <kipr/wombat.h>

// Notice the green highlighting: this helps you identify comments
int main () // The computer will ignore this: you can write anything here
{
    printf("Hello World!\n");
    return 0;
}
```

Activity 8.11: Functions

Activity 8.11: Functions

A “function” defines a list of actions to take, in a similar manner to a recipe. Executing a function or “calling” (using) the function means the controller will follow the instructions contained in the function.

Example: You might want a robot with a `clean_house()` function that could mean vacuum, dust, mop, change the linens, wash the windows, etc. . . all the commands specified in the function are executed.

Line 3 of the template defines the “main” function. When you run your program, the main function is always executed.

```c
1   #include <kipr/wombat.h>
2
3   int main ()
4   {
5       printf("Hello World!\n");
6       return 0;
7   }
```

Activity 8.12: Quick Reference

Activity 8.12: Quick Reference

There are many functions in the KIPR library. Here is a quick reference for some of the most common ones. Don't worry if this seems overwhelming, these will all be explained more thouroughly in later documents.

```
printf("text\n");                  // Prints text to the display
motor(port, %power);               // Activate motor in port at % power
msleep(milliseconds);              // Program pauses for specified millisecond
ao();                              // [a]ll [o]ff, turns all motors off
enable_servos();                   // Turns servo ports on
set_servo_position(port, position); // Moves servo in port to position
disable_servos();                  // Turns off servo ports
digital(port);                     // Get digital sensor value in port
analog(port);                      // Get digital sensor value in port
```

Activity 8.13: A Block of Code

Activity 8.13: A Block of Code

The area between the **{** and **}** (lines 4 and 7) is called a “**block of code.**” Inside this block, we write lines of code called “**programming statements.**”

```
1   #include <kipr/wombat.h>
2
3   int main ()
4   { // Block begins here
5       printf("Hello World!\n");
6       return 0;
7   } // Block ends here
```

© KISS Institute for Practical Robotics 1993-2026

Activity 8.14: Programming Statements

© KISS Institute for Practical Robotics 1993-2026

Activity 8.14: Programming Statements

Each “**programming statement**” is an action to be executed by the robot in the order that it is listed. A program may have any number of programming statements.

```c
#include <kipr/wombat.h>

int main ()
{ // Block begins here
    printf("Hello World!\n"); // Programming statement
    return 0; // Programming statement
} // Block ends here
```

Activity 8.15: Terminating Statements

© KISS Institute for Practical Robotics 1993-2026

Activity 8.15: Terminating Statements

Terminating statements end each programming statement. Use a semicolon (unless it is followed by a new block of code) to end the programing statement. This is similar to an English sentence, which ends with a period.

In English a statement that is missing punctuation is a run-on sentence or incomplete sentence. A semicolon is similar to an “enter” or “return” key on your keyboard, it tells the computer to proceed to the next line.

Notice how the pogramming statements on lines 5 and 6 end with semicolons:

```
1  #include <kipr/wombat.h>
2
3  int main ()
4  {
5      printf("Hello World!\n");
6      return 0;
7  }
```

Activity 8.16: Ending with return

© KISS Institute for Practical Robotics 1993-2026

Activity 8.16: Ending with return

The main() function ends with a return statement, which is the response or answer to the computer (or robot). In this case, the "answer" back to the computer is 0. The return statement is the **last line before the }** brace.

```c
#include <kipr/wombat.h>

int main ()
{
    printf("Hello World!\n");
    return 0;
}
```

Activity 8.17: Program Speed

Activity 8.17: Program Speed

Computers read a program like you read a book: starting at the top and reading line by line to the bottom. Computers read incredibly fast: the Wombat reads 800: **million** lines per second!

```c
#include <kipr/wombat.h>

int main ()
{
    printf("Hello World!\n");
    return 0;
}
```

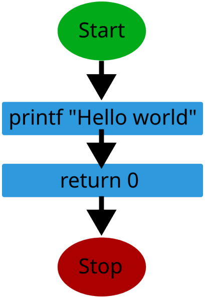


Figure 36: Program flowchart

Activity 8.18: Curly Braces

Activity 8.18: Curly Braces

The **curly braces** organizes the programming statements while executing them from top to bottom.

```
#include <kipr/wombat.h>

int main ()
{ // Start
    printf("Hello World!\n");
    return 0;
} // Stop
```

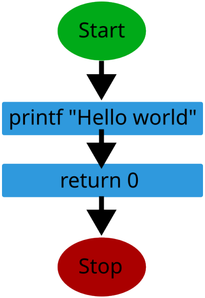


Figure 37: Program flowchart

© KISS Institute for Practical Robotics 1993-2026

Activity 8.19: Program Colors

© KISS Institute for Practical Robotics 1993-2026

Activity 8.19: Program Colors

The KISS IDE highlights certain parts of the program to make it easier to read.

- Comments appear in green.
- Keywords appear in bold blue.
- Text strings appear in red.
- Numbers appear in aqua.

```c
#include <kipr/wombat.h>

int main ()
{
    // This program will display "Hello World!"
    printf("Hello World!\n");
    return 0;
}
```

© KISS Institute for Practical Robotics 1993-2026

Programming Basics

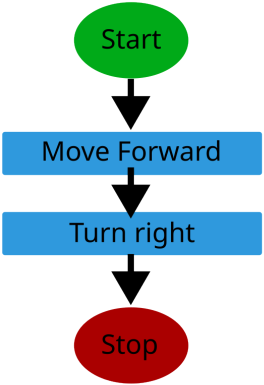
Activity 8.20: Comments and Pseudocode

Activity 8.20: Comments and Pseudocode

Read and discuss the next two slides with a partner to understand pseudocode. Pseudocode means “false code”. Easy to understand pseudocode can be used as commenting on what you expect your robot to do.

```
// Move forward
// Turn right
// Stop
```

Discuss with a partner why it might be important to create pseudocode. When finished, proceed to the next section.



Start

Move Forward

Turn right

Stop

Comments as pseudocode

Using comments as pseudocode can help you keep track of what is going on in the program. You can make a flow chart or list and then convert it to pseudocode.

As we saw earlier, a comment begins with two slashes: “//”. The computer will ignore anything in a comment, but you can refer to it later.

```c
#include <kipr/wombat.h>

int main ()
{
    printf("Hello World!\n"); // Prints "Hello World!" to the screen
    return 0;
}
```

Comments as attribution

Comments are also commonly used to give “attribution” in code. This means that:

- Anyone reading someones code knows who the rightful author is.
- If anyone wants to borrow some parts of the code they can ask permission and then accurately source where they got the code from.

```c
// Author: Jon Snow
// Program purpose: Prints text to the screen
// Created: 01/01/1970
#include <kipr/wombat.h>

int main ()
{
    printf("Hello World!\n"); // Prints "Hello World!" to the screen
    return 0;
}
```

Activity 8.21: Adding comments

Activity 8.21: Adding comments

Add the `// Prints "Hello World!" to screen` comment to the program.

Just like using Word or Google Docs, you can click to set your cursor and then make space for the comment. Type the comment into your program. The comment can go on the line before the `printf` function or on the same line as the function.

After adding the comment, compile your program and see what happens! When you have finished, continue to the next section.

```c
#include <kipr/wombat.h>

int main ()
{
    // You can put the comment here...
    printf("Hello World!\n"); // Or you can put the comment here
    return 0;
}
```

Activity 8.22: Running the Program with Comments

Activity 8.22: Running the Program with Comments

1. From the Wombat Home Screen, select “Programs.”
2. This will take you to a list of programs currently on your controller.
3. Select the program you just compiled.
4. Press “Run” to run the program.
5. Pay close attention: do the the comments appear on the screen?

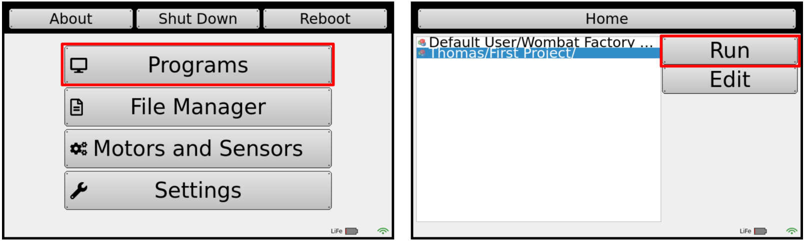


Figure 38: Running your program

Activity 8.23: The Importance of Commenting

Activity 8.23: The Importance of Commenting

You should start adding your own attribution to your code from now on! But remember, if you borrow even a small part of code from someone else, you must also give them attribution in your comments.

```
#include <kipr/wombat.h>

int main ()
{
    // Borrowed with permission from Sally
    // This code makes Wombat 0328 drive straight
    motor(0, 93);
    motor(3, 100);
    // End of borrowed code
    msleep(1000);
    return 0;
}
```

It's important to give appropriate attribution not only when copying code exactly, but even when taking someone's idea (or intellectual property) and changing it a little bit for your needs. When else might you need to give credit for someone else's work?

Commenting Multiple Lines of Code

```
#include <kipr/wombat.h>

int main ()
{
    // Forward
    motor(0, 100);
    motor(3, 100);
    msleep(1000);

    /*
    motor(0, -100);
    msleep(3000);
    motor(0, -100);
    motor(3, 100);
    msleep(3000);
    */
    return 0;
}
```

Block Comments

Everthing between the /* and */ is a comment and will be ignored by the computer. This is called a "block comment." You can use block comments to debug your code by making the computer ignore certain parts of your program.

© KISS Institute for Practical Robotics 1993-2026

Activity 8.24: Printing to the Screen

Using printf

1. Starting a new project.
2. Proceed back to “Project Explore” and select the User Name (folder) you created from the drop down.
3. Click “+ Add Project”, you are adding a project to your folder.
4. Name your project, “Printf Statements”.
5. Proceed to the next section.

Hello, who?

1. Write a program the displays “Hello World!”, then displays your name.
2. Compile and run the program on your Wombat.

Pseudocode (Task Analysis)

```
// Display "Hello world!" on the screen.
// Display "Hello name!" on the screen.
```

What function do you think you should use to print your name to the screen?

The printf() Function

The `printf()` function does exactly what we want! We just need to put the text we want inside quotation marks in the parenthesis.

```
printf("This will be printed to the Wombat screen\n");
```

What does the `\n` *do?* It is like telling you computer to press the “Enter” key at the end of the statement. Without it, multiple `printf()` statements will all try to print on the same line!

8.24: Possible Solution

```c
#include <kipr/wombat.h>

int main ()
{
    printf("Hello World!\n");

    printf("Hello Thomas!\n");

    return 0;
}
```

Running Your printf() Program

1. Make sure to compile your program. If you see a “Compilation Succeeded” message, proceed to the next step.
2. On the Wombat, follow the same procedure as before to run the program.

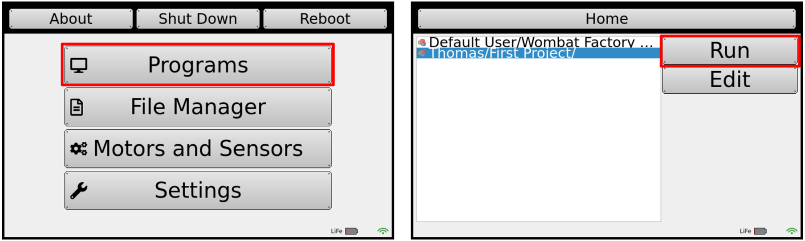
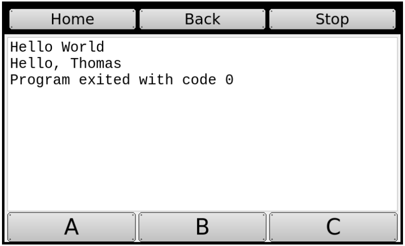


Figure 39: Running your program

Possible output

Your output may differ slightly, just make sure you see the lines “Hello World” and “Hello, name”



```
Hello World
Hello, Thomas
Program exited with code 0
```

Figure 40: `printf()` output

Making Observations

What did you notice when you ran the program? Try running it again, paying close attention to the output. Discuss your observations with your partner.

Possible Observations

1. The two statements are on different lines.
2. “Hello World” and “Hello name” seemed to appear at the same time.

We know that they are on different lines because we used `\n`, but why did they appear at the same time? Discuss with you partner.

Program Speed

Recall that the controller reads the code and goes to the next line faster than a blink of your eye. At 800MHz, the controller is executing ~800: Million lines of code/second!

What if we want to slow it down?

Activity 8.25: The msleep() function

Activity 8.25: The msleep() function

To slow the robot down, we can use the msleep() function to tell it to pause for a certain amount of time before it runs the next command. We tell it exactly how much time to wait by putting a number inside the parenthesis, like this:

```
// This will pause the robot for 1000 milliseconds (1 second)
msleep(1000);
```

Using msleep

1. Write a program that displays “Hello World”, pauses two seconds, then displays “Hello name.”
2. Place an `msleep(milliseconds);` between your two `printf()` statements.

Pseudocode (Task Analysis)

```
// 1. Display "Hello World!" on the screen.
// 2. Pause for 2 seconds.
// 3. Display your name on the screen.
```

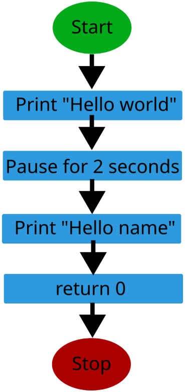
You saw before that `msleep(1000);` will make the controller “pause” for 1 second (the m stands for milliseconds or 1/1000 of a second) before going to the next line. Your program must tell the robot to wait for 2 seconds before going to the next command.

Guided Questions: How many seconds is 2000m? 3000m? How many milliseconds would you need to run the robot for 4 seconds?

msleep() Solution

Explain to a partner what this program will do.

```c
#include <kipr/wombat.h>

int main ()
{
    printf("Hello World!\n");
    msleep(2000);
    printf("Hello Thomas!\n");

    return 0;
}
```



© KISS Institute for Practical Robotics 1993-2026

Writing Code From Pseudocde

It is prudent to add your pseudocode as comments in your final program.

```
#include <kipr/wombat.h>

int main ()
{
    printf("Hello World!\n"); // Print "Hello World"
    msleep(2000); // Pause for 2 seconds
    printf("Hello Thomas!\n"); // Print "Hello Thomas"

    return 0;
}
```

Activity 8.26: Debugging

Activity 8.26: Debugging

Debugging is a very important skill for students to learn. It promotes problem solving, independence, and reading for meaning. This skill may have to be taught several times throughout the year. Remember, professional programmers spend most of their time debugging. Bugs don't make you a bad programmer!

Objectives: Students will learn the importance of and how to debug their programs.

Materials:

- Built Robot.
- Computer.

Activity: Follow the next slides to learn about debugging by leaving off important programing information, compiling, and find out how to read the compiler error messages.

Common Errors

Missing Semicolon

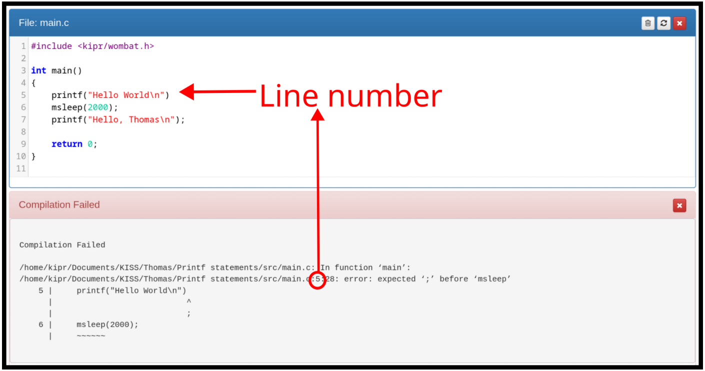
1. Leave off a terminating semicolon and see what happens.
2. Compile the program. What message appeared? Did the “Compilation succeeded” message appear?
3. Proceed to the next section to learn about reading errors.

```c
#include <kipr/wombat.h>

int main ()
{
    printf("Hello World!\n") // Oops! I forgot a semicolon!
    msleep(2000);
    printf("Hello Thomas!\n");

    return 0;
}
```

© KISS Institute for Practical Robotics 1993-2026

Missing Semicolon Error Message

Notice how the compiler helpfully tells you exactly where to look. We forgot a semicolon on line 5 (you can see the line numbers on the left margin)!



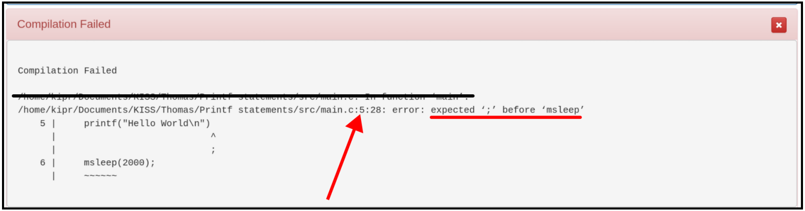
```
 1 #include <kipr/wombat.h>
 2 
 3 int main()
 4 {
 5     printf("Hello World\n")
 6     msleep(2000);
 7     printf("Hello, Thomas\n");
 8 
 9     return 0;
10 }
11 
```

```
Compilation Failed

/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c: In function ‘main’:
/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c:5:28: error: expected ‘;’ before ‘msleep’
    5 |    printf("Hello World\n")
      |                           ^
      |                           ;
    6 |    msleep(2000);
      |    ~~~~~~
```

Figure 41: Missing semicolon error message

Missing Semicolon Error Message

1. Ignore the first line and look at the second line to find the error.
2. This error says that line 5 it expected to see a ";" before `msleep()`, meaning on line 4.
   - It may also be the next programming statement before line 5 (`msleep()`). If there is white space it could be line 4 or line 3.
3. Fix **one error at a time** and then recompile. Fixing one might fix all the errors.
4. 5:28 means line 5 column 28. You cannot see columns, so just ignore that part.



Compilation Failed

```
Compilation Failed

~~/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c: In function 'main':~~
/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c:5:28: error: expected ‘;’ before ‘msleep’
    5 |     printf("Hello World\n")
      |                            ^
      |                            ;
    6 |     msleep(2000);
      |     ~~~~~~
```

Figure 42: Missing semicolon error message

Misspelled Function

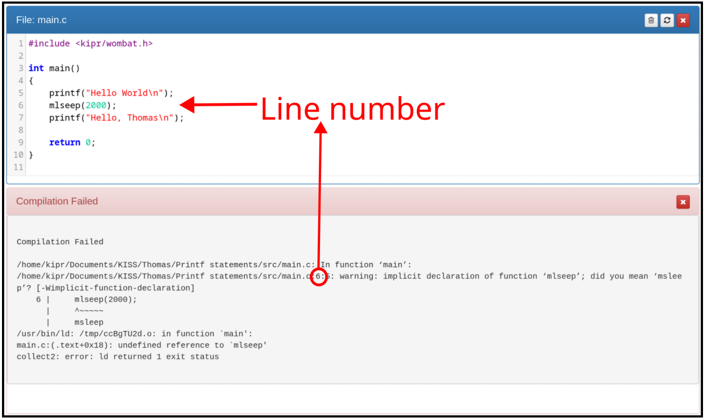
1. Spell `msleep` wrong.
2. Compile and read the error message. What does it say?
3. Proceed to the next slide for help reading the error.

Hints:

- Always correct the top error first, it may correct all the other errors.
- Look at the line number to help find the error.
- Remember if it says before line 10, then it is line 9, or, if line 9 is blank, it could be 8 or 7.

Misspelled Function Error Message

Example error message for `msleep()` misspelling (`mlseep()`).



File: main.c

```
 1 #include <kipr/wombat.h>
 2 
 3 int main()
 4 {
 5     printf("Hello World\n");
 6     mlseep(2000);
 7     printf("Hello, Thomas\n");
 8 
 9     return 0;
10 }
11 
```

Line number

Compilation Failed

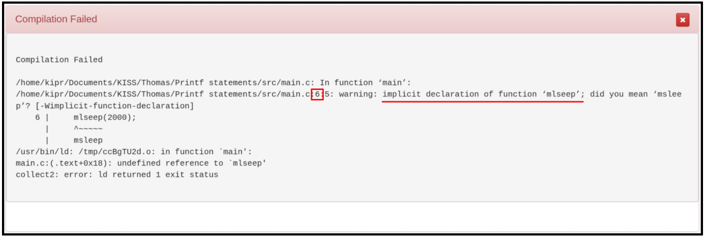
```
Compilation Failed

/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c: In function ‘main’:
/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c:6:5: warning: implicit declaration of function ‘mlseep’; did you mean ‘mslee
p’? [-Wimplicit-function-declaration]
    6 |     mlseep(2000);
      |     ^~~~~~
      |     msleep
/usr/bin/ld: /tmp/ccBgTU2d.o: in function `main':
main.c:(.text+0x18): undefined reference to `mlseep'
collect2: error: ld returned 1 exit status
```

Figure 43: Misspelled function error message

Misspelled Function Error Message

“Implicit declaration of function. . . ” is usually a spelling error.

1. Reading the Error: In this case proceed to the bottom of the errors. Notice it shows you in two places that it is spelled wrong.
2. This error says implicit declaration of function `msleep()`.
3. Fix **one error at a time** and then recompile. It might fix all the errors.
4. Fix your error and compile it before moving to the next slide.



```
Compilation Failed

/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c: In function ‘main’:
/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c:6:5: warning: implicit declaration of function ‘mlseep’; did you mean ‘mslee
p’? [-Wimplicit-function-declaration]
    6 |     mlseep(2000);
      |     ^~~~~~
      |     msleep
/usr/bin/ld: /tmp/ccBgTU2d.o: in function `main':
main.c:(.text+0x18): undefined reference to `mlseep'
collect2: error: ld returned 1 exit status
```

Figure 44: Misspelled function error message

Extra Function Arguments

1. Put a comma in your msleep, like this:

```
msleep(2,000);
```

2. Compile and read the error. What does it say?
3. Proceed to the next slide for help reading the error.

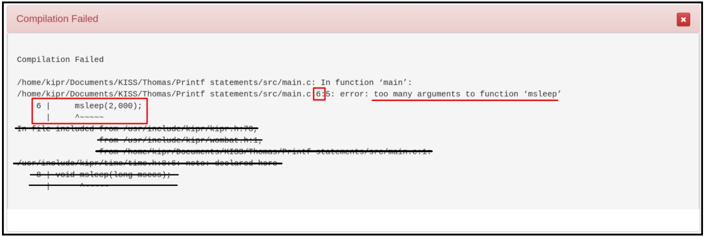
Hints:

- Always correct the top error first, it may correct all the other errors.
- Look at the line number to help find the error.
- Remember if it says before line 10, then it is line 9, or, if line 9 is blank, it could be 8 or 7.

© KISS Institute for Practical Robotics 1993-2026

Extra Function Arguments Error Message

1. Reading the Error: in this case the error is on line 6, “too many arguments to function `msleep`”.
2. `msleep()` has only one argument of time (2000).
3. Unlike normal writing, you cannot use commas in large numbers because the comma indicates that there are two arguments (“2” and “000”).
4. Fix **one error at a time** and then recompile. It might fix all the errors.
5. Fix your error and compile it before moving to the next slide.



```
Compilation Failed

/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c: In function ‘main’:
/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c 6:5: error: too many arguments to function ‘msleep’
    6 |     msleep(2,000);
      |     ^~~~~~
~~In file included from /usr/include/kipr/kipr.h:78,~~
                 ~~from /usr/include/kipr/wombat.h:1,~~
                 ~~from /home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c:1:~~
~~/usr/include/kipr/time/time.h:8:6: note: declared here~~
    ~~8 | void msleep(long msecs);~~
      ~~|      ^~~~~~~~~~
```

Figure 45: Extra function arguments error message

Missing Braces

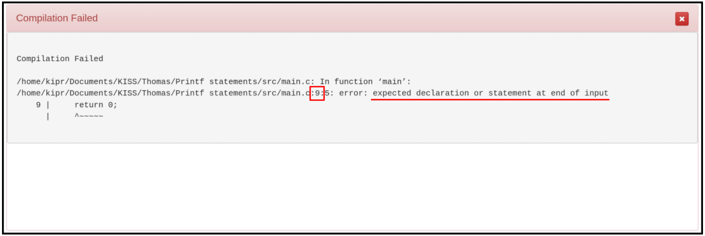
1. Remove a curly brace. { }
2. Compile and read the error message. What does it say?
3. Proceed to the next slide for help reading the error.

Hints:

- Always correct the top error first, it may correct all the other errors.
- Look at the line number to help find the error.
- Remember if it says before line 10, then it is line 9, or, if line 9 is blank, it could be 8 or 7.

Missing Closing Brace Error Message

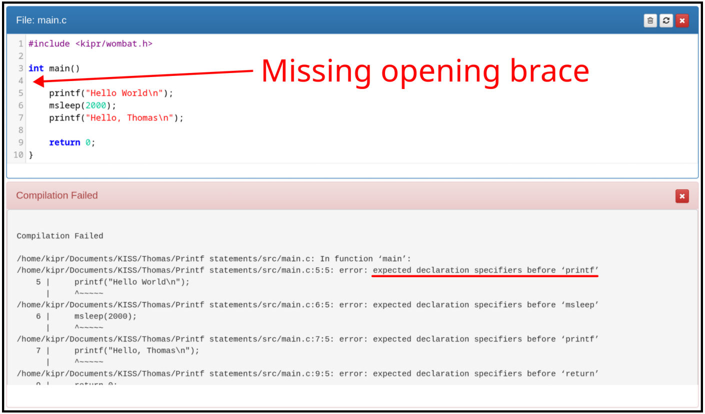
1. Reading the Error: in this case the error is on line 9, expected declaration or statement at end of input.
2. Missing a "}" after return 0; on line 9
3. Fix one error at a time and then recompile. It might fix all the errors.
4. Fix your error and compile it before moving to the next slide.



```
Compilation Failed

/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c: In function ‘main’:
/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c:9:5: error: expected declaration or statement at end of input
    9 |     return 0;
      |     ^~~~~~
```

Figure 46: Missing closing brace error message

Missing Opening Brace Error Message



File: main.c

```
#include <kipr/wombat.h>

int main()

    printf("Hello World\n");
    msleep(2000);
    printf("Hello, Thomas\n");

    return 0;
}
```

Missing opening brace

Compilation Failed

```
Compilation Failed

/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c: In function ‘main’:
/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c:5:5: error: expected declaration specifiers before ‘printf’
    5 |     printf("Hello World\n");
      |     ^~~~~~
/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c:6:5: error: expected declaration specifiers before ‘msleep’
    6 |     msleep(2000);
      |     ^~~~~~
/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c:7:5: error: expected declaration specifiers before ‘printf’
    7 |     printf("Hello, Thomas\n");
      |     ^~~~~~
/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c:9:5: error: expected declaration specifiers before ‘return’
    9 |     return 0;
```

Figure 47: Missing opening brace error message

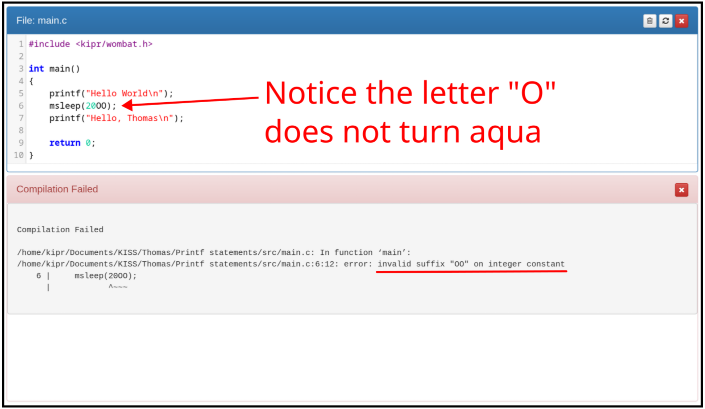
O vs. 0

1. Replace the number “0” with the letter “O”.
2. Compile and read the error message. What does it say?
3. Proceed to the next slide for help reading the error.

Hints:

- Always correct the top error first, it may correct all the other errors.
- Look at the line number to help find the error.
- Remember if it says before line 10, then it is line 9, or, if line 9 is blank, it could be 8 or 7.

O vs. 0 Error Message



```c
#include <kipr/wombat.h>

int main()
{
    printf("Hello World\n");
    msleep(20OO);
    printf("Hello, Thomas\n");

    return 0;
}
```

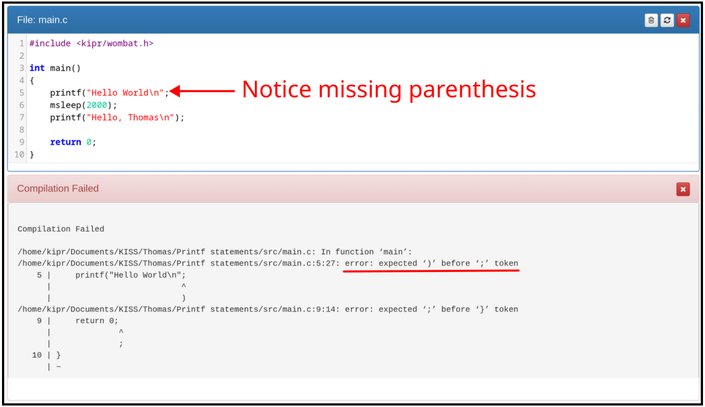
Notice the letter "O" does not turn aqua

```
Compilation Failed

/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c: In function ‘main’:
/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c:6:12: error: invalid suffix "OO" on integer constant
    6 |     msleep(20OO);
      |              ^~~~
```

Figure 48: O vs. 0 error message

More Common Errors

Missing Parenthesis Error Message



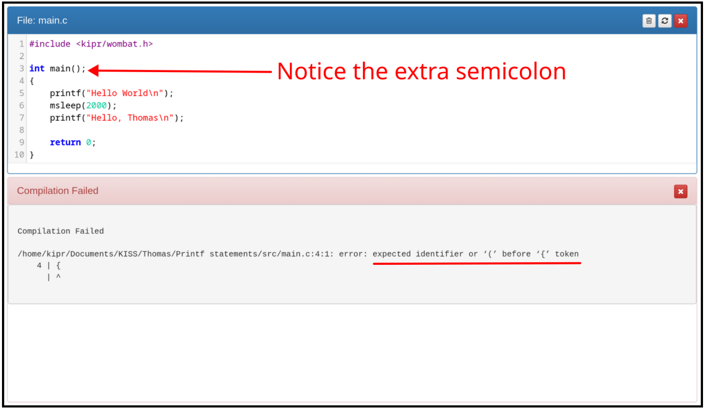
```
 1 #include <kipr/wombat.h>
 2
 3 int main()
 4 {
 5     printf("Hello World\n");
 6     msleep(2000);
 7     printf("Hello, Thomas\n");
 8
 9     return 0;
10 }
```

Notice missing parenthesis

```
Compilation Failed

/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c: In function ‘main’:
/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c:5:27: error: expected ‘)’ before ‘;’ token
    5 |     printf("Hello World\n";
      |                           ^
      |                           )
/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c:9:14: error: expected ‘;’ before ‘}’ token
    9 |     return 0;
      |              ^
      |              ;
   10 | }
      | ~
```

Figure 49: Missing parenthesis error message

Extra Semicolon Error Message



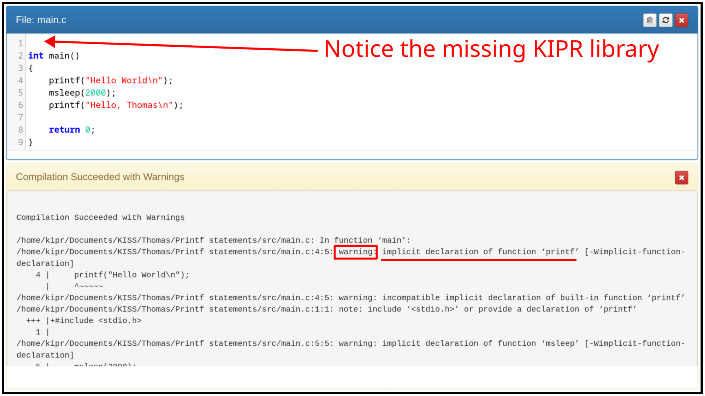
```
#include <kipr/wombat.h>

int main();
{
    printf("Hello World\n");
    msleep(2000);
    printf("Hello, Thomas\n");

    return 0;
}
```

Notice the extra semicolon

```
Compilation Failed

/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c:4:1: error: expected identifier or ‘(’ before ‘{’ token
    4 | {
      | ^
```

Figure 50: Extra semicolon error message

Missing Library Error Message



```c

int main()
{
    printf("Hello World\n");
    msleep(2000);
    printf("Hello, Thomas\n");

    return 0;
}
```

Notice the missing KIPR library

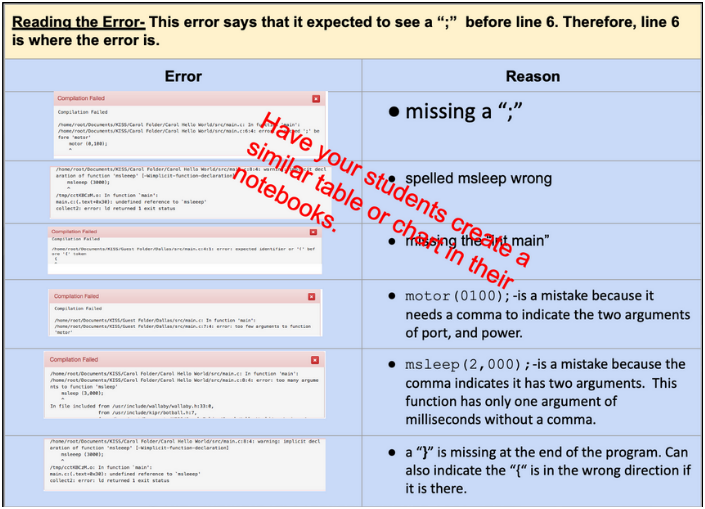
Compilation Succeeded with Warnings

```
Compilation Succeeded with Warnings

/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c: In function ‘main’:
/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c:4:5: warning: implicit declaration of function ‘printf’ [-Wimplicit-function-
declaration]
    4 |     printf("Hello World\n");
      |     ^~~~~~
/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c:4:5: warning: incompatible implicit declaration of built-in function ‘printf’
/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c:1:1: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
  +++ |+#include <stdio.h>
    1 |
/home/kipr/Documents/KISS/Thomas/Printf statements/src/main.c:5:5: warning: implicit declaration of function ‘msleep’ [-Wimplicit-function-
declaration]
    5 |     msleep(2000);
```

Figure 51: Missing library error message

Closing Notes

Closing Notes



Reading the Error- This error says that it expected to see a ";" before line 6. Therefore, line 6 is where the error is.

Error	Reason
Compilation Failed Compilation Failed /home/root/Documents/KISS/Carol Folder/Carol Hello World/src/main.c: In function 'main': /home/root/Documents/KISS/Carol Folder/Carol Hello World/src/main.c:6:4: error: expected ';' before 'motor' motor (0,100); ^	● missing a ";"
/home/root/Documents/KISS/Carol Folder/Carol Hello World/src/main.c:8:4: warning: implicit declaration of function 'msleeep' [-Wimplicit-function-declaration] msleeep (3000); ^ /tmp/cctKBCzM.o: In function `main': main.c:(.text+0x30): undefined reference to `msleeep' collect2: error: ld returned 1 exit status	● spelled msleep wrong
Compilation Failed Compilation Failed /home/root/Documents/KISS/Guest Folder/Dallas/src/main.c:4:1: error: expected identifier or '(' before '{' token { ^	● missing the "int main"
Compilation Failed Compilation Failed /home/root/Documents/KISS/Guest Folder/Dallas/src/main.c: In function 'main': /home/root/Documents/KISS/Guest Folder/Dallas/src/main.c:7:4: error: too few arguments to function 'motor'	● motor(0100); -is a mistake because it needs a comma to indicate the two arguments of port, and power.
Compilation Failed /home/root/Documents/KISS/Carol Folder/Carol Hello World/src/main.c: In function 'main': /home/root/Documents/KISS/Carol Folder/Carol Hello World/src/main.c:8:4: error: too many arguments to function 'msleep' msleep (3,000); ^ In file included from /usr/include/wallaby/wallaby.h:33:0, from /usr/include/kipr/botball.h:7,	● msleep(2,000); -is a mistake because the comma indicates it has two arguments. This function has only one argument of milliseconds without a comma.
/home/root/Documents/KISS/Carol Folder/Carol Hello World/src/main.c:8:4: warning: implicit declaration of function 'msleeep' [-Wimplicit-function-declaration] msleeep (3000); ^ /tmp/cctKBCzM.o: In function `main': main.c:(.text+0x30): undefined reference to `msleeep' collect2: error: ld returned 1 exit status	● a "}" is missing at the end of the program. Can also indicate the "{" is in the wrong direction if it is there.

Have your students create a similar table or chart in their notebooks.

Figure 52: Sample Chart

Closing Notes

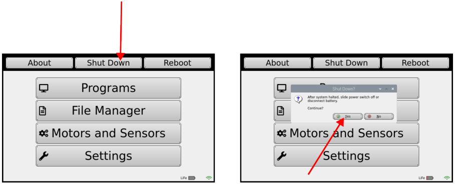
Hints for Teachers:

- Help students read the error, but do not give them the answer.
- Revisit debugging as students are struggling to read errors.
- Create a programs with errors and have students debug them.
- Create worksheets for students to debug.

Reminder: Powering Off Your Wombat

Reminder: Powering Off Your Wombat

1. From the Home Screen, press “Shutdown.”
2. Press “Yes” to confirm.
3. After shutting down from the Home Screen, flip the power switch to off.
4. Unplug the battery, being careful to only grab the yellow connectors, **not** the wires.



After system halted, slide power switch off or disconnect battery.

Continue?

Figure 53: Wombat shutdown procedure

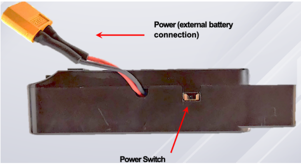


Figure 54: Wombat power switch