

Writing Your First Program

- **Key Concepts**
 - Students will learn about the C program language, functions `printf ()`, `msleep ()`, debugging, and connection failed
- **Pacing**
 - 30-45 minutes per Activity

Table Of Contents

[Writing Your First Program](#)

[Connecting Your Computer to Your Wallaby](#)

[Connecting your Computer to the Wallaby Using Wi-fi](#)

[Accessing KIPR Software Suite \(Using Wi-Fi\)](#)

[Connecting to Your Wallaby Using the USB Cable](#)

[Accessing Kipr Software Suite \(Using USB Cable\)](#)

[Welcome to the Software Suite](#)

[Making a Folder](#)

[Add a Project](#)

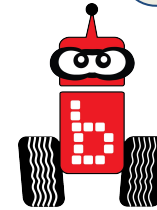
[Name Your Project](#)

[Compiling Your First Project](#)

[Running Your First Program](#)

[The C Template](#)

[Learning About The C Template](#) (2 Slides)



Click on me to return
to Table of Contents

Table of Contents (Cont.)

Template Components

Functions

Blocks of Code

Programming Statements

Terminating Statements

Ending the main function

Program speed

Curly Braces

Learning About Psuedocode (2 Slides)

Using the `printf()` Function- Activity 2 (12 Slides)

Extensions

Making Observations-Activity 3 (3 Slides)

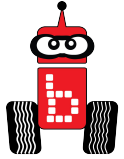
Introduction `msleep()` Function- Activity 4 (10 Slides)

Debugging-Activity 5 (5 Slides)

Connection Failed Errors

Assessment Rubrics

Writing Your First Program



Goals:

- Students will use a “Hello, World” template to write and compile their first program to the KIPR Link
- Students will develop an understanding of the components that make the C Template: Hello, World.
- Students will develop an understanding of the `msleep()` function.
- Students will learn the importance of and how to debug their programs.

Standards:

Common Core State Standards Math Practices

CCSSMP1: Make sense of problems and persevere in solving them

CCSSMP2: Reason abstractly and quantitatively

CCSSMP4: Model with mathematics

CCSSMP6: Attend to precision

CCSSMP8: Look for and express regularity in repeated reasoning

Next Generation Science and Engineering Practice

1: Asking questions and defining problems

2: Developing and using models

3: Planning and carrying out investigations

4: Analyzing and interpreting data

5: Using mathematics and computational thinking

6: Constructing explanations and designing solutions

7: Engaging in argument from evidence obtaining, evaluating, and communicating information

Standards Continued

Standards Continued:

2016 ISTE Standards

Empowered Learner

1c: Students use technology to seek feedback that informs and improves their practice and to demonstrate their learning in a variety of ways.

1d: Students understand the fundamental concepts of technology operations, demonstrate the ability to choose, use and troubleshoot current technologies and are able to transfer their knowledge to explore emerging technologies.

Knowledge Constructor

3d: Students build knowledge by actively exploring real-world issues and problems, developing ideas and theories and pursuing answers and solutions.

Innovative Designer

4a: Students know and use a deliberate design process for generating ideas, testing theories, creating innovative artifacts or solving authentic problems.

4b: Students select and use digital tools to plan and manage a design process that considers design constraints and calculated risks.

4c: Students develop, test and refine prototypes as part of a cyclical design process.

4d: Students exhibit a tolerance for ambiguity, perseverance and the capacity to work with open-ended problems.

Computational Thinker

5a: Students formulate problem definitions suited for technology-assisted methods such as data analysis, abstract models and algorithmic thinking in exploring and finding solutions.

Writing Your First Program

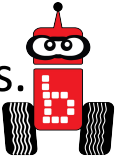
Objective: Students will use a “Hello, World” template to write and compile their first program on the KIPR Wallaby

Materials: Built robot, Computer, (usb cable optional)

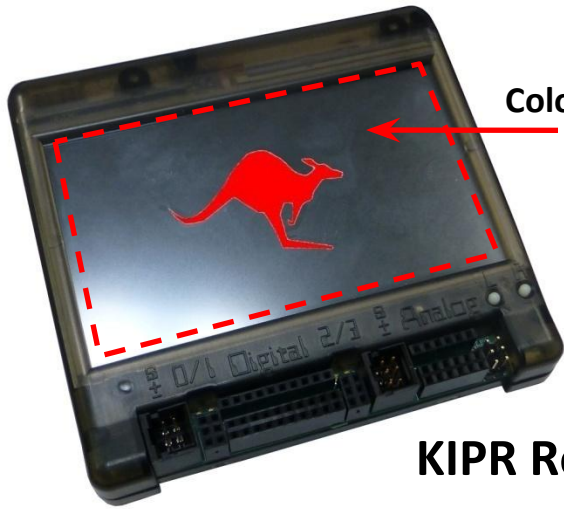
Lesson:

1. Use the [Thinking Notes Strategy](#) to read and discuss the following slides.

[Connecting to Your Wallaby Wi-fi](#)
[Loading the Starting Web Page](#)
[Connecting Your Wallaby USB Cord](#)
[Welcome to the Software Suite](#)
[Learning about the Software Suite](#)
[Making a Folder](#)
[Add a Project](#)
[Name Your Project](#)
[Compiling Your First Project](#)
[Running Your First Program](#)
[Connecting the Link](#)
[Compiling Your Program](#)
[Compile Succeed](#)
[Running Your Program](#) (3 Slides)

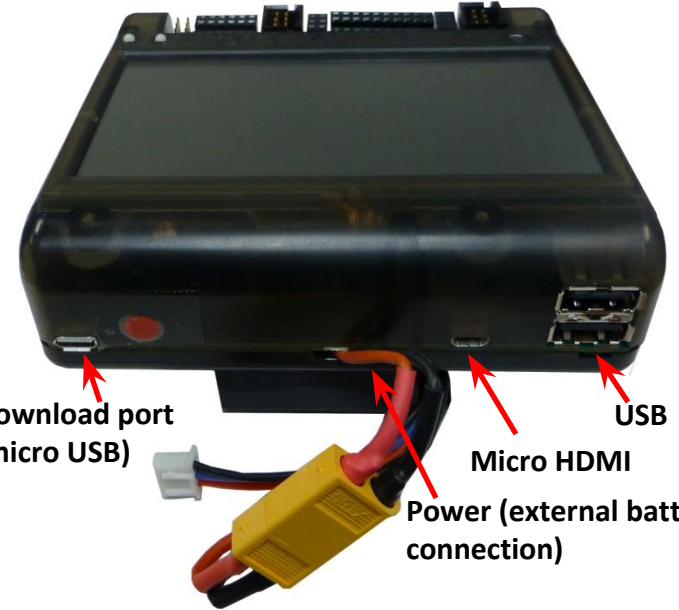


Controller Guide



Color Touch Screen

KIPR Robotics Controller Wallaby



Download port
(micro USB)

USB

Micro HDMI

Power (external battery
connection)



2 Servo
Motor Ports
(Port # 0 & 1)

2 Motor Ports
(Port # 0 & 1)

10 Digital
Sensor Ports
(Port # 0 - 9)

2 Motor Ports
(Port # 2 & 3)

2 Servo
Motor Ports
(Port # 2 & 3)

6 Analog
Sensor Ports
(Port # 0 - 5)



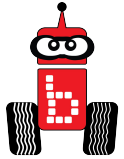
Power Switch

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 Wallaby off or unplug it from the battery while charging
 - Charge in a cool, open area away from flammable materials.



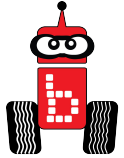
Important Information



All connections are as follows:

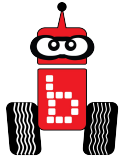
- Yellow to Yellow
- White small to White small (to the charger, may be white to black)
- Black to Black

Wallaby Power



- The KIPR Robotics Controller – Wallaby, uses an external battery pack for power.
 - It will void your warranty to use a battery pack with the Wallaby that hasn't been approved by KIPR.
- When your Wallaby is not in use please **be sure to do the following:**
 - TURN YOUR WALLABY OFF
 - UNPLUG THE BATTERY FROM THE WALLABY
 - Leaving your battery plugged in and your Wallaby 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 and purchase a replacement - 405-579-4609.

Connecting your Computer to your Wallaby

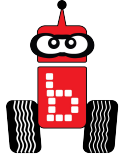


There are 2 ways to connect your wallaby controller to your computer in order to access the KIPR Software Suite.

- 1. Wi-Fi**
- 2. USB Cord**

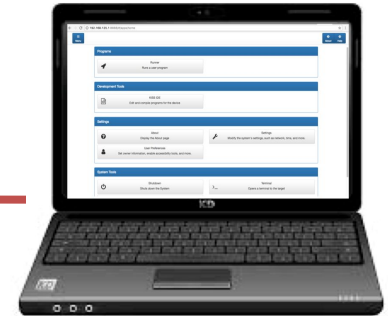
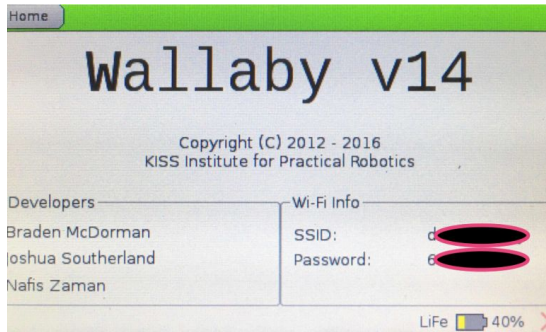
We are going to connect with Wi-Fi first!

Connecting Your Computer to the Wallaby (using Wi-Fi)



Connect the **Wallaby** to your Browser device via Wi-Fi

1. Turn on the Wallaby with the **black switch on the side**.
2. Every wallaby has a unique number/name.
3. To find the number of your wallaby click “about” on your controller.
4. Look at your Wi-Fi info and see the SSID: that is your wallaby number and wallaby password.

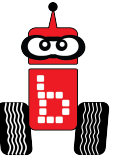


5. Go to the **Wi-Fi icon on your computer** and click on your wallaby number and add your password (this may take a few minutes). You will see a blue LED light.

Not recommended at Tournament

***** You may need to take Wi-Fi preference off of selecting your school or another site, so it will not keep reverting to it.**

Accessing KIPR Software Suite (using Wi-Fi)



1. Launch your web browser (such as Chrome or Firefox).
2. Copy this IP address into your browser's address bar followed by ":" and port number 8888; e.g.,

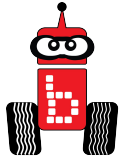
192.168.125.1:8888



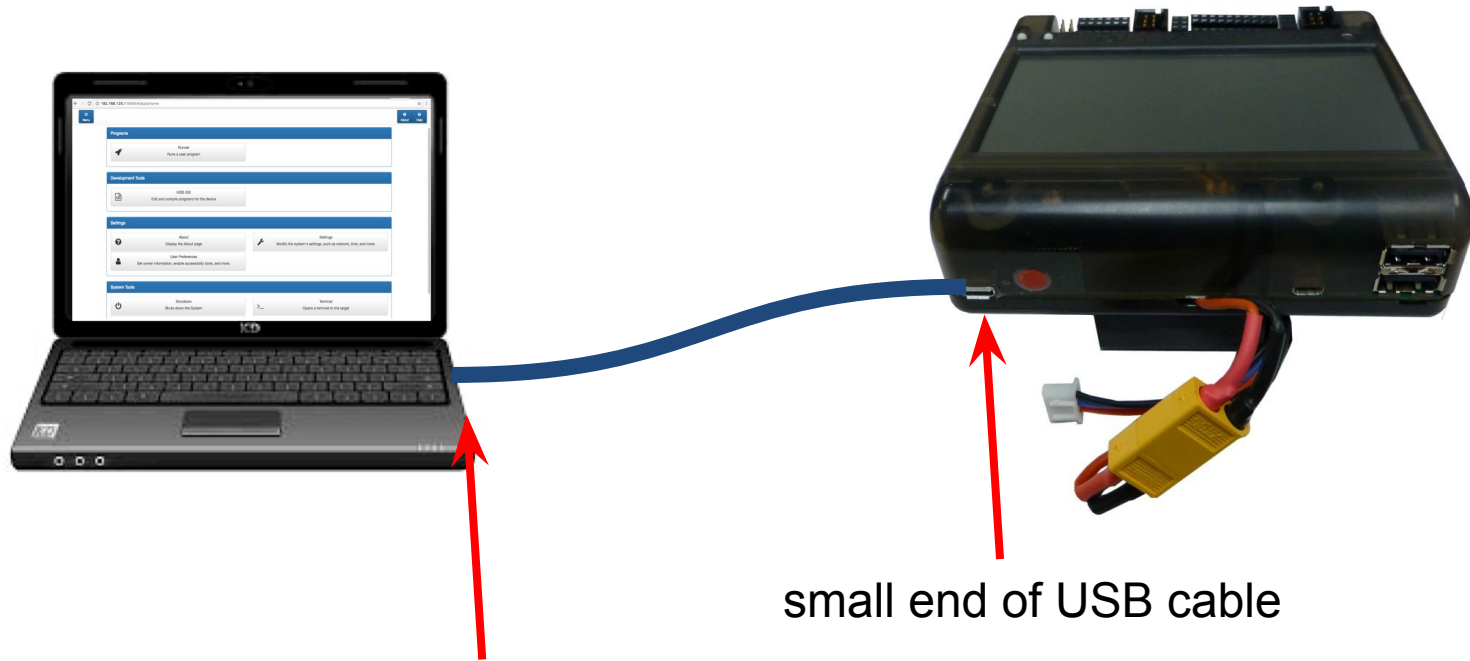
3. The KISS Software Suite will now come up in your browser.
4. You may use a computer, tablet or even a smart phone.

Connecting to Your Wallaby

Using the USB Cable



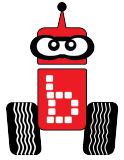
1. Connect your device to the Wallaby with a USB cable



small end of USB cable

large end of USB cable

Accessing the KIPR Software Suite Using the USB Cable

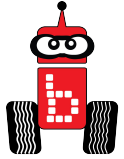


1. Launch your web browser (such as Chrome or Firefox).
2. Copy this IP address into your browser's address bar followed by ":" and port number 8888; e.g.,

192.168.1241:8888

└──────────┬──────────┘ └──┬──┘
IP address Port #

4. The KISS Software Suite will now come up in your browser.
5. You may use a computer, tablet or even a smart phone as long as they can connect through Wi-Fi.

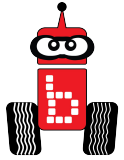


Wallaby Power Down

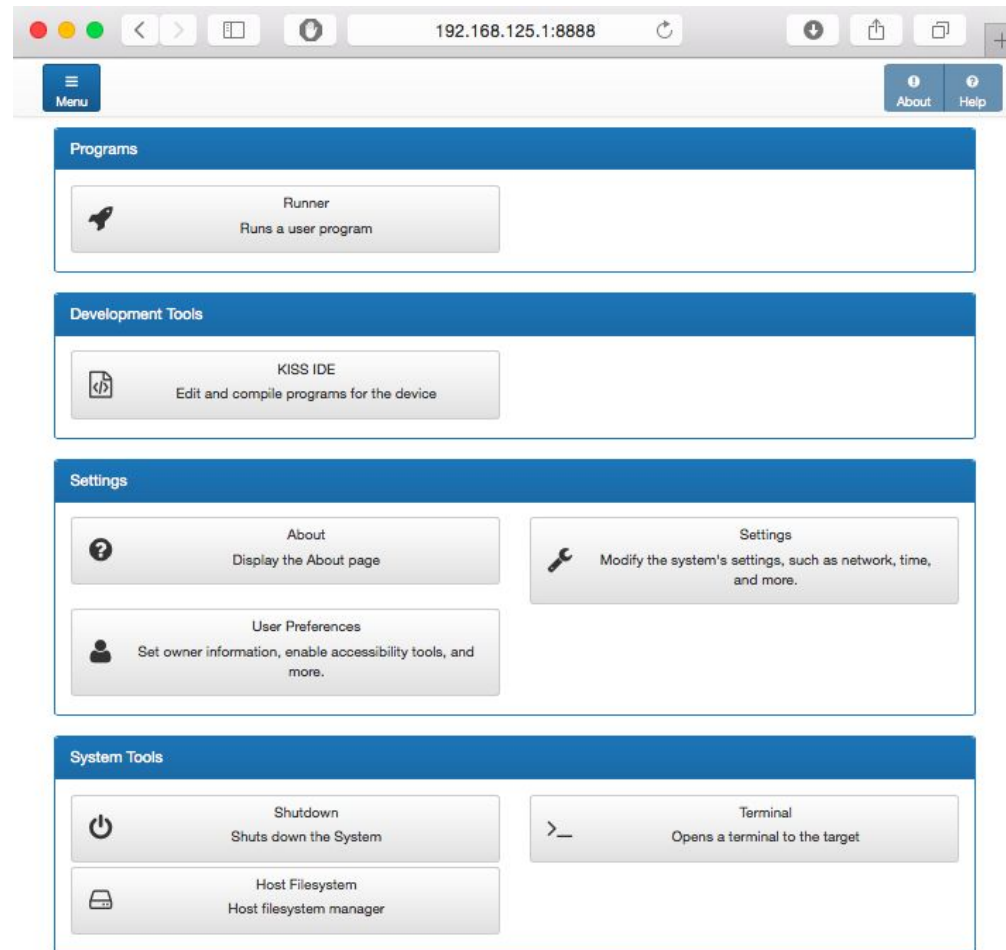
- From the Software Suite select "Shutdown"
 - Select "Yes"
- From the Wallaby Home Screen press "Shutdown"
 - Select "Yes"
- Go to your Wallaby screen and check to see if it is "halted"
- Slide the power switch to off
- Unplug the battery being careful not to pull on the wires

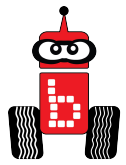
***If you powered down, go back to slide 10 to reconnect.

Welcome to the Software Suite



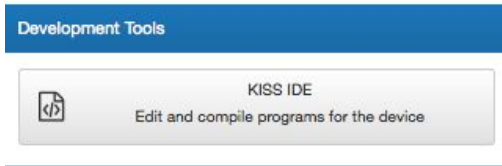
- To make it easier for you to learn and use a programming language, KIPR provides a web-based **Software Suite**, which will allow you to write and compile source code using the **C programming language**.
- The development package will work with almost any web browser **except Internet Explorer**.



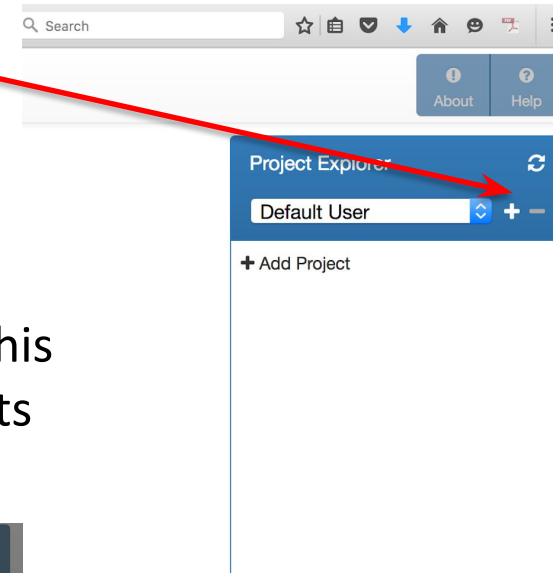


Making a Folder

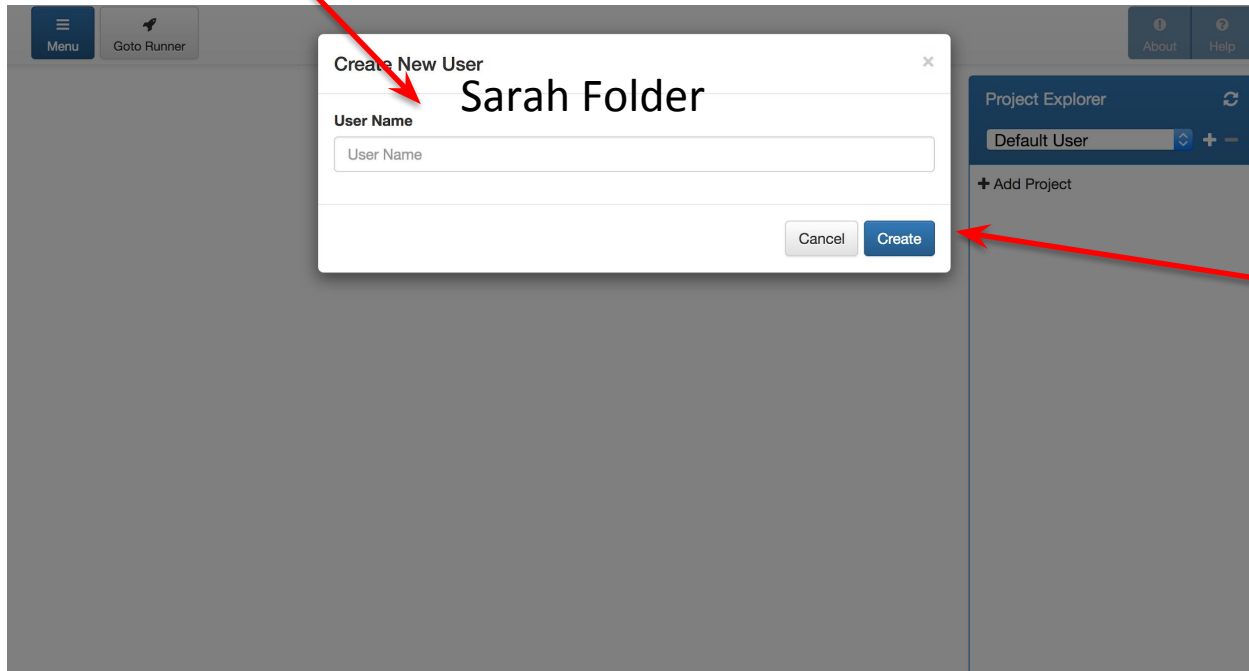
1. Click **KISS IDE**



2. Under Project explore you have options to add a user by clicking the **+** sign.



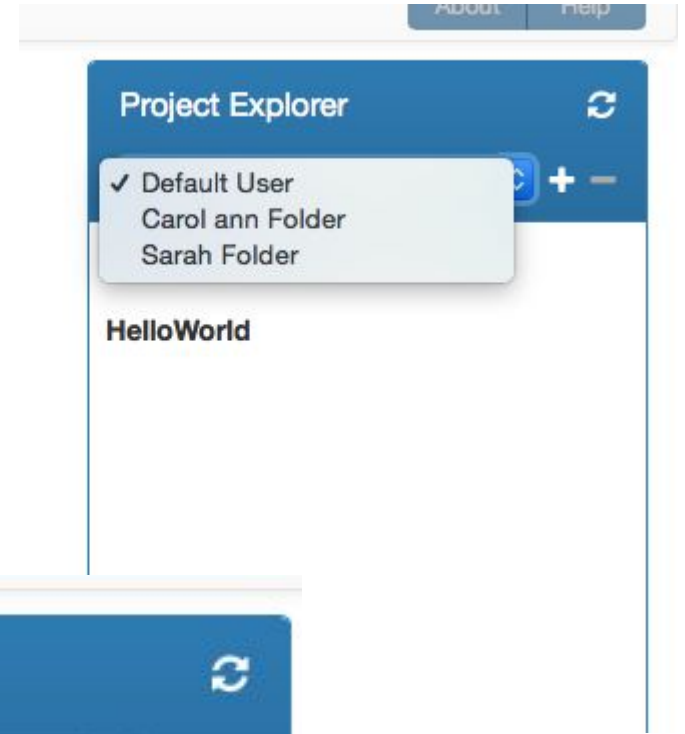
3. Name your New User, "Student name and folder" . This is similar to a folder you will put all your different projects into. (or "1 Sarah folder")



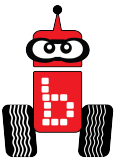
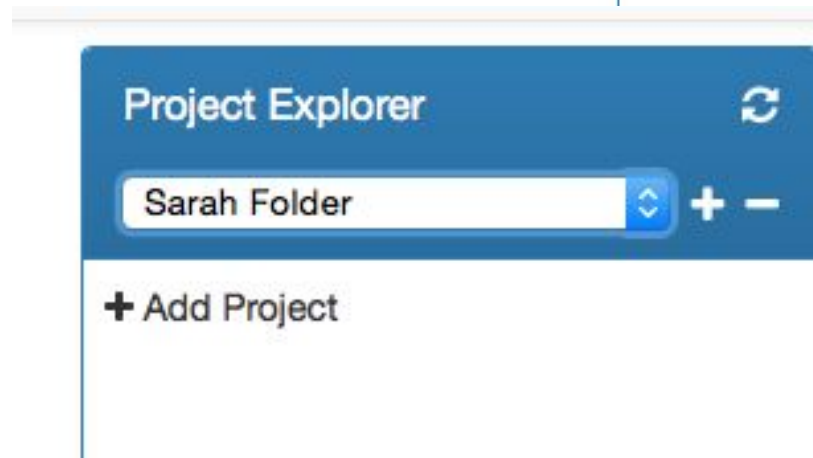
4. Click create

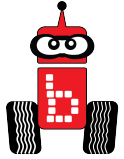
Add a Project

5. Go back to “Project Explorer” and select the **User Name** you created from the drop down. This is the folder you created



6. Click “+Add Project”
You are adding a project to your folder





Name your project

Give your *project* a **descriptive name**

- **Note:** you will have a lot of student's projects, so maybe use their first name followed by the name of the activity.
- Do not put any special characters or periods, etc. on it.

Create New Project

Project name

Sarah Hello World

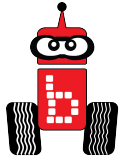
Source file name

main.c

Cancel Create

Click Create

Compiling Your First Project

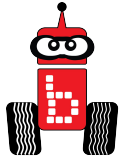


1. Make sure you are in your user name/folder. Find your project name (in this case Sarah Hello world).
2. Click “Compile” to translate your program to **machine language**.
Note: the “**Compile**” button sends the program to the robot.
3. Go to next slide

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

This will go away with update 23. Update software by going to www.kipr.org, under software/hardware tab

Compilation succeeded will appear!



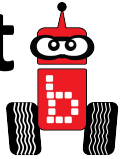
File: hello world.c

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

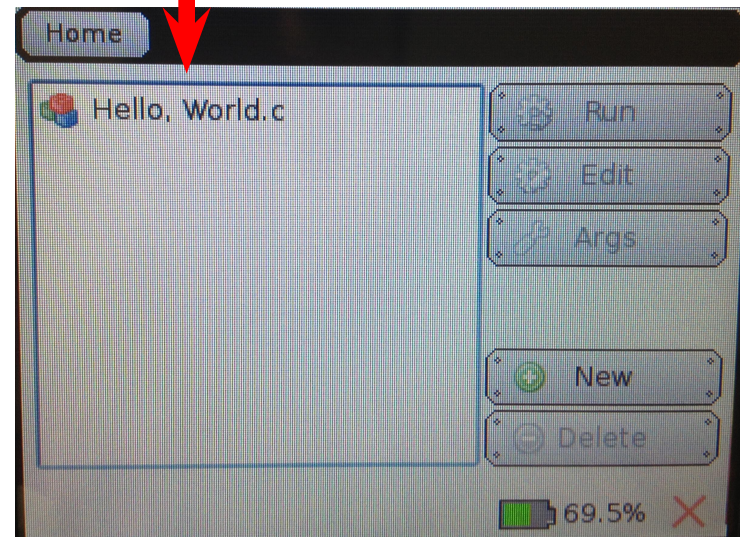
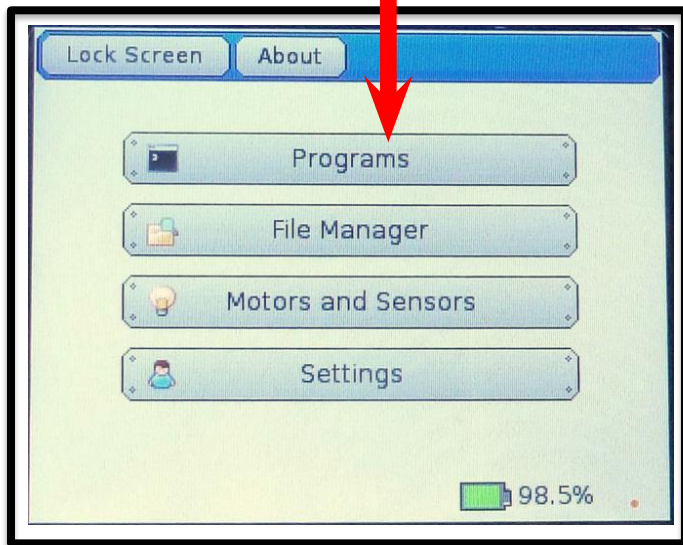
Compilation succeeded

Compilation succeeded

Running your Program on Your Robot



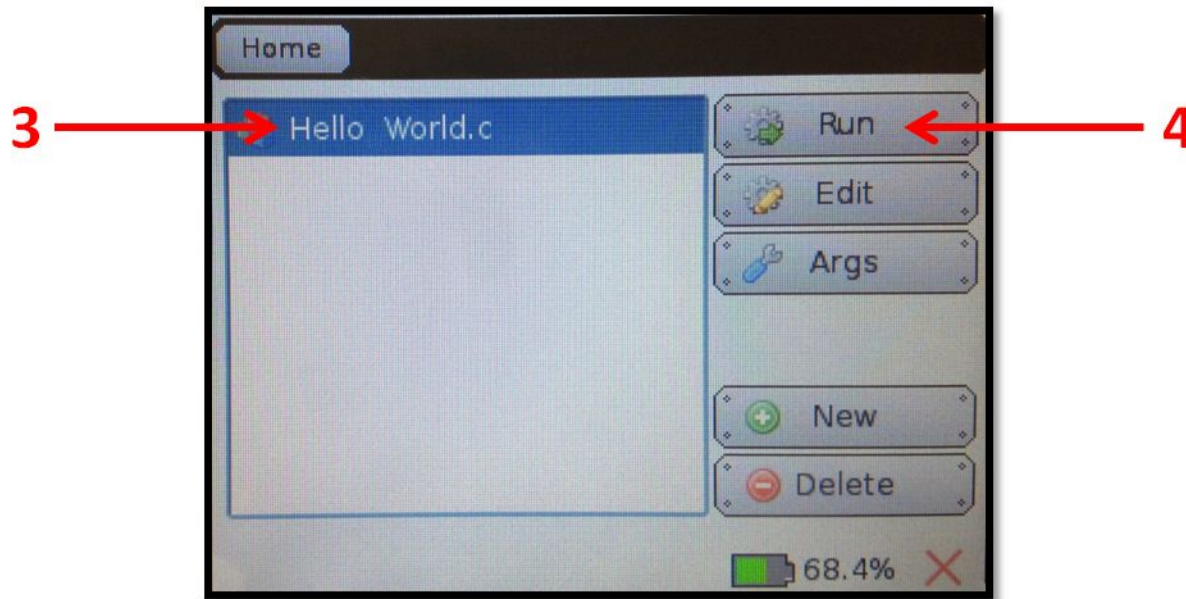
1. Press (touch) the **“Programs”** button on the **KIPR Wallaby**.
2. This will take you to a **list of programs** currently on your Wallaby.



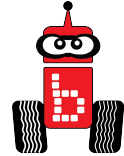
Go to the next slide

Running your program (continued)

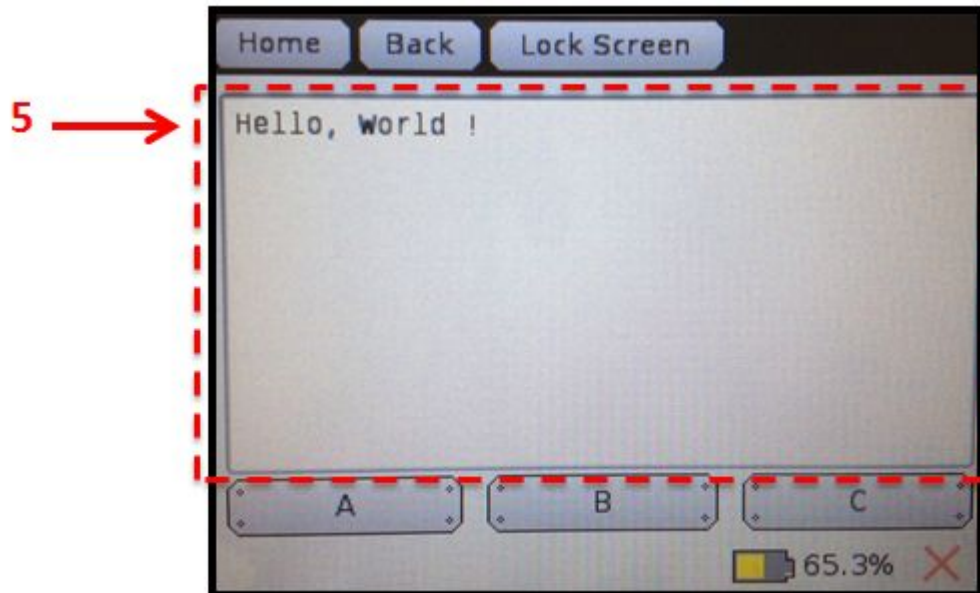
3. Select (touch) the **name of the program** you want to run.
 - **Note:** in the example below, this is “Hello, World.c”.
4. Press (touch) the **“Run” button** on the Wallaby.



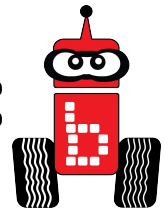
Running Your Program on the Wallaby (Cont.)



5. The phrase “**Hello, World!**” will appear on your *Wallaby screen*.



Learning About The C Template: Hello, World!



Objectives: Students will develop an understanding of the components that make the C Template: Hello, World

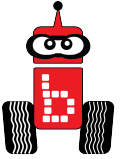
Materials:

Resource slides ([Functions](#), [Blocks of Code](#), [Programming Statements](#), [Terminating Statements](#), [Ending the main function](#), [Program speed](#), [Curly Braces](#)), science/engineering/robotics/digital notebooks, writing utensil

Lesson:

1. Provide student groups with a hand out of each resource slide.
2. Have students discuss within their groups and then share out to the class as a whole or use other [instructional strategies](#).
3. Students record the key information in their science/engineering/robotics/digital notebooks.
4. Identify different components of the [C Template](#).

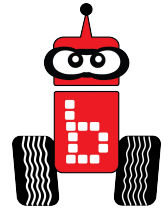
Learning About The C Template: Hello, World!



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

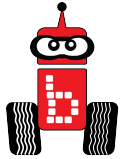
int main ()
{
    printf("Hello World!\n");
return 0:
}
```

Learning About The C Template: Hello, World! Components



1. [Functions](#)
2. [Blocks of Code](#)
3. [Programming Statements](#)
4. [Terminating Statements](#)
5. [Ending the main function](#)
6. [Program speed](#)
7. [Curly Braces](#)

Learning About The C Template:



Hello, World! - Functions

The **KISS IDE** contains a large library of functions you can use to program your robots. A function is like a title to an instruction book. When you call the function it does all of the commands in the book.

A “**function**” defines a list of actions to take.

A function is like a **recipe** for baking a cake.

When you “**call**” (use) the function, the program follows the instructions and bakes the cake.

A `clean_house()` function could mean vacuum, dust, mop, change the linens, wash the windows, etc... all the commands specified in the function are executed.

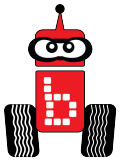
```
// Created on Thu January 10 2013
```

This is the “**main function**”.

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

When you run your program, the **main function** is executed.

Learning About The C Template: Hello, World!



Function Tent

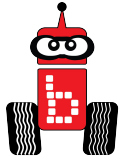
Until you are familiar with the functions that you will be using while programming, use your “Function Tent” for easy reference. Copy and paste is also very helpful.

[Printable Function Tent](#)

```
printf("text\n");
motor(port#, % power);
msleep (# milliseconds);
ao();
enable_servos();
set_servo_position(port#, position);
disable_servos ();
digital(port #);
analog (port#);
analog_et(port#);
camera_open();
camera_close();
camera_update();
get_object_count();
get_object_column(column,row);
get_object_row (row,object);
```

```
//Prints text to display
//Turns motor on at % power specified
//Program waits specified number of milliseconds
//All off, turns all motor ports off
//Turns servo ports on
//Moves servo in specified port to a set position
//Turns off servo ports
//Refers to a specific digital port #
//Refers to a specific analog port #
//Get an analog ET sensor reading
//Opens camera
//Closes camera
//retrieves current image
//Retrieves the number of object specified by channel
//retrieves center column coordinate value
// retrieves center row coordinate value
```

Learning About The C Template:



Hello, World! Blocks of Code

```
int main()
```

Block of code

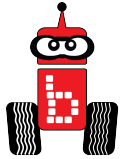


```
{  
    printf("Hello, World!\n");  
    return 0;  
}
```

Learning About The C Template:

Hello, World!

Programming Statements



```
// Created on Thu January 10 2013
```

```
int main()
```

```
{
```

```
Statement #1 → printf("Hello, World!\n");
```

```
Statement #2 → return 0;
```

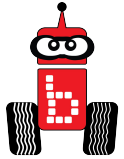
```
}
```

Inside the “**block of code**” (between the { and } braces), we write lines of code called “**programming statements**”.

Each **programming statement** is an action to be executed by the computer (or robot) **in the order that it is listed.**

There can be any number of programming statements.

Learning About The C Template:



Hello, World!

Terminating Statement

```
// Created on Thu January 10 2013
```

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

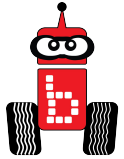
Terminating statements end each **programming statement**. Use a **semi-colon** ; (*unless* it is followed by a new **block of code**) to end the programming statement.

This is similar to an **English sentence**, which ends with a ***period***.

If an English statement is missing a period, then it is a run-on sentence.

A semi-colon is similar to an “enter” or “return” key on your keyboard, it tells the computer to go to the next line.

Learning About The C Template:



Hello, World!

Ending the `main ()` function

```
// Created on Thu January 10 2013
```

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

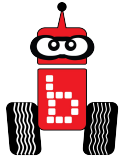


The `return` statement is the last line before the `}` brace.

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`.

Learning About The C Template:



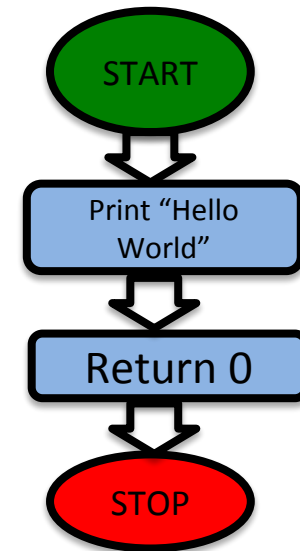
Hello, World!

Program Speed

Computers read a program just like you read a book, they start at the top and go to the bottom. Computers read incredibly fast- 800 MILLION lines per second!



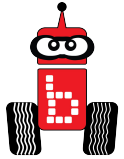
```
int main()  
{  
    printf("Hello, World!\n");  
    return 0;  
}
```



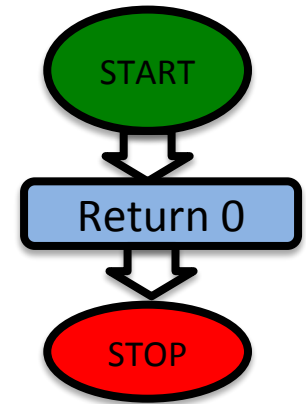
* The flow chart goes from top to bottom, left to right, line by line (just like you read). The flowchart is pseudocode.

Learning About The C Template:

Hello, World! Curly Braces



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



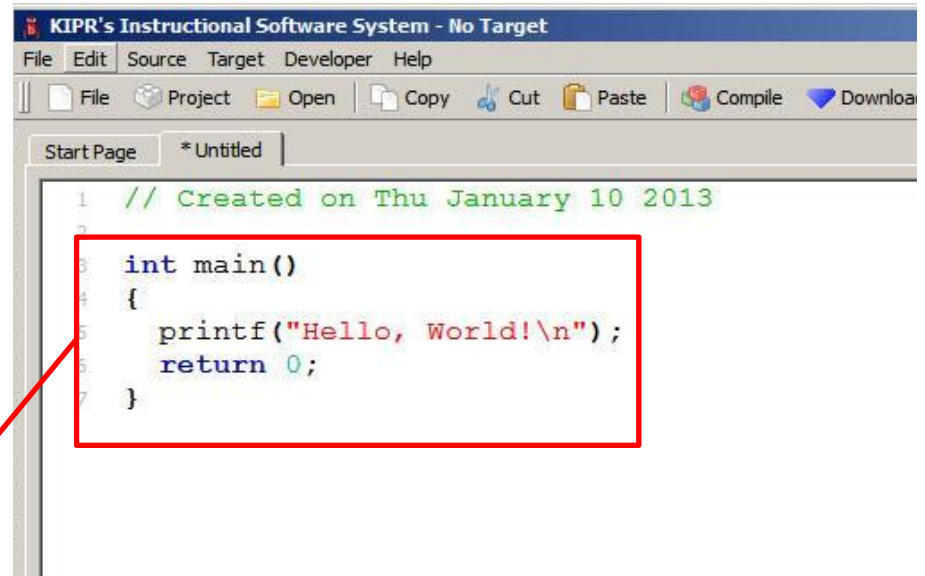
The curly braces organize everything you want the program to do (execute) when the computer comes to the last curly brace it will end the main program.

Looking at a Program Colors

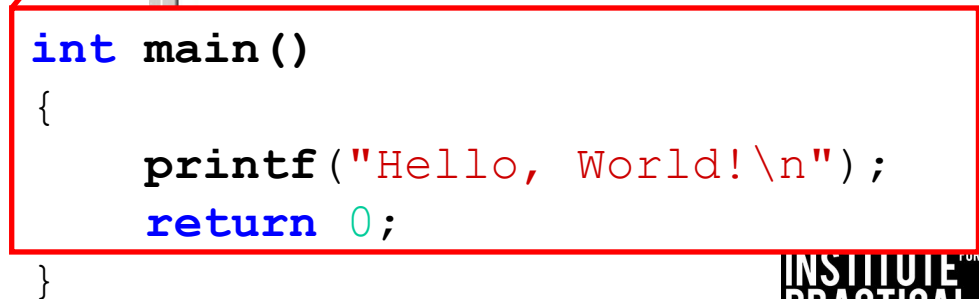
The KISS IDE highlights parts of a program to make it easier to read

- By default, the KISS IDE colors your code and adds line numbers

- Comments appear in green
- Key words appear in bold blue
- Text strings appear in red
- Numbers appear in aqua

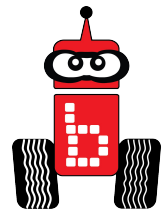


```
KIPR's Instructional Software System - No Target
File Edit Source Target Developer Help
File Project Open Copy Cut Paste Compile Download
Start Page *Untitled
1 // Created on Thu January 10 2013
2
3 int main()
4 {
5     printf("Hello, World!\n");
6     return 0;
7 }
```



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

Learning About Comments (Pseudocode)



1. Read and discuss with a partner this slide and the next slide to understand Pseudocode.

Pseudocode means "false code".

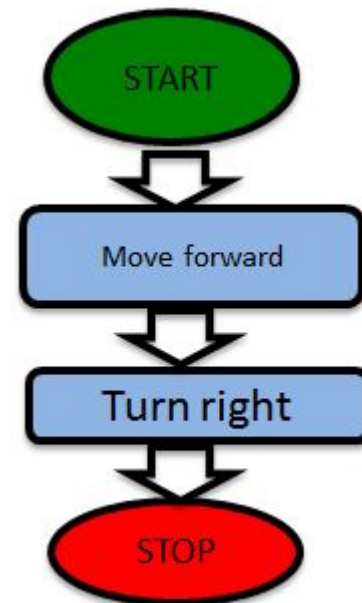
Pseudocode can be used to comment on what you expect your program to cause your robot to do, but that might not be what it will actually do.

//1. Move forward

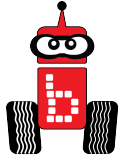
//2. Turn Right

//3. Stop

Why would it be important to create pseudocode?



Learning About Comments (Pseudocode)



- Comments as pseudocode 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.
- To make a comment two slash marks must be typed first: //

Complete Comment: //Prints Hello World to screen

- When you compile the program it will not execute the comment, but you can see it.

Sample Program

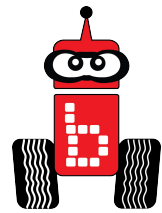
```
int main()  
{  
    printf("Hello, World!\n"); // Prints Hello World to screen  
    return 0;  
}
```

Why would you put a
comment here?



Add a Comment

Activity 1



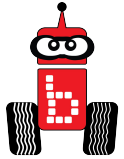
1. Add the `//Print Hello world to screen` comment to the program
 - Just like using a word processing program you can click to set your cursor and then use return to make space for the comment.
 - Type the comment into your program. The comment can go on the line before the printf function or on the same line as the function.
2. Continue to the next slide.

On same line as the
function

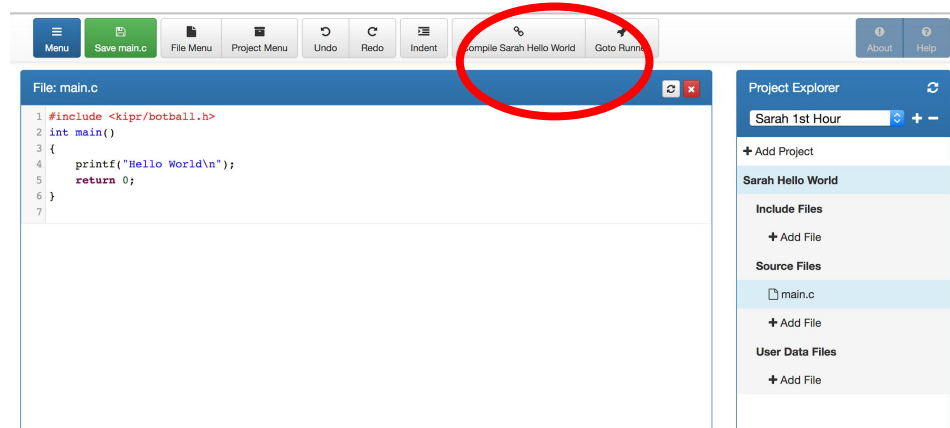
```
int main()  
{  
    printf("Hello, World!\n"); //Print Hello World to screen  
    return 0;  
}
```

Go to the next slide

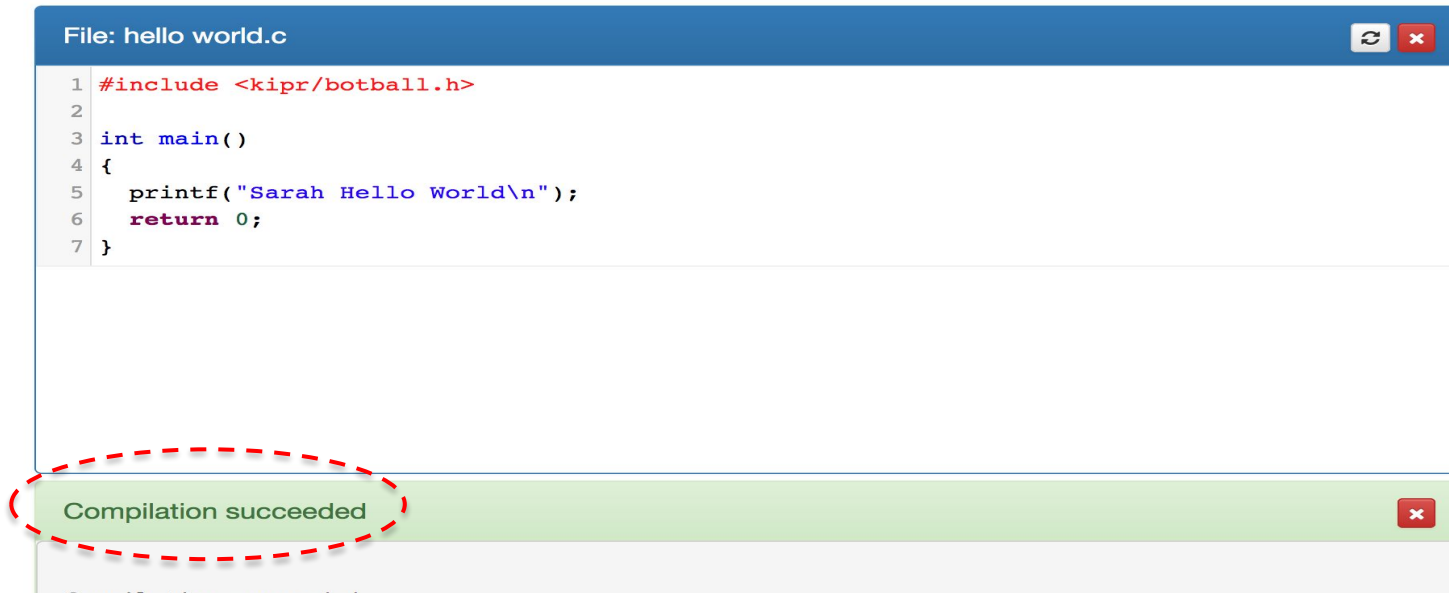
Add a Comment (cont.)



Compile

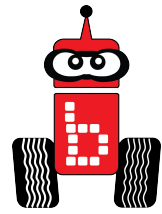


Watch to see if it Succeeded!



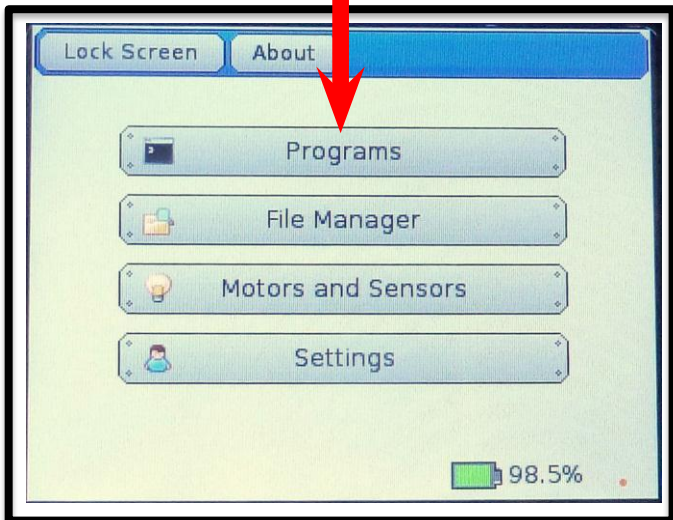
Go to the next slide

Add a Comment (cont.)



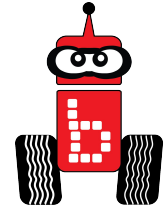
Running your program on the Wallaby

1. Press (touch) the **“Programs”** button on the **KIPR Wallaby**.
2. This will take you to a **list of programs** currently on your Link.

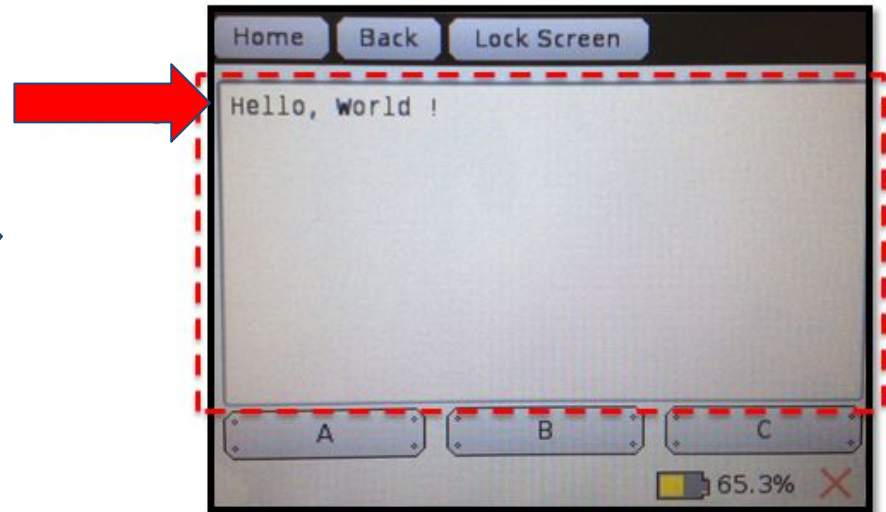
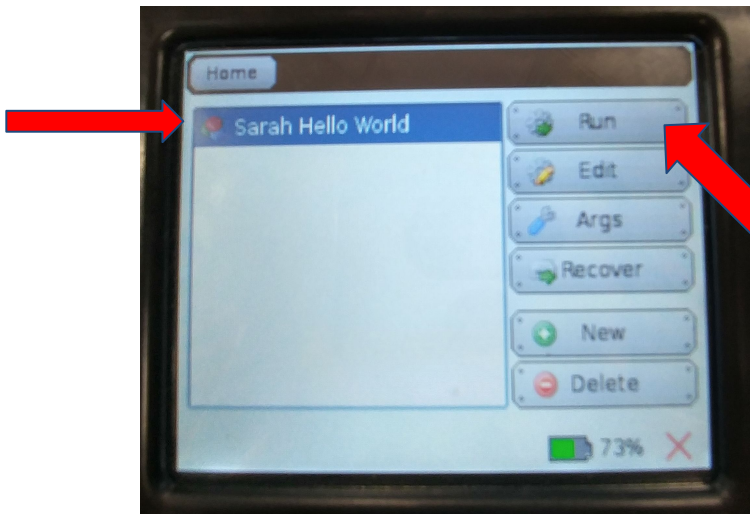


Go to the next slide

Add a Comment (cont.)



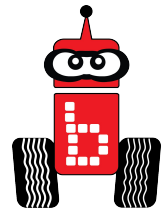
3. Select (touch) the **name of the program** you want to run.
 - **Note:** in the example below, this is “Hello, World.c”.
4. Press (touch) the **“Run” button** on the Wallaby.



***Notice that your comment did not show on the screen**

Using printf() Function

Activity 2



Objectives: Students will develop an understanding of the `printf()` function

Materials:

- Built robot, Computer, Mini USB cord

Lesson:

1. Read, discuss and follow the following slides.

[Starting a New Project](#)

[Name Your New Project](#)

[Browse Folders](#)

[Adding a New File](#) (2 Slides)

[Adding a printf \(\) Function](#) (2 slides)

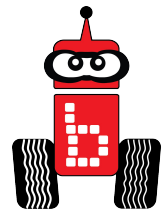
[Solution](#)

[Compiling](#)

[Running Your Program](#) (2 Slides)

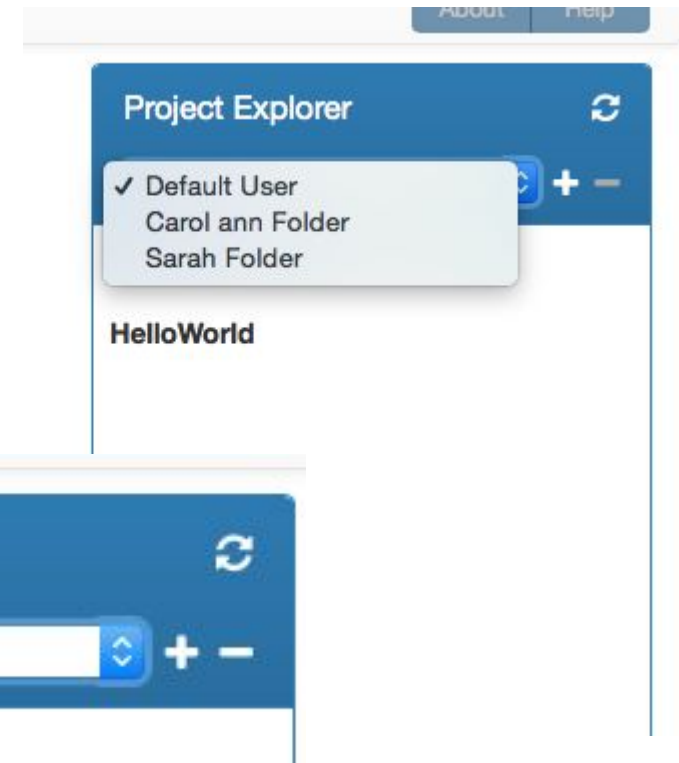
Using printf () Function

Step 1 - Start Another New Project

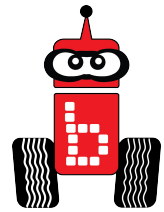


5. Go back to “Project Explore” and select the **User Name** you created from the drop down. This is the folder you created (if a newer version doesn't exist).

6. Click “+Add Project”
You are adding a project to your folder.



Name your project



Name your project “Name” Activity 2

- **Note:** you will have a lot of student’s projects, so maybe use their first name followed by the name of the activity.
- Do not put any special characters or periods, etc. on it.

Create New Project

Project name

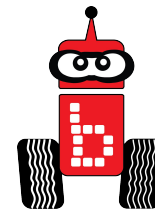
Sarah Activity 2

Source file name

main.c

Cancel Create

Click Create



1. Make sure you are in your user name/folder. Find your project name (in this case Sarah Activity 2)

2. Click “Compile” to translate your program to **machine language**.

Note: the “**Compile**” button sends the program to the **robot**.

**Our example doesn't have the project Sarah Activity 2

The screenshot shows a code editor with the following code in `main.c`:

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

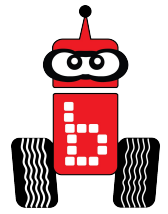
The Project Explorer on the right shows the project structure:

- Sarah Folder
 - + Add Project
 - Sarah Hello World
 - Include Files
 - + Add File
 - Source Files
 - main.c
 - + Add File
 - User Data Files
 - + Add File

The text below the screenshot states: "This will go away with update 22, version 24 is now available at kipr.org."

Using `printf ()` Function

Adding your name to the screen



Write a program for the KIPR Wallaby that displays "Hello World!" and then displays your "name". Compile, download and run it on your *Wallaby*.

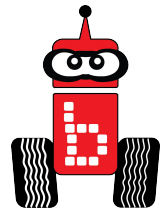
Pseudocode (Task Analysis)

1. `//Display "Hello World!" on the screen.`
2. `//Display your name on the screen.`

Questions: What function do you think you are going to use to print your name to the screen? [Answer](#)

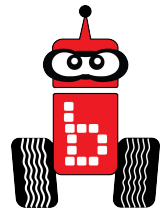
Using `printf ()` Function

Cont.



1. Add your comment after `printf ("Hello, World!\n");`
2. Use the function `printf` to write your name.
3. Add the comment `//print Your Name to the screen` after your `printf`

[Example of Solution](#)



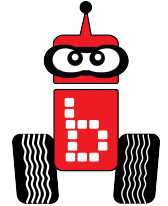
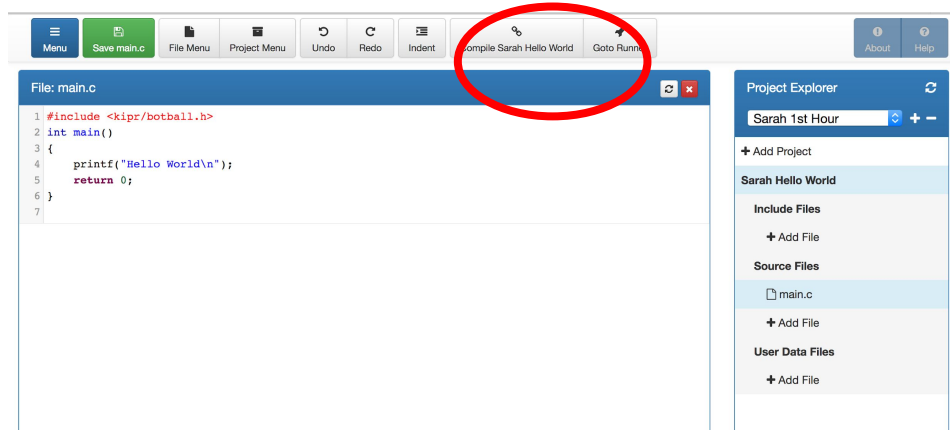
Using printf () Function

Answer

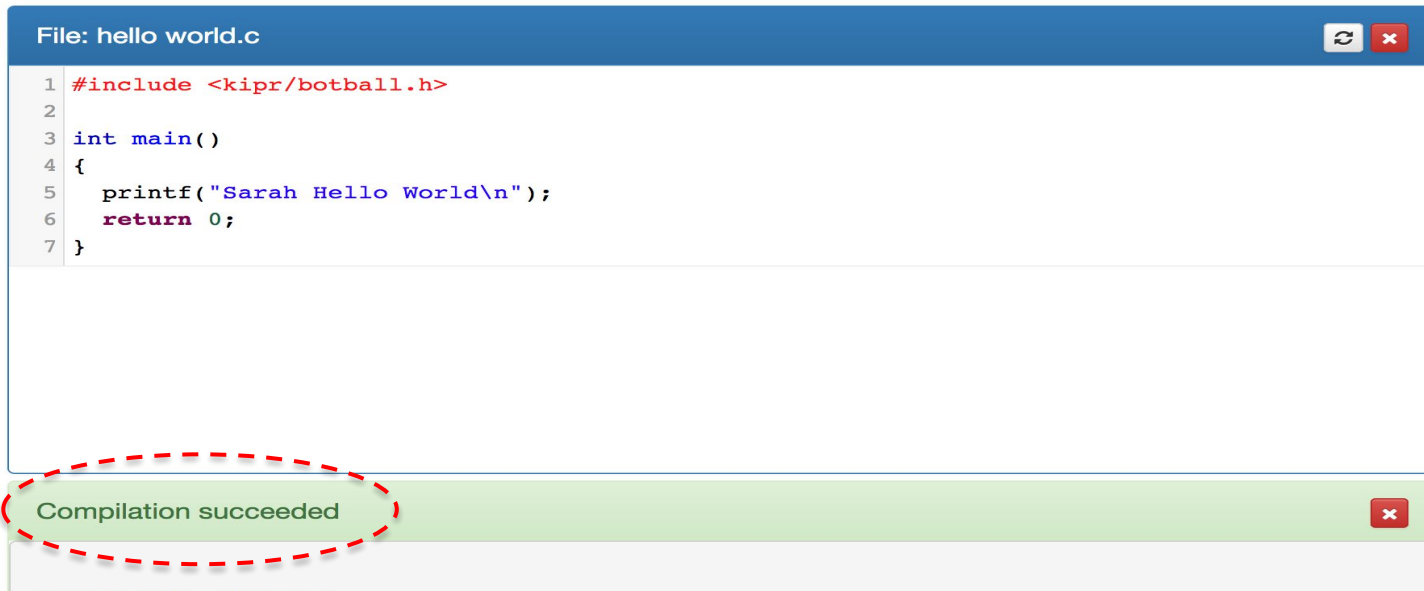
```
int main()  
{  
    printf("Hello, World!\n"); // print Hello, World! to the screen  
  
    printf("Hello, Sarah!\n"); // print Hello, Sarah! to the screen  
  
    return 0;  
}
```

Using printf () Function Step 6 - Compiling Your Program

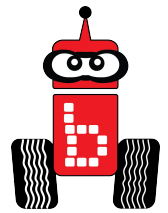
1. Compile



2. Watch to see if it Succeeded!

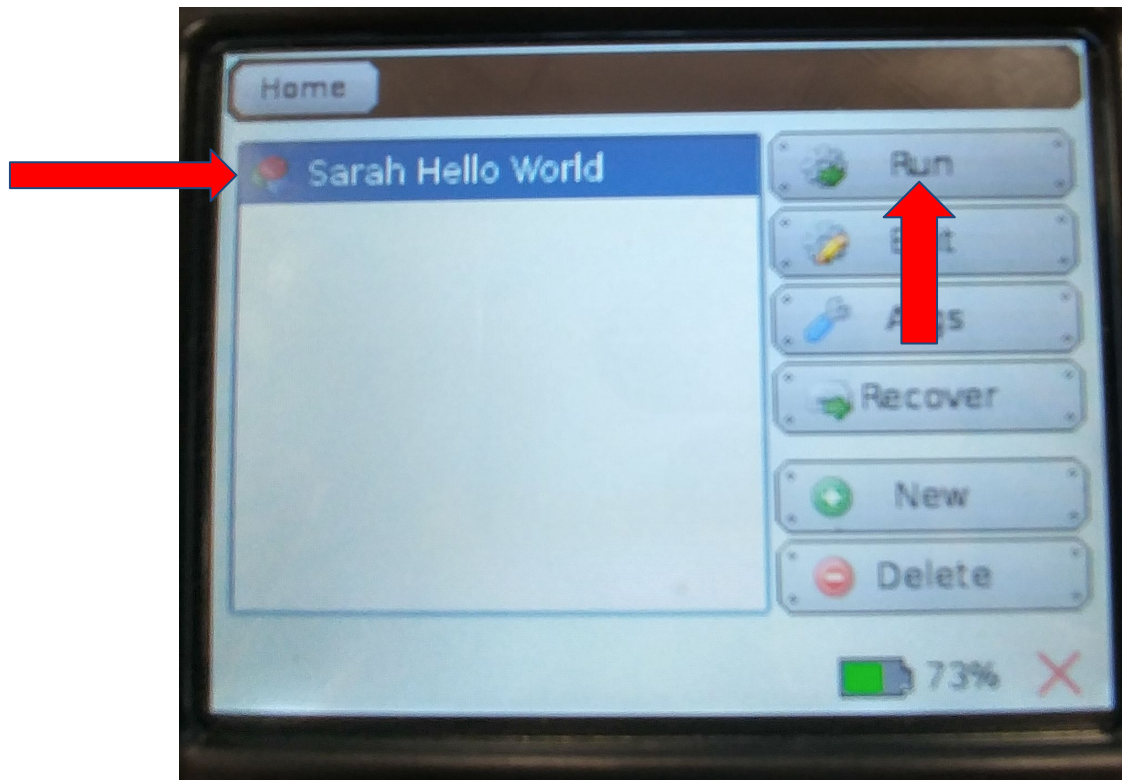


Using printf () Function



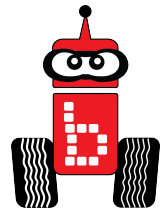
Running Your Program

1. Select (touch) the **name of the program** you want to run.
 - **Note:** in the example below, this is “Hello, World.c”.
2. Press (touch) the **“Run” button** on the Wallaby.



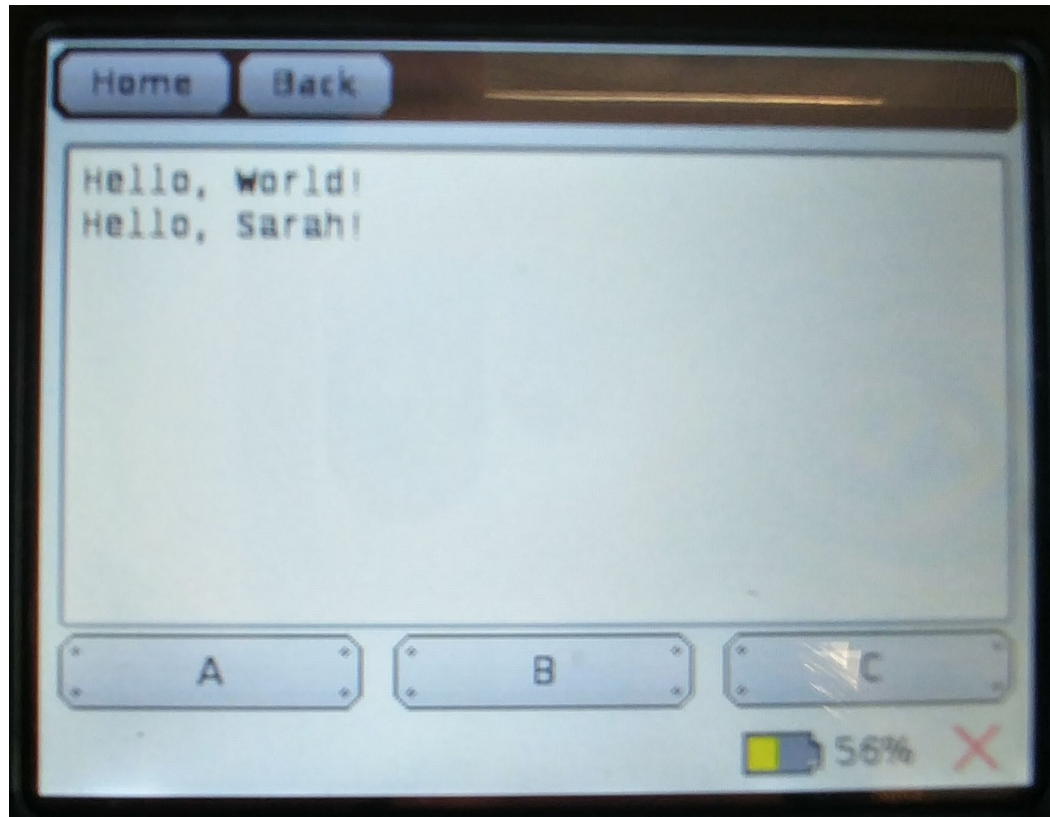
3. Continue to the next slide.

Using `printf ()` Function cont.



Running your program - Cont.

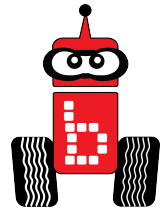
The phrase “**Hello, World!**” and “Your Name” will appear on your **Wallaby screen**.



[Print Function
Extensions](#)

Making Observations

Activity 3

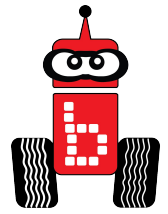


1. What did you notice when you ran the previous program?
2. Run your program again.
What observations can you make?
3. Turn to your elbow partner and discuss your observations.

[Answer](#)

Making Observations

Activity 3 (Cont.)



Answer:

- “Hello World” statement and “your name” statement appeared at almost the same time
- The two statements are on different lines

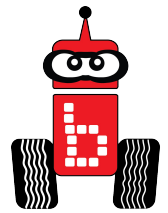
Questions:

Discuss with your elbow partner:

- Why did they appear at the same time?
- Read the next page for clarification

Making Observations

Activity 3 (Cont.)

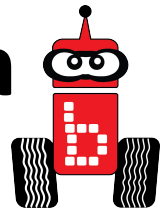


- The controller reads the code and goes to the next line faster than a blink of your eye.
- At 800MHz the controller is executing ~800 Million lines of code/second!

How do we slow down the controller?

- To control a robot you must give the function (command) **TIME** to run on the robot.
 - To do this, we use the built-in function called **msleep ()**.
 - The **msleep ()** command allows the program to pause (sleep) before it executes (runs) the next command.

Introduction `msleep ()` Function



Activity 4

Step 1

Follow the Steps 1-10 in the following slides

Objectives: Students will develop an understanding of the `msleep ()` function

Materials:

- Built robot, Computer

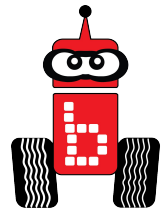
Lesson:

What does the `msleep ()` function do?

- To control a robot you must give the function (command) **TIME** to run on the robot
 - To do this, we use the built-in function called `msleep ()`
 - The `msleep ()` command allows the program to pause (sleep) before it executes (runs) the next command

Introduction `msleep()` Function

Step 2



Like `printf()`, `msleep()` is a built-in (library) function.

`msleep(1000)` causes the program to “pause” for 1 second (the m stands for milliseconds or 1/1000 of a second) before going to the next line.

- Example:

```
printf("Hello World!\n");  
msleep(2000);  
printf("Name\n");
```

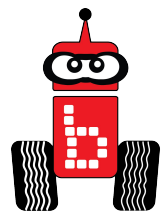
Tells the robot to wait for 2 seconds before going to the next command.

Questions:

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

Introduction msleep () Function

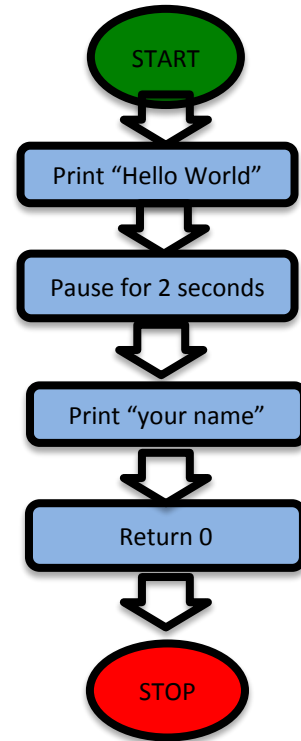
Step 3



Write a program for the KIPR Wallaby that displays "Hello World!" to the screen, delays two seconds, and then displays your name on the screen.

Pseudocode (Task Analysis)

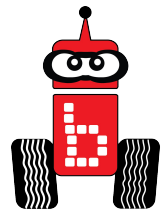
```
// 1. Display "Hello World!" on the screen.  
// 2. Pause for 2 seconds.  
// 3. Display your name on the screen.
```



Continue to the next slide for Activity instructions

Introduction msleep () Function

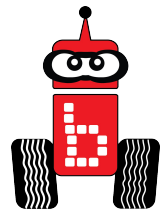
Step 4



1. Add a new project and name your new project (remember you will have a lot of student's projects- maybe use their first name followed by the challenge # or activity). **Name it "Name Activity 3"**
2. Go to the next slide for detailed directions.

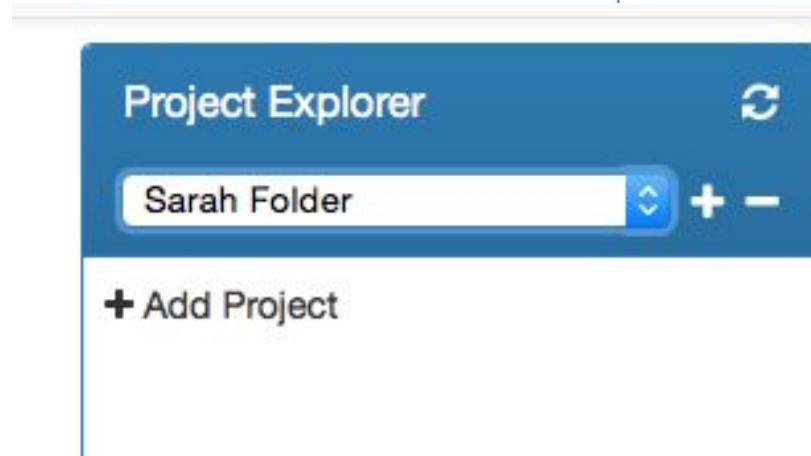
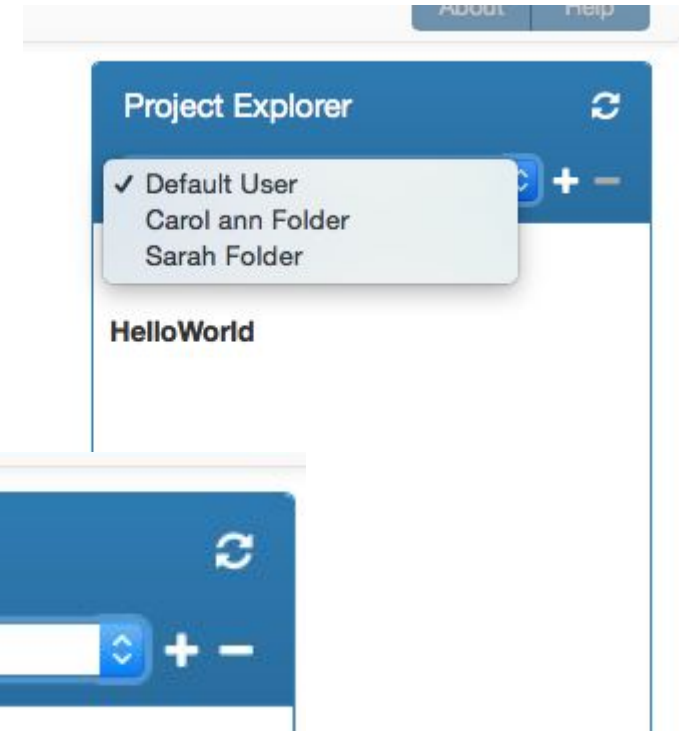
Introduction msleep () Function

Step 4

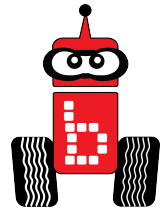


1. Go back to “Project Explore” and select the **User Name** you created from the drop down. This is the folder you created.

2. Click “+Add Project”
You are adding a project to your folder. **“Name Activity 3”**



Introduction msleep () Function



Pseudocode as Comments (notebook)

1. Print "Hello, World!"
2. Wait for 2 Seconds.
3. Print your name.



Source Code (Code in the KISS IDE)

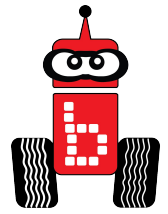
```
int main()
{
    printf("Hello, World!\n"); // print "Hello, World!"
    msleep(2000); // Wait for 2 seconds

    printf("Sarah\n"); // Print "Sarah" to screen.

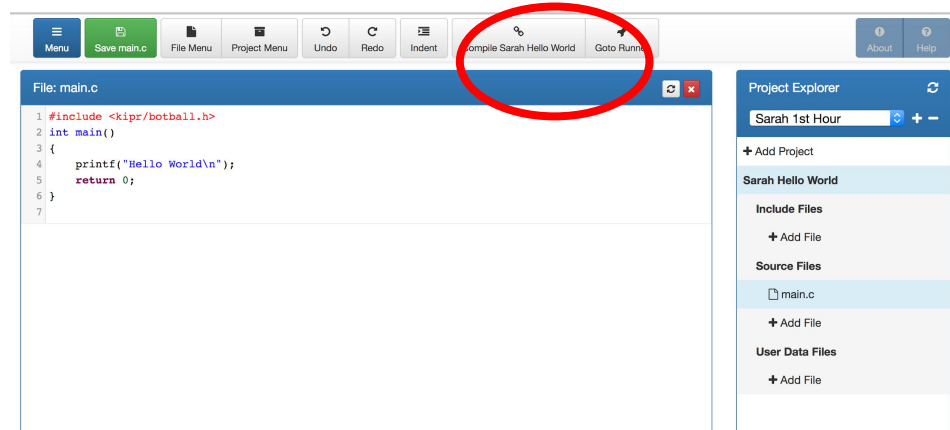
    return 0;
}
```

Continue to next
slides to compile

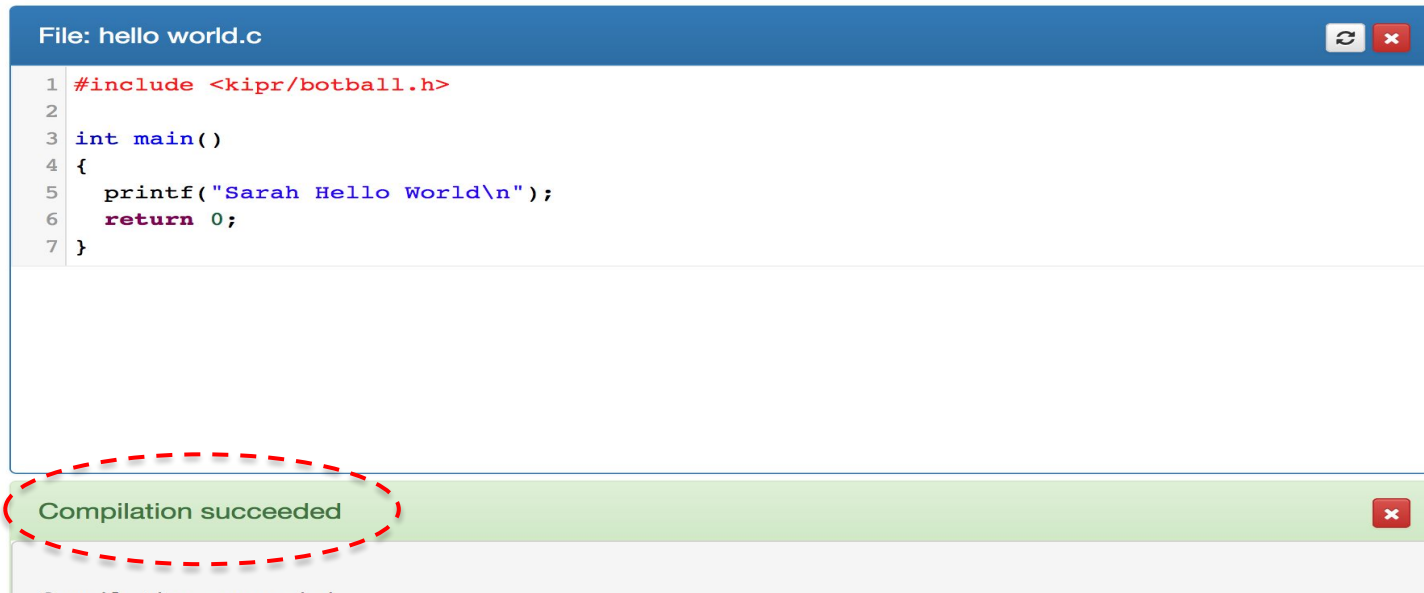
Add a Comment



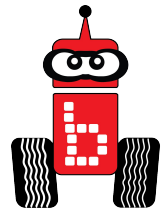
1. Compile



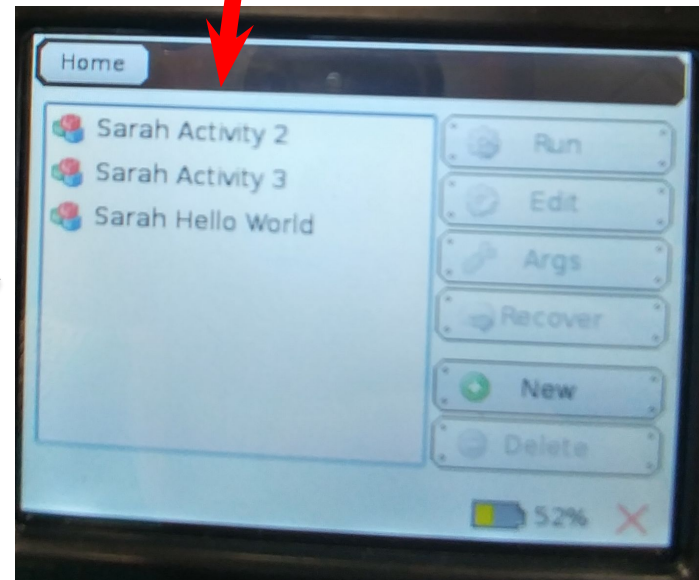
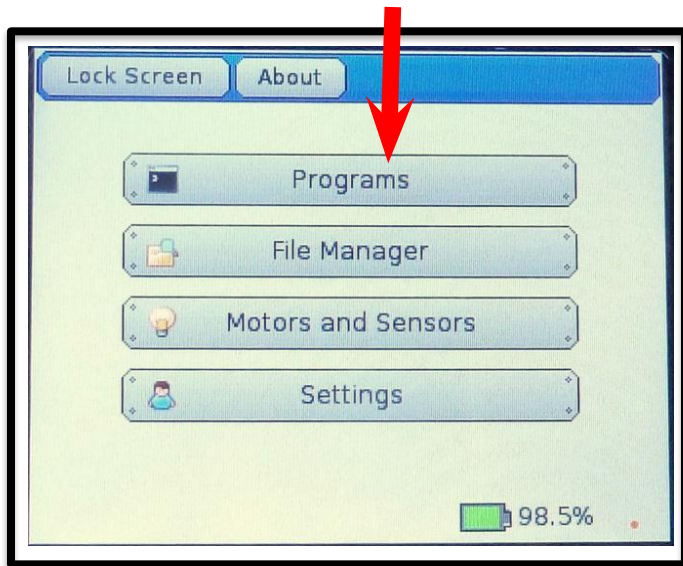
2. Watch to see if it Succeeded!



Introduction msleep () Function

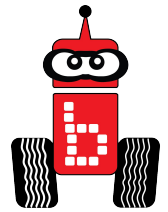


1. Press (touch) the **“Programs”** button on the **KIPR Wallaby**.
2. This will take you to a **list of programs** currently on your Link.



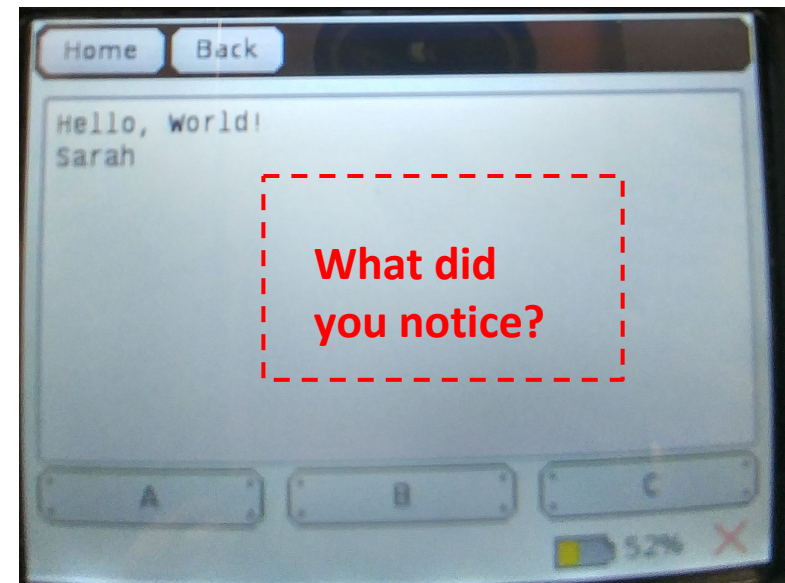
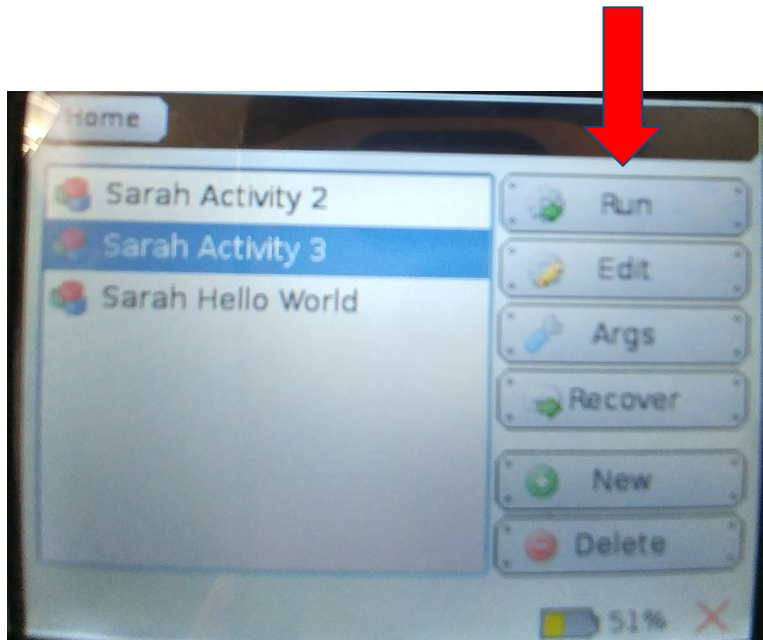
Continue to next slides to compile

Introduction msleep () Function

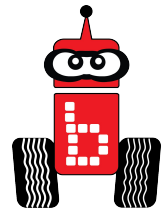


Select (touch) the **name of the program** you want to run.

- Press (touch) the **“Run”** button on the Wallaby.



Debugging Activity 5



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

Materials:

- Built robot, Computer, Mini USB cord,


Lesson:

1. Follow the Steps 1-3 in the next slides.

Debugging

1. Leave off the terminating semicolon and see what happens.
2. Compile.
3. Go to the next slide to understand how to read errors.

```
int main()  
{  
    printf("Hello, World!\n"); // print Hello World  
    msleep(2000); // Wait for 2 seconds  
  
    printf("Sarah\n"); // Print Sarah to screen.  
  
    return 0;  
}
```



\n doesn't show up on the printed output it simply tells the display to print to a new line similar to the return key on a keyboard

Debugging

Compile Failed “Debugging”

Example of an “Error Message” for missing a “;”

- Compile Failed will be the message at the bottom of the window



```
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 ';' be
fore 'motor'
  motor (0,100);
  ^
```

1. **Reading the Error**- Ignore the first line, and look at the second line to find the error.
2. This error says that it expected to see a “;” before “motor”
3. Fix one error **at a time** and then recompile. It might fix all the errors.

Debugging

Proceed to following
slides for sample

1. Spell **msleep** wrong. Compile and read the error.
What does it say?
 - Spell it correctly and compile
2. Put a comma in the **msleep (2,000)** . Compile and read the error. What does it say?
 - Take out the comma and compile
3. Remove a curly brace, compile, and read the error.
What does it say?
 - Replace the curly brace and compile
4. Replace a **0** with the letter o, compile, and read the error. What does it say?
 - Replace the **0** and compile

Debugging

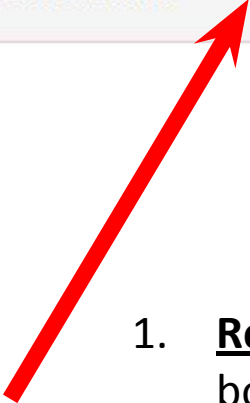
Compile Failed “Debugging”

Example of an “Error Message” for misspelling “msleep”

- Compile Failed will be the message at the bottom of the window

```
/home/root/Documents/KISS/Carol Folder/Carol Hello World/src/main.c:8:4: warning: implicit decl
aration of function 'msleep' [-Wimplicit-function-declaration]
    msleep (3000);
    ^
/tmp/cctKBCzM.o: In function `main':
main.c:(.text+0x30): undefined reference to `msleep'
collect2: error: ld returned 1 exit status
```

undefined reference is
always a spelling error.



1. **Reading the Error-** in this case go to the bottom of the errors.
2. This error says undefined reference to “msleep”. Spelled wrong.
3. Fix one error **at a time** and then recompile. It might fix all the errors.

Debugging

Compile Failed “Debugging”

Example of an “Error Message” for inserting a comma in the time for milliseconds.

- Compile Failed will be the message at the bottom of the window



```
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,
```

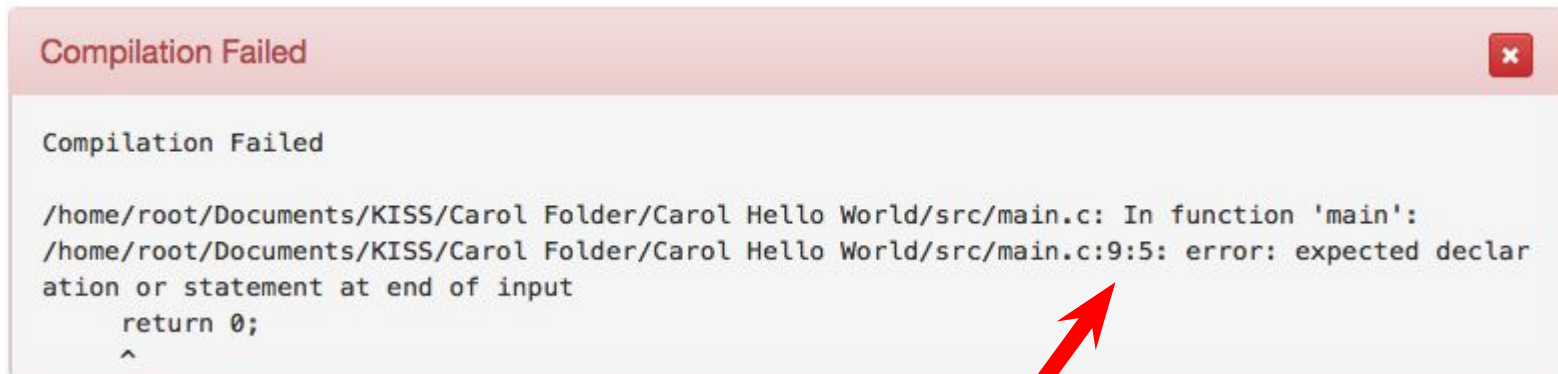
1. **Reading the Error-** in this case the error is on line 8, “too many argument”.
2. msleep has only one argument of time. The comma indicates that there is two argument.
3. Fix one error **at a time** and then recompile. It might fix all the errors.

Debugging

Compile Failed “Debugging”

Example of an “Error Message” for leaving off a “}”

- Compile Failed will be the message at the bottom of the window



```
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:9:5: error: expected declar
ation or statement at end of input
    return 0;
    ^
```

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`.
3. Fix one error **at a time** and then recompile. It might fix all the errors.

Debugging

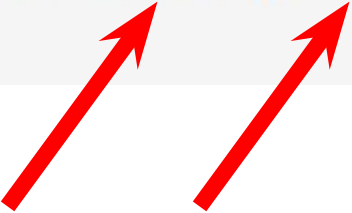
Compile Failed “Debugging”

Example of an “Error Message” for replace “o” for “0”

- Compile Failed will be the message at the bottom of the window

```
Compilation Failed
```

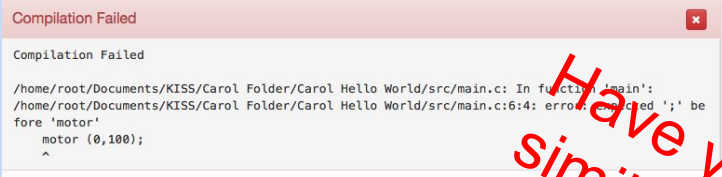
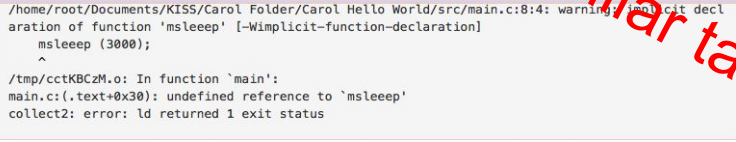
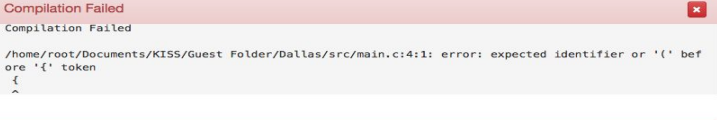

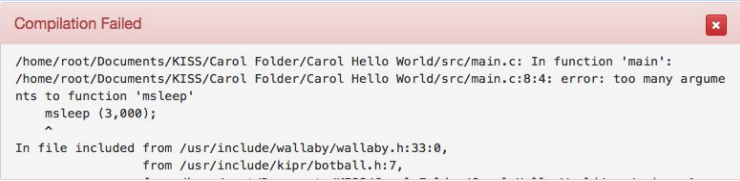

```
/home/root/Documents/KISS/Carol Folder/Carol Hello World/src/main.c: In function 'main':  
/home/root/Documents/KISS/Carol Folder/Carol Hello World/src/main.c:9:12: error: 'o' undeclared  
(first use in this function)  
    return o;  
           ^
```



1. **Reading the Error-** in this case the error is on line 9, ‘o’ undeclared (first use in this function).
2. Used a “o” instead of an 0.
3. Fix one error **at a time** and then recompile. It might fix all the errors.

Common Debugging Errors

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

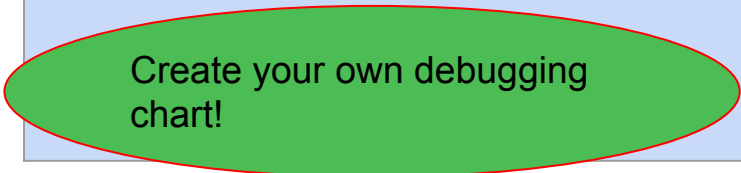
Error	Reason
	<ul style="list-style-type: none"> • missing a “;”
	<ul style="list-style-type: none"> • spelled msleep wrong
	<ul style="list-style-type: none"> • missing the “int main”
	<ul style="list-style-type: none"> • motor(0100); -is a mistake because it needs a comma to indicate the two arguments of port, and power.
	<ul style="list-style-type: none"> • 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.
	<ul style="list-style-type: none"> • 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.

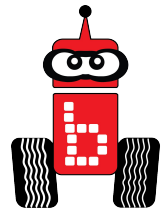
Common Debugging Errors

Have your students create a chart of common errors they have.

Reading the Error- This error says that it expected to see a “;” before line 6. Therefore, line 5 is where the error is, unless you have blank lines, then it refers to the closest line above 6.

Error	Reason
<ul style="list-style-type: none">• Undefined reference	<ul style="list-style-type: none">• spelling error; spelled the function name wrong. (spelling and case matter)
<ul style="list-style-type: none">• Expected declaration or state at end of the input	<ul style="list-style-type: none">• a “}” is missing at the end of the program. Can also indicate the “{” is in the wrong direction if it is there.
<ul style="list-style-type: none">• Expected declaration specifiers	<ul style="list-style-type: none">• a “{” is missing after <code>int main</code>. Can also indicate the “{” is in the wrong direction.
<ul style="list-style-type: none">• Expected identifier	<ul style="list-style-type: none">• missing the “<code>int main</code>”.
<ul style="list-style-type: none">• Too few arguments or too many arguments to function <code>msleep(2000);</code> - has only one argument between the parentheses.	<ul style="list-style-type: none">• <code>msleep(2,000);</code> -is a mistake because the comma indicates it has two arguments. This function has only one argument of milliseconds without a comma.
 <p>Create your own debugging chart!</p>	<ul style="list-style-type: none">• <code>motor(0100);</code> -is a mistake because it needs a comma to indicate the two arguments of port, and power.

Assessments and Rubrics



Suggestions: *Understanding* rubric
and or *Group Collaboration*