First Programs - Explaining the “Hello, World!” C Program

Professional Development Workshop
KISS Institute for Practical Robotics © 1993 – 2025 KIPR

First Programs - Explaining the "Hello, World!" C Program

KISS INSTITUTE FOR PRACTICAL ROBOTICS Botball®

Slide	Topic
3	"Hello, World!"
4	Program Flow and Line Numbers
5	Source Code
6	The main Function
7	Block of Code
8	Programming Statements
9	KIPR Functions Reference Sheet
10	Ending a Programming Statement
11	Ending the main Function

First Programs - Explaining the “Hello, World!” C Program

Slide	Topic
12	Comments
13	Text Color Highlighting
14	Print Your Name
15	Designing Your Own Program
16	Complex Tasks → Simple Subtasks
17-19	Practice Printing
20-22	Waiting for Some Time
23-24	Debugging Errors

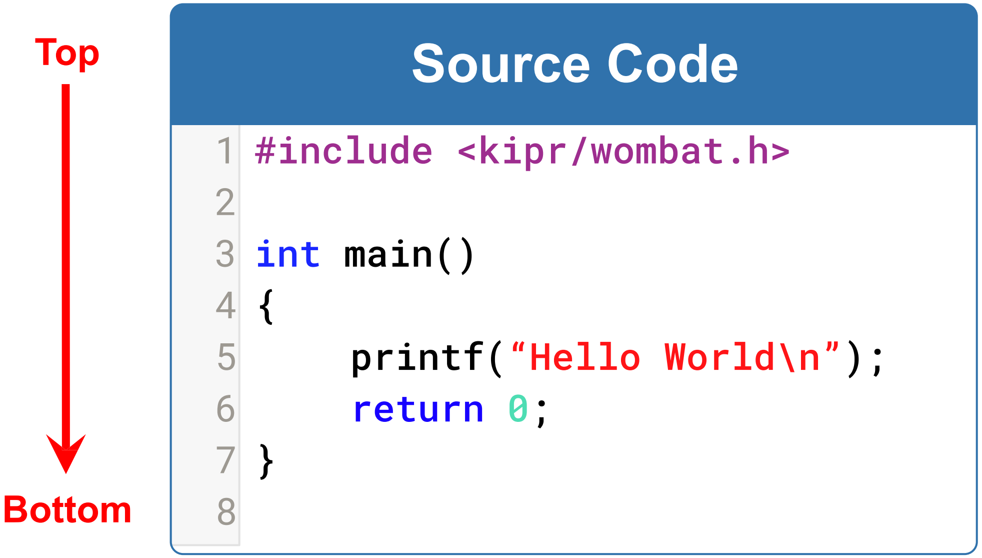
“Hello, World!”

Source Code

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

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

```

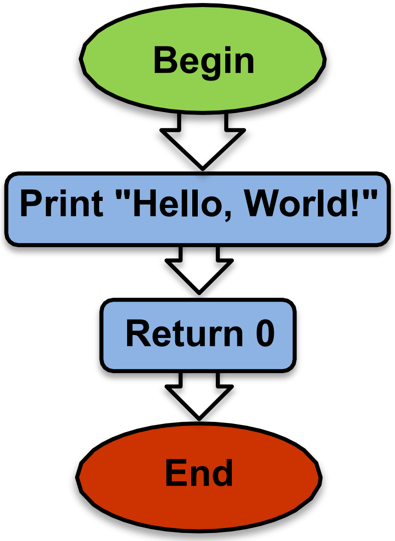
Note: We will use this template every time; we will delete lines we don’t want, and we will add lines that we do want.

Program Flow and Line Numbers



Source Code

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

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

```



Begin

Print "Hello, World!"

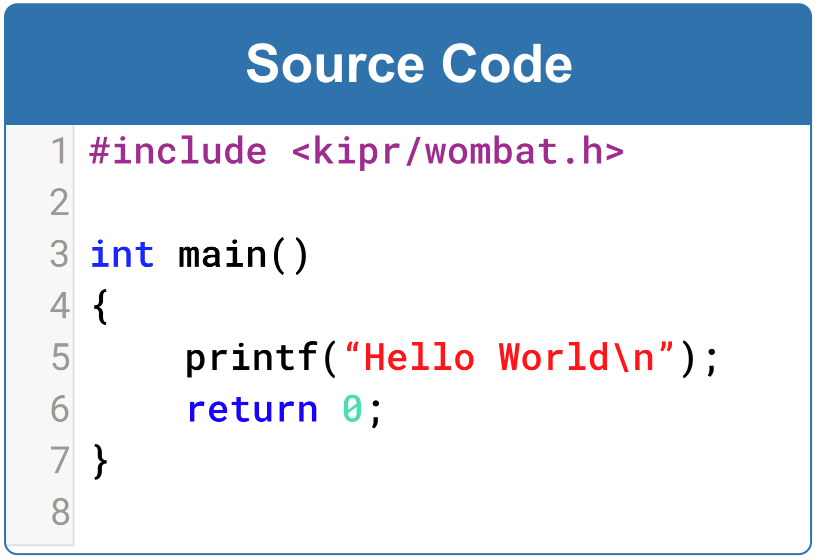
Return 0

End

Computers read a program just like you read a book—
they read each line starting at the top and go to the bottom.

Computers can read incredibly quickly—
Millions of lines per second!

Source Code



Source Code

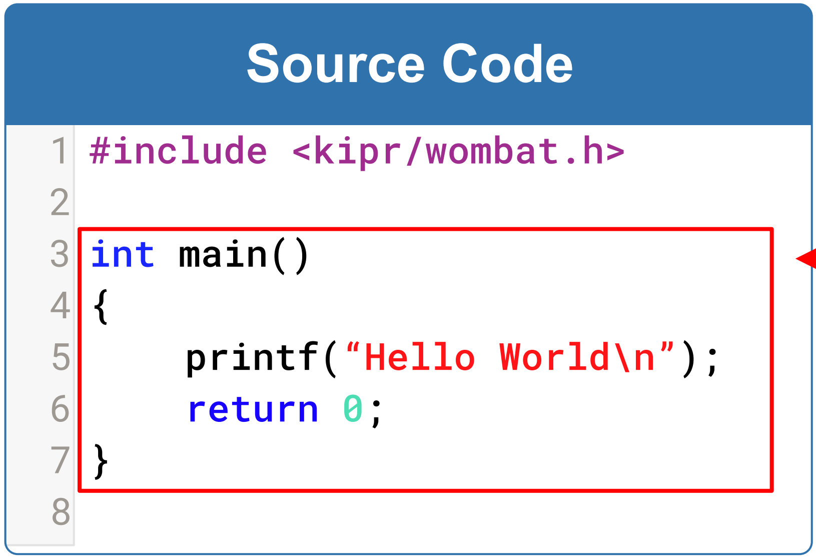
```c
#include <kipr/wombat.h>

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

```

This is the **source code** for our first **C program**.

Let’s look at each part of the **source code**.

The Main Function

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.



Source Code

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

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

```

This is the **`main()`** **function**.

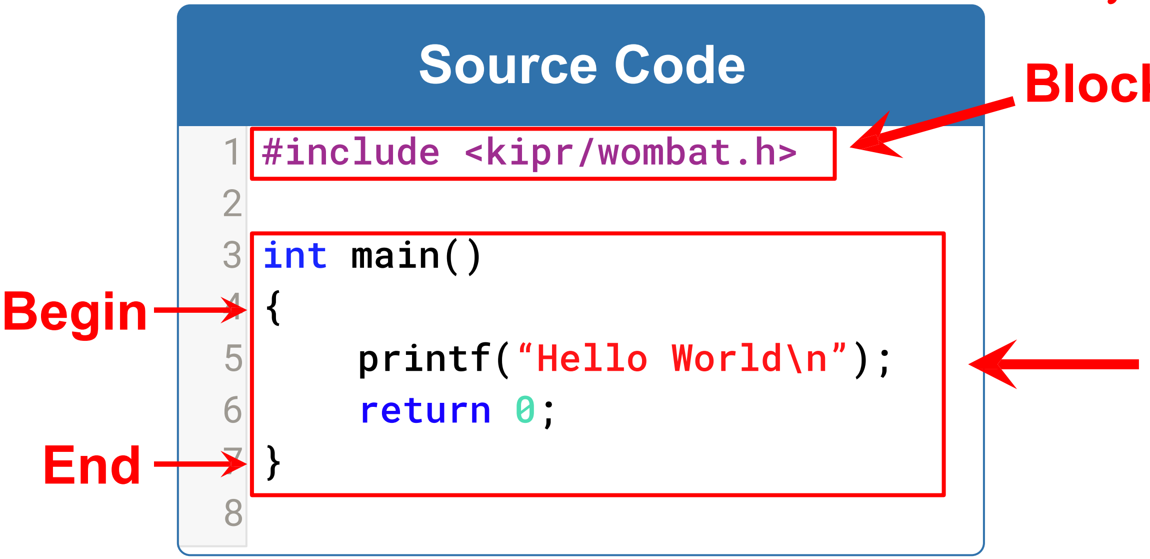
When you run your program, the **`main`** **function** is executed.

A C program must have exactly one **`main()`** **function**.

Block of Code

The list of actions that the function performs is defined inside a **block of code**.

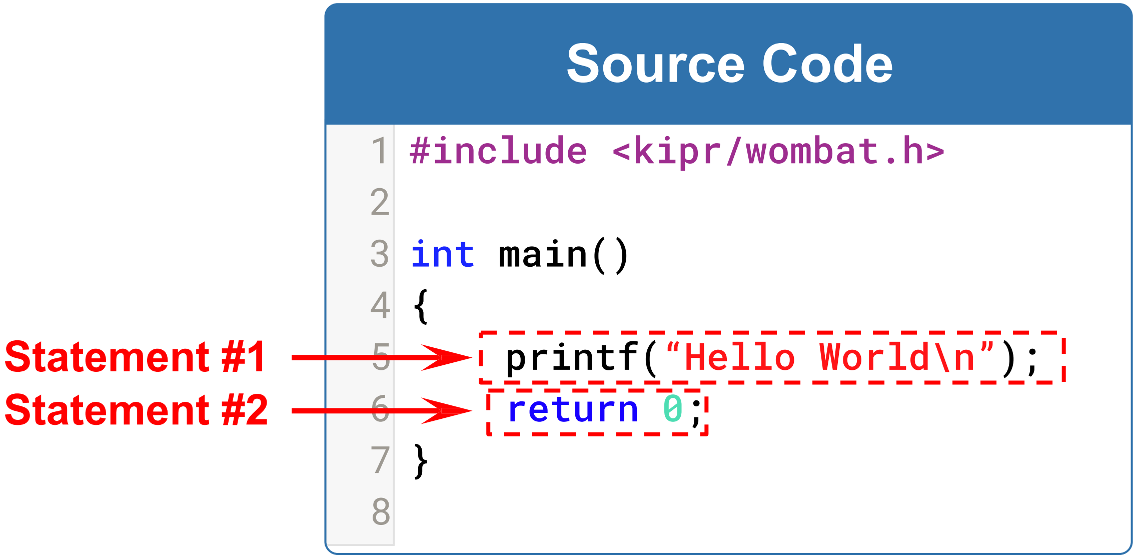
A block is defined between a **beginning** curly brace **{** and an **ending** curly brace **}**



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

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

```

This is a **block of code**.
A block of code should always be preceded by a **block header**

Programming Statements



Source Code

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

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

```

Statement #1

Statement #2

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

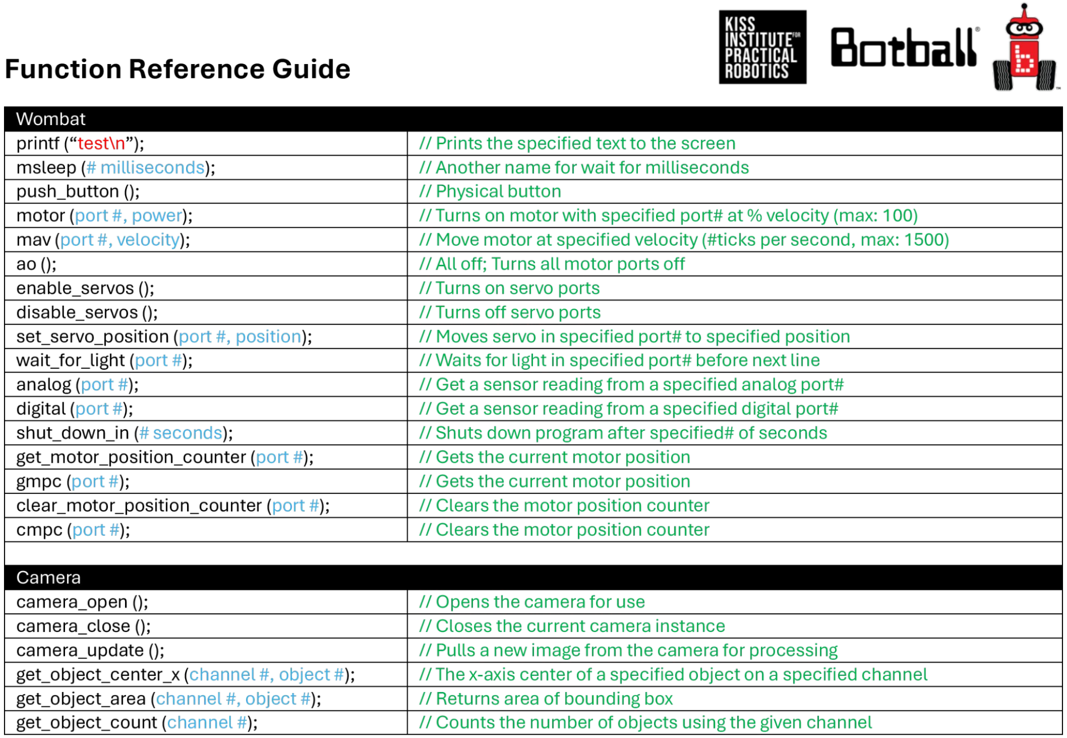
Each **programming statement** is an action to be executed by the computer (or robot) **in the order that it is listed**.

There can be any number of **programming statements** within a **block of code**.

KIPR Functions Reference Sheet

Until you are familiar with the functions that you will be using, use this function reference **sheet** as an easy reference.

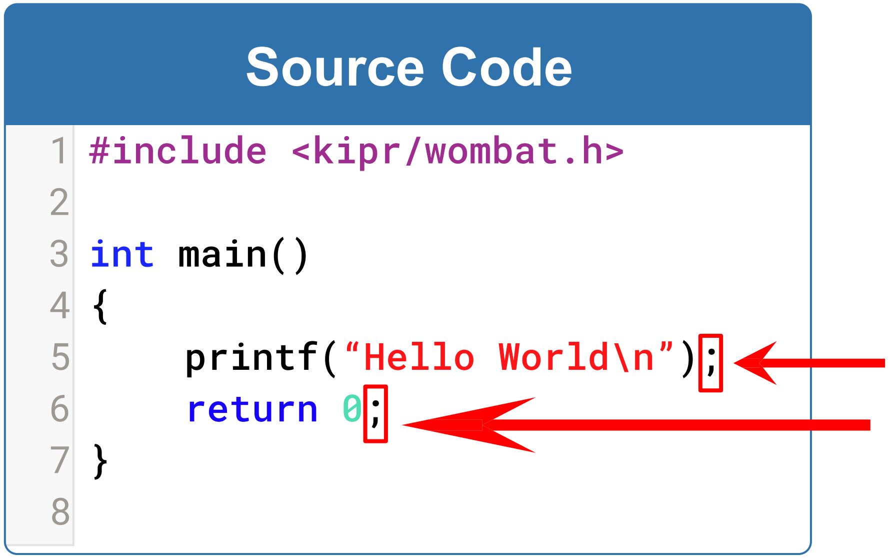
Copying and pasting your own code is also very helpful.



Function Reference Guide

Wombat	
printf ("test\n");	// Prints the specified text to the screen
msleep (# milliseconds);	// Another name for wait for milliseconds
push_button ();	// Physical button
motor (port #, power);	// Turns on motor with specified port# at % velocity (max: 100)
mav (port #, velocity);	// Move motor at specified velocity (#ticks per second, max: 1500)
ao ();	// All off; Turns all motor ports off
enable_servos ();	// Turns on servo ports
disable_servos ();	// Turns off servo ports
set_servo_position (port #, position);	// Moves servo in specified port# to specified position
wait_for_light (port #);	// Waits for light in specified port# before next line
analog (port #);	// Get a sensor reading from a specified analog port#
digital (port #);	// Get a sensor reading from a specified digital port#
shut_down_in (# seconds);	// Shuts down program after specified# of seconds
get_motor_position_counter (port #);	// Gets the current motor position
gmpc (port #);	// Gets the current motor position
clear_motor_position_counter (port #);	// Clears the motor position counter
cmpc (port #);	// Clears the motor position counter
Camera	
camera_open ();	// Opens the camera for use
camera_close ();	// Closes the current camera instance
camera_update ();	// Pulls a new image from the camera for processing
get_object_center_x (channel #, object #);	// The x-axis center of a specified object on a specified channel
get_object_area (channel #, object #);	// Returns area of bounding box
get_object_count (channel #);	// Counts the number of objects using the given channel

Ending a Programming Statement



Source Code

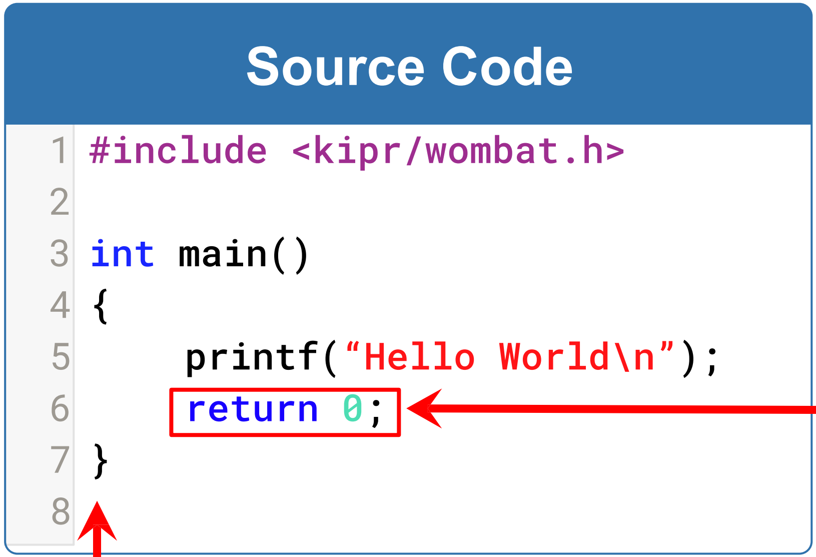
```c
#include <kipr/wombat.h>

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

```

Each **programming statement** ends with a **semicolon** ; (*unless* it is followed by a new **block of code**).

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

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

Ending the Main Function



Source Code

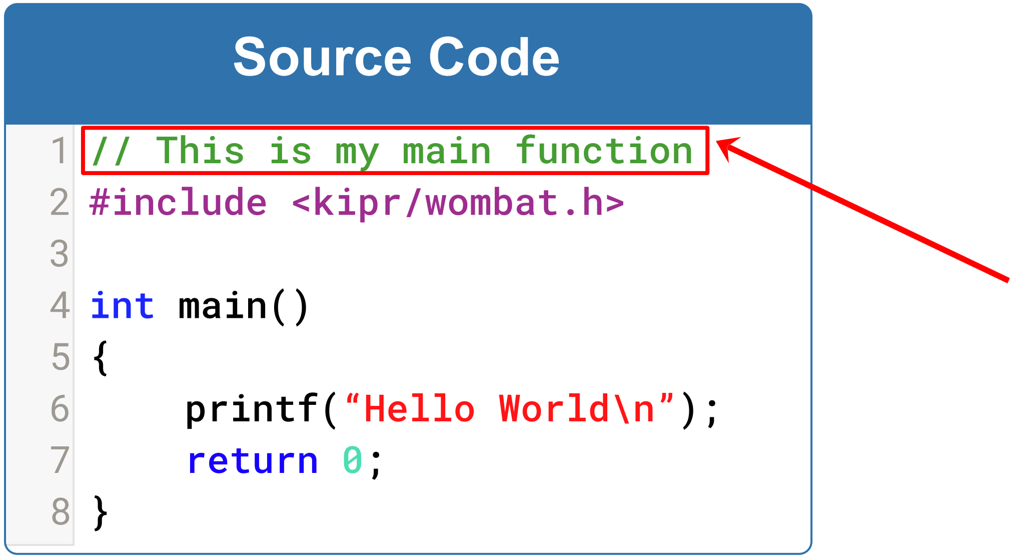
```
#include <kipr/wombat.h>

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

```

The **`main` function** ends with a **`return`** statement, which is a response or answer to the computer (or robot).
In this case, the “answer” back to the computer is `0`.

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

Comments

Text beginning with “//” is called a **comment**.



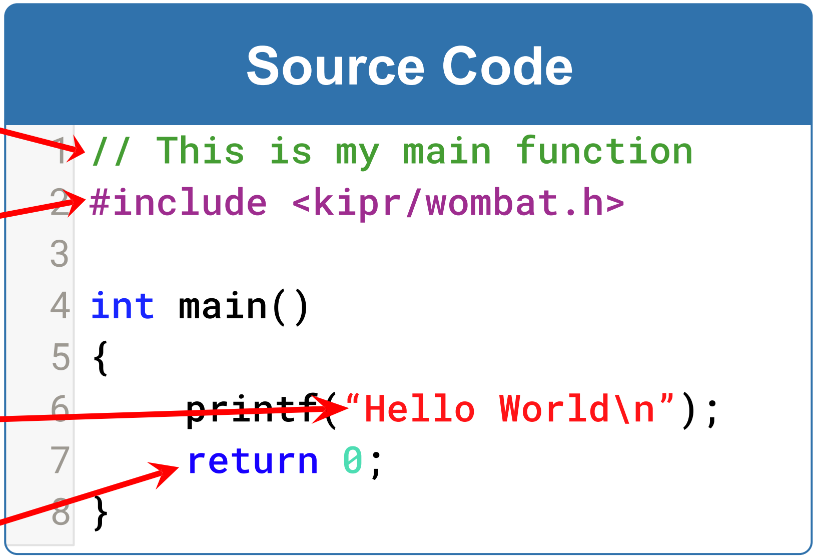
```
// This is my main function
#include <kipr/wombat.h>

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

Comments are helpful notes that can be read by you or your team—**they are *ignored* (<u>not</u> read) by the computer!**

Text Color Highlighting

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**
- **Includes** appear in **purple**
- **Text strings** appear in **red**
- **Keywords** appear in **blue**



Source Code

```c
// This is my main function
#include <kipr/wombat.h>

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

Print Your Name

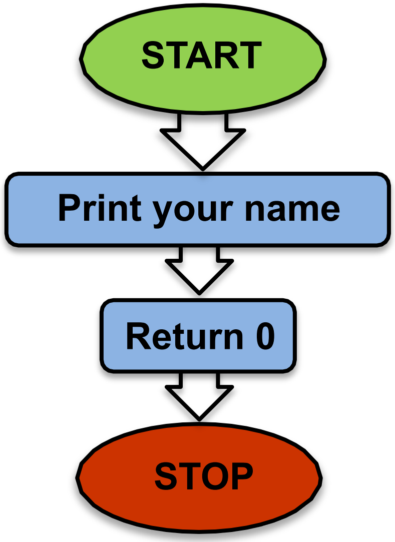
Description: Write a program for the KIPR Wombat that prints your name.

Solution:

Source Code

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

int main()
{
     // 1. Print your name.
     printf(“Botguy\n”);

     // 2. End the program.
     return 0;
}
```

Flowchart



Breaking Down a Task

Pseudocode, Flowcharts, and Comments

`msleep()` Function

Debugging Your Program

Complex Tasks — Simple Subtasks

- Break down the objectives (**complex tasks**) into smaller objectives (**simple subtasks**).

- Break down the smaller tasks into even smaller tasks. Continue this process until each subtask can be accomplished by a list of individual programming statements.

- For example, the larger task might be to make a PB&J Sandwich which has smaller tasks of getting the bread and PB&J ready and then combining them.

Practice Printing

<u>Description</u>: Write a program that prints "Hello, World!” on one line, and then prints your name on the next line.

<u>Analysis</u>: What is the program supposed to do?

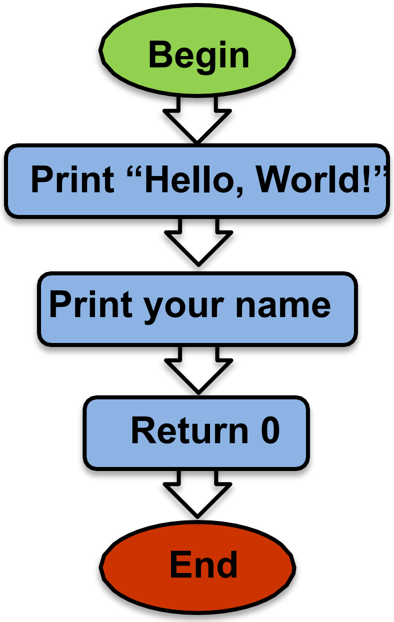
Pseudocode

```
1. Print “Hello, World!”
2. Print your name.
3. End the program.
```

Comments

```
// 1. Print "Hello, World!"
// 2. Print your name.
// 3. End the program.
```

Flowchart



In **<u>English</u>**, write a list of actions to solve an activity.

There are three different ways to do this.

Practice Printing

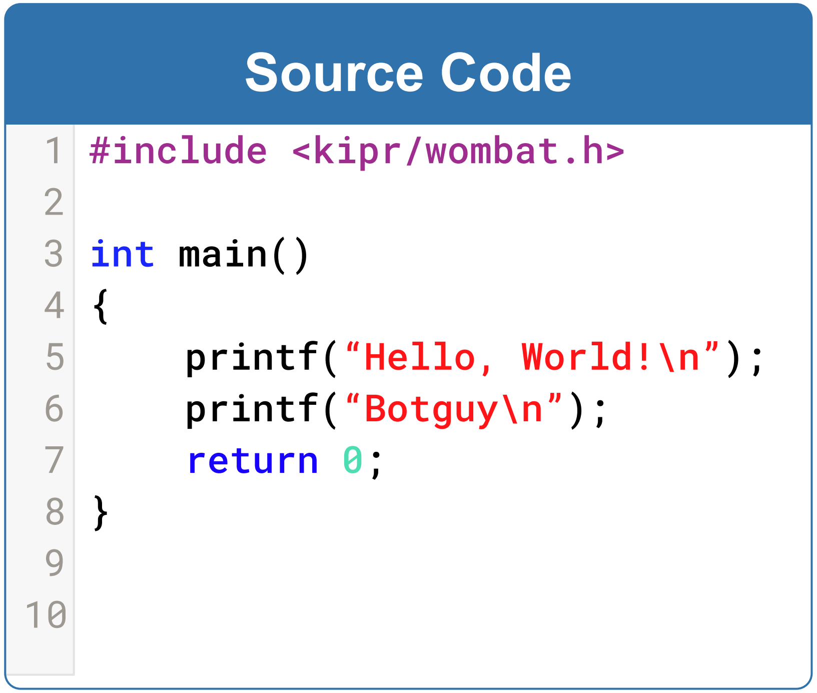
Solution: Create a **new project**, create a **new file**, and enter your **pseudocode** and **source code** in the `main` function.

- **Note:** remember to give your project and file descriptive (unique) names!

Pseudocode

```
1.   Print "Hello, World!"
2.   Print your name.
3.   End the program.
```

Helps you *write* the real code!



Source Code

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

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

Execution: Compile and run your program on the KIPR Wombat.

Practice Printing

Reflection: What did you notice after you ran the program?

- The KIPR Robotics Controller reads code and [generally] goes to the next line faster than a blink of your eye.
- The KIPR Robotics Controller is executing thousands of lines of code per second!
- To control a robot, sometimes it is helpful to **wait for some duration of time** after a function has been called so that it can actually perform the task.
- To do this, we use the built-in function called **`msleep()`**

Let's use this!

Waiting for Some Time

Description: Write a program that prints "Hello, [your name]!" on one line, waits two seconds, and then prints "Good bye." on the next line.

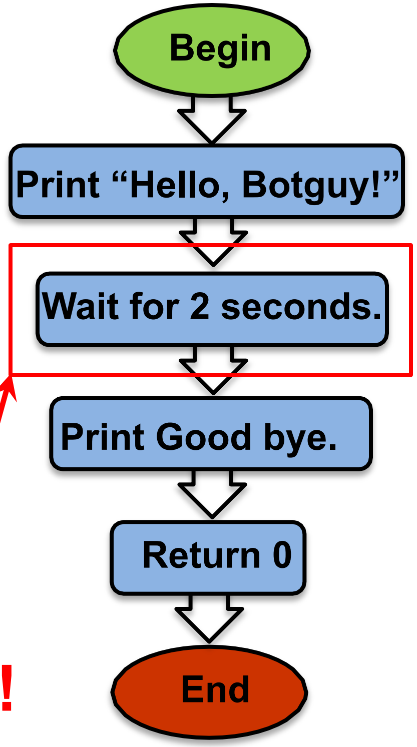
Analysis: What is the program supposed to do?

PseudocodeComments

1. Print “Hello, Botguy!”
2. Wait for 2 seconds.
3. Print “Good bye”.
4. End the program.

```
// 1. Print "Hello, Botguy!"
// 2. Wait for 2 seconds.
// 3. Print "Good bye."
// 4. End the program.
```

Flowchart



Begin

Print “Hello, Botguy!”

Wait for 2 seconds.

Print Good bye.

Return 0

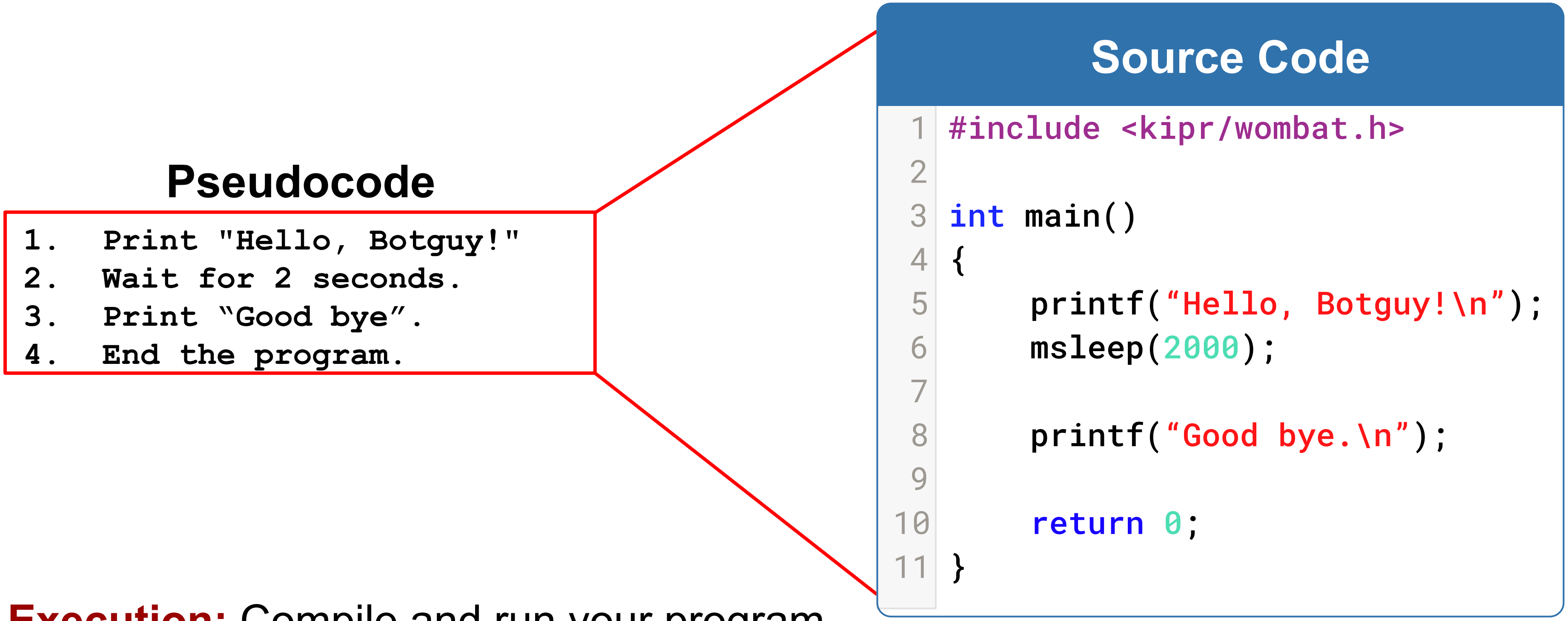
End

New!

Waiting for Some Time

Solution: Create a **new project**, create a **new file**, and enter your **pseudocode** and **source code** in the **`main`** function.

- **Note:** remember to give your project and file descriptive (unique) names!



Pseudocode

```
1.   Print "Hello, Botguy!"
2.   Wait for 2 seconds.
3.   Print “Good bye”.
4.   End the program.
```

Source Code

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

int main()
{
    printf(“Hello, Botguy!\n”);
    msleep(2000);

    printf(“Good bye.\n”);

    return 0;
}
```

Execution: Compile and run your program.

Waiting for Some Time

Reflection: What did you notice after you ran the program?

- Did your code work the first time you typed it in?
- Did you have any **errors**?

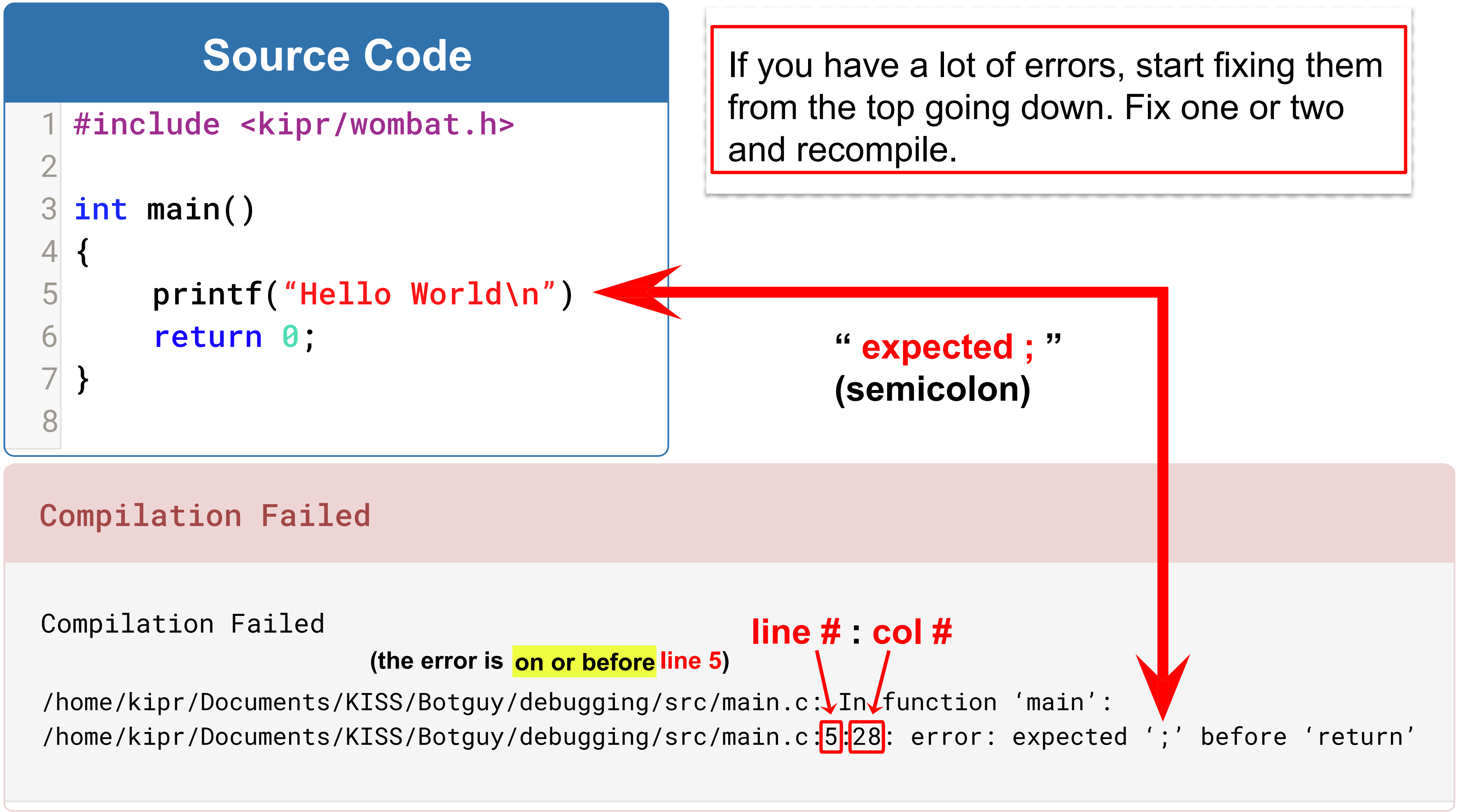
KISS Institute for Practical Robotics © 1993 – 2025 KIPR

Debugging Errors

!!! ERROR !!!

- If you do not follow the rules of the **programming language**, then the **compiler** will get confused and not be able to **translate** your **source code** into **machine code**—it will say "**Compile Failed!**"
- The Wombat will try to tell you where it *thinks* the **error** is located.
- The process of trying to resolve this **error** is called "**debugging**".
- **To test this**, remove a `;` from one of your programs and compile it.
  - How about if you remove a **"** from one of your printf statements?
  - What if you type `msleep()` as **`Msleep()`**?

Debugging Errors



Source Code

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

If you have a lot of errors, start fixing them from the top going down. Fix one or two and recompile.

“ expected ; ” (semicolon)

Compilation Failed

```
Compilation Failed

/home/kipr/Documents/KISS/Botguy/debugging/src/main.c: In function ‘main’:
/home/kipr/Documents/KISS/Botguy/debugging/src/main.c:5:28: error: expected ‘;’ before ‘return’
```

line # : col #

(the error is on or before line 5)