C Programming with Mini Sumo Robots

Editing and Compiling a Program

-Open Programmer's Notepad. This will bring up a tool for editing and compiling your program:

-Start Menu -> Programmers Notepad -> Programmers Notepad 2

-On the course website, download the firmware from the "Firmware" link on the Docs page and put it into your H: drive.

-Open the archive and extract it into a new folder.

-In Programmers Notepad, go to File -> Open Project(s), and select sumo_project.pnproj.

-Double-click myprogram.c and examine the source code. This is the default code that you will need to customize. You may want to save this file under a different name (e.g. fred_and_jane.c).

Variable Declaration

A variable represents a quantity that can change over time. Think of it as a line on a piece of scratchpad; only one piece of data can fit on the line, and wider pieces of paper can store more data. It's just like that with variables; they come in different sizes and types.

```
<VariableType> VariableName;
```

Here are some common variable definitions:

Type	Size	Range	Usage
`char`	8 bits	-128 to 127	Generally used for storing text, such as a single letter or character. Also useful for small counters.
`int`	16 bits	-32768 to 32767	Good for most arithmetic. Also useful for storing/changing robot
`long`	32 bits	-2147483648 to 2147483648	Use rarely, mainly with huge counters

Examples:

```
char letter;
int number;
long reallyBigNumber;
```

Make sure the declaration ends with a semicolon!

The range is generally 0 to 2^(bits) – 1, split between positive and negative values. Variables can be assigned when they're created, using the assignment operator (note the single equals sign). For example:

```
char letter = 'a';
int number = 0;
```

Arithmetic

The four main arithmetic operations are typed in just as you'd expect: +,-,/,*.

```
x = x + 5;
y = y - 5;
```

Note that x and y must be declared, or the compiler with complain.

Comments

A comment is a line of text in a program that doesn't actually do anything. C defines two ways to specify a comment:

```
// Double slashes cause the compiler to ignore everything
// else on the line.
/* For longer comments,
 * use a slash and asterisk,
 * followed by an asterisk and a slash.
 */
```

Conditionals

If-Then-Else

Conditionals enable your code to take a different path, depending on the value of a variable. For example, say you want to negate a value if it is less than zero:

```
if (var < 0)
     var = -var;
```

`else` statements execute if the conditional is false. For example:

```
if (var == 0)
     var = 1;
else
     var = 2;
```

If var equals zero, it will be set to one; otherwise, it will be set to two.

Be careful when checking equality! The single-equals (assignment) operator means something very different than the double-equals (equality) operator. Always use the double-equals in comparisons, and the single-equals in assignments.

Loops

Loops build upon if statements. While repeats until its condition is false. For example, this code will repeat forever:

```
while (1) {
     var = var + 1;
     var = var – 1;
     //This will do nothing… forever!
}
```

Note the use of curly braces; if you have multiple statements, these are required.

For loops are even more useful. They have three parts: initialization, condition, and action. Initialization set a variable to a default value. The condition is checked; if it is true, action is taken, and the body of the loop is executed. Then condition is checked again. For example, to do something ten times:

```
for (i = 1; i <= 10; ++i) {
     /*Look at the three parts of the syntax…
     * i = 1 means that the variable i will be set to 1.
     * The loop continues as long as i <= 10.
     * Before each loop, ++i means i is incremented by 1.
     * Hence, this code will be executed ten times:*/
     function(i);
}
```

Constants

The #define keyword is used to assign a constant. Use these where a value will be used multiple times, so that you can change it from one place in the code.

```
#define THRESHOLD 5
#define ROBOT_NAME “Psychobot”
#define IDIOT_TA “Avi”
```

Functions

To define a function, use the following form:

```
<ReturnType> FunctionName (Parameter1Type Parameter1Name,
…)
{
     // code goes here
     return (value of ReturnType)
}
```

Talk about params

You can look at the library files for examples.

Robot-specific Functions

These functions are called drivers; they use part of the processor on your robot, and using them makes life much easier for you.

The following functions will be very useful:

```
waitms(int milliseconds);
     // makes the robot do nothing for the specified number
of ms

led1_on();
led2_on();
led3_on();
led1_off();
led2_off();
led3_off();
     // turns the LEDs on and off

set_left_motor_pwm(int speed);
set_right_motor_pwm(int speed);
     // sets the left motor to a speed between -255 and
255, where 255 represents full forward, -255 is full
backwards, 0 is stopped, and numbers in between represent
variable speeds.

getadc(int sensorNumber);
     // sensorNumber is a constant already defined for you.
Just enter one of the following names:
LEFT_SHARP
LEFT_FLOOR
RIGHT_FLOOR
RIGHT_SHARP

buzz(int ms, int freq);
//buzzes the buzzer
```

```
  eg. buzz(10000,400) will buzz for 10 secs at a frequency
of 400 Hz.
```

Compiling your Program

Go to Tools->Make All. If you've made no errors, the line "Process Exit Code: 0" will show. If you have a compile error, look first for any undeclared variables, missing semicolons, or misspelled keywords. The line number will be shown for any error. If you can't find the error, ask a T.A.

Downloading the Program to Your Robot

-Go to the desktop, and open AVRStudio.
-Cancel the dialog that pops up
-Click "File->Open", and select myprogram.cof
-select JTAG ICE and ATMEGA16
-connect the emulator to the port marked JTAG (the one closest to the front of the robot)
-turn your robot on – the red LED should turn on
-click "finish" – you should see a status bar move across the bottom of the screen as the program downloads

Running Your Robot

-once the program has downloaded, click the "run" button

BOOYAH! It should run!

Alternatively, you can single-step through the program.