C-Programming: Variables and Functions

Variables

- What is a variable?
  - A variable is a facility for storing data in a program.
- How do they work?
  - When a variable is “declared”, the compiler sets aside a piece of memory to store the variable’s numeric value.
  - When the variable is called, the processor “retrieves” the numeric value of the variable and “returns” the value to be used in your program in that specific location.

Variables

- How do I create a variable?
  - To create ("declare") a variable you need 3 things.
    - Decide the data type of the variable (more on this soon)
    - Give the variable a name
    - Assign the variable a value (optional, but recommended)
  - Example:
    - **int myVariable = 1000;**
    - Type: Integer
    - Name: myVariable
    - Value: 1000

Data Types

- There are 8 different types of variables in ROBOTC
  - Integer (**int**) – Memory Usage: 16 bits / 2 bytes
    - Integer Numbers Only
    - Ranges in value from -32768 to +32767
  - Long Integer (**long**) – Memory Usage: 32 bits / 4 bytes
    - Integer Numbers Only
    - Ranges in value from -2147483648 to +2147483647
  - Floating Point (**float**) – Memory Usage: 32 bits / 4 bytes
    - Integer or Decimal Numbers
    - Variable precision, maximum of 4 digits after decimal

Data Types

- There are 8 different types of variables in ROBOTC
  - Single Byte Integer (**byte**) – Memory Usage: 8 bits/1 byte
    - Integer Numbers Only
    - Ranges in value from -128 to +127
  - Unsigned Single Byte Integer (**ubyte**) – Memory Usage: 8 bits / 1 byte
    - Integer Numbers Only
    - Ranges in value from 0 to +255
  - Boolean Value (**bool**) – Memory Usage 4 bits / .5 bytes
    - True (1) or False (0) values only.

Data Types

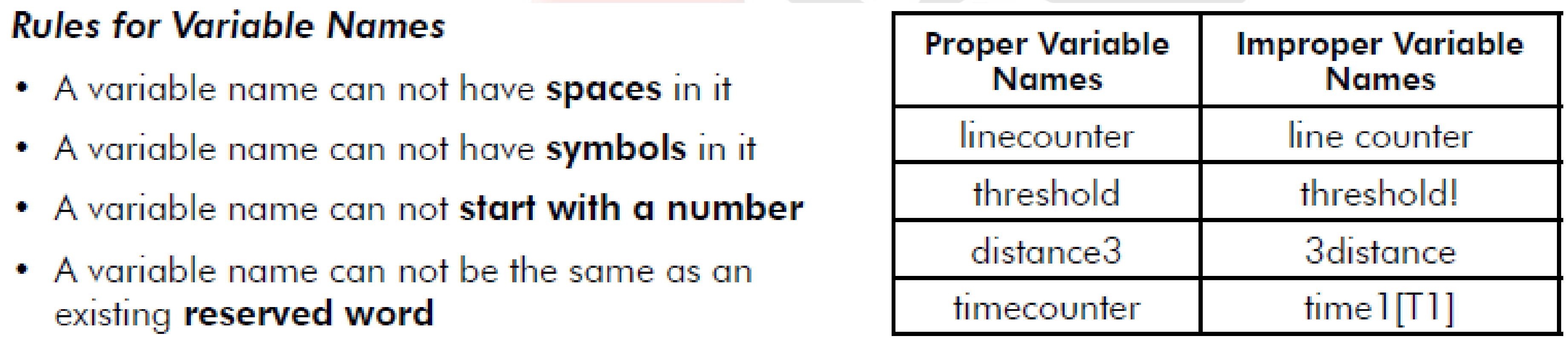
- There are 8 different types of variables in ROBOTC
  - Single Character (**char**) - Memory Usage: 8 bits / 1 byte
    - Single ASCII Character only
    - Declared with apostrophe – **‘A’**
  - String of Character (**string**) - Memory Usage: 160 bits / 20 bytes
    - Multiple ASCII Characters
    - Declared with quotations – **“ROBOTC”**
    - 19 characters maximum per string (NXT Screen limit)

Variables

- Some additional notes
  - Adding "const" in front of a variable will make that variable a constant.
    - This will prevent the variable from being changed by the program
  - Constants do not take up any memory on the NXT
  - The NXT has room for 15,000 bytes of variables

Variables

- Some additional notes
  - Variable's names must follow a specific rules:



Rules for Variable Names

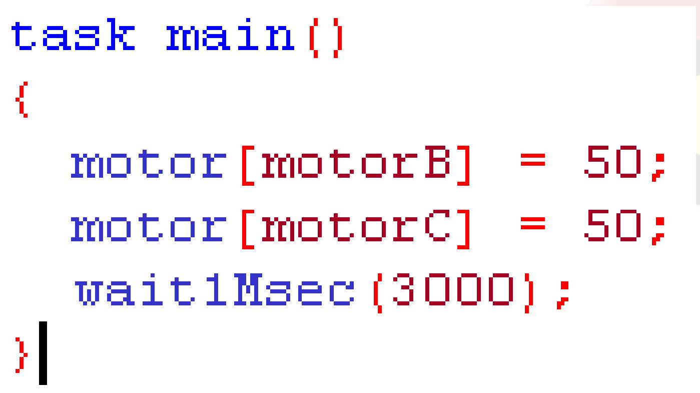
- A variable name can not have **spaces** in it
- A variable name can not have **symbols** in it
- A variable name can not **start with a number**
- A variable name can not be the same as an existing **reserved word**

Proper Variable Names	**Improper Variable Names**
linecounter	line counter
threshold	threshold!
distance3	3distance
timecounter	time1[T1]

Using Variables

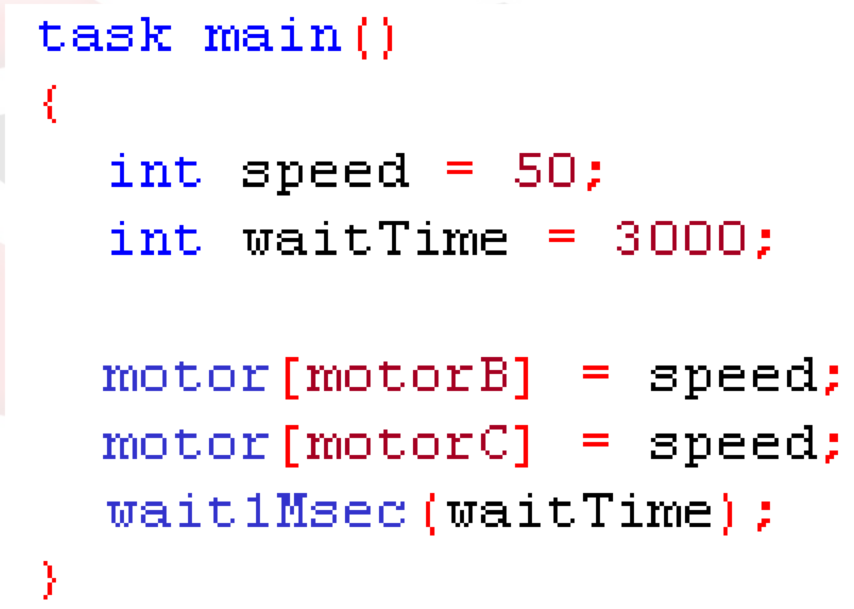
- Variables can be used in your program anywhere!
  - Motor Speeds, If/Else Loops, Conditional Statements
- Commands you know act just like variables
  - nMotorEncoder – Returns the value of a motor encoder
  - SensorValue – Returns the value of a sensor value
- Variables are just numbers
  - You can perform math operations on variables
  - **newVariable = oldVariable + 15;**

Using Variables



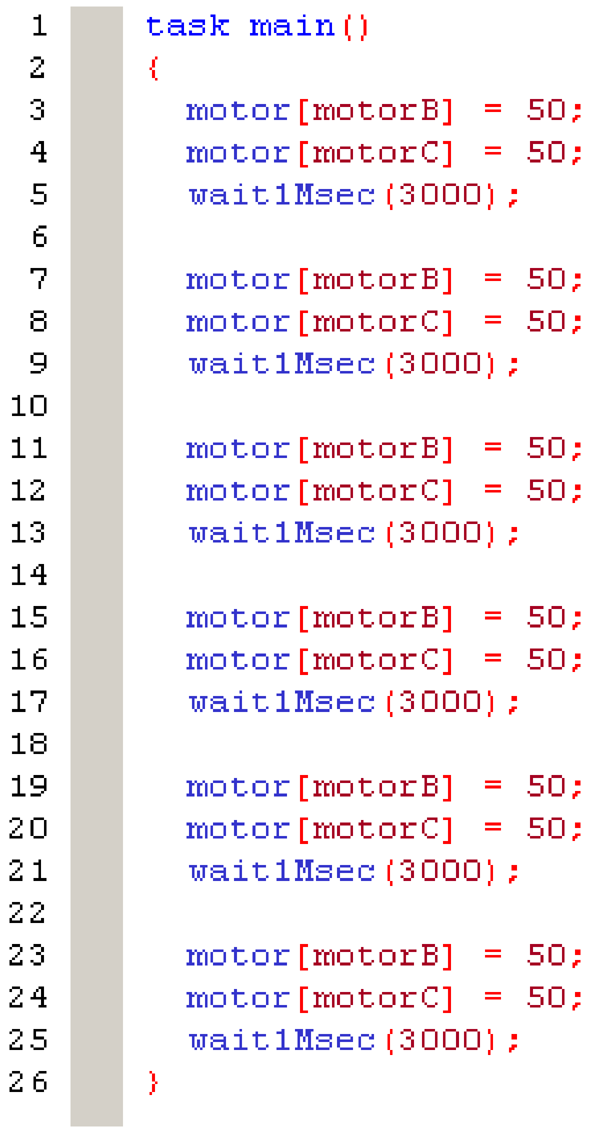
```
task main()
{
  motor[motorB] = 50;
  motor[motorC] = 50;
  wait1Msec(3000);
}
```

VS.



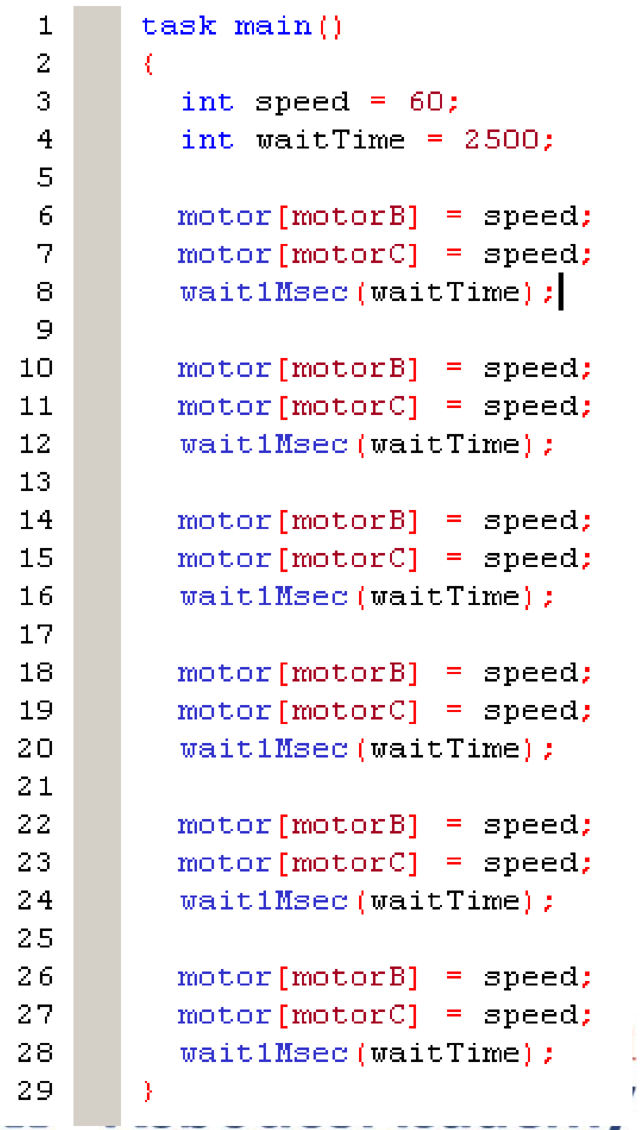
```
task main()
{
  int speed = 50;
  int waitTime = 3000;

  motor[motorB] = speed;
  motor[motorC] = speed;
  wait1Msec(waitTime);
}
```

Using Variables



```
task main()
{
  motor[motorB] = 50;
  motor[motorC] = 50;
  wait1Msec(3000);

  motor[motorB] = 50;
  motor[motorC] = 50;
  wait1Msec(3000);

  motor[motorB] = 50;
  motor[motorC] = 50;
  wait1Msec(3000);

  motor[motorB] = 50;
  motor[motorC] = 50;
  wait1Msec(3000);

  motor[motorB] = 50;
  motor[motorC] = 50;
  wait1Msec(3000);

  motor[motorB] = 50;
  motor[motorC] = 50;
  wait1Msec(3000);
}
```

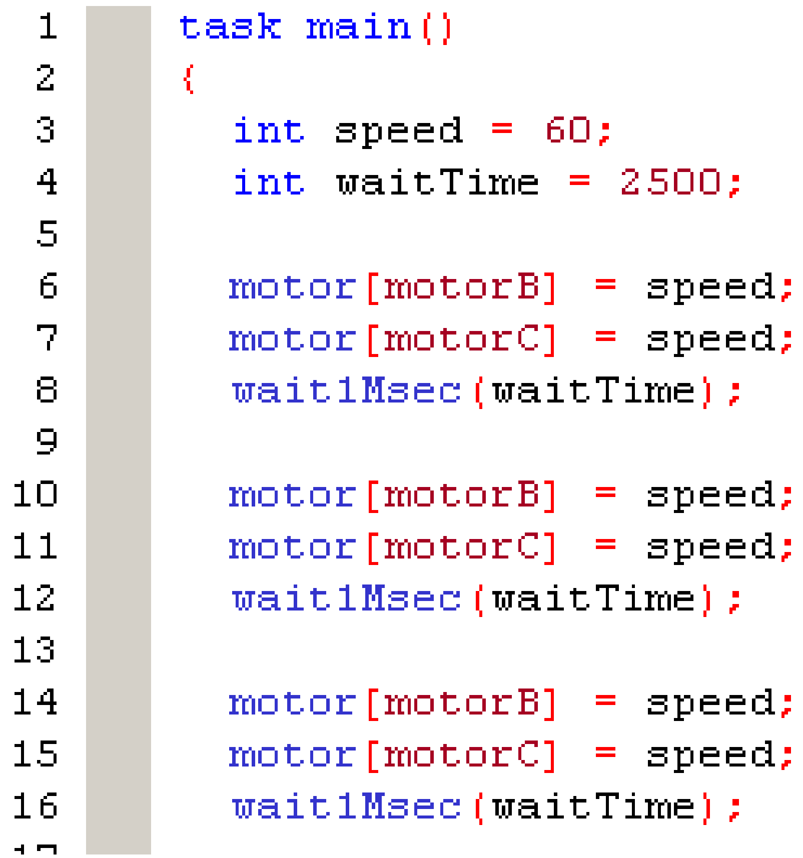
VS.



```
task main()
{
  int speed = 60;
  int waitTime = 2500;

  motor[motorB] = speed;
  motor[motorC] = speed;
  wait1Msec(waitTime);

  motor[motorB] = speed;
  motor[motorC] = speed;
  wait1Msec(waitTime);

  motor[motorB] = speed;
  motor[motorC] = speed;
  wait1Msec(waitTime);

  motor[motorB] = speed;
  motor[motorC] = speed;
  wait1Msec(waitTime);

  motor[motorB] = speed;
  motor[motorC] = speed;
  wait1Msec(waitTime);

  motor[motorB] = speed;
  motor[motorC] = speed;
  wait1Msec(waitTime);
}
```

Repeating Code…

- Copy and pasting only works so well.
- What if we could make each behavior one line of code?



```
task main()
{
   int speed = 60;
   int waitTime = 2500;

   motor[motorB] = speed;
   motor[motorC] = speed;
   wait1Msec(waitTime);

   motor[motorB] = speed;
   motor[motorC] = speed;
   wait1Msec(waitTime);

   motor[motorB] = speed;
   motor[motorC] = speed;
   wait1Msec(waitTime);
```

Functions

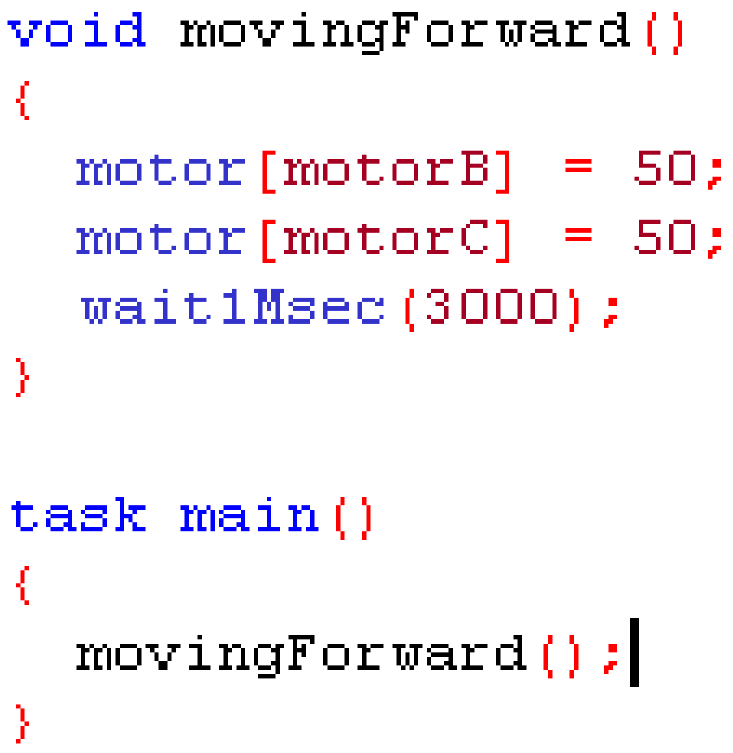
- What is a function?
  - A function (or subroutine) is a portion of code within a larger program, which performs a specific task and is relatively independent of the remaining code.
- How does a function work?
  - A function has to be first "declared" in your program, with code inside of the function.
  - Once the function is created and "declared" it can then be "called" from task main or another function to be executed.

Creating Functions

- Set the "type" of function by declaring the "data type" of the function.
  - Void is a special data type which means no value will be returned
- Give the function a name.
  - Following the same rules that variable have!
- Add a set of parenthesis and curly braces
  - Parenthesis are used for "parameters" – We'll cover this soon.
  - Curly braces define the beginning and end of the function.

```
void movingForward()
{
  motor[motorB] = 50;
  motor[motorC] = 50;
  wait1Msec(3000);
}
```

Using Functions

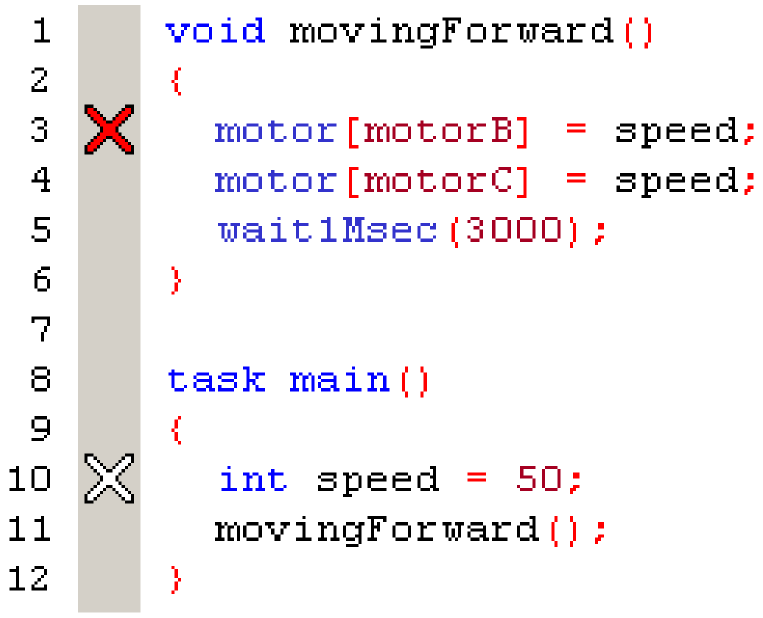


```
void movingForward()
{
  motor[motorB] = 50;
  motor[motorC] = 50;
  wait1Msec(3000);
}

task main()
{
  movingForward();
}
```

- Once the function is created, “call” the function inside of task main by referencing the name and passing any parameters.
- Don’t forget your semicolon
- Note: Keep functions above task main, or else you will have to “prototype your function”.

Functions and Variables

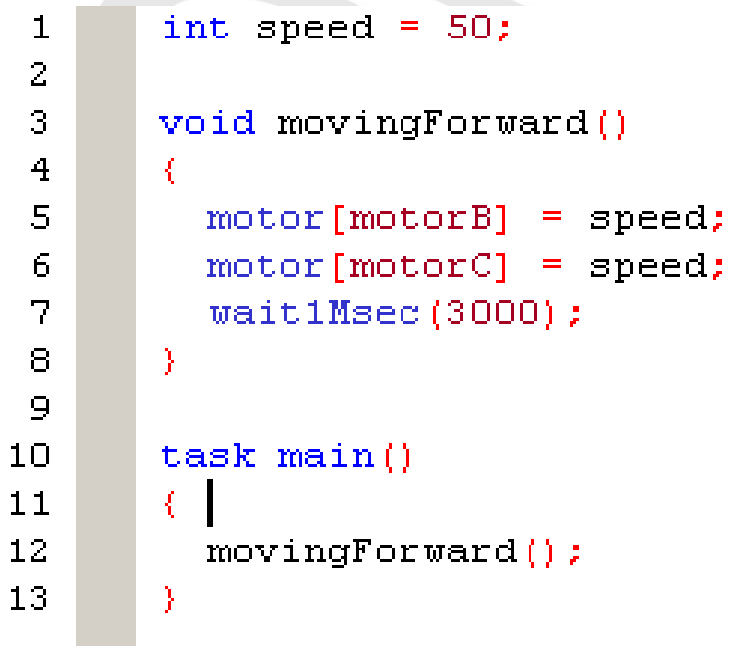
- Key Concept: Variable "Scope"
- Variables are "local" to where they are declared.
- Just because "speed" exists in "task main" doesn't mean it can be used in a function.
- "speed" is currently localized to only "task main"



```
void movingForward()
{
  motor[motorB] = speed;
  motor[motorC] = speed;
  wait1Msec(3000);
}

task main()
{
  int speed = 50;
  movingForward();
}
```

Functions and Variables

- Solution #1 – “Globalization”
  - Setting the variable outside any function will cause it to become a “global” variable.
  - This is not an ideal solution because multiple functions can use and modify this variable!
  - Use sparingly, but don’t be afraid to use it…



```
int speed = 50;

void movingForward()
{
  motor[motorB] = speed;
  motor[motorC] = speed;
  wait1Msec(3000);
}

task main()
{
  movingForward();
}
```

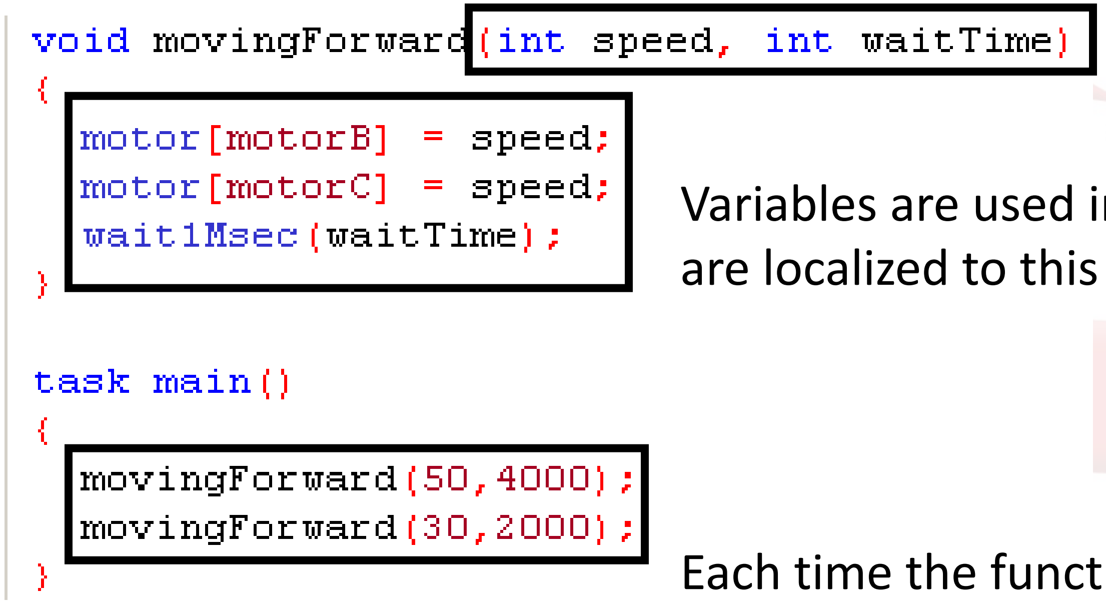
Functions and Variables

- Solution #2 – "Passing Values"
  - Instead of using the variable, we can just pass the value as a parameter to our function instead.
  - This method is ideal because it gives you flexibility in your functions.
  - This requires us to edit our existing functions, however.

```
void movingForward(int speed)
{
  motor[motorB] = speed;
  motor[motorC] = speed;
  wait1Msec(3000);
}

task main()
{
  movingForward(50);
}
```

Functions and Variables

Variables are now declared inside of the parameter field of the function. Multiple variables are separated by a comma.



```
void movingForward(int speed, int waitTime)
{
  motor[motorB] = speed;
  motor[motorC] = speed;
  wait1Msec(waitTime);
}

task main()
{
  movingForward(50,4000);
  movingForward(30,2000);
}
```

Variables are used inside of the function, but are localized to this function only.

Each time the function is called a different value can be passed. This promotes code flexibility and reuse!

Other Function Types

- Functions don’t always have to be a “void” function.
  - You can use any data type that you would assign to a variable! – int, float, bool, etc.
  - A special command “return” is required to return a value back to the parent function.

Other Function Types

Instead of “void” this function uses “int”. This tells us that this function will return an integer result.

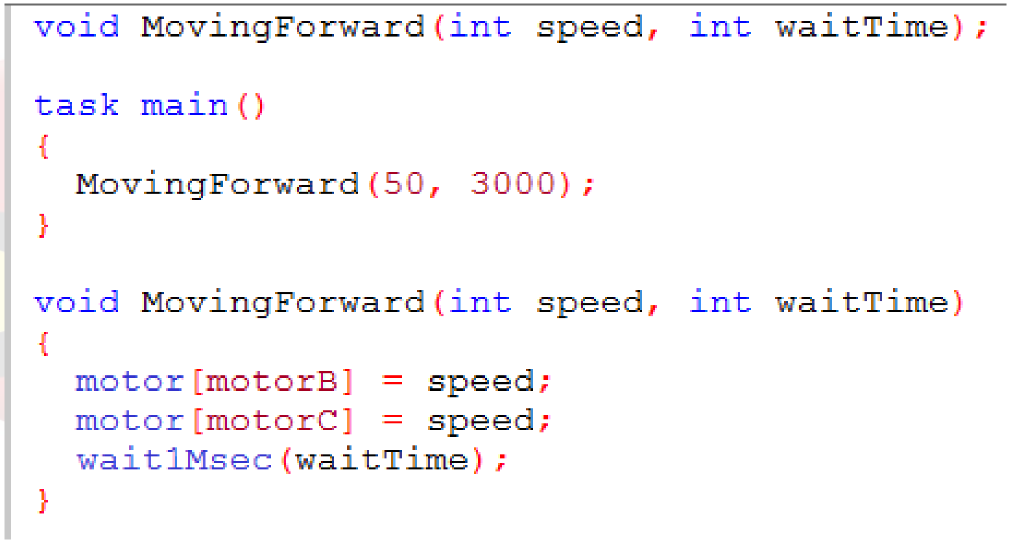
```
int addTen(int myValue)
{
  myValue = myValue + 10;
  return myValue;
}

task main()
{
  int starting = 30;
  int result = 0;
  result = addTen(starting);
}
```

After we add 10 to the parameter value that was passed to use by task main, we send the new value back by using the “return” command.

We assign the returned value to a variable so we can use the Debugger to verify the correct value was returned.

Prototyping Functions

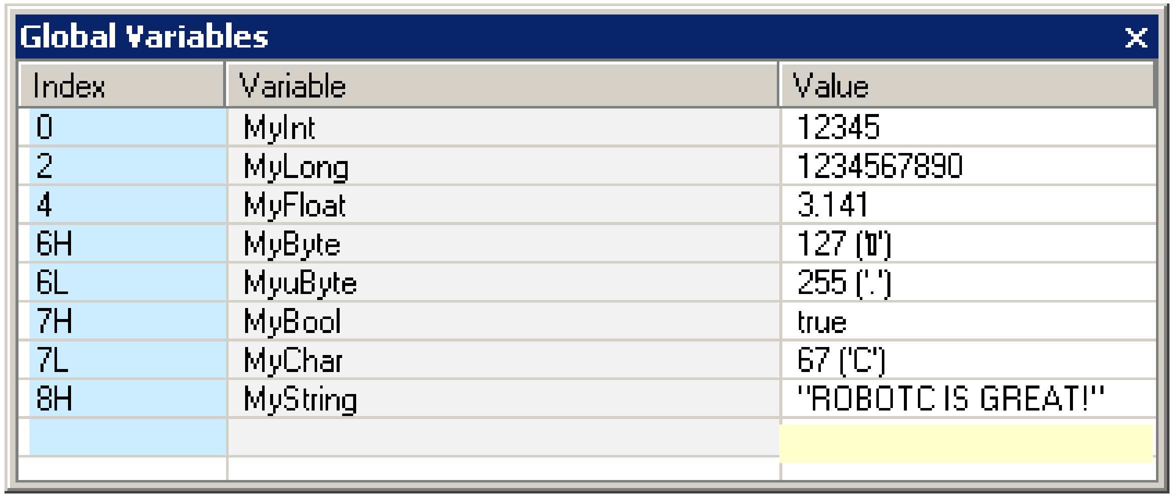
- When you declare a function below where the function is first "called", you will need to prototype your function.
- The idea is to inform the compiler that a function exists with a specific return type and parameter types.



```
void MovingForward(int speed, int waitTime);

task main()
{
  MovingForward(50, 3000);
}

void MovingForward(int speed, int waitTime)
{
  motor[motorB] = speed;
  motor[motorC] = speed;
  wait1Msec(waitTime);
}
```

Global Variables Debugger

```
task main()
{
   int MyInt = 12345;
   long MyLong = 1234567890;
   float MyFloat = 3.1415;
   byte MyByte = 127;
   ubyte MyuByte = 255;
   bool MyBool = true;
   char MyChar = 'C';
   string MyString = "ROBOTC IS GREAT!";
}
```



Global Variables

Index	Variable	Value
0	MyInt	12345
2	MyLong	1234567890
4	MyFloat	3.141
6H	MyByte	127 ('')
6L	MyuByte	255 ('.')
7H	MyBool	true
7L	MyChar	67 ('C')
8H	MyString	"ROBOTC IS GREAT!"