# Pointers and Data Structures in ROBOTC

## New capabilities enhance a leading programming environment

by Xander Soldaat (with assistance from Timothy Friez and John Watson)

*Editor's note: This article is for those with an interest in learning programming. To see the code involved, please scan the barcode or type in the URL.*

**Tens of thousands of kids are getting their first programming experience through robotics. From an educational software developer's perspective it is critical that the software has a low entry point, but a very high ceiling. With the release of ROBOTC 3.5, the ROBOTC development team included many new features designed to teach advanced programming concepts like variable pointers and recursive functions allowing students to learn the higher level programming concepts used by professional programmers today. Programmers are now able to create efficient and effective code while developing complex algorithms for applications such as autonomous path planning and advanced sensor processing using complex data structures. This article is a tutorial on how pointers work in the ROBOTC programming environment specifically with the LEGO MINDSTORMS NXT robotics controller. The tutorial is written with both new and experienced programmers in mind. New programmers will be introduced to the concepts of variables and pointers using diagrams and examples while experienced programmers will be able to see applications of these advanced concepts being used with a robotics-based application.**

The release of ROBOTC 3.5 brought a myriad of new features, including the long-awaited implementation of pointers to variables. Having this functionality opens a whole range of new possibilities, such as the implementation of complex data structures.

### WHAT ARE VARIABLES?

Prior to version 3.5, ROBOTC only supported normal variables; in other words, a variable, be it an int, float or anything else, only had a value, like 712, 0.383 or "howdy". For example:

```
long foo  = 76278;
float baz = 2.9121;
string greet= "hello";
```

In effect, when you declare a variable, the compiler puts aside an appropriately sized amount of memory and gives it a user-defined symbolic name, like "I" or "foo" or "baz". An integer (int) is 4 bytes large, a float point number (float) is also 4 bytes but a string of character (string - in the case of ROBOTC) is usually 20 bytes large. Take a look at the drawing (right—you see the memory blocks, the labels assigned to them, their contents and the memory address for that block (the hex numbers under the blocks).


### DISSECTING A VARIABLE

A variable has two parts, an r-value and an l-value. The term r-value refers to the right side of the assignment statement and l-value refers to the left side. Now consider this snippet of code:

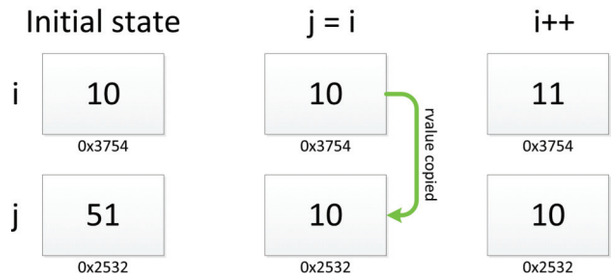
```
// ptr tutorial example 1
task main()
{
  int i, j;
  i = 10;
  j = 51;
  // This should print out - i: 10, j: 51
  writeDebugStreamLine("i: %d, j: %d", i, j);

  // Assign i's r-value to j
  j = i;
  // This should print out - i: 10, j: 10
  writeDebugStreamLine("i: %d, j: %d", i, j);

  i++;
  // This should print out - i: 11, j: 10
  writeDebugStreamLine("i: %d, j: %d", i, j);
}
```

When this program is executed, the output to ROBOTC's "Debug Stream" should look like this:

```
i: 10, j: 51
i: 10, j: 10
i: 11, j: 10
```

*(Note: To Open the ROBOTC "Debug Stream", make sure your ROBOTC interface is in "Expert" or "Super User" mode and open the Debug Stream from the normal debugger windows menu.)*

When looking at the memory locations and their contents after each operation, it would look something like this:




On the first assignment at the beginning of the program, variable i's r-value is given the value 10. Right below this assignment, there's a second assignment where variable j's r-value is given the value 51. This is considered setting an initial value.

Later in the program,what will happen with the line "j = i"? Simple, the r-value of i is copied and assigned to j's r-value. Now both j and i's r-values are 10. What happens wheni we incremeint i? Does it change j's r-value? In short, no, the two r-values are completely sepa-