Variable Values and Addresses

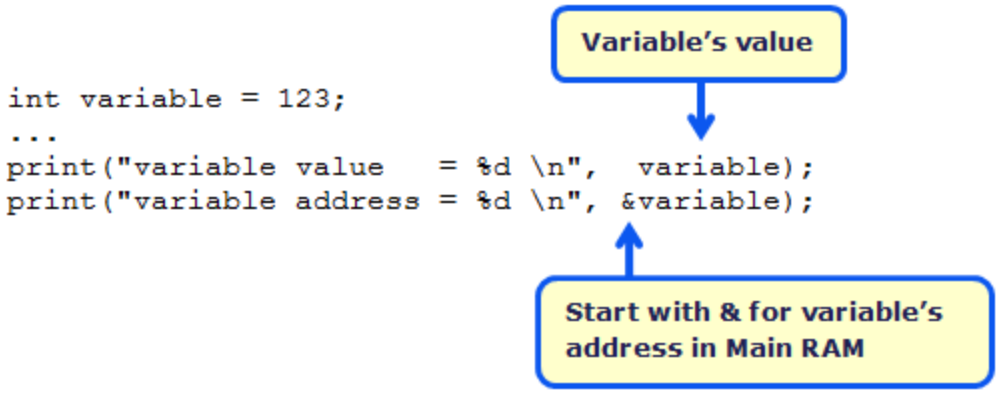
Submitted by Andy Lindsay on Thu, 10/02/2014 - 09:32

Lesson edited to work with **CodeBlocks** IDE by Jeff La Favre 3/12/26

*[SimpleIDE is the IDE for use with the robot. This lesson is edited so that we can use **CodeBlocks** as the IDE, which does not require a robot – J. La Favre]*

Up to now, we've been dealing with **what** values variables store, but there are some useful functions that need to know **where** the values are stored instead. So, to get ready for activities that use those kind of functions, let's look at a simple way to get the variable's address in RAM – by preceding it with the `&` operator.

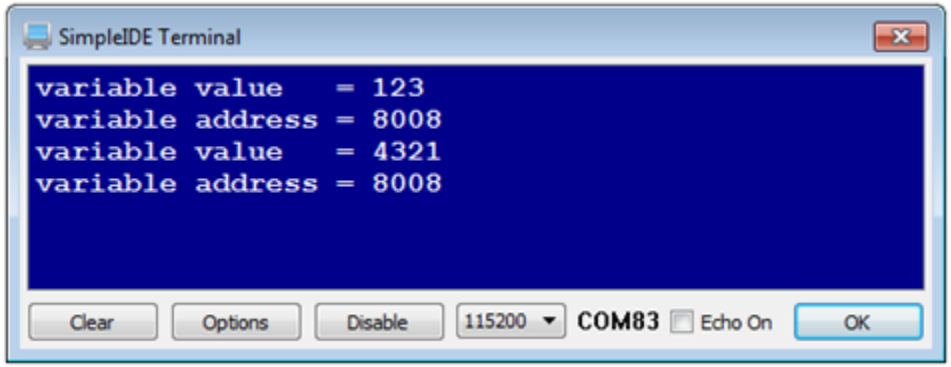
In this activity, you will also experiment with how addressing of array elements work in the Try This and Your Turn sections.



*Ed. Note: the code in the image to the left should have **printf** instead of **print** to work with CodeBlocks- J. La Favre*

Test Code

Here is an example where a **`printf`** statement displays the value of an **`int`** variable (that's named **`variable`**), and another **`printf`** statement that displays the address where that value is stored, with **`&variable`**. After changing **`variable`**'s value, a third **`printf`** displays the variable's new value, but the fourth **`printf`** (again with **`&variable`**) shows that the address has not changed.



- Start **CodeBlocks** (or the IDE you are using).
- Open the **File** menu and select **New**. Then select **Empty File**.
- Click the mouse in the text window of **CodeBlocks** and use the keyboard to enter the following text: **#include <stdio.h>**
- Open the **File** menu and select **Save file**, which opens a **Save file** dialog box.
- In the dialog box, open the drop-down labeled **Save as type** and select **C/C++ files**. In the **file name** slot enter this name for the file: **variable value and address.** At the top of the dialog box there is a **Save in** slot, which determines where the file will be saved. Make sure you know the location where you are saving your file so that you can find it later. Now click the **Save** button to save your program file.
- Using the keyboard type the text in the box on the next page into the text window of **CodeBlocks** under the first line of text you have already entered.

- Click the **Save** button to save the code you just entered with keyboard.
- Run the program by opening the **Build** menu and selecting **Build and Run**. If there are no errors in the program, a new program window will open. Verify that the program display resembles the one on page 2 (but the variable address values will be different).

```
int variable = 123;                              // Declare/initialize variable

int main()
{
 printf("variable value   = %d \n",  variable);  // Display variable value
 printf("variable address = %d \n", &variable);  // Display variable address
 variable = 4321;                                // change value of variable
 printf("variable value   = %d \n",  variable);  // Display variable value
 printf("variable address = %d \n", &variable);  // Display variable address

}
```

How it Works

The program starts by declaring an int variable (named variable) and initializing it to the value 123 with:

int variable = 123;

In the **`main`** function, the first printf call displays **`variable`**'s value with the expected result of 123.

printf("variable value = %d \n", variable);

So, at this point we expected the value to be 123, but what is **`variable`**'s address? A simple "address of" **`&`** operator to the left of **`variable`** returns its address instead of its value. The address is the number of bytes from the start of Main RAM.

```
printf("variable address = %d \n", &variable);
```

After setting the variable's value to something different, **`printf("variable value = %d \n", variable)`** displays the new value, but **`printf("variable address = %d \n", &variable)`** shows that the variable's address is unchanged.

Did You Know?

The addresses in the robot RAM are expressed as byte addresses. It's the number of bytes from the start of Main RAM.

This approach works for most variable types, including **`int`**, **`short`**, **`byte`**, **`and float`**.

The **`&`** operator can also be used to get addresses of array elements. For example, with the array:

```
int array[5] = {11, 13, 17, 19, 23};
```

…your code could get the address of the third element (containing 19) with **`&array[3]`**. Here is an example of a **`printf`** statement that displays it:

```
printf("array[3] address = %d \n", &array[3]);
```

There is more than one way to get the zeroth element's address (containing 11). Although code can use **`&array[0]`**, it's more common to just use **`array`**. So, these two **`printf`** statements do exactly the same thing:

```
printf("array[0] address = %d \n", &array[0]);  // Display address of array[0]
```

```
printf("array[0] address = %d \n",  array);     // Display it again
```

Each element in an **int** array has 4 bytes, so the address of the second element will be 4 higher than the address of the first element, and so on. A **char** array would be different since each element only has one byte. So, each element's address only increases by one over the previous element.

Try This

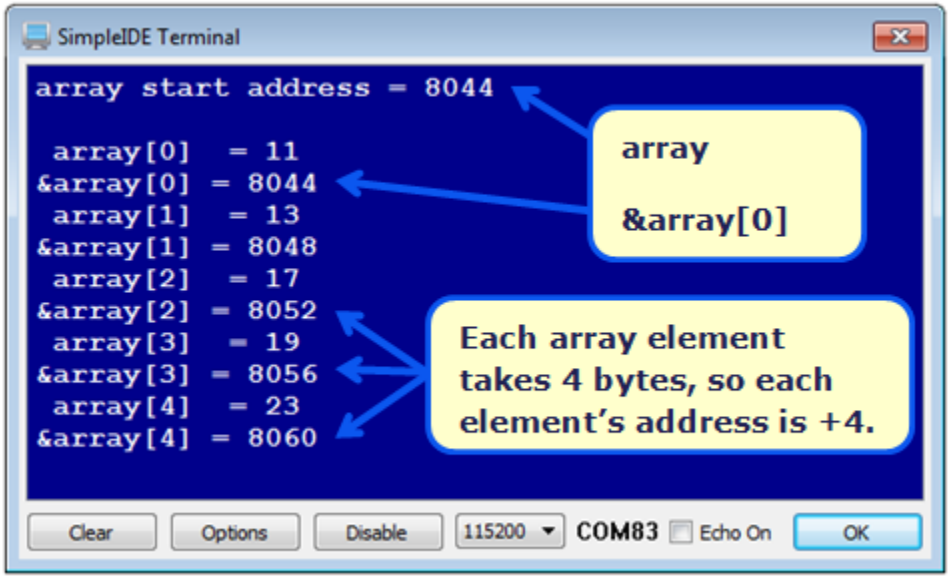
Let's prove that each array element takes four bytes, and also that **array** can be used interchangeably with **&array[0]** to get the starting address of an array.

- Open the **File** menu and select **Save file As** to save a copy of your project named **array values and addresses**.
- Modify the variable declaration and **main** function to make your code match what is shown below.

```
int array[5] = {11, 13, 17, 19, 23};

int main()
{

 printf("array start address = %d \n\n", array);

 for(int i = 0; i < 5; i++)
  {
   printf(" array[%d] = %d \n", i, array[i]);
   printf("&array[%d] = %d \n", i, &array[i]);
  }
}
```

- Run the program and verify the output.

Your modified program's output should resemble this.



IMPORTANT: Passing the address of the first element in an array by using the array's name is a very common practice; it is a good idea to get comfortable with it. Most code examples will use `array` instead of `&array[0]` for the starting address of an array.

Your Turn

The Did You Know section mentioned that a **`char`** array has one byte per element.

- Modify the Try This code to use a **`char`** array instead of an **`int`** array to test and verify this.