Get Values From Terminal

Submitted by Andy Lindsay on Wed, 10/15/2014 - 10:43

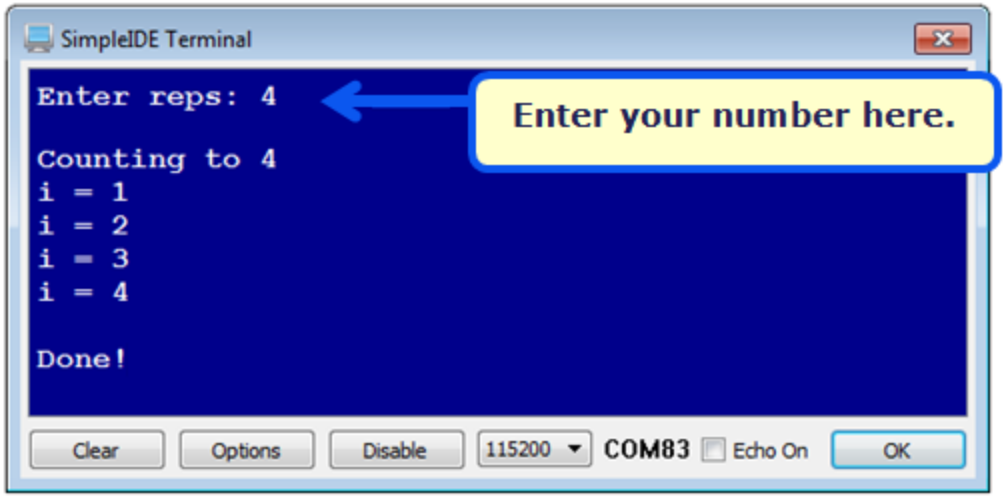
Lesson edited to work with **CodeBlocks** IDE by Jeff La Favre 3/12/26

*[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]*

Now that we know how to get variable addresses, let's try passing them a function that can do something useful with them. In this activity, you'll get to experiment with the **`scanf`** function to receive a value you type into the terminal window. [Ed. Note: for the robot IDE the function name is **scan** but when using CodeBlocks, we must use the function name **scanf** – J. La Favre]

Test Code

The **`scanf`** function allows you to type values into the terminal (the window that program runs in). It can then convert the characters you type into the value those numbers represent. For example, after displaying "Enter reps: " using a **`printf`** function call, your code can then make a **`scanf`** function call to load the number you typed into a variable.



- 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: **scan to int.** 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.
- Use your keyboard to 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 (the terminal).
- Type a value next to "Enter reps:" and press Enter.

- Verify the program counts up to the value you typed.

```c
#include <stdio.h>
int reps;                                   // Declare variable named reps

int main()
{
  printf("Enter reps: ");                    // User prompt to enter reps
  scanf("%d", &reps);                       // Scan reps user types
  printf("\nCounting to %d\n", reps);        // Display value scanned

  for(int n = 1; n <= reps; n++)            // Count to value scanned
  {
    printf("i = %d\n", n);                  // Display each step
  }

  printf("\nDone!");                        // Display when done
}
```

How it Works

After including the standard input output library (stdio), the program declares an **int** variable named **reps** that will be used to limit how high the computer counts.

```
#include <stdio.h>                              // Include stdio library
int reps;                                       // Declare variable named reps
```

The **main** function starts by printing the "enter reps" message.

```
int main()                                      // Main function
{
  printf("Enter reps: ");                        // User prompt to enter reps
```

The **scanf** function takes keyboard input from the terminal and uses it to set one or more values in one or more variables. This **scanf** function has a formatting string with **%d**, which tells it to store the decimal integer conversion of the characters you type. The second argument in the function call is **&reps**, which tells the **scanf** function the address of the variable where you want the result. That's the way the **scanf** function is designed – instead of a variable value, it needs a variable address to do its job.

```
  scanf("%d", &reps);                          // Scan reps user types
```

This **printf** call helps verify that the **scanf** function did its job by displaying the value **reps** stores.

```
  printf("\nCounting to %d\n", reps);             // Display value scanned
```

The rest of the program counts from 1 to **reps**, in a manner similar to the Counting Loops activity.

```
  for(int n = 1; n <= reps; n++)                // Count to value scanned
  {
    printf("i = %d\n", n);                       // Display each step
  }
```

```
  printf("\nDone!");                              // Display when done
}
```

Did You Know?

After **scanf** has stored the value at a variable's address, like **&reps**, the **reps** variable stores the value. So, your code can do things like **printf("reps = %d\n", reps)** to display the value of **reps**, and **for(int n = 1; n <= reps; n++)**, which uses the value of **reps** to limit how high the **for… loop** counts.

The **scanf** function is part of the stdio library. It is similar to the robot IDE simpletools library's **scan** function. Simpletools is used with the robot because it takes less memory with code that uses floating point variables. The **printf** function is also part of stdio, and similar to the **print** function from the simpletools library.

Forgetting to use **&** in the **scanf** function is a very common programming error.

scanf is kind of like the inverse of **printf**. For example, in **printf("%d\n", reps)**, the **%d** would make it display the decimal character representation of the value **reps** stores, and **\n** would make it display a newline. In **scanf("%d", &reps)**, the **%d** tells the computer to receive a decimal character representation of a value from the terminal, convert it to an **int** value, and store it in the **&reps** memory location. The one important difference is that **printf** would need the value of **reps**, but **scanf** uses **&reps** – the address of the **reps** variable.

The **scanf** function's formatting flags have the same meanings as their **printf** equivalents:

- **%d** – Decimal integer
- **%f** – floating point value
- **%c** – character
- **%s** – string

You will see more on **%c** and **%s** as you progress through the robot activities.

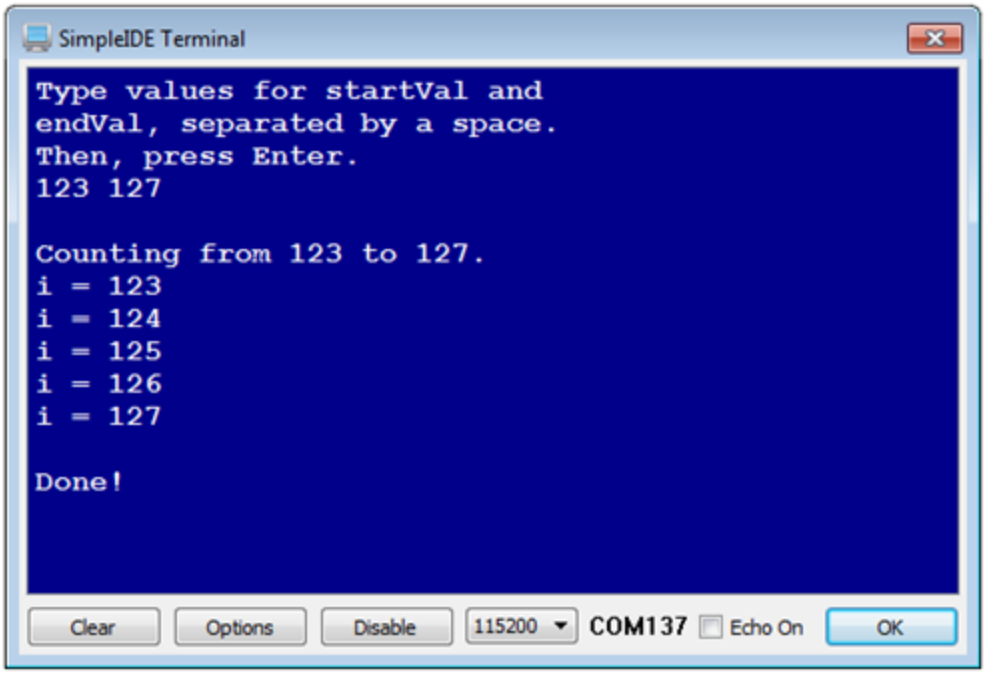
There are important differences between **scanf** and **printf**:

- The non-format flag characters in the formatting string don't matter
- Values input with **%d** terminate with the first non-numeric character.
- The **scanf** function terminates with **\n** regardless of whether you add it to the format string. [Ed. Note: in my testing with CodeBlocks the program does not work properly if **\n** is added after **%d**, so I did not include **\n** in the edited code –

For example, **scanf("%d%d", &startVal, &endVal)** will take two values that have to be separated by at least one non-numeric character, and the **scanf** call itself gets completed by **\n** [Ed. Note: but do not add **\n** to your code when using **scanf** – J. La Favre]. You could input 123, followed by a space, then 127, then press the Enter key. [But, you could also type abcdef123ghijklm127opq followed by the Enter key, and the results would be the same: **startVal** would store 123 and **endVal** would store 127. – this does not work with CodeBlocks - J. La Favre]

Try This

Here, a single **scanf** function call **scanf("%d %d", &startVal, &endVal)** captures two values. Then, a single **printf** call **printf("\nCounting from %d to %d.\n", startVal, endVal)** displays those two values, and then the **for… loop** counts from the start value to the end value.



- 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: **scan to int 2 values.** 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 your 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 pasted or entered with keyboard.

```
#include <stdio.h>


int startVal;
int endVal;


int main()
{
  printf("Type values for startVal and \n");
  printf("endVal, separated by a space. \n");
  printf("Then, press Enter. \n");
  scanf("%d%d", &startVal, &endVal);
  printf("\nCounting from %d to %d. \n", startVal, endVal);

  for(int n = startVal; n <= endVal; n++)
  {
    printf("i = %d\n", n);
  }
  printf("\nDone!");
}
```

- 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 (the terminal).

- In the terminal, type two values separated by a space. Then press the Enter key. **IMPORTANT:** Make sure the first value you type is smaller than the second value.
- Verify that the terminal displays "Counting from…to…" with the values you entered, and then counts from the first value to the second value.
- [Re-run the program, and try typing abcdefg123hijklm127opqr followed by the Enter key. Did it still work? – does not work with CodeBlocks - J. La Favre]

Your Turn

[Ed. Note: the remainder of this lesson should be skipped when using CodeBlocks because there is no function named getDec in its library – J. La Favre]

Since **`scan`** can be kind of tricky with the **`scan`** function's rule set, let's try functions from the simpletext library with a simpler set of rules for getting values from the SimpleIDE Terminal. The **`getDec`** function does not require pointers, and terminates with a press of the Enter key (meaning that instead of adding a space between each number as you have been doing, you would press enter instead). So, you could replace the **`scanf`** code block from the Try This section with code that uses **`getDec`**, and end up with the same output in your Terminal Window.

- Make sure your Try This code works.
- Replace this code block:

```
print("Type values for startVal and \n");
print("endVal, separated by a space. \n");
print("Then, press Enter.\n");

scan("%d%d", &startVal, &endVal);
```

... with this:

```
print("Type values for startVal and \n");
print("endVal.  Press Enter after each one. \n");

startVal = getDec();
endVal = getDec();
```

- Test and verify that the program still works the same as it did before modifying.

Like **print** and **scan**, the **getDec** function is part of the simpletext library, which the simpletools library includes. You can view a full list of the simpletext library's functions in SimpleIDE's Help -> Simple Library Reference. For displaying data in the SimpleIDE Terminal, look for functions that start with **put**, and for acquiring data from the terminal, look for functions that begin with **get**.

- Try replacing:

```
print("\nCounting from %d to %d.\n", startVal, endVal)
```

...with these calls to simpletext functions:

```
putStr("\nCounting from ");
putDec(startVal);
putStr(" to ");
putDec(endVal);
putStr(".\n");
```

- Again, test and verify that the program still works the same.

Although **print** and **scan** provide some convenience with fewer lines of code, there is also a penalty in terms of program size. Replacing all **print** and **scan** calls with equivalent sets of **get** and **put** calls, can often significantly reduce the program size.

- Reopen the unmodified Scan To Int project.
- Use the Build Project button to compile the code then check the total code size in the SimpleIDE window's bottom-left corner.

- Replace all **`print`** and **`scan`** functions with **`put`** and **`get`** functions.
- Use the Build Project button again, and check how much memory you saved.