Assignment 2: Variables and Arithmetic Operators

In Hwan “Chris” Baek and William Kaiser

chris.inhwan.baek@gmail.com ; kaiser@ee.ucla.edu

Introduction

In this assignment, you will learn to:

1. Declare variables,
2. Assign values to variables, and
3. Perform arithmetic operations.

Before beginning, it is important to note that this assignment must be done on the Intel Edison. Please do **NOT** use an IDE for this assignment.

Step 1: Variables and Arithmetic Operators

Before we jump into the concepts on variables [0] and arithmetic operators [1], let’s try a demo. First, you need to access the Intel Edison via serial connection or SSH. You can refer to IoT Tutorial 2 to make serial connection or IoT Tutorial 3 to connect via SSH. Once you log into the Intel Edison, enter the following commands to create a directory for this assignment and navigate into this:

```
$ mkdir c_program_assg2

$ cd c_program_assg2
```

Create and open a file in Vim using the following command:

```
$ vi variables.c
```

Press the “**i**” key to enable the **insert** mode. Then, write the following C code:

```
/*
 *  Variables and arithmetic operators
 */

#include <stdio.h>

int main() {
        int a;
        int b, c;
        a = 3;
        b = 4;
	float d;
	float e = 1234567890;
	double f = 1234567890;
	char g = 'C';
	char* h = "variables and arithmetic operators";

	c = a + b;
	d = (float) a / (float) b;

	printf("The sum of a and b: %d\n", c);
	printf("The quotient of a and b: %d\n", a / b);
	printf("The quotient of a and b: %f\n", d);
	printf("The value stored in e: %f\n", e);
	printf("The value stored in f: %f\n", f);
	printf("%c %s\n", g, h);

	return 0;
}
```

Press the **ESC** key to go back to the normal mode. Then, type "**:wq**" and press the **ENTER** key to save and quit. Now you have written C code that is ready to be compiled. Enter the following command to compile the code:

```
$ gcc -o variables variables.c
```

You can execute the program by using the following command:

```
$ ./variables
```

You will see the following output on the screen:

```
The sum of a and b: 7
The quotient of a and b: 0
The quotient of a and b: 0.750000
The value stored in e: 1234567936.000000
The value stored in f: 1234567890.000000
C variables and arithmetic operators
```

Step 2: Study the Code

Let us study **variables.c**. Some of the statements in the code may look similar to math equations to those without programming experience. However, these equation-like statements are not quite the same as math equations. This will become clear once you complete this assignment.

In mathematics, a variable is an alphabetic character that represents a numerical value. In computer science, a **variable** is a name that refers to some memory location [0], which is allocated to store values as shown in **Figure 1**. The variable type determines the size of the allocated memory. For example, memory that stores an integer "1004" and memory that stores a character "a" have different sizes.

When we **declare** [2] a variable in C, we need to specify the type in order to tell the compiler to allocate sufficient memory and check that the rules of the type are not violated in our code [3].

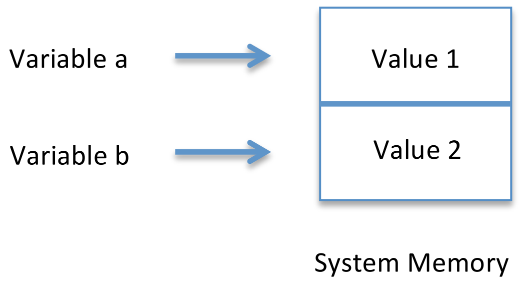

Figure 1. Variable

There are four basic variable **types** in C: **int**, **float**, **double**, and **char**. The **int** type stores integers and its size is 4 bytes (32 bits) in most platforms [4]. Due to its size, there is a range of values it can store. In other words, values beyond the range cannot be properly stored with the **int** type. The representation also affects the range and the most common representation of the **int** type is two's complement [5]. Two's complement will not be covered in this assignment. In most platforms, the **float** type stores real numbers in single-precision floating-point format [6], which also occupies 4 bytes. Again, single-precision floating-point format is not covered in this assignment. Due to its relatively small size, the **float** type does not have high precision. In case you need higher precision, you can use the **double** type. The **double** type also represents real numbers but in double-precision floating-point format [7], which occupies 8 bytes. In most platforms, the **char** type stores an **ASCII** [8] character and is size of 1 byte. **Table 1** below summarizes this.

Type	**Stored Value**	**Common Size**
int	Integers	4 Bytes (32 bits)
float	Real numbers in single-precision floating point format	4 Bytes (32 bits)
double	Real numbers in double-precision floating point format	8 Bytes (64 bits)
char	ASCII characters	1 Byte (8 bits)

Table 1. Variable Types

Enough with the concepts, let's look at the code and learn how to use variables in real practice. Line 8, `int a;` is an example of variable declaration. Declaration associates the type and the name (identifier) with the variable [2]. In this case, the variable is the **int** type and the name is "a". A name can only include letters, digits, and underscores. A name must not begin with a digit. Some examples are "myAccount", "x_axis", and "person1". As you can see in line 9, **`int b, c;`**, you can declare two or more variable in one statement if they share the same data type. Line 9 is equivalent to:

```
int b;
int c;
```

Now, let's look at line 10, **`a = 3;`**. This may look like a mathematical expression that "a" equals to 3. In fact, this is an example of value **assignment**. This rather means "store the numerical value of 3 in the memory location denoted by 'a' as an integer". You can declare a variable and assign a value to it in one statement as shown in line 13, **`float e = 1234567890;`**. In this case, it declares a float variable named "e" and assigns *1234567890* to this variable.

Line 15, **`char g = 'C';`**, declares a char variable and assigns *C* to this variable. Unlike in the previous value assignment statements, the value is surrounded by **'**s. In C, character values have to be surrounded by **'**s. Statements "a = 1" and "a = **'**1**'**" are different. The first one means "assign the numerical value of 1 to a" while the second one means "assign the character value of 1 to a".

Line 16, **`char* h = "variables and arithmetic operators";`**, seems like it declares a char variable. However, this declares a special variable called **pointer** [9]. Pointers are out of this assignment's scope. However, it is useful to know that a pointer to char variable (char*) is usually used to store a string. Unlike characters, strings are surrounded by **"**s as shown in the code.

You can also assign the values of other variables. An example of this is "float a = b;" where "b" is another float variable. Line 18, **`c = a + b;`**, is another example. In this case, the sum of value of "a" and that of "b" is assigned to "c". "**+**" is one of arithmetic operators that is used to add two operands. **Table 2** shows the arithmetic operators.

Operator	Description	Example
+	Addition	3+4 gives 7.
-	Substraction	10-6 gives 4.
*	Multiplication	6*2 gives 12.
/	Division	8/2 gives 4.
%	Modulus operator (It gives the remainder after integer division)	9/4 gives 1.
++	Increment operator	a = 10; a++ gives 11.

--	Decrement operator	a = 10; a-- gives 9.

Table 2. Arithmetic Operators

Let's skip to line 21, **`printf("The sum of a and b: %d\n", c);`**, which includes the printf function you are already familiar with. In the first assignment, we used the printf function to print simple strings such as "Hello World!". The printf function also lets us print **formatted** strings. The string argument may include **format specifiers** [10] such as *%d* in line 21. In this case, *%d* is replace with the integer value assigned to "c". Since the assigned integer value is 7, the printed string is "The sum of a and b: 7". *%d* is a format specifier, which is replaced by an unsigned integer. Line 23, **`printf("The quotient of d and e: %f\n", d);`**, includes a format specifier, *%f*, which is replaced by a floating-point number. Other format specifiers in the demo code are *%c* and *%s*. As you may guess, *%c* is replaced by a character and *%s* is replaced by a string. You can find more format specifiers at [10].

Let's look at these output strings:

```
The quotient of a and b: 0
The quotient of d and e: 0.750000
```

The first one shows the value of "a / b" and the second one shows the value of "(float) a / (float) b". In mathematics, we would say that "a / b" is 0.75. However, "a" and "b" are both int variables. The resulting value after the division is also an int type (i.e. it is an integer). Since integers do not have fractional parts, "a / b" loses its fractional part and becomes 0. On the other hand, "(float) a / (float) b" is a float type and hence does keep its fractional part as shown in the output. This is an example of **type casting**, which is a way to convert a variable from one type to another type [11]. Adding "(float)" in front of "a" converts "a" into a float variable. If "(double)" is added instead, "a" is converted into a double variable.

Now, let's look at the next output strings:

```
The value stored in e: 1234567936.000000
The value stored in f: 1234567890.000000
```

We assigned *1234567890* to both "e" (a float variable) and "f" (a double variable). However, the output shows that the value stored in "e" is changed. As mentioned above, a float variable has lower precision than a double variable. Due to its relatively low precision, a float variable cannot store *1234567890*.

Step 3: Write Code

Create a file named “**assg2_<your last name>_<your first name>.c**” (e.g. assg2_baek_chris.c). Your code should print the following:

The product of 6 and 7 is 42

1.000000 divided by 3.000000 is 0.333333

Your code must include **int** variables, **float** (or **double**) variables, **char** variables, and all arithmetic operators (those in **Table 2**). Be creative! Put comments that describe your code. Please submit a screen capture of the displayed message along with your C code.

References

[0] https://en.wikibooks.org/wiki/C_Programming/Variables
[1] https://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B#Arithmetic_operators
[2] https://www.macs.hw.ac.uk/~pjbk/pathways/cpp1/node45.html
[3] http://www.bottomupcs.com/types.html
[4] https://en.wikibooks.org/wiki/C_Programming/Variables#The_Four_Basic_Data_Types
[5] https://en.wikipedia.org/wiki/Two%27s_complement
[6] https://en.wikipedia.org/wiki/Single-precision_floating-point_format
[7] https://en.wikipedia.org/wiki/Double-precision_floating-point_format
[8] https://en.wikipedia.org/wiki/ASCII
[9] http://www.tutorialspoint.com/cprogramming/c_pointers.htm
[10]http://www.cplusplus.com/reference/cstdio/printf/
[11]http://www.tutorialspoint.com/cprogramming/c_type_casting.htm