Output: Hello, World!

Professor Don Colton

Brigham Young University Hawaii

Our task is to make the computer say hello. We will be using the Perl programming language.

Here is the first version of the program.

```
print "Hello, World!";
```

Computers take your meaning from the exact words you write. In this case, the computer will recognize your statement as having three parts. The first part is the word `print`. It recognizes this part, called a token, by the fact that it is an unbroken series of letters. Once it has identified the token it can look it up to find out what it means. The computer has a pre-defined process that goes with the word `print`.

The second part of the statement is `"Hello, World!"`, recognized in this case by the fact it starts and ends with a double quote mark. When a token starts with a quote mark (either single or double), it continues until the matching quote mark is found. The quote marks are called delimiters. Everything between is the token. This special kind of token is called a literal, or a string of letters. Notice that this token has a space in it.

The third part of the statement is the semi-colon at the end. It has a pre-defined meaning of ending the statement, or more correctly, separating the statement from the next statement. In some languages semi-colons terminate statements. In Perl they separate statements. This means that the last statement does not need a semi-colon after it.

Bad Syntax

Supposing I wrote this program. What would it do? What should it do?

```
Print "Hello, World!";
```

In this case the computer would parse three tokens just as before. However, the first token would be `Print` which is not on the computer's list of words it knows. You and I can look at it and decide it probably means `print` based on our own common sense. Someone could even program the computer to recognize `Print` as an alternate spelling for `print`. But at this point the computer would choke on our program and tell us that we have a syntax error.

What about this one?

```
print "Hello, World!".
```

The computer will not understand the dot (period, full stop) at the end of the line. That would be a syntax error. You or your friend or your tutor may look at the line and understand exactly what you mean. They can even tell you that you need to change the dot to a semi-colon. Why can't the computer do that? Computers don't have common sense.

Making it Run

We want to run our program. Let's assume we are using a Microsoft Windows computer (perating system) and that Perl is installed. Then we can use a text editor like Microsoft Notepad to key in the program and save it. When we save it, the file name must end with `.pl` (dot pee ell). That ending indicates that the file is a Perl program. Some computers will also allow `.perl`.

Let's assume you save your program to your desktop. After saving your program you can run it by double-clicking it. The screen should flash briefly. What happened? The program ran. What did it do? It printed the words "Hello, World!". Or rather, it opened a window, probably black, ran the program,

and then closed the window, all so fast that you could not tell what was happening.

Making it Wait

Let's add another line to the program. This line is intended to make the computer wait before closing the window. We will do this by preventing the program from ending.

```
print "Hello, World!";
$wait = <STDIN>;
```

In this case our program has two statements. The first will print the words "Hello, World!". The second will request input. Precisely the meaning is as follows:

`$wait` is the name of a variable. We will use many variables in the future. This is the first one we have seen. Variable names (in Perl, in the simple case) are constructed from a dollar sign, followed by a letter, followed by zero or more letters and digits. In this case, the variable name is dollar sign and the four letters w, a, i, and t.

`=` is the assignment operator. It tells Perl to put something into the variable on the left (that is, `$wait`).

`<STDIN>` is the standard input channel. The `<>` part indicates an input channel and the `STDIN` part means standard input. Normally that would be the keyboard. STDOUT is the screen.

When the program runs, this line will cause the computer to wait for the user (the human that is running the program) to type some input.

Save it and run the program by double-clicking it.

It should open a window, print the words "Hello, World!", and pause, waiting for you to type something (or nothing) and press enter (or return).

Press enter. The program will resume execution, take whatever you typed, copy it into the variable `$wait`, finish running the program, and close the window.