Learning to Program with Haiku

Lesson 1

Written by DarkWyrm

Programming involves communicating your ideas to the computer. The only problem is that computers are stupid. They will do exactly what you tell them and NOT what you mean unless what you mean is, in fact, what you have told them. If you don't completely understand that – believe me – you will soon enough.

The problem with communicating with the computer is that it doesn't speak any human language. All it knows is what to do when given a particular numerical instruction. For people to do this is very tough, so we write in one language accessible to people and the computer turns it into something it can understand. The translation can be done right when the program is run, called an interpreted language, or beforehand as is done in compiled languages. C and C++ are compiled languages, so when writing C++ programs, you write the human language (**source code**) and the compiler turns it into computer instructions (**machine code**).

In C++, all computer instructions are grouped together into blocks called **functions**. Here is an example:

```
void myFunction(void)
{
}
```

Functions may or may not require data to do their job, and they may or may not give back a result when they are done. This one, called `myFunction`, requires no data and returns none. It also does nothing, but that's OK. Here's the format for defining a function:

```
<outdata> <functionname>(indata)
{
        <instructions to do the function>
}
```

The input data is always inside a pair of parentheses and all of the function's instructions will go inside the pair of curly braces. For future reference, all parentheses and curly braces must appear in pairs.

Anyway, the compiler doesn't care how much space is in between all of this stuff, so there is plenty of room to make your code as (un)readable as you like. We could squish all of this together like this:

```
void myFunction(void){}
```

and it would still have the same result. Because whitespace, as it's called, doesn't matter, every instruction (not to be confused with a function) in C++ has a semicolon after it.

For what it's worth, we will use a style of writing code which is quite similar to what the Haiku developers use with a few tweaks. For example, code placed in between a set of curly braces is always indented one level using the tab key.

Here is our first function that really does something:

```
int TwoPlusTwo(void)
{
        return 2 + 2;
}
```

This function, called `TwoPlusTwo`, needs no input data, but returns a result. The result of this function is always an integer – any number without a decimal point. While a number is a number to people, C++ is very particular about keeping track of the types of data passed around.

Every program has one function which needs to be defined: main. It can be defined (as in telling the computer what to do for the `main()` function) in a couple of different ways, but we'll use the simplest:

```cpp
int main(void)
{
        return 1;
}
```

This program, when compiled and run, doesn't print anything, but it does return a one back to the OS when it finishes.

Let's actually make this program. Save the above code into a Source Code file named ReturnOne.cpp. Open the Terminal and navigate to the folder that has it and type this:

```
gcc -o ReturnOne ReturnOne.cpp
```

This tells `gcc` (GNU Compiler Collection) that you're going to make a program called ReturnOne (specified by the -o ReturnOne) and you're going to use ReturnOne.cpp to make it.

Functions can call other functions, too, but only if the computer knows about them. This will generate an error when compiled:

```cpp
int main(void)
{
      PushTheRedButton();
      return 1;
}
```

Save this as RedButton.cpp and compile it with this command:

```
gcc -o RedButton RedButton.cpp
```

The computer doesn't recognize the function `PushTheRedButton()`, so it doesn't know what to do when it sees it and complains. If we tell the computer what it does, it will compile the program properly. Change RedButton.cpp to look like this and compile:

```cpp
void PushTheRedButton(void)
{
}

int main(void)
{
      PushTheRedButton();
      return 1;
}
```

PushTheRedButton doesn't do anything, but the computer doesn't care. The other way to call other functions is to use libraries built into the system. We do this by linking a library with our program and telling the compiler what functions are in the library. Now we will do something almost useful. Save the below code into HelloWorld.cpp and compile with `gcc -o HelloWorld HelloWorld.cpp`

```
#include <stdio.h>

int main(void)
{
        printf("Hello world!\n");
        return 1;
}
```

There are two parts that are unfamiliar here. The line starting with #include tells a part of the compiler called the **preprocessor** to look up function definitions in the system header `stdio.h`. `stdio.h` is the name of a file which defines a lot of standard input and output functions, like `printf()`. The angle brackets – < and > – tell the preprocessor that these are system headers. We can make our own, but we'll get to that another time.

The other unfamiliar part here is the part inside the parentheses for `printf()`. Everything in the quotes will be printed on the screen if you run this program from the Terminal. This kind of data is called a **string** (as in string of characters).

Question:

What happens if you remove the backslash and the letter n from the end of the string so that it looks like this: `"Hello world!"`

Assignment:

Write a program which prints a box on the Terminal using minuses and pipe symbols like this:

```
 ----------
|          |
|          |
|          |
|          |
 ----------
```

Question for thought:

How could you write a program which draws two boxes without having to do a ton of typing?