Getting Started with Coding in C

This lab will introduce you to the basics of coding in C, give you experience with some of the common errors you may encounter, and cover some of the specific functions you will use in your shell project.

Hello World

Where else to start? Open your favorite text editor and enter the following code, then save the file as "`hello.c`".

```
#include <stdio.h>

int main() {
    printf("Hello World!\n");
    return 0;
}
```

Navigate in terminal to the folder where you saved your file, and compile your program using the command

```
gcc -o hello hello.c
```

If compilation is successful you should see a new file in the folder named `hello`. This is the executable that was just created. To run that executable simply type `./hello`

Now open `types_printf.c` and read through its code, especially the comments. Also read the doc in printf-doc.

Make sure you can answer the following questions about your Hello World program, and don't be afraid to experiment a little! These are simply a few things I thought of off the top of my head, try to think of some other questions yourself and see what happens when you make changes.

1. What is the purpose of the first line? What is "`stdio.h`"?
2. Did we have to use the function main()? Could we have named it something else? What is that "int" listed before main()? Could we have put "void" there instead?
3. What is the purpose of the "\n" in the printf statement? What if we had left it out?
4. Why do we return 0? Do we have to have a return at all? Could it be a different value?
5. When compiling the program, did we have to include the "-o hello"? What happens if you use a different name than "hello"? What happens if you leave the "-o name" part out completely?

printf/scanf

Getting user input and displaying output are actually simpler in C than in Java. The main functions used are printf for output and scanf for input. You saw how to use printf to display a single string in your Hello World program, but it's also straightforward to display any output with very precise formatting. Using the example program and printf-doc you looked at above, update your Hello World program as follows:

1. declare a variable of type "int" named "age" and give it an initial value.
2. display the following (in the same format) instead of "Hello World", where <age> displays the value of your age variable.

```
Hello!
I am <age> years old.
```

```
How old are you?
```

Now you can use scanf to get the user's response. scanf uses the same type codes as printf, and you must know what type you are intending to read from the user. In this case we want an integer, so the appropriate call to scanf would be

```
scanf("%d", &age)
```

where age is the name of your variable of type int. Notice we must use the & before age, that is because scanf wants the memory address at which to store the value it reads, and that's exactly what putting the & before the variable does, gives the memory address instead of the variable value. You can also use scanf to read formatted input, for example if you wanted the user to enter coordinates in the form "(x,y)" you could use the following:

```
int x, y;
scanf("(%d,%d)", &x, &y);
printf("(%d,%d)", x, y);
```

Try this program. Change your Hello World program to print your birthday using separate variables for month, day, and year in the format "month/day/year", then ask the user to input their birthday in the same format and read it correctly.

pointers

Look at the example program `pointers.c` from the previous link. Run it and read the comments to understand what's going on. Again, experiment and ask questions! The following link has a decent tutorial on C pointers as well:

http://pw1.netcom.com/~tjensen/ptr/pointers.htm

readline

A common way to get input is often to read from standard in as if it were a file instead of using scanf. The following code will read one full line of input as a string (so you would need to parse and convert it into other types as required).

```
size_t MAX_WORD_LENGTH = 128;
int bytes_read;
char *buf;
buf = (char*) malloc( MAX_WORD_LENGTH+1 );
bytes_read = getline(&buf, &MAX_WORD_LENGTH, stdin);
```

The file test.c gives an example of using readline() to get input from the terminal, as well as a couple of other string tricks.

For further reading

Here's a great set of slides that describes C in terms of Java, I highly recommend looking through it!

http://www.cs.cornell.edu/courses/cs414/2005sp/cforjava.pdf

Examples

The file strings.c has even more string examples.

The example C programs I showed in class are all included in the Shell Project materials folder on Moodle. Download those and make sure you really understand how they work!