

Lab 1

26 Mar 2009

In this lab, you will write some simple programs to get started with C.

1 But first: the executable path

When you want to execute a program, that program could be living in a number of different places; for instance, `vim` and `emacs` and `java` are in the directory `/usr/bin`. The operating system will search a bunch of standard directories, known as the ‘path’, for any command you try to execute.

By default, it doesn’t always include the *current* directory, i.e. the directory you’re in right now. So if you compile a program named `foo`, in the current directory, and type

```
foo
```

you’d get an error message. One way is to explicitly say that you’re running the `foo` in the current directory, by typing

```
./foo
```

The other is to explicitly add the current directory to your path. Right now, go to your home directory and edit the `.tcshrc` file that should be there. Add the line

```
set path=(${path} .)
```

This says to set the path to whatever it currently is, plus the current directory; it’s a bit like writing “`path = path + "."`”.

Having done that, open a new terminal window and make sure there aren’t any error messages; to see what your path currently is you can type

```
env | grep PATH
```

The colon-separated list should end in a period, if all has gone well.

2 Now for the C practice

Try the following: (Recall the compilation line `gcc -std=c99 -Wall name.c`.)

- Implement a hello world program. (Just to make sure that you can compile and run C programs.)

```
#include <stdio.h>

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

- Implement a program to compute factorials. The number to be factorialed will be provided on the command line. The builtin function to convert from a string (i.e. what is given on the command line) into an int is `atoi`, declared in `stdlib.h`. (Uses command-line parameters, for loops, library functions.)

Note: what happens if you forget to provide a command line argument? (Don't worry about error-checking yet, just observe what goes wrong.)

- Implement a program to compute the n th Fibonacci number; the 0th and 1th Fibonacci number are each 1, and subsequent Fibonacci numbers are the sum of the previous two. Perform the computation the linear way by starting at zero and storing each Fibonacci number in turn into an array, until you get to the one you were trying to compute. (Uses arrays.)

Note: How big does the array need to be?

Note: What happens if you provide an n of zero or one? Or negative? (Again, no need to error-check, just observe the error.)

- Write a program that uses a `prime` function to print all prime numbers less than 100. A number is prime if it can be evenly divided by no positive integer other than 1 and itself—use the `%` operator to check for this. Remember that using the keywords `bool`, `true`, and `false` requires the `stdbool.h` library. (Uses functions, booleans.)