UNIX and "C"

This document gives a starter in how to program in C. There's very little here that's UNIX specific – all the functions described here will run on any operating system.

For more info check out the Unix man pages

(i.e., do "man -k topic-you-wish-to-search")

-or-

Unix in a Nutshell

-and

C Language Reference Manual (K&R).

Remember printf?

- Sends text to standard output (stdout).

```
void main()
   {
   printf("Hello World \n");
   }
```

- printf is a routine that is part of the C standard library (i.e., libc)
- libc is linked into your program by default.
- Other libs (i.e., libm -- math -- is not)

What about Input?

- Scanf -- reads text from standard input (i.e., stdin)

```
void main()
{
     char buffer[32];
     int i;

     scanf("%s %d", buffer, &i);
} /* Note, we use buffer and not &buffer */
```

What are stdin, stdout, stderr?

- File descriptors...or more precisely a pointer to type FILE.
- These FILE descriptors are setup when your program is run.

- So, then what about regular user files...

File I/O Operations

- fopen -- opens a file
- fclose -- close a file
- fprintf -- “printf” to a file.
- fscanf -- read input from a file.
- ...and many other routines..

fopen

```
#include<stdio.h>
void main()
  {
       FILE *myfile;
       myfile = fopen( “myfile.txt”, “w”);
  }
```

- 2nd arg is mode:
  - w -- create/truncate file for writing
  - w+ -- create/truncate for writing and reading
  - r -- open for reading
  - r+ -- open for reading and writing

fclose

```
#include<stdio.h>
#include<errno.h>
void main()
    {
        FILE *myfile;
        if( NULL == (myfile = fopen( “myfile.txt”, “w”)))
                {
                perror(“fopen failed in main”);
                exit(-1);
                }
        fclose( myfile );
        /* could check for error here, but usually not needed */
    }
```

fscanf

```
#include<stdio.h>
#include<errno.h>
void main()
    {
        FILE *myfile;
        int i, j, k;
        char buffer[80];
        if( NULL == (myfile = fopen( “myfile.txt”, “w”)))
            {
            perror(“fopen failed in main”);
            exit(-1);
            }
        fscanf( myfile, “%d %d %d %s”, &i, &j, &k, buffer);
        fclose( myfile );
        /* could check for error here, but usually not needed */
    }
```

fprintf

```
#include<stdio.h>
#include<errno.h>
void main()
    {
          FILE *myfile;
          int i, j, k;
          char buffer[80];
          if( NULL == (myfile = fopen( “myfile.txt”, “w”)))
                    {
                    perror(“fopen failed in main”);
                    exit(-1);
                    }
          fscanf( myfile, “%d %d %d %s”, &i, &j, &k, buffer);
          fprintf( myfile, “%d %d %d %s, i, j, k, buffer );
          fclose( myfile );
          /* could check for error here, but usually not needed */
    }
```

Pipes

- They to are realized as a file descriptor which links either ouput to input or input to output.
  - recall doing shell commands of the form:
  - > ls -al | grep “Jan 1” | more
  - “|” is implemented as a libc call to “popen”

- Ex: let’s send e-mail from a C program...
- First, how do you “sendmail”???

To send mail use “sendmail”

- ***sendmail:*** is a unix command that allow the transmission and delivery of mail. Note – everything so far in this document has applied to “C” in general – but sendmail is a UNIX specific command.
- extremely complicated program and it is full of security holes (i.e., never run sendmail on your unix machine).
- To use sendmail:

```
> /usr/lib/sendmail -t
To: jbreecher@clarku.edu
From: bogus
Subject: test
This is a test!!.
.                    /* NOTE: the “.\n” here is needed to terminate */
>
```

Putting it all together with pipes

```
#include<stdio.h>
#include<errno.h>
void main()
    {
        FILE *mailpipe;
        if( NULL == (mailpipe = popen( “usr/lib/sendmail -t”, “w”)))
                {
                perror(“popen failed in main”);
                exit(-1);
                }
        fprintf( mailpipe, “To: chrisc@cs.rpi.edu \n” );
        fprintf( mailpipe, “From: bogus \n” );
        fprintf( mailpipe, “Subject: test \n” );
        fprintf( mailpipe, “This is a test. \n” );
        fprintf( mailpipe, “.\n” );
        pclose( mailpipe );
        /* could check for error here, but usually not needed */
    }
```

Other useful commands...

- fgets(char *buffer, int maxsize, FILE *f);
  - retrieves a whole line at a time up to newline or EOF.
- sscanf - does scanf on a string buffer.
- sprintf - does printf into a string buffer.

- You will use these in assignment 1...
- And speaking that, let’s cover that now!!