UNIX help sheet (cse.unl.edu)

Problem Solving in C
(CSCE 105, Spring 2006)
URL: http://www.cse.unl.edu/~cstrope/csce105s06/
Jan - May 2006

For programming purposes

Log into cse.unl.edu. If you would like to work from home, go to http://ftp.ssh.com/pub/ssh/, and download SSHSecureShellClient-3.2.9.exe. After logging in to cse.unl.edu, you will be at the *prompt*:

```
username:~ >
```

$\sim$ indicates that this is your *home directory*. By typing

```
username:~ > pwd
/home/grad/cstrope
```

you will get the *path* to your home directory.

We will start by creating a directory that will be used for projects and information related to this class

```
username:~ > mkdir cse105
```

We can see this new directory by *listing* the current directory's contents

```
username:~ > ls
cse105   Desktop    KDesktop   Mail      mail
```

Each of these items is a folder in the current directory. For this class, we want to work in the cse105 directory. To change from the *home* directory into the cse105 directory

```
username:~ > cd cse105
username:~/cse105 >
```

Notice that in the *prompt* (the text before the > symbol), the cse105 now appears. This tells us that we are in the cse105 directory. Now, we would like to create folders for each of the homework assignments that we will do over the semester. Name these hw1, hw2, hw3, hw4, and hw5 for the five assignments. Create the folders as done above. You may also want to create folders for each of the labs you will be doing over the course of the semester.

Change into the `hw1` directory. In order to create programs, we will need what is called a *text editor*. Text editors are simply programs in UNIX that are used to edit text documents, similar to Notepad in windows. The text editor we will often use is called *pico*. In order to create a new document, call it `program.c` (`.c` indicates that this will be a C program)

```
username:~/cse105/hw1 > pico program.c
```

After this is done, you will be taken to a new screen, with the heading in the middle reading `File: program.c`, and some commands on the bottom of the screen. The commands on the bottom of the screen are `pico` commands. For example, ^G is the "Get Help" command, performed by holding the `Ctrl` button and pressing 'G'. In order to exit `pico`, use the command ^X.

`pico` **Commands:**

After creating a C program, we will need to *compile* the program. To do this on `cse.unl.edu`

```
username:~/cse105/hw1 > cc program.c
```

This will compile the program. If there are any errors in the program, they will be listed below this prompt.

Check this document occasionally, as there may be additions throughout the semester. This document will be considered a FAQ page, where many of the questions that arise throughout the semester will be addressed. The web page where this document resides is:

http://cse.unl.edu/~cstrope/csce105s06/examples/unix.pdf

Common Error Messages:

- `"test.c", line 8: syntax error before or at: i`

  ⟶ Most often, this means that on the previous line, a semicolon has been omitted.

- `"test.c", line 10: newline in string literal`

  ⟶ C does not permit a *string literal*, i.e. " ", to have a new line, i.e., opening quotes on one line and closing them on any line. They must be on the same line.

- ```
  "test.c", line 8: warning: implicit function declaration: sqrt
  Undefined                       first referenced
   symbol                             in file
  sqrt                                test.o
  ld: fatal: Symbol referencing errors. No output written to a.out
  ```

  ⟶ The math library is not being linked when you compile the program. To compile, use `cc <filename>.c -lm`.

- `"test.c", line 9: syntax error before or at: else`

  ⟶ This may indicate that you have more than one statement being run for an `if` statement. For example, in the code below, note that the `if` statement on the left has **2** commands associated with it: (1) `i = 25` and (2) `i = 0`. On the right, the `if` statement has only **1** command, which is a compound command that has both `i = 25` and `i = 0`.

  ```
  ----------------------------------------
  |    if(1)           |    if(1) {       |
  |        i = 25;     |        i = 25;   |
  |        i = 0;      |        i = 0;    |
  |    else            |    } else        |
  |        i = 10;     |        i = 10;   |
  ----------------------------------------
  ```

  Btw, the above figure also shows why I'm jealous of the ASCII art people...

- *(Added: Mar. 2nd)*
  `Segmentation fault` occurs during the run of your program.
  ⟶ This is likely to occur in a `scanf` statement if you forget to add the `&` before your variable, i.e. `scanf("%d",int_value);` → `scanf("%d",&int_value);`

- *(Added: Mar. 2nd)*
  During the running of my program, one of my `scanf`'s is being ignored!
  ⟶ This occurs only when you are scanning in a character (`%c`) after a previous `scanf` of any type. The reason for this is as follows:

  - The first `scanf` stops the program, waits for input.
  - You type in the desired input, and then press *enter*.
  - The first `scanf` reads in the input.
  - The second `scanf` reads in the `newline`! Note that a newline is an ASCII 10, which is an acceptable value for a character. This newline character is not taken off of the buffer by the previous `scanf("%d"), scanf("%c")` or `scanf("%lf")`.

  To fix this, place a space between the opening quotes and the placeholder, i.e. `scanf(" %c",&ch);`

- **If you run across any other error messages, let me know!**
  ```