Assignment 5. C programming and debugging

Useful pointers

- Ian Cooke, C for C++ Programmers (2004-06-08). Note that it describes C89; C99, the current version of C, supports `//` comments, declarations after statements, and (if you include <stdbool.h>) `bool`.
- Steve Holmes, C Programming (1995)
- Parlante, Zelenski, et. al, Unix Programming Tools (2001), section 3 — `gdb`.
- Julian Seward et al., Valgrind Quick Start Guide (2010-10-21)
- Richard Crook, KDE / Qt programming tutorials using KDevelop & QtCreator (2009-11-07)
- Julian Seward et al., Valgrind User Manual (2010-10-21)
- Richard Stallman, Roland Pesch, Stan Shebs, *et al.*, Debugging with GDB (2010)

Laboratory: Debugging a C program

As usual, keep a log in the file `lab5.txt` of what you do in the lab so that you can reproduce the results later. This should not merely be a transcript of what you typed: it should be more like a true lab notebook, in which you briefly note down what you did and what happened.

You're helping to maintain an old stable version of `coreutils`, but that version has a bug in its implementation of the `ls` program. (Actually, it has two bad bugs, but for now we'll just look at the first one.)

The bug is that `ls -t` mishandles files whose time stamps are very far in the past. It seems to act as if they are in the future. For example:

```
$ touch -d '1918-11-11 11:00 GMT' wwi-armistice
$ touch now
$ sleep 1
$ touch now1
$ ls -lt wwi-armistice now now1
-rw-r--r-- 1 eggert eggert 0 Nov 11  1918 wwi-armistice
-rw-r--r-- 1 eggert eggert 0 Feb  5 15:57 now1
-rw-r--r-- 1 eggert eggert 0 Feb  5 15:57 now
```

Build this old version of coreutils as-is, and then again with this renaming patch. What problems did you have when building it as-is, and why did the renaming patch fix them?

Reproduce the problem. Use a debugger to figure out what went wrong and to fix the corresponding source file.

Construct a new patch file `lab5.diff` containing your fixes, in the form of a ChangeLog entry followed by a `diff -u` patch.

Homework: Binary sort and remove duplicates

Write a program `binsortu` that takes a single argument *N* (where *N* is a positive decimal integer), reads *N*-

byte records from standard input, sorts them in lexicographic order, removes duplicates, and outputs the result to standard output. If standard input ends in a partial record that contains fewer than *N* bytes, `binsortu` should treat it as if it were padded with trailing null (`'\0'`) bytes, which should cause this padded record to appear relatively early in the output.

Use lexicographic byte comparison to compare each record, in the style of the `memcmp` function, and remove duplicates; for example, the following input:

```
DACC
ABDB
CDDB
BCCB
BCCB
CDDB
DACC
BDCC
```

should generate the following output:

```
ABDB
BCCB
BDCC
CDDB
DACC
```

assuming the record size is five bytes (four letters plus a newline byte).

Use `<stdio.h>` functions to do I/O. Use `malloc`, `realloc` and `free` to allocate enough storage to hold all the input, and use `qsort` to sort the data. Do not assume that the input file is not growing: some other process may be appending to it while you're reading, and your program should continue to read until it reaches end of file. Accept any positive decimal integer string *N* that `strtoul` would. If the program encounters an error of any kind (including input, output or memory allocation failures, missing or extra arguments, an *N* that is not a positive integer or is too large to be represented), it should report the error to `stderr` and exit with status 1; otherwise, the program should succeed and exit with status 0. The program need not report `stderr` output errors.

For example, the command:

```
printf '\x00CA\x00D\x00\x00B' | ./binsortu 2 | od -c
```

should output:

```
0000000  \0   B  \0   C   A  \0   D  \0
0000010
```

Also, assuming there are no control characters other than tabs and newlines in your source file, the command:

```
export LC_ALL='C'
maxsize=$(expand <binsortu.c | awk '
  BEGIN { maxlen = 0 }
  { if (maxlen < length) maxlen = length }
  END { print maxlen + 1 }
')
```

```
expand <binsortu.c |
awk -v maxsize=$maxsize '{printf "%-*s\n", maxsize - 1, $0}' |
  ./binsortu $maxsize |
  sed 's/[[:space:]]*$//' > test1.txt
sed 's/[[:space:]]*$//' binsortu.c | sort -u >test2.txt
cmp test1.txt test2.txt
```

should output nothing.

Submit

Submit the following files.

- The files `lab5.txt` and `lab5.diff` as described in the lab.
- A single source file `binsortu.c` as described in the homework.

All files should be ASCII text files, with no carriage returns, and with no more than 200 columns per line. The C source file should contain no more than 132 columns per line. The shell commands

```
expand lab5.txt lab5.diff | awk '/\r/ || 200 < length'
expand binsortu.c | awk '/\r/ || 132 < length'
```

should output nothing.

$Id: assign5.html,v 1.21 2011/02/08 05:24:30 eggert Exp $