CSC161 2011S *Imperative Problem Solving*

Assignment 5: Writing Linux Utilities

Assigned: Friday, 25 February 2011
Due: 11:00 p.m., Wednesday, 2 March 2011

Summary: In this assignment, you will build a variety of simple Linux utilities.

Purposes: To give you more experience with conditionals in C. To further ground your understanding of binary search. To remind you of the power of the Unix model.

Expected Time: Three to four hours.

Collaboration: I encourage you to work in groups of two or three students. However, you may work on your own or in groups of up to size four. You may discuss the assignment with anyone you wish, provided you clearly document such discussions.

Submitting: Submit tarballs of your solutions on Pioneerweb.

Warning: So that this assignment is a learning experience for everyone, I may spend class time publicly critiquing your work.

Assignment

Problem 1: Sums

Recently, you wrote a program that printed all of the numbers read from standard input. Here's a variant of that program that prints all of the integers read from standard input.

```
/**
 * ints - extract all the integers from standard input and print them
 *   one per line.
 */

// +---------+------------------------------------------------------
// | Headers |
// +---------+

#include <stdio.h>      // For printf, getchar, and putchar
#include <ctype.h>      // For isdigit


// +-------+--------------------------------------------------------
// | Types |
// +-------+

/**
 * The states of our program.  We can either be inside our outside
 * of a number.
 */
enum state { IN, OUT };


// +------+-----------------------------------------------------------
// | Main |
// +------+

int
main ()
{
  int ch;                         // An input character
  int sign = 0;                   // The sign, if there is one.
  enum state state = OUT;         // The state of input; initially outside.

  // Repeatedly read and process characters
  while ((ch = getchar ()) != EOF)
    {
      // Is it a digit?  If so, print it out and note that we’re now
      // in a number.
      if (isdigit (ch))
        {
          // Print out the sign, if one is waiting
          if (sign)
            {
              putchar (sign);
              sign = 0;
            }
          // Print it out
          putchar (ch);
          // And note that we’re in a number.
          state = IN;
        } // if isdigit(ch)

      // Is it a plus or minus sign?  If so, remember it.
      else if ((ch == ’+’) || (ch == ’-’))
        sign = ch;

      // Otherwise, it’s neither a digit or a sign.  If we were in a
      // number, we should print a newline.
      else if (state == IN)
        {
          putchar (’\n’);
          state = OUT;
          sign = 0;
        } // leaving a number

      // It’s not a digit or a sign, and we’re not in a number.  Clear
      // any signs that are set.
      else
        sign = 0;
    } // while

  // And we’re done
  return 0;
} // main
```

Write a program, `sum`, that reads integers from standard input and computes the sum of those integers.

For example,

```
$ cat nums
5
1
2
6
$ cat nums | ./sum
14
```

As you might guess, we can use `ints` in conjunction with `sum` to extract all the integers from a file and then sum them.

```
$ cat grades
Joe:Smith:HWA:10
Joe:Smith:HWB:8
Joe:Smith:HWC:7
Joe:Smith:HWD:-1:Penalized for ...
$ cat grades | ./ints | ./sum
24
```

Problem 2: Selecting Numbers

Write a program, `select`, that takes two command-line parameters: an binary operator and an integer, and that reads numbers froms standard input and selects (prints out) those of the numbers for which the result of applying the binary operator to the value gives a nonzero result.

For example,

- `./select '>' 5` - prints out the numbers from standard input that are greater than 5
- `./select == 3` - prints out the numbers from standard input that are equal to 3. (Okay, that's somewhat silly, but it can be helpful.)
- `./select & 2` - prints out the numbers from standard input that have their two's bit set.

You should support the operators `<=`, `<`, `==`, `!=`, `>`, `>=`, `|`, `&`, and `^`.

Since < and > are used for redirection, you'll need to put them in quotation marks when you use them on the command line.

Problem 3: Counting Numbers

Suppose the file `grades.txt` contains grades in the form given above. Write a command that counts how many grades are in the range 50-80.

Problem 4: Improving Binary Search

Consider the following program that extends K&R's binary search.

```
/**
 * binsearch X V0 V1 ... VN
 *   Search for X in [V0,V1,V2,...,VN]
 *
 * Based on example on p. 58 of K&R 2nd edition.
 *
 * Disclaimer: This program does not do significant error checking.
 */

// +---------+-----------------------------------------------------
// | Headers |
// +---------+

#include <stdio.h>      // For printf
#include <stdlib.h>     // For atoi


// +-----------+---------------------------------------------------
// | Constants |
// +-----------+

/**
 * Are we testing our program, which means we want extra output?
 */
#define TESTING 0


// +-----------+---------------------------------------------------
// | Utilities |
// +-----------+

/**
 * binsearch (x, v[], n)
 *   Find the index of x in the sorted array v of size n.
 *   Return -1 if the value is not found.
 */
int
binsearch (int x, int v[], int n)
{
  int low;      // The low end of the part we're searching
  int high;     // The high end of the part we're searching
  int mid;      // Tee middle of the part we're searching

  // Initially, we want to search the whole array
  low = 0;
  high = n - 1;

  // Continue searching parts of the array until we find the
  // element (return in the middle) or we run out of elements
  // to look at.
  while (low <= high)
    {
      mid = (low+high) / 2;     // Potential bug, but ok for now
      if (TESTING)
        {
          printf ("x = %d, low = %d, mid = %d, high = %d\n",
                  x, low, mid, high);
        }
      // Does x belong in the left half?
      if (x < v[mid])
        high = mid - 1;
      // Does x belong in the right half?
      else if (x > v[mid])
        low = mid + 1;
      // Otherwise, x is v[mid]
      else
        return mid;
    } // while (low <= high)

  // Since we’ve looked at the whole array, the value isn’t there.
  return -1;
} // binsearch

/**
 * looks_like_int(str)
 *   Determine if the string looks like an integer.
 */
int
looks_like_int (char *str)
{
  int i = 0;

  // Skip over leading sign.
  if ((str[0] == ’+’) || (str[0] == ’-’))
    ++i;

  // Consider remaining characters.
  while (str[i] != ’\0’)
    {
      if (! isdigit (str[i]))
        return 0;
      ++i;
    } // while

  // We’ve read through all the characters.  It must be an integer.
  return 1;
} // looks_like_int

/**
 * print_vector(v, n)
 *   Print a vector of ints
 */
void
print_vector (int v[], int n)
{
  int i;        // A counter variable.  What else?
  printf ("{");
  if (n > 0)
    {
      printf ("%d", v[0]);
      for (i = 1; i < n; i++)
        printf (",%d", v[i]);
    } // if (n > 0)
  printf ("}");
} // print_vector


// +------+-------------------------------------------------------
// | Main |
// +------+

int
main (int argc, char *argv[])
{
  int x;                // The value we're searching for
  int n = argc-2;       // The size of the vector we're searching
  int v[n];             // The vector to search
  int i;                // Everyone's favorite counter variable
  int index;            // The index of x in v.

  // Sanity check
  if (argc < 2)
    {
      printf ("Usage: %s X V0 V1 ... VN-1\n", argv[0]);
      return -1;
    }

  // Grab x
  if (! looks_like_int (argv[1]))
    {
      printf ("Invalid X.  Given %s, expected an integer.\n", argv[1]);
      return 1;
    }
  x = atoi (argv[1]);

  // Build v
  for (i = 2; i < argc; i++)
    {
     if (! looks_like_int (argv[i]))
       {
        printf ("Invalid V[%d].  Given %s, expected an integer.\n",
                i-2, argv[i]);
        return i;
       }
      v[i-2] = atoi (argv[i]);
    } // for

  // Find the index of x
  index = binsearch (x, v, n);

  // Print the result
  if (index == -1)
    {
      printf ("Could not find %d in the vector ", x);
      print_vector (v, n);
      printf ("\n");
    } // if not found
  else
    {
      printf ("The index of %d in the vector ", x);
      print_vector (v, n);
      printf (" is %d.\n", index);
    } // if found

  // And we're done
  return 0;
} // main
```

Rewrite the `binsearch` function so that it does one test inside the loop instead of two.

Problem 5: Searching Strings

Consider the following program fragment:

```
  char *tlas[] = {
    "GNU",
    "IBM",
    "LOL",
    "SAD",
    "SAM",
    "TLA"
  };
  char *defns[] = {
    "GNU's Not Unix",
    "International Business Machines",
    "Laugh Out Loud",
    "Seasonal Affective Disorder",
    "Sam's Amazing Mastery",
    "Three-Letter Acronym"
  };
```

As you can tell, the *i*th element of `defns` contains a definition for the *i*th element of `tlas`.

Write a program, `tla`, that takes a three letter acronym from the command line and prints out its expansion. For example,

```
$ tla IBM
IBM is International Business Machines.
$ tla FOO
Sorry, FOO is undefined.
```

Your strategy is simple: Use binary search to determine the position of the tla in `tlas` and then print out the corresponding element of `defns`.

You'll need to modify `binsearch` to deal with strings. You'll find the `strcmp` procedure useful. `strcmp (`*`str0`*`, `*`str1`*`)` returns

- a negative number, if *str0* alphabetically precedes *str1*;
- a positive number, if *str0* alphabetically follows *str1*;
- zero, otherwise.

Submitting Your Homework

Using `script`, build logs of sample runs of your programs.

Put those logs, your source code, and any other files you deem appropriate in a directory called `usernames.hw5`.

Make a tarball of that directory.

Submit the tarball on Pioneerweb under the Assignment 5 link.