Burt Rosenberg

Problem Set 3

Out: 12 September, 1995
Due: 19 September, 1995

Tutor/Help Announcement

Our grader, Ashish Sehgal, will be available in Ungar, Room 426, on Wednesdays and Fridays, 1–2 PM, and Tuesdays, 11 AM until noon, to help with homeworks.

Programming Assignment

Write the "unique words program", my_uniq.c. Usage is:

```
cs> my_uniq < infile > outfile
```

The program produces a list of words in infile, each word listed only once, with an integer count of the number of times the word appears in infile.

The program actually reads from standard input and writes to standard output, but the usage above demonstrates the use of the symbols < and > to redirect output and input to and from files.

Write two versions of the program, a simple and an improved version, and compare their run time on the provided, large sample text:

1. **Simple:** Search a linked list for the occurrence of a word in the input. If found, increment the count, else add the word with an initial count of 1 to the head of the list.
2. **Advanced:** Search a linked list for the occurrence of a word in the input. If found, increment the count *and* move the word to the head of the list, else add the word with an initial count of 1 to the head of the list.

Copy the program,

```
/home/escambia/mth220/pub/my_uniq_template.c
```

into your work directory and use it as a head-start on this project. Provided in this program are:

- The struct definitions needed.
- The function one_word which takes care of input. Each call to one_word returns the next word from in the input, else the pointer value NULL if the end-of-file has been hit. This function changes all letters to lower-case and removes all punctuation and numbers.
- The specification of various functions performing small, important tasks towards the accomplishment on the goal. Write the bodies and then orchestrate their call in the main routine.

The large test input is found in,

```
/home/escambia/mth220/pub/constitution.txt
```

You might test your working program against the solution,

```
/home/escambia/mth220/pub/my_uniq_soln
```

Using the option `-x`, i.e. `my_uniq_soln -x` will enable the move-to-front code.

Assertions

An important topic of this course is *correctness*, writing programs with few bugs. This is a difficult task. Three important analytic tools have been developed to help you in this task:

1. *Assertions,*
2. *Loop Invariants,* and
3. *Data Structure Invariants.*

In this section we will discussion assertions.

Assertions are logical formulas which are evaluated during the program's running, as if they were part of a normal if-statement. But they do no real work. They exist to check the truth of crucial assumptions. You "assert" that the situation expressed in the logical formula is realized. Assertions fall into two broad categories:

1. *Preconditions* check that the prerequisites to run a block of code have been satisfied. These occur (are "asserted") just before the block of code.
2. *Postconditions* check that the goals of a block of code have been attained. These are asserted just after the block of code.

Assertions can be written either informally, using much language to explain the assertion, or formally, with a terse mathematical formula. They can either be in the form of comments, which actual do nothing except express the programmer's wish that at a precise point in the code, a certain fact be true, or actual code which can stop the program if the assertion is violated.

Here are some examples to illustrate these points:

- **A precondition to a subroutine:**

```
int do_not_spin_my_wheels( int i ) {
   /* ASSERT: i>=0 */
   while (i!=0) i-- ;
}
```

- **A postcondition on a block of code:**

```
void get_the_sizes_right( int *a, int * b ) {  int t ;
   if ( *a<*b ) { t = *a ; *a = *b ; *b = t ; }
   /* ASSERT: a is greater than or equal to b */
}
```

- **Fitting preconditions to postconditions:**

```
get_the_sizes_right(&a,&b) ;
do_not_spin_my_wheels( a-b ) ;
```

In the third example, note how the postconditions (goals) of the first function feed into the preconditions (prerequisites) of the second function. Do assertions make it clear that the pair of functions will work together correctly?