Burt Rosenberg

My_Uniq_Template

Out: 12 September, 1995

```
/*
 ************************************************
 * my_uniq_template -- template for             *
 *    Homework 3 - the Unique Words Program.    *
 *                                              *
 * Author: Burt Rosenberg                       *
 * Course: MTH 220/317                          *
 * Date:   September 1995                       *
 * Usage:  my_uniq < infile > outfile           *
 ************************************************
*/

#include<stdio.h>
#include<stdlib.h>  /* needed for malloc */
#include<strings.h> /* needed for strcat, and others */
#include<ctype.h>   /* needed for isupper */

#define LINEBUFSIZE 100
char line_buffer[LINEBUFSIZE] ;
char * lb_pntr = line_buffer ;

#define WORDSIZE 25
struct MyList {
   int count ;
   char word[WORDSIZE] ;
   struct MyList *next ;
   } ;

/*
 * a_list of type MyList is declared struct MyList *a_list
 * a_ticket of type Ticket is declared struct MyList *a_ticket,
 *  the same as a_list, except possibly for conventions about
 *  an empty ticket versus an empty list.
*/

/*
 ****************************************
 *
 *   SUBROUTINES: (in order of appearance)
 *
 *      (written by prof)
```

```
 *      clean_up
 *      one_word
 *
 *      (written by student)
 *      create_MyList
 *      push_MyList
 *      find_MyList
 *      count_MyList
 *      is_empty_Ticket
 *      print_MyList
 *      movetofront_MyList
 *
 *      (others that you add)
 *
 ****************************************
*/

void clean_up( char * buf ) {

/* cleans up the given string, forces characters to be either
   a-z (lowercase alpha) or a blank */

   while ( *buf!='\0') {
      if ( isupper(*buf) ) {
         *buf = tolower(*buf) ;
      }
      if ( !isalpha(*buf) ) {
         *buf = ' ' ;
      }
      buf++ ;
   }
}

char *one_word( void ) {

/* returns a pointer to the next word in the input
   stream, or NULL for end of file
*/

   char * temp ;

   while ( *lb_pntr==' ' ) lb_pntr++ ;

   /* LOOP INVARIANT: *lb_pntr is either a character or a null. */
```

```
   while ( *lb_pntr=='\0' ) {
     if ( fgets(line_buffer,sizeof(line_buffer),stdin)==NULL ) {
        /* end of file reached */
        return(NULL) ;
     }
     clean_up(line_buffer) ;
     lb_pntr = line_buffer ;
     /* ASSERT: some characters to process */
     while ( *lb_pntr==' ' ) lb_pntr++ ;
   }

   /* ASSERT: lb_pntr points to first character of work to return */
   temp = lb_pntr ;

   while ((*lb_pntr!=' ') && (*lb_pntr!='\0')) lb_pntr++ ;
   /* ASSERT: *lb_pntr either a blank or a null */
   if (*lb_pntr!='\0') {
      /* if the word to return is not followed by a null, place
         a null following the word, and advanced the pointer
         over the null, for the next call to one_word. */
      *lb_pntr++ = '\0' ;
   }

   return(temp) ;
}

/*
 *********************************************
 * subroutines written by student            *
 *********************************************
*/

struct MyList *create_MyList(void) {
/* Creates and returns an empty MyList */
}

struct MyList *push_MyList( struct MyList *ml, char *w ) {
/* Pushes word w onto the head of MyList ml, initializing
   word's count to 1, and returning the new MyList. */
}

struct MyList *find_MyList( struct MyList *ml, char *w ) {
/* Returns a ticket to found word w on MyList ml. A special
   value is returned if word is not found on ml. */
```

```
}

int is_empty_Ticket( struct MyList *ticket ) {
/* Returns TRUE (integer 1) if ticket is empty, that is,
   ticket return by find_MyList when word not found on list.
   Else returns FALSE (integer 0). */
}

int count_MyList( struct MyList *ticket ) {
/* Increments the count of item indicated by ticket. */
}

void print_MyList( struct MyList *ml ) {
/* Prints the MyList ml */
}

struct MyList *movetofront_MyList( struct MyList *ml,
   struct MyList *ticket ) {
/* moves the element ticket in MyList ml to the front of
   ml, returning the new MyList. */
}

main() {

   /* Demonstration program for using one_word. */
   char *s ;
   while ( s = one_word() ) {
      printf("%s\n", s ) ;
   }

}
```