8 Lecture: Programming Demonstration: uniq

Outline:

Announcements
Programming Review: uniq
Sec01's uniq
Sec03's uniq

8.1 Announcements

- Coming attractions:

Event	Subject	Due Date			Notes
lab03	htable	Fri	Oct 27	23:59	
asgn3	hencode/hdecode	Fri	Nov 3	23:59	
lab05	mypwd	Mon	Nov 6	23:59	
asgn4	mytar	Mon	Nov 27	23:59	
asgn5	mytalk	Fri	Dec 1	23:59	
lab07	forkit	Mon	Dec 4	23:59	
asgn6	shell	Fri	Dec 8	23:59	

  Use your own discretion with respect to timing/due dates.

- Asgn2:
  - Test scripts online:
    - `~pn-cs357/demos/tryAsgn2`
  - Only one late day allowed for asgn2.
  - `sizeof(char)` is 1
  - gprof
  - `getopt(3)` — there's a learning curve, but it's worth it.
- INTRO TO LAB03 (maybe)
- The purpose of an exercise! From here on out not a single scrap of code that is not your own.
- Thoughts brought on by uniq:
  - memory is uninitialized
  - Remember to free() things!
  - No reason to copy to your before line, move the pointers

8.2 Programming Review: uniq

Rather than publishing solutions, let's look at this problem.
See Figures 38, 40, 41 and 42.

char *fgets(char *s, int size, FILE *stream); reads in at most one less than size characters from stream and stores them into the buffer pointed to by `s`. Reading stops after an EOF or a newline. If a newline is read, it is stored into the buffer. A `'\0'` is stored after the last character in the buffer.

return s on success, and NULL on error or when end of file occurs while no characters have been read.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "readlongline.h"

int main(int argc, char *argv[]){
    /* read lines from stdin until there are no more lines.  For each line,
     * compare it to the previous line.  If they are the different, print
     * the previous line.  If the same, discard the previous line.
     */
    char *last, *next;

    last = readlongline(stdin);      /* read an initial line */

    /* now, keep reading lines until there are no more lines */
    while ( (NULL != last) && (NULL != (next=readlongline(stdin)))) {
        if ( strcmp(last, next) ) { /* print the old line if different */
            fputs(last, stdout);
        }
        free(last);                /* we're done with last now */
        last = next;
    }

    if ( last )                  /* print the last line if there is one */
        fputs(last, stdout);

    return 0;
}
```

Figure 38: A main program that uses `readlongline()`

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "readlongline.h"

int main(int argc, char *argv[]){
    /* read lines from stdin until there are no more lines.  For each line,
      * compare it to the previous line.  If they are the different, print
      * the previous line.  If the same, discard the previous line.
      */
    char *last, *next;

    last = readlongline(stdin);
    if ( last )
      fputs(last, stdout);

    while ( (next = readlongline(stdin)) ) {
      if ( !strcmp(last, next) ) {
        free(next);              /* they're the same, drop it     */
      } else {                   /* they're different, write it.  */
        fputs(next, stdout);
        free(last);             /* remember to clean up the old one. */
        last = next;
      }
    }

    return 0;
}
```

Figure 39: Another way of going about it.

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>

#define CHUNK 80

#define DEBUG

extern char *readlongline(FILE *where) {
 /* Read a string from given stream.  Returns the string, or NULL if
  * EOF is encoutered.  Approach: allocate a buffer of size CHUNK,
  * and as the string grows expand the buffer as necessary.
  * returns NULL on EOF;
  */
  char *buff;
  int sofar, len;

  /* get an initial buffer, and make it a well–formed string */
  if (NULL == (buff = (char *) malloc(CHUNK))) {
      perror("malloc");
      exit(−1);
  }
  buff[0] = '\0';
  sofar = 0;                  /*  we don't have anything yet */

  /* now, read the string, expanding as necessary.  Loop until
   * we either hit a newline of EOF
   */
  while (fgets(buff + sofar, CHUNK, where)) {
      len = strlen(buff + sofar);

      /* now, we either have a whole line or not.  Check. */
      sofar += len;           /* add in the new part of the string */
      if ( buff[sofar−1] == '\n' )
          break;              /* it's a newline, so we're done. */

      /* if we got here, it's not a newline, so we're both
       * not done and out of buffer.  Reallocate and go 'round again.
       */
      buff = (char*)realloc(buff, sofar+CHUNK);
      if ( NULL == buff ) {   /* realloc failed.   */
          perror("realloc");
          exit(2);
      }
  }

  /* Now we have the whole string, but we might have allocated too
   * much memory.  If it's empty we hit EOF, free it and return
   * NULL.  If not, trim it down to size.
   */

  if ( sofar == 0  ) {         /* EOF */
      free(buff);
      buff = NULL;
  } else {                     /* trim to size */
      buff = realloc(buff, sofar + 1);
      if ( NULL == buff ) {   /* realloc failed.   */
          perror("realloc");
          exit(2);
      }
  }

  return buff;
}
```

Figure 40: A function that reads a long line

```
#ifndef READLONGLINEH
#define READLONGLINEH

#include <stdio.h>
extern char *readlongline(FILE *where);
#endif
```

Figure 41: The header for `readlongline()`

```
MAIN=uniq

CC = gcc

CFLAGS = -g -Wall

OBJS = main.o readlongline.o

$(MAIN): $(OBJS)
	$(CC)  -o $(MAIN) $(OBJS)

readlongline.o: readlongline.c readlongline.h
	$(CC) $(CFLAGS) -c readlongline.c

main.o: main.c readlongline.h
	$(CC) $(CFLAGS) -c main.c

clean:
	rm -f $(OBJS) core* *~
```

Figure 42: And a makefile for it

8.3 Sec01's uniq

This is the uniq that we developed in sec01

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#include "rll.h"

int main(int argc, char *argv[]) {
  char *old, *new;

  /* read initial line */
  old = rll(stdin);

  if ( old ) {
    while ( (new=rll(stdin)) ) {
      if ( strcmp(old,new) ) {  /* different */
        puts(old);
        /*        free(old); */
        old=new;
      } else {                  /* same */
        free(new);
      }
    }
  }
  if ( old ) {
    puts(old);
    free(old);
  }
  return 0;
}
```

Figure 43: sec01 main program that uses `rll()`

```c
#ifndef RLLH
#define RLLH

char *rll(FILE *in);            /* read a line and return a pointer to it.
                                 * return NULL on EOF
                                 */

#endif
```

Figure 44: sec01: The header for `rll()`

```
#ifndef RLLH
#define RLLH

char *rll(FILE *in);            /* read a line and return a pointer to it.
                                 * return NULL on EOF
                                 */

#endif
```

Figure 45: sec01: `rll()`

```
CC = gcc

CFLAGS = -Wall -ansi -g -pedantic

uniq: rll.o uniq.o
        $(CC) $(CFLAGS) -o uniq rll.o uniq.o

rll.o: rll.c rll.h
        $(CC) $(CFLAGS) -c rll.c

uniq.o: uniq.c rll.h
        $(CC) $(CFLAGS) -c uniq.c

test: uniq
        /home/pnico/Class/cpe357/now/Asgn/Handin/lib/lab02/testuniq /home/pnico/Class/cpe357/now/Asgn/Handi
```

Figure 46: sec01: makefile for it

8.4 Sec03's uniq

This is the uniq that we developed in sec03

```
#include<stdio.h>
#include<stdlib.h>
#include <string.h>
#include "rll.h"

int main(int argc, char *argv[]) {
  char *old, *new;

  old = rll(stdin);

  if ( old ) {
    while ( (new=rll(stdin)) ) {
      if ( strcmp(old,new) ) {  /* different */
        puts(old);              /* print string w/newline */
        free(old);
        old=new;
      } else {                  /* same */
        free(new);
      }
    }
    puts(old);
  }

  return 0;
}
```

Figure 47: sec03 main program that uses `rll()`

```
#ifndef RLLH
#define RLLH

#include <stdio.h>

char *rll(FILE *in);          /* read a line of arbitrary size.  Return
                               * NULL on EOF
                               */
#endif
```

Figure 48: sec03: The header for `rll()`

```
#ifndef RLLH
#define RLLH

#include <stdio.h>

char *rll(FILE *in);            /* read a line of arbitrary size.  Return
                                 * NULL on EOF
                                 */
#endif
```

Figure 49: sec03: `rll()`

```
CC = gcc

CFLAGS = -Wall -ansi -pedantic -g

uniq: uniq.o rll.o
        $(CC) -o uniq $(CFLAGS) uniq.o rll.o

uniq.o: uniq.c rll.h
        $(CC) $(CFLAGS) -c uniq.c

rll.o: rll.c rll.h
        $(CC) $(CFLAGS) -c rll.c

test: uniq
        /home/pnico/Class/cpe357/now/Asgn/Handin/lib/lab02/testuniq /home/pnico/Class/cpe357/now/Asgn/Handi
```

Figure 50: sec03: makefile for it