8 Lecture: Unbuffered IO

Outline:

8.1 Announcements

- Coming attractions:

Event	**Subject**	**Due Date**	**Notes**

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

- getline is verboten
- Gradesheet snapshop
- Test, test, test. And Test!
- Reminder about the potential common final
- Assignments are out
- Enough rope to hang yourselves..
- qsort demo?
- Things to talk about

  - qsort(3)
  - pointers and memory
    * Pointers need to point to something to be useful
    * This does not mean you *must* call `malloc(3)`
    * Draw pictures as needed

8.2 qsort

```
#include <stdlib.h>

void qsort(void *base, size_t nmemb, size_t size,
           int (*compar)(const void *, const void *));
```

8.3 Thoughts on debugging technique

Slow and steady is the way...

- Build incrementally (and test at each step)
- Stress your program so faults show up early. (and test at each step)

  You want to break your program before somebody else does.
- Write defensive code: validate inputs, check return codes, etc.
- Be especially suspicious of memory manipulation:
  - Don't free things too soon.
  - Be sure to initialize things you expect to be initialized

Debug only what you wrote, not what you think you wrote

8.3.1 Programming stuff

We programmed some stuff that exist on the following pages

```
CC = gcc

CFLAGS = -Wall -ansi -g -pedantic

MAIN = baz

$(MAIN): $(MAIN).c
        $(CC) $(CFLAGS) -o $(MAIN) $(MAIN).c

test: $(MAIN) infile
        ./$(MAIN) < infile
```

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

typedef int (*ifun)(int);

int tryme(ifun fun, int x) {
  return (*fun)(x);              /* make it blindingly obvious what we're doing */
}

int foo(int x) {
  return 2*x;
}

int bar(int x) {
  return -1*x;
}

int main(int argc, char *argv[]) {
  int i,num;

  for(i=1;i<argc;i++) {
    num = atoi(argv[i]);
    printf("First function:  %d\n", tryme(foo,num));
    printf("Second function:  %d\n\n", tryme(bar,num));
  }

  return 0;
}
```

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

struct line {
  char *line;
  struct line *next;
};

#define MAX 1024

struct line* append(struct line *list, struct line *rest) {
  struct line *tail;
  if ( !list ) {
    list = rest;
  } else {
    for(tail=list;tail->next;tail=tail->next)
      /* whee */;
    tail->next = rest;
  }
  return list;
}

void print_list(struct line *l) {
  for(;l;l=l->next) {
    printf("%s",l->line);
  }
}

void free_list ( struct line *l ) {
  struct line *next;
  for(;l;l=next) {
    next = l->next;
    if ( l->line )
      free(l->line);
    free(l);
  }
}

int main(int argc, char *argv[]) {
  char buf[MAX];
  struct line *list,*new;

  list = NULL;
  while ( fgets(buf,MAX,stdin) ) {
    new = malloc(sizeof(struct line));
    if ( !new ) {
      perror("malloc");
      exit(EXIT_FAILURE);
    }
    new->line = malloc( strlen(buf) + 1 );
    if ( ! new-> line ) {
      perror("malloc");
      exit(EXIT_FAILURE);
    }
    strcpy(new->line, buf);
    new->next = NULL;
    list = append(list,new);
  }

  /* print the resutlt */
  print_list(list);

  free_list(list);

  return 0;
}
```

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

#define SIZE 15

#ifdef DONTLOOKHERE
int x;
int *xp;
void *foo(int);
void (*bar)(int);
int (*compar)(const void *, const void *)
#endif

int compare(const void *ap, const void *bp) {
  int a, b;
  a = *(int *) ap;
  b = *(int *) bp;

  return b−a;
}

void print_nums(int A[], int size) {
  int i;
  for(i=0;i<size;i++)
    printf("A[%02d] = %d\n",i,A[i]);
  putchar('\n');
}

int main(int argc, char *argv[]) {
  int A[SIZE],i;

  /* initialize array */
  for(i=0;i<SIZE;i++)
    A[i] = rand() % SIZE;

  /* print 'em */
  print_nums(A,SIZE);

  /* sort 'em */
  qsort(A,SIZE,sizeof(int),compare);

  /* print 'em */
  print_nums(A,SIZE);

  return 0;
}
```