Code Examples from Lectures on performance

word_frequency.pl

```
while ($line = <>) {
    $line =~ tr/A-Z/a-z/;
    foreach $word ($line =~ /[a-z]+/g) {
        $count{$word}++;
    }
}

@words = keys %count;

@sorted_words = sort {$count{$a} <=> $count{$b}} @words;

foreach $word (@sorted_words) {
    printf "%8d %s\n", $count{$word}, $word;
}
```

word_frequency.py

```
import fileinput,re, collections

count = collections.defaultdict(int)
for line in fileinput.input():
    for word in re.findall(r'\w+', line.lower()):
        count[word] += 1

words = count.keys()

sorted_words = sorted(words,  key=lambda w: count[w])

for word in sorted_words:
    print("%8d %s" % (count[word], word))
```

word_frequency.sh

```
tr -c a-zA-Z ' '|
tr ' ' '\n'|
tr A-Z a-z|
egrep -v '^$'|
sort|
uniq -c
```

word_frequency0.c

```
Written in C for "speed" but slow on large inputs:


% gcc -O3 -o word_frequency0  word_frequency0.c
% time word_frequency0 <WarAndPeace.txt >/dev/null
real    0m52.726s
user    0m52.643s
sys     0m0.020s
```

```
Profiling with gprof revels get function is problem


gcc -p -g word_frequency0.c -o word_frequency0_profile
head -10000 WarAndPeace.txt|word_frequency0_profile >/dev/null
% gprof word_frequency0_profile
 %   cumulative   self              self     total
time   seconds   seconds    calls  ms/call  ms/call  name
88.90      0.79     0.79    88335     0.01     0.01  get
 7.88      0.86     0.07     7531     0.01     0.01  put
 2.25      0.88     0.02    80805     0.00     0.00  get_word
 1.13      0.89     0.01        1    10.02   823.90  read_words
 0.00      0.89     0.00        2     0.00     0.00  size
 0.00      0.89     0.00        1     0.00     0.00  create_map
 0.00      0.89     0.00        1     0.00     0.00  keys
 0.00      0.89     0.00        1     0.00     0.00  sort_words
....
```

```c
#include <stdlib.h>
#include "time.h"
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

/*
 * returns the next word from the streeam
 * a word is a non-zero length sequence of
 * alphabetic characters
 *
 * NULL is returned if there are no more words to be read
 */
char *
get_word(FILE *stream) {
    int i, c;
    char *w;
    static char *buffer = NULL;
    static int buffer_length = 0;

    if (buffer == NULL) {
        buffer_length = 32;
        buffer = malloc(buffer_length*sizeof (char));
        if (buffer == NULL) {
            fprintf(stderr, "out of memory\n");
            exit(1);
        }
    }

    i = 0;
    while ((c = fgetc(stream)) != EOF) {
        if (!isalpha(c) && i == 0)
            continue;
        if (!isalpha(c))
            break;
        if (i >= buffer_length) {
            buffer_length += 16;
            buffer = realloc(buffer, buffer_length*sizeof (char));
```

word_frequency1.c

```
word_frequency0.c  with linked list replaced by binary tree - much faster:


% gcc -O3 word_frequency1.c -o word_frequency1
% time word_frequency1 <WarAndPeace.txt >/dev/null
real    0m0.277s
user    0m0.268s
sys     0m0.008s
```

```
#include <stdlib.h>
#include "time.h"
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

/*
 * returns the next word from the streeam
 * a word is a non-zero length sequence of
 * alphabetic characters
 *
 * NULL is returned if there are no more words to be read
 */
char *
get_word(FILE *stream) {
    int i, c;
    char *w;
    static char *buffer = NULL;
    static int buffer_length = 0;

    if (buffer == NULL) {
        buffer_length = 32;
        buffer = malloc(buffer_length*sizeof (char));
        if (buffer == NULL) {
            fprintf(stderr, "out of memory\n");
            exit(1);
        }
    }

    i = 0;
    while ((c = fgetc(stream)) != EOF) {
        if (!isalpha(c) && i == 0)
            continue;
        if (!isalpha(c))
            break;
        if (i >= buffer_length) {
            buffer_length += 16;
            buffer = realloc(buffer, buffer_length*sizeof (char));
```

cp0.c

copy input to output using read/write system calls for each byte - very inefficient and Unix/Linux specific

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

void
copy_file_to_file(int in_fd, int out_fd) {
    while (1) {
        char c[1];
        int bytes_read = read(in_fd, c, 1);
        if (bytes_read < 0) {
                perror("cp: ");
                exit(1);
        }
        if (bytes_read == 0)
            return;
        int bytes_written = write(out_fd, c, bytes_read);
        if (bytes_written <= 0) {
            perror("cp: ");
            exit(1);
        }
    }
}

int
main(int argc, char *argv[]) {
     if (argc != 3) {
        fprintf(stderr, "cp <src-file> <destination-file>\n");
        return 1;
    }

    int in_fd = open(argv[1], O_RDONLY);
    if (in_fd < 0) {
        fprintf(stderr, "cp: %s: ", argv[1]);
        perror("");
        return 1;
    }

    int out_fd = open(argv[2], O_WRONLY|O_CREAT|O_TRUNC, S_IRWXU);
    if (out_fd <= 0) {
```
cp1.c

copy input to output using read/write system calls for every 4096 bytes - efficient but Unix/Linux specific

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

void
copy_file_to_file(int in_fd, int out_fd) {
    while (1) {
        char c[8192];
        int bytes_read = read(in_fd, c, sizeof c);
        if (bytes_read < 0) {
            perror("cp: ");
            exit(1);
        }
        if (bytes_read <= 0)
            return;
        int bytes_written = write(out_fd, c, bytes_read);
        if (bytes_written <= 0) {
            perror("cp: ");
            exit(1);
        }
    }
}

int
main(int argc, char *argv[]) {
     if (argc != 3) {
        fprintf(stderr, "cp <src-file> <destination-file>\n");
        return 1;
    }

    int in_fd = open(argv[1], O_RDONLY);
    if (in_fd < 0) {
        fprintf(stderr, "cp: %s: ", argv[1]);
        perror("");
        return 1;
    }

    int out_fd = open(argv[2], O_WRONLY|O_CREAT|O_TRUNC, S_IRWXU);
    if (out_fd <= 0) {
```

cp2.c

copy input to output using stdio functions stdio buffers reads & writes for you - efficient and portable

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

void
copy_file_to_file(FILE *in, FILE *out) {
    while (1) {
        int ch = fgetc(in);
        if (ch == EOF)
             break;
        if (fputc(ch, out) == EOF) {
            fprintf(stderr, "cp:");
            perror("");
            exit(1);
        }
    }
}

int
main(int argc, char *argv[]) {
    FILE *in, *out;

    if (argc != 3) {
        fprintf(stderr, "cp <src-file> <destination-file>\n");
        return 1;
    }

    in = fopen(argv[1], "r");
    if (in == NULL) {
        fprintf(stderr, "cp: %s: ", argv[1]);
        perror("");
        return 1;
    }

    out = fopen(argv[2], "w");
    if (out == NULL) {
        fprintf(stderr, "cp: %s: ", argv[2]);
        perror("");
        return 1;
    }
    copy_file_to_file(in, out);
    return 0;
}
```
cp3.c

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

// copy input to output using stdio functions
// stdio buffers reads & writes for you - efficient and portable

void
copy_file_to_file(FILE *in, FILE *out) {
    char input[4096];

    while (1) {
        if(fgets(input, sizeof input, in) == NULL) {
            break;
        }
        if (fprintf(out, "%s", input) == EOF) {
            fprintf(stderr, "cp:");
            perror("");
            exit(1);
        }
    }
}

int
main(int argc, char *argv[]) {
    FILE *in, *out;

    if (argc != 3) {
        fprintf(stderr, "cp <src-file> <destination-file>\n");
        return 1;
    }

    in = fopen(argv[1], "r");
    if (in == NULL) {
        fprintf(stderr, "cp: %s: ", argv[1]);
        perror("");
        return 1;
    }

    out = fopen(argv[2], "w");
    if (out == NULL) {
        fprintf(stderr, "cp: %s: ", argv[2]);
        perror("");
```

cp4.pl

Simple cp implementation reading entire file into array

```
die "Usage: cp <infile> <outfile>\n" if @ARGV != 2;
$infile = shift @ARGV;
$outfile = shift @ARGV;
open IN, '<', $infile or die "Cannot open $infile: $!\n";
open OUT, '>', $outfile or die "Cannot open $outfile: $!\n";
print OUT <IN>;
```

cp5.pl

Simple cp implementation reading entire file into array

```
die "Usage: cp <infile> <outfile>\n" if @ARGV != 2;
$infile = shift @ARGV;
$outfile = shift @ARGV;
open IN, '<', $infile or die "Cannot open $infile: $!\n";
open OUT, '>', $outfile or die "Cannot open $outfile: $!\n";
undef $/;
print OUT <IN>;
```

fib0.c

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

int fib(int n) {
    if (n < 3) return 1;
    return fib(n-1) + fib(n-2);
}

int main(int argc, char *argv[]) {
    int i;
    for (i = 1; i < argc; i++) {
        int n = atoi(argv[i]);
        printf("fib(%d) = %d\n", n, fib(n));
    }
    return 0;
}
```

fib0.pl

```
sub fib($);
printf "fib(%d) = %d\n", $_, fib($_) foreach @ARGV;
sub fib($) {
    my ($n) = @_;
    return 1 if $n < 3;
    return fib($n-1) + fib($n-2);
}
```

fib1.pl

```
sub fib($);
printf "fib(%d) = %d\n", $_, fib($_) foreach @ARGV;
sub fib($) {
    my ($n) = @_;
    return 1 if $n < 3;
    return $fib{$n} || ($fib{$n} = fib($n-1) + fib($n-2));
}
```

fib2.pl

```
use Memoize;
sub fib($);
memoize('fib');
printf "fib(%d) = %d\n", $_, fib($_) foreach @ARGV;
sub fib($) {
    my ($n) = @_;
    return 1 if $n < 3;
    return fib($n-1) + fib($n-2);
}
```

cachegrind_example.c

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

void test0(int x, int y, int a[x][y]) {
    int i, j;
    fprintf(stderr, "writing to array i-j order\n");
    for (i = 0; i < x; i++)
        for (j = 0; j < y; j++)
            a[i][j] = i+j;
}

void test1(int x, int y, int a[x][y]) {
    fprintf(stderr, "writing to array j-i order\n");
    int i, j;
    for (j = 0; j < y; j++)
        for (i = 0; i < x; i++)
            a[i][j] = i+j;
}


int main(int argc, char*argv[]) {
    int x = atoi(argv[2]);
    int y = atoi(argv[3]);
    fprintf(stderr, "allocating a %dx%d array = %lld bytes\n", x, y, ((long long)x)*y*sizeof (int));
    void *m = malloc(x*y*sizeof (int));
    assert(m);
    switch (atoi(argv[1])) {
    case 0: test0(x, y, m); break;
    case 1: test1(x, y, m); break;
    }
    return 0;
}
```