# CSC 209 additional problems regarding shell scripting

1. Write a "glob" expression (filename wildcards in the shell, e.g. patterns like "*.c") which matches all filenames in the current directory with the following properties. Do not worry about the issue of filenames beginning with a dot.

a) filenames beginning with an 'x'

```
x*
```

b) filenames ending with an 'x'

```
*x
```

c) filenames containing an 'x', anywhere in the filename (multiple x's also match, e.g. xqcxnkxe is a match)

```
*x*
```

d) filenames beginning with a lowercase letter

```
[a-z]*
```

e) filenames containing one or more digits, anywhere in the file name

```
*[0-9]*
```

---

2. Write a *for* loop to be executed in sh which goes through your directory "/usr/wilma/letters" (you are Wilma) and for each file in the directory, executes the command "grep Fred *file*". (Don't use xargs or find.)

```
for i in /usr/wilma/letters/*
do
    grep Fred "$i"
done
```

---

3. The file "/etc/passwd" lists all user accounts on the system, one per line. The command "wc -l" counts and reports the number of lines in its input. Write a single pipeline which outputs (only) the count of how many users have an 'e' in their user information somewhere (anywhere in their line in /etc/passwd).

```
grep e /etc/passwd | wc -l
```

---

4. Write a sed command which changes all occurrences of the three letters "the" to "five". The letters must be in sequence and must be lower-case, but they needn't be a separate word, e.g. "northern" would be changed to "norfivern", and "tothether" would be changed to "tofivefiver".

```
sed s/the/five/g
```

---

5. Write shell commands to perform the following actions. You may not use perl. You can assume that no files in the affected directory/directories have names beginning with a dot, except of course for dot and dotdot ('.' and '..').

a) Delete all files in /tmp with an 'x' in their name.

b) Delete all files in /tmp with an 'x' in the file (the contents of the file, as opposed to the file name).

c) Make subdirectories named "even" and "odd", and move all other files in the current directory into one of the subdirectories in accordance with whether the size of the file in bytes is even or odd. (Don't move a newly-created "even" or "odd" directory itself into a subdirectory! Also, you might want to know that 'expr' has a '%' operator (although there is also a good answer which doesn't use expr).)

[sample solutions]

a)

```
rm /tmp/*x*
```

b)
One solution:

```
find /tmp -type f | while read filename
                    do
                       if grep -q x "$filename"
                       then
                           rm "$filename"
                       fi
                    done
```

A solution I like more:

```
cd /tmp
rm `grep -l x *`
```

These solution differ in whether they remove files in subdirectories of /tmp, but that wasn't specified in the question.

c)

```
mkdir even odd
for filename in *
do
    case "$filename" in
        even|odd)
            ;;
        *)
            case `wc -c <"$filename"` in
                *[02468])
                    mv "$filename" even
                    ;;
                *[13579])
                    mv "$filename" odd
                    ;;
            esac
            ;;
    esac
done
```

---

6. Write a shell script to delete all files in the directory /home/ajr/q6 which have exactly three lines in them (in the opinion of "wc -l"). There may be subdirectories and symlinks and other special files in this directory, which you must skip over (you must not attempt to read their contents in any way, such as invoking "wc" on them).

```
PATH=/bin:/usr/bin
for i in /home/ajr/q6/*
do
    if test -f "$i"
    then
        if test `wc -l <"$i"` -eq 3
        then
            rm "$i"
        fi
    fi
done
```

---

7. There is a plagiarism-checking program called "cheating", where "cheating file1 file2" outputs the probability, as a percentage value between 0 and 100 inclusive, that file1 and file2 are programs of students cheating off of each other. Write a shell script to compare all pairs of files in the directory /u/submit/csc209h/a1 (which contains a single plain file per student) and output the ten most likely cheating cases, one pair per line, in the format:`98 c4abcdef c4ghijkl`

```
PATH=/bin:/usr/bin

cd /u/submit/csc209h/a1
for i in *
do
    for j in *
    do
        echo `cheating "$i" "$j"` "$i" "$j"
    done
done | sort -nr | head
```

10. Write a shell script to perform Euclid's greatest common divisor algorithm. The two input values will appear on the command line. You can assume that they are integers (if present -- you still have to check the argument count). Note that the "expr" command has a '%' operator.

Sample interaction, where '$' is the shell prompt:

```
$ sh gcd 12 15
3
$ sh gcd 12 15 26
usage: gcd x y
$
```

# CSC 209 additional problems for C programming

1. Write the "nth" command, which outputs the nth field of each line (one-origin), based on a strtok-like delimitation of fields. It prints a blank output line when the input line has fewer than the specified number of fields.
Similar to grep, the first non-option argument is the n, and subsequent non-option arguments, if any, are the names of files to be processed. If there are no file names listed, the standard input is processed.
Example:
If the input consists of the two lines
**Hello, world, how are you**
**today? Myself, I am a C program with no feelings.**
then "nth 4" would produce the output
**are**
**am**
and "nth 7" would produce a blank line and then the line "program".

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


static int n;  /* the "n" in "nth" */


int main(int argc, char **argv)
{
    int c, status = 0;
    char garbage;
    FILE *fp;
    extern void process(FILE *fp);

    while ((c = getopt(argc, argv, "")) != EOF) {
        switch (c) {
        case '?':
        default:
            status = 1;
            break;
        }
    }

    if (status || optind >= argc
                || sscanf(argv[optind++], "%d%c", &n, &garbage) != 1) {
        fprintf(stderr, "usage: %s fieldnum [file ...]\n", argv[0]);
        return(1);
    }
    if (n < 1) {
        fprintf(stderr, "%s: field number must be >= 1\n", argv[0]);
        return(1);
    }

    if (optind >= argc) {
        process(stdin);
    } else {
        for (; optind < argc; optind++) {
```

```
            if (strcmp(argv[optind], "-") == 0) {
                process(stdin);
            } else {
                if ((fp = fopen(argv[optind], "r")) == NULL) {
                    perror(argv[optind]);
                    status = 1;
                } else {
                    process(fp);
                    fclose(fp);
                }
            }
        }
    }
    return(status);
}


void process(FILE *fp)
{
    char buf[500], *p;
    int i;

    while (fgets(buf, sizeof buf, fp)) {
        for (p = strtok(buf, " \t\n"), i = 1;
                p && i < n;
                p = strtok((char *)NULL, " \t\n"), i++)
            ;
        if (p)
            printf("%s", p);
        printf("\n");
    }
}
```

---

2. Write the "add" program, which takes no command-line arguments and simply adds together all of the numbers in its standard input. The input consists of free-format integers and should be read with scanf("%d",&x) or similar. scanf("%d",&x) returns 1 for a successful input, EOF (i.e. -1) for end-of-file, and 0 if you are trying to read non-numeric input. In the non-numeric-input case, you should write "add: non-numeric input\n" to stderr and not write anything to stdout. Otherwise, you should write the total (and nothing else except a terminating \n) to stdout.

Note that "`add </dev/null`" should produce the output "0".

Example use: `ls -l | nth 5 | add`

```
#include <stdio.h>

int main()
{
    int x, sum, status;

    sum = 0;
    while ((status = scanf("%d", &x)) == 1)
        sum += x;
```

```
    if (status != EOF) {
        fprintf(stderr, "add: non-numeric input\n");
        return(1);
    }

    printf("%d\n", sum);
    return(0);
}
```

---

## 3. Write a C program whose output is its input, reversed line by line.

```
#include <stdio.h>

int main()
{
    char buf[82];

    if (fgets(buf, sizeof buf, stdin)) {
        main();
        fputs(buf, stdout);
    }
    return(0);
}
```

---

**4. Write a void function which takes two FILE* arguments. It reads a sequence of integers from the first file, bubblesorts them, then outputs the sorted list to the second, one per line, no additional data. The input file is free-format; i.e. you will want to use fscanf(). It is okay if your program mistakes non-numeric input for end-of-file; i.e. you will probably want to use while (fscanf(...) == 1).**

**You could test your sortfile.c with the following:**

```
#include <stdio.h>
int main()
{
    sortfile(stdin, stdout);
    return 0;
}
```

**if you add in an appropriate declaration for sortfile [mini-solution: the above with sortfile declaration added in].**

**Your sortfile.c should accept up to a hundred numbers (exactly). If there are more, print an error message to stderr and exit; in this case, do not output anything to the output file.**

```
#include <stdio.h>

#define MAXNUMS 100


void sortfile(FILE *in, FILE *out)
{
    int a[MAXNUMS + 1], size, i;
    extern void sort(int *a, int size);

    /*
     * Read ints from "in", up to array limit or eof.
```

```
     * The assignment handout says it's ok to mistake non-numeric for EOF.
     * The array is one too big so that we can easily see if there are more
     * ints after.
     */
    for (size = 0;
            size < sizeof a / sizeof a[0] && fscanf(in, "%d", &a[size]) == 1;
            size++)
        ;
    /* if we hit array limit, i.e. have exceeded MAXNUMS, then crab & exit */
    if (size > MAXNUMS) {
        fprintf(stderr, "limit of %d values exceeded\n", MAXNUMS);
        exit(1);
    }

    /* sort and output */
    sort(a, size);
    for (i = 0; i < size; i++)
        fprintf(out, "%d\n", a[i]);
}


/* bubblesort, from lecture */
void sort(int *a, int size)
{
    int i, j, t;

    for (i = 0; i < size; i++) {
        for (j = 0; j < size - 1; j++) {
            if (a[j] > a[j+1]) {
                /* swap */
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
            }
        }
    }
}
```

---

5. Write a main() to prompt for and read the input file and output file names. Then open them both (with fopen(), calling perror() and exiting if the fopen() fails), and call the above sortfile(infp, outfp). Put this function in a new file called main.c. This main.c should be able to be compiled with your sortfile.c (i.e. "gcc -Wall main.c sortfile.c").

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

int main()
{
    char filename[100];
    FILE *in, *out;
    extern void sortfile(FILE *in, FILE *out);
    extern void myget(char *prompt, char *s, int size);

    /* get filenames and open files */
    myget("Input file: ", filename, sizeof filename);
```

```
    if ((in = fopen(filename, "r")) == NULL) {
        perror(filename);
        return 1;
    }
    myget("Output file: ", filename, sizeof filename);
    if ((out = fopen(filename, "w")) == NULL) {
        perror(filename);
        return 1;
    }

    /* call sortfile on 'em; files closed upon program exit */
    sortfile(in, out);

    return 0;
}


void myget(char *prompt, char *s, int size)
{
    char *p;

    printf("%s", prompt);
    if (fgets(s, size, stdin) == NULL)
        exit(0);
    if ((p = strchr(s, '\n')) != NULL)
        *p = '\0';
}
```

---

6. Write a new version of the above sortfile() which calls the qsort() library function instead of using the bubblesort algorithm. You can link it with either the previous tiny test main() or your main() from question 3 above.

qsort() is discussed in section 17.7 of the King book, and summarized on page 622. The on-line manual page "man qsort" is more comprehensive although equally terse.

Here is a suitable comparison function to pass to qsort() for an array of ints:

```
    int comparator(const void *p, const void *q)
    {
        return *(int *)p - *(int *)q;
    }


void sortfile(FILE *in, FILE *out)
{
    int a[MAXNUMS + 1], size, i;
    extern void sort(int *a, int size);

    /*
     * Read ints from "in", up to array limit or eof.
     * The assignment handout says it's ok to mistake non-numeric for EOF.
     * The array is one too big so that we can easily see if there are more
     * ints after.
     */
```

```
    for (size = 0;
            size < sizeof a / sizeof a[0] && fscanf(in, "%d", &a[size]) == 1;
            size++)
        ;
    /* if we hit array limit, i.e. have exceeded MAXNUMS, then crab & exit */
    if (size > MAXNUMS) {
        fprintf(stderr, "limit of %d values exceeded\n", MAXNUMS);
        exit(1);
    }

    /* sort and output */
    sort(a, size);
    for (i = 0; i < size; i++)
        fprintf(out, "%d\n", a[i]);
}
```

**OFF TAO'S WEBSITE:**

```
/* call qsort to do the sorting */
void sort(int *a, int size)
{
    extern int comparator(const void *p, const void *q);
    qsort((void *)a, size, sizeof(int), comparator);
}


int comparator(const void *p, const void *q)
{
    return *(int *)p - *(int *)q;
}
```

---

7. Write a new main() which expects the two file names to be specified on the unix command-line, instead of prompting for and inputting them.

```
int main(int argc, char **argv)
{
    FILE *in, *out;
    extern void sortfile(FILE *in, FILE *out);

    if (argc != 3) {
        fprintf(stderr, "usage: %s infile outfile\n", argv[0]);
        return 1;
    }

    /* open files */
    if ((in = fopen(argv[1], "r")) == NULL) {
        perror(argv[1]);
        return 1;
    }
    if ((out = fopen(argv[2], "w")) == NULL) {
        perror(argv[2]);
        return 1;
    }

    /* call sortfile on 'em; files closed upon program exit */
    sortfile(in, out);

    return 0;
```

# CSC 209 extra problems regarding software tools

1. Write a C program which acts as a standard unix tool (i.e. takes file name arguments or stdin if no args; outputs to stdout; etc) which turns all funny characters in its input to visible sequences as follows:

- characters less than ' ' (space) are represented as '^' and the character they're the control character of, e.g. the byte 1 is represented as "^A". 64 is the number to add; 'A' is 1 + 64.
- the character 127 is represented as "^?".
- characters greater than or equal to 128 are represented as "M-" (for "meta") and then the character minus 128. Note that one of the above two points might then apply

```
int main(int argc, char **argv)
{
   int i;
   FILE *fp;
   extern void process(FILE *fp);

   if (argc == 1) {
         process(stdin);
   } else {
         for (i = 1; i < argc; i++) {
            if (strcmp(argv[i], "-") == 0) {
                  process(stdin);
            } else if ((fp = fopen(argv[i], "r")) == NULL) {
                  perror(argv[i]);
            } else {
                  process(fp);
                  fclose(fp);
            }
         }
   }
   return(0);
}
```

```
void process(FILE *fp)
{
   int c;
   while ((c = getc(fp)) != EOF) {
         if (c >= 128) {
            printf("M-");
            c -= 128;
         }
         if (c == 127 || c < ' ' && c != '\n') {
            putchar('^');
            c = (c + 64) % 128;
         }
         putchar(c);
   }
}
```

---

2. Write a C program to calculate byte frequencies of the input, like this:

```
% echo hellooooooo | bfreq
012: 1
e: 1
h: 1
l: 2
o: 6
%
```

Use isprint() to make the decision as to whether to use %c or 0%o before the colon.

solution

2b. Write a command-line which sorts the occurrences into frequency order using sort. (Be sure that a frequency of 15 is sorted after a frequency of 7, despite occurring earlier in alphabetical order.)

solution `bfreq | sort -n +1`

---

3. Command-line arguments are file names. Stat them and decrease their mtime by one hour using utimes(). Example session:

```
% ls -l foo
-rw-r--r--    1 ajr      instrs       1759 Apr 14 11:29 foo
% back1 foo
% ls -l foo
-rw-r--r--    1 ajr      instrs       1759 Apr 14 10:29 foo
%
```

solution

```
int main(int argc, char **argv)
{
    struct stat statbuf;
    struct timeval times[2];
    int status = 0;

    if (argc < 2) {
        fprintf(stderr, "usage: %s file ...\n", argv[0]);
        return(1);
    }

    times[0].tv_usec = times[1].tv_usec = 0;

    while (--argc > 0) {
        argv++;
        if (stat(*argv, &statbuf)) {
            perror(*argv);
            status++;
        } else {
            times[0].tv_sec = times[1].tv_sec = statbuf.st_mtime - ONEHOUR;
            utimes(*argv, times);
        }
    }
return(status);
```

4. Write simple versions of any of the following unix tools. That is, implement a minimum of the command-line arguments or other special processing; mostly, just do the simple case, and you can impose arbitrary limits if necessary.

- cat [solution]
- echo (include -n option; get all spacing and newline exactly right) [solution without -n]
- ls (then add the -l option, although you will want to cheat on some fields)
- cmp
- cmp with a -r option for "recursive"
- tr
- fgrep (just grep for a fixed string)
- sort (just sort whole lines in strict lexical order)
- mkdir, including the -p option

## CAT

```
int main(int argc, char **argv)
{
   int i;
   FILE *fp;
   extern void process(FILE *fp);

   if (argc == 1) {
         process(stdin);
   } else {
         for (i = 1; i < argc; i++) {
            if (strcmp(argv[i], "-") == 0) {
                  process(stdin);
            } else if ((fp = fopen(argv[i], "r")) == NULL) {
                  perror(argv[i]);
                  return(1);
            } else {
                  process(fp);
                  fclose(fp);
            }
         }
   }
   return(0);
}

void process(FILE *fp)
{
   int c;
   while ((c = getc(fp)) != EOF)
         putchar(c);
```

# ECHO

```
#include <stdio.h>

int main(int argc, char **argv)
{
    for (argc--, argv++; argc > 0; argc--, argv++)
        printf("%s%c", *argv, (argc == 1) ? '\n' : ' ');
    return 0;
}
```

# LS

```
#include <sys/types.h>
#include <sys/dir.h>
#include <sys/param.h>
#include <stdio.h>
#include <stdlib.h>

#define FALSE 0
#define TRUE !FALSE

extern  int alphasort(); //Inbuilt sorting function

char pathname[MAXPATHLEN];

void die(char *msg)
{
  perror(msg);
  exit(0);
}

int file_select(struct direct *entry)
{
    if ((strcmp(entry->d_name, ".") == 0) || (strcmp(entry->d_name,
"..") == 0))
        return (FALSE);
    else
        return (TRUE);
}
```

```
int main()
{
    int count,i;
    struct direct **files;

    if(!getcwd(pathname, sizeof(pathname)))
        die("Error getting pathname\n");

    printf("Current Working Directory = %s\n",pathname);
    count = scandir(pathname, &files, file_select, alphasort);

    /* If no files found, make a non-selectable menu item */
    if(count <= 0)
      die("No files in this directory\n");

    printf("Number of files = %d\n",count);
    for (i=1; i<count+1; ++i)
        printf("%s  ",files[i-1]->d_name);
    printf("\n"); /* flush buffer */
}
```

## CMP exists

## LS

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

int main(int argc, char **argv)
{
    DIR *dp;
    struct dirent *p;
    int i, status = 0;

    for (i = 1; i < argc; i++) {
        if ((dp = opendir(argv[i])) == NULL) {
            perror(argv[i]);
            status++;
        } else {
            while ((p = readdir(dp)))
                printf("%s\n", p->d_name);
            closedir(dp);
        }
    }

    return(status);
}
```

## STRSORT

```
void
strsort(char *p)
{
        int i, flag;
        char t;

        if (strlen (p) < 2)
                return;
        for (;;) {
                flag = 0;
                for (i = 0; p[i + 1]; i++) {
                        if (p[i] > p[i + 1]) {
                                t = p[i + 1];
                                p[i + 1] = p[i];
                                p[i] = t;
                                flag = 1;
                        }
                }
                if (flag == 0)
                        break;
        }
}
```

## FGREP (this is C++ but Idk... lol)

```
#include<stdio.h>
#include<dirent.h>
int main()
{
    char fn[10], pat[10], temp[200];
    FILE *fp;
    printf("\n Enter file name : ");
    scanf("%s", fn);
    printf("Enter the pattern: ");
    scanf("%s", pat);
    fp = fopen(fn, "r");
    while (!feof(fp))
    {
        fgets(temp, sizeof(fp), fp);
        if (strcmp(temp, pat))
            printf("%s", temp);
    }
    fclose(fp);
    return 1;
}
```

# GREP

```
void match_pattern(char *argv[])
{
    int fd,r,j=0;
    char temp,line[100];
    if((fd=open(argv[2],O_RDONLY)) != -1)
    {
        while((r=read(fd,&temp,sizeof(char)))!= 0)
        {
            if(temp!='\n')
            {
                line[j]=temp;
                j++;
            }
            else
            {
                if(strstr(line,argv[1])!=NULL)
                    printf("%s\n",line);
                memset(line,0,sizeof(line));
                j=0;
            }

        }
    }
}
main(int argc,char *argv[])
{
    struct stat stt;
    if(argc==3)
    {
        if(stat(argv[2],&stt)==0)
            match_pattern(argv);
        else
        {
            perror("stat()");
            exit(1);
        }   } }
```

# MKDIR

```
21.     static void _mkdir(const char *dir) {
22.             char tmp[256];
23.             char *p = NULL;
24.             size_t len;
25.
26.             snprintf(tmp, sizeof(tmp),"%s",dir);
27.             len = strlen(tmp);
28.             if(tmp[len - 1] == '/')
29.                     tmp[len - 1] = 0;
30.             for(p = tmp + 1; *p; p++)
31.                     if(*p == '/') {
32.                             *p = 0;
33.                             mkdir(tmp, S_IRWXU);
34.                             *p = '/';
35.                     }
```

```
36.             mkdir(tmp, S_IRWXU);
```

# Part A: Problems not requiring C

# Part A: Problems not requiring C

---

2. Sketch the inode and directory information involved in the following output from an "ls -liR" of the directory which is inode number 10. ('R' is "recursive", and 'i' is "list inode numbers" (which appear in the left column, i.e. the numbers which are 12 through 15 below)).

That is, list the inode numbers and what data you know is in those inodes; and state the contents of the directories involved.

```
total 8
12 drwxr-xr-x  3 ajr  users 4096 Feb 27 10:05 bar
13 drwxr-xr-x  3 ajr  users 4096 Feb 27 10:05 baz

./bar:
total 8
14 -rw-r--r--  1 ajr  users   20 Feb 27 10:06 gar

./baz:
total 8
15 -rw-r--r--  1 ajr  users   50 Feb 27 10:06 gaz
```

Solution:

Inodes:

| | | |
|---|---|---|
| 10 | D | ??? |
| 12 | D | rwxr-xr-x, ajr, users, Feb 27, etc |
| 13 | D | rwxr-xr-x, ajr, users, Feb 27, etc |
| 14 | | rw-r--r--, ajr, users, Feb 27, etc |
| 15 | | rw-r--r--, ajr, users, Feb 27, etc |

Contents:

| | |
|---|---|
| 10 | . 10, .. ??, bar 12, baz 13 |
| 12 | . 12, .. 10, gar 14 |
| 13 | . 13, .. 10, gaz 15 |
| 14 | *contents of gar file* |
| 15 | *contents of gaz file* |

## #3

a) Indicate the link count for each of these files:

Solution:
handout: 2 *(the link in the a2 directory, and '.' in the handout directory)*
starter: 3 *(the link in the a2 directory, '.' in the starter directory, and '..' in splitexpr)*
soln: 2 *(the link in the a2 directory, and '.' in the soln directory)*
wfold.c: 1

b) Suppose the user is cd'd to the handout directory and types:

```
ln a2.pdf example.pdf
```

Which file(s) will change their link count, and what will the link count(s) be now?

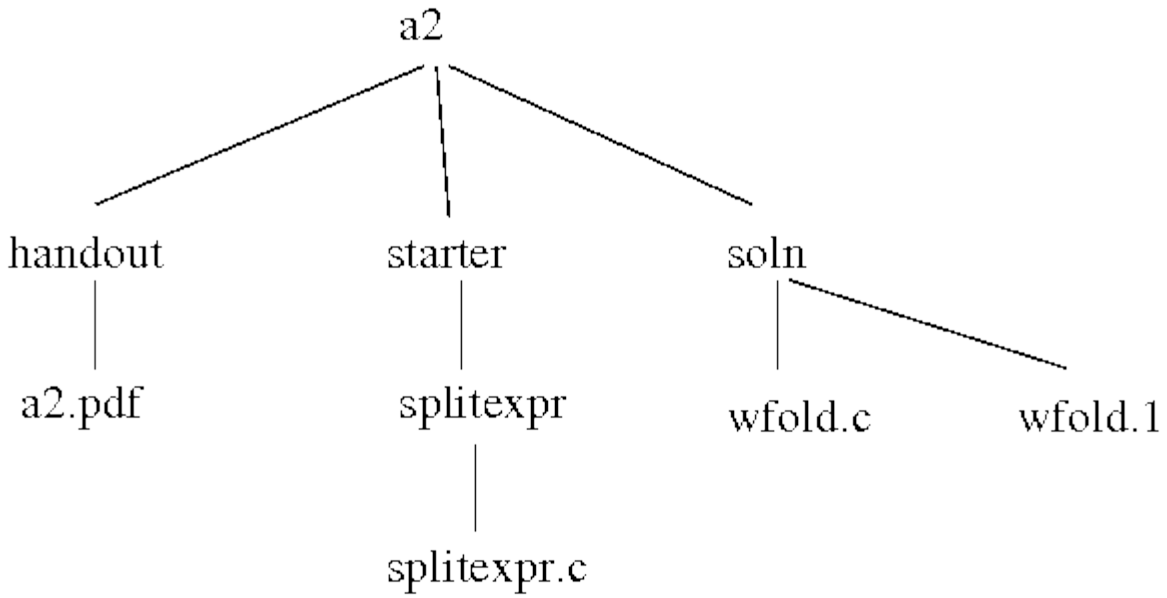
Solution: a2.pdf's link count will change from 1 to 2
*(because there are two links to it; no changes are involved in links to directories)*

c) Suppose the user is cd'd to the soln directory and types:

```
mkdir new
```

Which file(s) will change their link count, and what will the link count(s) be now?

Solution: soln's link count will change to 3
*(because it is now also linked in as ".." in the new directory)*



2. Using opendir() and friends and running stat() on each file in the directory, write a program which displays the size in bytes of the largest file in a directory specified on the command-line (i.e. in argv).

[solution]

```
int main(int argc, char **argv)
{
    DIR *dp;
    struct dirent *p;
    int i, status = 0, len;
    long max;
    char buf[200];
    struct stat statbuf;

    for (i = 1; i < argc; i++) {
        if ((len = strlen(argv[i])) + 2 > sizeof buf) {
            fprintf(stderr, "%s: directory name too long\n", argv[i]);
            status++;
        } else if ((dp = opendir(argv[i])) == NULL) {
            perror(argv[i]);
            status++;
        } else {
            sprintf(buf, "%s/", argv[i]);
            len++;  /* now buf[len] is the first byte after that slash */
            max = 0;
            while ((p = readdir(dp))) {
                if (strlen(p->d_name) + len + 1 > sizeof buf) {
                    buf[len] = '\0';
                    fprintf(stderr, "%s%s: filename too long\n", buf,
                            p->d_name);
                    /*
                     * use status=1 instead of status++ in case we have a lot
                     * of these
                     */
                    status = 1;
                } else {
                    strcpy(buf + len, p->d_name);  /* a concatenation */
                    if (stat(buf, &statbuf)) {
                        perror(buf);
                        status++;
                    } else if (statbuf.st_size > max) {
                        max = statbuf.st_size;
                    }
                }
            }
            closedir(dp);
            printf("%s: %ld\n", argv[i], max);
        }
    }

    return(status);
}
```

**3. Write a program which, for each directory specified on the command line, *recursively* finds the *total* byte size of the files in that directory *and* all subdirectories.**

```
int main(int argc, char **argv)
{
    int i;
    extern long totalbytes(char *);

    for (i = 1; i < argc; i++)
        printf("%s: %ld\n", argv[i], totalbytes(argv[i]));

    return(status);
}

long totalbytes(char *dirname)
{
    DIR *dp;
    struct dirent *p;
    long retval = 0;
    char buf[200];
    int len;
    struct stat statbuf;

    if ((len = strlen(dirname)) + 2 > sizeof buf) {
        fprintf(stderr, "%s: directory name too long\n", dirname);
        status++;
        return(0);
    }

    if ((dp = opendir(dirname)) == NULL) {
        perror(dirname);
        status++;
        return(0);
    }

    sprintf(buf, "%s/", dirname);
    len++;  /* now buf[len] is the first byte after that slash */
    while ((p = readdir(dp))) {
        if (strlen(p->d_name) + len + 1 > sizeof buf) {
            buf[len] = '\0';
            fprintf(stderr, "%s%s: filename too long\n", buf, p->d_name);
            /*
             * use status=1 instead of status++ in case we have a lot of these
             */
            status = 1;
        } else if (strcmp(p->d_name, ".") && strcmp(p->d_name, "..")) {
            strcpy(buf + len, p->d_name);  /* a concatenation */
            if (stat(buf, &statbuf)) {
                perror(buf);
                status++;
            } else if (S_ISDIR(statbuf.st_mode)) {
                retval += totalbytes(buf);
            } else {
                retval += statbuf.st_size;
            }
        }
    }
    closedir(dp);
    return(retval);}
```

# CSC 209 extra problems regarding processes

---

1. Write a program which creates five child processes, numbered 0 through 4. Each child process computes the sum of the integers 5i through 5i+4, where i is its index number. It then returns this value as the process exit status. The parent waits for them all and outputs the total of the exit statuses, which will be the sum of the integers from 0 to 24 inclusive

```
int main()
{
    int sum, i, status;
    extern void doit(int i);

    for (i = 0; i < 5; i++)
        if (fork() == 0)
            doit(i);
    for (sum = i = 0; i < 5; i++) {
        wait(&status);
        sum += (status >> 8);
    }
    printf("%d\n", sum);
    return(0);
}


void doit(int i)
{
    int j, sum = 0;
    i *= 5;
    for (j = 0; j < 5; j++)
        sum += i + j;
    _exit(sum);
}
```

---

2. There is a network server program in /usr/local/sbin/myserver. It is buggy and frequently crashes. Fortunately, when it gets into trouble it tends to give a segmentation exception and terminate, so it is easy to tell when it has crashed.

Write a complete C program which invokes /usr/local/sbin/myserver, and restarts it when it exits. After /usr/local/sbin/myserver exits, you should sleep for three seconds and then execute it again, and loop.

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

int main()
{
    int status;
    while (1) {
        switch (fork()) {
        case -1:
            perror("fork");
            exit(1);
        case 0:
            execl("/usr/local/sbin/myserver",
"/usr/local/sbin/myserver", (char *)NULL);
            perror("/usr/local/sbin/myserver");
            exit(1);
        default:
            wait(&status);
        }
        sleep(3);
    }
}
```

---

3. Write a very trivial shell: print a prompt; call fgets() to read a line and remove any terminating newline; execute that whole line as a program; and loop around until eof. The input line must be a complete path name for the program you're trying to run (e.g. /bin/cat rather than cat) and it must be only the program name (e.g. if you type "/bin/cat file" it will try to run a program named "cat file" in the /bin directory).

**MARK**

---

4. Someone has removed the 'ps' command for alleged security reasons. Write a program to list all of the processes you own, assuming that the current process's pid is larger than all previously-existing pids. The algorithm is to loop from 1 to getpid(), doing a kill with "signal" 0 on each process id number. This just checks for your ability to signal the process without actually doing it, thus checking existence and ownership.

```
int main()
{
    int i, max = getpid();
    for (i = 1; i < max; i++)
        if (kill(i, 0) == 0)
            printf("%d\n", i);
    return(0);
}
```

---

6. Write a program which runs an arbitrary program with a time limit of 10 seconds. The program to run is specified on the command-line; pass argv+1 to execve or similar. (Its first element will be a path name; you do not need to use a search path.) You will fork, but the parent will not do a wait(), but rather, a sleep and then possibly a kill(pid, 9). If you do terminate the program in this way, print a suitable message to stderr.

```
void sigchld(int x)
{
    _exit(0);
}


int main(int argc, char **argv)
{
    int pid;

    if (argc < 2) {
        fprintf(stderr, "usage: limiter command\n");
        return(1);
    }
    if ((pid = fork()) == -1) {
        perror("fork");
        return(1);
    }
    if (pid) {
        signal(SIGCHLD, sigchld);
        sleep(10);
        /* won't get here if SIGCHLD happened */
        signal(SIGCHLD, SIG_DFL);
        if (kill(pid, 9) == 0 || errno != ESRCH /* No such process */)
            fprintf(stderr, "Time limit exceeded.  Probably in an infinite
loop.\n");
    } else {
        argv++;
        if (execv(argv[0], argv))
            perror(argv[0]);
    }

    return(0);
}
```

---

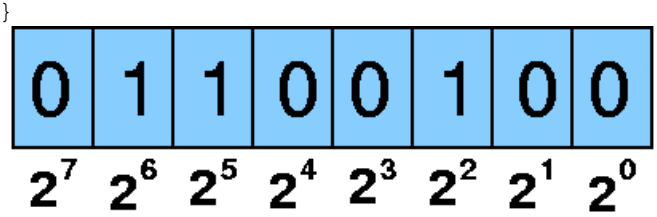
7. Write a program which runs an arbitrary program with an output limit (rather than the time limit in question #5). Use a pipe and a fork to read the stdout *and* stderr from the subprocess. (It's a better exercise if you refrain from using popen(), although in real life we would prefer to use popen() if it did what we needed.)

```
int main(int argc, char **argv)
{
    FILE *fp;
    int line = 0, col = 0, c;

    if (argc != 2) {
        fprintf(stderr, "usage: outputlimiter command\n");
        fprintf(stderr, "       The entire command should be in argv[1] -- it
is passed to popen().\n");
        return(1);
    }

    /*
     * merge stderr and stdout to stdout.  This isn't really for my own
     * benefit, it's to be inherited by the child process.
     */
    (void)dup2(1, 2);

    if ((fp = popen(argv[1], "r")) == NULL) {
        perror(argv[1]);
        return(1);
    }
    while ((c = getc(fp)) != EOF) {
        putchar(c);
        if (c == '\n' || ++col == 80) {
            line++;
            col = 0;
        }
        if (line > 60) {
            if (col)
                putchar('\n');
            printf("\nFurther output lines discarded.\n");
            exit(1);
        }
    }

    if (col != 0)
        putchar('\n');

    return(0);
}
```

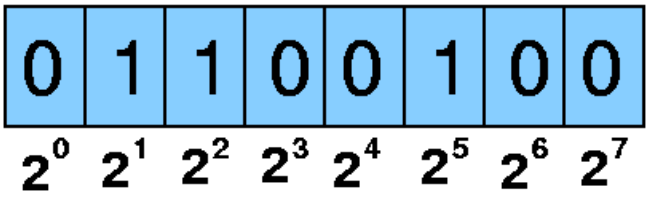
# CSC 209 extra problems regarding interprocess communication

## TELLS YOU if you are eindian or not

```
int main()
{
    int i = 38;

    printf("Approach 1: ");
    if (i == ntohs(i))
        printf("big-endian\n");
    else
        printf("little-endian\n");

    printf("Approach 2: ");
    if (((char *)&i)[0] == 0)
        printf("big-endian\n");
    else
        printf("little-endian\n");

    printf("Approach 3: ");
    i = ('1' << 24) | ('2' << 16) | ('3' << 8) | '4';
    printf("Byte order is %.4s\n", (char *)&i);

    return(0);
}
```




**Big Endian**
**= 0x64 = 100**




**Little Endian**
**= 0x26 = 38**