Andrew login ID:_____

Full Name:_____

CS 15-123, Spring 2010

Sample Exam 3

Mon. April 6, 2009

Instructions:

- Make sure that your exam is not missing any sheets, then write your full name and Andrew login ID on the front.
- Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.
- The exam has a maximum score of 100 points
- The problems are of varying difficulty. The point value of each problem is indicated. Pile up the easy points quickly and then come back to the harder problems.
- The exam is OPEN BOOK/NOTES. You may use any books or notes you like. No calculators or other electric devices are allowed.

Problem 1:

a) When should you use a Macro vs. a Function?

b) What is the purpose of #ifndef / #define / #endif in a header file is. Provide a simple example if it helps your explanation.

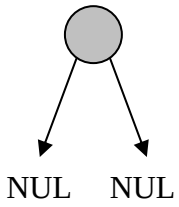
Problem 2:

Design and implement a C function that counts the number of leaves in a binary tree when passed the root of the tree.

Include the function prototype and the definition of the function.

```
typedef struct tree_node
{
        int data;
        struct tree_node * left;
        struct tree_node * right;
} TreeNode;
```

Leaf:



Problem 3:

For each program listing below, mark the potential output as correct (☑) or incorrect (**X**).

Question 3.a:

```
void main ()
{
    int i = 2;

    printf("i = %d\n", i);

    if (fork() == 0)
    {
     i++;
     printf("i = %d\n",i);

     printf("i = %d\n",i);
    }

    printf("i = %d\n",i);
}
```

i = 2 i = 3 i = 3 i = 3 i = 2	i = 2 i = 2 i = 3 i = 3 i = 3	i = 2 i = 3 i = 2 i = 3 i = 2

Question 3.b:

```
void main ()
{
    int i = 2;

    printf("i = %d\n", i);

    if (fork() == 0)
    {
     i++;
     printf("i = %d\n",i);

     execvp("./prog1", NULL);
     i++;
     printf("i = %d\n",i);
    }

    printf("i = %d\n",i);
}
```

Program prog1:

```
void main ()
{
    int i;

    i++;
}
```

i = 2 i = 2 i = 3 i = 4 i = 4	i = 2 i = 3 i = 5 i = 5 i = 2	i = 2 i = 2 i = 3	i = 2 i = 3 i = 4 i = 4 i = 2	i = 2 i = 3 i = 2

Question 3.c:

```
void main ()
{
    int i = 2;

    printf("i = %d\n", i);

    if (fork() == 0)
    {
     i++;
     printf("i = %d\n",i);

     if (fork() != 0)
     {
         i++;
         printf("i = %d\n",i);
         fork();
     }
     printf("i = %d\n",i);
    }
    printf("i = %d\n",i);
}
```

i = 2	i = 2	i = 2	i = 2
i = 3	i = 3	i = 3	i = 3
i = 3	i = 3	i = 2	i = 2
i = 3	i = 3	i = 4	i = 3
i = 4	i = 4	i = 4	i = 3
i = 4	i = 4	i = 4	i = 4
i = 4	i = 4	i = 4	i = 4
i = 4	i = 2	i = 4	i = 4
i = 4	i = 4	i = 3	i = 4
i = 2	i = 4	i = 3	i = 4

Problem 4:

Write a Perl program that reads in two files and writes to a third. All these files names are passed as command line arguments. The first file contains pairs of strings for which occurrences of the first string should be replaced by the second string. The second file is the file for which this substitution should be performed.

e.g. Given:

replacements.txt (The first file):

Tessa => Alex
Java => Perl
Windows => Linux

file.txt (The Second file):

Tessa loves programming on Windows.
Tessa's favorite language is Java.
Alex prefers ML.

If you run: './your_program.pl replacements.txt file.txt output.txt', it should produce an output.txt file containing:

Alex loves programming on Linux.
Alex's favorite language is Perl.
Alex prefers ML.

Problem 5:

When you use #define, what happens in your source code when the #define is expanded by the pre-processor? Provide a simple example if it helps your explanation...[3 points]

Suppose you are implementing a simple calculator that will be called from another program (like lab3.c). Which of the following belong(s) in the calculator.h file (the calculator interface) and which belong(s) in the corresponding calculator.c file? Just put an H or a C next to each component: the #define, the typedef, EACH function prototype, and EACH function definition...[9 points]

```
#define PI 3.14159

typedef struct {
    int operand1, operand2;
    double result;
} Data;

void add(Data d);
void subtract(Data d);
void multiply(Data d);
void divide(Data d);
void fatalError(char *msg);

void add(Data d)
{
    d.result = d.operand1 + d.operand2;
}

void subtract(Data d)
{
    d.result = d.operand1 - d.operand2;
}

void multiply(Data d)
{
    d.result = d.operand1 * d.operand2;
}

void divide(Data d)
{
    d.result = (double)d.operand1 / d.operand2;
}

void fatalError(char *msg)
{
    fprintf(stderr,"\nFATAL ERROR: %s!!\n\n", msg);
    exit(EXIT_FAILURE);
}
```

Problem 6 (20 points):

In Lab 6 (myls), you wrote the -s option that printed out the files and directories in ascending-order.

Write a function that **prints out ONLY files (DON'T PRINT THE DIRECTORIES) in the current directory in descending-order (by size of the file). YOU MUST USE QSORT.**

There is some information on system functions you might want to use on the next page. You don't need to use them if you think they are not necessary.

NAME
opendir - open a directory

SYNOPSIS
#include <sys/types.h>
#include <dirent.h>

DIR *opendir(const char *name);

DESCRIPTION
The opendir() function opens a directory stream corresponding to the directory name, and returns a pointer to the directory stream. The stream is positioned at the first entry in the directory.

RETURN VALUE
The opendir() function returns a pointer to the directory stream or NULL if an error occurred.

NAME
readdir - read a directory

SYNOPSIS
#include <sys/types.h>
#include <dirent.h>

struct dirent *readdir(DIR *dir);

DESCRIPTION
The readdir() function returns a pointer to a dirent structure representing the next directory entry in the directory stream pointed to by dir. It returns NULL on reaching the end-of-file or if an error occurred.

According to POSIX, the dirent structure contains a field char d_name[] of unspecified size, with at most NAME_MAX characters preceding the terminating null character. Use of other fields will harm the portability of your programs. POSIX 1003.1-2001 also documents the field ino_t d_ino as an XSI extension.

The data returned by readdir() may be overwritten by subsequent calls to readdir() for the same directory stream.

RETURN VALUE
The readdir() function returns a pointer to a dirent structure, or NULL if an error occurs or end-of-file is reached.

The dirent structure is declared as follows:
struct dirent
{
long d_ino; /* inode number */
off_t d_off; /* offset to this dirent */
unsigned short d_reclen; /* length of this d_name */
char d_name [NAME_MAX+1]; /* file name (null-terminated) */
}
NAME
getcwd - Get current working directory

SYNOPSIS
#include <unistd.h>

char *getcwd(char *buf, size_t size);

DESCRIPTION

The getcwd() function copies an absolute pathname of the current working directory to the array pointed to by buf, which is of length size.

If the current absolute path name would require a buffer longer than size elements, NULL is returned, and errno is set to ERANGE; an application should check for this error, and allocate a larger buffer if necessary.

If buf is NULL, the behaviour of getcwd() is undefined.

NAME
stat - get file status

SYNOPSIS
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int stat(const char *file_name, struct stat *buf);
DESCRIPTION
These functions return information about the specified file. You do not need any access rights to the file to get this information but you need search rights to all directories named in the path leading to the file.

stat stats the file pointed to by file_name and fills in buf.

stat returns a stat structure, which contains the following fields:

```
struct stat {
dev_t st_dev; /* device */
ino_t st_ino; /* inode */
mode_t st_mode; /* protection */
nlink_t st_nlink; /* number of hard links */
uid_t st_uid; /* user ID of owner */
gid_t st_gid; /* group ID of owner */
dev_t st_rdev; /* device type (if inode device) */
off_t st_size; /* total size, in bytes */
blksize_t st_blksize; /* blocksize for filesystem I/O */
blkcnt_t st_blocks; /* number of blocks allocated */
time_t st_atime; /* time of last access */
time_t st_mtime; /* time of last modification */
time_t st_ctime; /* time of last status change */
};
```

The value st_size gives the size of the file (if it is a regular file or a symlink) in bytes. The size of a symlink is the length of the pathname it contains, without trailing NUL.

The value st_blocks gives the size of the file in 512-byte blocks. (This may be smaller than st_size/512 e.g. when the file has holes.) The value st_blksize gives the "preferred" blocksize for efficient file system I/O. (Writing to a file in smaller chunks may cause an inefficient read-modify-rewrite.)

Not all of the Linux filesystems implement all of the time fields. Some file system types allow mounting in such a way that file accesses do not cause an update of the st_atime field. (See 'noatime' in mount(8).)

The field st_atime is changed by file accesses, e.g. by execve(2), mknod(2), pipe(2), utime(2) and read(2) (of more than zero bytes). Other routines, like mmap(2), may or may not update st_atime.

The field st_mtime is changed by file modifications, e.g. by mknod(2), truncate(2), utime(2) and write(2) (of more than zero bytes). Moreover,

st_mtime of a directory is changed by the creation or deletion of files in that directory. The st_mtime field is not changed for changes in owner, group, hard link count, or mode.

The field st_ctime is changed by writing or by setting inode information (i.e., owner, group, link count, mode, etc.).

The following POSIX macros are defined to check the file type:
S_ISREG(m) is it a regular file?
S_ISDIR(m) directory?
S_ISCHR(m) character device?
S_ISBLK(m) block device?
S_ISFIFO(m) fifo?
S_ISLNK(m) symbolic link? (Not in POSIX.1-1996.)
S_ISSOCK(m) socket? (Not in POSIX.1-1996.)

NAME
chdir - change working directory

SYNOPSIS
#include <unistd.h>

int chdir(const char *path);

DESCRIPTION
chdir changes the current directory to that specified in path.

RETURN VALUE
On success, zero is returned. On error, -1 is returned, and errno is set appropriately.

NAME
qsort - sort functions

SYNOPSIS
#include <stdlib.h>

void
qsort(*void *base*, *size_t nel*, *size_t width*,
int (**compar*)(*const void *, const void **));

DESCRIPTION
The **qsort**() function is a modified partition-exchange sort, or quicksort.
The **qsort**()function sort an array of *nel* objects, the initial member of which is pointed to by *base*. The size of each object is specified by *width*.
The contents of the array *base* are sorted in ascending order according to a comparison function pointed to by *compar*, which requires two arguments pointing to the objects being compared.

The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

RETURN VALUE
The **qsort**()function return no value.