Duration: **50 minutes**
Aids Allowed: **1 - 8.5x11 sheet**

Student Number: |_|_|_|_|_|_|_|_|_|_|_|

Last Name: SOLUTION

First Name: _____

Instructor: Karen Reid

---

*Do **not** turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above, and read the instructions below *carefully*.)

This midterm test consists of 5 questions on 9 pages (including this one). *When you receive the signal to start, please make sure that your copy of the test is complete.* Extra space was left for each of the programming questions. Please indicate clearly the part of your work that should be marked.

IMPORTANT: Assume all #includes have been done for you. You do not need to perform error checking unless the question specifically asks you to.

MARKING GUIDE

1: _____/ 7
2: _____/ 4
3: _____/ 2
4: _____/ 8
5: _____/ 7

TOTAL: _____/28

Good Luck!

Question 1. [7 MARKS]

Consider the following C program.

```
int main() {

    char **strs;
    char a[20] = "bobsled";
    char *b = "rock";
    char *c[4];
    char *d = malloc(20);

    strncpy(d, "curling", ____________________);

    //Fill in the argument to malloc with
    //the correct type
    strs = malloc(4 * sizeof(____________________));
    strs[0] = a;
    strs[1] = b;
    strs[2] = d;
    strs[3] = a + 3;
    strs[4] = NULL;

    strncat(*strs, "2, 4",

            ____________________________);
    strs[1] = &strs[2][strlen(strs[2])];



    strncpy(strs[1], b,_________________________);
    b[1] = 'o';
    c[0] = a;

    int i;
    for(i = 0; i < 4; i++) {
        printf("%s\n", strs[i]);
    }


    return 0;
}
```

Part (a) [2 MARKS]

For each of the following variables, is the value of the variable stored on the stack or the heap?

`strs` – stack

`a[3]` – stack

`d` – stack

`d[2]` – heap

Part (b) [2 MARKS]

Fill in the blank arguments in the code with the correct values, according to the intended purpose of the parameters.

Part (c) [1 MARK]

Comment out any lines that might or will lead to a segmentation fault, and explain briefly why they cause an error.

Part (d) [2 MARKS]

Give the output of the program assuming the line or lines with errors in them are commented out.

Solutions to Q1

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

```
int main() {

    char **strs;
    char a[20] = "bobsled";
    char *b = "rock";
    char *c[4];
    char *d = malloc(20);
    strncpy(d, "curling", 20);

    strs = malloc(4 * sizeof(char *));
    strs[0] = a;
    strs[1] = b;
    strs[2] = d;
    strs[3] = a + 3;
    strs[4] = &a[4];

    strncat(*strs, "2, 4", 20 - strlen(*strs));
    strs[1] = &strs[2][strlen(strs[2])];
    strncpy(strs[1], b, strlen(strs[2]));
    //b[1] = 'o'; //Cannot modify a string literal
    c[0] = a;

    int i;
    for(i = 0; i < 4; i++) {
        printf("%s\n", strs[i]);
    }
    return 0;
}
```

```
Output:
bobsled2, 4
rock
curlingrock
sled2, 4
```

Question 2. [4 MARKS]

My current working directory contains the following files:

```
Makefile        input        runtests.c
```

The Makefile has the following contents.

```
all : runtests results

runtests : runtests.o
        gcc -Wall -g -o runtests runtests.o
runtests.o : runtests.o
        gcc -Wall -g -c runtests.c
results : runtests input
        runtests input > results
```

Recall that `touch` creates a file if it doesn't exist and updates the last modified time on a file if it does exist.

Part (a) [2 MARKS]

List the files that are created when I run `make` the first time.

runtests.o runtests results

Part (b) [2 MARKS]

If I then modify `input` and run `make results`, which actions (commands) in the Makefile are executed?

runtests input ¿ results

Question 3. [2 MARKS]

Consider the following C file:

```
#include <stdio.h>

    int g(int y);
    int num;
    void f(int x) {
        num = x * x;
        printf("%d %d\n", num, g(x));
    }

    int main(){
        int i = 42;
        f(i);
        return 0;
    }
```

Part (a) [1 MARK]

Give 2 symbols in this code that are imported but not exported. *g, printf*

Part (b) [1 MARK]

Give 2 symbols in this code that are exported. *main, f, num,*

Question 4. [8 MARKS]

Complete the C function below that inserts the string `src` into `dest` at the beginning of `dest`. The `size` argument gives the number of bytes left in `dest` that may be occupied by `src`. If the length of `src` is greater than `size`, only the first `size` bytes of `src` will be inserted into `dest`.

You may not use any library functions other than `strlen`.

Below is an example of calling `prepend`:

```
char a[30] = "Games";
char b[30] = "Olympic";
prepend(a, b, 30 - strlen(a) - 1);
printf("%s", a);  // prints "OlympicGames"

void prepend(char *dest, char *src, int size) {
```

```
char *prepend(char *str1, char *str2, int size) {
    int max, i, j;
    // Figure out how many characters from str2 to copy
    if(strlen(str2) + strlen(str1) < size) {
        max = strlen(str2) + strlen(str1);
    } else {
        max = size;
    }

// Shift over characters from str1
    for(j = 0, i = strlen(str2); i < max; i++, j++){
        str1[i] = str1[j];
    }
    str1[i] = '\0';

    for(i = 0; i < strlen(str2); i++) {
        str1[i] = str2[i];
    }
    return str1;
}
```

Question 5. [7 MARKS]

Part (a) [1 MARK]

Rewrite the following shell statement so that fewer processes are created:

```
cat myfile | cut -f 1 -d " "
```

Part (b) [6 MARKS]

Write a Bourne shell program `pathinfo` that takes an absolute or relative path as an argument and prints out information about about each directory in the path: the path to that directory, the number of subdirectories, and the sum of the size in kilobytes of the regular files (not directories) in that directory.

You may find some of the following shell commands useful:

- `basename NAME` Print `NAME` with any leading directory components removed.
- `dirname NAME`: Print `NAME` with its trailing /component removed; if `NAME` contains no /'s, output '.'
- `du FILE`: Print the number of kilobytes used by `FILE`, and `FILE` separated by a tab. (E.g. `du file1` prints `4 file1`)

Recall that `basename NAME` prints the `NAME` with any leading directory components removed.

Following are two examples of running the `pathinfo`. (E.g., There are 0 subdirectories of a1a, and the size of the regular files in `a1a` totals 317 kb. There are 9 subdirectories in `pub` and the size of the files in pub totals 1 kb.)

```
$ pathinfo /u/csc209h/winter/pub/a1a
/u/csc209h/winter/pub/a1a 0 317
/u/csc209h/winter/pub 9 1
/u/csc209h/winter 4 0
/u/csc209h 6 0
/u 381 8
$ ~/209/tmp/pathinfo ./a1a
./a1a 0 317
```

```
#!/bin/sh

path=$1

if [ ! -f $path -a ! -d $path ]
then
        echo "Error $path is not a valid path"
        exit 1
fi

while [ "$path" != "/" -a "$path" != "." ]
do
        count=0
```

```
        size=0
        for f in $path/*
        do
                if [ -d "$f" ]
                then
                        count=‘expr $count + 1‘
                elif [ -f "$f" ]
                then
                        fsize=‘du $f | cut -f 1 ‘
                        size=‘expr $size + $fsize‘
                fi
        done
        echo $path $count $size
        path=‘dirname $path‘
done
```

(This page intentionally left blank)

C function prototypes and structs:

```
int sprintf(char *s, const char *format, ...)
char *strchr(const char *s, int c)
size_t strlen(const char *s)
char *strncat(char *dest, const char *src, size_t n)
int strncmp(const char *s1, const char *s2, size_t n)
char *strncpy(char *dest, const char *src, size_t n)
char *strrchr(const char *s, int c)
char *strstr(const char *haystack, const char *needle)
```

Shell comparison operators

Shell	Description
-d filename	Exists as a directory
-f filename	Exists as a regular file.
-r filename	Exists as a readable file
-w filename	Exists as a writable file.
-x filename	Exists as an executable file.
-z string	True if empty string
str1 = str2	True if str1 equals str2
str1 != str2	True if str1 not equal to str2
int1 -eq int2	True if int1 equals int2
-ne, -gt, -lt, -le	For numbers
!=, >, >=, <, <=	For strings
-a, -o	And, or.

Useful shell commands:

du, echo, ls, head, tail, read, sort, uniq, wc
grep (returns 0 if match is found, 1 if no match was found, and 2 if there was an error)
set (Note: with no arguments set prints the list of environment variables) ps aux - prints the list of currently running processes grep -v displays lines that do not match