CSE 374 Midterm Exam

April 28, 2017

Name ____________**Answer Key**_______________________ Id # _______________

There are 6 questions worth a total of 112 points. Please budget your time so you get to all of the questions. Keep your answers brief and to the point.

The exam is closed book, closed notes, closed electronics, closed telepathy, etc.

Many of the questions have short solutions, even if the question is somewhat long. Don't be alarmed.

If you don't remember the exact syntax of some command or the format of a command's output, make the best attempt you can. We will make allowances when grading.

Relax, you are here to learn.
Please wait to turn the page until everyone is told to begin.

Score __________________ / 122

1. ______ / 10
2. ______ / 35
3. ______ / 25
4. ______ / 20
5. ______ / 20
6. ______ / 12

Reference Information

Some of this information might be useful while answering questions on the exam. Feel free to remove this page for reference while you work.

Shell tests

Some of the tests that can appear in a [ ] test command in a bash script:

- string comparisons: =, !=
- numeric comparisons: -eq, -ne, -gt, -ge, -lt, -le
- -d *name* test for directory
- -f *name* test for regular file
- *fname1* -nt *fname2* to test if the first file is newer than the second
- Shell variables: $# (# arguments), $? (last command result), $@, $* (all arguments), $0, $1, ... (specific arguments), shift (discard first argument)

Strings and characters (<string.h>, <ctype.h>)

Some of the string library functions:

- `char*strncpy(`*`dest,src,n`*`)`,copies exactly *n* characters from *src* to *dst*, adding ‘\0’s at end if fewer than *n* characters in *src* so that *n* chars. are copied.
- `char*strcpy(`*`dest,src`*`)`
- `char* strncat(`*`dest, src, n`*`)`, append up to *n* characters from *src* to the end of *dest*, put ‘\0’ at end, either copy from *src* or added if no ‘\0’ in copied part of *src*.
- `char*strcat(`*`dest,src`*`)`
- `int strncmp(`*`string1, string2, n`*`)`, <0, =0, >0 if compare <, =, >
- `int strcmp(`*`string1, string2`*`)`
- `char* strstr(`*`string, search_string`*`)`
- `int strnlen(`*`s, max_length`*`)`
- `int strlen(`*`s`*`)`
- Character tests: `isupper(`*`c`*`), islower(`*`c`*`), isdigit(`*`c`*`), isspace(`*`c`*`)`
- Character conversions: `toupper(`*`c`*`), tolower(`*`c`*`)`

1) True or False (2 Points each)
Circle T or F

a) / **F**: Newlines (also called line breaks) are represented in the exact same way on every computer.

b) / **F**: The null byte and the null value are the same

c) / **F**: gcc is the only C compiler

d) **T** / : In C, it's never OK to read from a variable before you initialize it

e) / **F**: The output of the `script` command is an executable shell script

2) Short Answer

a) (5 points) Write the type of `argv` (there are multiple correct answers, just give one)

char** argv or char* argv[]

b) (9 points) Suppose you've just written a C program, compiled it, and run it. Immediately the program exits due to a segmentation fault (SEGFAULT). Assuming you've already compiled with the correct flags, list the 3 concrete commands which, if executed in order, would show you the number of the line in your program which triggers the segfault. You may assume the binary is called `prog`, and it takes 2 arguments `arg1` and `arg2`.

1) gdb ./prog

2) run prog arg1 arg2

3) bt (not technically necessary)

c) (3 points) Why is executing the line: `PATH="Hello, world"` a bad idea?

You will no longer be able to run most programs, as PATH stores where the shell looks to find executable programs

d) (18 points) Assuming the file `/usr/dict/words` contains all the words in the English language, write a single bash command (which could be multiple commands connected with pipes) to print out all of the legal prefixes to the English words which end with “berry”. For instance, for the word “blueberry”, your command should print “blue”. However, your command should print out this prefix for every word in `/usr/dict/words` which ends in “berry” (and nothing else).

```
cat /usr/dict/words | grep "berry$" | sed -E -e 's/(.*)berry$/\1/g'
```

Notes: make sure they have the “$” on the end of the grep and sed commands, it’s necessary for both. When in doubt try it out, whoever grades this should make sure that their command prints the right thing for any word that ends in berry, and make sure it removes all lines that don’t end in berry (even if they contain berry somewhere else). Best concrete example is the word strawberry-raspberry, which should turn into strawberry-rasp.

3) Bash Scripting

a) `varsearch.sh`: (20 points) You have stored in your bash shell variable SRCH a string. Write a shell script (including everything you would include in a shell script file) which takes any number of filenames as arguments, and uses grep to print out any lines from those files containing the string in SRCH.

```
#!/bin/bash

while [ $# -gt 0 ]
do
   cat $1 | grep "$SRCH"
   shift
done
```

NOTE: the double quotes around $SRCH are super important, single quotes don't work here

b) (5 points) Suppose you have the SRCH variable set in your current terminal. What command would you need to use to run the script above (`varsearch.sh`)? Why is it necessary to run this particular script in this way?

```
source varsearch.sh arg1 arg2 etc…
```

If we ran it with ./varsearch, the SRCH variable would be brand new, and we wouldn't be able to use the value we've already set

4) Bash Scripting

a) `nth_arg.sh`: (7 points) Write a shell script (including everything you would in a shell script file) which takes 1 argument, and then some additional arguments. The first argument to this script will be a number n. This script should then print out the nth argument it was given, assuming it was given enough arguments. For example, if the first argument is 1, the script should always print out “1”. For another example, ‘`./nth_arg.sh 3 a b c`’ should print ‘b’.

```
#!/bin/bash

x=$1

while [ $x -gt 1 ];
do
    shift
    ((x=x-1))
done

echo $1
```

b) `compile.sh`: (13 points) Write a shell script which takes 2 arguments: the first is a C source filename, and the second is an executable. Your shell script should invoke gcc to compile the C source into the executable, but only if the C source file is newer than the executable, or the executable doesn't exist. Hint: look at the first page references.

```
#!/bin/bash

if [ -f $2 ]
then
    if [ $1 -nt $2 ]
    then
        gcc $1 -o $2 -Wall -std=c11 -g
    fi
else
    gcc $1 -o $2 -Wall -std=c11 -g
fi
```

5) C Programming

a) (10 points) Write a C function named `decap` which takes one argument, a string, and truncates (shortens) it to be the empty string. Your function should modify the given string in place, and not return anything.

```
void decap (char* str) {

str[0] = '\0';

}
```

b) (6 points) Write a C function `decap2` which takes one argument, which may be a string, or may be NULL. The function `decap2` should call `decap` on its argument unless it's NULL.

```
void decap2 (char* x) {
    If (x) {
        decap(x);
    }
}
```

c) (4 points) Why couldn't I call `decap` on `x` declared the following way?

```
char* x = "Hello";
```

`The string "Hello" is immutable here, so we can't change it. Decap will attempt to mutate the character H, which is immutable.`

6) Hint: Draw a Picture (12 points)
What does the following C program print during its execution? Fill in the blanks

```
#include <stdio.h>
#define i 17

//HINT: Draw a picture

int* a(int x, int *y, int *z) {
    if (x) {
        *z = 5;
        return y;
    } else {
        *y = 18;
        return z;
    }
}

int b(int* x, int* y) {
    int res = *x + *y;
    *x = 4;
    *y = 12;
    int * r = a(res,x,y);
    printf("*x = %d\n", *x);
    printf("*y = %d\n", *y);
    printf("*r = %d\n", *r);
    return i;
}

int main() {
    int x = 5;
    int y = 9;

    int* p = &x;
    int* q = &y;

    int res1 = b(p,q);

    printf("res1 = %d\n", res1);

    return 0;
}
```

```
Output:

*x =   _4__

*y =   _5__

*r =   _4__

res1 = _17_
```