CSE 374 Midterm Exam 11/2/15

Name ____ Id # _______________

There are 8 questions worth a total of 100 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 _________________ / 100

1.	______ / 10
2.	______ / 12
3.	______ / 12
4.	______ / 16
5.	______ / 4
6.	______ / 10
7.	______ / 16
8.	______ / 20

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 [ ] or [[ ]] 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

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*)

Files (<stdio.h>)

Some file functions and information:

- Default streams: `stdin`, `stdout`, and `stderr`.
- `FILE* fopen`(*filename*, *mode*), modes include "r" and "w"
- `char* fgets`(*line, max_length, file*), returns `NULL` on end of file
- `int feof`(*file*), returns non-zero if end of *file* has been reached
- `int fputs`(*line, file*)
- `int fclose`(*file*)

A few `printf` format codes: `%d` (integer), `%c` (char), `%s` (char*)

Question 1. (10 points) Suppose the following files and subdirectories exist in a directory:

```
.bashrc                      proj/test.exe
.emacs                       proj/test.c
.bash_profile                proj/test.o
proj                         proj/thing.c
proj/data                    proj/thing.h
proj/data/dict.txt           proj/thing.o
proj/data/smalldict.txt
proj/notes
proj/notes/todo.txt
proj/notes/readme.txt
```

Answer the following questions assuming that this directory is the initial current directory when each of the following sets of commands are executed.

(a) (5 points) What output is produced by the following commands?

```
cd  proj
ls  *.[ch] > xyzzy
ls  notes/* >> xyzzy
cat xyzzy
```

(b) (5 points) What do the following commands do?

```
cd  proj
mv  */*.txt  ..
```

Question 2. (12 points, 4 each) Give regular expressions that could be used with `grep` (or `egrep`) to search the wordlist we used as an example in class for words that match the description given. For each answer, circle "grep" or "egrep" to indicate whether your answer is using basic or extended regular expressions.

Simplification: for this problem, assume that all letters are lower-case 'a'-'z'. You do not need to consider upper-case letters 'A'-'Z'.

(a) Words that contain only vowels (one or more of the letters aeiou). Examples include "eau" and "oui".

Circle: grep egrep

Answer:

(b) Words that contain exactly 7 characters and are palindromes, i.e., are the same forwards or backwards. Examples include "pip-pip", "rotator", and "repaper".

Circle: grep egrep

Answer:

(c) Words that contain the same sequence of three characters repeated three or more times. Examples include "hemidemisemiquaver" ("emi" repeated three times), "expressionlessness" ("ess"), "cha-cha-cha".

Circle: grep egrep

Answer:

Question 3. (12 points) (debugging) Suppose you have a large program named `app` that contains a function `f`. For each part below, be *very specific* about how you would run commands and programs and in what order to answer the question.

(a) (6 points) You would like to determine if `f` is ever called when `app` is run with the command-line parameter 42. You cannot change the source code but you can use the `gcc` and `gdb` commands however you like.

(b) (6 points) Extending the previous question, we have learned that `f` is called many, many times. Now suppose that `app` also has a function `g` and you would like to know if running `app 42` causes `f` to be called after `g` is called but before `g` returns. How would you answer this question efficiently without having to examine the situation every time `f` is called? (You may assume that `g` is called a small number of times.) As before, you cannot modify the code, but you can use `gcc` and `gdb` however you want.

Question 4. (16 points) (A little scripting) For this problem write a shell script that takes as arguments a simple string and a list of file names. The script should write to standard output only the names of the files that contain one or more copies of the given string. Nothing else should be written to standard output. Your script should use `grep` to check files to see if they contain the string. For example, if the script is name `listfiles`, then the command

```
  ./listfiles stdio foo.c hamlet.txt /usr/bin hw4.c handout
```

would produce the output

```
foo.c
hw4.c
handout
```

if these three files are the only ones that contain the string "stdio".

Your script should ignore any file names on the command line that are not ordinary files (i.e., ignore special files and directories like `/usr/bin` in the above example). The script should work properly if any of the files have names containing embedded spaces.

The script should exit with an appropriate error message if there is not at least one argument (the search string). This message can be written to either `stdout` or `stderr` – your choice.

Useful `grep` information: the exit status code returned from `grep` when it terminates is one of the following:

- 0 – some line in the file was selected
- 1 – no lines were selected
- 2 – some error occurred

Restriction: `grep` has dozens of options and those likely include ones that might even do some or all of what this script is asked to do. But you *may not* use these. Use `grep` in your script to search ordinary files in the list and use shell commands and options to suppress any unwanted output that `grep` would ordinarily write to `stdout`.

Please write your answer on the next page.

(You may remove this page for reference if you wish.)

Question 4. (cont.) Write your answer on this page.

Question 5. (4 points) (Aliases). Although we'll eventually learn about `make` and how to automate the build process, for right now it would be nice just to have something so we can compile programs without having to type all of the necessary `gcc` options each time.

Give a shell command to define an alias `build` so that `build` *`x y z...`* will execute the command `gcc -Wall -g -std=c11` *`x y z...`* (where *x y z...* are any additional options, file names, or arguments to be supplied to `gcc`).

Question 6. (10 points) (`sed` and style) The clint style checker flags several things that we could probably fix with a simple `sed` command. Fill in the blanks in the `sed` commands below so they will read the file x.c and write to `stdout` a copy of the file in which the identified style issue has been fixed.

Restrictions: you must use basic regular expressions, not extended, and each solution must use a single `sed` "s" command with appropriate patterns to do the job.

(a) (5 points) Change all occurrences of "`while(`", "`for(`", and "`if(`" by adding a blank before the "("s to get "`while (`", "`for (`", and "`if (`". Hint: in a `sed` pattern, '`\(`' is a `sed` parenthesis for grouping parts in a pattern; a plain '`(`' is an ordinary parenthesis character. The regexp operator `\|` can be used as "or", e.g., *p1*`\|`*p2* matches either *p1* or *p2*.

sed -e 's/ _______________________________________ / ___________________ / __ ' x.c

(a) (5 points) Remove all trailing whitespace from all source lines. For this question, only tabs (`\t`) and blanks are considered to be whitespace.

sed -e 's/ _______________________________________ / ___________________ / __ ' x.c

Question 7. (16 points) The traditional, annoying C program.

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

void confuse(int *a, int *b, int n) {
  *a = 20;
  b[1] = *b + n;
  a = b + 2;
  *a = 15;
  printf("confuse: *a = %d, *b = %d, n = %d\n", *a, *b, n);
}

int main() {
  int x = 17;
  int a[4];
  a[0] = 10;  a[1] = 11;
  a[2] = 12;  a[3] = 13;
  int * p = &x;
  int * q = a;
  q[2] = 42;
  printf("x = %d, *p = %d, *q = %d\n", x, *p, *q);
  printf("a = {%d, %d, %d, %d}\n", a[0], a[1], a[2], a[3]);

  confuse(p, q, x);

  printf("x = %d, *p = %d, *q = %d\n", x, *p, *q);
  printf("a = {%d, %d, %d, %d}\n", a[0], a[1], a[2], a[3]);
  return 0;
}
```

Fill in the lines below to show the output produced when this program is executed? (It does compile and execute with no errors.) Although not absolutely required, you should draw diagrams showing variables and pointers to help answer the question and to help us award partial credit if needed. Output:

```
x = __________, *p = __________ *q = __________

a = { _________, ________, _________, _________ }

confuse: *a = _________, *b = _________, n = _________

x = __________, *p = __________ *q = __________

a = { _________, _________, _________, _________ }
```

(extra space provided on the next page for any diagrams or scratch work)

Question 7. Extra work space if needed.

Question 8. (20 points) (The small C programming exercise.) On the next page, give an implementation of the standard C library function `strncat`. The basic idea of this function is to add to a string by appending the contents of another string. Example:

```
char s[20];
strncpy(s, "sea", 20);
strncat(s, "hawks", 5);
printf("%s\n", s);
```

prints `seahawks`.

The full specification of `strncat` is:

```
char* strncat(char *dest, char *src, int n);
```

- Append up to *n* characters from the contents of *src* to the end of *dest*.
- If the null ('\0') character that terminates *src* is encountered before *n* characters have been copied, then the null character is copied but no more.
- If no null character appears among the first *n* characters of *src*, then the first *n* characters of *src* are copied and a null character is supplied to terminate *dest*, i.e., *n*+1 characters in all are written.
- If $n \leq 0$ then calling `strncat` has no effect.
- The function returns the value *dest* (i.e., a copy of the original *dest* pointer).

Restriction: you ***may not*** call any other library functions in <string.h> or elsewhere. You should implement `strncat` by processing the strings (character arrays) directly.

You may not need nearly as much space as is available for your answer.

Write your answer

on

the

next

page

(You may detach this page from the exam if that is convenient.)

Question 8. (cont). Complete the definition of function `strncat` below.

```
   char* strncat(char *dest, char *src, int n) {













}
```