CSE 303 Midterm Exam

October 31, 2007

Name _____Sample Solution_____

The exam is closed book, except that you may have a single page of hand-written notes for reference.

If you don't remember the details of how a specific function or command works (parameter lists, whether an error is indicated by a non-zero return code or null pointer, etc.), write down the assumptions you need to make and as long as they're reasonable you're ok.

If you have questions during the exam, raise your hand and someone will come to you. Don't leave your seat.

Please wait to turn the page until everyone has their exam and you have been told to begin.

Advice: The solutions to many of the problems are quite short. Don't be alarmed if there is a lot more room on the page than you actually need for your answer.

1	/ 12
2	/ 16
3	/ 14
4	/ 14
5	/ 15
6	/ 15
7	/ 14
Total	/ 100

Question 1. (12 points) Suppose that the current directory contains the following files:

.bashrc	cow.cat	foo.c
.d.c	cpp.cc	tickle.c
.profile	dog.cow	

What output is produced when each of the following commands is executed in this directory?

(a) **echo *c**

cpp.cc foo.c tickle.c

(b) **echo .*c**

.bashrc .d.c

(c) **echo *.c**

foo.c tickle.c

(d) **echo .*.*c**

.d.c

(e) **echo *.*.c**

***.*.c (full credit if you said no match or left this blank)**

(f) **echo *c*c***

cow.cat cpp.cc tickle.c

Question 2. (16 points) For each of the following, give a regular expression suitable for **grep** that matches the lines described.

(a) Lines that contain the string **uw** with either, both, or neither characters capitalized.

[uU][wW]

(b) Lines that contain two or more of the vowels a, e, i, o, u.

[aeiou].*[aeiou]

(c) Lines that contain a url that starts with **http://** followed by anything, then followed by either **.edu** or **.org**. There may be anything else on the line before or after this url.

http://.*\.(edu|org) **(egrep only)**

The question should have been better worded to say that regular expressions that worked with either grep or egrep would be ok.

(d) Lines that begin with the four-character string **($1)**

^(\$1) **(grep)**

^\(\$1\) **(egrep)**

^[(][$]1[)] **(either grep or egrep)**

Any of these answers received full credit

Question 3. (14 points) Give the code for a bash shell script that accepts a list of file extensions as arguments and turns on the execute (x) permissions for all files in the current directory that end with those extensions. For example, if the shell script is named **addx**, then entering

addx exe sh

should set the execute permissions for all files whose names end in **.exe** or **.sh**. Execute permissions should be changed for all users (owner, group, world), but other permissions (read, write) should not be altered.

Fine print: If there are no arguments when the script is executed, it should quietly do nothing; similarly, if any of the arguments don't appear as an extension on any file, they can either be ignored or an error message can appear – whichever makes your job easier (but remember to process all of the arguments even if some of them don't appear on any file). You don't need to have explicit tests for any of these conditions as long as your script works properly without them.

Here are two possible solutions

```
#!/bin/bash
while [ $# != 0 ]
do
   chmod +x *.$1
   shift
done
```

```
#!/bin/bash
for ext in $@          (the "in $@" part is optional)
do
   chmod +x *.$ext
done
```

Question 4. (14 points) The file **oldnames** contains a list of names in the format *lastname, firstname*, i.e., last name, then a comma, then the first name. Write a **sed** command that reads the file **oldnames** and produces a file **newnames** where the names in the new file are rearranged in the order *firstname lastname* with no comma between them. Each *lastname, firstname* pair appears on a separate line in the original file.

Be sure that your script is robust enough to deal with names that consist of more than a single word in either the *firstname* or *lastname* part or both. For example, if the input contains

Smith, John
van Winkle, the Hon. Rip
Rowling OBE, Ms. J. K.

then the output should contain

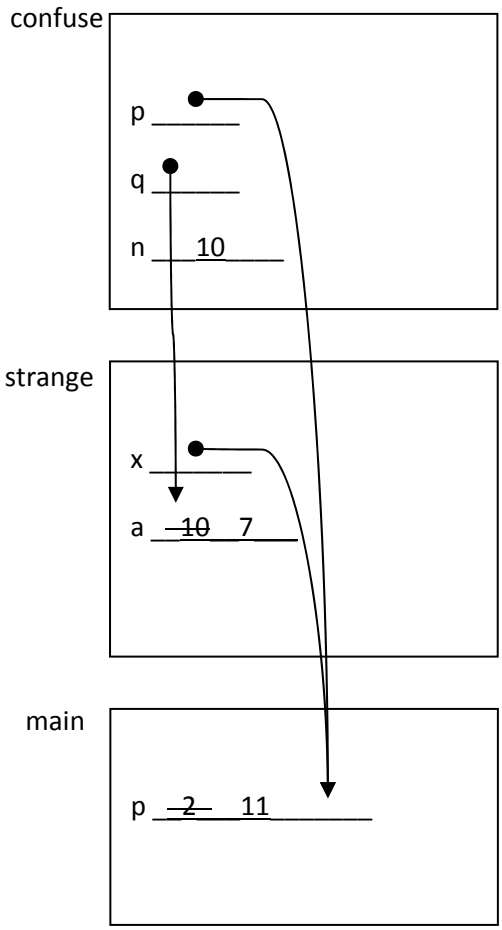
John Smith
the Hon. Rip van Winkle
Ms. J. K. Rowling OBE

You are guaranteed that there is only a single comma in each line and it separates *firstname* and *lastname*; and there is no other text in the input besides the names. The output file may have additional spaces before, after, or between parts of the names if this is helpful.

```
sed 's/\(.*\),\(.*\)/\2 \1' oldnames >newnames
```

Question 5. (15 points) Consider the following program (which is legal C and compiles and runs without errors):

```
#include <stdio.h>
void confuse(int* p, int* q, int n) {
  *q = n-3;
  *p = n+1;
  /* draw picture at this point for part (a) */
  printf("confuse: %d %d %d\n", *p, *q, n);
}
void strange(int* x) {
  int a;
  a = 10;
  printf("before: %d %d\n", *x, a);
  confuse(x, &a, a);
  printf("after: %d %d\n", *x, a);
}
int main(int argc, char* argv[]) {
  int p;
  p = 2;
  strange(&p);
  printf("done: %d\n", p);
  return 0;
}
```



(apologies for the truly awful drawing, but your instructor's "office art" skills aren't up to doing battle with Word)

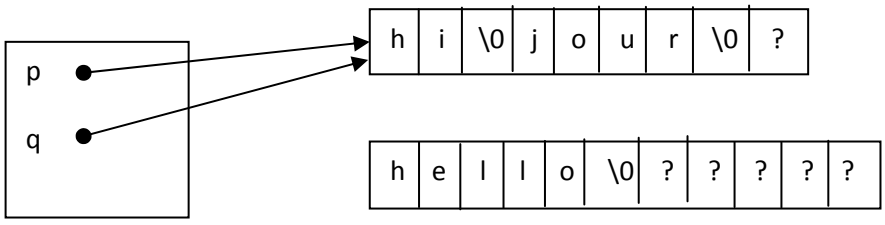
(a) Draw a diagram with boxes and arrows showing the situation right at the point when execution reaches the `printf` statement in function `confuse`. Be sure to show the values of all of the variables in each active function. If a variable is a pointer to another variable, indicate its value by drawing an arrow between the variable name and the storage location (variable) that it points to. Draw your diagram above, either next to or below the code.

(b) What output is produced when this program is executed? (Literal text in the format strings like "before:" is copied to the output and appears with the numbers.)

before: 2 10
confuse: 11 7 10
after: 11 7
done: 11

Question 6. (15 points) Consider the following function

```
void f() {
  char* p = (char *)malloc(9*sizeof(char));
  char* q = (char *)malloc(11*sizeof(char));
  strcpy(p, "bonjour");
  strcpy(q, "hello");
  free(q);   // added for part (c)
  q = p;
  strcpy(p, "hi");
  printf("%s\n", p);
  printf("%s\n", q);
  free(p);   // added for part (c);  free(q) would also work
  /* draw picture for part (a) at this point */
}
```



(a) Suppose we execute this function. Draw a diagram above showing all of the local and dynamically allocated storage used by this program at the point right before the end of the function execution. Use arrows to indicate what the pointers refer to. Include all allocated storage in your picture, even if some of the allocated storage is no longer referenced by any pointer. Use '?' to indicate the contents of any uninitialized storage locations or array elements. Be sure to show `'\0'` bytes explicitly (note that uninitialized bytes do not necessarily have the value `'\0'`).

(b) What output is produced when this function is executed?

hi
hi

(c) This function allocates storage without freeing it, i.e., it contains memory leaks. Add **free** statements to the code above at appropriate places to properly free all dynamically allocated memory before the function returns but after the memory is no longer needed by the function.

Question 7. (14 points) Write a C program that reads text from stdin and copies it to stdout, but when it reads a line with more than 80 characters, it only prints the first 80 characters on the output line (not counting the trailing '\0' at the end of a line). You may assume that no input line has more than 5000 characters, including any trailing '\0' returned when the line is read. Various #include statements and the beginning and end of the program's main function are written for you below (but you may cross out or rearrange this code if needed). You may include all of your code in a single main function if you like.

(Hint: Storing a '\0' in an appropriate place might be something useful to do. But there are many ways to approach the problem.)

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


int main(int argc, char* argv[]) {
   char line[5000];
   while (gets(line) != NULL) {
      line[80] = '\0';
      printf("%s\n", line);
   }

    return 0;   /* end of main */
}
```