

NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
SEMESTER EXAMINATION FOR
Semester 3 AY2003/2004

CS2281 Programming in UNIX

June 2004

Time Allowed 2 hours

MATRICULATION NUMBER: _____

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains TEN (10) questions and comprises ELEVEN (11) printed pages, including this page.
2. Answer **ALL** questions.
3. Answer all questions in the space provided in the script. Please indicate clearly (with an arrow) if you also use the other sides of the sheets for your answers.
4. This is an **OPEN BOOK** examination.
5. Write your matriculation number in the space provided above.

EXAMINER'S USE ONLY				
Question	Mark	Score	Graded By	Checked By
Q1	10			
Q2	10			
Q3	10			
Q4	10			
Q5	10			
Q6	10			
Q7	10			
Q8	10			
Q9	10			
Q10	10			
TOTAL	100			

1. (10 points) The following piece of C code suffers from buffer overflow security flaw. Explain how buffer overflow can occur in this example, and how an attacker can trick the system to execute arbitrary code on the system. State any assumptions that you make.

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    int array[5];
    int i;
    for (i = 0; i < argc; i++)
    {
        array[i] = atoi(argv[i]);
    }
}
```

Answer:

2. (10 points) Write down FIVE mistakes that you can spot in the following C code. The program is supposed to substitute all spaces in the given command line argument with underscore, and output the result to stdout. Mistakes can include: syntax errors, logical errors, bad coding style etc. Explain, for each mistake, what are the consequences (segmentation fault, compiler warning, violate spirit of UNIX, etc.) and how you would fix it. You may refer to the given line numbers in your answer. You should consider the serious mistakes first.

```
1 int main(int argc, char *argv[])
2 {
3     char *in = argv[1];
4     char *out = "";
5     while (in)
6         if (in == " ")
7             out = '_';
8         else
9             out = in;
10    printf("%s\n", out);
11 }
```

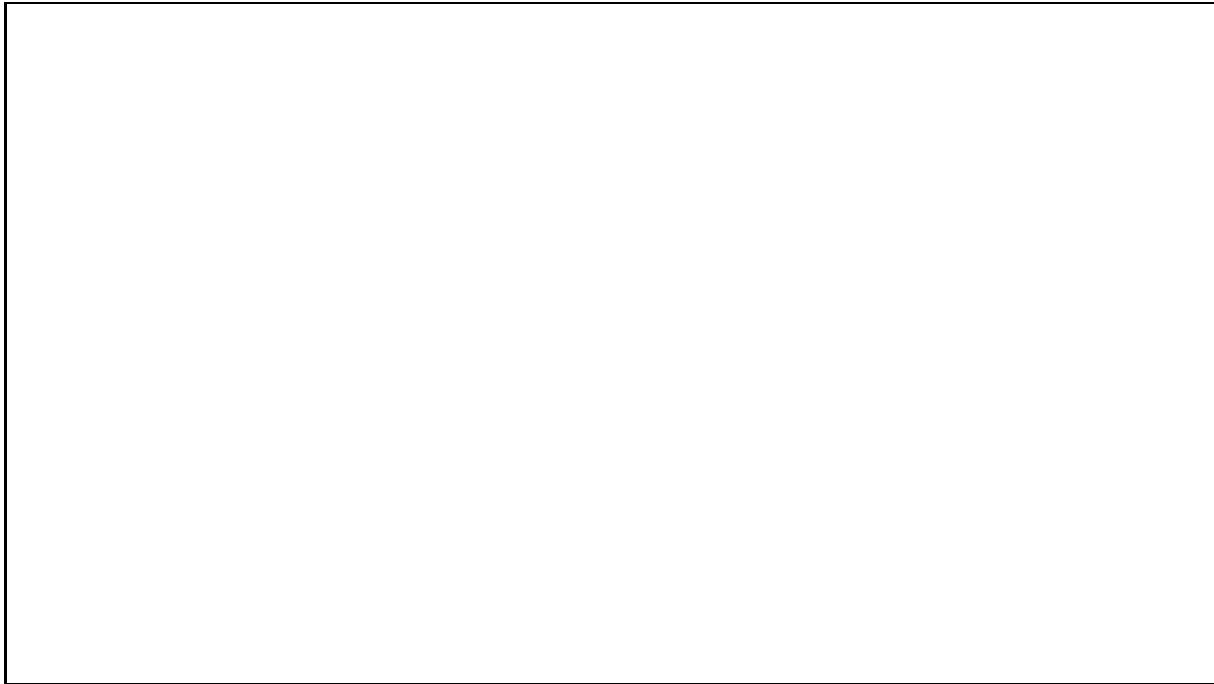
Answer:

3. (10 points) Write a function that takes in an `int` argument, and returns the number of bits in the argument that is set to 1.

For example, `count_bits(7)` should return 3, and `count_bits(1024)` should return 1.

Answer:

```
int count_bits (int x) {
```



```
}
```

4. (10 points) I have written a program using two separate C files, A.c and B.c, and two header files D.h and E.h. The contents of the C files and header files are given below.

```
$ cat A.c
#include "D.h"
int main() { callB(); }
```

```
$ cat B.c
#include "E.h"
int callB() { }
```

```
$ cat D.h
#ifndef D_H
#define D_H
#include "E.h"
int callB();
#endif
```

```
$ cat E.h
#ifndef E_H
#define E_H
#define CONST_PI 3.1415926
#endif
```

Write a Makefile that automatically compiles the above program and creates an executable call main. The dependencies among the files must be properly indicated in the Makefile, so that only the affected files are recompiled when changes are made to one of the files.

Answer:

5. (10 points) Write an `awk` script called `reverse.awk` that reads in a file and prints all lines in the file in reverse order. For example:

```
$ cat DATA
ABC
DEF
GHI
$ awk -f reverse.awk DATA
GHI
DEF
ABC
$
```

Answer:

6. (10 points) `uniq` is a utility in UNIX that removes duplicate, *consecutive* lines. We have seen in class that, to remove *all* duplicate lines, we typically sort the file first, before sending the file to `uniq`. This method will not work if the order of the lines in the original file is important.

Write an `awk` script called `uniq.awk` that removes all duplicate lines, but still maintains the order of the lines in the file. For example:

```
$ cat DATA
AAA
CCC
ZZZ
CCC
AAA
XXX
$ awk -f uniq.awk DATA
AAA
CCC
ZZZ
XXX
$
```

Answer:

7. (10 points) A file called DATA contains the following lines.

```
$ cat DATA
4.32
4.3.2
.4.3.2
432.
-432
+43+2
.432
...
+.
+++
```

What is the output of the following command?

```
$ egrep '[-+]?([0-9]*\.)+[0-9]*' DATA
```

Answer:

8. (10 points) The environment variable `$PATH` stores a list of directories (separated by `:`) which is used by the shell to search for a particular command.

```
$ echo $PATH
/local/bin:/usr/local/bin:/usr/bin:/usr/ucb:/usr/local/java/jdk/bin
```

Write a bash script that takes a command name, and checks if the command exists in *each* directory of `$PATH`. If the command exists in a directory, print out the directory name and the command name.

For example, suppose there are three copies of the command `ls` located in directory `/local/bin`, `/usr/local/bin` and `/usr/ucb`, running your script (named `wheregot.sh`) with `ls` as argument will produce the following.

```
$ wheregot.sh ls
/local/bin/ls
/usr/local/bin/ls
/usr/ucb/ls
```

Answer:

9. (10 points) How many lines of output will you see from the following program? How many '0's will be printed? Explain your answer.

```
#include <stdio.h>
int main()
{
    int i = 0;
    fork();
    fprintf(stderr, "%d\n", i++);
    fork();
    fprintf(stderr, "%d\n", i++);
    wait();
}
```

Answer:

10. (10 points) Explain what will happen when you run each of the following two programs. Compare the behaviours of both programs.

Program 1: An executable binary produced from the following C code, `x.c`. Assume that the executable is executed using its full path name.

```
int main(int argc, char *argv[])
{
    execl(argv[0], argv[0], NULL);
    return 0;
}
```

Program 2: A shell script called `x.sh` containing the following lines.

```
#!/bin/bash
$0
```

Answer: