Instructions:

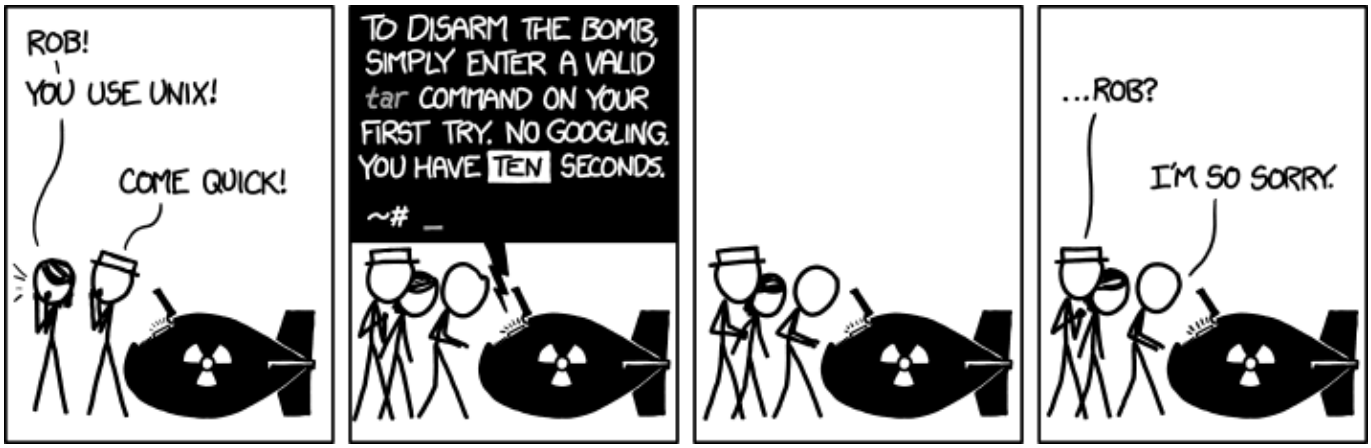
- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the permitted one-page formula sheet. No calculators or other electronic devices may be used. The use of any such device will be interpreted as an indication that you are finished with the test and your test form will be collected immediately.
- Answer each question in the space provided. If you need to continue an answer onto the back of a page, clearly indicate that and label the continuation with the question number.
- If you want partial credit, justify your answers, even when justification is not explicitly required.
- There are 7 questions, some with multiple parts, priced as marked. The maximum score is 100.
- When you have completed the test, sign the pledge at the bottom of this page and turn in the test.
- If you brought a fact sheet to the test, write your name on it and turn it in with the test.
- Note that either failing to return this test, or discussing its content with a student who has not taken it is a violation of the Honor Code.

Do not start the test until instructed to do so!

Name Solution
printed

Pledge: On my honor, I have neither given nor received unauthorized aid on this examination.

signed



xkcd.com

In the key, I've written answers in blue and explanatory commentary (that I wouldn't have expected as part of your answer) in green.

1. Suppose that the PATH variable for your Linux user account is currently set to be:

```
./:/usr/local/bin:/bin:/sbin:/usr/bin:/usr/sbin:~/bin
```

You've written a C program to provide a customized version of the standard `ls` command (which is located in the directory `/bin`). You want to install the executable for your custom program so that whenever you type the name of your executable program it will be executed by default, instead of the standard `ls`, but you want the standard version of `ls` to also be available.

You may assume that (possibly aside from the standard `ls`) there are no other executables on the system with the same name as your version. You may not modify the PATH variable unless there is no other way to achieve the given goal.

a) [6 points] Suppose that you've named the executable for your program `ls` (arguably a bad decision), and you have root permissions, so you can modify the contents of any of the directories shown above. Describe ONE place you could you put your executable to achieve your goal.

Key fact: when you enter the name of a program as a command, the shell looks for a matching executable file, checking the directories in the path in the order they're listed in the PATH variable. The directories in the path are, in order:

/usr/local/bin
/bin
/sbin
/usr/bin
/usr/sbin
~/bin

./ doesn't really count… it doesn't name a particular directory, but rather it refers to the one that you are currently in, which could be any directory on the system.

So, you must put your executable in a directory that's in the path before /bin.

Putting it into ./ doesn't make sense (unless you only use one directory on your system). Therefore, you would have to install your executable in /usr/local/bin.

You could add some other directory to the path, before /bin, and install your executable there, but the instructions are to avoid that unless it's necessary to modify the path.

b) [6 points] Reconsider part a), but now you've decided to name your executable `myls` instead of `ls`.

The difference now is that you can just put your executable into ~/bin or any other directory in the path, since there aren't any other executables on the system named myls.

So, possible answers would include all of the directories listed in the discussion of part a).

c) [6 points] Reconsider part b), with the restriction that you cannot modify any of the directories that are directly under the root directory.

Now it gets a bit more restrictive, but you can still use /usr/local/bin. Other possible answers include:

/usr/bin
/usr/sbin
~/bin

That wasn't the intent of the question... I should have said you could only modify directories that belonged to your user account. In that case, you'd have to change the path; the simplest solution then would have been to move ~/bin to be earlier in the path list.

2. Suppose that your user id is `johokie` and that a listing (`ls –l`) of the contents of your current working directory provides the following information:

```
[CentOS tmp 1234]$ ls –l
total 32
-r-------- 1 johokie academic    7 2011-09-03 11:56 data.txt
-rw-rw-rw- 1 johokie academic  512 2011-09-03 11:56 results.txt
-rwx--x--x 1 johokie academic 9045 2011-09-03 11:50 newton
-rwxr----x 1 johokie academic 8734 2011-09-03 11:35 raphson
```

List the Linux command(s) you would use to accomplish each of the following goals, in a minimal fashion. That is, there should be no other rights granted or removed unless it is unavoidable. Assume the commands will be entered in the current working directory whose contents are shown above. The parts are independent; i.e., you start with the initial settings shown above.

a) [5 points] Allow only the owner and members of the group academic to *execute* `raphson`.

chmod g+x,o-x raphson or chmod 750 raphson

b) [5 points] Prevent the group and others from *modifying* the contents of the file `results.txt`.

chmod go-w results.txt or chmod 644 results.txt

c) [8 points] Write a bash function that will swap the contents of two files. The command could be invoked as:

```
swap data.txt results.txt
```

There should be no additional side effects from this command, so any temporary files created should be cleaned up, etc. You do not need to verify that either file exists.

Something like this would work:

```
swap () {
   mv $1 temp
   mv $2 $1
   mv temp $2
}
```

Most people lost points for not writing the code inside of a function or if the function they wrote wasn't generic, i.e. only working with results.txt and data.txt.

3. Analyze the C function given below. Be sure to pay careful attention to the comments. The function is syntactically correct, but the logic of the design is not entirely valid.

```
/**  Determines the smaller of two integers, and communicates that value
 *   to the caller.
 */
void min(int x, int y, int smaller) {

   if ( x >= y ) {
      smaller = y;
   }
   else {
      smaller = x;
   }
}
```

a) [6 points] Explain clearly why the given implementation will not achieve the stated goals.

Parameters are passed by value, so min() is working with a copy of the caller's variables.

Therefore, assigning a value to smaller within min() is just modifying a variable that's local to min(), and there's no way for the caller to be aware of that.

There were a number of bogus answers, including:

- **"min() doesn't return a value", "min() has a return type of void", etc.**
  TRUE, but that doesn't explain WHY nothing is communicated to the caller. It's obvious the programmer is trying to use the parameter bigger to communicate a value to the caller… why does that not work?
- **if x == y the function will return x, which is not smaller than y**
  TRUE, but not relevant. Both the Java and C libraries define max() and min() functions that do precisely what this function is trying to do. If you want to know which of two values is bigger, and the values are the same, then that's the biggest value under consideration.

b) [6 points] Write a corrected version of the function that will achieve the stated goals. You may make any changes you like, as long as you don't use any file-scoped variables.

Aside from using file-scoped variables (banned) or pointers (not covered yet), the only way to fix this is to have the function return a value to the caller:

```
/**  Determines the smaller of two integers, and communicates that value
 *   to the caller.
 */
int min(int x, int y) {

   if ( x <= y ) {
      return x;
   }
   else {
      return y;
   }
}
```

(Since there's no use for the third parameter, it should be eliminated.)

There were a number of incorrect versions of this. The most common was to call printf() and write the bigger value to standard output. That does not "communicate that value to the caller".

What IS the caller? It's another C function, not a human.

4. For each part of this question, write a declaration in the code below that will achieve the desired goal. Label each declaration with a comment indicating which part of the question it goes with.

```
static int D;   // part d ("static" makes it private to the file)
int B;          // part b (file-scoped, so has static duration by default)

int F() {


   while ( . . . ) {

      static int A;    // part a

   }

}

int G() {

   int C;   // part c

}
```

A significant number of answers used bogus modifiers, like:

intern, internal, global
public, private, final

This isn't about Java!

a) [3 points] I want an integer variable named `A` that can only be accessed within the while loop in `F()`, and that retains its value between calls to `F()`.

To be accessible only within the loop in F(), A must be declared inside that loop. To retain its value between calls, A must be declared as static.

b) [3 points] I want an integer variable named `B` that can be accessed from within `F()` and `G()`, and outside the file, and that retains its value persistently as long as the program is running.

To be accessed from both F() and G(), B must be declared at file scope. But now, we want B to be accessible outside the file, so B cannot be static.

c) [3 points] I want an integer variable named `C` that can be accessed anywhere in the body of the function `G()`, but nowhere else.

For C to be accessible throughout the body of G() but nowhere else, A must be declared at the beginning of the body of G() (or be a formal parameter in G() but that was irrelevant).

It would be OK for C to be static.

d) [3 points] I want an integer variable named `D` that can be accessed from within `F()` and `G()`, but not from outside the file containing them.

As in part b), D must be declared at file scope. To NOT be accessible from outside the file, D must be declared as static.

5. [14 points] Implement the C function described below. Be sure to note the restrictions, violating the restrictions will result in a very low score.

```
/* Each call computes and returns a number in the Fibonacci sequence.
 * The Fibonacci sequence is defined by the recurrence:
 *       F_n = F_n-1 + F_n-2
 *       with seed values F_0 = 1 and F_1 = 1
 *
 * Example usage:
 * Fib() = 1   // first call
 * Fib() = 1   // second call
 * Fib() = 2   // etc...
 * Fib() = 3
 * Fib() = 5
 *
 *  Restrictions:
 *   - does not use global variables.
 *   - does not use any additional libraries.
 *   - does not make any use of character variables or arrays.
 *   - does not read input or write output.
 */
uint32_t Fibo() {
        // This is the most straightforward solution; there are simpler ways to implement this.
        static uint32_t count = 0;
        static uint32_t f1 = 1, f2 = 1;
        uint32_t current;

        if(count < 2)
        {
              current = 1;
              count++;
        }
        else
        {
              current = f1 + f2;

              f2 = f1;
              f1 = current;
        }

        return current;
    }
```

You really needed to use static variables here. Per the stated restrictions, if you created global variables, or rewrote the function signature, then you lost most of the points. I suggest you also read the extra information at the end of Q7 regarding uninitialized variables in C, they don't start out as NULL or 0, etc.

6. Analyze the short C program given below:

```
#include <stdio.h>

int main() {

   int a = 42;
   int b = 27;

   if ( a = b )        // used "=" instead of "==" for comparison
      f(a, a);         // calling f() w/o a prior declaration
   else
      f(a, b);

   return 0;
}

void f(int x, int y) {

   printf("The sum of %d and %d is %d.\n", x, y, x + y);
}
```

When this code is compiled with `-Wall`, `gcc` will issue warnings about two different issues (but, curiously, no error messages). Explain why each warning will be issued (what's going on in the code), and how to fix it. Be specific but brief. You don't have to discuss the warnings in the order the compiler would issue them.

[6 points] Warning 1:

The programmer evidently erred in writing the if-test, using assignment instead of equality. It's legal code, but suspicious, hence the warning.

We see this is a logic error from the code in the clauses of the if-then statement.

The fix is obvious, replace "=" with "==".

[6 points] Warning 2:

The programmer failed to write a declaration for f(). Functions must be declared before they are called.

The compiler encounters the first call to f(), does not see a declaration, and therefore issues an implicit declaration warning. It's a warning (unfortunately) because the compiler will infer a declaration for f() and proceed.

The compiler also issues a warning because it assumes the return type is int, and then sees the definition of f() with a return type of void.

There were lots of incorrect suggestions here, including:

- **need braces around the if/else clauses --- nope, not even in Java; if there are not braces, the body of the if/else is just the next single statement**
- **the else clause will never be executed --- true, but there's no warning for that in C**
- **f() should have a return type of int --- ???**

- the use of an expression, x + y, as a parameter to printf() is not allowed --- if so, that would cause an error message, not a warning; but it's perfectly legal, and entirely unobjectionable, to use an expression as a parameter in a function call
- the format specifier %d expects a value of type double --- nope
- the "." after the "%d" is not legal, having something to do with specifying the number of digits to print after the decimal point (precision) --- nope, it's just putting a period at the end of the sentence that's going to be printed.
- there's no return at the end of the body of f() --- yep, and it's not required; there's an implied return at the end of a function body, and that's also true in Java

7. [14 points] Implement a C function that will read IP addresses and port numbers from a file and sum them. Below is a sample input file:

```
128.255.77.11:22
124.55.88.99:458
127.33.47.84:
```

Each IP address is represented by 4 integers (between 0 and 255) separated by periods, followed by a colon, and then there may or may not be a port number (between 0 and 65535). The colon will always be there even if the port number is not. If the port number is not included you can assume a default value of 22. Given the above input file, your function will print the following:

```
128 * 255 * 77 * 11 * 22 = 608213760
124 * 55 * 88 * 99 * 458 = 27212454720
127 * 33 * 47 * 84 * 22 = 364013496
```

You should use this code as a starting point:

```
const unsigned int MAX_LINE_LENGTH = 101;

/* Pre:
 *     filename is a valid file.
 * Post:
 *     The function prints the results as described above.
 * Returns:
 *     The number of IP addresses read from the file.
 */
uint32_t read_file(char * filename)  {

   char Line[MAX_LINE_LENGTH];
   FILE * Input = fopen(filename, "r");

   unsigned int a, b, c, d, p;

   uint32_t count = 0;

   while (fgets(Line, MAX_LINE_LENGTH, Input) != NULL )
   {
         p = 22;

         if(sscanf(Line, "%u.%u.%u.%u:%u", &a, &b, &c, &d, &p)  >= 4)
         {
               unsigned int product = a * b * c * d * p;
               printf("%u * %u * %u * %u * %u = %u\n", a, b, c, d, p, product);
               count++;
         }

   }

   return count;
}
```

Here some things to consider for both this question and future C questions on tests:

- Enough of you forgot about sscanf (or some other scanf variant, I was forgiving here) and instead decided to try and parse the string yourself using C strings or arrays. If I saw something I thought would work in C I gave credit for it, the problem is most people who did this used Java/Python while trying to pass it off as C. There are a lot features in Java that don't exist in C. Arrays and strings don't know how long they are, you can't use operators like +=, you can't just cast a character to a number, parseInt() isn't a function supported by C. If you write a Java specific solution for part/all of a question, you shouldn't expect to receive credit for those parts.
- Uninitialized variables in C don't have a default value, when you declare variable without a value it can be anything. Sometimes you get "lucky" (is it lucky if bugs are hidden?) and the initial value is something sensible like 0, but this is not always the case. You can't assume int x; will create a variable with starting value 0, it could be 0, but that's just luck of the draw, it could also be -2129034.
- Unless you are talking about pointers you really shouldn't be using NULL. Variables are not references or pointers by default in C, they are just values. Variables without a value are not NULL; see the above bullet. NULL "is an integral constant expression that evaluates to zero" [www.cplusplus.com]. So it's 0, nothing more. Semantically, it's poor form to use NULL in place of 0 (though I don't think many of you were actually doing this intentionally) for normal variables, and comparing a standard integer with NULL will result in a compiler warning.