Instructions:

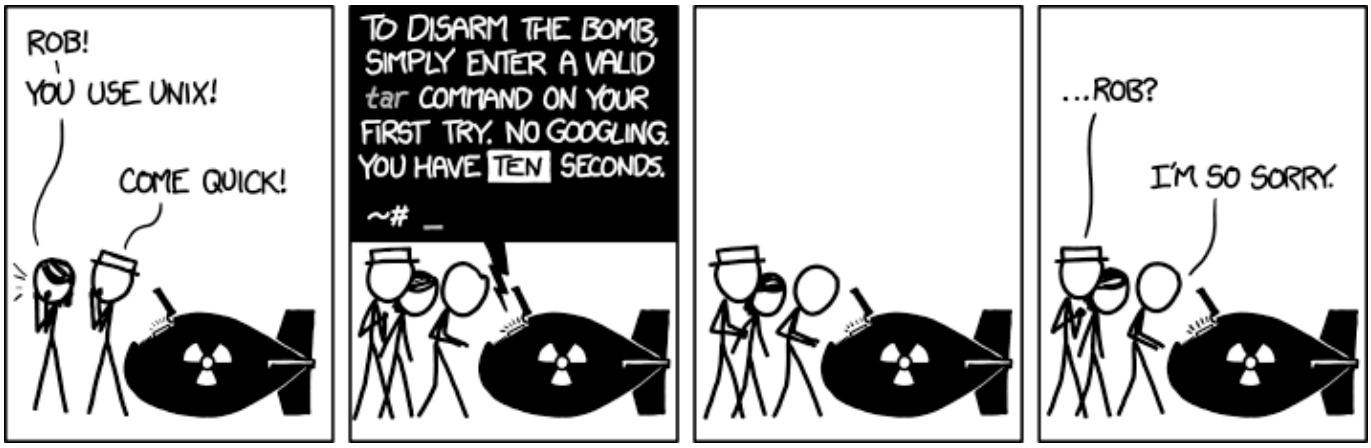
- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the permitted one-page formula sheet. This examination is closed book and closed notes, aside from the permitted one-page fact sheet. Your fact sheet may contain definitions and examples, but it may not contain questions and/or answers taken from old tests or homework. You may include examples from the course notes.
- 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 5 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


ROB!
YOU USE UNIX!
COME QUICK!
TO DISARM THE BOMB, SIMPLY ENTER A VALID tar COMMAND ON YOUR FIRST TRY. NO GOOGLING. YOU HAVE TEN SECONDS.
~# _
...ROB?
I'M SO SORRY.

xkcd.com

1. A CentOS user, `radagast`, executes the following commands in a bash shell:

```
1047 radagast in mirkwood > ls -l
total 32
-rwxrwx---. 1 radagast brown   13749 Feb 12 12:50 exterminate
-rw-rw----. 1 radagast brown    6407 Feb 12 12:50 runSleigh
-r--r-----. 1 radagast brown   23932 Feb 12 12:50 MyRabbits.txt
-r--r-----. 1 radagast brown   73429 Feb 12 12:50 NewRabbits.txt
-rw-------. 1 radagast brown   84730 Feb 12 12:50 WoodSpiders.odf

1048 radagast in mirkwood > file --mime-type MyRabbits.txt
MyRabbits.txt:  text/plain

1049 radagast in mirkwood > file --mime-type NewRabbits.txt
NewRabbits.txt:  text/plain

1050 radagast in mirkwood > cat NewRabbits.txt >> MyRabbits.txt
bash: MyRabbits.txt: Permission denied...
```

a) **[5 points]** Why did the command numbered 1050 not cause `NewRabbits.txt` to added to the tail end of `MyRabbits.txt`?

radagast does not have write permissions for MyRabbits.txt

radagast only needs read permission for NewRabbits.txt, and he has that.

b) **[5 points]** What command should `radagast` enter in order to be able to execute the command he tried in **a)**, without making it possible for any other user to do the same thing?

chmod u+w MyRabbits.txt

The point is to fix permissions for MyRabbits.txt w/o granting any unnecessary permissions. Many answers used numeric settings, which is fine. But, it's not OK to set execute permissions for the file, or to modify the permissions for any other files.

c) **[5 points]** What command should `radagast` enter in order to prevent all other users from executing `exterminate`, without making it impossible for him to do the same thing?

chmod g-x exterminate (although g-rwx is arguably better)

The "other" category already lacks execute permissions, so the only issue is to take them away from the "group" category. If you also deny "group" rw permissions, then those users can't make a copy of the file for their own use; arguably, that's a better solution.

d) **[5 points]** What command should `radagast` enter, working in his current directory, in order to create a subdirectory named `mydocs` in his home directory?

mkdir ~/mydocs

The common error here was to not correctly specify where the new directory is to be created.

e) **[5 points]** What single command should `radagast` enter in order to create a tar file containing all the files in his current directory, so that the tar file is named `backup.tar` and is located in the directory he created in part **d)**?

tar cf ~/mydocs/backup.tar *

There were some common errors here:

- **using -C to specify the destination; that only works when extracting a tar file, not when creating one**
- **using *.* to specify the files to be tar'd; that would not include "exterminate" for example**

2. **a)** **[10 points]** Write an implementation of the following C function:

```
/**  Returns the 10^k digit of N.
 *
 *   Pre: N and K are initialized
 *
 *   Examples:
 *     digitK(785, 0) == 5
 *     digitK(6543210, 3) == 3
 *     digitK(42, 5) == 0
 */
unsigned int digitK(unsigned int N, unsigned int K) {

   unsigned int pos = 0;

   while ( pos < K ) {
      N = N / 10;
      pos++;
   }

   return ( N % 10 );

}
```

There were many less efficient solutions than the one shown above. One error was to use a count-down loop like for (; K >= 0; K--); that's infinite since K is an unsigned int. Another common error was "off by one" on the digit, usually by dividing the wrong number of times, or computing the wrong divisor.

b) **[6 points]** Complete the implementation of the following main() function, following the requirements in the comments. Assume any needed include directives and function declarations are present.

```
int main() {

   unsigned int N = 0;          // the integer to be processed
   unsigned int K = 0;          // the power of 10
   unsigned int KthDigit = 0;   // the Kth digit

   // Prompt the user to values for N and K:
   printf("Type nonnegative values for N and K, separated by a space:\n");

   // Read those values into N and K:

   scanf("%u %u", &N, &K);

   Missing '&' on the parameters was common; otherwise most errors involved wildly wrong
   syntax.

   // Call the function to get the specified digit:

   KthDigit = digitK(N, K);

   The most common error, by far, was to not capture the return value by assigning it to the
   supplied variable.

   // The rest is irrelevant...

   return 0;
}
```

3. A C programmer writes a short program consisting of three files:

```
// main.c
#include <stdio.h>


int main() {

   int x = 19, y = 53;
   int max;

   Max(x, y);

   max = Bigger;  // use var from Max.c

   printf("x  :%3d\ny  :%3d\nmax:%3d\n",
           x, y, max);

   return 0;
}
```

```
// Max.h
#ifndef MAX_H
#define MAX_H

void Max(int A, int B);

#endif
```

```
// Max.c

int Bigger = 0;

void Max(int A, int B) {

   if ( A < B )
      A = B;

   Bigger = A;
}
```

a) **[8 points]** Two things are missing from the file `main.c`; without them the code will not compile. State what they are, and where they belong.

```
#include "Max.h"        // to "import" declaration of Max()

extern int Bigger;      // to link to variable Bigger defined in other file
```

Normally, both would go at file scope in main.c; however, each could be inside the function main() so long as each was before the related statement in main().

The common error was to #include Max.c; that violates the design of the program implementation. Usually, that was accompanied by a #include for Max.h; if you do both, you don't understand what's going on. If you did #include Max.c, then there is no need to also #include Max.h.

As for Bigger, the comment indicates we are trying to access the variable defined in Max.c; that's possible since Bigger has external linkage (declared at file scope, not static). But it requires an extern declaration in main.c

b) **[6 points]** If the two things referred to in part **a)** are added, the given code will compile, and the correct value will be printed for the maximum of 19 and 53.

State what value would be printed for `x`, and explain why.

Value: 19

Why: parameters are passed by copy (value), so Max() cannot modify any variable in main().

The common error here was in the justification. Many answers discussed the execution of the code inside the function Max(). That is entirely irrelevant, since parameters are passed by copy. If you discussed that at all, you were clearly missing the point.

4. The Folly numbers are defined as the sequence 0, 1, 3, 10, 33, 109, 360, 1189, …, where each element (after the first two) equals the sum of the element that is two positions before it, and three times the number that is one position before it.

a) [12 points] Write an implementation of the following function:

```
/**  Initializes the array with Perrin numbers.
 *
 *   Pre:  Sequence[] is an array of dimension Limit, or larger
 *   Post: Sequence[i] = ith Folly number, i = 0 to end of array
 */
void folly(uint32_t Sequence[], uint32_t Limit) {

      Sequence[0] = 0;
      Sequence[1] = 1;

      for (int i = 2; i < Limit; i++) {
            Sequence[i] = Sequence[i-2] + 3 * Sequence[i-1];
      }
}
```

Ideally, you would include some extra conditions in this code to check for cases where Limit is 0 or 1 to avoid writing the initial Sequence values outside of the bounds of the array, but I wasn't strict on this because you're creating 30 Folly numbers in the next subpart.

Common errors in this question included:

- Reinitializing the array (or Limit) with malloc – this isn't necessary, because you're receiving the array from the calling function
- Returning the array – note that the function has a void return type, and remember that any modification made to the array within the function will persist in the calling function.
- Using the dereference operator to access array positions – it is a valid method for stepping through the array, but shouldn't be used along with array notation (e.g., *Sequence[1]=1 is incorrect, *(Sequence+1)=1 is OK).
- Off by one errors in the loop bounds, either using <= instead of <, or using Limit-1.

b) [12 points] Complete the implementation of the following function that calls the one above:

```
int main() {

   // Declare variables for the call; we want to create 20 Perrin numbers.

   uint32_t Lim = 30;
   uint32_t Seq[Lim];
```

Common errors here included:

- Impromper syntax when declaring the array, often by using Java "= new()" syntax
- Using a datatype other than uint32_t – although C *should* handle the implicit cast from int here to uint32_t in the function, that's never a guarantee.

```
// Call the function to fill in the array.

Folly(Seq, Lim);
```

Common errors here included:

- **Setting the function equal to some variable, expecting a return array. Note that the function has a void return type.**
- **Including brackets with the array in the function call.**

```
// Print the EVEN values in the array with their indices,
//   in the format shown at right:

for (int i = 0; i < Lim; i++) {
   if (Seq[i] % 2 == 0) {
      printf("%3d%6"PRIu32"\n",i,Seq[i]);
   }
}
```

```
  0:       0
  3:      10
  6:     360
  9:   12970
. . .
```

Common errors here included:

- **Not using format specifiers to right-align the values into columns**
- **Using the wrong format specifiers for the datatype (note PRIu32 for uint32_t)**
- **Forgetting to perform an even/odd check before printing the values**
- **Print even array indices rather than even values**
- **Forgetting a newline \n character**

```
   return 0;
}
```

5. A double factorial (n!!) in mathematics is defined as the product of all integers beginning at 1, continuing through all non-negative integers with the same parity (even or odd) as n, and ending with a final multiplication by n. For example, 7!! = 7*5*3*1, while 8!! = 8*6*4*2. Both 0!! and 1!! are set to 1 by definition.

A novice programmer is asked to write a recursive implementation of the double factorial, and naturally codes up a horrid, buggy version. We'll try to track down the error.

First, we'll try to run the executable that the programmer has provided us.

```
Linux > ./doubleFac
Enter a positive integer pls: 7
Segmentation fault (core dumped)
```

The execution was obviously unsuccessful. Let's try to run it in `gdb` now. We'll begin by setting a breakpoint at `main()`.

a) [4 points] What is the command to set this breakpoint at `main()`?

b main

You could also use "break main"

After setting the breakpoint, we'll begin to run through the code. By stepping through the code in `main()` line-by-line, we see that the program prompts the user for input, receives that input, and then calls the recursive `doubleFac()` function.

```
Breakpoint 1 at 0x4005a5: file doubleFac.c, line 11.
(gdb) run
Starting program: /home/dumbprogrammer/doubleFac
Breakpoint 1, main () at doubleFac.c:11
11          printf("Enter a positive integer pls: ");
(gdb) n
12          scanf("%"PRIu8, &facBound);
(gdb) n
Enter a positive integer pls: 12
14          uint64_t dblFac = doubleFac(facBound);
```

We enter into the `doubleFac()` function, and begin to make recursive calls. The `doubleFac()` function takes a `uint8_t` variable called `bound` as a parameter, and it would be useful for us to keep track of the value of `bound` as the function continues to recurse.

b) [4 points] What is the command to continue to print the value of the variable `bound` after every instruction?

display bound

The key words here are "continue to print... ...after every instruction." The print instruction will only print a value once, rather than automatically printing it.

Some tried to use the "continue" command – this continues executing the code until the next breakpoint is reached. Others tried to set a watchpoint – this halts execution when the value of a variable reaches a set limit.

We continue to recurse through `doubleFac()` several times, watching the value of `bound` decrease from 12 to 10 to 8 and so on as it should. We'll pick up the execution during as we get closer to invoking the base case of the recursion.

```
1: bound = 12 '\f'
(gdb) n
26                return bound * doubleFac(bound-2);
1: bound = 12 '\f'
(gdb) n

Breakpoint 2, doubleFac (bound=10 '\n') at doubleFac.c:23
23          if (bound == 0 && bound == 1) {
1: bound = 10 '\n'
(gdb) n
26                return bound * doubleFac(bound-2);
1: bound = 10 '\n'
(gdb) n

...

Breakpoint 2, doubleFac (bound=2 '\002') at doubleFac.c:23
23          if (bound == 0 && bound == 1) {
1: bound = 2 '\002'
(gdb) n
26                return bound * doubleFac(bound-2);
1: bound = 2 '\002'
```

```
(gdb) n

Breakpoint 2, doubleFac (bound=0 '\000') at doubleFac.c:23
23          if (bound == 0 && bound == 1) {
1: bound = 0 '\000'
(gdb) n
26                return bound * doubleFac(bound-2);
1: bound = 0 '\000'
(gdb) n

Breakpoint 2, doubleFac (bound=254 '\376') at doubleFac.c:23
23          if (bound == 0 && bound == 1) {
1: bound = 254 '\376'
(gdb) n
26                return bound * doubleFac(bound-2);
1: bound = 254 '\376'
```

As you can see, the value of `bound` continued to decrease from 2 to 0, and then wrapped around to 254. Puzzled, we'll use the `list` command to print the full `doubleFac()` function.

```
(gdb) list doubleFac.c:26
21    uint64_t doubleFac(uint8_t bound) {
22
23          if (bound == 0 && bound == 1) {
24                return 1;
25          } else {
26                return bound * doubleFac(bound-2);
27          } //if-else
28
29    } //doubleFac
```

c) [8 points] What is the bug in the programmer's code? What evidence from the gdb output provided leads you to that conclusion? How do you fix it so that the program behaves correctly?

Bug:

- **In line 23 of the code, the programmer uses && in the recursion base conditional. As a result, the base case can never execute to halt the recursion, and the recursive case is continually called, always subtracting 2 from bound until the program crashes.**

gdb evidence:

- **As we traced through the recursion, we saw that the value of bound went from 2 to 0 to 254. This indicates that the base case was not halting the recursion at bound=0 as it should; instead, the recursive case continued to be called. When we listed the code for the doubleFac function, we could see the && operator used.**

Fix:

- **The && operator should be replaced with || in order for the base case to successfully halt the recursion.**

The most common issue with this question was not addressing one of the components, often forgetting to list the gdb evidence. Otherwise, some students keyed on the transition from 0 to 254 and claimed that the unsigned variable was the issue. However, allowing bound to decrease to -2 would violate the definition of double factorial that was provided. Others thought the discrepancy between the uint8_t bound parameter and the uint64_t return value was the issue, but the "bound * doubleFac(bound-2)" multiplication will still execute correctly, and there was no gdb evidence presented that indicated a datatype issue.

d) **[5 points]** We saw a test case with `gdb` that used an even number. If we had used an odd number instead, would the behavior on the original code have changed, or would the outcome still be the same? Why?

The outcome would still be the same. Instead of bound reaching 0 and wrapping around to 254, it would instead reach 1 and wrap around to 255. However, the high-level behavior remains the same: the code will continue to infinite loop through the recursion until it crashes, because bound can never be 0 and 1 at the same time.

I believe that everyone who correctly identified the bug in the previous question gave a correct explanation for this subpart.