Please do not turn the page until everyone is ready.

Rules:

- The exam is closed-book, closed-note, except for one side of one 8.5x11in piece of paper.
- **Please stop promptly at 3:20.**
- You can rip apart the pages, but please staple them back together before you leave.
- There are **70 points** total, distributed **unevenly** among 7 questions.
- When writing code, style matters, but don’t worry about indentation.

Advice:

- Read questions carefully. Understand a question before you start writing.
- **Write down thoughts and intermediate steps so you can get partial credit.**
- The questions are not necessarily in order of difficulty. **Skip around.**
- If you have questions, ask.
- Relax. You are here to learn.
1. (9 points) For each of the following, give a regular expression suitable for grep (or egrep) that matches the lines described:

(a) Lines containing two or more q characters
(b) Lines containing a word with two or more q characters (where a “word” is a consecutive sequence of English letters)
(c) Lines containing a q character that is not followed by a u character.

Solution:

(a) q.*q
(b) q[a-zA-Z]*q (full credit without capitals)
(c) (q[^u]) | (q$) (only .5 points off for omitting the end-of-line possibility)
2. (5 points) Explain precisely what this command does:

```bash
sed 's/\([0-9]\)\.(\([0-9]\)/\1,\2/g' foo.txt > bar.txt
```

**Solution:**
It make `bar.txt` be a copy of `foo.txt` except every period in `foo.txt` that is immediately preceded and immediately followed by a number is replaced with a comma.

(Full credit for the above answer, but...) Actually, this is not quite right; for example, `0.1.2` will become `0,1.2` since the 1 is matched for the first replacement and the `g` does not help with that. This was not the point of the question, though. Note `0.11.2` will become `0,11,2` as expected.
3. (13 points) Write a bash script that takes two filenames as arguments and deletes the file that has fewer words (deleting neither if they have the same number of words). You may assume without checking that the script is passed two arguments that are regular files and the filenames are sane (e.g., have no spaces in them). Hint: wc.

Solution:

#!/bin/bash

x='cat $1 | wc -w' # full credit for wc -w $1, but it does not quite work
y='cat $2 | wc -w' # full credit for wc -w $2, but it does not quite work
if [ $x -lt $y ]
then
  rm $1
fi
if [ $y -lt $x ]
then
  rm $2
fi
4. (12 points) Here are three similar program fragments, in Java, bash and C.

// Java
int[] arr = new int[3];
arr[0]=17;
arr[1]=17;
arr[2]=17;
for(int i=0; i < 5; ++i)
  System.out.println(Integer.toString(arr[i]));

// bash
arr[0]=17
arr[1]=17
arr[2]=17
i=0
while [ $i -lt 5 ]
do
  echo ${arr[$i]}
  (( i=$i+1 ))
done

// C
int arr[3] = {17,17,17};
int i=0;
for(; i < 5; ++i)
  printf("%d\n",arr[i]);

(a) What does the Java code do when run?
(b) What does the bash code do when run?
(c) What does the C code do when run?

Solution:

(a) It prints 17 three times and then throws an array-bounds exception.
(b) It prints 17 three times and then two blank lines.
(c) It prints 17 three times and then who knows; it might crash or more likely just prints whatever two “numbers” are adjacent to the array in memory.
5. (8 points) For each C function below, explain why it has a memory-management error. Explain what would/could go wrong when running the code.

(a) void f(int * p) {
    free(&p);
}

(b) int * g(int sz) {
    int * ans = (int*)malloc(sz*sizeof(int));
    int ok = h(sz,ans); // h a helper function, assume: int h(int,int*);
    if(ok)
        return ans;
    else
        return g(sz*2); // recur with bigger size
}

Solution:

(a) The call to free is with the address of an argument, i.e., a pointer into the stack, but free must always be called with a pointer into the heap.

(b) If h returns 0, this function has a space leak; the result of the malloc becomes unreachable and is never freed.
6. (16 points) Consider these C declarations:

```c
struct IntList {
    int value;
    struct IntList * next;
};
int * to_array(struct IntList * list);
```

Define the `to_array` function so that it returns a new heap-allocated array where the $i^{th}$ element is the $i^{th}$ element of the argument list:

- Assume the list ends with NULL. You may assume it has at least one element (though this is not that helpful).
- Do not deallocate any memory.
- Hint: Traverse the list twice. Sample solution 15 lines.

**Solution:**

```c
#include <stdlib.h>
int * to_array(struct IntList * list) {
    struct IntList * ptr = list;
    int len = 0;
    while(ptr != NULL) {
        ++len;
        ptr=ptr->next;
    }
    int * ans = (int*)malloc(len*sizeof(int));
    int i;
    for(i=0; i<len; ++i) {
        ans[i]=list->value;
        list=list->next;
    }
    return ans;
}
```
7. (7 points)
Consider this Makefile:

```
all: myprog

foo.c: foo.c foo.h bar.h
    gcc -c foo.c

bar.c: bar.c foo.h bar.h
    gcc -c bar.c

myprog: foo.o bar.o
    gcc -o myprog foo.o bar.o

clean:
    rm myprog *.o
```

(a) What is wrong with this Makefile?

(b) Do one of the following, making clear which you are answering. For either, be clear about what files already exist, etc and assume make does not have any special knowledge about compiling C.

i. Describe a likely situation where this Makefile would lead make not to recompile something that needs recompiling.

ii. Describe a likely situation where this Makefile would lead make to report that it does not know about some target.

**Solution:**

(a) The targets `foo.c` and `bar.c` should be `foo.o` and `bar.o`.

(b) There are multiple possible answers for each. For example:

i. If `foo.o` and `bar.o` exist, `myprog` does not, and some relevant C code has changed, the Makefile will just relink, not recompile.

ii. If `foo.o` does not exist, running `make` will lead to it needing to be made, but there is no target for it.