Name:_____

CSE 303, Winter 2006, Midterm Examination
8 February 2006

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 1:20.**
- You can rip apart the pages, but please write your name on each page.
- There are **80 points** total, distributed **unevenly** among 5 questions (all of which have multiple parts).
- 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.

Name:_____

1. (**13** points)

   (a) What does this bash command do? Give a simpler way (not using `cat`) to do the same thing.

   ```
   cat foo > bar
   ```

   (b) What does this bash command do? (Hint: It's probably *not* what the user/script-writer intended.)

   ```
   echo i > 3
   ```

   Solution:

   (a) It copies the contents of file `foo` to the file `bar` (both in the current directory).
   Simpler is `cp foo bar`.

   (b) It creates (or replaces) a file named `3` and has it hold the one character `i`.

Name:_____

2. (**15** 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 `a` characters.

   (b) Lines containing two `a` characters that have exactly 4 other characters between them.

   (c) Lines that start with a dollar-sign.

   (d) Lines containing any particular lower-case English letter three or more times (e.g., `abbaa` and `ababa` would match but `ababcc` would not).

   Solution:

   (a) `a.*a`

   (b) `a....a`

   (c) `^\$`

   (d) `([a-z]).*\1.*\1`

Name:_____

3. (**12** points) Explain the behavior of this bash script. Do not explain *how* it works, just what a user of the script would see. Note: It is not buggy; it does something semi-useful.

```
#!/bin/bash
ans=0
while [ `pwd` != "/" ]
do
  (( ans = ans + 1 ))
  cd ..
done
echo $ans
```

Solution:
It prints to standard out how many levels of subdirectories the current directory is below the topmost directory. For example, if the current directory is `/foo/bar/baz`, it prints 3.

Note the current directory of the shell calling this script does *not* change.

Name:_____

4. (**25** points) Consider this definition:

```
struct TwoPtrs {
  int * p1;
  int * p2;
};
```

(a) Write a function `makeIt` that takes two `int` arguments and returns a pointer to a new heap-allocated `struct TwoPtrs` whose fields are pointers to new heap-allocated `int`s, one for each argument.

(b) Write a function `areSame` that takes a `struct TwoPtrs` and returns:

- 0 if the fields point to locations holding different `int` values.
- 1 if the fields point to different locations holding the same `int` value.
- 2 if the fields point to the same location.

(c) Explain how a caller could use this function to produce an illegal ("may set the computer on fire") C program. Assume there are no dangling pointers before the function is called and the argument is actually a `struct TwoPtrs *`.

```
void free_TwoPtrs_and_exit(struct TwoPtrs * x) {
  free(x->p1);
  free(x->p2);
  free(x);
  exit(1); /* exit immediately; does not return */
}
```

Solution:

(a)
```
struct TwoPtrs * makeIt(int x, int y) {
  struct TwoPtrs * ans = (struct TwoPtrs *)malloc(sizeof(struct TwoPtrs));
  ans->p1 = (int*)malloc(sizeof(int));
  ans->p2 = (int*)malloc(sizeof(int));
  *(ans->p1) = x;
  *(ans->p2) = y;
  return ans;
}
```

(b)
```
int areSame(struct TwoPtrs x) {
  if(x.p1 == x.p2) return 2;
  if(*(x.p1)== *(x.p2)) return 1;
  return 0;
}
```

(c) If `x->p1==x->p2` (i.e., they are aliases; pointers to the same `int` location), then the second-line is a double-free.

Name:_____

5. (**15** points)

   (a) Consider this program

```
#ifdef DEBUG
#define DEBUG_PRINT(x) (printf("%d",(x)))
#else
#define DEBUG_PRINT(x) /* nothing */
#endif
int main(int argc, char** argv) {
  int x=0;
  DEBUG_PRINT(++x);
  DEBUG_PRINT(x++);
  return x;
}
```

   i. If the program is compiled with `DEBUG` defined, what does `main` print and what does it return.

   ii. If the program is compiled with `DEBUG` not defined, what does `main` print and what does it return.

   (b) Consider this program

```
void print_int(int x) { printf("%d",x); }
void ignore_int(int x) { }
void (*DEBUG_PRINT)(int);
int main(int argc, char** argv) {
#ifdef DEBUG
  DEBUG_PRINT = &print_int;
#else
  DEBUG_PRINT = &ignore_int;
#endif
  int x=0;
  DEBUG_PRINT(++x);
  DEBUG_PRINT(x++);
  return x;
}
```

   i. If the program is compiled with `DEBUG` defined, what does `main` print and what does it return.

   ii. If the program is compiled with `DEBUG` not defined, what does `main` print and what does it return.

Solution:

(a) i. prints: 11 returns: 2

ii. prints: (nothing) returns: 0

(b) i. prints: 11 returns: 2

ii. prints: (nothing) returns: 2