CSE 110 Midterm Exam

Mystery Instructor

21 July, 1992

Name ____

Penn ID Number ____

	Tracing:	30 pts	
1.1.		____________	10 pts
1.2.		____________	10 pts
1.3.		____________	10 pts
	Writing Code:	50 pts	
2.1.		____________	10 pts
2.2.		____________	20 pts
2.3.		____________	20 pts
	Debugging:	20 pts	
3.1.		____________	5 pts
3.2.		____________	5 pts
3.3.		____________	5 pts
3.4.		____________	5 pts
	Total	____________	100 pts

Welcome to the exam.

I will award partial credit for partially correct answers that show thought or insight. If you can't satisfy one of the requirements of a programming problem, ignore that part and work on writing a good program that satisfies the other requirements. It's much better to turn in a program that does a few things well than it is to turn in a program that does everything badly.

Make sure to make it clear whhich parts of the things you write are the ones I should grade. Be sure to label each answer with a number of the problem it's the answer to.

1 Tracing

1.1 10 Points

What does this program fragment print?

```
int name[]={5, 23, 119};
int *p, *q;

p = name;
q = name + 1;
printf("%d %d %d\n", *name, *p, *q);

*(p++);
(*q)++;
printf("%d %d\n", *p, q[0]);
```

What are the contents of the array `name` when this code is finished?

1.2 10 points

What does this program print?

```
#include <stdio.h>

int main(void)
{
  int t = 4;

  while (t-1) {
    printf("%d\n", t);
    if (t%3)
      t = 2*t + t%3;
    else
      t /= 3;
  }

  return 0;
}
```

1.3 10 points

What does this program print?

```
#include <stdio.h>

void scramble(int x, int *y);

int main(void)
{
  int a=5, b=23;

  printf("%d %d\n", a, b);
  scramble(b, &a);
  printf("%d %d\n", a, b);
  scramble(a, &b);
  printf("%d %d\n", a, b);

  return 0;
}

void scramble(int x, int *y)
{
  int temp;

  printf("%d %d\n", x, *y);
  temp = x + 1;
  x = *y;
  *y = temp;
  printf("%d %d\n", x, *y);
}
```

2 Writing Code

2.1 10 points

Suppose we want to compute the value of 3^4, which is $3 \cdot 3 \cdot 3 \cdot 3$, or 81. C, unlike some languages, has no operator for this, so we'll write a function. Write the function `pow`, which accepts an integer `n` and a non-negative integer `p`, and which computes and returns the value of `n` raised to the `p` power, or n^p. Have `pow` return a <`long int`> value, since the return values are likely to be large.

2.2 20 points

Write the function `strrev`, whose argument is a string, and which reverses its argument in place. That means that if we do:

```
char word[] = "Foo";
printf("%s\n", word);
strrev(word);
printf("%s\n", word);
```

the output should be `Foo`, followed by `ooF`.

2.3 20 points

Write a function `get_int` which accepts three arguments: `min` and `max`, which are <`int`>s, and `prompt`, which is a string. `get_int` should print the string `prompt` to prompt the user to enter numeric input, and try to read an <`int`> value from the user. The input must be between `min` and `max`, inclusive. `get_int` should repeatedly prompt and get input until the user enters an integer between `min` and `max`. `get_int` should handle non-numeric input gracefully. When the user finally enters a valid input, `get_int` should return that value as its return value.

3 Debugging

3.1 5 points

Suppose that `getline` is a function which reads input stream up to a newline character, somehow finds enough space in memory (an <array of `char`>) to store the line there, stores the line in the space, and returns a pointer to the first character in the space. Suppose also that `getline` somehow terminates the program if it reaches `EOF`.

Let's use `getline` to write a program which copies its input, a line at a time, to the screen:

```
char * getline(void);
int main(void)
{
  char *s;

  while(1) {
    s = getline();
    printf(s);
  }
}
```

This program, as written, has a very bad bug that will cause undefined behavior under certain circumstances. (Typically the program's behavior under these circumstances will be garbage output or immediate termination.)

Find the bug, explain the circumstances under which it occurs, and write correct code to replace the buggy code above.

3.2 5 points

Dafydd Painter, a world-famous nostrilologist, is writing C programs to help him with his work. In one of them he has the directive

```
#define NUMBER_OF_NOSTRILS 2
```

He explains: “This constant represents the number of nostrils possessed by the typical human being. It appears many places in my program. I represented the number of nostrils with a manifest constant so that if the value ever changes, it’ll be easy to change the code.”

Of course that’s silly, because we don’t expect the typical number of nostrils to change. Nevertheless, there is a good reason for `#define`’ing `NUMBER_OF_NOSTRILS`. What is it?

3.3 5 points

Farina Granville, noted quinquagintaseptologist, tried to write a function that would set the value of a variable to 57:

```
void set_to_57(int var)
{
  var = 57;
}
```

Of course, it didn’t work. You explained to her that in order for the function to change the value of a variable, it must know where that variable is, and so Farina will have to pass the address of the variable she wants to change into the function `set_to_57`.

Farina is obstinate: “I don’t want to pass in a pointer,” she says. “People shouldn’t have to know about pointers to use this simple function. I’ll have the function itself get the address once it’s called.” Then she writes this:

```
void set_to_57(int var)
{
  int *p;      /* Address of argument */

  p = &var;
  *p = 57;
}
```

Does this work? If so, how? If not, why not?

3.4 5 points

Kelvin R. Réaumur, a noted temperaturologist, wants to write a function which accepts a Celsius temperature as an argument and which returns the equivalent

Fahrenheit temperature. The formula for converting from degrees Celsius to degrees Fahrenheit is:

$$^\circ\mathrm{F} = {}^\circ\mathrm{C} \cdot \frac{9}{5} + 32$$

Here is what Dr. Réaumur tried:

```
double celsius_to_fahrenheit(double c)
{
  double f;

  f = c * (9/5) + 32;
  return f;
}
```

First, Dr. Réaumur tested the function by passing in 0, the Celsius freezing point of water, and it correctly returned 32, the Fahrenheit freezing point of water. Heartened, he then tried it on 37, which is normal human body temperature, but the return value was 69, instead of 98.6 as it ought to have been.

What is wrong with this function? Fix it.