CS16, UCSB
Pre-lab #2: Worth 50% of Lab 2 score (50 total points)

Print this form, staple loose pages together, and write your answers on it.

Accepted: On paper, in lab Wednesday, July 11 or in lecture Thursday, July 12.

Name (2 pts): _____

Email (2 pts): _____

Lab section (2 pts) Circle one: 2:00 3:30

If you have the book, read to the end of Chapter 2 in the Etter text. Otherwise consult the lecture slides. Then answer the following questions.

1. Consider the following code (from Etter 410):

```
int a = 27, b = 6;
float c;
...
c = a / (float)b;
```

a.(6 pts) What is the meaning and what is the effect of the `(float)b` part of the last statement above?

Treat “b” as a float, and force “b: into a floating-point representation. (Anything that says that “b” ends up in a floating point representation gets full credit.)

b.(2 pts) What would be the resulting value of c without the `(float)` part (i.e., just c = a / b)?

4.0 (4 is also acceptable for full credit)

c.(6 pts) Would the `(float)` part be necessary if the variables were declared as follows? Explain your answer.

```
int b = 6;
float a = 27, c;
```

No. Since “a” would be a float, floating point division would be used without the `float` part.

d.(2 pts) The term used above - “`(float)` part” - is obviously very non-technical, but there is a specific term to property call this type of operator. What is the proper term?

Cast.

2.If you have the textbook, read the section titled “Overflow and Underflow” on page 49. If not, you may have to use the Internet as a resource.

a.(6 pts) The book states that the actions generated by overflow and underflow are ultimately “system dependent”. What does “system dependent” mean here? (Hint: it is related to “undefined” behavior.)

The system is free to do whatever it wants to under these circumstances. (Any answer that says that the result cannot be relied on should also get full credit.)

b.(6 pts) What can you do to find out how your system responds to an overflow condition?

Force an overflow. There should also be a specific example to do this, i.e.:

```
unsigned char x = 255;
X++;
```

If they only said to force an overflow without any explanation of how to do this, 4 pts should be given instead of 6. If their explanation was something like “add two really big numbers together”, 5 pts should be given.

c.(6 pts) Describe a way you can test how your system responds to an underflow condition.

Force an underflow. Again, specific examples are good, as with:

```
unsigned char x = 0;
x--;
```

If they only said to force an underflow without any explanation of how to do this, 4 pts should be given instead of 6. If their explanation was something like "subtract a big number from a small number", 5 pts should be given.

3.If you have the text, refer to "Character I/O" on page 70. A summary of that section follows.

In addition to printf and scanf, C has two other functions for writing and reading from the terminal, namely putchar() and getchar(). putchar() writes a single character to the terminal, like so:

```
putchar( 'a' ); // prints 'a' to the terminal
```

getchar(), in contrast, reads a single character from the terminal, returning it. The return type is int, so it can return a special non-character value (namely EOF, or End Of File) if it could not read a character in. It is used like so:

```
int c = getchar(); // character read in is in variable "c"
putchar( c ); // prints out the same character read in
```

a.(4 pts) Write one proper C statement using scanf instead of getchar that will have exactly the same effect (on the variable "c") as the following statement:

```
int c = getchar();
```

(answer):

```
int c;
scanf( "%c", &c );
```

Still award full credit if they don't redeclare c.

2 points: format string is equivalent to "%c"
1 point: one extra actual parameter "c"
1 point: address-of operator was used on "c"

b.(4 pts) Write one proper C statement using printf instead of putchar that will produce exactly the same

output as the following two statements (referencing the same variable “c” from before):

```
putchar( c );
putchar( ‘\n’ );
```

(answer):

```
printf( “%c\n”, c );
```

1 point: “%c” is used as the placeholder
1 point: “\n” was used in the statement
1 point: “c” was used as an extra actual parameter
1 point: The format string is exactly “%c\n”

c.(2 pts) Imagine that the user entered 0 (zero) in part a above, and suppose your printf statement in part b used “%i” instead of “%c” to print this value. What would be the output in this case? (Hint: not 0 - see ASCII codes.)

Pre-lab End. Adapted from Michael Costanzo by Kyle Dewey.