Name: Meg Ryan NetID: mryan47

Understanding the difference between the unary * and unary & operators is a truly important skill in C. The * operator has many meanings. It mean multiplication when used as a binary operator (7 * 8), but does "address dereferencing" when used as unary operator. When used in a variable/parameter declaration statement, it means the variable/parameter will store an address location.

Consider the following code (from lab1a.c):

```
int x = 5;
int *y;

y = &x;

printf ("Print line 1: %d\n", x);
printf ("Print line 2: %p\n", &x);
printf ("Print line 3: %p\n", y);
printf ("Print line 4: %p\n", &y);
printf ("Print line 5: %d\n", *y);
```

1a. What type of variable is x? What value is stored in x?

x is an integer which stores the value 5.

1b. What type of variable is y? What value is stored in y?

y is a pointer to an integer which stores the address of x.

2. Will all the above printf( ) statements always display the same results? Why or why not? (I.E. If I ran the above code multiple times on the same or different machines, will I always get the same output?)

No, the variables being printed reference different locations in memory where they are being stored. The addresses of these locations will vary accross machines and executions (on the same machine) depending on the memory makeup of the machines at the times of execution.

Name: Meg Ryan NetID: mryan47

3. What would happen if you could to include the following line to the above program? Why?

```
printf ("Print line 6: %d\n", *x);
```

We receive the following error:
error: indirection requires pointer operand ('int' invalid)

This is because the print statement is expecting an int value to be passed (due to the "%d"), and we are instead passing an int *.

In C, we use & and * to simulate pass-by-reference parameters. A pass-by-reference parameter that is changed in a function will reflect that modification at the calling code. Assume we have the function:

```
void funct1 (int p1, int *p2)
{
  printf ("Print line 101: %d\n", p1);
  printf ("Print line 102: %p\n", &p1);
  printf ("Print line 103: %p\n", p2);
  printf ("Print line 104: %p\n", &p2);
  printf ("Print line 105: %d\n", *p2);

  p1 = p1 + 5;
  *p2 = *p2 + 5;
}
```

Which is called by the following code in main:

```
int a = 2;
int b = 4;
printf ("Print line 11: For variable a: value: %d, address:  %p\n", a, &a);
printf ("Print line 12: For variable b: value: %d, address:  %p\n", b, &b);
funct1 (a, &b);
printf ("Print line 13: For variable a: value: %d, address:  %p\n", a, &a);
printf ("Print line 14: For variable b: value: %d, address:  %p\n", b, &b);
```

4. Do the **values** associated with variables a and b change because of the call to funct1( )? Why?

The value associated with variable b changes, while the value of variable a does not.
This is because of the use of a reference parameter, "&b". The value of b can then be returned from the function after it has been manipulated by "*p2 = *p2 + 5;".

5. Does the **address** information from Print line 11 through 14 match to any information printed in Print line 101 through 105? If so, describe all the **address** information that matches between the two sets of Print line statements.

The following address' match:
Print line 12: For variable b: value: 4, address:  0x7fff53dc7928
Print line 103: 0x7fff53dc7928
Print line 14: For variable b: value: 9, address:  0x7fff53dc7928

Name: Meg Ryan NetID: mryan47

Access the program lab1b.c from the Lab Exercises page. This program is to read in a list of integers until the value of -999 is input. It then is to print out the number of value read, the total of these values and the average of these values. However, this program does contain some errors.

6. Fill in the table below with the expected values for the variables **count**, **total** and **average** if the program is run with input of:

2 4 6 8 -999

Compile and run the program. Fill in the table below with the actual values produced by the program?

| | Expected Value | Actual Value |
|---|---|---|
| Value of variable: **count** | 4 | 1427091460 |
| Value of variable: **total** | 20 | 32787 |
| Value of variable: **average** | 5.00000 | 0.00000 |

Does the program produce the expected output? If not, you may need to fix the error before moving to the next question.

No, it does not.

7. Fill in the table below with the expected values for the variables **count**, **total** and **average** if the program is run with input of:

2 3 5 -999

Compile and run the program. Fill in the table below with the actual values produced by the program?

| | Expected Value | Actual Value |
|---|---|---|
| Value of variable: **count** | 3 | -824134957 |
| Value of variable: **total** | 10 | 32777 |
| Value of variable: **average** | 3.00000 | 0.00000 |

Does the program produce the expected output? If not, you may need to fix the error before moving to the next question.

No, it does not.

Name: Meg Ryan NetID: mryan47

8. Fill in the table below with the expected values for the variables **count**, **total** and **average** if the program is run with input of:

-999

Compile and run the program. Fill in the table below with the actual values produced by the program?

| | Expected Value | Actual Value |
|---|---|---|
| Value of variable: **count** | 0 | -338162368 |
| Value of variable: **total** | 0 | 32767 |
| Value of variable: **average** | 0 | 0.00000 |

Does the program produce the expected output?

No

9. What errors contained in the program were exposed in the above test cases?

The variables (total, val, count) were not initialized before being used in computations.

10. Give a definition that would define when we can call a program "error free".

The program compiles without error or warning messages.
In addition, the program is robust enough to successfully handles various inputs, including invalid types and values.