My Answers to the Sample Midterm

These are sample questions that are very similar to the ones I will ask on the midterm.

1. Which of the following is *not* a valid C variable name?

   (a) hello
   (b) chomp_burp
   (c) whilex
   (d) more-more-more
   (e) TrUcKiNg

   Answer: (d). The "-" is a subtraction operator in this context.

2. Evaluate the following expressions, assuming `a = 1`, `b = -3`, and `c = 0`. Treat them independently, so (for example) after evaluating (b), the above values are used for (c).

   (a) `a + b < c`
   (b) `c == !a == c`
   (c) `a = ++b`; give the values of both `a` and `b` as well
   (d) `a-- == !b`; give the value of `a` as well
   (e) `a / b`

   Answer:

   (a) `1`; this is `(a + b) < c`, or `(1 + (-3)) < 0`, `-2 < 0`, which is true (`1`).
   (b) `0`; this is `(c == (!a)) == c`, or (0 == (!1)) == 0, or `(0 == 0) == 0`, or `1 == 0`, which is false (`0`).
   (c) `-2`; this is `a = (++b)` and the value of `b` is incremented before use, so `b` becomes `-2` and `a` also is assigned `-2`.
   (d) `0`; this is `(a--) == (!b)` or `(1) == (!(-3))` (as we use the value of `a` before it is decremented), so `1 == 0`, which is false (`0`). The value of `a` after this expression is `0`.
   (e) `0`; as both `a` and `b` are integers, the result is the integer part of `1/(-3)`, which is `0`.

3. True or False: If `x = -1`, then `if (x) printf("1"); else printf("2");` prints `2`.

   Answer: False. As `-1` is not 0, the expression in the **`if`** statement is true, so this prints `1`.

4. What are all possible outputs of the following code fragment?

```
void f(int a, int b)
{
    printf("%d %d\ n", a, b);
}

void main(void)
{
    int i = 5;
    f(++i, ++i);
}
```

   Answer: The key point is that the function arguments can be evaluated in any order. So, the function can be called as `f(6, 7)` or `f(7, 6)`. So, the two possible outputs are:

```
6 7
```

   and

```
7 6
```

5. Given the definitions

```
int numbs[10];
int *ptr = numbs;
```

which of the following are equivalent, and why?

(a) `numbs[3]`
(b) `numbs + 3`
(c) `*(numbs + 3)`
(d) `*(ptr + 3)`
(e) `*ptr + 3`

Answer: (a) refers to the third element of the array numbs. (b) refers to the address of the third element of the array. (c) refers to the quantity at the address of the third element of the array, which is the third element of the array. As `ptr` is assigned `numb`, (d) refers to the same thing as (c). (e) is the value of the element stored at `numb`, plus 3. Hence (a), (c), and (d) are equivalent.

6. Use the following code fragment to answer parts (a), (b), and (c):

```
for(x = i = 0; i <= 100; i += 2, x += i);
```

(a) In one short sentence, what does this for loop do?
Answer: It stores in `x` the sum of the even numbers from 0 to 102 inclusive.

(b) Is the following **`while`** loop equivalent? If not, how does its result differ? (*Hint:* : look at the values of both `x` and `i`.)

```
x = i = 0;
while( i++ <= 100)
    x += ++i;
```

Answer: No. The loop places in `x` the sum of the even numbers from 0 to 102, because when `i is 100`, `i++ <= 100` is true (remember, the value of `i` is used before the "++" operator increments `i`). So `x` is the same. But `i` is 103, because the loop exits before `i` can be incremented again.

(c) Does the following **`for`** loop do the same thing? If not, what does it do?

```
for(x = i = 0; i <= 100; i++){
    if (!(i % 2))
        continue;
    x = x + i;
}
```

Answer: No. This sums the odd integers from 0 to 100 inclusive and stores the value in `x`.

7. What does this function do?

```
char *x(char *s, char c)
{
    char *r = NULL;

    do{
        while(*s && *s != c)
            s++;
        if (*s)
            r = s;
    } while(*s++);
    return(r);
}
```

Answer: It returns a pointer to the last occurrence in argument `s` of the character in argument `c`. If the character does not occur in that string, it returns the **`NULL`** pointer.

8. The following segment of code is supposed to print the number of times the routine `a_again` is called. Yet, regardless of the input, it prints 0. Why? How would you fix it?

```
void a_again(int acount)
{
    ++acount;
}

int main(void)
{
    int c;
    int counter = 0;

    while((c = getchar()) != EOF)
        if (c == 'a' || c == 'A')
            a_again(counter);

    printf("%d\n", counter);
    return(0);
}
```

Answer: The problem is that `acount` is passed as a parameter to `a_again`. As C calls by value, not reference, the value of `counter` is not changed by `a_again`. One way to fix this is to pass the counter as a pointer, and have `a_again` increment what that points to:

```
void a_again(int *acount)
{
    ++*acount;
}

int main(void)
{
    int c;
    int counter = 0;

    while((c = getchar()) != EOF)
        if (c == 'a' || c == 'A')
            a_again(&counter);

    printf("%d\n", counter);
    return(0);
}
```