1. **Describe the condition** of a while loop. (“The condition is a...”)

2. Are the properties of the condition of a while loop the same as, or different from, the properties of the condition of an if block?

3. **What are the steps** that describe the execution of a while loop?
   
   (a)

   (b)

   (c)

4. **How** does a while loop differ from an if block?
5. For each of these kinds of statements, mark **CAN** if it can appear in the body of a **while** loop, and mark **CANNOT** if it cannot appear in the body of a **while** loop. **EXPLAIN**.

(a) A named constant declaration

(b) A variable declaration

(c) A `printf` statement

(d) A `scanf` statement

(e) An assignment statement

(f) A `exit` statement

(g) An `if` block

(h) A **while** loop
6. **TRACE** the example program on slides 23 - 25 of the lecture packet titled “while Loop Lesson,” using the input values shown on slides 26 - 27. Your trace should show the following variables: `users_number`, `users_distance`, `users_last_distance` and `correct_number_hasnt_been_input`, but in the trace you can abbreviate their names as `un`, `ud`, `uld` and `cnhbi`, respectively.
7. **DRAW A FLOWCHART** for the Infinite Loop program on slide 15 of the lecture slide packet titled “*while Loop Lesson.*”
8. What are the **FIVE STEPS** that describe the execution of a *for* loop?

(a)

(b)

(c)

(d)

(e)

9. For each of these kinds of statements, mark **CAN** if it can appear in the body of a *for* loop, and mark **CANNOT** if it cannot appear in the body of a *for* loop. **EXPLAIN**.

(a) A named constant declaration

(b) A variable declaration

(c) A *printf* statement

(d) A *scanf* statement

(e) An assignment statement

(f) A *exit* statement

(g) An *if* block

(h) A *while* loop

(i) A *for* loop
10. Convert the following while loop into a for loop.

```c
count = initial_value;
while (count <= final_value) {
    printf("count = %d\n", count);
    count += stride;
} /* while (count <= final_value) */
```

11. Convert the following for loop into a while loop.

```c
for(count = 1; count <= n; count++) {
    n_factorial *= count;
} /* for count */
12. What is the **OUTPUT** of each of these programs? If you aren’t confident of an answer, type in, compile and run the program to test it.

(a) #include <stdio.h>
int main ()
{ /* main */
    int count;
    int sum;

    sum = 0;
    count = 1;
    while (count <= 10) {
        sum = sum + count;
        count = count + 1;
    } /* while (count <= 10) */
    printf("sum = %d\n", sum);
    return 0;
} /* main */

(b) #include <stdio.h>
int main ()
{ /* main */
    int count;
    int sum;

    sum = 0;
    count = 1;
    while (count <= 10) {
        sum = sum + count;
        count = count + 2;
    } /* while (count <= 10) */
    printf("sum = %d\n", sum);
    return 0;
} /* main */

(c) #include <stdio.h>
int main ()
{ /* main */
    int count;
    int product;

    product = 1;
    count = 1;
    while (count <= 15) {
        product = product * count;
        count = count + 5;
    } /* while (count <= 15) */
    printf("product = %d\n", product);
    return 0;
} /* main */
(d) #include <stdio.h>
int main ()
{ /* main */
    int count;
    int product;

    product = 1;
    count = 1;
    while (count <= 16) {
        product = product * count;
        count = count + 5;
    } /* while (count <= 16) */
    printf("product = %d\n", product);
    return 0;
} /* main */

(e) #include <stdio.h>
int main ()
{ /* main */
    const int lower_bound = 1;
    const int upper_bound = 17;
    const int stride = 5;
    int count;
    int product;

    product = 1;
    count = lower_bound;
    while (count <= upper_bound) {
        product = product * count;
        count = count + stride;
    } /* while (count <= upper_bound) */
    printf("product = %d\n", product);
    return 0;
} /* main */