Chapter 5 Review

1. With an *entry-condition* loop, the test expression is evaluated before each loop cycle. With an *exit-condition* loop, the body of the loop is executed first and then the test expression is evaluated afterwards (i.e., after each loop cycle).

2. `01234`

3.
```
0369
12
```

4.
```
6
8
```

5. k = 8

6.
```
for (int i = 1; i <=64; i *= 2)
   std::cout << i << " " ;
```

7. By inserting a compound statement (or block) using a pair of braces. For example:

```
for (int i = 0; i < 4; i++)
{
     std::cout << "The first value is: ";
     std::cout << i << std::endl;
}
```

8. The statement

```
int x = (1,024);
```

is valid and assigns the value of `024` to the variable `x`. The value `1` is ignored. This is because the comma operator has lower precedence than parentheses. The leading `0` tells the compiler that `024` is in base 8 form, which the compiler converts to decimal form as the value `20`.

The statements

```
int y;
y = 1,024;
```

are also valid, but these statements assign the value of `1` to the variable `x.` This time, the value `024` is ignored. This is because the comma operator has lower precedence than the equal sign.

9. When reading type `char` values, `cin >> ch` skips over spaces and newline characters. `cin.get(ch)` reads spaces and newlines and places the value in the `ch` argument. The function return value is an `istream` object. `ch = cin.get()` uses the function return value to assign the input character to `ch`.