Solution to Problem 3.28

Notational Preliminary:

$$Define\ S(k) = \begin{cases} 0\ if\ k = 0 \\ 1 + 2 + \cdots + k\ \ if\ k > 0 \end{cases}$$

The loop invariant developed in class (`sum = S(n) – S(count)`) is indeed a correct loop invariant that gives us the desired answer. I made a *stupid* error in coming up with a weakest pre-condition for the statement **count = count-1**. (See below for details.) The following corrects the error. Once I did this, the rest followed quite easily.

Let's see the step-by-step development. I will show assertions in red to make things clearer.

```
{n > 0}
count = n;
sum = 0;

while count <> 0 do
  sum = sum + count;
  count = count - 1;
end
{sum = 1 + 2 + ... + n}
```

The final condition is equivalent to `sum = S(n) – S(0)`, which is in turn equivalent to `sum = S(n) – S(count) AND count = 0.`
This suggests that the assertion `sum = S(n) – S(count)` could be a reasonable loop invariant.

Read the three post-conditions below from bottom to top to see how the tentative loop invariant was developed.

```
{n > 0}
count = n;
sum = 0;

while count <> 0 do
  sum = sum + count;
  count = count - 1;
end
{sum = S(n) – S(count) AND count = 0}
{sum = S(n) = S(n) – 0 = S(n) – S(0)}
{sum = 1 + 2 + ... + n}
```

The loop invariant must be true after the last assignment statement. The *weakest precondition* for the assignment statement is obtained by replacing **count** in the post-condition by **count – 1**. (Whatever is true about **count** after the assignment must be true about **count-1** before.)

Filling this in, we get:

```
{n > 0}
count = n;
sum = 0;

while count <> 0 do
  sum = sum + count;
  {sum = S(n) – S(count-1)}
  count = count - 1;
  {sum = S(n) – S(count)
end
{sum = S(n) – S(count) AND count = 0}
{sum = S(n) – S(0)}
{sum = 1 + 2 + ... + n}
```

Now, for previous assignment (sum = sum + count) we can once again compute the weakest precondition by replacing **sum** in the post-condition by **sum+count**. We can then work backwards through some mathematically equivalent assertions, until we arrive at out loop invariant (shown in bold at the beginning and end of the loop.

```
{n > 0}
count = n;
sum = 0;

while count <> 0 do
  {sum = S(n) – S(count)}
  {sum = S(n) – (S(count-1) + count)}
  {sum = S(n) – S(count-1) – count}
  {sum + count = S(n) – S(count-1)}
  sum = sum + count;
  {sum = S(n) – S(count-1)}
  count = count - 1;
  {sum = S(n) – S(count)
end
{sum = S(n) – S(count) AND count = 0}
{sum = S(n) – S(0)}
{sum = 1 + 2 + ... + n}
```

Since the assertion **sum = S(n) – S(count)** is clearly true after the two initial assignments (since **count = n** and **sum = 0**), the desired post-condition for the loop is verified.

Technically, we should use weakest preconditions here too, as follows:

```
{0 = S(n) – S(n)}
count = n;
{0 = S(n) – S(count) }
sum = 0;
{sum = S(n) – S(count)}
```

Since the first condition is obviously always true, this establishes the loop invariant as a precondition for the loop.

NOTE: The basic problem was that I was trying to replace **count** in the post-condition by the expression **count+1** in the pre-condition (or **count-1** by **count)**, instead of the expression **count-1** from the right-hand side of the assignment statement. Once I made the correct replacement, as you can see, the correctness of the loop invariant followed easily.