To implement the function $f(x) = \log(1 + \exp(x))$ we will need to address two issues:

1. The exponential function will cause overflow when x is large. To prevent this, we will use the following property:

   $$\log(1 + \exp(x)) = \log(\exp(x)(\exp(-x) + 1)) = x + \log(1 + \exp(-x)).$$

   This reformulation is only necessary when $\exp(x)$ is large.
2. The function $\log(x)$ has a large condition number when x is close to 1, and hence $\log(1 + \exp(x))$ is sensitive to rounding errors when $\exp(x)$ is small. To address this issue, we will use the `log1p()` function defined in `math.h` instead of `log()`. This function is designed to evaluate $g(x) = \log(1 + x)$ reliably, and hence $f(x) = g(\exp(x)) = x + g(\exp(-x))$.

Combining the above observations, we arrive at the following implementation:

```c
double softmax(double x) {
    if (x > 0)
        return x + log1p(exp(-x));
    else
        return log1p(exp(x));
}
```

Equivalently, using a ternary operator:

```c
double softmax(double x) {
    return x > 0 ? x + log1p(exp(-x)) : log1p(exp(x));
}
```