Assignment 2

Numerical Computing

Rudolf C. Kischer

260956107

Floating point in C, Overflow and Underflow, numerical cancellation

---

Q1.

- a) Write a program according to this pseudocode to compute X(0.1,200,2000) . Notice the true value of X(0.1,200,2000) is approximately equal to 0.03 . Comment on the difficulties you encounter.

```
float X(float p, float k, float N) {
    float x = 1.0f;
    for (int j = 1 ; j <= k ; j++) {
        x *= (N + 1-j);
    }

    for (int j = 1 ; j <= k ; j++) {
        x /= j;
    }

    for (int j = 1 ; j <= k ; j++) {
        x *= p;
    }

    float q = 1 - p;

    for (int j = (k+1) ; j <= N ; j++) {
        x *= q;
    }

    return x;
}
```

Computed Result Input:

```
int main() {

    // Question a)
    std::cout << "ASGN-1" << std::endl;
    printf("Part A\n");
    float x = X(0.1f,200.0f,2000.0f);
    printf("X(0.1,200,2000) =  %.12f\n", x);
}
```

Output:

```
ASGN-1
Part A
X(0.1,200,2000) =  inf
```

Explanation: - The trouble with this code and these inputs , is that at certain parts of the computation, our intermediate result exceeds the largest number that can be represented by a float. This is called an overflow. - When this happens, the result is set to infinity. This is what we see in the output. - This is interesting because our final result is representable by a float, because as a the calculation continues the result is divided by a large number, which brings it back down to a representable number. - Most operations that include infinity, will result in infinity. This means that there is no way to recover from this error.

- Note: If we performed our operations in a different order we may get an underflow, Nan, or a 0, which is might also not be recoverable.
- b) **Question:** Modify your program and try it on the problem in 1(a).The final result for X(p,k,n) should be a floating point number (in decimal exponent format)

Answer:

This code is responsible for growing or shrinking the number to keep it in the range of representable numbers. We also keep track of how often this is done with the b_count variable.

```
float numerical_upkeep(float x, float b, int* b_count) {
    float s = 1.0f / b;
    if (x >= b) {
        (*b_count)++;
        x *= s;
    }
    if (x <= s) {
        (*b_count)--;
        x *= b;
    }
    return x;
}
```

Here we have modified the original code so that we perform the house keeping on the number after each operation. This ensures that the number is always in the range of representable numbers. At the end we multiply the number by `b` to get the final result.

```
float X_fixed(float p, float k, float N) {

   float x = 1.0f;
   int b_count = 0;
   float b = pow(2, 100);
   float s = 1.0f / b;

   //max: 2 ^ 117
   //min : 2 ^ -115
   for (int j = 1 ; j <= k ; j++) {
       x *= (N + 1-j);
       x = numerical_upkeep(x, b, &b_count);
   }

   for (int j = 1 ; j <= k ; j++) {
       x /= j;
       x = numerical_upkeep(x, b, &b_count);
   }

   for (int j = 1 ; j <= k ; j++) {
       x *= p;
       x = numerical_upkeep(x, b, &b_count);
   }

   float q = 1.0f - p;

   for (int j = (k+1) ; j <= N ; j++) {
       x *= q;
       x = numerical_upkeep(x, b, &b_count);
   }

   printf("b_count: %d\n", b_count);
   for (int i = 0; i < b_count; i++) {
       x *= b;
   }
   return x;

}
```

Input:

```
int main() {

   // Question b)
   printf("Part B\n");
   float x_fixed = X_fixed(0.1f,200.0f,2000.0f);
   printf("X_fixed(0.1,200,2000) =  %.8e\n", x_fixed);

   return 0;
}
```

Output:

```
Part B
b_count: 0
X_fixed(0.1,200,2000) =  2.97215302e-02
```

Explanation: Here we can see that the numerical answer is no longer an `inf` calue and is within the range of the expected answer which was `0.03`