ECE 4960

Spring 2017

Lecture 3

Exception Handling: NaN and INF

Edwin C. Kan
School of Electrical and Computer Engineering
Cornell University

NaN

- When your programming environment does not know how to calculate a floating-point operation, instead of halting or core dump, an NaN (not a number) is assigned.
- NaN in double precision: the exponential field will be: $e_{\max} + 1$ = 2047 $(11111111111)_2$, while the mantissa is NOT zero and the signed bit is arbitrary.

Operation	NaN produced by
+, –	INF – INF, INF + NINF, etc.
×	0 × INF
/	0/0, INF/INF, INF/NINF, etc.
% or REM	x%0, INF%y, etc.
sqrt(x)	$x < 0$

NaN Exception Handling

- When NaN is encountered, a global environmental variable such as SIGFPE will be “set” or “thrown”
- In C++, you can also test with std::fetestexcept
- Most operations with NaN will generate NaN (time to abandon that branch of computation)

Exceptions in Integers

- Unlike floating points, integers are EXACT and have no precision errors.
- Round-off during type conversion can be platform dependent.

```
long i = 5/3; return (i);
```

- Most C/C++ and Python will return 1 (truncation), while some Matlab will return 2 (0.5 roundoff rules).
- 32-bit integers can easily overflow, such as 13!

Overflowing: Integers

- For 32-bit long integers, with the signed bit, the integer range is between -2^{31} to 2^{31}.
- If you are anticipating large numbers, either tolerate the limited range, or you would have to concatenate multiple integers to one, or you will just perform binaries directly.
- The standard practice in cryptography involves factorization of integers represented by more than 4,000 bits.

Group Discussion

- What do you think is the best strategy to implement 1/0 in integer operations?

1) Give the largest integer;
2) Reserve a symbol in integers and signal the exception;
3) Change to floating points INF and signal the exception;
4) Do not regulate it.

Overflowing: Floating Point

- Overflow exception cannot be handled by DBL_MAX
  Ex: Computing $\sqrt{x^2 - y^2}$
  where (double) $x = 5\times10^{160}$ and $y = 4\times10^{160}$.

- The best answer will be 3×10^{160}, but during the computation x^2 and y^2 will overflow. If x^2 and y^2 are replaced with the largest number (DBL_MAX), we obtain a false impression of 0.0!!!!
- If we use INF x^2 and y^2, then the answer is a warning of NaN!
- INF rules: 1/INF = 0; 0/INF =0; 1/0 = INF; –1/0 = NINF
- INF rules: INF/INF = NaN; 0/0 =NaN; 1/0 = INF; -1/0 = NINF
- You can do better if you have more “existing knowledge” (say using L’Hospital rules in the symbolic domain)…

Overflow Exception Can Be Problematic

Computing the function $f(x) = x/(x^2 + 1)$.

For $x = 2\times10^{154}$, f(x) will be evaluated to 0 (it should be 5×10^{-155})

Compute the equivalent function: $1/(x + x^{-1})$.

This expression will not overflow prematurely for large x!

Because of infinity arithmeti, we will have the correct value even when $x = 0$ or INF: $1/(x + x^{-1}) = 1/(0 + \mathrm{INF}) = 1/\mathrm{INF} = 0$.

Overflow Exception Still Useful

- Without infinity arithmetic, the expression $1/(x + x^{-1})$ requires a test for $x = 0$, which not only adds extra instructions, but may also disrupt a pipeline in parallel computing.

- INF arithmetic often avoids the need for special case checking; however, formulas need to be carefully inspected to make sure they do not have spurious behavior

Hacker Practice

- Observe the exception handling on your platform:

```
// Generating NaN and INF in double
//
double x = 0.0; y = 0.0; doubleResult1; doubleResult2;
doubleResult1 = 1/x;  doubleResult2 = y/x;
print(doubleResult1, doubleResult2);

// Observe NaN and INF handling in integers
//
long m = 0; n = 0; intResult1; intResult2;
intResult1 = 1/m;  intResult2 = m/n;
print(intResult1, intResult2);

// Observe overflow handling in integers
//
long i = 1; intFactorial = 1;
for (i= 2; i < 30; i++) {
intFactorial *= i;
print(i, intFactorial)
}
```