UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical & Computer Engineering

ECE 150 *Fundamentals of Programming*

Catalan numbers

Douglas Wilhelm Harder, M.Math. LEL
Prof. Hiren Patel, Ph.D., P.Eng.
Prof. Werner Dietl, Ph.D.

Outline

- In this lesson, we will:
  - Look at the definition of the Catalan numbers and their application
  - Consider the various formulas for these numbers
  - Implement these formulas in C++
  - Observe that:
    - Some formulas are incredibly (exponentially) slow
    - Others are just slow
    - Most result in overflow
    - The most efficient algorithm is the last formula listed

Catalan numbers

- The n^{th} Catalan number, C_n, equals:
  - The number of different binary trees with n nodes
    - Useful in specific tree algorithms
  - The number of ways that a convex polygon with $n + 2$ sides can be subdivided into triangles
    - Useful in tessellations and graphics
  - The number of ways that n pairs of parentheses can appear in such a way that they are properly nested

    ()() or (()) ()()(), (())(), (()()), ()(()), or ((()))
    - Useful for compilers and programming languages

Catalan numbers

- There are numerous ways of defining Catalan numbers:

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{n!(n+1)!} = \prod_{k=2}^{n}\frac{n+k}{k}$$

  - However, all of these simplify to an integer

- There are recursive definitions, as well:

$$C_0 = 1 \qquad C_0 = 1$$

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-k-1} \qquad C_n = \frac{2(2n-1)}{n+1} C_{n-1}$$

  - This is approximately the order they are listed on the Wikipedia page as of March, 2022

Catalan numbers

- Here are C_0 through C_{36}:

1, 1, 2, 5,
14, 42, 132,
429, 1430, 4862,
16796, 58786, 208012,
742900, 2674440, 9694845,
35357670, 129644790, 477638700,
1767263190, 6564120420, 24466267020,
91482563640, 343059613650, 1289904147324,
4861946401452, 18367353072152, 69533550916004,
263747951750360, 1002242216651368, 3814986502092304,
1454463603922690, 5553406487704819, 21233613041224311,
81294404214973076, 311628549490730126, 1195979838586045349

  - Note: $C_{37} \geq 2^{64}$ and thus cannot be represented as an `unsigned long`

Catalan numbers

- A reminder:
  - In most compilers, `long` is 64 bits
  - In some compilers, `long` is (like `int`) only 32 bits, and you must use `long long` for a 64 bit integer

  - You can try one of these:

```
assert( sizeof( long ) == 8 );

int main() {
    std::cout << "int:       " << sizeof ( int ) << std::endl;
    std::cout << "long:      " << sizeof ( int ) << std::endl;
    std::cout << "long long: " << sizeof ( long long )
                               << std::endl;

    return 0;
}
```

Binomial coefficients

- This is a fast and accurate implementation of binomial coefficients:

```
unsigned long binomial( unsigned long n, unsigned long k ) {
  if ( k > n ) {
    return 0;
  } else if ( (1 >= k) || ((k + 1) >= n) ) {
    return ( (k == 0) || (k == n) ) ? 1 : n;
  } else {
    if ( k > (n - k) ) {
      k = n - k;
    }

    if ( k > 33 ) {
      throw std::range_error{ "The result is not representable by 'unsigned long'" };
    }

    unsigned long pascal[k - 1];

    for ( unsigned long i{ 0 }; i < k - 1; ++i ) {
      pascal[i] = 1;
    }

    for ( unsigned long m{ 1 }; m <= (n - k); ++m ) {
      pascal[0] += m + 1;

      for ( unsigned long i{ 1 }; i < k - 1; ++i ) {
        unsigned long previous{ pascal[i] };

        pascal[i] += pascal[i - 1];

        if ( pascal[i] < previous ) {
          throw std::range_error{ "The result is not representable by 'unsigned long'" ;
        }
      }
    }

    return pascal[k - 2];
  }
}
```

Implementations

- Let's implement all of these functions:

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

```
unsigned long C( unsigned long n ) {
    return binomial( 2*n, n )/(n + 1);
}
```

Implementations

- Here is a difference of two binomial coefficients:

$$C_n = \binom{2n}{n} - \binom{2n}{n+1}$$

```
unsigned long C( unsigned long n ) {
    return binomial( 2*n, n ) - binomial( 2*n, n + 1 );
}
```

$$\binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{n!n!} - \frac{(2n)!}{(n-1)!(n+1)!}$$

$$= \frac{(2n)!}{n!n!} - \frac{n(2n)!}{n!(n+1)n!}$$

$$= \frac{(2n)!}{n!n!}\left(1 - \frac{n}{n+1}\right)$$

$$= \frac{(2n)!}{n!n!}\frac{1}{n+1} = \frac{1}{n+1}\binom{2n}{n}$$

Implementations

- This is simply a ratio of factorials:

$$C_n = \frac{(2n)!}{n!(n+1)!}$$

```
unsigned long C( unsigned long n ) {
    return factorial( 2*n )/factorial( n )
                           /factorial( n + 1 );
}
```

$$\frac{1}{n+1}\binom{2n}{n} = \frac{1}{n+1}\frac{(2n)!}{n!n!}$$

$$= \frac{(2n)!}{(n+1)!n!}$$

Implementations

- Here we have a product of $n - 1$ rational numbers:

$$C_n = \prod_{k=2}^{n} \frac{n+k}{k}$$

```
unsigned long C( unsigned long n ) {
    unsigned long result{ 1 };

    for ( unsigned long k{ 2 }; k <= n; ++k ) {
        result *= (n + k)/k;
    }

    return result;
}
```

  - Why can this not work?

Implementations

- Here we have a product of $n-1$ rational numbers:

$$C_n = \prod_{k=2}^{n} \frac{n+k}{k}$$

```
unsigned long C( unsigned long n ) {
    unsigned long numer{ 1 };
    unsigned long denom{ 1 };

    for ( unsigned long k{ 2 }; k <= n; ++k ) {
        numer *= n + k;
        denom *= k;
    }

    return numer/denom;
}
```

$$\frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$$

$$= \frac{(2n)(2n-1)\cdots(n+2)!}{n!}$$

$$= \frac{(2n)(2n-1)\cdots(n+2)!}{n(2n-1)\cdots 2\cdot 1}$$

$$= \frac{n+n}{n}\frac{n+(n-1)}{n-1}\frac{n+(n-2)}{n-2}\cdots\frac{n+2}{2}$$

Implementations

- Here is the first recursive formula:

$$C_0 = 1$$

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-k-1}$$

```
unsigned long C( unsigned long n ) {
    if ( n == 0 ) {
        return 1;
    }

    unsigned long sum{ 0 };

    for ( unsigned long k{ 0 }; k < n; ++k ) {
        sum += C( k )*C( n - k - 1 );
    }

    return sum;
}
```

Implementations

- This defines C_0, and then calculates C_1, C_2, etc.

$$C_0 = 1$$

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-k-1}$$

```
unsigned long C( unsigned long n ) {
    unsigned long catalan[n + 1];
    catalan[0] = 1;

    for ( unsigned long i{ 1 }; i <= n; ++i ) {
        catalan[i] = 0;

        for ( unsigned long k{ 0 }; k < i; ++k ) {
            sum += catalan[k]*catalan[i - k - 1];
        }
    }

    return catalan[n];
}
```

Implementations

- Here is an alternative recursive definition:

```
unsigned long C( unsigned long n ) {
    if ( n == 0 ) {
        return 1;
    } else {
        return (2*(2*n - 1)*C( n - 1 ))/(n + 1);
    }
}
```

$$C_0 = 1$$

$$C_n = \frac{2(2n-1)}{n+1} C_{n-1}$$

$$C_n = \frac{(2n)!}{(n+1)!n!}$$

$$= \frac{(2n)(2n-1)(2n-2)!}{(n+1)n!n(n-1)!}$$

$$= \frac{(2n)(2n-1)}{(n+1)n} \frac{(2(n-1))!}{n!(n-1)!}$$

  - Why place parentheses around $2\,(2n-1)C_{n-1}$?

Implementations

- The same recursive formula, but calculating $C_1, C_2, \ldots$

$$C_0 = 1$$

$$C_n = \frac{2(2n-1)}{n+1} C_{n-1}$$

```
unsigned long C( unsigned long n ) {
    unsigned long result{ 1 }; // C(0)

    for ( unsigned int k{ 1 }; k <= n; ++k ) {
        result = (2*(2*k - 1)*result)/(k + 1);
    }

    return result;
}
```

  - Again, we need parentheses around $2\ (2n-1)C_{k-1}$

Issues with these implementations

- Problems:
  - What if $\binom{2n}{n} \geq 2^{64}$, but $\frac{1}{n+1}\binom{2n}{n} < 2^{64}$ and $\binom{2n}{n} - \binom{2n}{n+1} < 2^{64}$?
  - What if $(2n)! \geq 2^{64}$ but $\frac{(2n)!}{n!(n+1)!} < 2^{64}$?
  - This formula is purely additive, and is thus not subject to overflow: $C_n = \sum_{k=0}^{n-1} C_k C_{n-k-1}$
  - Now, for $n > 5$, $\frac{2(2n-1)}{n+1}$ is not an integer and $\frac{C_{n-1}}{n+1}$ is not always an integer...
    - Thus, what if $2(2n-1)C_{n-1} \geq 2^{64}$, but $\frac{2(2n-1)}{n+1} C_{n-1} < 2^{64}$?

Issues with these implementations

- The formulas using binomial coefficients or the factorials or ratios of products are non-starters
  - It is very difficult and expensive to ensure that overflow occurs in neither the numerator or denominator
- The formula $C_n = \sum_{k=0}^{n-1} C_k C_{n-k-1}$ can be very expensive
  - With the recursive algorithm, it is exponentially slow
  - With the array, if you double *n*, it takes four times longer to run
- Can we do something with the formula $\frac{2(2n-1)}{n+1} C_{n-1}$?
  - We could find $\gcd(n+1, C_{n-1})$
  - Fortunately, we observe at least one of these is an integer:

$$\frac{C_{n-1}}{n+1} \quad \frac{2C_{n-1}}{n+1} \quad \frac{3C_{n-1}}{n+1}$$

Optimal implementation

```
unsigned long C( unsigned long n ) {
    assert( n <= 36 );
    unsigned long result{ 1 };

    for ( unsigned int k{ 1 }; k <= n; ++k ) {
        unsigned long numer{ 2*(2*k - 1) };
        unsigned long denom{ k + 1 };

        if ( denom%2 == 0 ) {
            numer /= 2;
            denom /= 2;
        }

        if ( (numer%3 == 0) && (denom%3 == 0) ) {
            numer /= 3;
            denom /= 3;
        }

        // This must be a product of three integers, hence, overflow cannot occur
        assert( result%denom == 0 );
        result = numer*(result/denom);
    }

    return result;
}
```

$$C_0 = 1$$

$$C_n = \frac{2(2n-1)}{n+1} C_{n-1}$$

Optimal implementation

```
unsigned long *C() {
    unsigned long *array{ new unsigned long[37] };
    array[0] = 1;

    for ( unsigned int k{ 1 }; k <= n; ++k ) {
        unsigned long numer{ 2*(2*k - 1) };
        unsigned long denom{ k + 1 };

        if ( denom%2 == 0 ) {
            numer /= 2;
            denom /= 2;
        }

        if ( (numer%3 == 0) && (denom%3 == 0) ) {
            numer /= 3;
            denom /= 3;
        }

        assert( array[k - 1]%denom == 0 );
        array[k] = numer*(array[k - 1]/denom);
    }

    return array;
}
```

$$C_0 = 1$$

$$C_n = \frac{2(2n-1)}{n+1} C_{n-1}$$

This just creates an array of all 37 Catalan numbers that can be stored as `unsigned long`

Observations

- There were multiple formulas presented on the Wikipedia page with respect to Catalan numbers
  - The most useful formula was the last one listed
  - Most implementations shown online unfortunately pick the sub-optimal algorithms for implementing these calculations
- Moral: don't use the first formula you find online,
  and certainly don't rely on public websites for your algorithms
- These are all implemented online
  - Try to guess which is the first formula to fail

Summary

- Following this lesson, you now
  - Understand the definition of Catalan numbers
  - Are aware that there are multiple formulas to calculate these
  - Also understand that not all these formulas can or should be implemented
  - Observed that the formula that was most suited for an implementation is the last one listed on Wikipedia, at least

References

[1] https://en.wikipedia.org/wiki/Catalan_numbers

Colophon

These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`.

The photographs of lilacs in bloom appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens on May 27, 2018 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.

Disclaimer

These slides are provided for the ECE 150 *Fundamentals of Programming* course taught at the University of Waterloo. The material in it reflects the authors' best judgment in light of the information available to them at the time of preparation. Any reliance on these course slides by any party for any other purpose are the responsibility of such parties. The authors accept no responsibility for damages, if any, suffered by any party as a result of decisions made or actions based on these course slides for any other purpose than that for which it was intended.