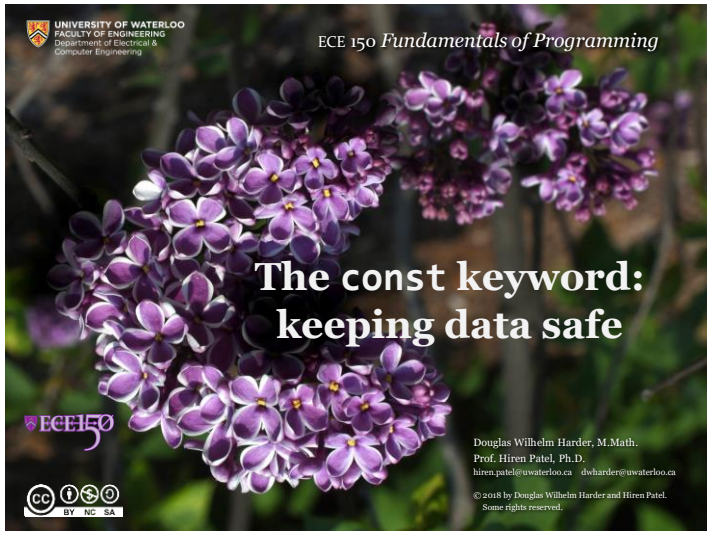

Outline

- In this lesson, we will:
  - Understand the need to protect stored values
  - Examine constant local variables and constant references
  - Examine constant parameters and constant reference parameters
  - See various applications
  - Understand the software engineering principle behind this keyword

Programming practice

- Up to this point, we have looked most at principles of programming
- We have seen the structured programming theorem
  - This is responsible for our ability to write software projects on the scale we see today
  - By restricting flow control to conditional statements and looping statements, code maintainability is greatly increased

- We will now look at another software engineering tool that can be used to reduce both development and maintenance time and costs

Software problem

- Some variables or parameters contain values not meant to be changed:

```
int main() {
    double pi{3.1415926535897932};
    // do something...

    if ( pi > x ) {
        // Do something...
    } else if ( pi = x ) {
        // Do something else...
    }
}
```

- What happened here?

Local constants

- To prevent anyone from assigning to a local variable after it has been initialized, that variable can be declared constant:

  typename **const** *IDENTIFIER*{*value*};

- During the software design phase, all local variables should be inspected to determine which are truly variable, and which should be constant
  - Indicating that a local variable is constant can allow for optimizations by the compiler
  - Constants are identified using ALL CAPS s
    - This ensures other developers immediately differentiate between constants and variables

- You can also use

  const *typename IDENTIFIER*{*value*};

Local constants

- It is a compile-time error if a local constant is initialized:

```
example.cpp: In function 'int main()':
example.cpp:6:12: error: uninitialized const 'SAMPLE_COUNT'
    int const SAMPLE_COUNT;
              ^
```

- It is a compile-time error to ever assign to a local constant:

```
example.cpp: In function 'int main()':
example.cpp:7:15: error: assignment of read-only variable 'SAMPLE_COUNT'
SAMPLE_COUNT = 64;
             ^
```

- A local constant is also referred to as a *"read-only variable"*

Mathematical constants

- Mathematical and physical constants are declared as such:

```
int main() {
    double const PI{3.1415926535897932};
    double const TWO_PI{2.0*PI};
    double const PI_BY_2{PI/2.0};
    double const HC{1.98644586e-25};      // Jm
    double const AVOGADRO{6.02214076e23}; // 1/mol
    // Do something...
}
```


Constant parameters

- It is also possible to declare a parameter constant

```
double average( unsigned int const num_values ) {
    // cannot accidentally assign to 'num_values'
}
```

- This prevents any future editor of this code from accidentally changing the value
  - There may be no reason for the function to change this value
  - Most functions we have written would benefit from this:

```
unsigned long factorial( unsigned long const n ) {
    assert( n <= 20 );  // 21! = 51090942171709440000 > pow( 2, 64 )

    unsigned long result{1};

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

    return result;
}
```

Why not remove const?

- If the const keyword keeps you from doing something you think you want to do, it's up to you to revisit the specification:
  - Why was it required that the parameter be declared const?
  - Have the circumstances changed since the specification was written?
  - Who is responsible for the specification, and what is their opinion?

Synopsis

- A perfectly functioning program that uses const will still be a perfectly functioning program if **all** instances of const are removed
- A perfectly function program that uses const correctly, however, is less likely to have errors introduced by subsequent programmers
- A program design that uses const is less likely to have programmers *go down the wrong path* during development than one that does not use const

- All const says to the programmer or compiler:
  - This variable, parameter or reference should not be changed

Summary

- Following this lesson, you now:
  - Know that the const keyword is there for one reason only:
    - To prevent you from assigning to a variable or parameter that must not be changed
  - Understand that anything declared const must be initialized
  - Know the relationships:

    local variable ↔ local constant

    parameter ↔ constant parameter

References

[1] https://en.wikipedia.org/wiki/Const_(computer_programming)

[2] http://www.cplusplus.com/doc/tutorial/constants/

[3] http://www.dietmar-kuehl.de/mirror/c++-faq/const-correctness.html

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.