In this lesson, we will:
- Understand the need to protect stored values
- Examine constant local variables and constant parameters
- See how this is used
- Understand the software engineering principle behind this keyword

Some local variables or parameters contain values that are not meant to be changed:

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

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

    // Keep doing stuff...
}
```

- What happened here?
Local constants

- To prevent assignment to a local variable after initialization, that identifier can be declared constant:
  ```cpp
typename const IDENTIFIER(value);
```
- During the software design phase, all local variables should be inspected to determine if their value need be changed after initialization
  - Indicating that a local variable is constant can allow for optimizations by the compiler
  - Constants are identified using ALL CAPS
    - This ensures other developers immediately differentiate between local constants and local variables
- You can also use
  ```cpp
  const typename IDENTIFIER(value);
  ```

Mathematical constants

- Mathematical and physical constants are declared to be constant:
  ```cpp
  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{1.986445866e-25}; // J/m
      double const AVOGADRO{6.02214076e23}; // 1/mol
      // Do something...
  }
  ```

Constant parameters

- It is also possible to declare a parameter constant
  ```cpp
  double average(unsigned int const num_values) {
      // cannot accidentally assign to 'num_values'
  }
  ```
**Constant parameters**

- 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:
    ```c
    unsigned long factorial(unsigned long const n) {
        //
        // 21! = 51090942171709440000 > 2
        assert( n <= 20 );

        unsigned long result{1};
        for ( unsigned long k{2}; k <= n; ++k ) {
            result *= k;
        }
        return result;
    }
    ```

**Synopsis**

- A perfectly functioning program that uses `const` will still be a perfectly functioning program if **all** instances of `const` are removed
- A perfectly functioning program that uses `const` correctly, however, is less likely to have errors introduced by subsequent programmers
- All `const` says to the programmer or compiler:
  - This local variable or parameter 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**

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.