Modern C++ Overview Part One Solutions

Universal Initialization

- Write a program that uses universal initialization to define the following variables
  - x, type int, initial value 7
  - s, type std::string, initial value "Let us begin"
  - y, type int, initial value 7.7
- Why does the definition of y not compile? Try again with the traditional form of initialization
  - Initializing an int with a double value is a narrowing conversion. This is not allowed with brace initialization
- Print out the values of x, s, and y
- Test and run your program

Universal Initialization contd

- Still using universal initialization, add the following variables to your program
  - v, type std::vector of int, values 4, 2, 3, 5, 1
  - hello, type std::string, initial values 'H', 'e', 'l', 'l', 'o'
- Print out the values of v and hello
- Test and run your program

nullptr

- Describe the nullptr feature
  - nullptr should be used instead of NULL to represent a null pointer. Its type is compatible with any pointer, but cannot be converted to int
- Write two overloaded functions, one taking an int by value and one taking pointer to int. Each function prints out the type of its argument
- Write a program which makes two calls to the function, one with argument NULL and one with argument nullptr
- Run your program. Explain your observations
  - The nullptr argument calls the function which takes a pointer. This is because nullptr is a pointer type
  - The result of calling with a NULL argument is compiler dependent. This is because the type of NULL is not specified

std::chrono

- Using C++11 syntax, write down expressions which represent intervals of
  - 2 seconds

    seconds(2)
  - 20 milliseconds

    milliseconds(20)
  - 50 microseconds

    microseconds(50)

std::chrono literals

- Repeat the previous exercise, using C++14 syntax
  - 2 seconds

    2s
  - 20 milliseconds

    20ms
  - 50 microseconds

    50us

Automatic Type Deduction

- Briefly describe the auto keyword
  - The auto keyword can be used instead of a type specifier. The compiler will deduce the type. Qualifiers such as const and & are ignored
- Write down an expression which uses the auto keyword to define a variable whose initial value is 6. What will be the type of this variable?
  - int

Automatic Type Deduction

- Write down an expression which uses the auto keyword to define a variable whose initial value is an iterator to the first element in a vector of string

```
auto it = v.begin();
```

- Write down an expression which defines the same variable, using traditional syntax for the type

```
vector<string>::iterator it = v.begin();
```

auto with qualifiers

- What happens when we want to use auto to create a variable which is const, or is a reference?
  - The type of the variable will be the underlying type, e.g. int
  - If we want qualifiers, such as const or reference, we must type them explicitly

auto and for loops

- Write a program that creates a vector whose elements are 4, 2, 5, 3 and 1
- Using iterators with traditional syntax, write a loop that adds 2 to each element
- Using iterators with traditional syntax, write a loop that prints out each element
- Rewrite your program so that it uses auto for the iterator type instead of an explicit type

Range for loops

- Rewrite your program from the previous exercise to use range-for loops