```
//============================
// Section 6 - Variables and constants
//============================

//what is a variable
It's a name that makes sense to us to move data around
- Abstraction for a memory location
- We can use names that make sense instead of memory addresses
- They have two main properties
    Type (integer, double, string, Person class, etc) - statically typed
    Value - the contents
- Variables must be declared before they are used
- Variable's values may change

// Declaring and intializing variables
VariableType Variablename;
int age; // uninitialized
int age1 = 21; // C-style initialization
int age(21); // constructor initialization
int age{21}; // c++11 list initialization syntax - use this one

Naming rules
- Can contain letters, numbers, underscores
- Must begin with a letter or _ (cannot begin with number)
- Cannot use C++ reserved keywords
- Cannot redeclare a name in same scope

Styles and best practices
- Be consistent - camelCase vs PascalCase
- Use meaningful names
- Never use variables before initializing them
- Declare them close to where you need them in the code

// Global variables
Declared outside of any function
Can be accessed by any part of the program
Automatically initialized to 0
Local variables are declared inside of and are scoped to their functions
Compiler will prefer local variables over global ones

// C++ Built in primitive types
AKA fundamental types implemented by the language itself
Includes:
- Character types
- Integer types (signed and unsigned)
- Floating point types
- Boolean types

Size and precision is compiler dependant (use #include <climits>)

Type sizes:
Expressed in bits
The more bits you have, the more values can represent, and the more storage you need

Size(in bits)       Representable values        Scientific notation
    8                   256                         2e8
    16                  65,536                      2e16
    32                  4,294,967,296               2e32
    64                  18,446,744,073,709,551,615  2e64

Character types:
- Used to represent single characters 'a', 'b'
- Wider types are used to represent wide character types

Type name               Size\Precision
    char                1 byte\8 bits
    char16_t            16 bits
    char32_t            32 bits
    wchar_t             Largest char set
```

```
Integer types
- Whole numbers
- signed and unsigned are supported

Type name               Size\Precision
signed short int        16 bits
signed int              16 bits
signed long int         32 bits
signed long long int    64 bits

unsigned short int      16 bits
unsigned int            16 bits
unsigned long int       32 bits
unsigned long long int  64 bits

You can also store signed and unsigned integers in the char data type
By default integers are signed
If you want only positive values, you must specify unsigned

// Floating point types
Non-integer numbers (real numbers)
Represented by mantissa and exponent (scientific notation)
Precision is the number of digits in the mantissa
Precision and size depend on the compiler

Type Name           Size\Typical precision              Typical Range
float                       /7 decimal digits           1.2 x 10e-38 to 3.4 x 10e38
double              No less than float\15 dec digits    2.2 x 10e-308 to 1.8 x 10e308
long double         No less than double\19 dec digits   3.3 x 10e-4932 to 1.2 x 10e4932

Computers have a finite amount of storage - real numbers can have infinite digits after
dec
The computer will store an approximation of the number - it can't store Pi precisely

// Boolean
Used to represent true and false
Zero is false
Any non-zero is true
keywords true and false are also used

Type name   Size\Precision
bool        Usually 8 bits
            true or false (c++ keywords)

// Code examples of data types

// Integer Types
char middleInitial{'M'};
unsigned short int examScore{55}; // can also just say unsigned short
int countriesRepresented{65};
long peopleInFlorida{2061000};

// This will warn you of narrowing because this number is too big for the long
// as long as you are using the list initialization. If you use the older C style
// initialization it will overflow and have unpredictable results
long peopleOnEarth{7'600'000'000}; // C++ 14 allows you to use ' in between numbers
// it will strip them back out on the compiler side

long long distanceToAlphaCentauri{9'461'000'000'000};

// Floating point types
float carPayment{401.23};
double pi{3.14159}; // Use for larger floats (double float)
long double{2.7e120};

// Boolean Types
bool gameOver{false}; // will print out 0 instead of the word false
```

```
// Overflow examples
short value1{30000};
short value2{1000};
short sum{value1 * value2}; // This will overflow and return unpredictable results

// What is the size of a variable (sizeof)
sizeof- returns the number of bytes of a type or variable
// Examples:
sizeof(int);
sizeof(double);
sizeof(someVariable);
sizeof someVariable; // Parens are optional for variables

Data comes from <climits> and <cfloat>
// Handy constants defined in these #includes:
INT_MAX
INT_MIN
LONG_MIN
LONG_MAX
FLT_MIN
FLT_MAX
SHRT_MIN
LLONG_MIN

// What is a constant
Similar to variables
- Have names
- Occupy storage
- usualyl strongly typed

Different as well
- Cannot be changed

// Types of constants
- Literal constants
- Declared constants
const keywords

- Constant Expressions
constexpr keyword

-Enumerated Constants
enum keyword

-Defined constants
#define

// Literals
x = 12;
name = "Josie";
// Can be explicit with the types of literal constants
12      - an integer
12U     - and unsigned integer
12L     - a long integer
12LL    - a long long integer
12.1    - a double
12.1F   - a float
12.1L   - a long double

// Character literal constants
\n      - newline
\r      - return
\t      - tab
\b      - backspace
\'      - single quote
\"      - double quotes
\\      - backslash
```

```
// Declared Constants (most common);
const double pi{3.1415926}; // Must initialize when you declared
const int monthsInYear{12};

// Defined constants (common in older C and C++ code)
#define pi 3.1415926
// This is a preprocessor directive and the preprocessor will
// do a blind find and replace of anything named pi
// And since it doesn't know c++, it won't do typechecking.
NOTE: Do not use in modern C++!

//============================
// Example program - carpet cleaning service
//============================

#include <iostream>
#include <string>
#include <vector>

int main(){

    const double roomCleaningPrice{ 30.00 };
    int numberOfRooms{ 0 };

    // Prompt the user for the number of rooms
    std::cout << "=====================================" << std::endl;
    std::cout << "Welcome to the cleaning service. " << std::endl;
    std::cout << "=====================================" << std::endl;
    std::cout << std::endl;
    std::cout << "Please enter the number of rooms to clean:" << std::endl; \
    std::cin >> numberOfRooms;

    // Display the number of rooms
    std::cout << "You entered: " << numberOfRooms << std::endl;

    // Display the price per room
    std::cout << "The price to clean a room is: " << roomCleaningPrice << std::endl;

    // Display the cost (number of rooms * price per room)
    std::cout << "The base cost to clean these rooms will be: " << numberOfRooms *
    roomCleaningPrice << std::endl;

    const double taxRate{ 0.06 };
    double totalTax{ 0 };
    double totalPrice{ 0 };
    totalTax = (double)numberOfRooms * (double)roomCleaningPrice * taxRate;
    totalPrice = ((double)numberOfRooms * (double)roomCleaningPrice * taxRate) + (
    numberOfRooms * roomCleaningPrice);

    // Display the tax (number of rooms * price per room * tax rate)
    std::cout << "The tax to clean these rooms will be: " << totalTax << std::endl;

    // Display the total estimate (number of rooms * price per room) + (number of rooms
    * price per room * tax rate)
    std::cout << "The total cost to clean these rooms will be: " << totalPrice << std::
    endl;
}

//============================
// End Example program - carpet cleaning service
//============================
```

```cpp
//============================
// Section 6 Challenge
//============================

#include <iostream>
#include <string>
#include <vector>

// Second room cleaning service program
// $25 for a small room, $35 for a large room
// Sales tax is 6%
// Estimates valid for 60 days
// Ask how many of each room then do the calculation

int main(){

    const double smallRoomPrice{ 25.00 };
    const double largeRoomPrice{ 35.00 };
    const double salesTax{0.06};
    const int estimatesValidFor{ 60 };

    std::cout << "Welcome to the cleaning service." << std::endl;
    std::cout << "\nPlease type the number of small rooms and press enter: " << std::
    endl;
    int numberOfSmallRooms{ 0 };
    std::cin >> numberOfSmallRooms;
    std::cout << "Please enter the number of large rooms and press enter: " << std::endl;
    int numberOfLargeRooms{ 0 };
    std::cin >> numberOfLargeRooms;

    const double baseRoomCost = (numberOfSmallRooms * smallRoomPrice) + (
    numberOfLargeRooms * largeRoomPrice);
    const double totalTax = (baseRoomCost * salesTax);
    const double totalCost = (baseRoomCost + totalTax);

    std::cout << "Base cost: " << baseRoomCost << std::endl;
    std::cout << "Tax: " << totalTax << std::endl;
    std::cout << "Total Cost: " << totalCost << std::endl;
    std::cout << "Estimates are valid for: " << estimatesValidFor << " days." << std::
    endl;

    return 0;
}

//============================
// End Section 6 Challenge
//============================
```