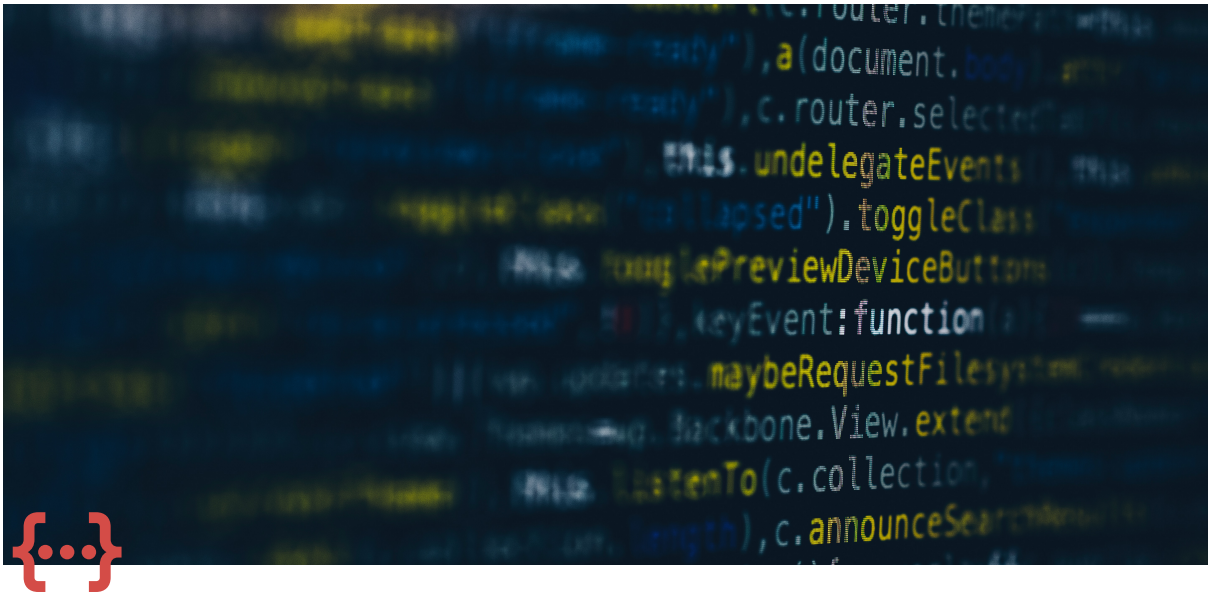

Lesson 2: Variables, Data Types and Operators

Variables

A variable in C++ is a named location in memory where a program can manipulate data. This location is used to hold the value of the variable. You can consider a variable as a container that holds a particular value during the execution of the program. To be able to use a variable, it must first be declared. A variable declaration tells the compiler the variable name and the type of data it will be holding. For example:

```
int myVariable;
```

In this statement, we have declared a variable named "myVariable" of the type int.

Data Types

Data types in C++ specify the size and type of values to be stored in a variable during its lifespan. Here are some of the fundamental (primitive) data types in C++:

1. **Integers (`int`)**: Used to store numerical values without a decimal i.e., whole numbers. It typically requires 4 bytes of memory and values can range from

-2147483648 to 2147483647.

1. **Char (`char`)**: Used to store a single character enclosed within single quotes. It typically requires 1 byte of memory.
2. **Boolean (`bool`)**: Used to store a true or false boolean value.
3. **Floating Point (`float`)**: Used to store decimal numbers (real numbers), like 3.14, -0.004, 25.67 etc.
4. **Double Floating Point (`double`)**: It is similar to `float` but is more precise (has a larger precision and size).
5. **Void (`void`)**: Represents the absence of type. It is used to indicate that a function does not return a value.
6. **Wide Character (`wchar_t`)**: It is a larger-sized character data type used to store a wider range of characters.

Here is an example demonstrating how to use these data types:

```
int age = 25;
char gender = 'M';
bool isMarried = false;
float average = 98.6;
double balance = 10234.50;
```

Indicating a value at the time of declaring the variable, as shown above, is known as initialization.

Data Type Modifiers

C++ provides data type modifiers that you can use to alter the capacity of fundamental data types. They include:

- Signed
- Unsigned
- Short

- Long

Here's an example demonstrating how to use these data types and modifier:

```
signed int x;// can store both negative and positive numbers
unsigned int y;// can store only positive numbers
long long z;// can store very large whole numbers
short int a;// used for storing small whole numbers
```

Remember, you should choose the most appropriate data type for a given variable based on the nature of the data it should store. Choosing an inefficient data type can waste memory resources, and might even lead to unexpected results due to overflow.

sizeof() Operator

The `sizeof()` operator is a special operator that returns the size of the specified data type, or the data type of a variable. This can be particularly useful when the size varies from one machine to another, such as with `int` and `float`.

```
int a;
cout << "Size of int : " << sizeof(a) << endl;
```

Practical Exercises:

1. Write a C++ program to declare two integer variables, one float variable, and one character variable then initialize them to your favorite numbers and letter. Display them in the console.
2. Write a program to find the size of the `double` data type on your machine using `sizeof()`.
3. Declare a variable `isCodingFun` of Boolean type and assign it `true`. Declare another variable `codingIsHard` and assign it `false`. Display both variables.
4. Can an `unsigned int` variable hold a negative number? Try to write a program and observe what happens.

Operators in C++

Operators are special symbols in programming languages that carry out some form of computation on one or more operands. In this lesson, we will look at different types of operators in C++ and how to use them in your programs.

Arithmetic Operators

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, division, and so on. These include:

- `+` : Addition
- `-` : Subtraction
- `*`: Multiplication
- `/`: Division
- `%`: Modulus (remainder after division)

For example:

```
int a = 10, b = 2;
cout << "Addition: " << a + b << endl;  // outputs: 12
```

Remember that the division of two integers results in an integer. If you want a decimal result, one or both of the operands must be a float or double.

Comparison Operators

Comparison operators are used to compare values. They return a boolean result (`true` or `false`). These include:

- `==`: Equal to
- `!=`: Not equal to
- `>` : Greater than
- `<` : Less than
- `>=`: Greater than or equal to
- `<=`: Less than or equal to

For example:

```
if (a == b)
    cout << "a is equal to b" << endl;
else
    cout << "a is not equal to b" << endl;
```

Logical Operators

Logical operators are used to combine two or more conditions. They return a boolean result. These include:

- `&&`: Logical AND (both conditions must be true)
- `||`: Logical OR (at least one condition must be true)
- `!` : Logical NOT (reverses the truth value)

For example:

```
if (a > b && a != 0)
    cout << "a is greater than b and a is not zero" << endl;
```

It's important to know how these operators can be combined in conditions. `&&` has higher precedence than `||`, similar to how multiplication comes before addition in arithmetic.

Assignment Operators

The assignment operator `(=)` is used to assign a value to a variable. There are also compound assignment operators that perform an operation and assignment in one step. These include:

- `+=` : Add and assign
- `=` : Subtract and assign
- `=` : Multiply and assign
- `/=` : Divide and assign
- `%=` : Modulus and assign

For example:

```
a += b;  // equivalent to a = a + b;
```

Increment and Decrement Operators

These operators are used to increment (`++`) or decrement (`--`) a variable's value by 1. There are two types: postfix `(a++)` and prefix `(++a)`. The difference between them is the timing when the increment/decrement is applied.

```
int a = 5;
cout << a++ << endl;  // outputs 5, then increments
cout << ++a << endl;  // increments first, then outputs 7
```

That's the basics of C++ operators. Continue experimenting with these operators to understand them better.

Exercises

1. Write a program to add, subtract, multiply, and divide two numbers input by the user.
2. Write a program to compare two numbers and determine which one is the larger.
3. Write a program that increments a number by 1 and then by 2, using both postfix and prefix operators. Note what happens.

Remember, practice makes perfect. Try to use these operators as much as possible in your practice code.

Happy coding!