Data Types

Terms

Array
Binary system
Boolean values
Casting
Characters
Compile-time error
Data type
Decimal system
Floating-point number
Hexadecimal system
Index
Overflow
Run-time error
Stream manipulator
String
Underflow

Summary

- C++ has several built-in *data types* for storing *integers* (whole numbers), *floating-point numbers* (numbers with a decimal point), characters, and Boolean values (true/false).
- Floating-point numbers are interpreted as **double** by default. To represent a **float**, we have to add the **F** suffix to our numbers (eg 1.2F).
- Whole numbers are interpreted as **int** by default. To represent a **long,** we have to use the **L** suffix (eg 10L).
- Using the **auto** keyword, we can let the compiler infer the type of a variable based on its initial value.
- Numbers can be represented using the *decimal*, *binary*, and *hexadecimal systems*.
- If we store a value larger or smaller than a data type's limits, *overflowing* or *underflowing* occurs.

- Using the **sizeof()** function, we can see the number of bytes taken by a data type.
- We can use *stream manipulators* to format data sent to a stream. The most common manipulators are **setw, fixed, setprecision, boolalpha, left,** and **right.**
- The Boolean **false** is represented as 0. Any non-zero number is interpreted as the Boolean **true**.
- In C++, characters should be surrounded with single quotes.
- Characters are internally represented as numbers.
- A *string* is a sequence of characters and should be surrounded by double quotes.
- We use *arrays* to store a sequence of items (eg numbers, characters, etc).
- Array elements can be accessed using an *index*. The index of the first element in an array is 0.
- When we store a smaller value in a larger data type, the value gets automatically *cast* (converted to) the larger type. When storing a large value in a smaller data type, we have to explicit cast the value.
- C-style casting involves prefixing a variable with the target data type in parentheses. In C++, we use the **static_cast** operator.
- C++ casting is safer because conversion problems can be caught at the *compile-time*. With C-style casting, we won't know about conversion issues until the *run-time*.


```cpp
// Data types
double price = 9.99;
float interestRate = 3.67F;
long fileSize = 90000L;
char letter = 'a';
string name = "Mosh";
bool isValid = true;
auto years = 5;

// Number systems
int x = 255;
int y = 0b111111;
int z = 0xFF;


// Data types size and limits
int bytes = sizeof(int);
int min = numeric_limits<int>::min();
int max = numeric_limits<int>::max();

// Arrays
int numbers[] = { 1, 2, 3 };
cout << numbers[0];

// C-style casting
double a = 2.0;
int b = (int) a;

// C++ casting
int c = static_cast<int>(a);
```

