C++- Variables and Types

Identifiers

A valid identifier is a sequence of one or more letters, digits, or underscore characters (_). Spaces, punctuation marks, and symbols cannot be part of an identifier. In addition, identifiers shall always begin with a letter. They can also begin with an underline character (_), but such identifiers are “on most cases” considered reserved for compiler-specific keywords or external identifiers, as well as identifiers containing two successive underscore characters anywhere. In no case can they begin with a digit.

The C++ language is a "case sensitive" language.

Fundamental data types

Fundamental data types are basic types implemented directly by the language that represent the basic storage units supported natively by most systems. They can mainly be classified into:

- **Character types**: They can represent a single character, such as 'A' or '$'. The most basic type is char, which is a one-byte character. Other types are also provided for wider characters.
- **Numerical integer types**: They can store a whole number value, such as 7 or 1024. They exist in a variety of sizes, and can either be signed or unsigned, depending on whether they support negative values or not.
- **Floating-point types**: They can represent real values, such as 3.14 or 0.01, with different levels of precision, depending on which of the three floating-point types is used.
- **Boolean type**: The boolean type, known in C++ as bool, can only represent one of two states, true or false.

Here is the complete list of fundamental types in C++:

Group	Type names*	Notes on size / precision
Character types	**char**	Exactly one byte in size. At least 8 bits.
	char16_t	Not smaller than char. At least 16 bits.
	char32_t	Not smaller than char16_t. At least 32 bits.
	wchar_t	Can represent the largest supported character set.
Integer types (signed)	**signed char**	Same size as char. At least 8 bits.
	signed **short** *int*	Not smaller than char. At least 16 bits.
	signed **int**	Not smaller than short. At least 16 bits.
	signed **long** *int*	Not smaller than int. At least 32 bits.
	signed **long long** *int*	Not smaller than long. At least 64 bits.
Integer types (unsigned)	**unsigned char**	(same size as their signed counterparts)
	unsigned short *int*	
	unsigned *int*	
	unsigned long *int*	
	unsigned long long *int*	
Floating-point types	**float**	
	double	Precision not less than float
	long double	Precision not less than double
Boolean type	**bool**	

Void type	**void**	no storage
Null pointer	**decltype(nullptr)**	

Type sizes above are expressed in bits; the more bits a type has, the more distinct values it can represent, but at the same time, also consumes more space in memory:

Size	Unique representable values	Notes
8-bit	256	= 2^8
16-bit	65 536	= 2^{16}
32-bit	4 294 967 296	= 2^{32} (~4 billion)
64-bit	18 446 744 073 709 551 616	= 2^{64} (~18 billion billion)

Declaration of variables

C++ is a strongly typed language and requires every variable to be declared with its type before its first use.

A variable definition specifies a data type, and contains a list of one or more variables of that type as follows:

```
type variable_list;
```

Examples:

```
int a;
float mynumber;
int b, c;
```

Initialization of variables

In C++, there are three ways to initialize variables. They are all equivalent and are reminiscent of the evolution of the language over the years:

```
int x = 0; or int x (0); or int x {0};
```

Local Variables

Variables that are declared inside a function or block are local variables.

```
#include <iostream>
using namespace std;
int main () {
   // Local variable declaration:
   int a, b;
   int c;
   // actual initialization
   a = 10;
   b = 20;
   c = a + b;
   cout << c;
   return 0;
}
```

Global variable

In C++, variables can also be declared outside of a function. Such variables are called global variables. Unlike local variables, which are uninitialized by default, static variables are zero-initialized by default.

```
#include <iostream>
 // Variables declared outside of a function are global variables
int g_x {}; // global variable g_x
 void doSomething()
{
    // global variables can be seen and used everywhere in the file
    g_x = 3;
    std::cout << g_x << '\n';
}
 int main()
{
    doSomething();
    std::cout << g_x << '\n';
     // global variables can be seen and used everywhere in the file
    g_x = 5;
    std::cout << g_x << '\n';
     return 0;
}
// g_x goes out of scope here
```

Sizeof Operator

The sizeof is a keyword, but it is a compile-time operator that determines the size, in bytes, of a variable or data type.

The sizeof operator can be used to get the size of classes, structures, unions and any other user defined data type.

The syntax of using sizeof is as follows:

```
sizeof (data type)
```

Type deduction: auto and decltype

When a new variable is initialized, the compiler can figure out what the type of the variable is automatically by the initializer. For this, it suffices to use auto as the type specifier for the variable:

```
int foo = 0;
auto bar = foo;  // the same as: int bar = foo;
```

Variables that are not initialized can also make use of type deduction with the decltype specifier:

```
int foo = 0;
decltype(foo) bar;  // the same as: int bar;
```

Exercises

1. Write a program that displays memory that is occupied by all types of data you can manipulate in C++.
2. Write a program that displays the values of different local and global variables which are not initialized.

3. Write a program that allows you to enter with the keyboard and display in the console all types of data you can manipulate in C++.