C++ Variables, Data Types, and Operators

Contents

1 Introduction

This guide explores the fundamental concepts of variables and data types in C++, emphasizing primitive data types, variable declaration, and initialization, along with detailed discussions on casting and operators.

2 Variables, Data Types, and Operators

2.1 Primitive Data Types

Primitive data types in C++ include `bool`, `int`, `char`, `float`, and `double`. They have different sizes and characteristics.

`float` is a primitive data type that can hold floating-point values up to 7 digits, while `double` can hold floating-point values up to 15 digits.

```cpp
int a = pow(2,32);
std::numeric_limits<int>::max();
std::cout<<a<<"\n";
std::cout<<std::numeric_limits<int>::max()<<"\n";
```

`int foo;` declares an integer variable `foo`. The first value is assigned automatically, and it is often assigned to a garbage value.

2.2 Namespace

Using `using namespace std;` introduces all elements in the `std` namespace into the current scope.

```cpp
int main() {
    using namespace std;
    int foo; // Undefined Behavior
    cout<<"The value of foo is : "<< foo <<endl;
    int bar = 5; // Initialization
    int baz {5}; // Prefer initialization with curly braces
    int bam = 0.6; // Possible loss of data
    bool foo {5}; // Incorrect initialization for bool

    const int var = 0;
    constexpr int thx =  5; // Difference between const and
    constexpr
}
```

auto is a type deduction keyword used to automatically deduce the data type of the variable.

```cpp
auto bil142 = 25;
bil142 = 28.2;
cout<<"The value of bil142 is : "<< bil142 <<"\n";
```

The sizeof operator returns the size of its operand in bytes.

```cpp
sizeof(double)
double foo{0.0};
sizeof(foo);
```

Other types like uint32_t, uint8_t, and uint64_t exist. uint8_t and char are the same. Limits of types can be checked using std::numeric_limits.

2.3 Casting

Static Cast: Used for explicit conversions considered safe by the compiler.

```cpp
int num = 10;
double numDouble = static_cast<double>(num);
```

C-style Casting:

```cpp
int num = 10;
double numDouble = (double)(num);
```

2.4 Arithmetic Operators

Arithmetic operators in C++ include addition, subtraction, multiplication, division, and modulus.

```cpp
int a{10}, b{20};
int sum = a + b;
int modOperator = 20 % 10;
```

result would be 0 in the first case and 1.5 in the second case.

```cpp
int a = 10, b = 20;
float result = a / b;
float a = 3.0 / 2.0;
```

result would be 0.0 due to integer division.

```cpp
int a = 10, b = 20;
double result = static_cast<double>(a / b);
```

result would be 0.5.

```
1  int a = 10, b = 20;
2  double result = static_cast<double>(a) / b;
```

result would be 0.5.

```
1  int a = 10, b = 20;
2  double result = static_cast<double>(a) / static_cast<double>(
       b);
```

2.5   Bitwise Operators

```
1  int main() {
2      using namespace std;
3      unsigned char a = 5, b = 9;
4      unsigned char result{0};
5
6      result = a & b;
7      result = a | b;
8      result = a ^ b;
9      result = ~a;
10     result = b << 1;
11     result = b >> 1;
12 }
```

result would be 0.

```
1  using namespace std;
2  unsigned char a = 5;
3  unsigned char result{0};
4  result = a >> 3;
5  cout<<static_cast<int>(result);
```

2.6   Logical Operators

Logical operators in C++ include logical AND (&&), logical OR (||), and logical NOT (!).

```
1  bool isTrue = true, isFalse = false;
2  bool result = (isTrue && !isFalse) || (5 > 3);
```

2.7 Assignment Operators

```
1 int x = 10;
2 x += 5; // Equivalent to x = x + 5;
3 x *= 2; // Equivalent to x = x * 2;
```

2.8 Increment Operators

```
1 int foo{2};
2 ++foo; // Pre-increment
3 foo++; // Post-increment
```

Equivalent to:

```
1 int foo{2};
2 foo = foo + 1;
```

```
1 int foo{2};
2 int bar {++foo};
```

```
1 int foo{2};
2 int bar {foo++};
```

```
1 uint8_t foo = 255;
2 std::cout<<static_cast<int>(foo++);
```

```
1 uint8_t foo = 255;
2 std::cout<<static_cast<int>(++foo);
```