Programming Fundamentals with C++

Lecture 3 – Fundamentals - 1

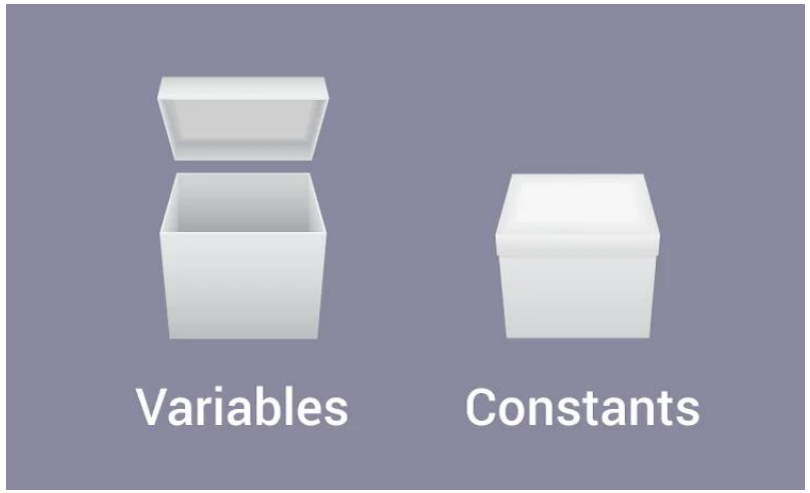
Dr. Muhammad Sajjad

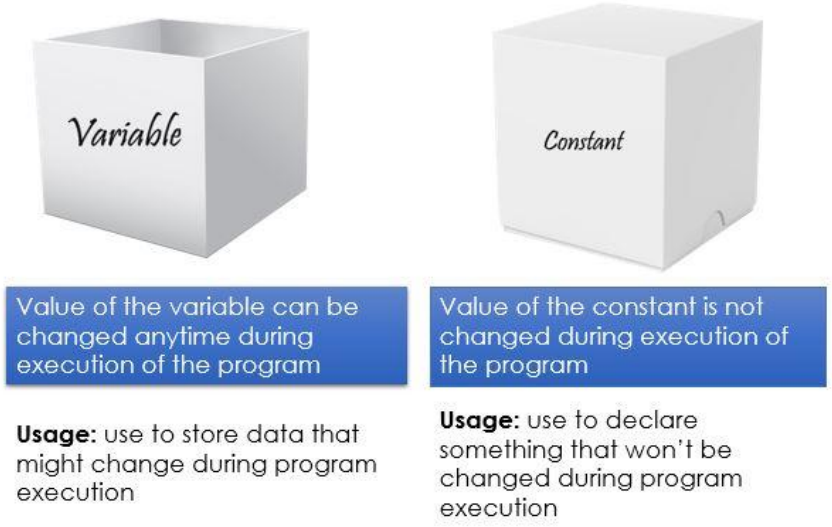
Overview

- **Variables and Constant**
  - What is a variable?
  - What is a constant?
  - Why we use constant
- **Type Conversion**
  - What is Type Conversion?
  - Types
  - Example Code
- **Arithmetic Operators**
  - What are Arithmetic Operators?
  - Examples of Each Operator
  - Notes on Division and Modulus
- **Arithmetic Expression & Order of Precedence**
  - What is Operator Precedence?
  - Operator Precedence Table (Basic)
  - Using Parentheses to Control Order

Variables and Constant

- **Variables**
  - **Definition**: Variables are like containers in a program that hold data values which can be changed as the program runs.
  - **Syntax**: type variable_name = value;
  - **Example**: int age = 20; // 'int' is the type, 'age' is the variable name, and 20 is the value stored
- **Constant**
  - **Definition**: Constants are like variables, but once assigned a value, they cannot be changed. This is useful for values that should stay the same, such as mathematical constants.
  - **Syntax**: const type variable_name = value;
  - **Example**: const float PI = 3.14; // PI will remain 3.14 throughout the program
  - Constants ensure that a value does not accidentally change. For instance, PI is a constant because the value of π doesn't change.

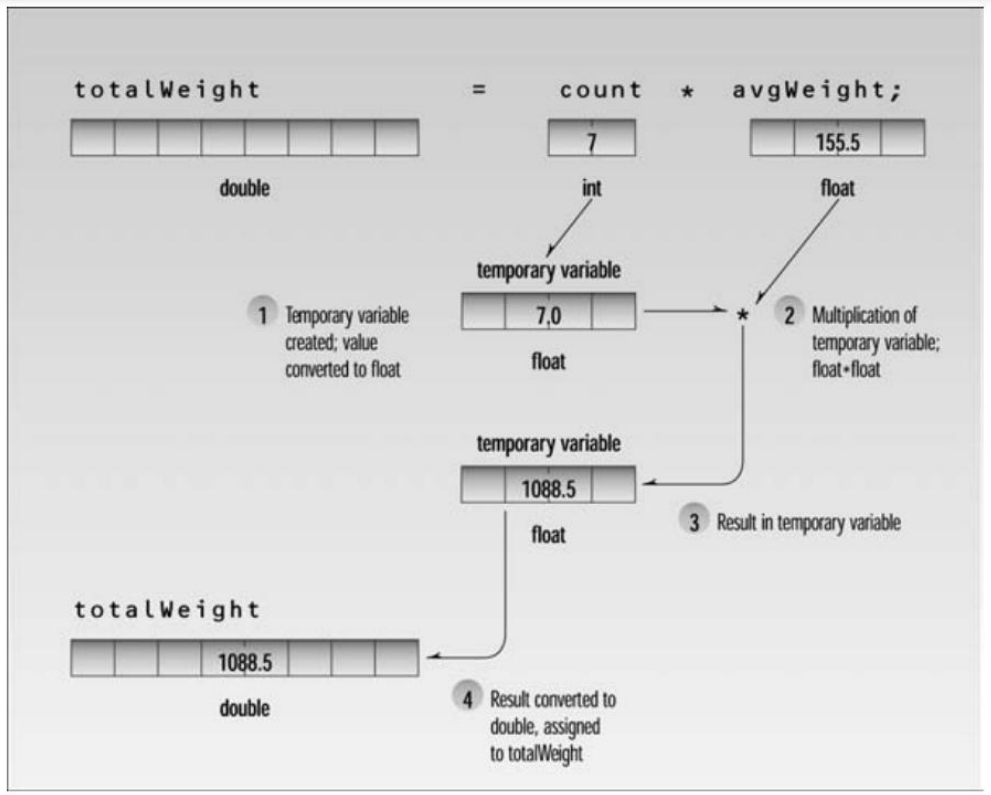




1_Constants.cpp

Type Conversion

- **Definition**
  - Type conversion is the process of converting a variable from one data type to another. This can be **implicit** (automatic) or **explicit** (manual or forced by the programmer).
- **Types of Type Conversion**
  - Type conversion is the process of converting a variable from one data type to another. This can be **implicit** (automatic) or **explicit** (manual or forced by the programmer).
  - **Implicit Conversion**:
    - C++ will automatically convert data types when it makes sense, such as converting a smaller type to a larger one (e.g., int to float).
    - This is often safe but can sometimes lead to precision loss (e.g., from float to int).




```
int x = 5;
float y = x;  // 'x' (int) is implicitly converted to 'y' (float)
```

Type Conversion

- **Implicit Conversion**:
  - **Examples**

2_interger_to_float_implicit.cpp

3_int_to_double_implicit.cpp

4_float_to_double_implicit.cpp

5_char_to_int_implicit.cpp

6_char_to_float_implicit.cpp

7_int_to_double_division_implicit.cpp

Type Conversion

- **Explicit Conversion (Casting)**:
  - The programmer specifically tells C++ to convert a type using casting.
  - **Syntax**: (new_type) variable

- **Why Use Type Conversion?**
  - Type conversion allows mixing different types, like combining integers and decimals.
  - **Example scenario**: Calculating an average where dividing integers might require a float for a precise result.

TABLE 2.4 Order of Data Types

Data Type	*Order*
`long double`	Highest
`double`	
`float`	
`long`	
`int`	
`short`	
`char`	Lowest

```
float x = 7.8;
int y = (int)x;  // 'x' is explicitly cast to 'int', so 'y' becomes 7
```

Type Conversion

- **Explicit Conversion (Casting)**:
  - **Examples**

1_double_to_int_explicit.cpp

2_int_to_double_explicit.cpp

3_int_to_char_explicit.cpp

4_float_to_int_explicit.cpp

5_percentage_example.cpp

Arithmetic Operators

- **What are Arithmetic Operators?**
  - Arithmetic operators perform basic mathematical operations on variables and values.

Common Arithmetic Operators

Operator	Description	Example
`+`	Addition	`5 + 2` → `7`
`-`	Subtraction	`5 - 2` → `3`
`*`	Multiplication	`5 * 2` → `10`
`/`	Division	`5 / 2` → `2` (if integers) or `2.5` (if float)
`%`	Modulus (Remainder)	`5 % 2` → `1`

- **Notes on Division and Modulus?**
  - Division of two integers results in an integer, removing the decimal part. Use float if you need a decimal result.
  - Modulus is used only with integers to get the remainder.

Arithmetic Expression & Order of Precedence

- **What are Arithmetic Expressions?**
  - Arithmetic expressions in C++ involve using operators to perform mathematical calculations on variables and constants. These operations follow standard mathematical rules.
- **Order of Precedence**
  - When combining multiple operators in a single expression, C++ follows a specific order to decide which operations to perform first. This is called *operator precedence*.

1_students_marks_calculator.cpp

2_Modulus_Operator.cpp

3_order_of_precedence.cpp

```cpp
#include <iostream>
using namespace std;

int main() {
    double price = 100.0;
    double discountRate = 0.10;  // 10% discount
    double taxRate = 0.05;       // 5% tax

    // Calculate discounted price
    double discountedPrice = price * (1 - discountRate);  // Apply discount first

    // Calculate total cost with tax applied to discounted price
    double totalCost = discountedPrice * (1 + taxRate);

    cout << "Total cost after discount and tax: $" << totalCost << endl;

    return 0;
}
```

Arithmetic Expression & Order of Precedence

- **Precedence Rules**
  - **Parentheses ():** Operations inside parentheses are performed first.
  - **Multiplication *, Division /, and Modulus %:** These have higher precedence than addition and subtraction.
  - **Addition + and Subtraction - :** These are evaluated last among the basic arithmetic operators.
- **Associativity**
- Operators with the same precedence level are evaluated based on associativity:
- **Left-to-Right Associativity**: Operators having same precedence like (+, -), (*, /, and %) are evaluated from left to right.

```
int result = 5 + 3 * 2; // result is 11, because * has higher precedence than +
```

- To override precedence, use parentheses:

```
int result = (5 + 3) * 2; // result is 16
```

Thank You