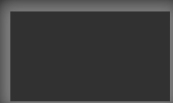


C++ Type Conversion



Type Conversion

- When defining a variable, the value stored in the variable is associated with a certain data type, i.e.,
 - “**int** myIdx {100};”
 - “**float** myLength {100.0f};”
- This association tells the compiler how to interpret the value
- What happens, when you start assigning variables or values to **myIdx**, that are of **different type**, i.e., **float** than **myIdx**?



Type Conversion

```
int myIdx {100};  
float myLength {100.0f};  
  
myIdx = myLength; // what happens?
```

- When compiling the code, the variable defined in (high level) code is translated into machine code, its binary representation
- Binary representation (machine code) of an integer is not the same as machine code of a floating point data type!

Implicit Type Conversion

```
int myIdx {100};  
float myLength {100.0f};  
  
myIdx = myLength; // what happens?
```

- In the above example, the compiler cannot simply assign the float value to the int variable — instead, it **implicitly converts** it from float to int
- **Implicit type conversion works fine for primitive data types** (i.e., int, float, double, char, etc.)

Implicit Type Conversion

```
int firstOperand = 10;  
int secndOperand = 4;  
// What will be the result of this operation:  
float result = firstOperand / secndOperand;
```

?



Implicit Type Conversion

```
int firstOperand = 10;  
int secndOperand = 4;  
// What will be the result of this operation:  
float result = firstOperand / secndOperand;
```

- The result will be “2” because the compiler interprets the variables as integer variables
- To overcome this issue, **explicit type conversion**, or **type casting**, from the integer data type to the floating point data type is required

Explicit Type Conversion

```
int firstOperand = 10;
int secndOperand = 4;
// Introducing the static_cast operator:
float result = static_cast<float>(firstOperand) / secndOperand;
```

- Syntax of the operator:
“`static_cast<data_type>(value_or_var)`”
- Always use the `static_cast<>()` operator to make clear what is happening in the code