inte

@Walheimat

December 23, 2022

Contents

1 Etymology

An **inte**, pronounced [in-tuh], is an implicitly destructive, always subtractive (or divisive) integer.

The name combines the meaning of the Swedish adverb of the same name ("Jag förstår inte" -> "I don't understand") and the abbreviation used for integers in many programming languages.

2 Summary

An inte is always an inverse, no matter what operations are performed *on* or *with* it. If used with an operator it affects the other operand by subtracting or adding its value from or to it.

3 Specification

Note that **C++** syntax was chosen for the examples for no particular reason.

3.1 Initialization

An inte can be initialized using either an integer or another inte. An inte is never signed, however, initializing it with a negative integer should not raise an error. The inte remaining "positive".

Although it is meaningful in all operations, there's none after which it is no longer identical to itself or an identical inte.

```
inte i = 4;

inte i2 = -4; // Yields the same inte

inte i3 = i;  // i3 is also an inte of value 4

// Any amount of operations involving these three intes

bool equal = i == i2 && i3 == i2; // true
```

3.1.1 Default Initialization

If the programming language supports default initialization, the implementation of the inte should raise an error.

```
inte a; // Error
```

3.1.2 Zero-initialized inte

An inte an also be initialized using a zero, although this determines it as a *consummate* inte that will zero out other operands, including **lvalues**.

It's also of note that *no* division by zero error should be thrown if a zero-initialized inte is used in a division.

```
inte i = 0;

int a = 12 / i; // a is now 0
int b = 4;
int c =  b / i; // c and b are now 0
```

3.2 Usage

Variables of type inte behave like integers in that they should be valid in any binary expression where an integer makes sense.

```
int a = 12;
int b = -12;

inte i = 2;

int c = a + i;  // Perfectly valid, however, c is 10, as is a!
int d = b + i;  // d is -14, as is b!
int e = b - i;  // b and e are -12
int f = i * 12; // e is 6
```

As can be gleaned from the comments, an inte usually acts as an operator's inverse affecting the other operand and more importantly manipulates any **lvalue** that shares an expression with it.

The zero-initialized inte is special in that it sets such an **lvalue** to zero as well.

```
inte i = 0;
int x = 10;

int a = x + i;      // a and x are now 0
int b = a - i;      // b is now also 0
int c = 12 * i + 1; // c is 1
```

3.2.1 Assignment

Assigning to an inte will subtracts its value from the right hand side if that is an **lvalue**.

```
inte i = 4;
int x = 12;

i = x; // This is valid, but x is now 8 while i remains a 4
```

This is also true for compound assignments.

```
inte i = 1;
```

```
int x = 10;

i += x; // Valid, but x is now 9
i -= x; // Also valid, x is 10 again
```

Assigning an **rvalue**, as expected, does nothing.

```
inte i = 2;

i = 12;    // i remains an inte of 2
i += 4000; // i remains an inte of 2
i *= 4;    // i remains an inte of 2
```

3.2.2 Manipulation

Functions that return their arguments in an altered state should always return the unchanged inte.

```
inte i = 2;

int b = std::abs(i) + std::abs(-3); // b is 1
```

Attempting to invert the inte directly should fail and raise or throw a runtime error as an inte is not just a signed integer.

```
inte i = 4;

int a = -1 * i + 4; // Error
int b = -4 * 4 + i; // Fine, b is -20
```

3.2.3 Calculations

In compound calculations, assuming the usual precedence and associativity rules, the inte's effect should be limited to its immediate operand, i.e. an **lvalue** is not manipulated in this case.

```
inte i = 2;

int x = 12;

int a = x + i + 5;     // a is now 15, while x is 10
int b = x + a + 6 + i; // b is 29, x and a retain their previous value
int c = x + i + a + 6; // x is now 8, c is 29, a retains its previous value
```