# Operators

There are different types of operators in C++. More detail can be studied at this article.

## Various Operators

### 1. Arithmetic

```
void arithmatic() {
  int x = 0;

  x = x + 1;
  x = x - 1;
  x = x * 2;
  x = x / 3;
  x = x % 2;

  x++;
  x--;}
```

Bitwise Operators

```
void bitwise() {
  int x = 0;

  x = x | 0b0001;
  x = x & 0b0000;
  x = x ^ 0b0001;
}
void great_example_of_xor_operator() {
```

```
  int a = 1;
  int b = 0;

  // Check if a and b are exclusive from each other
  if ((a == 1 && b == 0) || (a == 0 && b == 1)) {
    // ...
  }

  // We can actually do this:
  if (a xor b)
  // or
  if (a ^ b)}
```

## 2. Assignment

```
void assignment() {
  int x = 0;

  x = x + 3; // full form
  x += 3;    // shortcut
  x -= 3;
  x *= 3;
  x /= 3;
  x %= 3;
  x &= 3;
  x |= 3;
  x ^= 3;
  x >>= 3;}
```

## 3. Comparison

```
void comparison() {
  int x = 0;
  int y = 1;
```

```
  if (x == y)
  if (x != y)
  if (x > y)
  if (x < y)
  if (x >= y)
  if (x <= y)}
```

## 4. Logical

```
void logical() {
  int x = 0;
  int y = 1;

  if (x == 1 && y == 1)
  if (x == 1 || y == 1)
  if (! (x == 1 && y == 1) )}
```

# Operator Overloading

The operators would be boring if they were only applied to integers as demonstrated in the examples above. We can actually inform the C++ compiler what operators should do for our classes. Let's reuse the Vector of integers we built before and define some interesting operators.

```
// file: vector.hh
class Vector {
private:
    int* m_array;   // Pointer to dynamically allocated array
    int m_max_size; // Max size of the vector
    int m_size;     // Current size of the vector
public:
    Vector(int max_size);
    ~Vector();

    bool push_back(int value);
    int pop_back();
```

```
    Vector operator+=(const Vector& other) const; // operator +=
    Vector operator+(const Vector& other) const; // operator +
    bool operator==(const Vector& other) const;  // operator==
    bool operator!=(const Vector& other) const;  // operator!=

    Vector operator*=(int multiply_with); // * operator to multiply all integers by a number

    // ...
};
// operator+ definition
Vector Vector::operator+=(const Vector& other) const {
    // allocate memory that can hold data from both vectors
    Vector result(this->m_max_size + other.m_max_size);

    for (int i = 0; i < m_size; ++i) {
        result.push_back(m_array[i]);
    }
    for (int i = 0; i < other.m_size; ++i) {
        result.push_back(other.m_array[i]);
    }

    return result;
}
Vector Vector::operator+(const Vector& other) const {
    Vector result(*this);
    result += other;
    return result;
}
// operator== definition
bool Vector::operator==(const Vector& other) const {
    const bool is_equal = true;
    if (m_size != other.m_size) {
        return !is_equal;
    }
    for (int i = 0; i < m_size; ++i) {
        if (m_array[i] != other.m_array[i]) {
```

```
            return !is_equal;
        }
    }
    return is_equal;
}
// operator!= definition
bool Vector::operator!=(const Vector& other) const {
    return !(*this == other);
}
Vector operator*=(int multiply_with) {
  for (int i = 0; i < m_size; ++i) {
    m_array[i] *= multiply_with;
  }
  return *this;}
```

Here is how the operators may be used:

```
void vector_plus_operator_example() {
  puts("Let's practice strings");
  Vector v1(6);
  v1.push_back(1);
  v1.push_back(2);
  v1.push_back(3);
  Vector v2(3);
  v1.push_back(4);
  v1.push_back(5);
  v1.push_back(6);
  // Use our operator to add contents of two vectors
  v1 = v1 + v2;
  v1.print();
}
void vector_multiply_operator_example() {
  Vector v1(6);
  v1.push_back(1);
  v1.push_back(2);
  v1.push_back(3);
```

```
  // Multiply operator in action
  v1 *= 5;
  v1.print();}
```

# Exercises

## Vector library operators

Let's implement a few more operators for your vector library. Typically, the `[ ]` operator is implemented such that it returns a reference to one of the elements of the vector, but in our case, we will return a read-only value.

```
class Vector {
private:
    int* m_array;   // Pointer to dynamically allocated array
    int m_max_size; // Max size of the vector
    int m_size;     // Current size of the vector
public:
    Vector(int max_size);
    ~Vector();

    // First implement an "at()" API
    // Return an element at a particular index
    int at(int index);

    // Secondly, implement the [] operator:
    const int operator[](int index);}
```

## String library operators

Implement the following string operators, and also write unit-test code in `main.cpp` to test that the code you wrote actually functions correctly.

```
class string
{
    std::unique_ptr<char[]> m_string;
    int m_max_length;
```

```
public:
    string(int max_length);
    string(const char *c_string);

    // -------------------------------------
    // Implement the following operators
    // -------------------------------------

    // Adds two strings together
    string operator+=(const string& other) const;
    // "hello world" - "world" = "hello"
    string operator-=(const string& other) const;
    // "hello" * 2 = "hellohello"
    string operator*=(int how_many_times) const;
    // Implement shift operators to trim beginning or end of string
    // 0b1101 >> 1 ==> 0b0110
    // "hello" >>= 1 ==> "hell"
    // "hello" >>= 3 ==> "he"
    string operator>>=(int shift_right_by) const;
    // similar to python for slice operation
    string operator<<=(int shift_right_by) const;
    // All comparison operators are applicable
    // string s1("hello"); string s2("world")
    // if (s1 == s2)
    // if (s1 != s2)
    bool operator!=(const string &compare_with) const {
        return !(*this == compare_with);
    }
    bool operator==(const string &compare_with) const {
        // todo
        return false;
    }

    // Implement comparison operators
    bool operator<=(const string &compare_with) const;
    bool operator>=(const string &compare_with) const;
```

```
    bool operator<(const string &compare_with) const;
    bool operator>(const string &compare_with) const;
    bool operator!=(const char* compare_with) const;
    bool operator==(const char* compare_with) const;
}
```

---

Revision #24
Created 10 months ago by Preet Kang
Updated 9 months ago by Preet Kang