سلسلة تعلم البرمجة بلغة C++ الحديثة

Learn Modern C++ Programming Course

إعداد المهندس أحمد الديب

#38: Three-Way Comparison

strcmp()

```
int compare(const char* a, const char* b) {
  //  A negative return value means less-than,
  // 0 means equal,
  // and a positive value means greater-than.
  return std::strcmp(a, b);
}


int main() {
  std::cout << compare("B", "A") << "\n";
  std::cout << compare("A", "A") << "\n";
  std::cout << compare("A", "B") << "\n";
}
```

```
 1
 0
-1
```

C++20

Equality Operator

```
template <typename T>
class Number {
 public:
  Number(T value) : _value{value} {}
  bool operator==(const Number&) const = default;


 private:
  T _value;
};


int main() {
  //
  std::cout << std::boolalpha << '\n';
  Number num1{2020}, num2{1010};
  std::cout << (num1 == num2) << "\n";
  std::cout << (num1 != num2) << "\n";
}
```

When you define or request the equality operator from the compiler with = default, you automatically get the equality and inequality operators: ==, and !=.

Source: A Tour of C++ (3th Edition), Bjarne Stroustrup

operator <=>

```
template <typename T>
class Number {
 public:
  Number(T value) : _value{value} {}
  auto operator<=>(const Number&) const = default;


 private:
  T _value;
};


int main() {
  //
  std::cout << std::boolalpha << '\n';
  Number num1{2020}, num2{1010};
  std::cout << (num1 == num2) << "\n";
  std::cout << (num1 >= num2) << "\n";
  std::cout << (num1 <= num2) << "\n";
  //  ...
}
```

The three-way comparison operator <=>, or spaceship operator, determines, for two values A and B, whether A < B, A == B, or A > B.

By declaring the three-way comparison operator default, the compiler will attempt to generate a consistent relational operator for the class. In this case, you get all six comparison operators: ==, !=, <, <=, >, and >=.

Source: C++20 Get the Details, Rainer Grimm

User Defined operator <=>

```
template <typename T>
class Number {
 public:
  Number(T value) : _value{value} {}
  auto operator<=>(const Number& other) const {
    return _value <=> other._value;
  }


 private:
  T _value;
};


int main() {
  //
  std::cout << std::boolalpha << '\n';
  Number num1{2020}, num2{1010};
  std::cout << (num1 == num2) << "\n"; // error
  std::cout << (num1 >= num2) << "\n";
  std::cout << ((num1 <=> num2) > 0) << "\n";

  //  ...
}
```

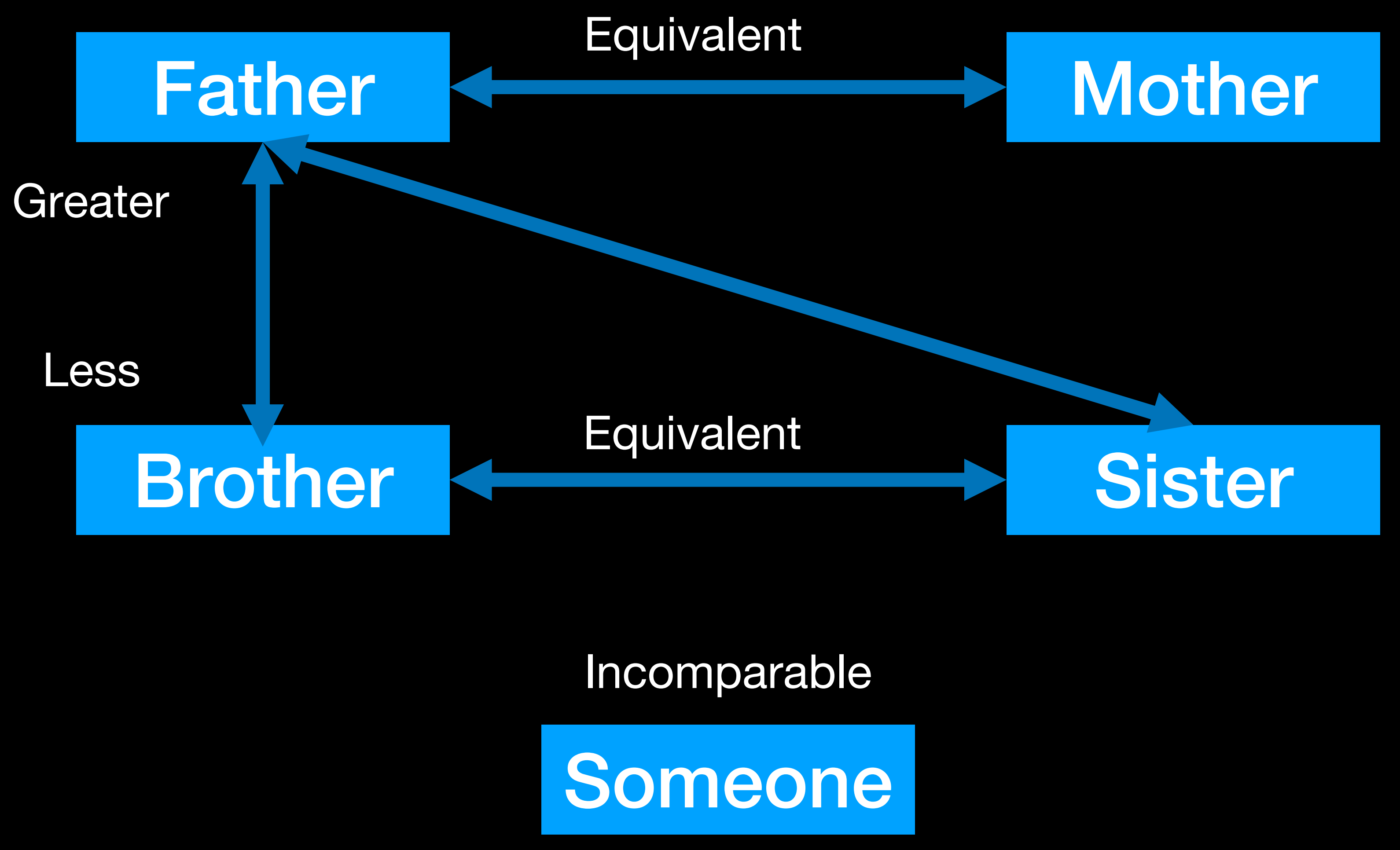
Note: The compiler-generated comparison operator compares the pointers but not the referenced objects.

Source: C++20 Get the Details, Rainer Grimm

Comparison Categories

- When the default semantics are not suitable, such as when the members must be compared out of order, or must use a comparison that's different from their natural comparison, then the programmer can write operator<=> and let the compiler generate the appropriate two-way comparison operators.

Return type	Equivalent values are..	Incomparable values are..
`std::strong_ordering`	`indistinguishable`	`not allowed`
`std::weak_ordering`	`distinguishable`	`not allowed`
`std::partial_ordering`	`distinguishable`	`allowed`



Source: C++20 Get the Details, Rainer Grimm

Source: https://en.cppreference.com/w/cpp/language/default_comparisons

C++20

Comparison Categories

```
class Person {
 public:
  auto operator<=>(const Person& other) const {
    if (is_equivalent(other)) return std::partial_ordering::equivalent;
    if (is_parent_of(other)) return std::partial_ordering::greater;
    if (other.is_parent_of(*this)) return std::partial_ordering::less;
    return std::partial_ordering::unordered;
  }
  //...

};


int main() {
  //
  if (father > son) {  // or (father <=> son) > 0
    std::cout << "father is greater than son” << "\n";
  }
  if (son < father) {  // (son <=> father) < 0
    std::cout << "son is less than father” << "\n";
  }
  if (std::is_eq(son <=> daughter)) {  // or (son <=> daughter) == 0
    std::cout << "son is equivalent to daughter” << "\n";
  }
  if (std::is_neq(other <=> daughter)) {
    std::cout << "other is no related to daughter” << "\n";
  }
}
```

Thank you