```
// Assignment operators:
Type& operator&=(Type& lhs, const Type& rhs); // Assign bitwise and
Type& operator^=(Type& lhs, const Type& rhs); // Assign exclusive or
Type& operator|=(Type& lhs, const Type& rhs); // Assign bitwise or
Type& operator-=(Type& lhs, const Type& rhs); // Assign difference
Type& operator<<=(Type& lhs, const Type& rhs); // Assign left shift
Type& operator*=(Type& lhs, const Type& rhs); // Assign product
Type& operator/=(Type& lhs, const Type& rhs); // Assign quotient
Type& operator%=(Type& lhs, const Type& rhs); // Assign remainder
Type& operator>>=(Type& lhs, const Type& rhs); // Assign right shift
Type& operator+=(Type& lhs, const Type& rhs); // Assign sum
//Other modification operators
Type& operator--(Type& lhs); // Prefix decrement - decrement and return new value
Type operator--(Type& lhs, int unused); // Postfix decrement - decrement and retur
Type& operator++(Type& lhs); // Prefix increment - increment and return new value
Type operator++(Type& lhs, int unused); // Postfix increment - increment and retur

//Comparison operators
bool operator==(const Type& lhs, const Type& rhs); // Equal
bool operator>(const Type& lhs, const Type& rhs); // Greater than
bool operator>=(const Type& lhs, const Type& rhs); // Greater than or equal
bool operator<(const Type& lhs, const Type& rhs); // less than
bool operator<=(const Type& lhs, const Type& rhs); // less than or equal
bool operator!(const Type& lhs); // logical complement
bool operator!=(const Type& lhs, const Type& rhs); // no equal

//Other operators
Type operator+(const Type& lhs, const Type& rhs); // Addition
Type operator+(const Type& lhs); // Unary plus
Type operator-(const Type& lhs, const Type& rhs); // Subtraction
Type operator-(const Type& lhs); // Unary minus
ContainedType* operator&(const Type& lhs); // Address of
Type operator&(const Type& lhs, const Type& rhs); // Bitwise and
Type operator~(const Type& lhs, const Type& rhs); // Bitwise complement
Type operator^(const Type& lhs, const Type& rhs); // Bitwise exclusive or
Type operator|(const Type& lhs, const Type& rhs); // Bitwise or
Type operator/(const Type& lhs, const Type& rhs); // Division
Type operator<<(const Type& lhs, const Type& rhs); // Left shift
Type operator*(const Type& lhs, const Type& rhs); // Multiplication
ContainedType& operator*(const Type& lhs); // Dereference
Type operator%(const Type& lhs, const Type& rhs); // Remainder
Type operator>>(const Type& lhs, const Type& rhs); // Right shift


class Type {
    // Overloads which must be member functions
    ContainedType& operator[](const IndexType& index); // Array subscript
    Type& operator=(const Type& rhs); // Assignment
    ContainedType& operator->*(); // Member reference
    const ContainedType& operator->*() const; // Member reference
    ContainedType& operator->(); // Member reference
    const ContainedType& operator->() const; // Member reference
```

```
    // Assignment operators
    Type& operator&=(const Type& rhs); // Assign bitwise and
    Type& operator^=(const Type& rhs); // Assign exclusive or
    Type& operator|=(const Type& rhs); // Assign bitwise or
    Type& operator-=(const Type& rhs); // Assign difference
    Type& operator<<=(const Type& rhs); // Assign left shift
    Type& operator*=(const Type& rhs); // Assign product
    Type& operator/=(const Type& rhs); // Assign quotient
    Type& operator%=(const Type& rhs); // Assign remainder
    Type& operator>>=(const Type& rhs); // Assign right shift
    Type& operator+=(const Type& rhs); // Assign sum

    //Other modification operators
    Type& operator--(Type& lhs); // Prefix decrement - decrement and return new va
    Type operator--(Type& lhs, int unused); // Postfix decrement - decrement and r
    Type& operator++(); // Prefix increment - increment and return new value
    Type operator++(int unused); // Postfix increment - increment and return copy

    //Comparison operators
    bool operator==(const Type& rhs) const; // Equal
    bool operator>(const Type& rhs) const; // Greater than
    bool operator>=(const Type& rhs) const; // Greater than or equal
    bool operator<(const Type& rhs) const; // Less than
    bool operator<=(const Type& rhs) const; // Less than or equal
    bool operator!=(const Type& rhs) const; // Not equal

    //Other operators
    Type operator+(const Type& rhs) const; // Addition
    Type operator+() const; // Unary plus
    Type operator-(const Type& rhs) const; // Subtraction
    Type operator-() const; // Unary minus
    ContainedType* operator&(); // Address of
    const ContainedType* operator&() const; // Address of
    Type operator&(const Type& rhs) const; // Bitwise and
    Type operator~(const Type& rhs) const; // Bitwise complement
    Type operator^(const Type& rhs) const; // Bitwise exclusive or
    Type operator|(const Type& rhs) const; // Bitwise or
    ContainedType& operator*(); // Dereference
    const ContainedType& operator*() const; // Dereference
    Type operator/(const Type& rhs) const; // Division
    Type operator<<(const Type& rhs) const; // Left shift
    bool operator!() const; // Logical complement
    Type operator*(const Type& rhs) const; // Multiplication
    Type operator%(const Type& rhs) const; // Remainder
    Type operator>>(const Type& rhs) const; // Right shift
};
```

```
PointerType operator->() const { return pointee_; }

ReferenceType operator*() const { return *pointee_; }
```