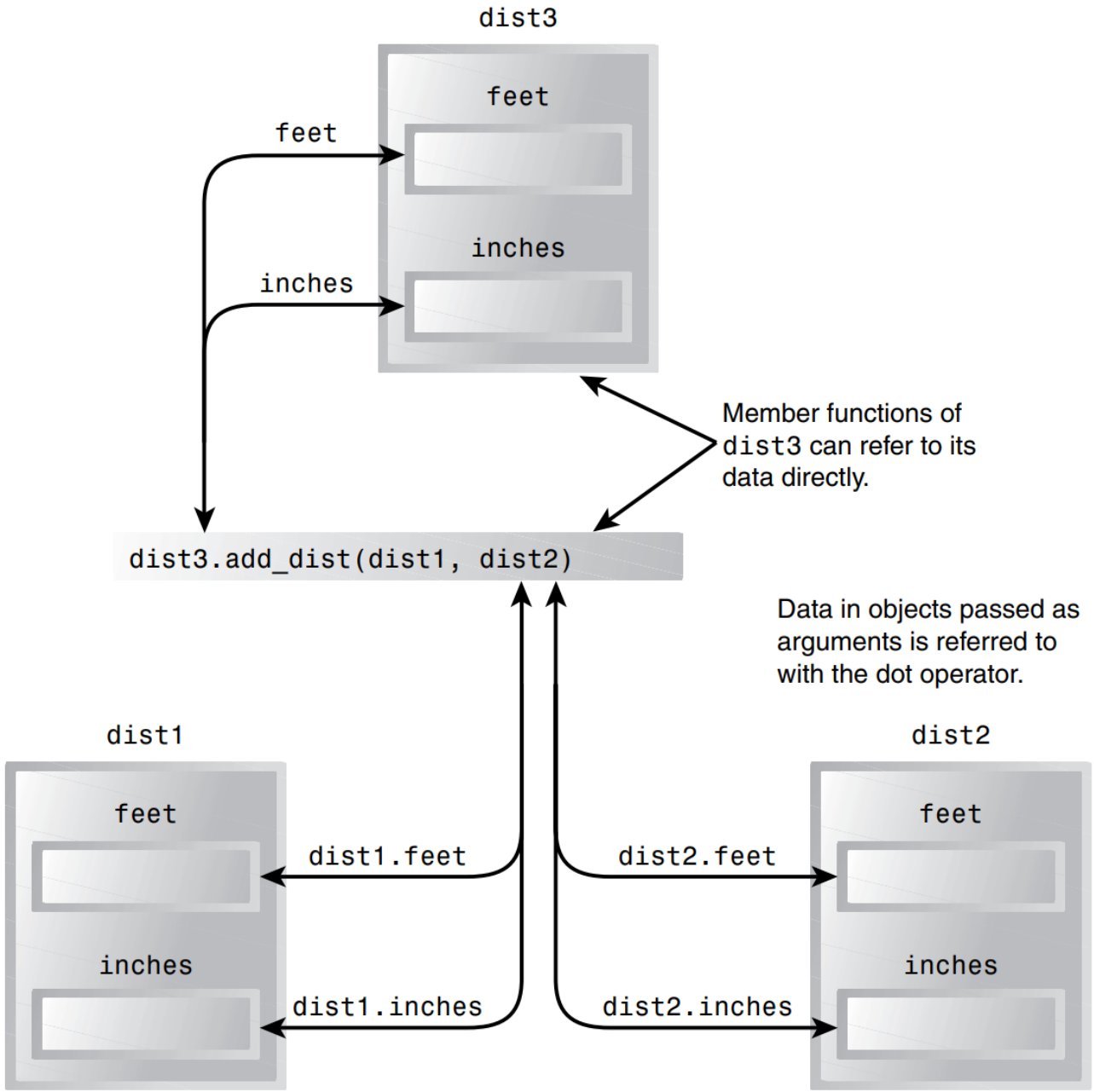
Object Oriented programming – Operator Overloading

Objects as Function Arguments and return data type

- Objects are user defined datatypes of a sort.
- Hence they can be passed as function arguments
- On a side note, we use constructor like this
  - Constructor()
  - { count = 0; }
- But its not a preferred way. Its better to initialize the variables like this instead
  - Constructor() : count(0)
  - Where count is int data member in Constructor class.


dist3
feet
feet
inches
inches
Member functions of dist3 can refer to its data directly.
dist3.add_dist(dist1, dist2)
Data in objects passed as arguments is referred to with the dot operator.
dist1
feet
inches
dist1.feet
dist1.inches
dist2
feet
inches
dist2.feet
dist2.inches

Overloading Unary Operators

- Unary operators take only one argument
- **The operator Keyword**
  - Used to teach a normal C++ operator to act on a user-defined operand.
  - Syntax: void operator ++ ()
  - Syntax order: Return type, keyword operator, operator (++), empty argument list.
- **Operator arguments**
  - Unary operator overloading typically doesn't require any arguments in the function signature.
  - Since unary operators operate on the object itself, they can directly access the member variables or state of the object they are applied to.

Overloading Unary Operators (cont)

- **Operator Return values**
  - The return type of the overloaded operator determines the behavior after the operator is applied.
  - For example, if the ++ operator is overloaded to return **void**, it implies that the operation modifies the **object itself** without returning any value.
  - Overloading allows custom behavior for objects, enabling operations tailored to their specific characteristics.
  - This customization can involve modifying the internal state of the object or even returning a new object with modified state, depending on the requirements of the application.

Overloading binary operators

- Overloading binary operators in C++ allows custom behavior for operators like addition (+), subtraction (-), multiplication (*), etc., for objects of a class. Here's a brief overview:
- **Operator Keyword and Syntax:**
  - Binary operators can be overloaded using the operator keyword followed by the specific binary operator being overloaded.
  - **Syntax: return_type operator op (parameters),** where op represents the binary operator being overloaded and parameters represent the arguments passed to the operator.

Overloading binary operators (cont’)

- **Operator Arguments:**
  - Binary operator overloading typically takes one or two arguments, depending on the operator being overloaded.
  - For binary operators, one argument is often the object itself (this) and the other argument is the operand being used in the operation.
  - These arguments can be passed by value, reference, or const reference depending on the specific requirements of the operation.
- **Return Type and Overloading for an Object:**
  - The return type of the overloaded operator determines the behavior after the operator is applied.
  - Overloading allows custom behavior for objects, enabling operations tailored to their specific characteristics.
  - This customization can involve modifying the internal state of the object, returning a new object with modified state, or performing any other operation relevant to the class.

guidelines for Operator Overloading

- Use Similar Meanings
- Use Similar Syntax
  - alpha += beta; should work the same as alpha = alpha + beta; where + is overloaded
- Show Restraint
- Avoid Ambiguity
- Not All Operators Can Be Overloaded
  - member access or dot operator (.)
  - the scope resolution operator (::)
  - conditional operator (?:)
  - the pointer-to-member operator (->)
  - Also you cant create new operators s (like *&) and try to overload them

task

- 1. Operator overloading is a. making C++ operators work with objects. b. giving C++ operators more than they can handle. c. giving new meanings to existing C++ operators. d. making new C++ operators.
- 2. Assuming that class X does not use any overloaded operators, write a statement that subtracts an object of class X, x1, from another such object, x2, and places the result in x3.