

SRS Setup

Login: student.turningtechnologies.com

Session ID: 20220223<A|D>

Replace <A|D> with this section's letter

Copy Control II

CS 2124: Object Oriented Programming
Darryl Reeves, Ph.D.

Agenda

- The assignment operator
- In-class problem
- Vector class design

The assignment operator

—

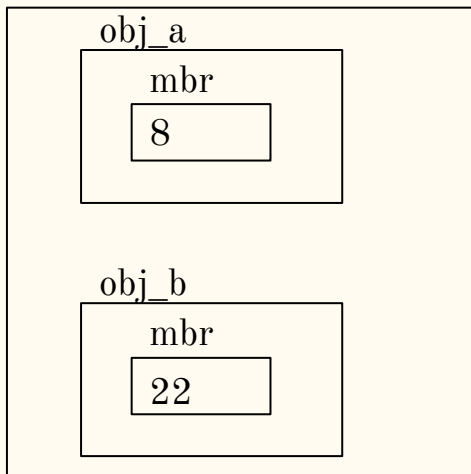
Big 3's last member: assignment operator

- key difference between copy constructor and assignment operator
 - copy constructor *initializes* an object

```
1  SomeClass obj_a;  
   SomeClass obj_b(obj_a);  
  
2  SomeClass obj_a;  
   SomeClass obj_b = obj_a; initial value
```

Big 3's last member: assignment operator

- key difference between copy constructor and assignment operator
 - copy constructor *initializes* an object
 - assignment operator *modifies* an object



```
class SomeClass {
```

```
public:
```

```
    SomeClass(int val): mbr(val) {}
```

```
private:
```

```
    int mbr;
```

```
}
```

```
int main() {
```

```
    SomeClass obj_a(8);
```

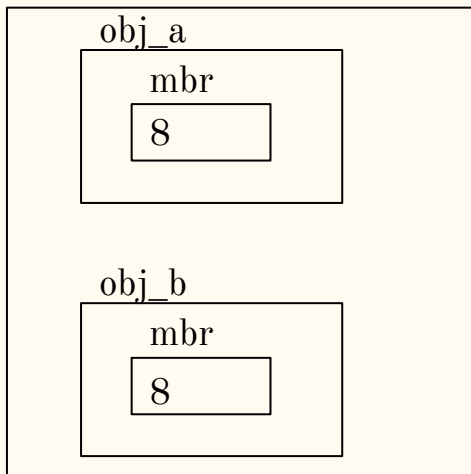
```
    SomeClass obj_b(22);
```

```
    obj_b = obj_a;
```

```
}
```

Big 3's last member: assignment operator

- key difference between copy constructor and assignment operator
 - copy constructor *initializes* an object
 - assignment operator *modifies* an object



```
class SomeClass {  
    public:  
        SomeClass(int val): mbr(val) {}  
    private:  
        int mbr;  
}
```

```
int main() {  
    SomeClass obj_a(8);  
    SomeClass obj_b(22);  
    obj_b = obj_a;  
}
```

Big 3's last member: assignment operator

- key difference between copy constructor and assignment operator
 - copy constructor *initializes* an object
 - assignment operator *modifies* an object
- responsibilities of assignment operator
 - get rid of previous values *same as destructor*
 - replace previous values
 - copy values from object on "right-hand side" of operator
 - return value of proper type
 - check for self-assignment

} *new tasks*

} *same as copy constructor*
- implemented as a member function

Return value of proper type

```
int x_val, y_val, z_val;  
z_val = 10;  
x_val = y_val = z_val; supported in C++
```

↑
equivalent to
↓

```
x_val = (y_val = z_val);
```

must evaluate to
same value as
assigned to y_val

assignment operator returns a
reference of the same type as the class

Check for self-assignment

x_val = x_val; *allowed in C++*

need to ensure assignment
evaluates to object

when **memory addresses** are
the same

- responsibilities of assignment operator
 - ~~get rid of previous values~~
 - ~~replace previous values~~
 - ~~copy values from object on "right-hand side" of operator~~
 - return value of proper type
 - check for self-assignment

Implementing the assignment operator

```
class SimpleClass {  
public:  
    SimpleClass() { ptr = new int(17); }  
    SimpleClass(const SimpleClass& rhs) {  
        ptr = new int(*rhs.ptr);  
    }  
    ~SimpleClass() { delete ptr; }  
    ___ operator= (___ ___ rhs) { }  
  
private:  
    int* ptr;  
  
};
```

Implementing the assignment operator

```
class SimpleClass {
public:
    SimpleClass() { ptr = new int(17); }
    SimpleClass(const SimpleClass& rhs) {
        ptr = new int(*rhs.ptr);
    }
    ~SimpleClass() { delete ptr; }

    ___ operator= (___ ___ rhs) {

    }

private:
    int* ptr;

};
```

Implementing the assignment operator

```
class SimpleClass {  
public:  
    SimpleClass() { ptr = new int(17); }  
    SimpleClass(const SimpleClass& rhs) {  
        ptr = new int(*rhs.ptr);  
    }  
    ~SimpleClass() { delete ptr; }  
  
    ___ operator= (___ SimpleClass& rhs) {  
  
    }  
  
private:  
    int* ptr;  
  
};
```

Implementing the assignment operator

```
class SimpleClass {
public:
    SimpleClass() { ptr = new int(17); }
    SimpleClass(const SimpleClass& rhs) {
        ptr = new int(*rhs.ptr);
    }
    ~SimpleClass() { delete ptr; }

    ___ operator= (const SimpleClass& rhs) {

    }

private:
    int* ptr;

};
```

Implementing the assignment operator

```
class SimpleClass {  
public:  
    SimpleClass() { ptr = new int(17); }  
    SimpleClass(const SimpleClass& rhs) {  
        ptr = new int(*rhs.ptr);  
    }  
    ~SimpleClass() { delete ptr; }  
  
    SimpleClass& operator= (const SimpleClass& rhs) {  
  
    }  
  
private:  
    int* ptr;  
  
};
```

Implementing the assignment operator

```
class SimpleClass {  
public:  
    SimpleClass() { ptr = new int(17); }  
    SimpleClass(const SimpleClass& rhs) {  
        ptr = new int(*rhs.ptr);  
    }  
    ~SimpleClass() { delete ptr; }  
  
    SimpleClass& operator= (const SimpleClass& rhs) {  
  
    }  
  
private:  
    int* ptr;  
  
};
```

Implementing the assignment operator

```
class SimpleClass {
public:
    SimpleClass() { ptr = new int(17); }
    SimpleClass(const SimpleClass& rhs) {
        ptr = new int(*rhs.ptr);
    }
    ~SimpleClass() { delete ptr; }

    SimpleClass& operator= (const SimpleClass& rhs) {
        if (this != &rhs) {

        }
        return *this;
    }

private:
    int* ptr;

};
```


Implementing the assignment operator

```
class SimpleClass {
public:
    SimpleClass() { ptr = new int(17); }
    SimpleClass(const SimpleClass& rhs) {
        ptr = new int(*rhs.ptr);
    }
    ~SimpleClass() { delete ptr; }

    SimpleClass& operator= (const SimpleClass& rhs) {
        if (this != &rhs) {
            // free up resources (as needed)
            delete ptr;
            // allocate new resources (as needed)
            ptr = new int;
            // copy over all data
            *ptr = *rhs.ptr;
        }
        return *this;
    }

private:
    int* ptr;
};
```

*handles
self-assignment*

sames as destructor

*sames as copy
constructor*

correct type and value

```
SimpleClass obj1;
```

```
SimpleClass obj2;
```

```
obj1.operator=(obj2);
```

*not usually
written like this*

equivalent to

```
obj1 = obj2;
```

Implementing the assignment operator

```
class SimpleClass {
public:
    SimpleClass() { ptr = new int(17); }
    SimpleClass(const SimpleClass& rhs) {
        ptr = new int(*rhs.ptr);
    }
    ~SimpleClass() { delete ptr; }

    SimpleClass& operator= (const SimpleClass& rhs) {
        if (this != &rhs) {
            // free up resources (as needed)
            delete ptr;
            // allocate new resources (as needed)
            ptr = new int;
            // copy over all data
            *ptr = *rhs.ptr;
        }
        return *this;
    }

private:
    int* ptr;
};
```

*handles
self-assignment*

sames as destructor

*sames as copy
constructor*

correct type and value

```
SimpleClass obj1;
SimpleClass obj2;

obj1 = obj2;
```

In-class problem

Adding an assignment operator

```
class Thing {  
    friend ostream& operator<<(ostream& os, const Thing& rhs) {  
        return os << "Thing: " << *rhs.i_ptr;  
    }  
public:  
    Thing(int val) : { i_ptr = new int(val); }  
  
    Thing(const Thing& another_thing) : name(another_thing.name) {  
        i_ptr = new int( *another_thing.i_ptr );  
    }  
  
    void set_value(int val) { *i_ptr = val; }  
    int get_value() const { return *i_ptr; }  
  
    ~Thing() { delete i_ptr; }  
  
private:  
    int* i_ptr;  
};
```

Adding an assignment operator

```
class Thing {  
    friend ostream& operator<<(ostream& os, const Thing& rhs);  
public:  
    Thing(int val) { i_ptr = new int(val); }  
  
    Thing(const Thing& another_thing) {  
        i_ptr = new int( *another_thing.i_ptr );  
    }  
  
    void set_value(int val) { *i_ptr = val; }  
    int get_value() const { return *i_ptr; }  
  
    ~Thing() { delete i_ptr; }  
  
private:  
    int* i_ptr;  
    // add a member of non-primitive type  
};
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val) { i_ptr = new int(val); }

    Thing(const Thing& another_thing) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

private:
    int* i_ptr;
    // add a member of non-primitive type
    string name;
};
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) { i_ptr = new int(val); }

    Thing(const Thing& another_thing) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

private:
    int* i_ptr;
    string name;
};
```

Adding an assignment operator

```
class Thing {  
    friend ostream& operator<<(ostream& os, const Thing& rhs);  
public:  
    Thing(int val, const string& name) {  
        i_ptr = new int(val);  
    }  
  
    Thing(const Thing& another_thing) {  
        i_ptr = new int( *another_thing.i_ptr );  
    }  
  
    void set_value(int val) { *i_ptr = val; }  
    int get_value() const { return *i_ptr; }  
  
    ~Thing() { delete i_ptr; }  
  
private:  
    int* i_ptr;  
    string name;  
};
```


Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

private:
    int* i_ptr;
    string name;
};
```

Adding an assignment operator

```
class Thing {  
    friend ostream& operator<<(ostream& os, const Thing& rhs);  
public:  
    Thing(int val, const string& name) : name(name) {  
        i_ptr = new int(val);  
    }  
  
    Thing(const Thing& another_thing) : name(____) {  
        i_ptr = new int( *another_thing.i_ptr );  
    }  
  
    void set_value(int val) { *i_ptr = val; }  
    int get_value() const { return *i_ptr; }  
  
    ~Thing() { delete i_ptr; }  
  
private:  
    int* i_ptr;  
    string name;  
};
```

Adding an assignment operator

```
class Thing {  
    friend ostream& operator<<(ostream& os, const Thing& rhs);  
public:  
    Thing(int val, const string& name) : name(name) {  
        i_ptr = new int(val);  
    }  
  
    Thing(const Thing& another_thing) : name(_1_) {  
        i_ptr = new int( *another_thing.i_ptr );  
    }  
  
    void set_value(int val) { *i_ptr = val; }  
    int get_value() const { return *i_ptr; }  
  
    ~Thing() { delete i_ptr; }  
  
private:  
    int* i_ptr;  
    string name;  
};
```

Adding an assignment operator

```
class Thing {  
    friend ostream& operator<<(ostream& os, const Thing& rhs);  
public:  
    Thing(int val, const string& name) : name(name) {  
        i_ptr = new int(val);  
    }  
  
    Thing(const Thing& another_thing) : name(_1_) {  
        i_ptr = new int( *another_thing.i_ptr );  
    }  
  
    void set_value(int val) { *i_ptr = val; }  
    int get_value() const { return *i_ptr; }  
  
    ~Thing() { delete i_ptr; }  
  
private:  
    int* i_ptr;  
    string name;  
};
```

TurningPoint

SRS Setup

Login: student.turningtechnologies.com

Session ID: 20220223<A|D>

Replace <A|D> with this section's letter

Which expression replaces blank #1 to initialize the name of the current object in the copy constructor?

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(_1_) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

private:
    int* i_ptr;
    string name;
};
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

private:
    int* i_ptr;
    string name;
};
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

private:
    int* i_ptr;
    string name;
};

int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    delete thing_ptr;
}
```


Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

private:
    int* i_ptr;
    string name;
};

int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    // assign Thing named "curly" to thing2

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

private:
    int* i_ptr;
    string name;
};

int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = ___; // assign Thing named "curly" to thing2

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

private:
    int* i_ptr;
    string name;
};

int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *_thing_ptr; // assign Thing named "curly" to thing2

    delete thing_ptr;
}
```

Which expression replaces blank #2 in order to assign the Thing pointed to by thing_ptr to thing2?

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

private:
    int* i_ptr;
    string name;
};

int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = _2_; // assign Thing named "curly" to thing2

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

private:
    int* i_ptr;
    string name;
};

int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

What needs to be added to the Thing class implementation to avoid the compilation error?

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

private:
    int* i_ptr;
    string name;
};

int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    // implement the assignment operator member function

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```


Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    // implement the assignment operator member function
    --- ----(---) { }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    // implement the assignment operator member function
    --- _3_(---) { }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

What name is given to the member function that implements the assignment operator for a class (replacing blank #3)?

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    // implement the assignment operator member function
    --- _3_(---) { }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    // implement the assignment operator member function
    ___ operator=(___ rhs) { }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    // implement the assignment operator member function
    ___ operator=(_4_ rhs) { }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

What replaces blank #4 to complete the parameter declaration for the operator= member function?

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    // implement the assignment operator member function
    ___ operator=(_4_ rhs) { }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    // implement the assignment operator member function
    ___ operator=(const Thing& rhs) { }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    // implement the assignment operator member function
    _5_ operator=(const Thing& rhs) { }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```


Which return type needs to be declared for the operator= member function for the Thing class (replacing blank #5)?

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    // implement the assignment operator member function
    _5_ operator=(const Thing& rhs) { }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    // implement the assignment operator member function
    Thing& operator=(const Thing& rhs) { }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    // implement the assignment operator member function
    Thing& operator=(const Thing& rhs) {
        // check for self-assignment
        // free resources (if needed)
        // allocate memory (if needed)
        // copy values from rhs object
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        // check for self-assignment
        // free resources (if needed)
        // allocate memory (if needed)
        // copy values from rhs object
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        // check for self-assignment
        ---
        // free resources (if needed)
        // allocate memory (if needed)
        // copy values from rhs object
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        // check for self-assignment
        if (___) {
            // free resources (if needed)
            // allocate memory (if needed)
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        // check for self-assignment
        if (___ != ___) {
            // free resources (if needed)
            // allocate memory (if needed)
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Which values associated to the current object and the Thing named rhs must be compared to check for self-assignment?

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        // check for self-assignment
        if (___ != ___) {
            // free resources (if needed)
            // allocate memory (if needed)
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```


Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        // check for self-assignment
        if (_7_ != ___) {
            // free resources (if needed)
            // allocate memory (if needed)
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Which expression evaluates to the address of the current object (replacing blank #7)?

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        // check for self-assignment
        if (_7_ != ___) {
            // free resources (if needed)
            // allocate memory (if needed)
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        // check for self-assignment
        if ( this != &rhs ) {
            // free resources (if needed)
            // allocate memory (if needed)
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        // check for self-assignment
        if ( this != &rhs ) {
            // free resources (if needed)
            // allocate memory (if needed)
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Which expression evaluates to the address of the object named rhs (replacing blank #8)?

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        // check for self-assignment
        if ( this != &rhs ) {
            // free resources (if needed)
            // allocate memory (if needed)
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        // check for self-assignment
        if ( this != &rhs ) {
            // free resources (if needed)
            // allocate memory (if needed)
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            // free resources (if needed)
            // allocate memory (if needed)
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            // free resources (if needed)

            ---
            // allocate memory (if needed)
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```


Which resource(s) is/are the current object responsible for freeing?

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            // free resources (if needed)

            ---
            // allocate memory (if needed)
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            // free resources (if needed)

            ---
            // allocate memory (if needed)
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            // free resources (if needed)
            _9_
            // allocate memory (if needed)
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Which statement will free the memory allocated for the current object (replacing blank #9)?

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            // free resources (if needed)
            _9_
            // allocate memory (if needed)
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            // free resources (if needed)
            delete i_ptr;
            // allocate memory (if needed)
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            // allocate memory (if needed)
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            // allocate memory (if needed)
            ---
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            // allocate memory (if needed)
            _10_
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```


Which statement will allocate a new integer from the heap and assign the address of the integer to the current object's `i_ptr` member (replacing blank #10)?

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            // allocate memory (if needed)
            _10_
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            // allocate memory (if needed)
            i_ptr = new int;
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            i_ptr = new int;
            // copy values from rhs object
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            i_ptr = new int;
            // copy values from rhs object
            ---
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            i_ptr = new int;
            // copy values from rhs object
            _11_
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            i_ptr = new int;
            // copy values from rhs object
            _11_
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Which statement will assign the integer value associated with rhs to the memory address "pointed to" by the current object's i_ptr member (replacing blank #11)?

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            i_ptr = new int;
            // copy values from rhs object
            _11_
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            i_ptr = new int;
            // copy values from rhs object
            *i_ptr = *rhs.i_ptr;
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```


Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            i_ptr = new int;
            // copy values from rhs object
            *i_ptr = *rhs.i_ptr;
            name = ___;
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            i_ptr = new int;
            // copy values from rhs object
            *i_ptr = *rhs.i_ptr;
            name = _12_;
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Which expression replaces blank #12 to assign (a copy of) the string name associated with rhs to the name associated with the current object?

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            i_ptr = new int;
            // copy values from rhs object
            *i_ptr = *rhs.i_ptr;
            name = _12_;
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            i_ptr = new int;
            // copy values from rhs object
            *i_ptr = *rhs.i_ptr;
            name = rhs.name;
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            i_ptr = new int;
            *i_ptr = *rhs.i_ptr;
            name = rhs.name;
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            i_ptr = new int( *rhs.i_ptr );
            name = rhs.name;
        }
        // return value of correct type
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            i_ptr = new int( *rhs.i_ptr );
            name = rhs.name;
        }
        // return value of correct type
        ---
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            i_ptr = new int( *rhs.i_ptr );
            name = rhs.name;
        }
        // return value of correct type
        _13_
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```


Which statement (replacing blank #13) will ensure that the correct value is returned by `operator=` for the `Thing` class?

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            i_ptr = new int( *rhs.i_ptr );
            name = rhs.name;
        }
        // return value of correct type
        _13_
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            i_ptr = new int( *rhs.i_ptr );
            name = rhs.name;
        }
        // return value of correct type
        return *this;
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            i_ptr = new int( *rhs.i_ptr );
            name = rhs.name;
        }
        return *this;
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            i_ptr = new int( *rhs.i_ptr );
            name = rhs.name;
        }
        return *this;
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr; compilation error

    delete thing_ptr;
}
```

Adding an assignment operator

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& rhs);
public:
    Thing(int val, const string& name) : name(name) {
        i_ptr = new int(val);
    }

    Thing(const Thing& another_thing) : name(another_thing.name) {
        i_ptr = new int( *another_thing.i_ptr );
    }

    void set_value(int val) { *i_ptr = val; }
    int get_value() const { return *i_ptr; }

    ~Thing() { delete i_ptr; }

    Thing& operator=(const Thing& rhs) {
        if ( this != &rhs ) {
            delete i_ptr;
            i_ptr = new int( *rhs.i_ptr );
            name = rhs.name;
        }
        return *this;
    }

private:
    int* i_ptr;
    string name;
};
```

```
int main() {
    Thing* thing_ptr = new Thing(6, "curly");
    Thing thing2 = Thing(10, "moe");

    thing2 = *thing_ptr;

    delete thing_ptr;
}
```

Vector class design

—

The C++ vector

```
#include <vector>
using namespace std;
```

```
int main() {
    vector<int> int_vec;

    int_vec.push_back(10);           // adds 10 to end of int_vec
    int_vec.push_back(5);            // adds 5 to end of int_vec
    size_t vec_size = int_vec.size(); // returns 2
    int_vec[0] = 20;                  // 0th element of vector becomes 20
    int second = int_vec[1];          // returns 5
    int_vec.pop_back();               // removes 5
    int last = int_vec.back();         // returns 20
    int_vec.clear();                  // removes all elements (size is 0)
}
```

What is a vector exactly?

How is it able to provide this functionality?

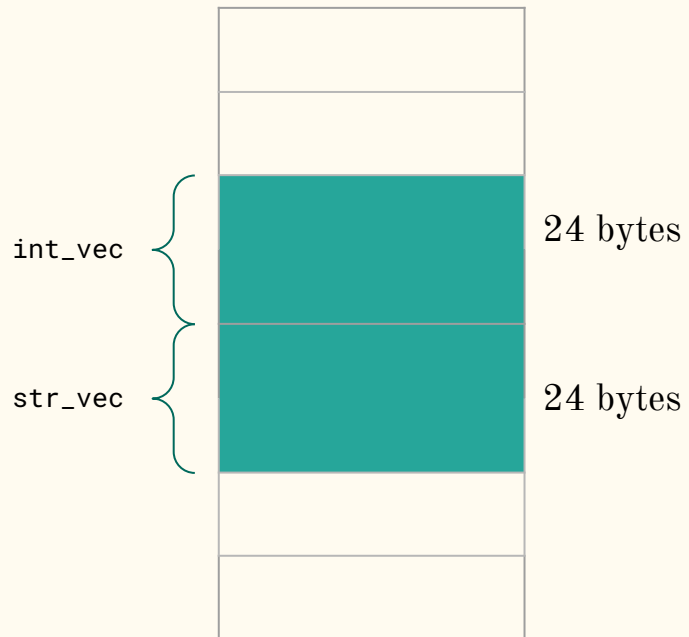
What does it look like in memory?

Implementation details of C++ vector

- every vector uses 24 bytes (minimum)

```
cout << sizeof(vector<int>) << endl; // always 24
```

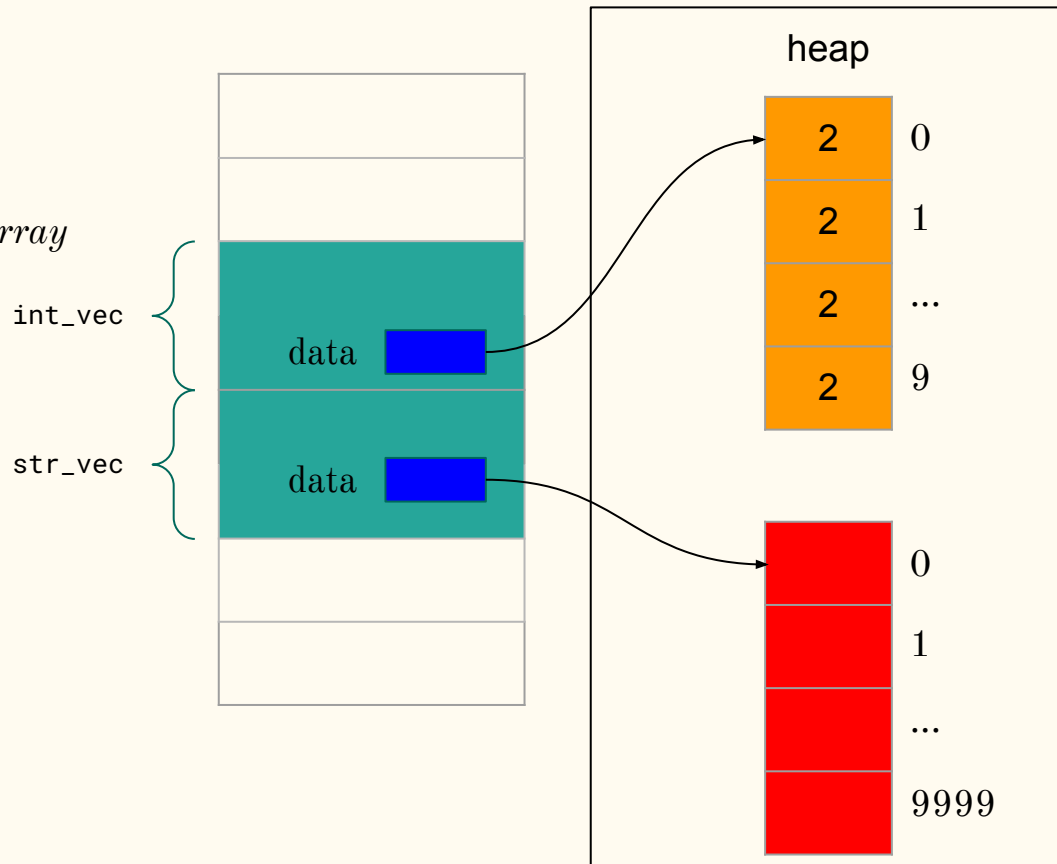
```
vector<int> int_vec(10, 2);  
vector<string> str_vec(10000);
```



Implementation details of C++ vector

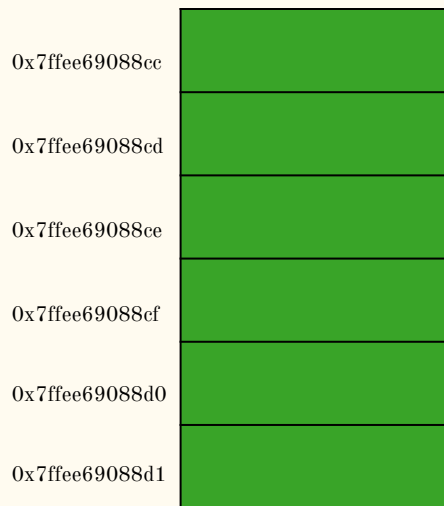
- every vector uses 24 bytes (minimum)
- data stored on heap using array
 - array on heap known as *dynamic array*

```
vector<int> int_vec(10, 2);  
vector<string> str_vec(10000);
```



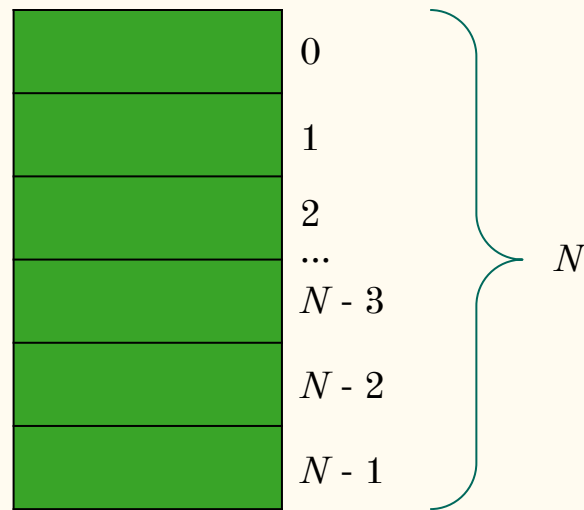
The array type

- contiguous block of memory



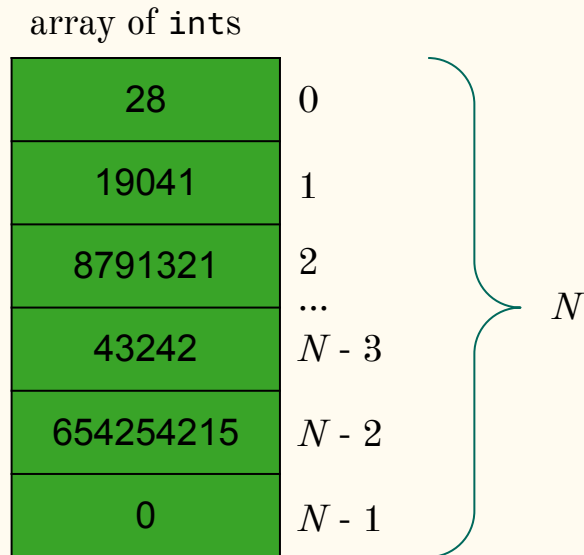
The array type

- contiguous block of memory
- fixed size (N elements)
- each element at sequential index
- all array elements same type



The array type


- contiguous block of memory
- fixed size (N elements)
- each element at sequential index
- all array elements same type



The array type

- contiguous block of memory
- fixed size (N elements)
- each element at sequential index
- all array elements same type

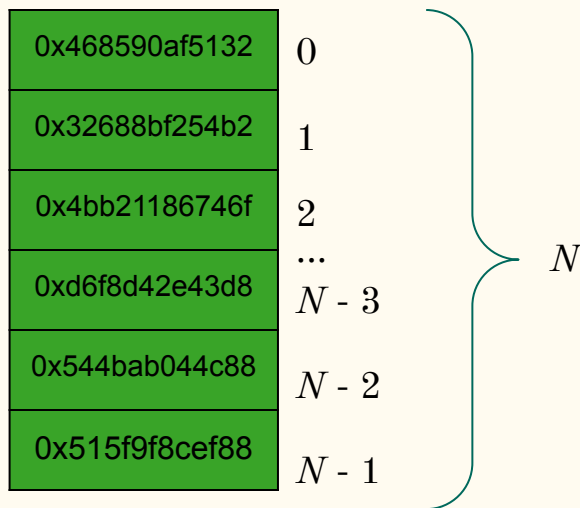
array of doubles

2.845	0	 N
190.41	1	
8.791321	2	
...	...	
4324.2	$N - 3$	
654254.215	$N - 2$	
0.	$N - 1$	

The array type

- contiguous block of memory
- fixed size (N elements)
- each element at sequential index
- all array elements same type

array of `int*`s



The array type

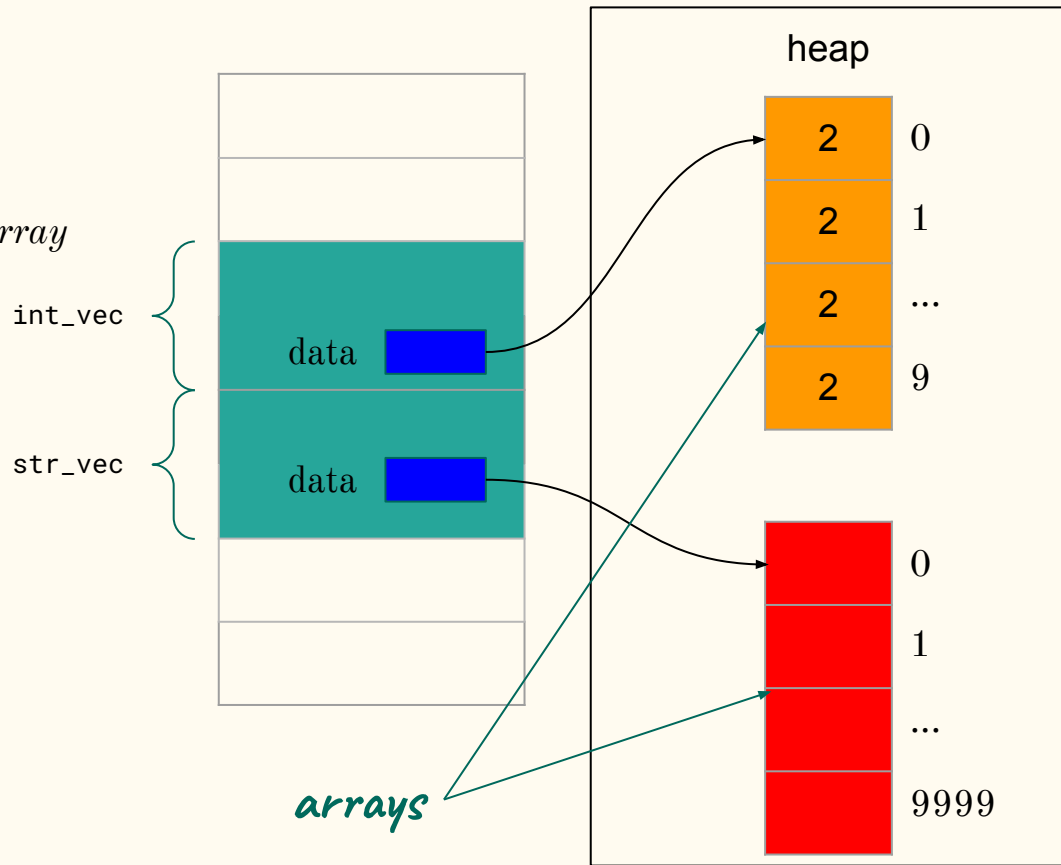
Limitations

- number of elements (size) not automatically stored with array
- unable to increase array size
- unable to decrease array size

Implementation details of C++ vector

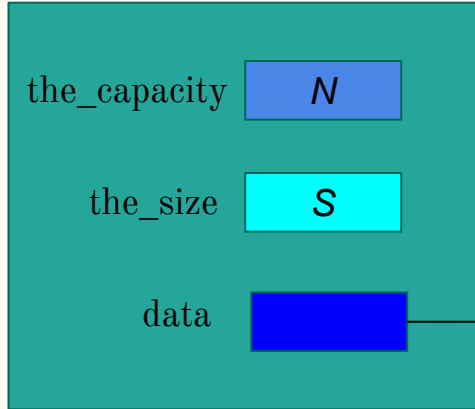
- every vector uses 24 bytes (minimum)
- data stored on heap using array
 - array on heap known as *dynamic array*

```
vector<int> int_vec(10, 2);  
vector<string> str_vec(10000);
```

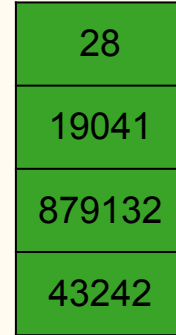


Implementing a Vector class

Vector



heap

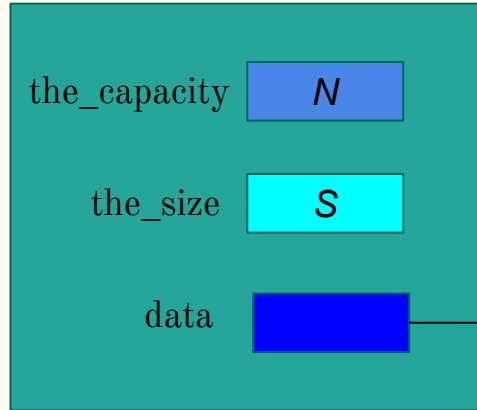


Type of data pointer

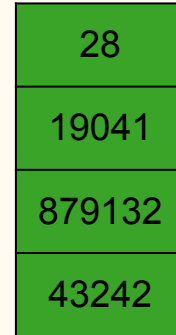
- depends on array type
- same type as pointer to first element of array

Implementing a Vector class

Vector



heap



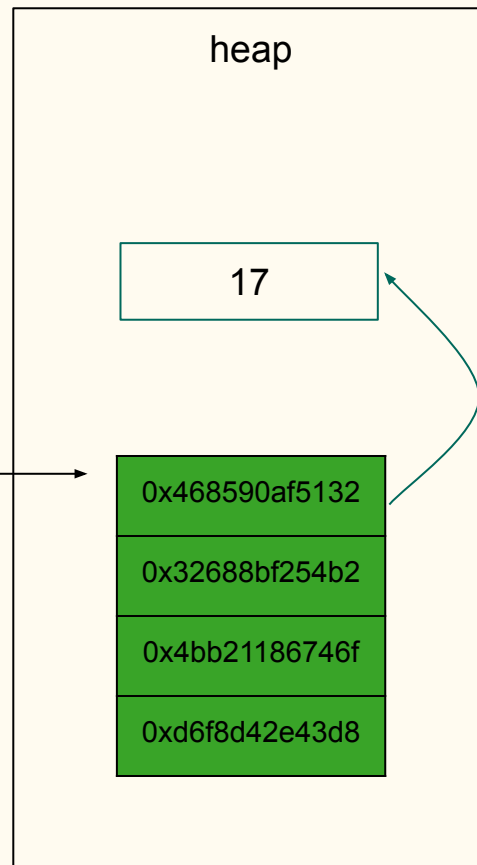
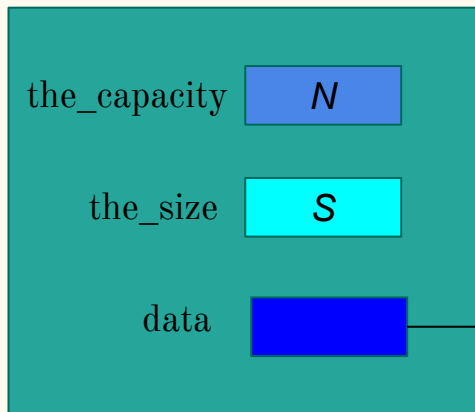
int data;*

Type of data pointer

- depends on array type
- same type as pointer to first element of array

Implementing a Vector class

Vector



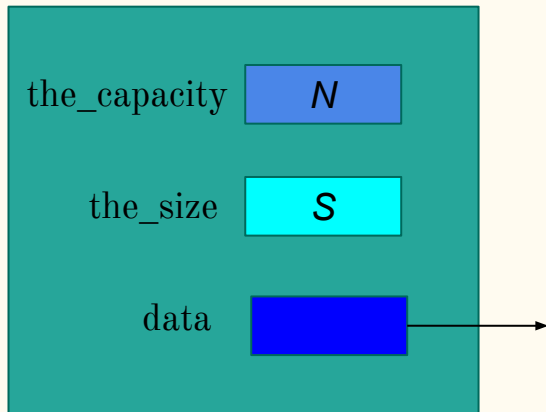
*int** data;*

Type of data pointer

- depends on array type
- same type as pointer to first element of array

Implementing a Vector class

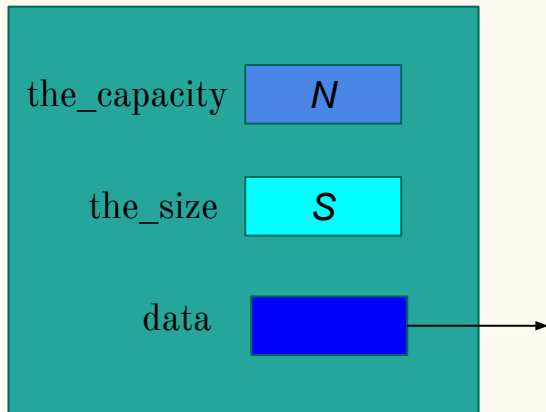
Vector



```
class Vector {  
public:  
private:  
    int* data;  
    size_t the_size;  
    size_t the_capacity;  
};
```

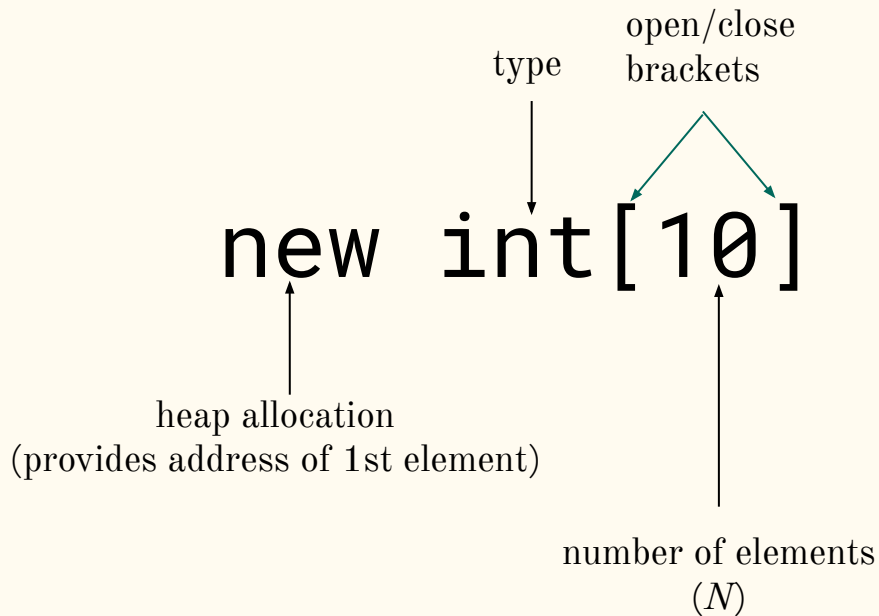
Implementing a Vector class

Vector



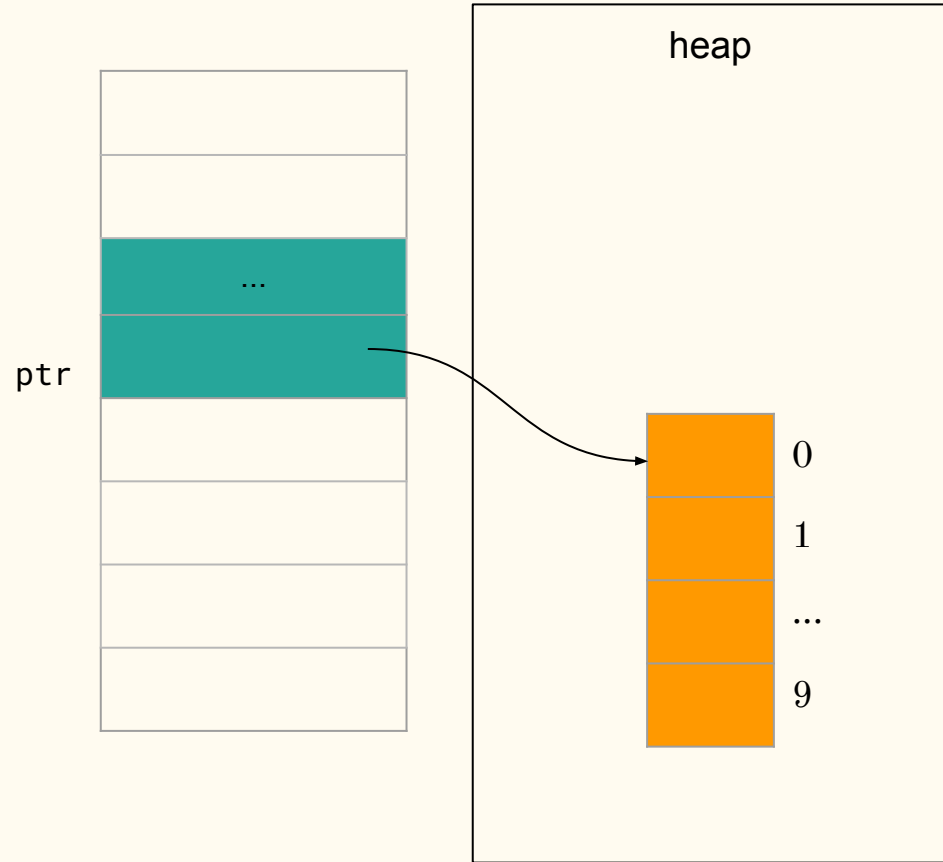
```
class Vector {  
public:  
private:  
    int* data;  
    size_t the_size;  
    size_t the_capacity;  
};
```

Creating a dynamic array



Creating a dynamic array

```
int* ptr = new int[10];
```

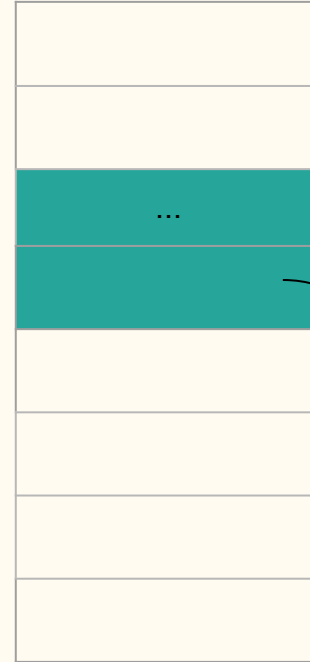


Creating a dynamic array

```
int** ptr = new int*[10];
```

extra asterisks

ptr



heap

17

0x468590af5132

0x32688bf254b2

...

0xd6f8d42e43d8

0

1

...

9