

OPERATOR OVERLOADING

RULE OF THREE

Problem Solving with Computers-II

C++

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

int main(){
    cout<<"Hola Facebook\n";
    return 0;
}
```



Will this code compile?

```
int main(){  
    Complex p;  
    p.conjugate();  
    p.print();  
}
```

- A. Yes
- B. No
- C. I am not sure . . .

```
class Complex  
{  
private:  
    double real;  
    double imag;  
public:  
    double getMagnitude() const;  
    double getReal() const;  
    double getImaginary() const;  
    void print() const;  
    void conjugate();  
    void setReal(double r);  
    void setImag(double r);  
};
```

Will this code compile?

```
int main(){
    Complex p;
    Complex w(1, 2);
    p = w;
    p.conjugate();
    p.print();
}
```

- A. Yes
- B. No
- C. I am not sure . . .

```
class Complex
{
private:
    double real;
    double imag;
public:
    Complex(double re, double im):
real(re), imag(im){}
    double getMagnitude() const;
    double getReal() const;
    double getImaginary() const;
    void print() const;
    void conjugate();
    void setReal(double r);
    void setImag(double r);
};
```

Will this code compile?

```
int main(){
    Complex p;
    Complex w(1, 2);
    p = w;
    p.conjugate();
    p.print();
}
```

- A. Yes
- B. No
- C. I am not sure . . .

```
class Complex
{
private:
    double real;
    double imag;
public:
    Complex(double re = 0, double im = 0):
        real(re), imag(im){}
    double getMagnitude() const;
    double getReal() const;
    double getImaginary() const;
    void print() const;
    void conjugate();
    void setReal(double r);
    void setImag(double r);
};
```

Operator Overloading

We would like to be able to perform operations on two objects of the class using the following operators:

<<

==

!=

+

-

and possibly others

Overloading the + operator for Complex objects

$$p = q + w;$$

Goal: We want to apply the + operator to Complex type objects

New method: add()

```
int main(){  
    Complex p;  
    Complex q(2, 3);  
    Complex w(10, -5);  
    w.conjugate();  
    p = _____;  
    p.print();  
}
```

Approach 1

```
int main(){  
    Complex p;  
    Complex q(2, 3);  
    Complex w(10, -5);  
    w.conjugate();  
    p = _____;  
    p.print()  
}
```

Approach 2

New method: add()

```
int main(){  
    Complex p;  
    Complex q(2, 3);  
    Complex w(10, -5);  
    w.conjugate();  
    p = add(q, w);  
    p.print();  
}
```

Approach 1

```
int main(){  
    Complex p;  
    Complex q(2, 3);  
    Complex w(10, -5);  
    w.conjugate();  
    p = q.add(w);  
    p.print()  
}
```

Approach 2

Overloading the + operator for Complex objects

```
p = add(q, w);
```

```
p = q.add(w);
```

```
p = q + w;
```

Goal: We want to apply the + operator to Complex type objects

Overloading the << operator

```
int main(){  
    Complex w(10, -5);  
    w.conjugate();  
    w.print();  
}
```

```
int main(){  
    Complex w(10, -5);  
    w.conjugate();  
    cout << w;  
}
```

Before overloading the << operator

After overloading the << operator

```
cout << w;
```

Select any equivalent C++ statement:

```
w.operator<<(cout);
```

A

```
cout.operator<<(w);
```

B

```
operator<<(cout, w);
```

C

```
operator<<(cout, w);
```

Select the function declaration that does NOT match the above call

A

```
void operator<<(ostream &out,  
               const Complex &c);
```

B

```
void Complex::operator<<(ostream &out);
```

C

```
Complex operator<<(ostream &out,  
                  Complex c);
```

Overloading Operators for IntList

In lab01 you will overload operators for the IntList ADT

==

!=

+ (list concatenation)

<< (overloaded stream operation to print the sequence)

RULE OF THREE

If a class defines one (or more) of the following it should probably explicitly define all three:

1. Destructor
2. Copy constructor
3. Copy assignment

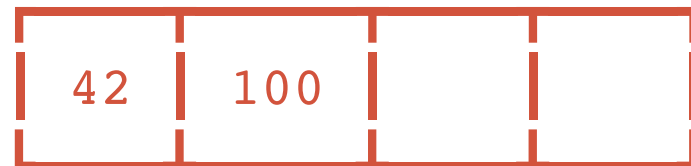
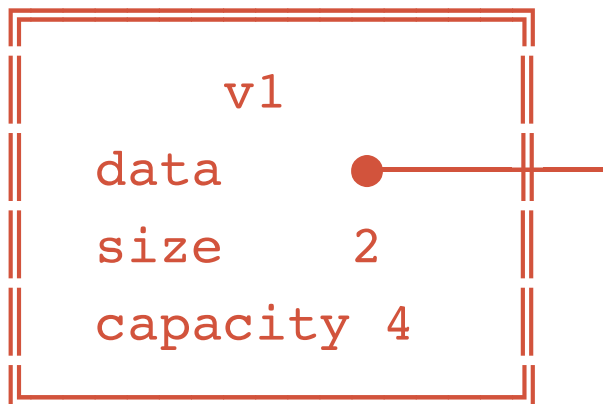
THE CODE:

```
CustomVector v1;  
v1.push_back(42);  
v1.push_back(100);
```

MEMORY AFTER `v1.push_back(100)`:

STACK

HEAP



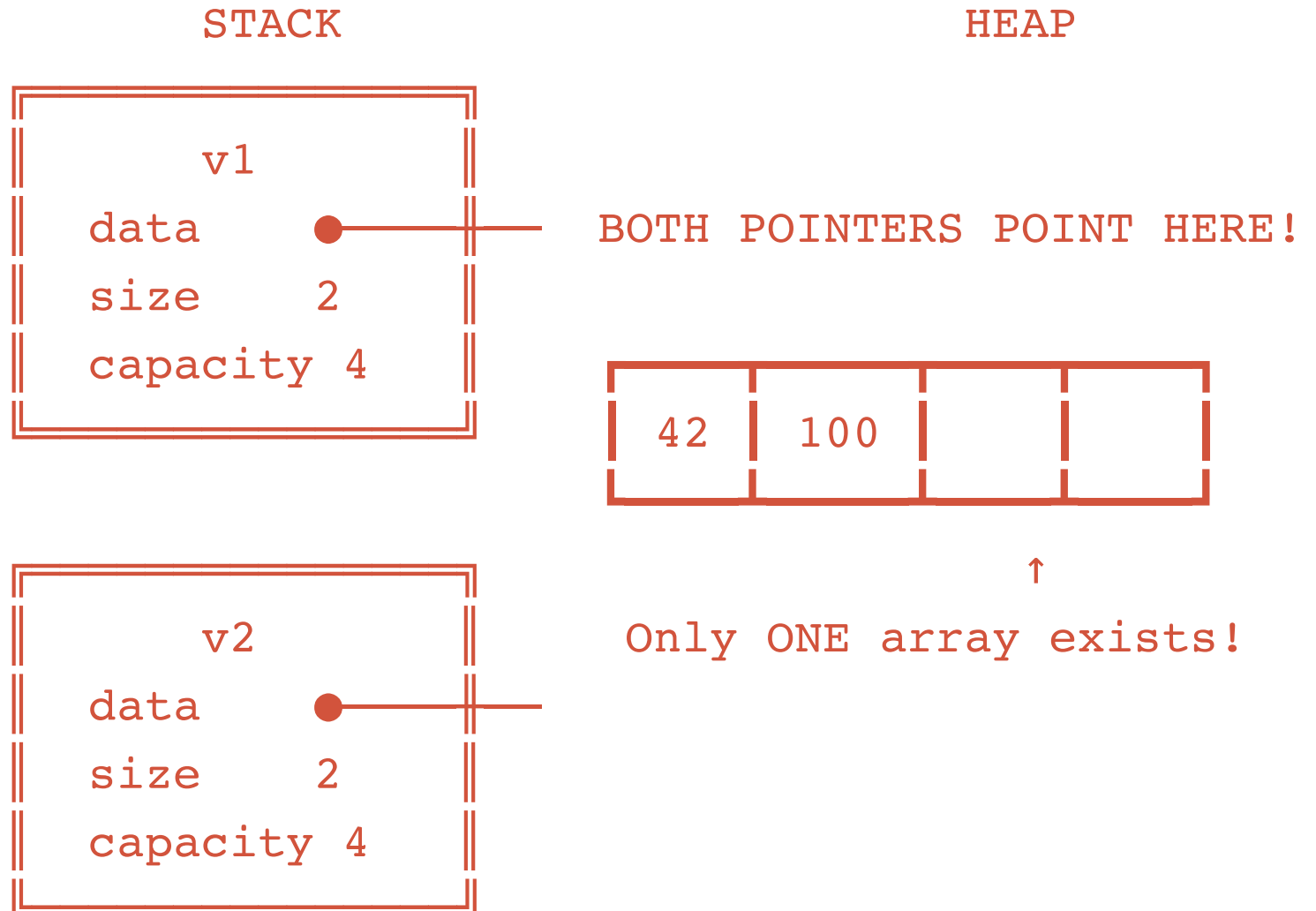
THE PROBLEM: SHALLOW COPY WITH DYNAMIC MEMORY

THE CODE:

```
CustomVector v1;  
v1.push_back(42);  
v1.push_back(100);  
CustomVector v2 = v1;
```

Default copy = SHALLOW
Copies pointers,
NOT the data!

MEMORY AFTER `v2 = v1`:



THE PROBLEM: SHALLOW COPY WITH DYNAMIC MEMORY

WHEN BOTH GO OUT OF SCOPE:

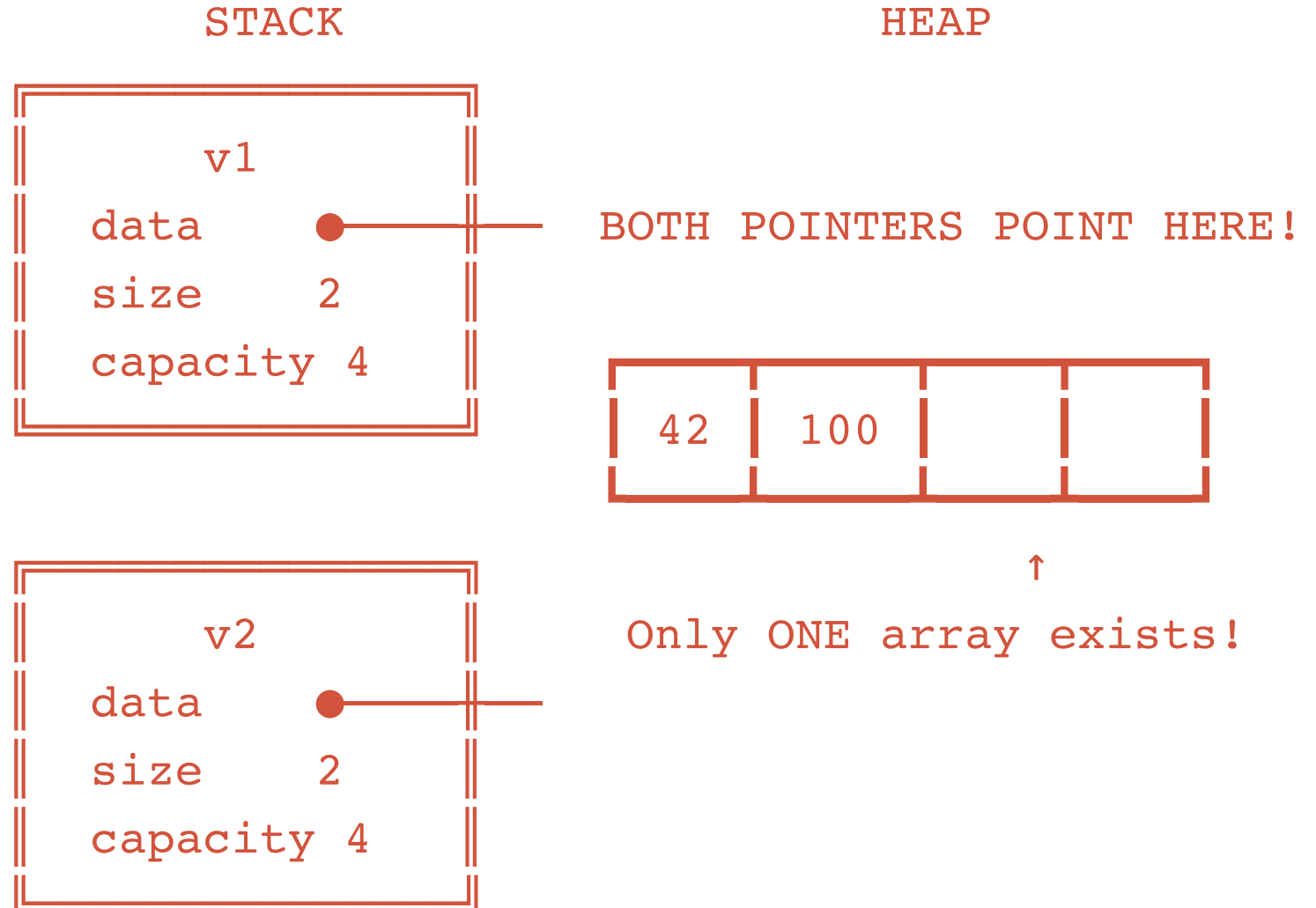
Step 1: v2's destructor runs → delete[] data; ✓
Frees the array

Step 2: v1's destructor runs → delete[] data; ✗
CRASH!

Already freed!

⚠ DOUBLE DELETION =
UNDEFINED BEHAVIOR
(crash or corruption)

MEMORY AFTER v2 = v1:



THE PROBLEM: SHALLOW COPY WITH DYNAMIC MEMORY

MEMORY AFTER $v2 = v1$:

THE SOLUTION:

THE BIG THREE:

1. Destructor
2. Copy Constructor
3. Copy Assignment

(Deep copy needed!)

STACK

HEAP

