SRS Setup
Login: student.turningtechnologies.com
Session ID: 20220228<A|D>

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

Implementing vectors

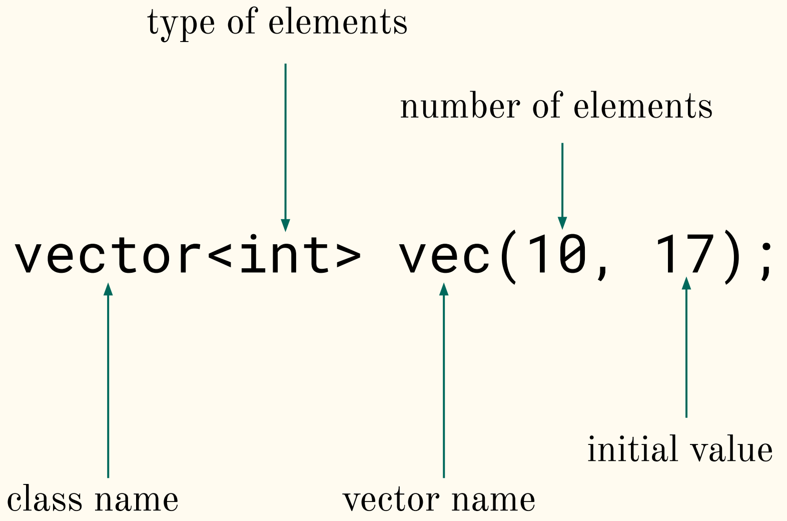
CS 2124: Object Oriented Programming
Darryl Reeves, Ph.D.

Agenda

- Vector constructor
- Vector destructor
- Vector copy constructor
- Vector assignment operator
- Vector class methods

The Vector constructor

C++ `vector` constructor



type of elements

number of elements

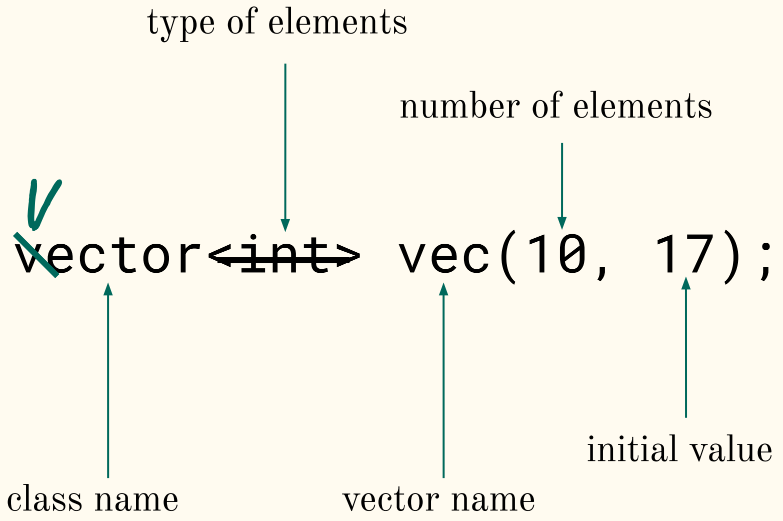
```
vector<int> vec(10, 17);
```

class name

vector name

initial value

CS2124 `Vector` constructor



type of elements

number of elements

V

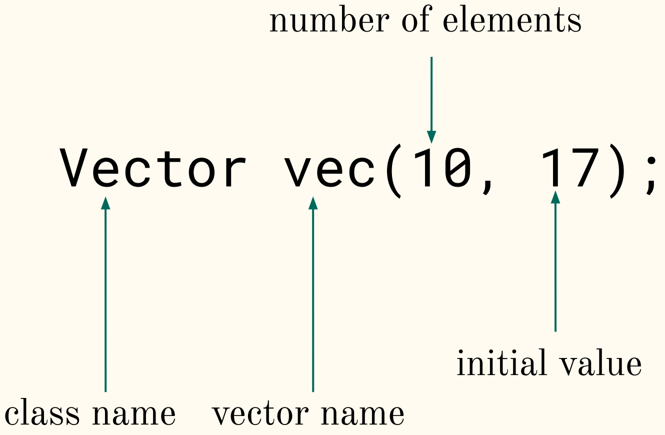
~~v~~ector~~<int>~~ vec(10, 17);

class name

vector name

initial value

CS2124 `Vector` constructor



number of elements

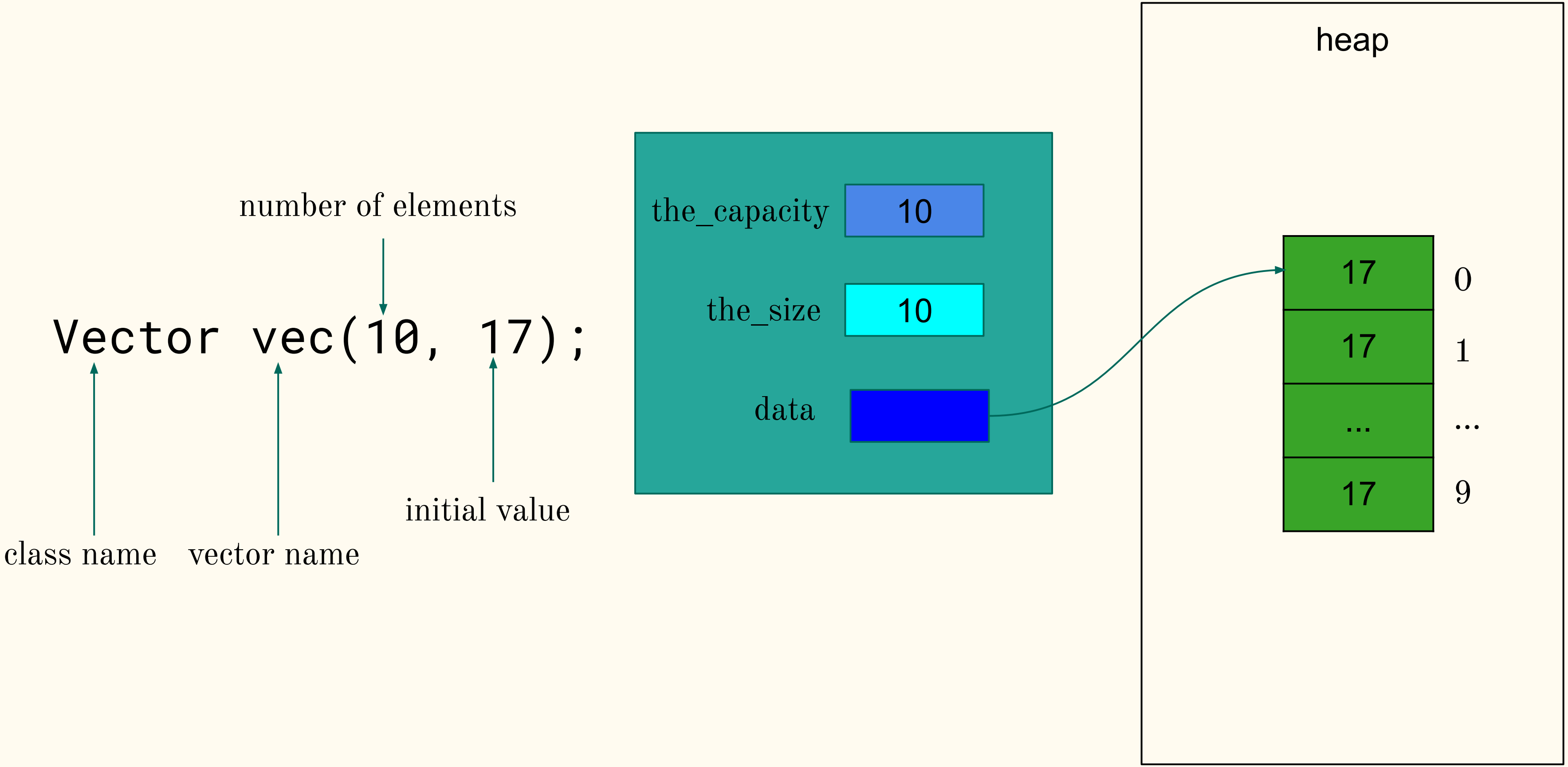
```
Vector vec(10, 17);
```

class name

vector name

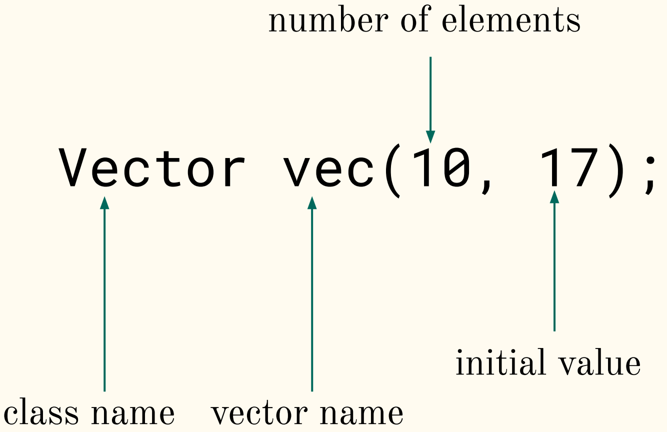
initial value

CS2124 Vector constructor



```
Vector vec(10, 17);
```

CS2124 Vector constructor



number of elements

```
Vector vec(10, 17);
```

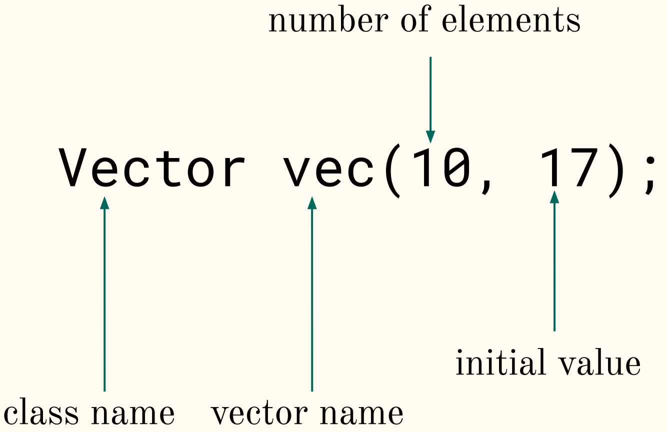
class name

vector name

initial value

```
class Vector {
public:
    // define constructor

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

CS2124 Vector constructor



number of elements

```
Vector vec(10, 17);
```

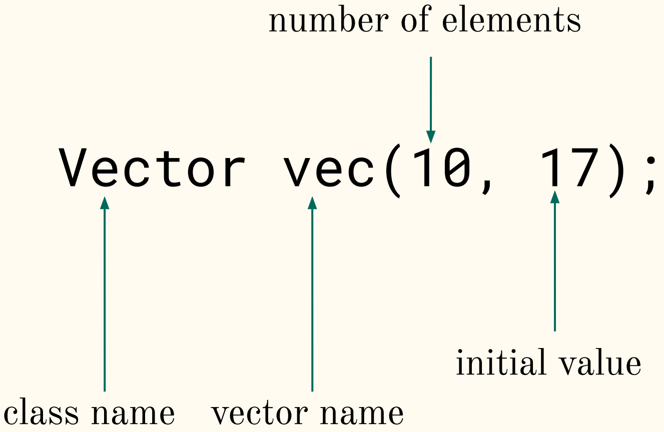
class name

vector name

initial value

```
class Vector {
public:
    // define constructor

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

CS2124 Vector constructor



number of elements

```
Vector vec(10, 17);
```

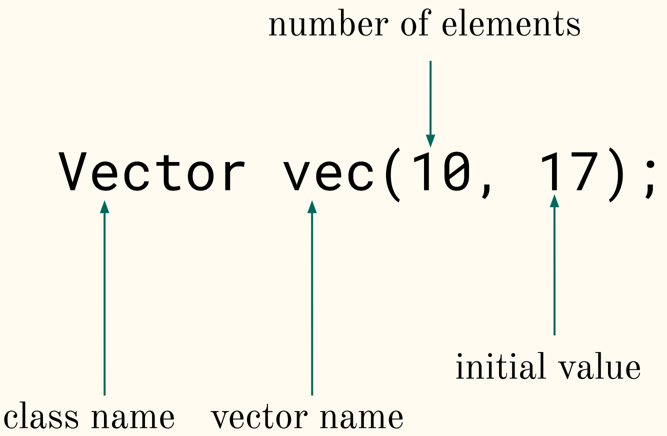
class name

vector name

initial value

```
class Vector {
public:
    Vector(size_t size, int value) {

    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

CS2124 Vector constructor



number of elements

```
Vector vec(10, 17);
```

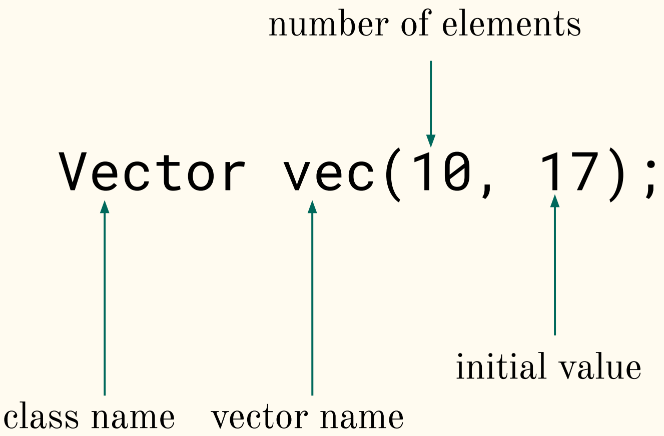
class name

vector name

initial value

```
class Vector {
public:
    Vector(size_t size, int value) {
       the_size = size;
       the_capacity = size;
       data = new int[size];
       for (size_t i = 0; i < the_size; ++i) {
           data[i] = value;
       }
    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

CS2124 Vector constructor



number of elements

```
Vector vec(10, 17);
```

class name

vector name

initial value

```
class Vector {
public:
    Vector(size_t size, int value) {
        the_size = size;
        the_capacity = size;
        data = new int[size];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = value;
        }
    }

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Great!!

CS2124 Vector constructor

```
Vector vec;
```

compilation error

```
class Vector {
public:
    Vector(size_t size, int value) {
        the_size = size;
        the_capacity = size;
        data = new int[size];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = value;
        }
    }

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

add a default constructor?

CS2124 Vector constructor

```
class Vector {
public:
    Vector(size_t size = 0, int value = 0) {
        the_size = size;
        the_capacity = size;
        data = new int[size];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = value;
        }
    }


private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

`Vector vec;` ~~compilation error~~

The `explicit` keyword

```
Vector vec(17);
```

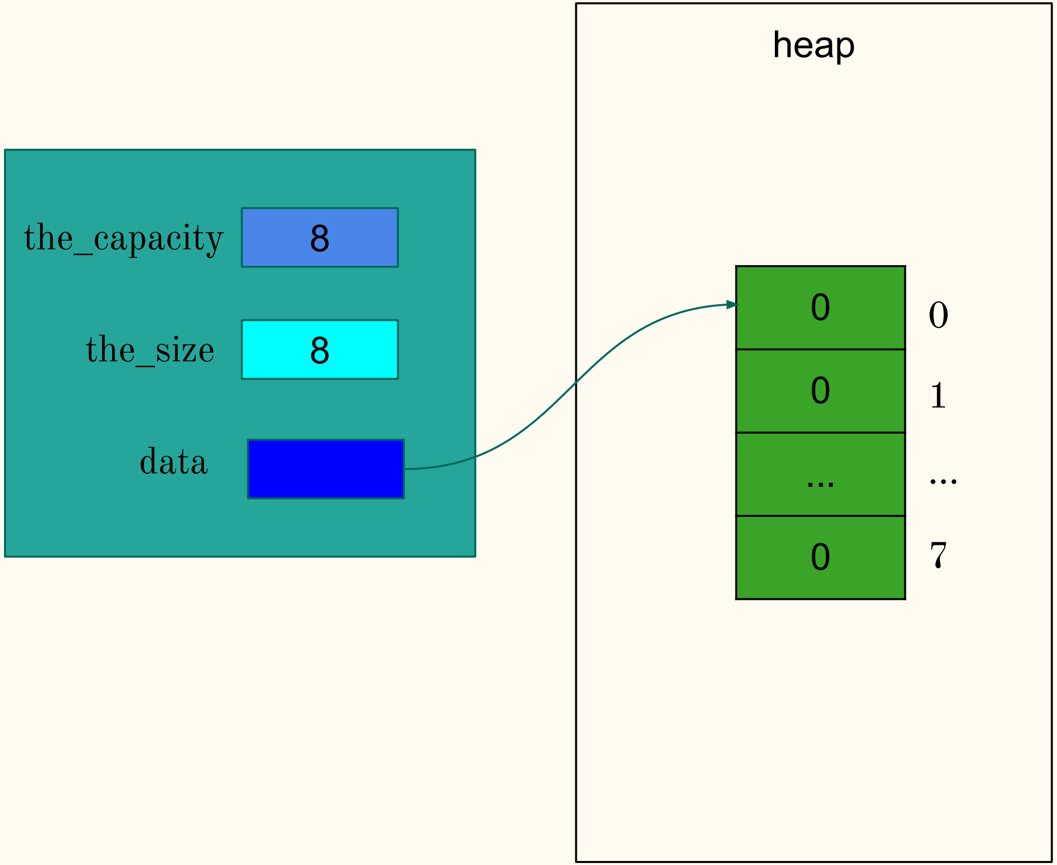
require size parameter

```
class Vector {
public:
    Vector(size_t size, int value = 0) {
        the_size = size;
        the_capacity = size;
        data = new int[size];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = value;
        }
    }


private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

The `explicit` keyword

```
Vector vec(17);
Vector vec2(8);
...

vec2 = 65;
```

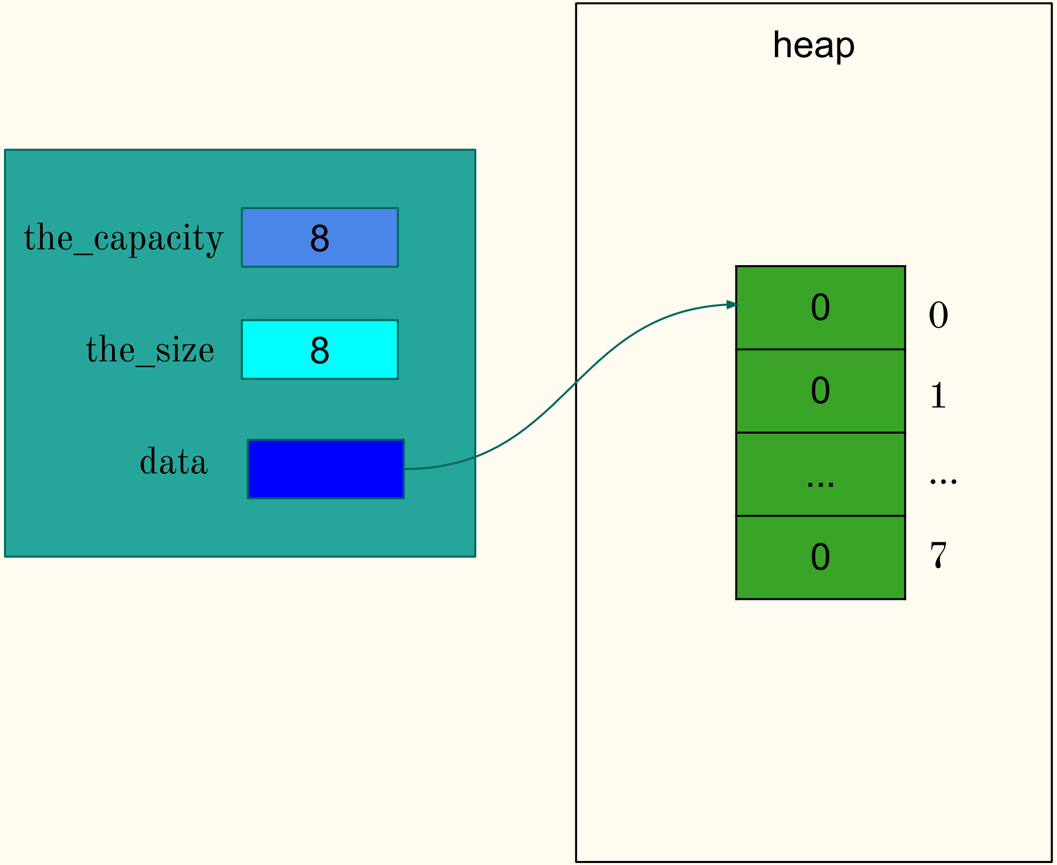
typo



The `explicit` keyword

```
Vector vec(17);
Vector vec2(8);
...

vec2 = 65;
```

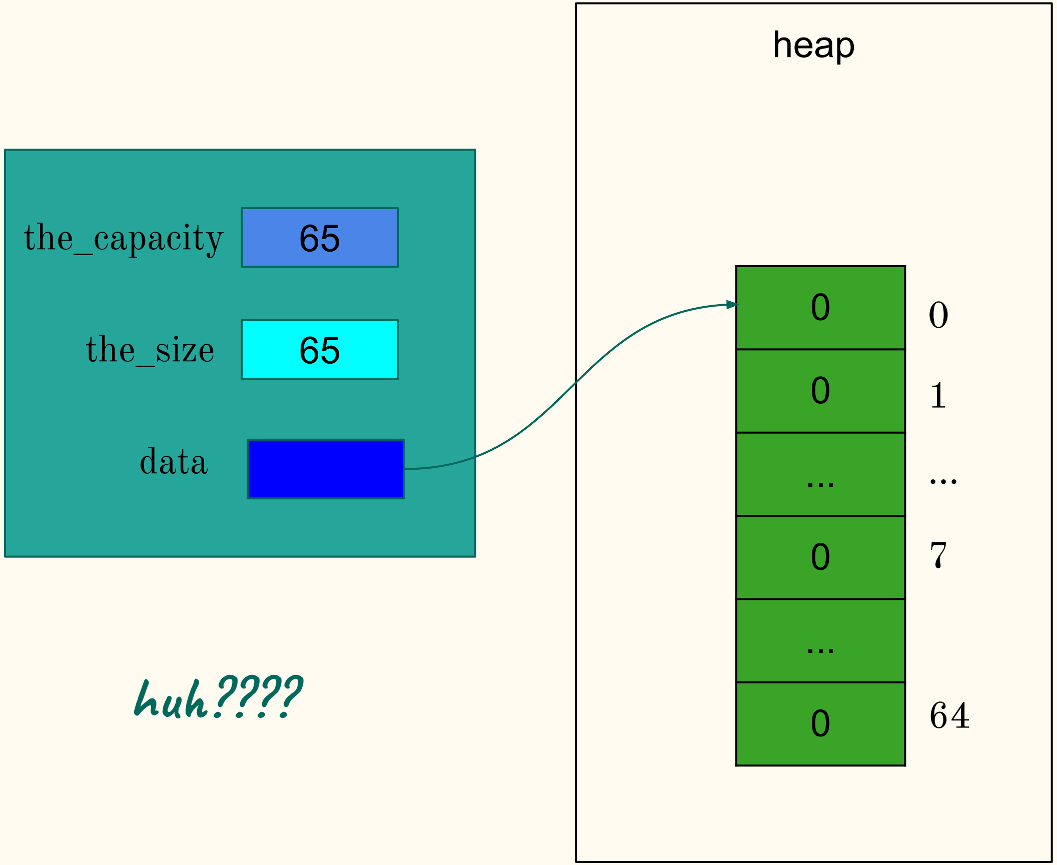
typo



The `explicit` keyword

```
Vector vec(17);
Vector vec2(8);
...

vec2 = 65;
```

typo



huh????

Implicit conversion

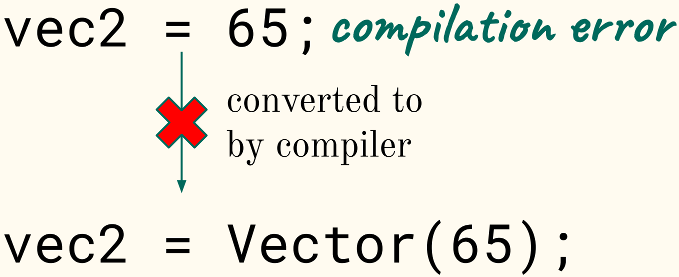
```
vec2 = 65;
```

↓ converted by compiler to

```
vec2 = Vector(65);
```

```
class Vector {
public:
    Vector(size_t size, int value = 0) {
        the_size = size;
        the_capacity = size;
        data = new int[size];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = value;
        }
    }


private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

The `explicit` keyword



```
vec2 = 65;
```

compilation error

converted to by compiler

```
vec2 = Vector(65);
```

```
class Vector {
public:
    explicit Vector(size_t size, int value = 0) {
        the_size = size;
        the_capacity = size;
        data = new int[size];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = value;
        }
    }


private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

The Vector destructor

CS2124 `Vector` destructor

```
class Vector {
public:
    Vector(size_t size = 0, int value = 0) {
        the_size = size;
        the_capacity = size;
        data = new int[size];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = value;
        }
    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

TurningPoint

SRS Setup
Login: student.turningtechnologies.com
Session ID: 20220228<A|D>

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

What responsibility would a Vector class destructor have?

```
class Vector {
public:
    Vector(size_t size = 0, int value = 0) {
        the_size = size;
        the_capacity = size;
        data = new int[size];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = value;
        }
    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

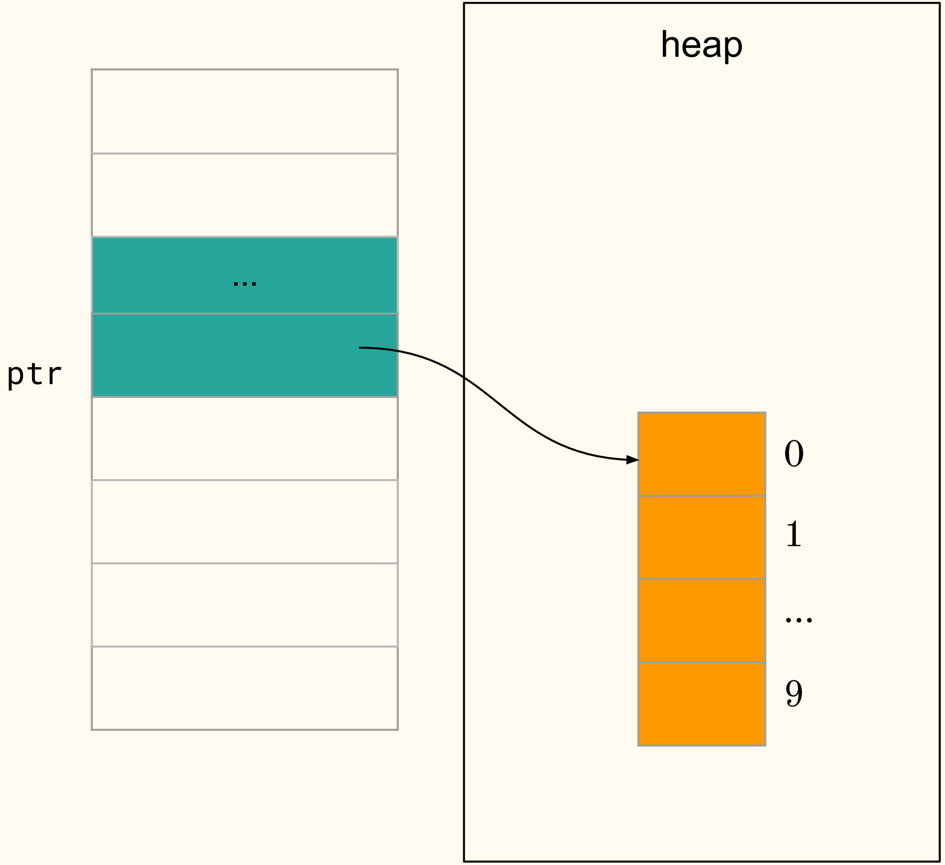
CS2124 Vector destructor

```
class Vector {
public:
    Vector(size_t size = 0, int value = 0) {
        the_size = size;
        the_capacity = size;
        data = new int[size];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = value;
        }
    }
    // define destructor
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Creating a dynamic array

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

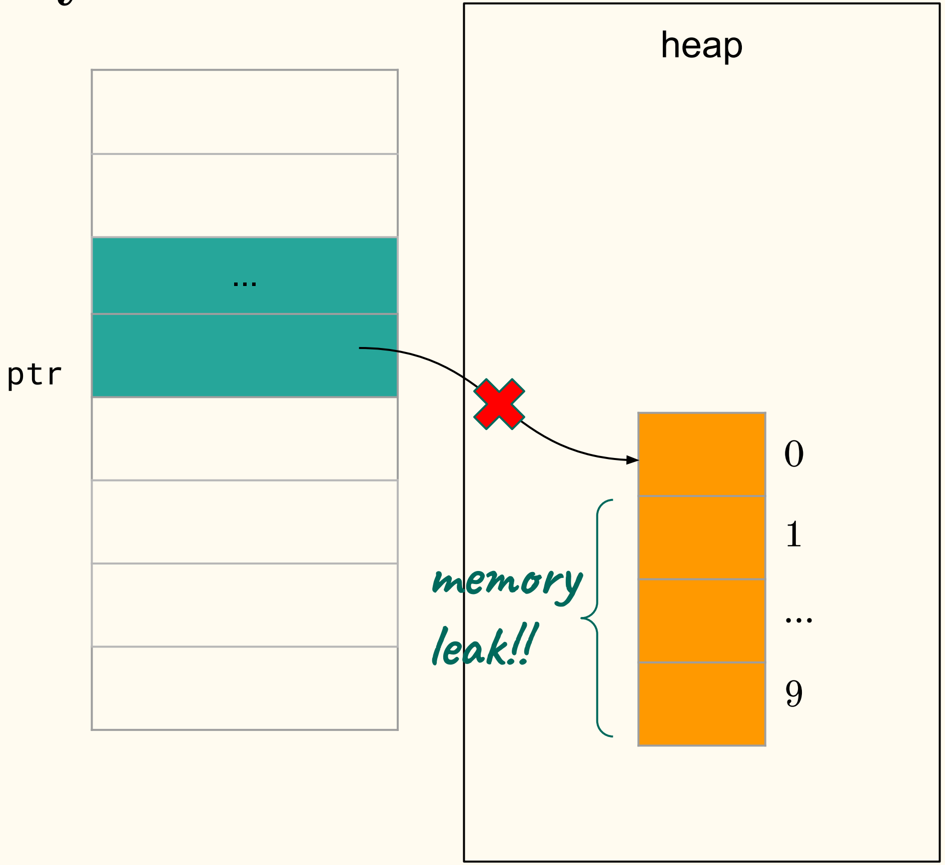
brackets and integer specify array size



Freeing dynamic array memory

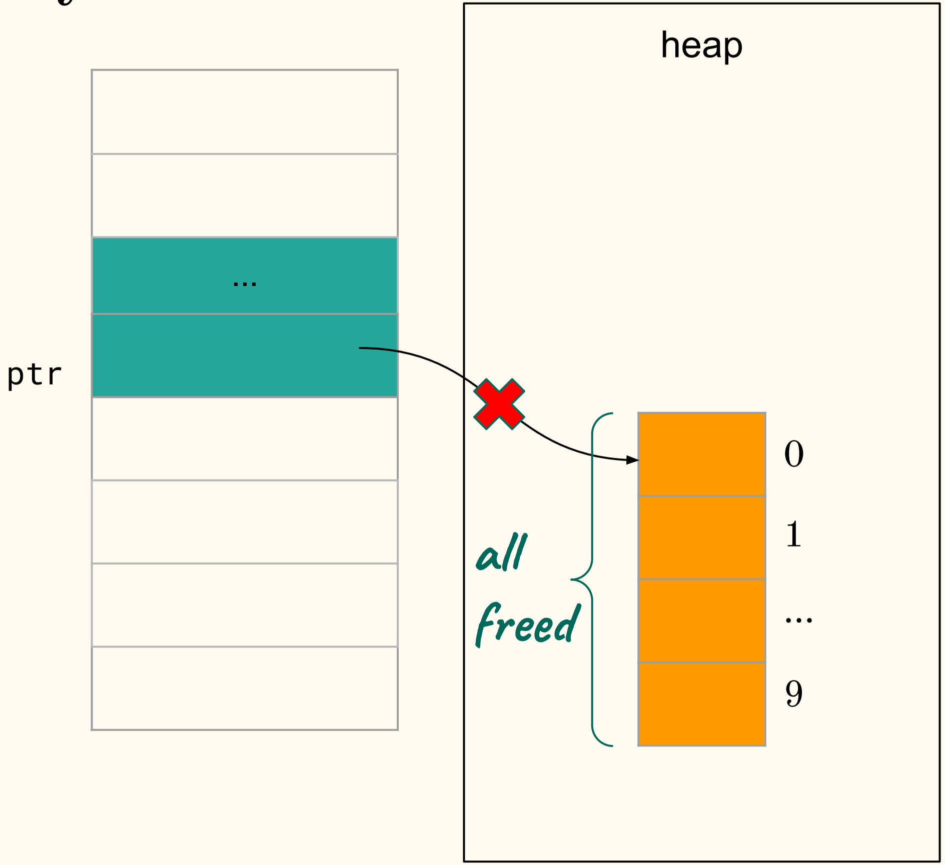
```
int* ptr = new int[10];

delete ptr; // free memory
```

No



Freeing dynamic array memory

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

delete [] ptr; // free memory
```
✔



CS2124 Vector destructor

```
class Vector {
public:
    Vector(size_t size = 0, int value = 0) {
        the_size = size;
        the_capacity = size;
        data = new int[size];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = value;
        }
    }
    // define destructor
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

CS2124 Vector destructor

```
class Vector {
public:
    Vector(size_t size = 0, int value = 0) {
        the_size = size;
        the_capacity = size;
        data = new int[size];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = value;
        }
    }
    ~Vector() { delete [] data; }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

The Vector copy constructor

Initializing one Vector from another

```
Vector vec1(10, 17);
Vector vec2(vec1);
```

memory error

```
class Vector {
public:
    Vector(size_t size = 0, int value = 0) {
        the_size = size;
        the_capacity = size;
        data = new int[size];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = value;
        }
    }
    ~Vector() { delete [] data; }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Initializing one Vector from another

```
Vector vec1(10, 17);
Vector vec2(vec1);
```

memory error

```
class Vector {
public:
    Vector(size_t size = 0, int value = 0) {
        the_size = size;
        the_capacity = size;
        data = new int[size];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = value;
        }
    }
    ~Vector() { delete [] data; }
    // add copy constructor
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Initializing one Vector from another

```
Vector vec1(10, 17);
Vector vec2(vec1);
```

memory error

```
class Vector {
public:
    ...
    // add copy constructor
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Initializing one Vector from another

```
Vector vec1(10, 17);
Vector vec2(vec1);
```

memory error

Requirements of copy constructor

- copy *size* and *capacity*
- allocate memory for `data`
- copy values to new array

```
class Vector {
public:
    ...
    // add copy constructor
    Vector(const Vector& rhs) {

    }


private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Initializing one Vector from another

```
Vector vec1(10, 17);
Vector vec2(vec1);
```

memory error

Requirements of copy constructor

- ~~copy *size* and *capacity*~~
- allocate memory for `data`
- copy values to new array

```
class Vector {
public:
    ...
    Vector(const Vector& rhs) {
        the_size = rhs.the_size;
        the_capacity = rhs.the_capacity;
    }

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Initializing one Vector from another

```
Vector vec1(10, 17);
Vector vec2(vec1);
```

memory error

Requirements of copy constructor

- ~~copy *size* and *capacity*~~
- ~~allocate memory for `data`~~
- copy values to new array

```
class Vector {
public:
    ...
    Vector(const Vector& rhs) {
        the_size = rhs.the_size;
        the_capacity = rhs.the_capacity;
        data = new int[the_capacity];
    }

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Why is the capacity used to determine the size of the memory to allocate?

```
Vector vec1(10, 17);
Vector vec2(vec1);
```

memory error

Requirements of copy constructor

- ~~copy *size* and *capacity*~~
- ~~allocate memory for `data`~~
- copy values to new array

```
class Vector {
public:
    ...
    Vector(const Vector& rhs) {
        the_size = rhs.the_size;
        the_capacity = rhs.the_capacity;
        data = new int[the_capacity];
    }

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Initializing one Vector from another

```
Vector vec1(10, 17);
Vector vec2(vec1);
```

memory error

Requirements of copy constructor

- ~~copy *size* and *capacity*~~
- ~~allocate memory for `data`~~
- copy values to new array

```
class Vector {
public:
    ...
    Vector(const Vector& rhs) {
        the_size = rhs.the_size;
        the_capacity = rhs.the_capacity;
        data = new int[the_capacity];
    }

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Initializing one Vector from another

```
Vector vec1(10, 17);
Vector vec2(vec1);
```

~~memory error~~

Requirements of copy constructor

- ~~copy *size* and *capacity*~~
- ~~allocate memory for `data`~~
- ~~copy values to new array~~

```
class Vector {
public:
    ...
    Vector(const Vector& rhs) {
        the_size = rhs.the_size;
        the_capacity = rhs.the_capacity;
        data = new int[the_capacity];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = rhs.data[i];
        }
    }
```

hmmm...looks familiar

```
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Initializing one Vector from another

```
Vector vec1(10, 17);
Vector vec2(vec1);
```

✔

```
class Vector {
public:
    ...
    Vector(const Vector& rhs) {
        the_size = rhs.the_size;
        the_capacity = rhs.the_capacity;
        data = new int[the_capacity];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = rhs.data[i];
        }
    }

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

The Vector class (so far)

```
class Vector {
public:
    Vector(size_t size = 0, int value = 0) {
        the_size = size;
        the_capacity = size;
        data = new int[size];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = value;
        }
    }
    Vector(const Vector& rhs) {
        the_size = rhs.the_size;
        the_capacity = rhs.the_capacity;
        data = new int[the_capacity];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = rhs.data[i];
        }
    }
    ~Vector() { delete [] data; }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

constructor

copy constructor

destructor

The assignment operator

Assigning a Vector to an existing variable

```
Vector vec1(10, 17);
Vector vec2(1000, 5);

vec2 = vec1;
```

memory error

```
class Vector {
public:
    ...
    Vector(const Vector& rhs) {
        the_size = rhs.the_size;
        the_capacity = rhs.the_capacity;
        data = new int[the_capacity];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = rhs.data[i];
        }
    }

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

What type of memory error will be introduced by using the default assignment operator for the `Vector` class?

```
Vector vec1(10, 17);
Vector vec2(1000, 5);

vec2 = vec1; memory error
```

```
class Vector {
public:
    ...
    Vector(const Vector& rhs) {
        the_size = rhs.the_size;
        the_capacity = rhs.the_capacity;
        data = new int[the_capacity];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = rhs.data[i];
        }
    }

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Assigning a Vector to an existing variable

```
Vector vec1(10, 17);
Vector vec2(1000, 5);

vec2 = vec1;
```

memory error

Requirements of assignment operator

- check for self-assignment
- free old memory (if needed)
- allocate new memory (if needed)
- copy values
- return proper type and object

```
class Vector {
public:
    ...
    // add assignment operator

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Assigning a Vector to an existing variable

```
Vector vec1(10, 17);
Vector vec2(1000, 5);

vec2 = vec1;
```

memory error

Requirements of assignment operator

- check for self-assignment
- free old memory (if needed)
- allocate new memory (if needed)
- copy values
- return proper type and object

```
class Vector {
public:
    ...
    Vector& operator=(const Vector& rhs) {

    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Assigning a Vector to an existing variable

```
Vector vec1(10, 17);
Vector vec2(1000, 5);

vec2 = vec1;
```
memory error

Requirements of assignment operator

- ~~check for self-assignment~~
- free old memory (if needed)
- allocate new memory (if needed)
- copy values
- return proper type and object

```
class Vector {
public:
    ...
    Vector& operator=(const Vector& rhs) {
        if (this != &rhs) {

        }

    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Assigning a Vector to an existing variable

```
Vector vec1(10, 17);
Vector vec2(1000, 5);

vec2 = vec1;
```

memory error

Requirements of assignment operator

- ~~check for self-assignment~~
- ~~free old memory (if needed)~~
- allocate new memory (if needed)
- copy values
- return proper type and object

```
class Vector {
public:
    ...
    Vector& operator=(const Vector& rhs) {
        if (this != &rhs) {
            delete [] data;
        }

    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Assigning a Vector to an existing variable

```
Vector vec1(10, 17);
Vector vec2(1000, 5);

vec2 = vec1;
```

memory error

Requirements of assignment operator

- ~~check for self-assignment~~
- ~~free old memory (if needed)~~
- ~~allocate new memory (if needed)~~
- copy values
- return proper type and object

```
class Vector {
public:
    ...
    Vector& operator=(const Vector& rhs) {
        if (this != &rhs) {
            delete [] data;
            data = new int[rhs.the_capacity];
        }

    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Assigning a Vector to an existing variable

```
Vector vec1(10, 17);
Vector vec2(1000, 5);

vec2 = vec1;
```

memory error

Requirements of assignment operator

- ~~check for self-assignment~~
- ~~free old memory (if needed)~~
- ~~allocate new memory (if needed)~~
- copy values
- return proper type and object

```
class Vector {
public:
    ...
    Vector& operator=(const Vector& rhs) {
        if (this != &rhs) {
            delete [] data;
            data = new int[rhs.the_capacity];
            the_size = rhs.the_size;
            the_capacity = rhs.the_capacity;
        }

    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Assigning a Vector to an existing variable

```
Vector vec1(10, 17);
Vector vec2(1000, 5);

vec2 = vec1;
```

memory error

Requirements of assignment operator

- ~~check for self-assignment~~
- ~~free old memory (if needed)~~
- ~~allocate new memory (if needed)~~
- ~~copy values~~
- return proper type and object

```
class Vector {
public:
    ...
    Vector& operator=(const Vector& rhs) {
        if (this != &rhs) {
            delete [] data;
            data = new int[rhs.the_capacity];
            the_size = rhs.the_size;
            the_capacity = rhs.the_capacity;
            for (size_t i = 0; i < the_size; ++i) {
                data[i] = rhs.data[i];
            }
        }

    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Assigning a Vector to an existing variable

```
Vector vec1(10, 17);
Vector vec2(1000, 5);

vec2 = vec1;
```

~~memory error~~

Requirements of assignment operator

- ~~check for self-assignment~~
- ~~free old memory (if needed)~~
- ~~allocate new memory (if needed)~~
- ~~copy values~~
- ~~return proper type and object~~

```
class Vector {
public:
    ...
    Vector& operator=(const Vector& rhs) {
        if (this != &rhs) {
            delete [] data;
            data = new int[rhs.the_capacity];
            the_size = rhs.the_size;
            the_capacity = rhs.the_capacity;
            for (size_t i = 0; i < the_size; ++i) {
                data[i] = rhs.data[i];
            }
        }
        return *this;

    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Assigning a Vector to an existing variable

```
Vector vec1(10, 17);
Vector vec2(1000, 5);

vec2 = vec1;
```

Requirements of assignment operator

- ~~check for self-assignment~~
- ~~free old memory (if needed)~~
- ~~allocate new memory (if needed)~~
- ~~copy values~~
- ~~return proper type and object~~

```
class Vector {
public:
    ...
    Vector& operator=(const Vector& rhs) {
        if (this != &rhs) {
            delete [] data;
            data = new int[rhs.the_capacity];
            the_size = rhs.the_size;
            the_capacity = rhs.the_capacity;
            for (size_t i = 0; i < the_size; ++i) {
                data[i] = rhs.data[i];
            }
        }
        return *this;
    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

The Vector class (so far)

```
class Vector {
public:
    Vector(size_t size = 0, int value = 0) {
        the_size = size;
        the_capacity = size;
        data = new int[size];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = value;
        }
    }
    Vector(const Vector& rhs) {
        the_size = rhs.the_size;
        the_capacity = rhs.the_capacity;
        data = new int[the_capacity];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = rhs.data[i];
        }
    }
    Vector& operator=(const Vector& rhs) {
        if (this != &rhs) {
            delete [] data;
            data = new int[rhs.the_capacity];
            the_size = rhs.the_size;
            the_capacity = rhs.the_capacity;
            for (size_t i = 0; i < the_size; ++i) {
                data[i] = rhs.data[i];
            }
        }
        return *this;
    }
    ~Vector() { delete [] data; }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

constructor

copy constructor

assignment operator

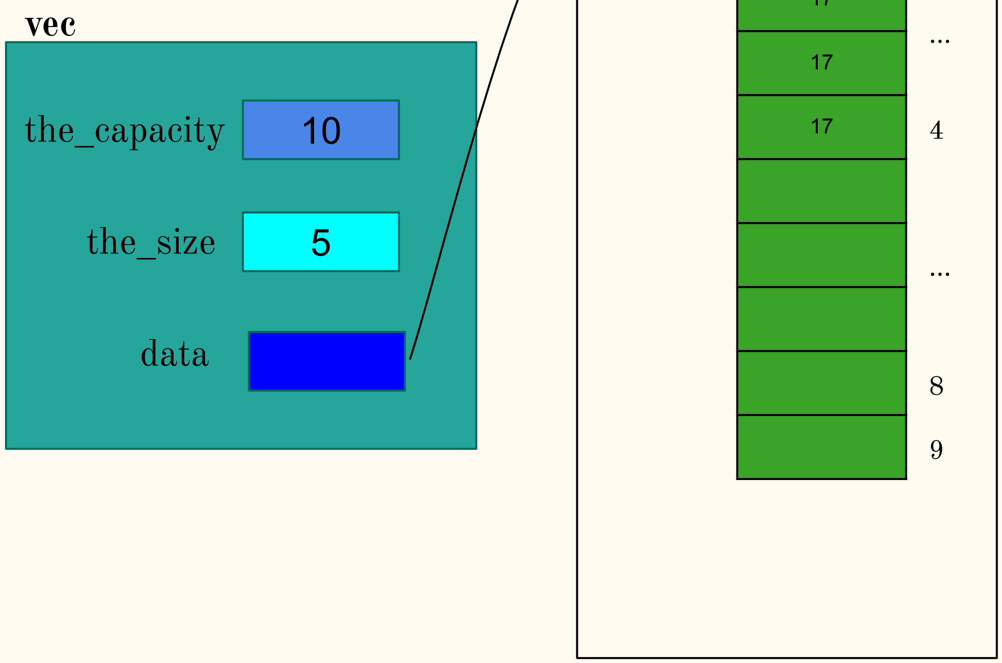
destructor

Vector class methods

Implementing a `push_back()` method

- `push_back()` adds an element to the end of vector
    - element added to end of array
- two possibilities
    - array is not full

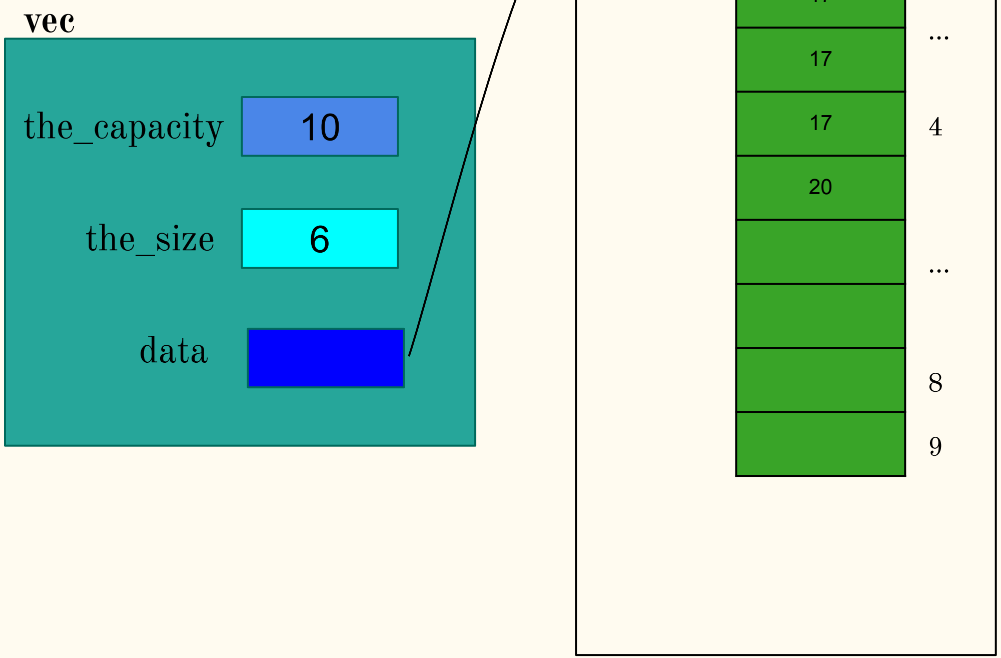
```
Vector vec;

... // modifications

vec.push_back(20)
```



Implementing a `push_back()` method

- `push_back()` adds an element to the end of vector
  - element added to end of array
- two possibilities
  - array is not full

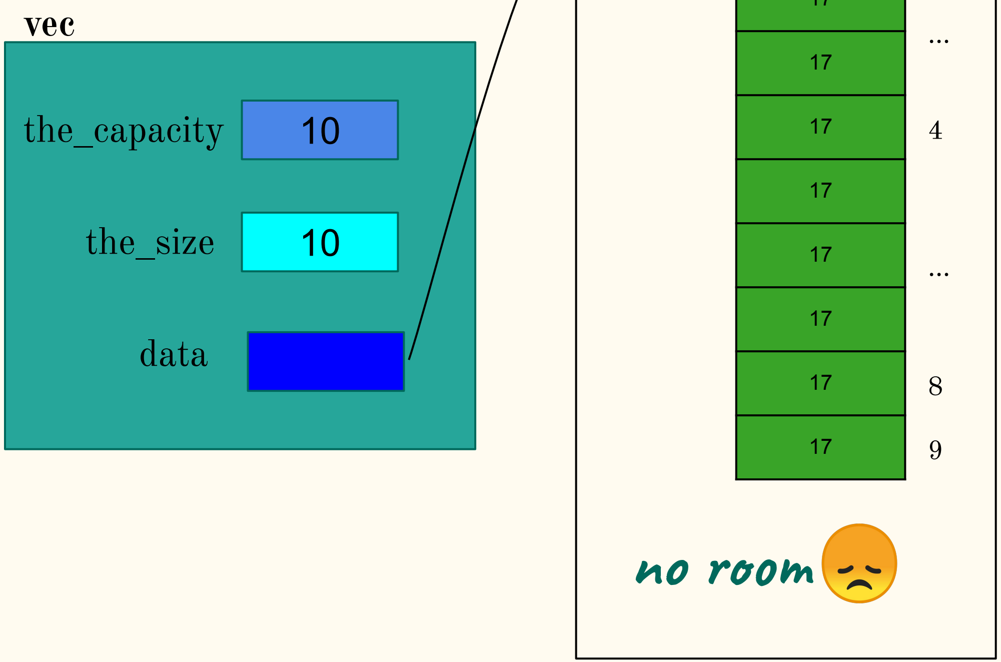
```
Vector vec;

... // modifications

vec.push_back(20)
```



Implementing a `push_back()` method

- `push_back()` adds an element to the end of vector
  - element added to end of array
- two possibilities
  - array is not full
  - array is full

```
Vector vec(10, 17);
vec.push_back(20);
```



vec

the_capacity 10

the_size 10

data

heap

17	0
17	1
17	
17	…
17	4
17	
17	
17	…
17	8
17	9

no room 😞

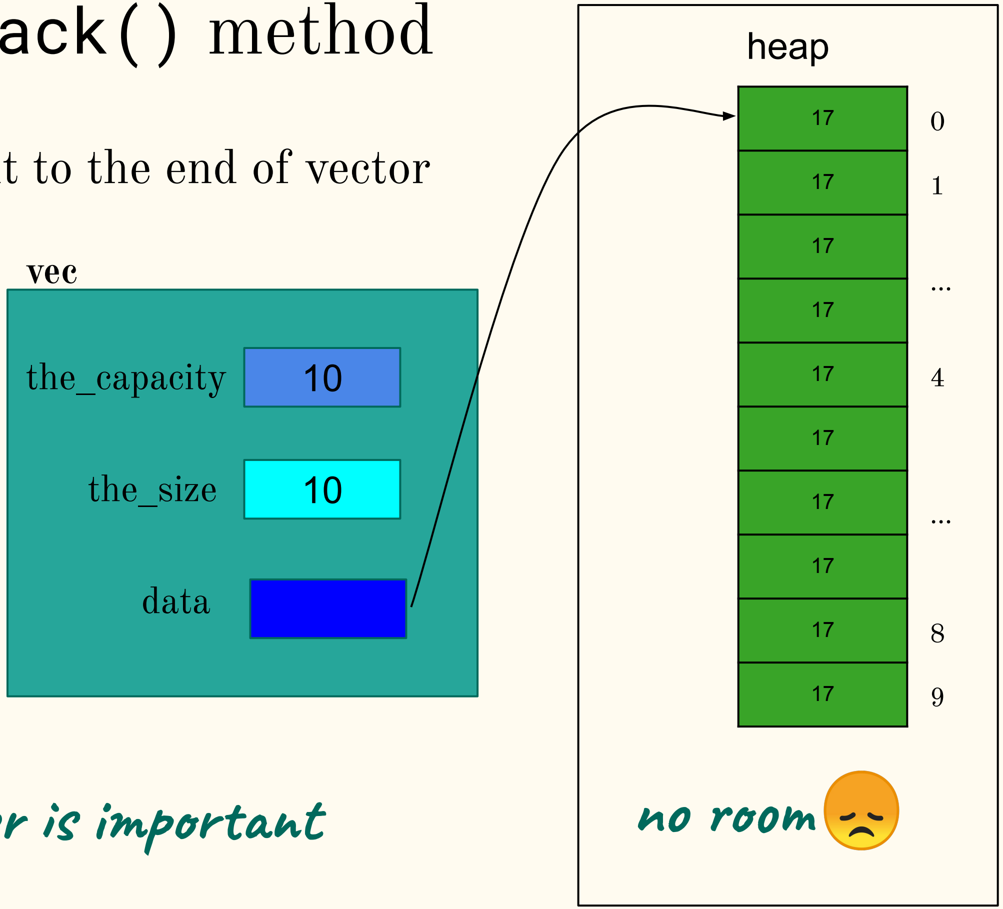
Implementing a `push_back()` method

- `push_back()` adds an element to the end of vector
  - ○ element added to end of array
- two possibilities
  - ○ array is not full
  - ○ array is full

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) allocate a new, larger array
2) copy values to new array
3) free memory from old array
4) point `data` at new array
5) add new value
6) increment size



order is important

no room 😞

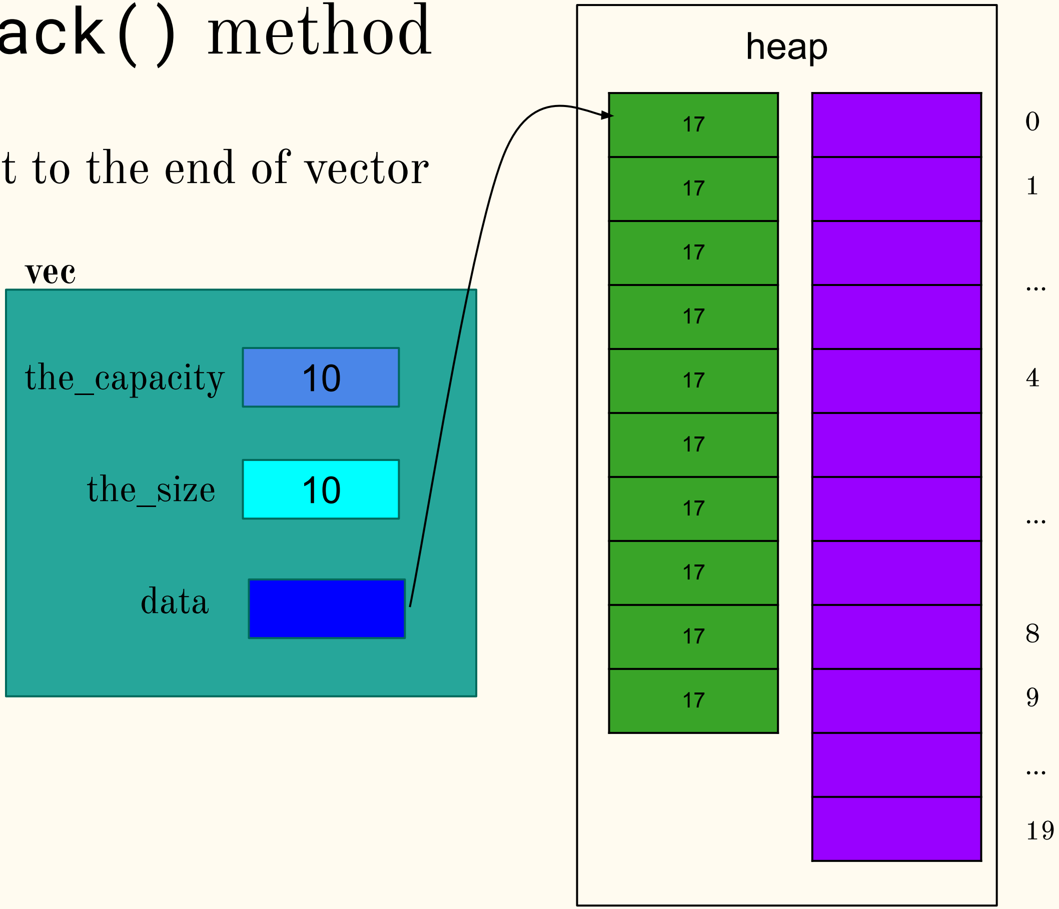
Implementing a `push_back()` method

- `push_back()` adds an element to the end of vector
  - element added to end of array
- two possibilities
  - array is not full
  - array is full

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) allocate a new, larger array



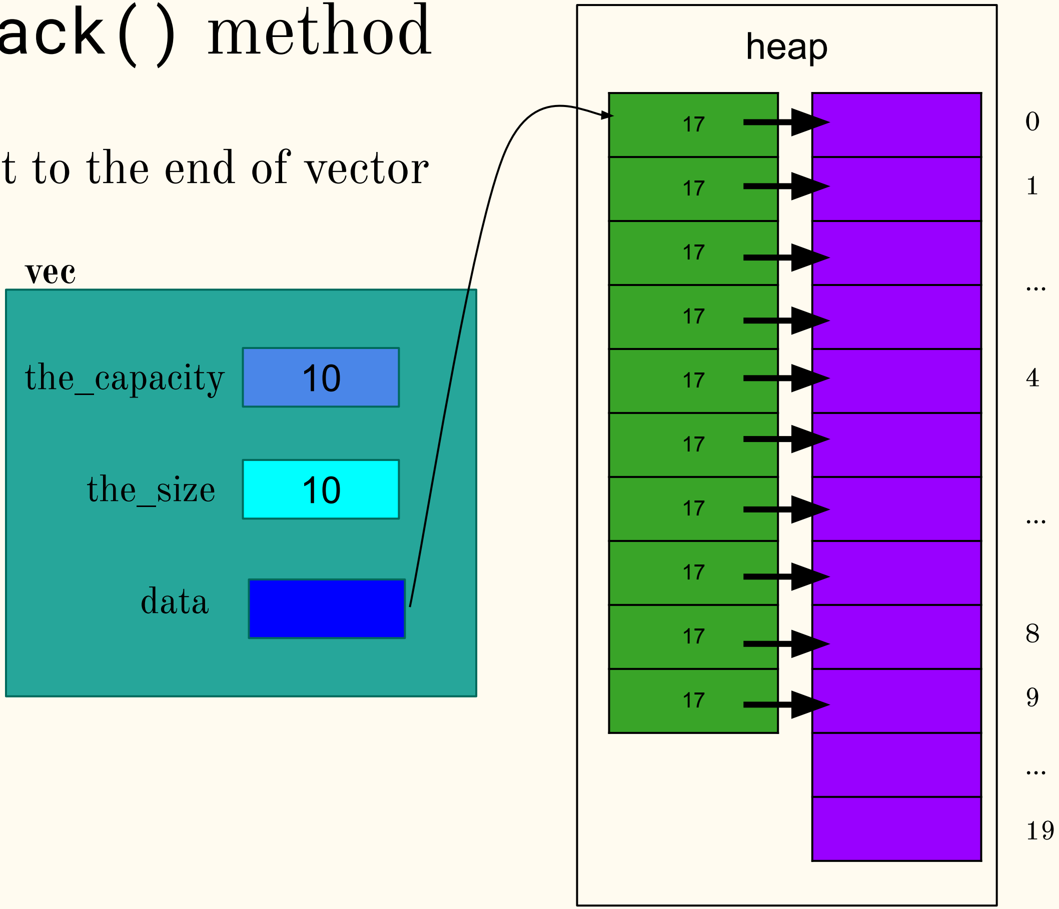
Implementing a push_back() method

- push_back() adds an element to the end of vector
    - element added to end of array
- two possibilities
    - array is not full
    - array is full

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) allocate a new, larger array
2) copy values to new array



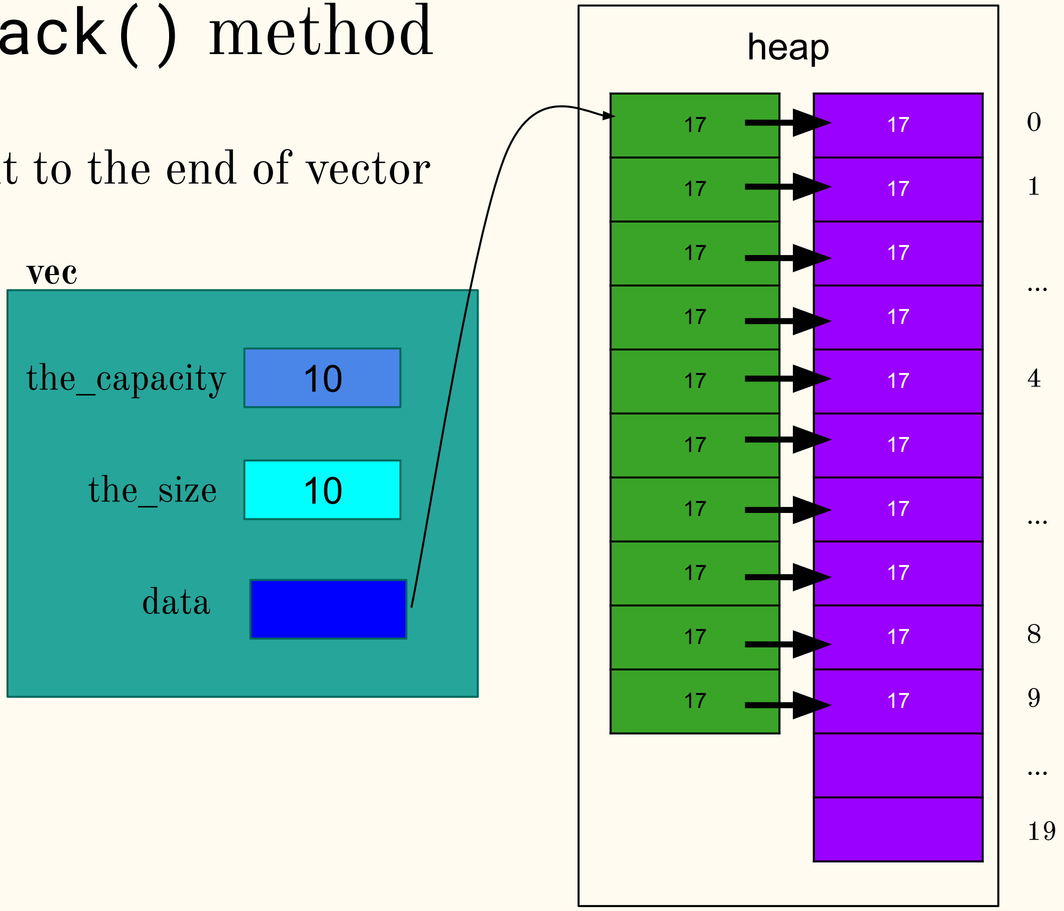
Implementing a push_back() method

- push_back() adds an element to the end of vector
    - element added to end of array
- two possibilities
    - array is not full
    - array is full

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) allocate a new, larger array
2) copy values to new array



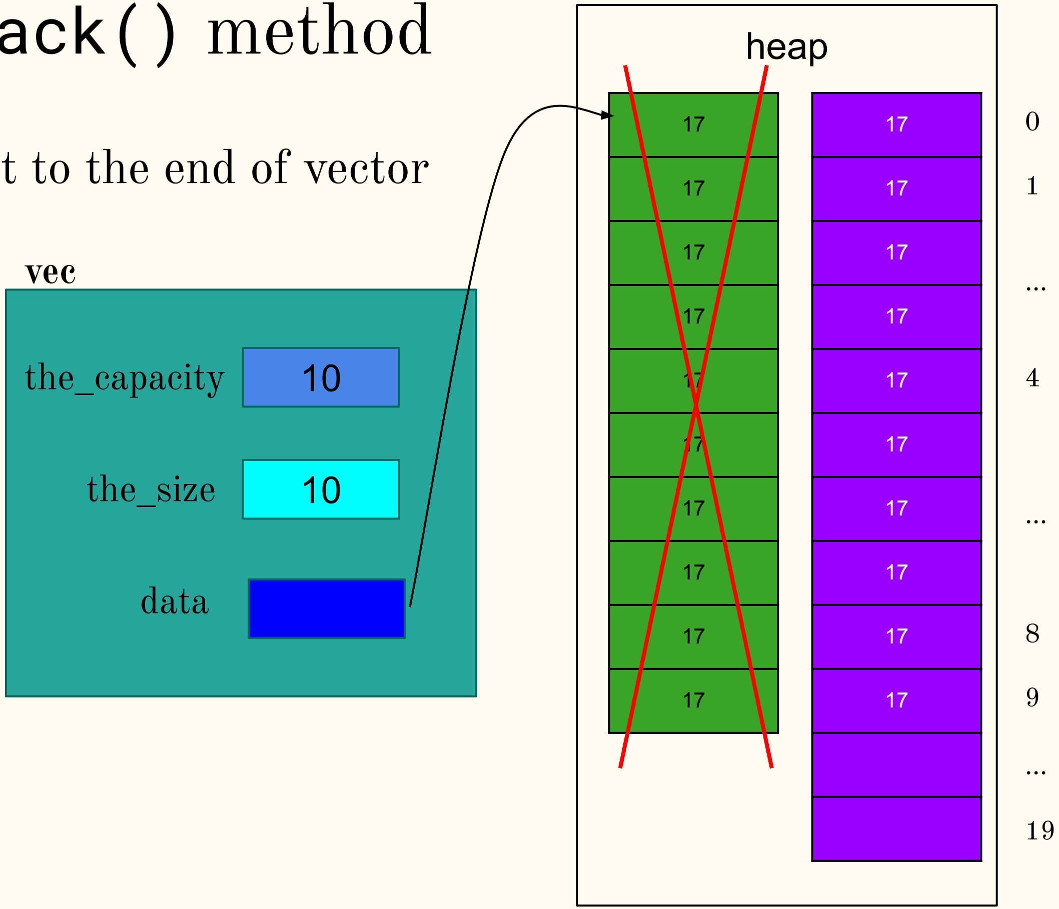
Implementing a `push_back()` method

- `push_back()` adds an element to the end of vector
  - element added to end of array
- two possibilities
  - array is not full
  - array is full

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) allocate a new, larger array
2) copy values to new array
3) free memory from old array



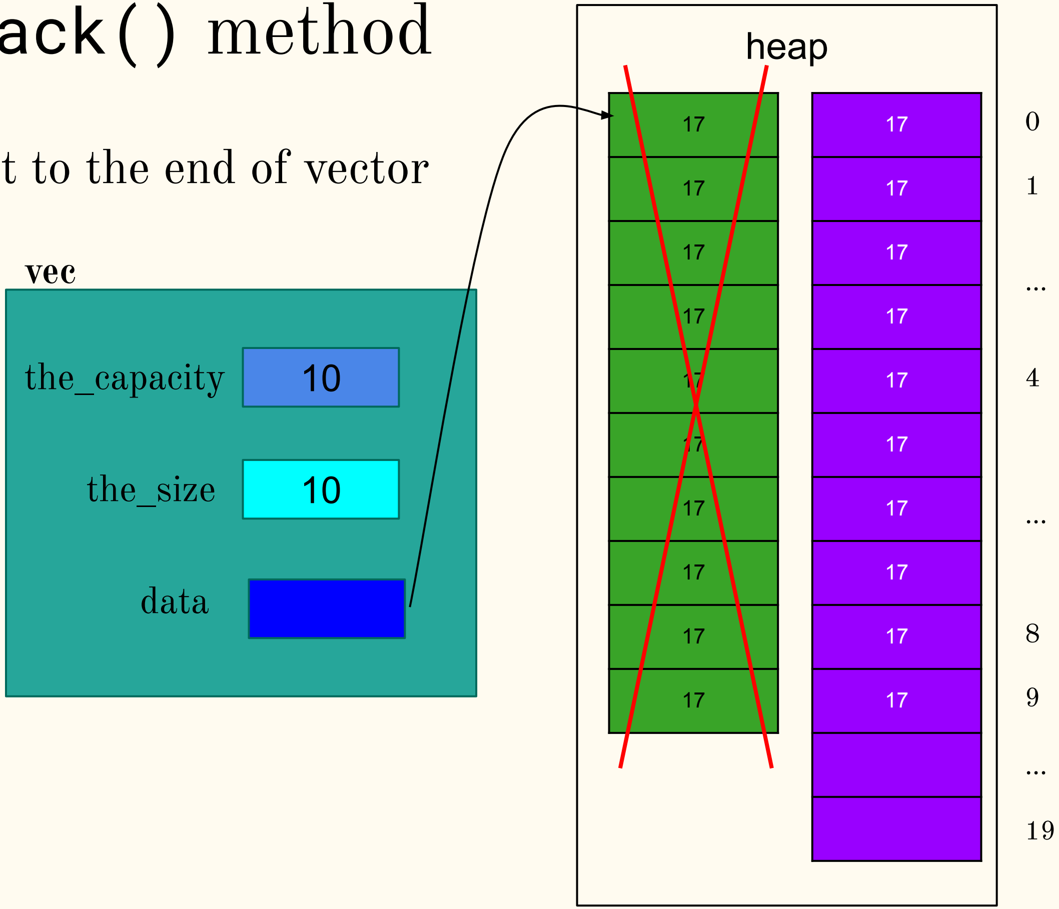
Implementing a `push_back()` method

- `push_back()` adds an element to the end of vector
  - element added to end of array
- two possibilities
  - array is not full
  - array is full

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) allocate a new, larger array
2) copy values to new array
3) free memory from old array
4) point `data` at new array



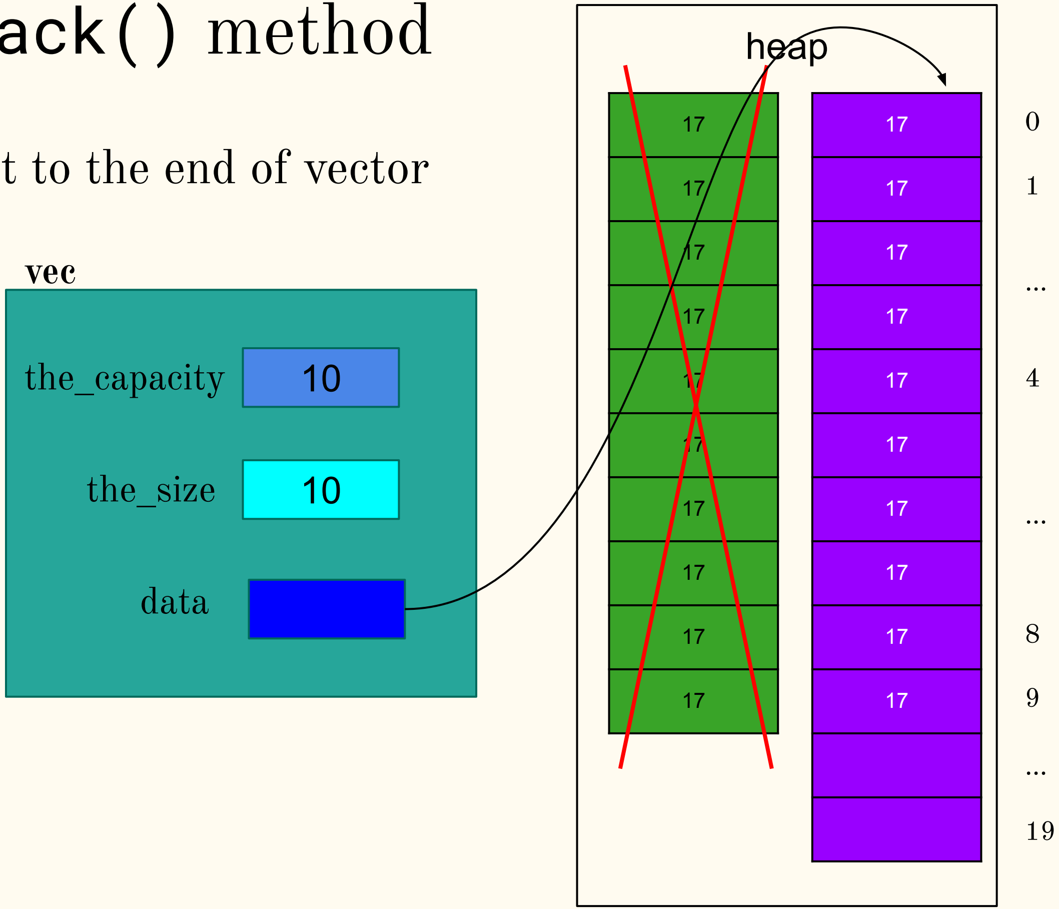
Implementing a `push_back()` method

- `push_back()` adds an element to the end of vector
  - element added to end of array
- two possibilities
  - array is not full
  - array is full

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) allocate a new, larger array
2) copy values to new array
3) free memory from old array
4) point `data` at new array



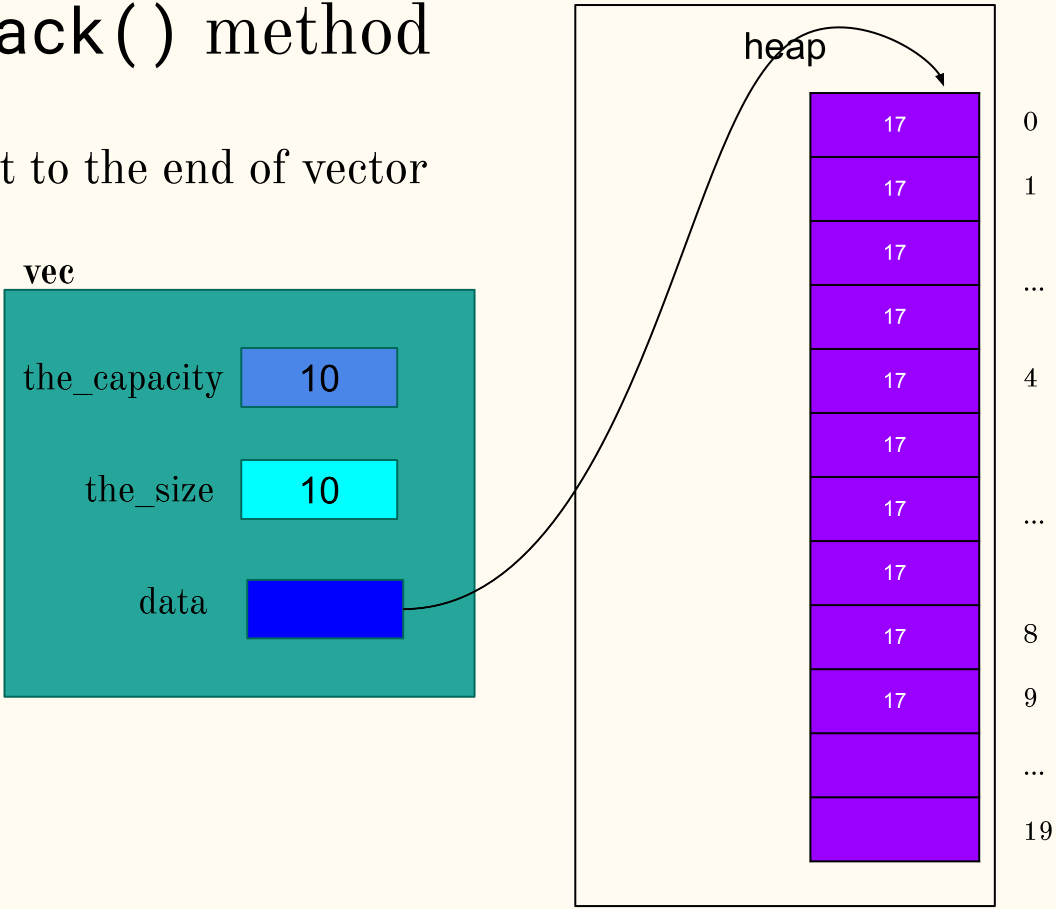
Implementing a `push_back()` method

- `push_back()` adds an element to the end of vector
  - element added to end of array
- two possibilities
  - array is not full
  - array is full

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) allocate a new, larger array
2) copy values to new array
3) free memory from old array
4) point `data` at new array



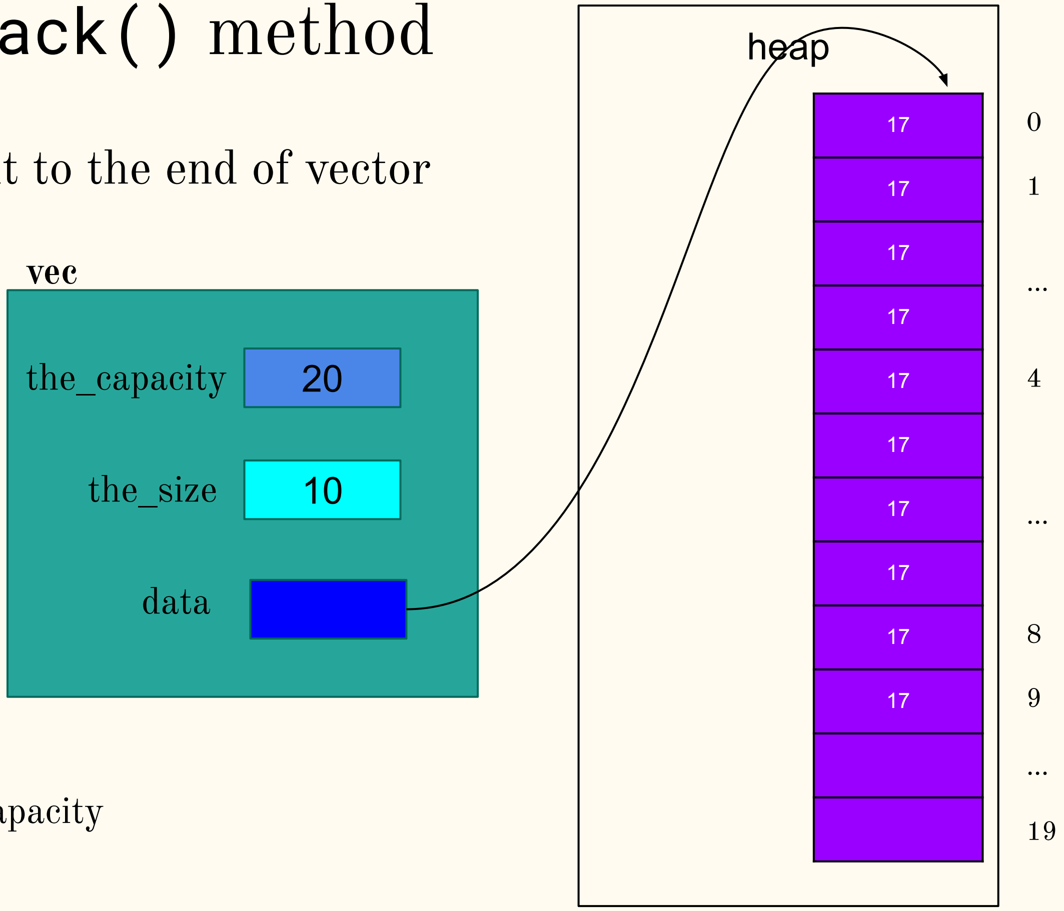
Implementing a `push_back()` method

- `push_back()` adds an element to the end of vector
  - element added to end of array
- two possibilities
  - array is not full
  - array is full

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) allocate a new, larger array
2) copy values to new array
3) free memory from old array
4) point `data` at new array and update capacity



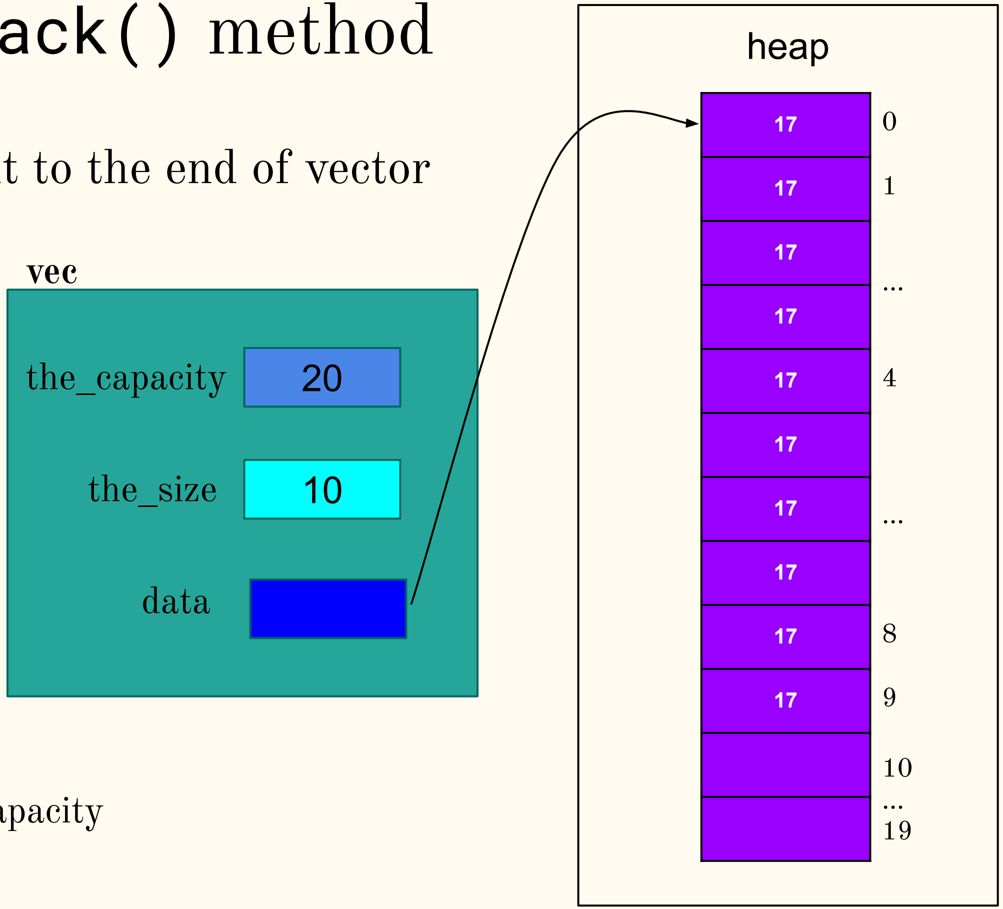
Implementing a `push_back()` method

- `push_back()` adds an element to the end of vector
  - element added to end of array
- two possibilities
  - array is not full
  - array is full

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) allocate a new, larger array
2) copy values to new array
3) free memory from old array
4) point `data` at new array and update capacity



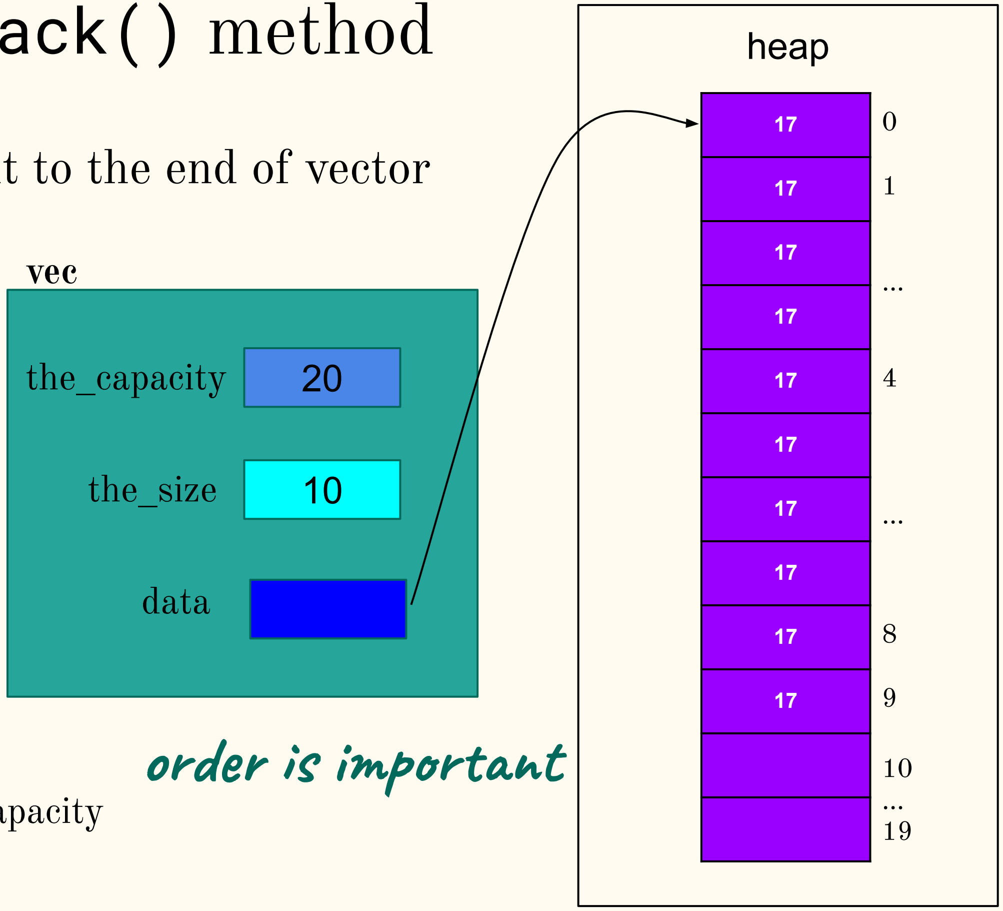
Implementing a `push_back()` method

- `push_back()` adds an element to the end of vector
  - element added to end of array
- two possibilities
  - array is not full
  - array is full

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) allocate a new, larger array
2) copy values to new array
3) free memory from old array
4) point `data` at new array and update capacity



order is important

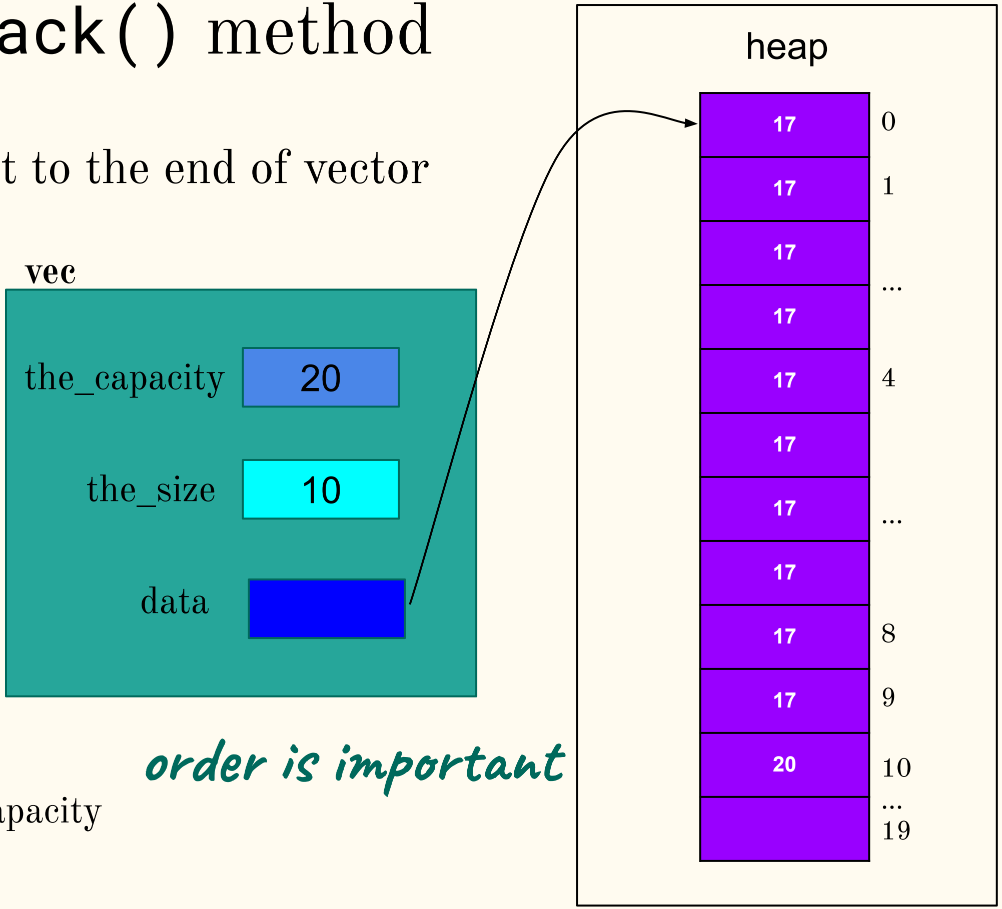
Implementing a `push_back()` method

- `push_back()` adds an element to the end of vector
  - element added to end of array
- two possibilities
  - array is not full
  - array is full

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) allocate a new, larger array
2) copy values to new array
3) free memory from old array
4) point `data` at new array and update capacity
5) add new value



order is important

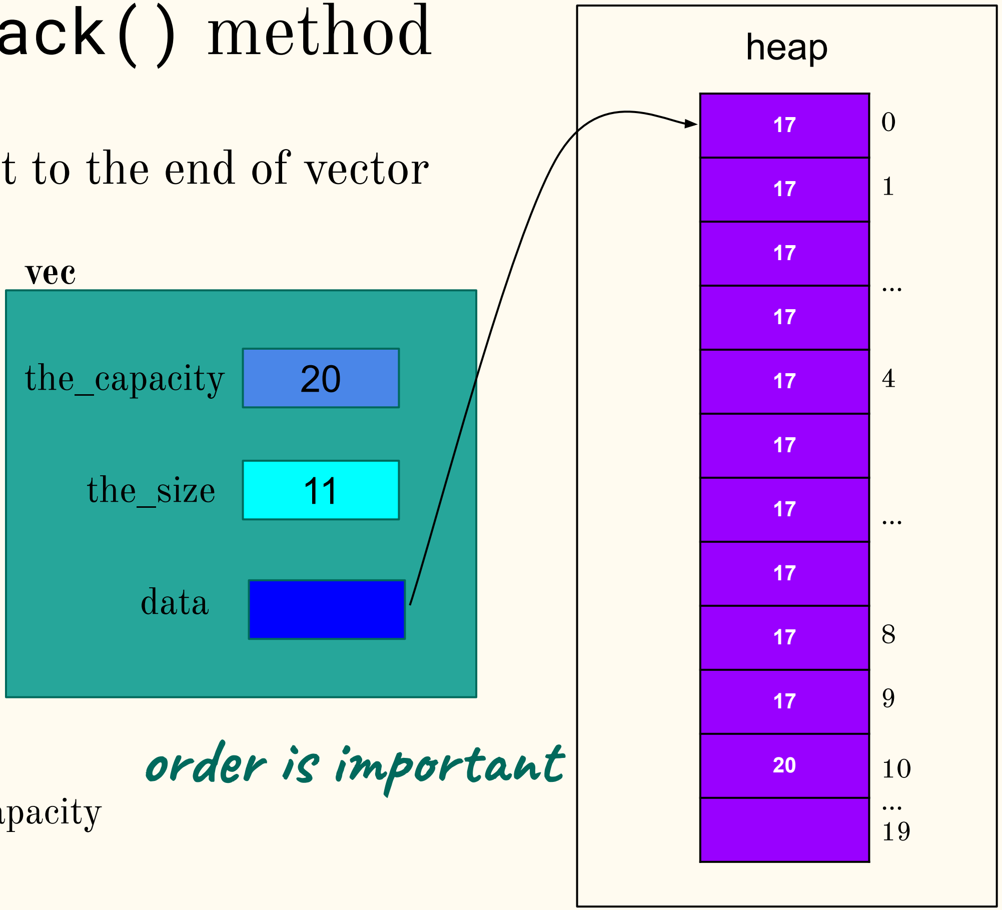
Implementing a `push_back()` method

- `push_back()` adds an element to the end of vector
  - element added to end of array
- two possibilities
  - array is not full
  - array is full

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) allocate a new, larger array
2) copy values to new array
3) free memory from old array
4) point `data` at new array and update capacity
5) add new value
6) increment size



order is important

Implementing a `push_back()` method

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) allocate a new, larger array
2) copy values to new array
3) free memory from old array
4) point `data` at new array and update capacity
5) add new value
6) increment size

```
class Vector {
public:
    ...
    Vector& operator=(const Vector& rhs) {
        if (this != &rhs) {
            delete [] data;
            data = new int[the_capacity];
            the_size = rhs.the_size;
            the_capacity = rhs.the_capacity;
            for (size_t i = 0; i < the_size; ++i) {
                data[i] = rhs.data[i];
            }
        }
        return *this;
    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Implementing a `push_back()` method

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) allocate a new, larger array
2) copy values to new array
3) free memory from old array
4) point `data` at new array and update capacity
5) add new value
6) increment size

```
class Vector {
public:
    ... // constructors, destructor, assignment
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Implementing a `push_back()` method

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) allocate a new, larger array
2) copy values to new array
3) free memory from old array
4) point `data` at new array and update capacity
5) add new value
6) increment size

```
class Vector {
public:
    ... // constructors, destructor, assignment

    // implement push_back()
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Implementing a `push_back()` method

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) allocate a new, larger array
2) copy values to new array
3) free memory from old array
4) point `data` at new array and update capacity
5) add new value
6) increment size

```
class Vector {
public:
    ... // constructors, destructor, assignment

    void push_back(int val) {

    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Implementing a `push_back()` method

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) allocate a new, larger array
2) copy values to new array
3) free memory from old array
4) point `data` at new array and update capacity
5) add new value
6) increment size

```
class Vector {
public:
    ... // constructors, destructor, assignment

    void push_back(int val) {
        // in case of a vector with capacity of 0
        if (the_capacity == 0) {

        }


    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Implementing a `push_back()` method

```
Vector vec(10, 17);

vec.push_back(20);
```

Requirements to increase capacity

1) allocate a new, larger array
2) copy values to new array
3) free memory from old array
4) point `data` at new array and update capacity
5) add new value
6) increment size

```
class Vector {
public:
    ... // constructors, destructor, assignment

    void push_back(int val) {
        // in case of a vector with capacity of 0
        if (the_capacity == 0) {
            delete [] data;
            ++the_capacity;
            data = new int[the_capacity];
        }
    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Implementing a `push_back()` method

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) allocate a new, larger array
2) copy values to new array
3) free memory from old array
4) point `data` at new array and update capacity
5) add new value
6) increment size

```
class Vector {
public:
    ... // constructors, destructor, assignment

    void push_back(int val) {
        // in case of a vector with capacity of 0
        if (the_capacity == 0) {
            delete [] data;
            ++the_capacity;
            data = new int[the_capacity];

        }
        // more room needed?
    }


private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Implementing a `push_back()` method

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) ~~allocate a new, larger array~~
2) copy values to new array
3) free memory from old array
4) point `data` at new array and update capacity
5) add new value
6) increment size

```
class Vector {
public:
    ... // constructors, destructor, assignment

    void push_back(int val) {
        // in case of a vector with capacity of 0
        if (the_capacity == 0) {
            delete [] data;
            ++the_capacity;
            data = new int[the_capacity];
        }
        if (the_size == the_capacity) {
            int* new_data = new int[2 * the_capacity];

        }

    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Implementing a `push_back()` method

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) ~~allocate a new, larger array~~
2) ~~copy values to new array~~
3) free memory from old array
4) point `data` at new array and update capacity
5) add new value
6) increment size

```
class Vector {
public:
    ... // constructors, destructor, assignment

    void push_back(int val) {
        // in case of a vector with capacity of 0
        if (the_capacity == 0) {
            delete [] data;
            ++the_capacity;
            data = new int[the_capacity];
        }
        if (the_size == the_capacity) {
            int* new_data = new int[2 * the_capacity];
            for (size_t i = 0; i < the_size; ++i) {
                new_data[i] = data[i];
            }

        }

    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Implementing a push_back() method

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) ~~allocate a new, larger array~~
2) ~~copy values to new array~~
3) ~~free memory from old array~~
4) point `data` at new array and update capacity
5) add new value
6) increment size

```
class Vector {
public:
    ... // constructors, destructor, assignment

    void push_back(int val) {
        // in case of a vector with capacity of 0
        if (the_capacity == 0) {
            delete [] data;
            ++the_capacity;
            data = new int[the_capacity];
        }
        if (the_size == the_capacity) {
            int* new_data = new int[2 * the_capacity];
            for (size_t i = 0; i < the_size; ++i) {
                new_data[i] = data[i];
            }
            delete [] data;

        }

    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Implementing a `push_back()` method

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) ~~allocate a new, larger array~~
2) ~~copy values to new array~~
3) ~~free memory from old array~~
4) ~~point `data` at new array and update capacity~~
5) add new value
6) increment size

```
class Vector {
public:
    ... // constructors, destructor, assignment

    void push_back(int val) {
        // in case of a vector with capacity of 0
        if (the_capacity == 0) {
            delete [] data;
            ++the_capacity;
            data = new int[the_capacity];
        }
        if (the_size == the_capacity) {
            int* new_data = new int[2 * the_capacity];
            for (size_t i = 0; i < the_size; ++i) {
                new_data[i] = data[i];
            }
            delete [] data;
            data = new_data;
            the_capacity *= 2;

        }

    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Implementing a `push_back()` method

```
Vector vec(10, 17);

vec.push_back(20);
```

Requirements to increase capacity

1) ~~allocate a new, larger array~~
2) ~~copy values to new array~~
3) ~~free memory from old array~~
4) ~~point `data` at new array and update capacity~~
5) add new value
6) increment size

```
class Vector {
public:
    ... // constructors, destructor, assignment

    void push_back(int val) {
        // in case of a vector with capacity of 0
        if (the_capacity == 0) {
            delete [] data;
            ++the_capacity;
            data = new int[the_capacity];
        }
        if (the_size == the_capacity) {
            int* new_data = new int[2 * the_capacity];
            for (size_t i = 0; i < the_size; ++i) {
                new_data[i] = data[i];
            }
            delete [] data;
            data = new_data;
            the_capacity *= 2;
        }
        // add val and update size
    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Implementing a `push_back()` method

```
Vector vec(10, 17);
vec.push_back(20);
```

Requirements to increase capacity

1) ~~allocate a new, larger array~~
2) ~~copy values to new array~~
3) ~~free memory from old array~~
4) ~~point `data` at new array and update capacity~~
5) ~~add new value~~
6) increment size

```
class Vector {
public:
    ... // constructors, destructor, assignment

    void push_back(int val) {
        // in case of a vector with capacity of 0
        if (the_capacity == 0) {
            delete [] data;
            ++the_capacity;
            data = new int[the_capacity];
        }
        if (the_size == the_capacity) {
            int* new_data = new int[2 * the_capacity];
            for (size_t i = 0; i < the_size; ++i) {
                new_data[i] = data[i];
            }
            delete [] data;
            data = new_data;
            the_capacity *= 2;
        }
        data[the_size] = val;
    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Implementing a `push_back()` method

```
Vector vec(10, 17);

vec.push_back(20);
```

Requirements to increase capacity

1) ~~allocate a new, larger array~~
2) ~~copy values to new array~~
3) ~~free memory from old array~~
4) ~~point `data` at new array and update capacity~~
5) ~~add new value~~
6) ~~increment size~~

```
class Vector {
public:
    ... // constructors, destructor, assignment

    void push_back(int val) {
        // in case of a vector with capacity of 0
        if (the_capacity == 0) {
            delete [] data;
            ++the_capacity;
            data = new int[the_capacity];
        }
        if (the_size == the_capacity) {
            int* new_data = new int[2 * the_capacity];
            for (size_t i = 0; i < the_size; ++i) {
                new_data[i] = data[i];
            }
            delete [] data;
            data = new_data;
            the_capacity *= 2;
        }
        data[the_size] = val;
        ++the_size;
    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Implementing a `push_back()` method

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);
}
```

how do we know if implementation correct?

```
class Vector {
public:
    ... // constructors, destructor, assignment

    void push_back(int val) {
        // in case of a vector with capacity of 0
        if (the_capacity == 0) {
            delete [] data;
            ++the_capacity;
            data = new int[the_capacity];
        }
        if (the_size == the_capacity) {
            int* new_data = new int[2 * the_capacity];
            for (size_t i = 0; i < the_size; ++i) {
                new_data[i] = data[i];
            }
            delete [] data;
            data = new_data;
            the_capacity *= 2;
        }
        data[the_size] = val;
        ++the_size;
    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Implementing a `push_back()` method

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);
}
```

how do we know if implementation correct?

Displaying data in Vector

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);

    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

}
```

compilation error

Displaying data in Vector

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);
```

compilation error

```
    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

}
```

```
class Vector {
public:
    ... // constructors, destructor, assignment

    void push_back(int val) {
        // in case of a vector with capacity of 0
        if (the_capacity == 0) {
            delete [] data;
            ++the_capacity;
            data = new int[the_capacity];
        }
        if (the_size == the_capacity) {
            int* new_data = new int[2 * the_capacity];
            for (size_t i = 0; i < the_size; ++i) {
                new_data[i] = data[i];
            }
            delete [] data;
            data = new_data;
            the_capacity *= 2;
        }
        data[the_size] = val;
        ++the_size;
    }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Displaying data in Vector

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);
```

compilation error

```
    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

}
```

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Displaying data in Vector

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);
```

compilation error

```
    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

}
```

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    // implement size() method
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Which special type of method is size() given that it simply returns the value of a private member variable?

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);


    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

}
```

compilation error

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    // implement size() method
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Displaying data in Vector

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);
```

compilation error

```
    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

}
```

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    // implement size() method

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Displaying data in Vector

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);


    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

}
```

compilation error

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    size_t size() const { return the_size; }

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Displaying data in Vector

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);


    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

}
```

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    size_t size() const { return the_size; }

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Displaying data in Vector

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);


    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

}
```

compilation error

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    size_t size() const { return the_size; }

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Implementing the [ ] operator

`vec[i]` :

- desired: `data[i]`
- compiler does not know that

- *operator overloading*
  - providing customized behavior for operators applied to instances of a class
  - informs compiler which member function to call
  - can be member or non-member function (depends on operator)

`vec[i]` matches → `vec.operator[](i)`

[] just part of name

Displaying data in Vector

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);


    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

}
```

compilation error

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    size_t size() const { return the_size; }

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Displaying data in Vector

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);


    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

}
```

compilation error

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    size_t size() const { return the_size; }

    // overload [] operator

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Displaying data in Vector

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);


    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

}
```

~~compilation error~~

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    size_t size() const { return the_size; }

    int operator[](size_t i) const { return data[i]; }

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Displaying data in Vector

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);


    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

}
```

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    size_t size() const { return the_size; }

    int operator[](size_t i) const { return data[i]; }

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Index-based assignment

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);


    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

}
```

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    size_t size() const { return the_size; }

    int operator[](size_t i) const { return data[i]; }

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Index-based assignment

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);


    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

    vec[1] = -5; compilation error

}
```

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    size_t size() const { return the_size; }

    int operator[](size_t i) const { return data[i]; }

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Why does the attempt to assign a new value to `vec` at index 1 produce a compilation error?

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);


    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

    vec[1] = -5; compilation error

}
```

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    size_t size() const { return the_size; }

    int operator[](size_t i) const { return data[i]; }

private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Index-based assignment

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);


    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

    vec[1] = -5; compilation error

}
```

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    size_t size() const { return the_size; }

    int operator[](size_t i) const { return data[i]; }

    // implement operator[] allowing modification
    int operator[](size_t i) ~~const~~ { return data[i]; }


private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Index-based assignment

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);


    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

    vec[1] = -5; compilation error

}
```

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    size_t size() const { return the_size; }

    int operator[](size_t i) const { return data[i]; }

    // implement operator[] allowing modification
    int operator[](size_t i) { return data[i]; }


private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Index-based assignment

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);


    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

    vec[1] = -5; compilation error

}
```

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    size_t size() const { return the_size; }

    int operator[](size_t i) const { return data[i]; }

    // implement operator[] allowing modification
    ___ operator[](size_t i) { return data[i]; }


private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Index-based assignment

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);


    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

    vec[1] = -5; compilation error

}
```

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    size_t size() const { return the_size; }

    int operator[](size_t i) const { return data[i]; }

    // implement operator[] allowing modification
    _1_ operator[](size_t i) { return data[i]; }


private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Which return type replaces blank #1 to allow modification of the integer that is returned?

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);


    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

    vec[1] = -5; compilation error

}
```

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    size_t size() const { return the_size; }

    int operator[](size_t i) const { return data[i]; }

    // implement operator[] allowing modification
    _1_ operator[](size_t i) { return data[i]; }


private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Index-based assignment

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);


    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

    vec[1] = -5; ~~compilation error~~

}
```

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    size_t size() const { return the_size; }

    int operator[](size_t i) const { return data[i]; }

    // implement operator[] allowing modification
    int& operator[](size_t i) { return data[i]; }


private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

Index-based assignment

```
int main() {
    Vector vec;

    vec.push_back(20);
    vec.push_back(47);
    vec.push_back(102);
    vec.push_back(7000);


    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }

    vec[1] = -5;

}
```

```
class Vector {
public:
    ... // constructors, destructor, assignment, push_back()

    size_t size() const { return the_size; }

    int operator[](size_t i) const { return data[i]; }

    int& operator[](size_t i) { return data[i]; }


private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```