**SRS Setup** 

Login: student.turningtechnologies.com

Session ID: 20220228<A|D>

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

### Implementing vectors

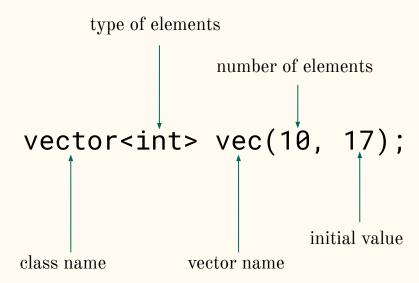
CS 2124: Object Oriented Programming Darryl Reeves, Ph.D.

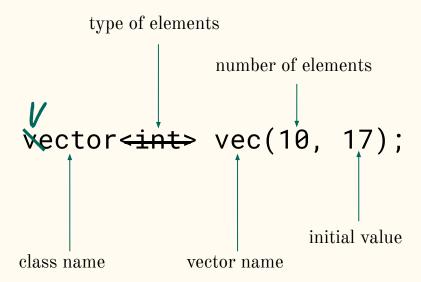
### Agenda

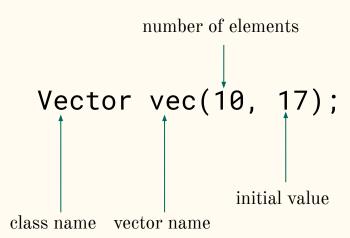
- Vector constructor
- Vector destructor
- Vector copy constructor
- Vector assignment operator
- Vector class methods

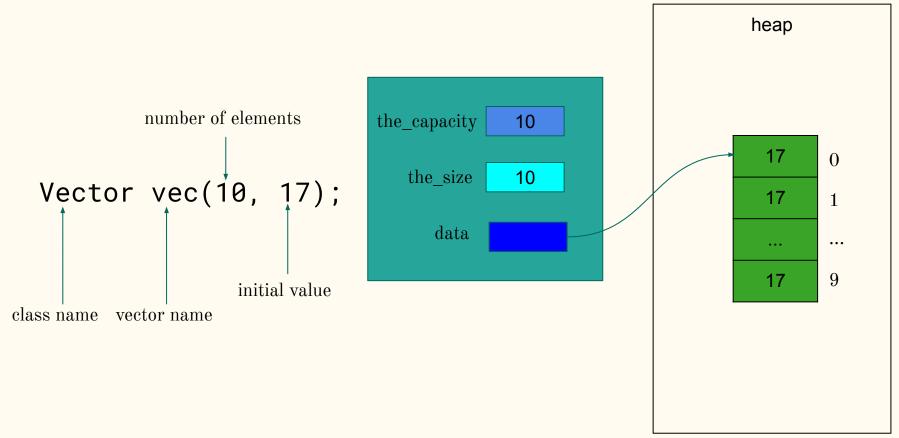
# The Vector constructor

### C++ vector constructor









```
class Vector {
                                 public:
                                     // define constructor
            number of elements
                                 private:
                                     int* data;
                                     size_t the_size;
  Vector vec(10, 17);
                                 size_t the_capacity;
                    initial value
class name vector name
```

```
class Vector {
                                 public:
                                     // define constructor
            number of elements
                                 private:
                                     int* data;
                                     size_t the_size;
  Vector vec(10, 17);
                                 size_t the_capacity;
                    initial value
class name vector name
```

```
class Vector {
                                  public:
                                      Vector(size_t size, int value) {
            number of elements
                                  private:
                                      int* data;
  Vector vec(10, 17);
                                      size_t the_size;
                                      size_t the_capacity;
                     initial value
class name vector name
```

```
class Vector {
                                  public:
                                      Vector(size_t size, int value) {
                                         the_size = size;
            number of elements
                                         the_capacity = size;
                                         data = new int[size];
                                         for (size_t i = 0; i < the_size; ++i) {
  Vector vec(10, 17);
                                             data[i] = value;
                                  private:
                                      int* data;
                     initial value
                                      size_t the_size;
class name vector name
                                      size_t the_capacity;
                                  };
```

```
class Vector {
                                  public:
                                      Vector(size_t size, int value) {
                                          the_size = size;
            number of elements
                                          the_capacity = size;
                                          data = new int[size];
                                          for (size_t i = 0; i < the_size; ++i) {
  Vector vec(10, 17);
                                              data[i] = value;
                                                                  Great
                     initial value
                                  private:
                                      int* data;
class name vector name
                                      size_t the_size;
                                      size_t the_capacity;
                                  };
```

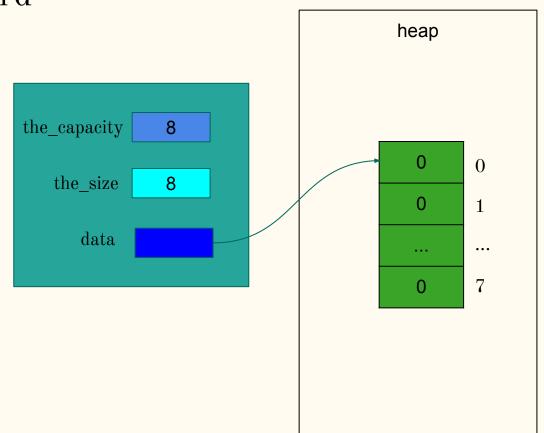
```
class Vector {
                               public:
                                   Vector(size_t size, int value) {
                                       the_size = size;
                                       the_capacity = size;
                                       data = new int[size];
                                       for (size_t i = 0; i < the_size; ++i) {
Vector vec; compilation error
                                           data[i] = value;
                                          add a default constructor?
                               private:
                                   int* data;
                                   size_t the_size;
                                   size_t the_capacity;
                               };
```

```
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) {
Vector vec; compilation error
                                           data[i] = value;
                              private:
                                   int* data;
                                   size_t the_size;
                                   size_t the_capacity;
                               };
```

```
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) {
Vector vec(17);
                                           data[i] = value;
             require size
             parameter
                               private:
                                   int* data;
                                   size_t the_size;
                                   size_t the_capacity;
                               };
```

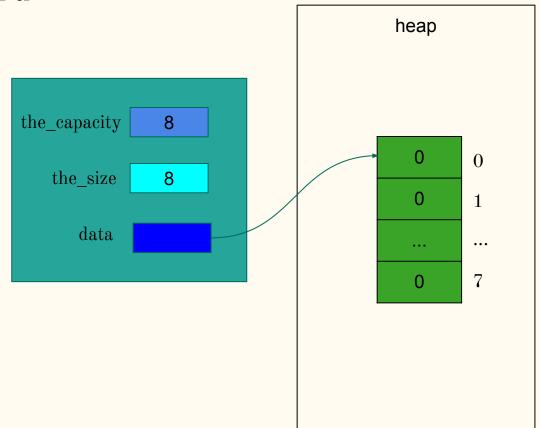
```
Vector vec(17);
Vector vec2(8);
```

vec2 = 65; *typo* 



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

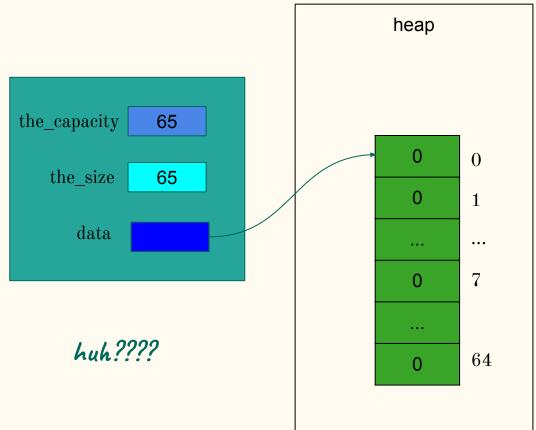
→ vec2 = 65; typo



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

. . .

vec2 = 65; typo



### Implicit conversion

```
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;
vec2 = 65;
         converted by
         compiler to
                              private:
vec2 = Vector(65);
                                  int* data;
                                  size_t the_size;
                                  size_t the_capacity;
                              };
```

```
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;
vec2 = 65; compilation error
                              private:
vec2 = Vector(65);
                                  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

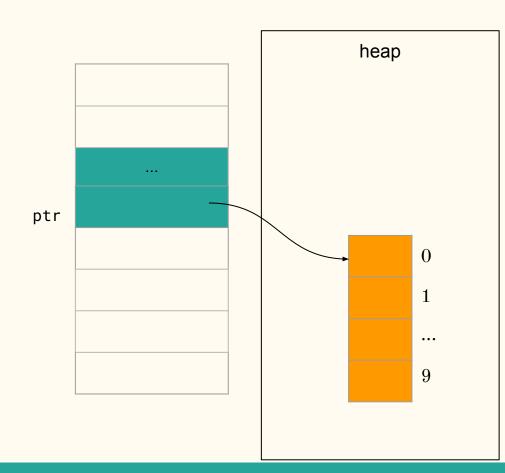
### What responsibility would a Vector class destructor have?

```
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

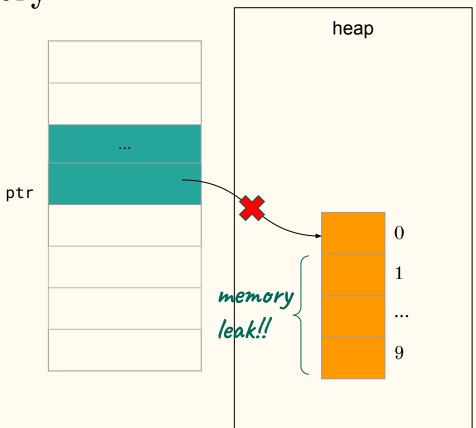
```
class Vector {
public:
    Vector(size_t size = 0, int value = 0) {
        the_size = size;
        the_capacity = size;
        data = new int[size];
        for (size_t i = 0; i < the_size; ++i) {
            data[i] = value;
    // define destructor
private:
    int* data;
    size_t the_size;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

### Creating a dynamic array



### Freeing dynamic array memory

```
int* ptr = new int[10];
delete ptr; // free memory No
```



### Freeing dynamic array memory

```
heap
int* ptr = new int[10];
delete [] ptr; // free memory ✔
                                         ptr
                                                                             0
                                                               freed
```

### 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

```
class Vector {
                                  public:
Vector vec1(10, 17);
                                       Vector(size_t size = 0, int value = 0) {
Vector vec2(vec1); memory error
                                           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;
                                   };
```

```
class Vector {
                                  public:
Vector vec1(10, 17);
                                       Vector(size_t size = 0, int value = 0) {
Vector vec2(vec1); memory error
                                           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;
                                   };
```

```
class Vector {
Vector vec1(10, 17);
Vector vec2(vec1); memory error

// add copy constructor
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

```
class Vector {
                                      public:
Vector vec1(10, 17);
Vector vec2(vec1); memory error
                                           // add copy constructor
                                           Vector(const Vector& rhs) {
Requirements of copy constructor
    copy size and capacity
    allocate memory for data
                                      private:
    copy values to new array
                                           int* data;
                                           size_t the_size;
                                           size_t the_capacity;
```

**}**;

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

#### Requirements of copy constructor

- <u>copy size and capacity</u>
- 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;
};
```

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

#### Requirements of copy constructor

- <u>copy size</u> 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?

```
class Vector {
                                     public:
Vector vec1(10, 17);
Vector vec2(vec1); memory error
                                         Vector(const Vector& rhs) {
                                              the_size = rhs.the_size;
                                              the_capacity = rhs.the_capacity;
                                              data = new int[the_capacity];
Requirements of copy constructor
  copy size and capacity
   - allocate memory for data
                                     private:
    copy values to new array
                                         int* data;
                                          size_t the_size;
                                          size_t the_capacity;
                                     };
```

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

#### Requirements of copy constructor

- <u>copy size</u> 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;
};
```

```
class Vector {
                                     public:
Vector vec1(10, 17);
Vector vec2(vec1); memory error
                                         Vector(const Vector& rhs) {
                                             the_size = rhs.the_size;
                                             the_capacity = rhs.the_capacity;
                                             data = new int[the_capacity];
Requirements of copy constructor
                                             for (size_t i = 0; i < the_size; ++i) {
  -copy size and capacity
                                                 data[i] = rhs.data[i];
   -allocate memory for data
    copy values to new array
                                               hmmm...looks familiar
                                    private:
                                         int* data;
                                         size_t the_size;
                                         size_t the_capacity;
```

**}**;

```
class Vector {
                                    public:
Vector vec1(10, 17);
Vector vec2(vec1);
                                        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;
                                                    constructor
        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;
                                                    copy constructor
        data = new int[the_capacity];
        for (size_t i = 0; i < the_size; ++i) {</pre>
            data[i] = rhs.data[i];
                                                destructor
   ~Vector() { delete [] data; }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

# The assignment operator

```
class Vector {
Vector vec1(10, 17);
                                       public:
Vector vec2(1000, 5);
                                           Vector(const Vector& rhs) {
vec2 = vec1; memory error
                                               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?

```
class Vector {
Vector vec1(10, 17);
                                        public:
Vector vec2(1000, 5);
                                            Vector(const Vector& rhs) {
vec2 = vec1; memory error
                                                the_size = rhs.the_size;
                                                the_capacity = rhs.the_capacity;
                                                data = new int[the_capacity];
                                                for (size_t i = 0; i < the_size; ++i) {</pre>
                                                    data[i] = rhs.data[i];
                                       private:
                                            int* data;
                                            size_t the_size;
                                            size_t the_capacity;
                                       };
```

```
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

```
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;
};
```

```
class Vector {
Vector vec1(10, 17);
                                         public:
Vector vec2(1000, 5);
                                             Vector& operator=(const Vector& rhs) {
vec2 = vec1; memory error
                                                  if (this != &rhs) {
Requirements of assignment operator
    -check for self-assignment
                                         private:
    free old memory (if needed)
                                             int* data:
     allocate new memory (if needed)
                                             size_t the_size;
    copy values
                                             size_t the_capacity;
                                         };
    return proper type and object
```

```
class Vector {
Vector vec1(10, 17);
                                         public:
Vector vec2(1000, 5);
                                             Vector& operator=(const Vector& rhs) {
vec2 = vec1; memory error
                                                  if (this != &rhs) {
                                                      delete [] data;
Requirements of assignment operator
    -check for self-assignment
                                         private:
    free old memory (if needed)
                                             int* data:
     allocate new memory (if needed)
                                             size_t the_size;
     copy values
                                             size_t the_capacity;
                                         };
     return proper type and object
```

```
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;
```

```
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;
};
```

```
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;
};
```

```
class Vector {
Vector vec1(10, 17);
                                         public:
Vector vec2(1000, 5);
                                             Vector& operator=(const Vector& rhs) {
vec2 = vec1; memory error
                                                 if (this != &rhs) {
                                                     delete [] data;
                                                     data = new int[rhs.the_capacity];
                                                     the_size = rhs.the_size;
Requirements of assignment operator
                                                     the_capacity = rhs.the_capacity;
    -check for self-assignment
                                                     for (size_t i = 0; i < the_size; ++i) {
    free old memory (if needed)
                                                         data[i] = rhs.data[i];
    -allocate new memory (if needed)
    copy values
    return proper type and object
                                                 return *this;
                                        private:
                                             int* data;
                                             size_t the_size;
                                             size_t the_capacity;
```

};

```
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;
                                                         constructor
       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;
                                                       copy constructor
       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;
                                                       assignment operator
           the_capacity = rhs.the_capacity;
           for (size_t i = 0; i < the_size; ++i) {
               data[i] = rhs.data[i];
        return *this;
                                                   destructor
   ~Vector() { delete [] data; }
private:
    int* data;
    size_t the_size;
    size_t the_capacity;
};
```

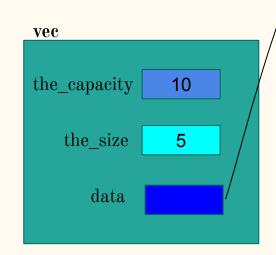
# Vector class methods

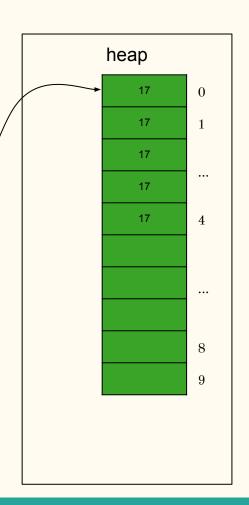
- push\_back() adds an element to the end of vector
  - element added to end of array
- two possibilities
  - o array is not full

Vector vec;

... // modifications

vec.push\_back(20)



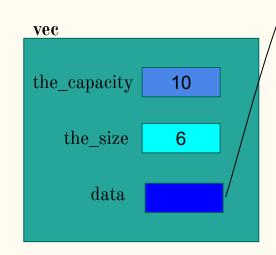


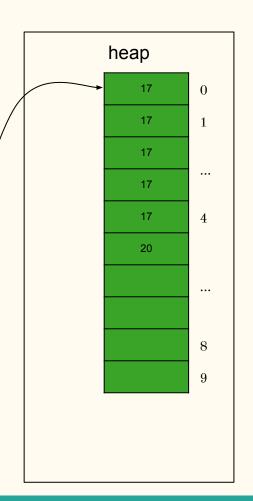
- push\_back() adds an element to the end of vector
  - element added to end of array
- two possibilities
  - array is not full

Vector vec;

... // modifications

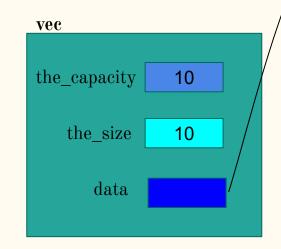
vec.push\_back(20)

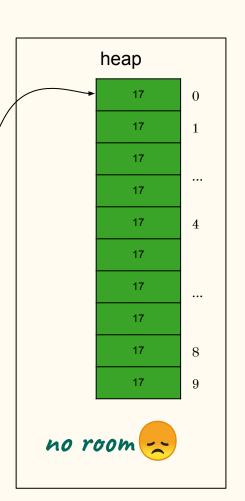




- push\_back() adds an element to the end of vector
  - element added to end of array
- two possibilities
  - o array is not full
  - o array is full

Vector vec(10, 17);
vec.push\_back(20);



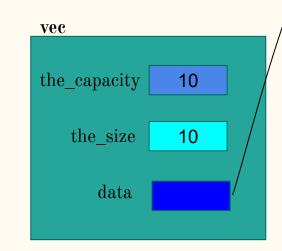


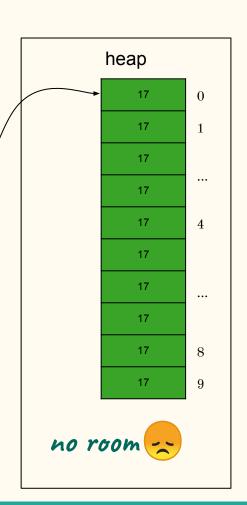
- push\_back() adds an element to the end of vector
  - element added to end of array
- two possibilities
  - o array is not full
  - o array is full

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



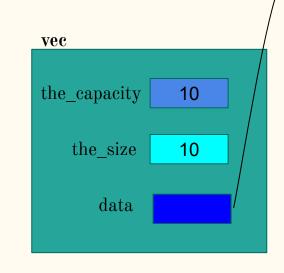


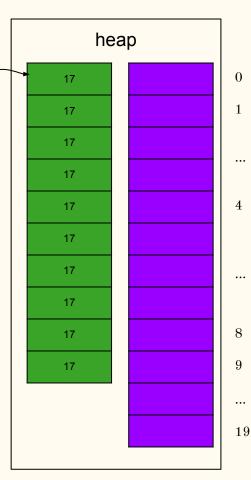
- push\_back() adds an element to the end of vector
  - element added to end of array
- two possibilities
  - o array is not full
  - o array is full

Vector vec(10, 17);
vec.push\_back(20);

Requirements to increase capacity

1) allocate a new, larger array



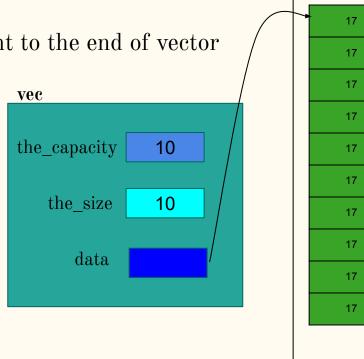


- push\_back() adds an element to the end of vector
  - element added to end of array
- two possibilities
  - array is not full
  - array is full 0

Vector vec(10, 17); vec.push\_back(20);

Requirements to increase capacity

- allocate a new, larger array
- copy values to new array



heap 0

8

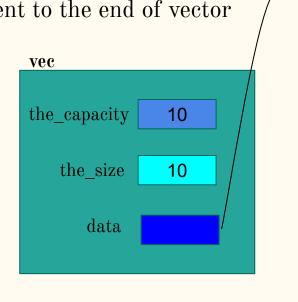
19

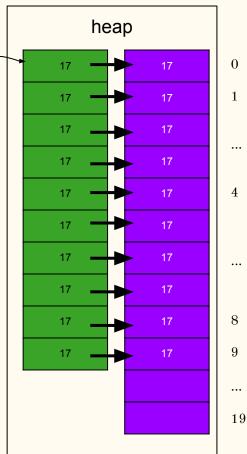
- push\_back() adds an element to the end of vector
  - element added to end of array
- two possibilities
  - o array is not full
  - o array is full

Vector vec(10, 17);
vec.push\_back(20);

Requirements to increase capacity

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



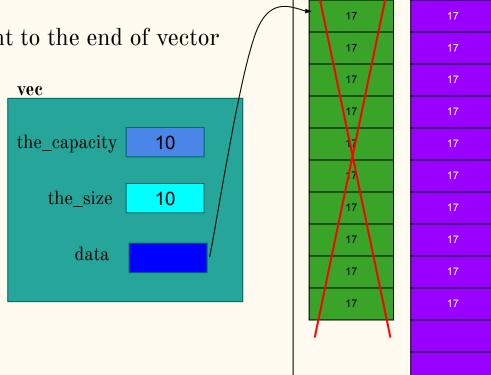


- push\_back() adds an element to the end of vector
  - element added to end of array
- two possibilities
  - o array is not full
  - o array is full

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



heap

0

8

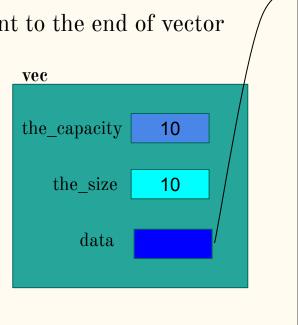
19

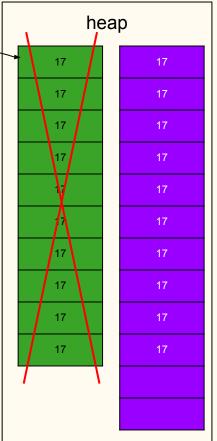
- push\_back() adds an element to the end of vector
  - element added to end of array
- two possibilities
  - o array is not full
  - o array is full

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





0

8

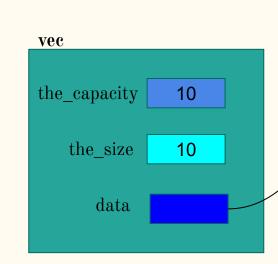
19

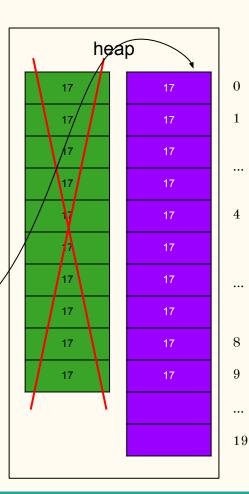
- push\_back() adds an element to the end of vector
  - element added to end of array
- two possibilities
  - o array is not full
  - o array is full

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



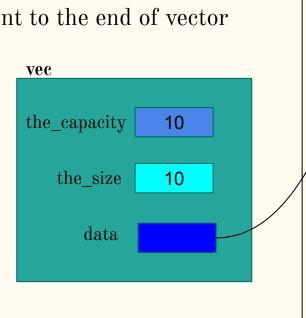


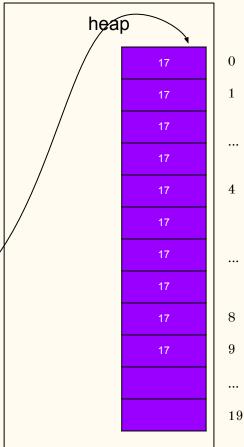
- push\_back() adds an element to the end of vector
  - o element added to end of array
- two possibilities
  - o array is not full
  - o array is full

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



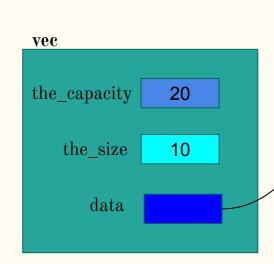


- push\_back() adds an element to the end of vector
  - element added to end of array
- two possibilities
  - o array is not full
  - o array is full

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

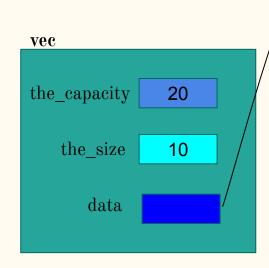


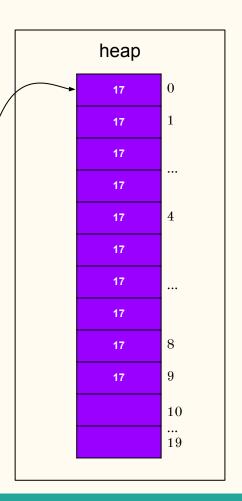
- push\_back() adds an element to the end of vector
  - element added to end of array
- two possibilities
  - o array is not full
  - o array is full

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



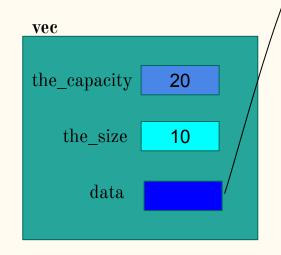


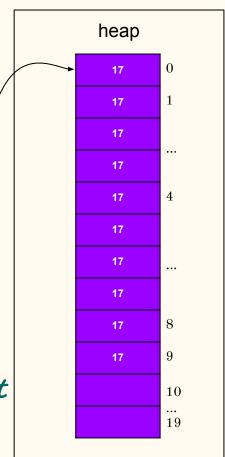
- push\_back() adds an element to the end of vector
  - element added to end of array
- two possibilities
  - o array is not full
  - o array is full

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



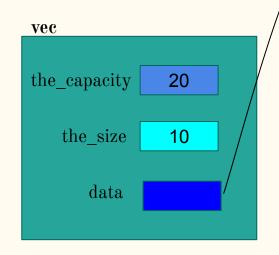


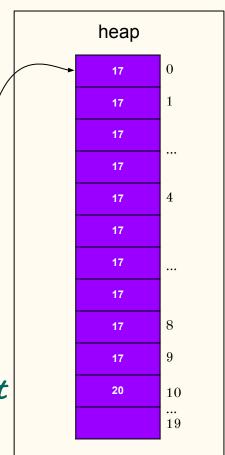
- push\_back() adds an element to the end of vector
  - o element added to end of array
- two possibilities
  - o array is not full
  - o array is full

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



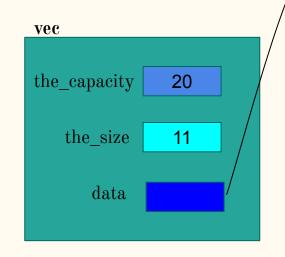


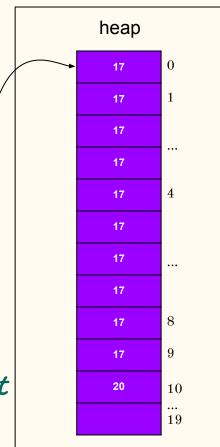
- push\_back() adds an element to the end of vector
  - element added to end of array
- two possibilities
  - o array is not full
  - o array is full

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





```
class Vector {
Vector vec(10, 17);
                                          public:
vec.push_back(20);
                                              Vector& operator=(const Vector& rhs) {
                                                   if (this != &rhs) {
                                                       delete [] data;
Requirements to increase capacity
                                                       data = new int[the_capacity];
1)
     allocate a new, larger array
                                                       the_size = rhs.the_size;
     copy values to new array
                                                       the_capacity = rhs.the_capacity;
     free memory from old array
                                                       for (size_t i = 0; i < the_size; ++i) {
     point data at new array and update capacity
                                                           data[i] = rhs.data[i];
     add new value
5)
6)
     increment size
                                                   return *this;
                                          private:
                                               int* data;
                                               size_t the_size;
                                               size_t the_capacity;
                                          };
```

```
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

6)

increment size

```
class Vector {
                                          public:
Vector vec(10, 17);
                                               ... // constructors, destructor, assignment
vec.push_back(20);
                                               // implement push_back()
Requirements to increase capacity
                                          private:
                                               int* data;
     allocate a new, larger array
                                               size_t the_size;
     copy values to new array
                                               size_t the_capacity;
     free memory from old array
     point data at new array and update capacity
     add new value
5)
```

6)

increment size

```
class Vector {
                                                public:
Vector vec(10, 17);
                                                   ... // constructors, destructor, assignment
vec.push_back(20);
                                                   void push_back(int val) {
Requirements to increase capacity
                                                private:
                                                   int* data:
1)
      allocate a new, larger array
                                                   size_t the_size;
     copy values to new array
                                                   size_t the_capacity;
                                                };
     free memory from old array
     point data at new array and update capacity
      add new value
5)
```

```
class Vector {
                                                public:
Vector vec(10, 17);
                                                    ... // constructors, destructor, assignment
vec.push_back(20);
                                                    void push_back(int val) {
                                                        // in case of a vector with capacity of 0
                                                        if (the_capacity == 0) {
Requirements to increase capacity
 1)
      allocate a new, larger array
      copy values to new array
      free memory from old array
                                                private:
      point data at new array and update capacity
                                                    int* data;
                                                    size_t the_size;
      add new value
                                                    size_t the_capacity;
                                                };
 6)
      increment size
```

```
class Vector {
                                                 public:
Vector vec(10, 17);
                                                     ... // constructors, destructor, assignment
vec.push_back(20);
                                                     void push_back(int val) {
                                                        // in case of a vector with capacity of 0
                                                        if (the_capacity == 0) {
Requirements to increase capacity
                                                            delete [] data;
                                                            ++the_capacity;
 1)
      allocate a new, larger array
                                                            data = new int[the_capacity];
      copy values to new array
      free memory from old array
                                                 private:
      point data at new array and update capacity
                                                     int* data;
                                                     size_t the_size;
      add new value
                                                     size_t the_capacity;
                                                 };
 6)
      increment size
```

```
class Vector {
                                                 public:
Vector vec(10, 17);
                                                     ... // constructors, destructor, assignment
vec.push_back(20);
                                                     void push_back(int val) {
                                                         // in case of a vector with capacity of 0
                                                         if (the_capacity == 0) {
Requirements to increase capacity
                                                             delete [] data:
                                                            ++the_capacity;
      allocate a new, larger array
                                                            data = new int[the_capacity];
      copy values to new array
      free memory from old array
                                                         // more room needed?
      point data at new array and update capacity
      add new value
6)
      increment size
                                                 private:
                                                     int* data:
                                                     size t the size:
                                                     size_t the_capacity;
                                                 };
```

```
class Vector {
                                                  public:
Vector vec(10, 17);
                                                      ... // constructors, destructor, assignment
vec.push_back(20);
                                                     void push_back(int val) {
                                                         // in case of a vector with capacity of 0
                                                         if (the_capacity == 0) {
Requirements to increase capacity
                                                             delete [] data:
                                                             ++the_capacity;
     -allocate a new, larger array
                                                             data = new int[the_capacity];
      copy values to new array
                                                         if (the_size == the_capacity) {
      free memory from old array
                                                             int* new_data = new int[2 * the_capacity];
      point data at new array and update capacity
      add new value
6)
      increment size
                                                  private:
                                                     int* data:
                                                     size t the size:
                                                     size_t the_capacity;
                                                  };
```

```
class Vector {
                                                  public:
Vector vec(10, 17);
                                                      ... // constructors, destructor, assignment
vec.push_back(20);
                                                     void push_back(int val) {
                                                         // in case of a vector with capacity of 0
                                                         if (the_capacity == 0) {
Requirements to increase capacity
                                                             delete [] data:
                                                             ++the_capacity;
     -allocate a new, larger array
                                                             data = new int[the_capacity];
     copy values to new array
                                                         if (the_size == the_capacity) {
      free memory from old array
                                                             int* new_data = new int[2 * the_capacity];
                                                             for (size_t i = 0; i < the_size; ++i) {
      point data at new array and update capacity
                                                                 new_data[i] = data[i];
      add new value
      increment size
                                                  private:
                                                      int* data;
                                                      size_t the_size;
                                                      size_t the_capacity;
                                                  };
```

```
class Vector {
                                                  public:
Vector vec(10, 17);
                                                      ... // constructors, destructor, assignment
vec.push_back(20);
                                                     void push_back(int val) {
                                                         // in case of a vector with capacity of 0
                                                         if (the_capacity == 0) {
Requirements to increase capacity
                                                             delete [] data:
                                                             ++the_capacity;
    -allocate a new, larger array
                                                             data = new int[the_capacity];
     copy values to new array
                                                         if (the_size == the_capacity) {
     -free memory from old array
                                                             int* new_data = new int[2 * the_capacity];
                                                             for (size_t i = 0; i < the_size; ++i) {
      point data at new array and update capacity
                                                                 new_data[i] = data[i];
      add new value
                                                             delete [] data:
      increment size
                                                  private:
                                                     int* data:
                                                      size t the size:
                                                      size_t the_capacity;
```

```
class Vector {
                                                  public:
Vector vec(10, 17);
                                                      ... // constructors, destructor, assignment
vec.push_back(20);
                                                      void push_back(int val) {
                                                          // in case of a vector with capacity of 0
                                                         if (the_capacity == 0) {
Requirements to increase capacity
                                                             delete [] data:
                                                             ++the_capacity;
    -allocate a new, larger array
                                                             data = new int[the_capacity];
     -copy values to new array
                                                         if (the_size == the_capacity) {
     -free memory from old array
                                                             int* new_data = new int[2 * the_capacity];
    -point data at new array and update capacity-
                                                             for (size_t i = 0; i < the_size; ++i) {
                                                                 new_data[i] = data[i];
      add new value
                                                             delete [] data:
      increment size
                                                             data = new_data;
                                                             the_capacity *= 2;
                                                  private:
                                                      int* data:
                                                      size_t the_size;
```

};

size\_t the\_capacity;

```
Vector vec(10, 17);
vec.push_back(20);
Requirements to increase capacity
    -allocate a new, larger array
    copy values to new array
    -free memory from old array
   -point data at new array and update capacity-
     add new value
     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;
```

```
Vector vec(10, 17);
vec.push_back(20);
Requirements to increase capacity
    -allocate a new, larger array
    copy values to new array
    -free memory from old array
   point data at new array and update capacity
    add new value
     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;
```

```
public:
Vector vec(10, 17);
vec.push_back(20);
Requirements to increase capacity
    -allocate a new, larger array
    copy values to new array
    -free memory from old array
   point data at new array and update capacity
    add new value
    -increment size
```

```
class Vector {
    ... // 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;
```

```
class Vector {
                                               public:
int main() {
                                                   ... // constructors, destructor, assignment
      Vector vec;
                                                  void push_back(int val) {
                                                      // in case of a vector with capacity of 0
                                                      if (the_capacity == 0) {
      vec.push_back(20);
                                                          delete [] data:
                                                         ++the_capacity;
      vec.push_back(47);
                                                          data = new int[the_capacity];
      vec.push_back(102);
                                                      if (the_size == the_capacity) {
                                                          int* new_data = new int[2 * the_capacity];
      vec.push_back(7000);
                                                          for (size_t i = 0; i < the_size; ++i) {
                                                             new data[i] = data[i]:
                                                          delete [] data:
                                                          data = new_data;
how do we know if
                                                          the capacity *= 2:
implementation correct?
                                                      data[the_size] = val;
                                                      ++the_size;
                                               private:
                                                   int* data:
                                                   size_t the_size;
                                                   size_t the_capacity;
```

```
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?
```

```
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) { compilation error</pre>
        cout << vec[i] << endl;</pre>
```

```
class Vector {
                                                      public:
int main() {
                                                          ... // constructors, destructor, assignment
    Vector vec;
                                                          void push_back(int val) {
    vec.push_back(20);
                                                              // in case of a vector with capacity of 0
    vec.push_back(47);
                                                              if (the_capacity == 0) {
    vec.push_back(102);
                                                                  delete [] data;
    vec.push_back(7000);
                                                                  ++the_capacity;
                                                                  data = new int[the_capacity];
                             compilation error
                                                              if (the_size == the_capacity) {
    for (size_t i = 0; i < vec.size(); ++i) {
                                                                  int* new_data = new int[2 * the_capacity];
        cout << vec[i] << endl;</pre>
                                                                  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;
                                                      };
```

```
class Vector {
                                                    public:
int main() {
                                                        ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                    private:
                                                        int* data;
    vec.push_back(20);
                                                        size_t the_size;
    vec.push_back(47);
                                                        size_t the_capacity;
    vec.push_back(102);
                                                    };
    vec.push_back(7000);
                            compilation error
    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;</pre>
```

```
class Vector {
                                                     public:
int main() {
                                                          ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                         // implement size() method
    vec.push_back(20);
                                                     private:
    vec.push_back(47);
                                                         int* data;
    vec.push_back(102);
                                                         size_t the_size;
    vec.push_back(7000);
                                                         size_t the_capacity;
                                                     };
                             compilation error
    for (size_t i = 0; i < vec.size(); ++i) {</pre>
        cout << vec[i] << endl;</pre>
```

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

```
class Vector {
                                                     public:
int main() {
                                                         ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                         // implement size() method
    vec.push_back(20);
                                                     private:
    vec.push_back(47);
                                                         int* data;
    vec.push_back(102);
                                                         size_t the_size;
    vec.push_back(7000);
                                                         size_t the_capacity;
                            compilation error
    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl:</pre>
```

```
class Vector {
                                                    public:
int main() {
                                                        ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                        // implement size() method
    vec.push_back(20);
    vec.push_back(47);
                                                    private:
    vec.push_back(102);
                                                       int* data;
    vec.push_back(7000);
                                                        size_t the_size;
                                                        size_t the_capacity;
                            compilation error };
    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;</pre>
```

```
class Vector {
                                                    public:
int main() {
                                                        ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                        size_t size() const { return the_size; }
    vec.push_back(20);
    vec.push_back(47);
                                                    private:
    vec.push_back(102);
                                                       int* data;
    vec.push_back(7000);
                                                        size_t the_size;
                                                        size_t the_capacity;
                            compilation error };
    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;</pre>
```

```
class Vector {
                                                     public:
int main() {
                                                         ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                         size_t size() const { return the_size; }
    vec.push_back(20);
    vec.push_back(47);
                                                     private:
    vec.push_back(102);
                                                         int* data;
    vec.push_back(7000);
                                                         size_t the_size;
                                                         size_t the_capacity;
                                                    };
    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;</pre>
```

```
class Vector {
                                                     public:
int main() {
                                                         ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                         size_t size() const { return the_size; }
    vec.push_back(20);
    vec.push_back(47);
                                                     private:
    vec.push_back(102);
                                                        int* data;
    vec.push_back(7000);
                                                         size_t the_size;
                                                         size_t the_capacity;
                                                    };
    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;</pre>
                 compilation error
```

## Implementing the [] operator

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

```
class Vector {
                                                     public:
int main() {
                                                         ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                         size_t size() const { return the_size; }
    vec.push_back(20);
    vec.push_back(47);
                                                     private:
    vec.push_back(102);
                                                        int* data;
    vec.push_back(7000);
                                                         size_t the_size;
                                                         size_t the_capacity;
                                                    };
    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;</pre>
                 compilation error
```

```
class Vector {
                                                     public:
int main() {
                                                         ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                         size_t size() const { return the_size; }
    vec.push_back(20);
    vec.push_back(47);
                                                         // overload [] operator
    vec.push_back(102);
    vec.push_back(7000);
                                                     private:
                                                        int* data;
                                                         size_t the_size;
                                                         size_t the_capacity;
    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;</pre>
                 compilation error
```

```
class Vector {
                                                     public:
int main() {
                                                         ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                         size_t size() const { return the_size; }
    vec.push_back(20);
    vec.push_back(47);
                                                         int operator[](size_t i) const { return data[i]; }
    vec.push_back(102);
    vec.push_back(7000);
                                                     private:
                                                         int* data;
                                                         size_t the_size;
                                                         size_t the_capacity;
    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;</pre>
```

```
class Vector {
                                                     public:
int main() {
                                                         ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                         size_t size() const { return the_size; }
    vec.push_back(20);
    vec.push_back(47);
                                                         int operator[](size_t i) const { return data[i]; }
    vec.push_back(102);
    vec.push_back(7000);
                                                     private:
                                                         int* data;
                                                         size_t the_size;
                                                         size_t the_capacity;
    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;</pre>
```

```
class Vector {
                                                     public:
int main() {
                                                         ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                         size_t size() const { return the_size; }
    vec.push_back(20);
    vec.push_back(47);
                                                         int operator[](size_t i) const { return data[i]; }
    vec.push_back(102);
    vec.push_back(7000);
                                                     private:
                                                         int* data;
                                                         size_t the_size;
                                                         size_t the_capacity;
    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;</pre>
```

```
class Vector {
                                                     public:
int main() {
                                                         ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                         size_t size() const { return the_size; }
    vec.push_back(20);
    vec.push_back(47);
                                                        int operator[](size_t i) const { return data[i]; }
    vec.push_back(102);
    vec.push_back(7000);
                                                     private:
                                                        int* data;
                                                        size_t the_size;
                                                        size_t the_capacity;
    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;</pre>
    vec[1] = -5; compilation error
```

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

```
class Vector {
                                                     public:
int main() {
                                                         ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                         size_t size() const { return the_size; }
    vec.push_back(20);
    vec.push_back(47);
                                                         int operator[](size_t i) const { return data[i]; }
    vec.push_back(102);
    vec.push_back(7000);
                                                     private:
                                                         int* data:
                                                         size_t the_size;
                                                         size_t the_capacity;
    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;</pre>
    vec[1] = -5; compilation error
```

```
class Vector {
                                                     public:
int main() {
                                                         ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                         size_t size() const { return the_size; }
    vec.push_back(20);
    vec.push_back(47);
                                                         int operator[](size_t i) const { return data[i]; }
    vec.push_back(102);
    vec.push_back(7000);
                                                         // implement operator[] allowing modification
                                                         int operator[](size_t i) const { return data[i]; }
    for (size_t i = 0; i < vec.size(); ++i) {
                                                     private:
        cout << vec[i] << endl;</pre>
                                                         int* data:
                                                         size_t the_size;
                                                         size_t the_capacity;
    vec[1] = -5; compilation error
                                                     };
```

```
class Vector {
                                                     public:
int main() {
                                                         ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                         size_t size() const { return the_size; }
    vec.push_back(20);
    vec.push_back(47);
                                                         int operator[](size_t i) const { return data[i]; }
    vec.push_back(102);
    vec.push_back(7000);
                                                         // implement operator[] allowing modification
                                                         int operator[](size_t i) { return data[i]; }
    for (size_t i = 0; i < vec.size(); ++i) {
                                                     private:
        cout << vec[i] << endl;</pre>
                                                         int* data:
                                                         size_t the_size;
                                                         size_t the_capacity;
    vec[1] = -5; compilation error
                                                     };
```

```
class Vector {
                                                     public:
int main() {
                                                         ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                         size_t size() const { return the_size; }
    vec.push_back(20);
    vec.push_back(47);
                                                         int operator[](size_t i) const { return data[i]; }
    vec.push_back(102);
    vec.push_back(7000);
                                                         // implement operator[] allowing modification
                                                         ___ operator[](size_t i) { return data[i]; }
    for (size_t i = 0; i < vec.size(); ++i) {
                                                     private:
        cout << vec[i] << endl;</pre>
                                                         int* data:
                                                         size_t the_size;
                                                         size_t the_capacity;
    vec[1] = -5; compilation error
                                                     };
```

```
class Vector {
                                                     public:
int main() {
                                                         ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                         size_t size() const { return the_size; }
    vec.push_back(20);
    vec.push_back(47);
                                                         int operator[](size_t i) const { return data[i]; }
    vec.push_back(102);
    vec.push_back(7000);
                                                         // implement operator[] allowing modification
                                                         _1_ operator[](size_t i) { return data[i]; }
    for (size_t i = 0; i < vec.size(); ++i) {
                                                     private:
        cout << vec[i] << endl;</pre>
                                                         int* data:
                                                         size_t the_size;
                                                         size_t the_capacity;
    vec[1] = -5; compilation error
                                                     };
```

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

```
class Vector {
                                                     public:
int main() {
                                                         ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                         size_t size() const { return the_size; }
    vec.push_back(20);
    vec.push_back(47);
                                                         int operator[](size_t i) const { return data[i]; }
    vec.push_back(102);
    vec.push_back(7000);
                                                         // implement operator[] allowing modification
                                                         _1_ operator[](size_t i) { return data[i]; }
    for (size_t i = 0; i < vec.size(); ++i) {
                                                     private:
        cout << vec[i] << endl;</pre>
                                                         int* data:
                                                         size t the size:
                                                         size_t the_capacity;
    vec[1] = -5; compilation error
                                                     };
```

```
class Vector {
                                                     public:
int main() {
                                                         ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                         size_t size() const { return the_size; }
    vec.push_back(20);
    vec.push_back(47);
                                                         int operator[](size_t i) const { return data[i]; }
    vec.push_back(102);
    vec.push_back(7000);
                                                         // implement operator[] allowing modification
                                                         int& operator[](size_t i) { return data[i]; }
    for (size_t i = 0; i < vec.size(); ++i) {
                                                     private:
        cout << vec[i] << endl;</pre>
                                                         int* data:
                                                         size_t the_size;
                                                         size_t the_capacity;
    vec[1] = -5; compilation er
                                                     };
```

```
class Vector {
                                                      public:
int main() {
                                                          ... // constructors, destructor, assignment, push_back()
    Vector vec;
                                                          size_t size() const { return the_size; }
    vec.push_back(20);
    vec.push_back(47);
                                                         int operator[](size_t i) const { return data[i]; }
    vec.push_back(102);
                                                         int& operator[](size_t i) { return data[i]; }
    vec.push_back(7000);
                                                     private:
    for (size_t i = 0; i < vec.size(); ++i) {</pre>
                                                         int* data;
        cout << vec[i] << endl;</pre>
                                                         size_t the_size;
                                                          size_t the_capacity;
                                                     };
    vec[1] = -5:
```