SRS Setup
Login: student.turningtechnologies.com
Session ID: 20220302<A|D>

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

Implementing Vectors II

—

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

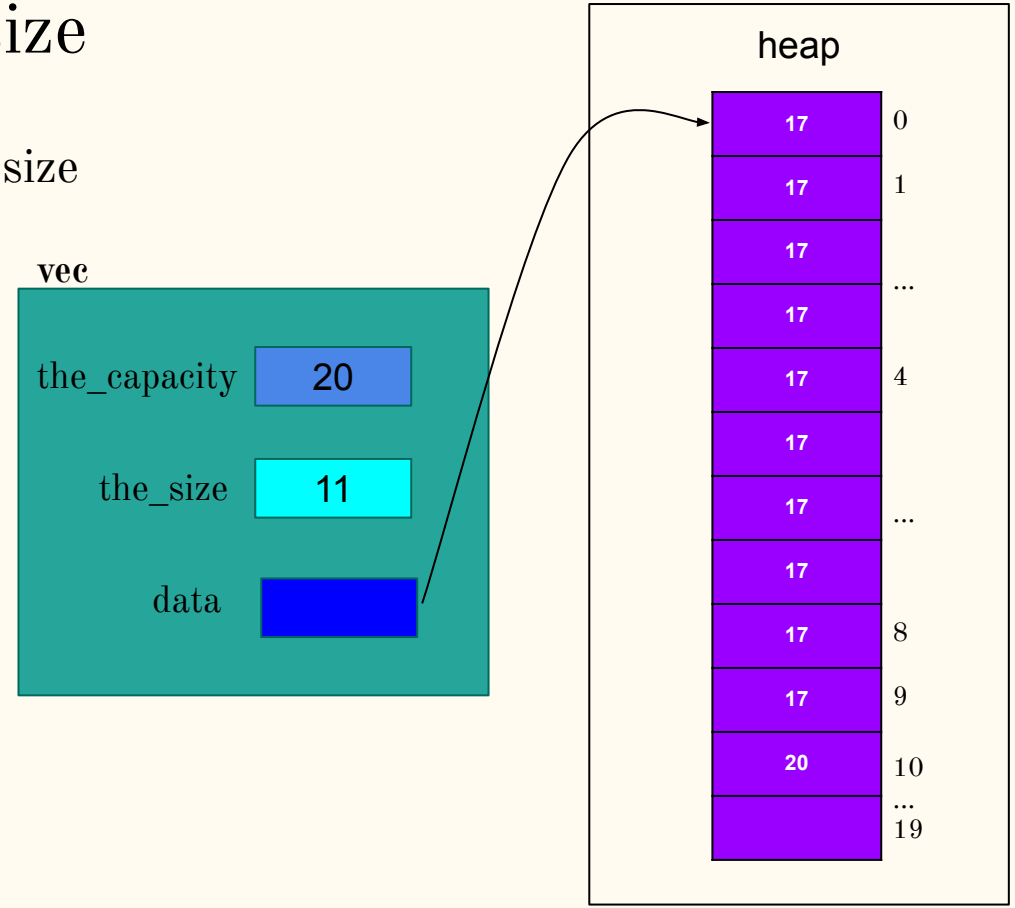
Agenda

- Vector class methods
- Supporting ranged `for`
  - Pointer arithmetic

Vector class methods

Reducing the Vector's size

- two methods reduce a vector's size
  - pop_back()
  - clear()
- dynamic array unchanged

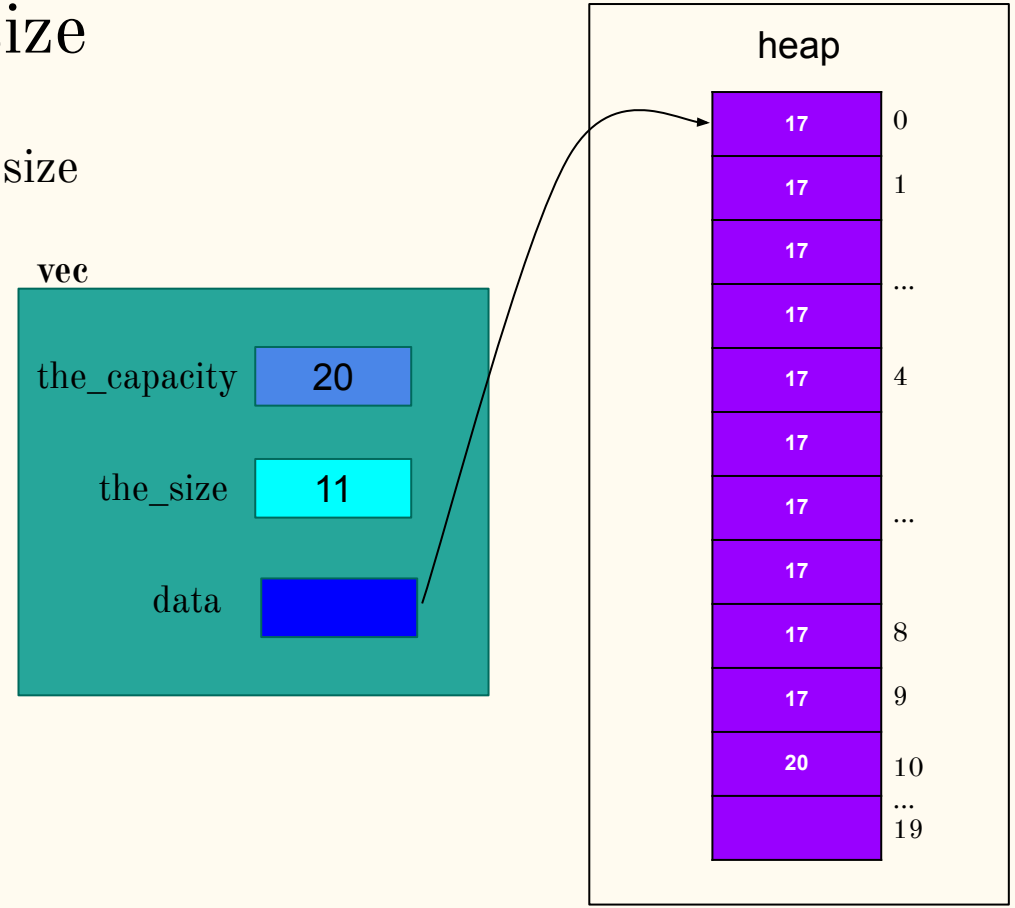


Reducing the Vector's size

- two methods reduce a vector's size
  - pop_back()
  - clear()
- dynamic array unchanged

```
vec.clear();
```

- heap memory remains allocated
- capacity value unchanged
- size updated (set to 0)

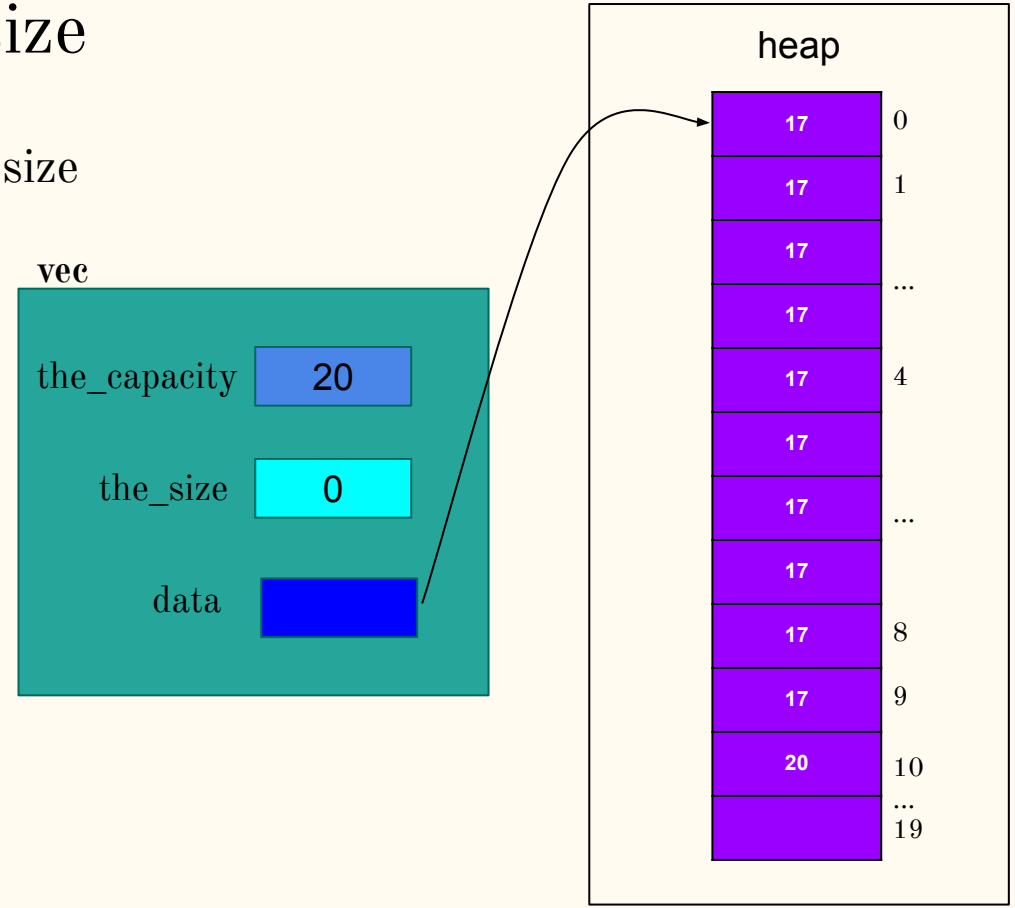


Reducing the Vector's size

- two methods reduce a vector's size
  - pop_back()
  - clear()
- dynamic array unchanged

```
vec.clear();
```

- heap memory remains allocated
- capacity value unchanged
- size updated (set to 0)

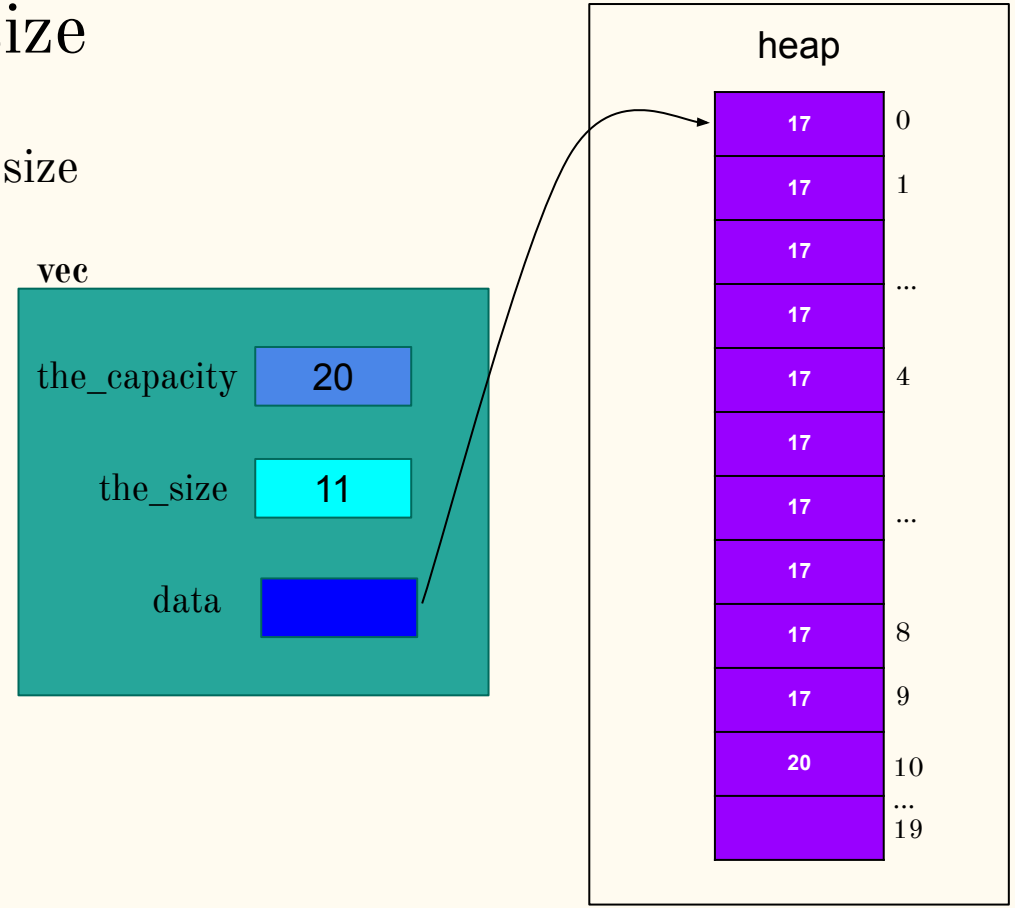


Reducing the Vector's size

- two methods reduce a vector's size
  - pop_back()
  - clear()
- dynamic array unchanged

```
vec.pop_back();
```

- heap memory unchanged
- capacity value unchanged
- size updated (decreased by 1)

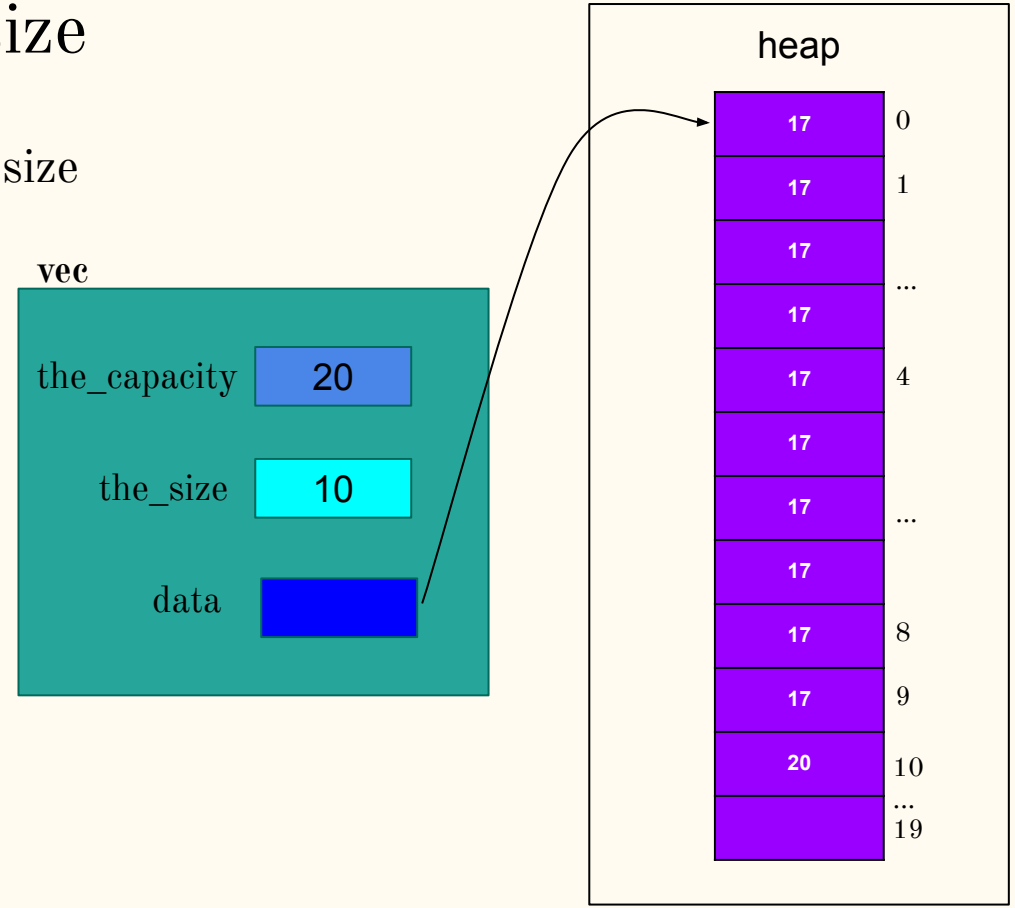


Reducing the Vector's size

- two methods reduce a vector's size
  - pop_back()
  - clear()
- dynamic array unchanged

```
vec.pop_back();
```

- heap memory unchanged
- capacity value unchanged
- size updated (decreased by 1)



Reducing the Vector's size

`vec.clear();`

- heap memory remains allocated
- capacity value unchanged
- size updated (set to 0)

`vec.pop_back();`

- heap memory unchanged
- capacity value unchanged
- size updated (decreased by 1)

```
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;
};
```

Reducing the Vector's size

`vec.clear();`

- heap memory remains allocated
- capacity value unchanged
- size updated (set to 0)

`vec.pop_back();`

- heap memory unchanged
- capacity value unchanged
- size updated (decreased by 1)

```
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]; }

    // implement clear()

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

Reducing the Vector's size

`vec.clear();`

- heap memory remains allocated
- capacity value unchanged
- size updated (set to 0)

`vec.pop_back();`

- heap memory unchanged
- capacity value unchanged
- size updated (decreased by 1)

```
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]; }

    // implement clear()

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

Reducing the Vector's size

`vec.clear();`

- heap memory remains allocated
- capacity value unchanged
- size updated (set to 0)

`vec.pop_back();`

- heap memory unchanged
- capacity value unchanged
- size updated (decreased by 1)

```
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]; }

    void clear() { the_size = 0; }

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

Reducing the Vector's size

`vec.clear();`

- heap memory remains allocated
- capacity value unchanged
- size updated (set to 0)

`vec.pop_back();`

- heap memory unchanged
- capacity value unchanged
- size updated (decreased by 1)

```
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]; }

    void clear() { the_size = 0; }

    // implement pop_back()

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

Reducing the Vector's size

`vec.clear();`

- heap memory remains allocated
- capacity value unchanged
- size updated (set to 0)

`vec.pop_back();`

- heap memory unchanged
- capacity value unchanged
- size updated (decreased by 1)

```
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]; }

    void clear() { the_size = 0; }

    void pop_back() { --the_size; }

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

Supporting ranged `for`

A typical `for` loop

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

int main() {

    vector<int> vec(5, -1);

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

}
```

```
% g++ -std=c++11 test.cpp -o test.o
% ./test.o
-1
-1
-1
-1
-1
```

A typical `for` loop

begin end

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

A ranged `for` loop

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

int main() {

    vector<int> vec(5, -1);

     for (int elem: vec) {
         cout << elem << endl;
     }

}
```

```
% g++ -std=c++11 test.cpp -o test.o
% ./test.o
-1
-1
-1
-1
-1
```

A ranged `for` loop

begin? *end?*

```
for (int elem: vec) {
    cout << elem << endl;
}
```

A typical `for` loop (using CS 2124 Vector)

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

int main() {
    Vector vec(5, -1);
     for (size_t i = 0; i < vec.size(); i++) {
         cout << vec[i] << endl;
     }
}
```

implemented size()

implemented operator[]

```
% g++ -std=c++11 test.cpp -o test.o
% ./test.o
-1
-1
-1
-1
-1
```

A ranged `for` loop

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


int main() {

    Vector vec(5, -1);

     for (int elem: vec) {
         cout << elem << endl;
     }

}
```

compilation error

to use ranged `for` compiler requires:

- `begin()` member function
- `end()` member function

```
error: invalid range expression of type 'Vector'; no viable 'begin' function available
 for (int elem: vec1) {
            ^ ~~~~
1 error generated.
```

`begin()` and `end()` methods

```
int main() {
    Vector vec(5, -1);

        compilation error

    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    ___ begin() { ___ }
    ___ end() { ___}


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

Both `begin()` and `end()` return pointers

`begin()` and `end()` methods

```
int main() {
    Vector vec(5, -1);

        compilation error

    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    _2_ begin() { ___ }
    _2_ end() { ___}


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

Both `begin()` and `end()` return pointers

TurningPoint

SRS Setup
Login: student.turningtechnologies.com
Session ID: 20220302<A|D>

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

Which pointer type replaces blanks #2 for the return types of `begin()` and `end()`?

```
int main() {
    Vector vec(5, -1);

        compilation error

    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    _2_ begin() { ___ }
    _2_ end() { ___}


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

Both `begin()` and `end()` return pointers

`begin()` and `end()` methods

```
int main() {
    Vector vec(5, -1);

        compilation error

    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { ___ }
    int* end() { ___}


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

Both `begin()` and `end()` return pointers

`begin()` and `end()` methods

```
int main() {
    Vector vec(5, -1);

        compilation error

    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return _3_; }
    int* end() { ___}


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

Both `begin()` and `end()` return pointers

Which memory address represents the start of the array of integers stored for the current `Vector` object (replacing blank #3)?

```
int main() {
    Vector vec(5, -1);

        compilation error

    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return _3_; }
    int* end() { ___}


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

Both `begin()` and `end()` return pointers

`begin()` and `end()` methods

```
int main() {
    Vector vec(5, -1);

        compilation error

    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() { ___}


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

Both `begin()` and `end()` return pointers

`begin()` and `end()` methods

```
int main() {
    Vector vec(5, -1);

    for (int elem: vec) {
        cout << elem << endl;
    }

}
```

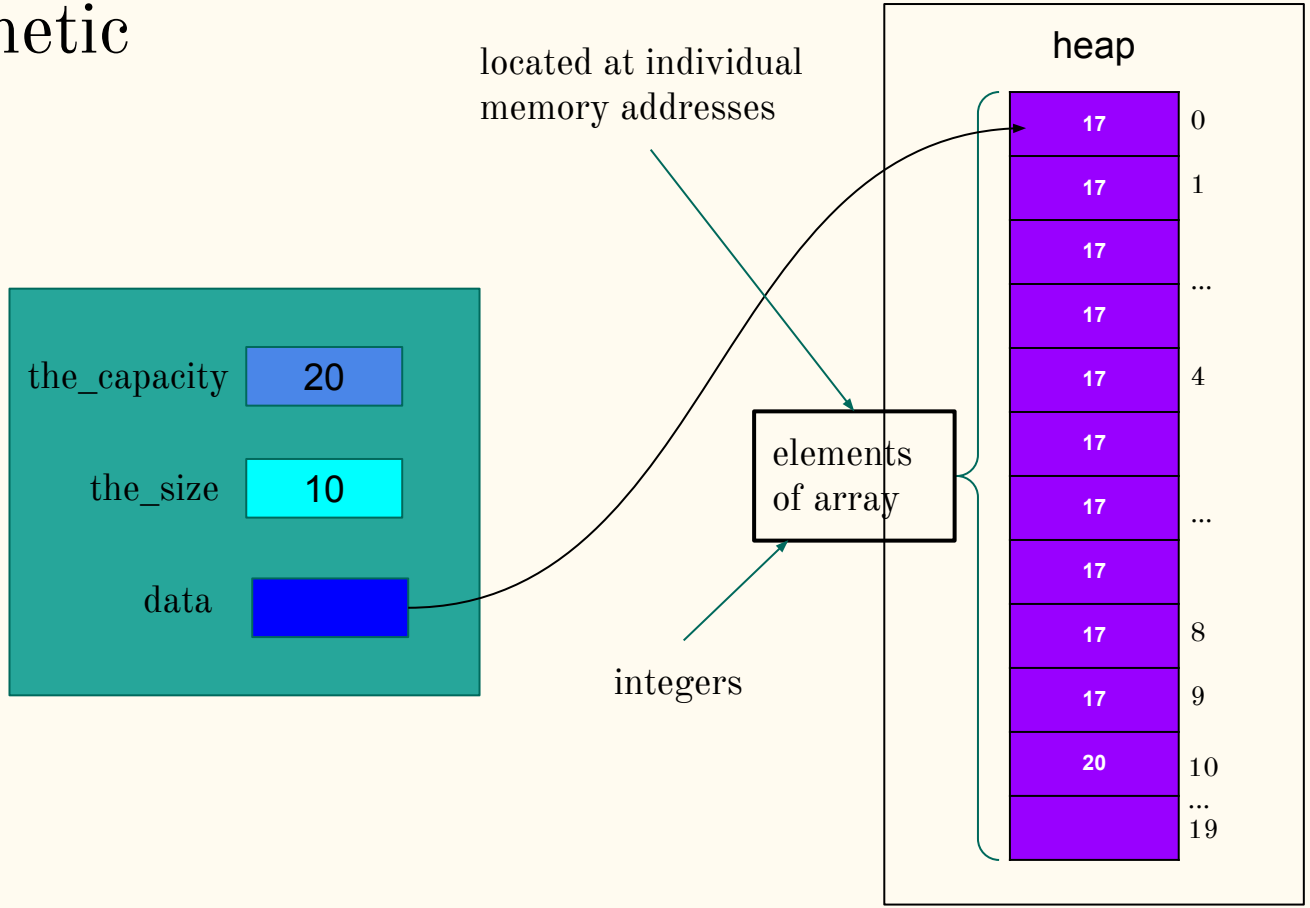
compilation error

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

    size_t size() const { return the_size; }
    int operator[](size_t i) const { return data[i]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() { ___}

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

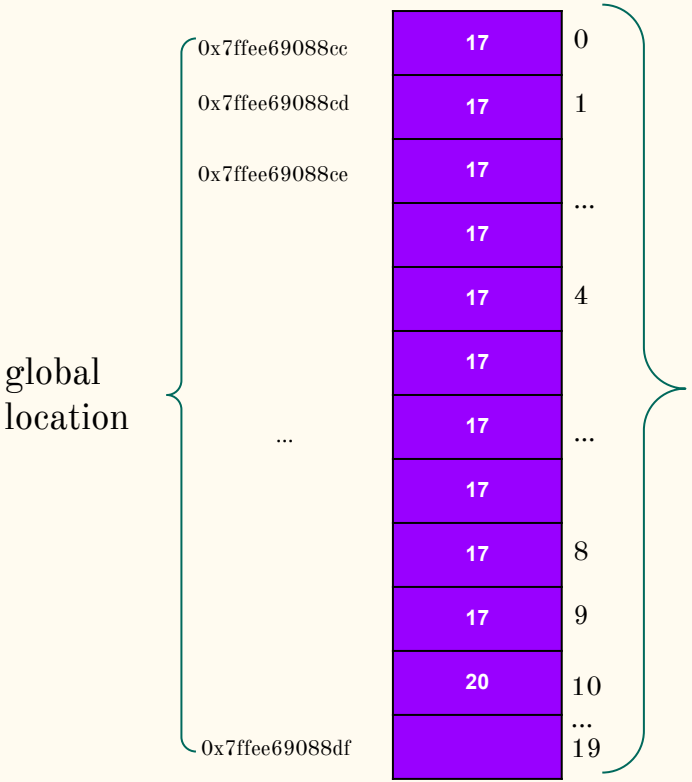
How to represent the end of the vector?

Both `begin()` and `end()` return pointers

Pointer arithmetic

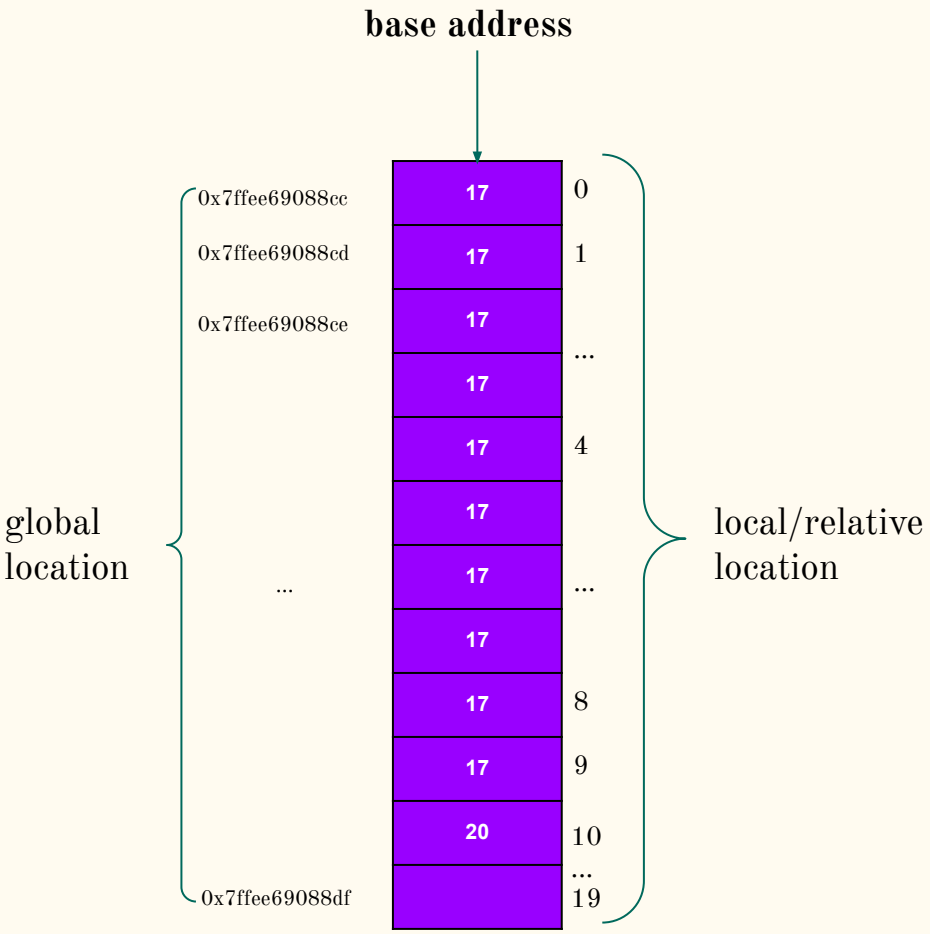


Pointer arithmetic



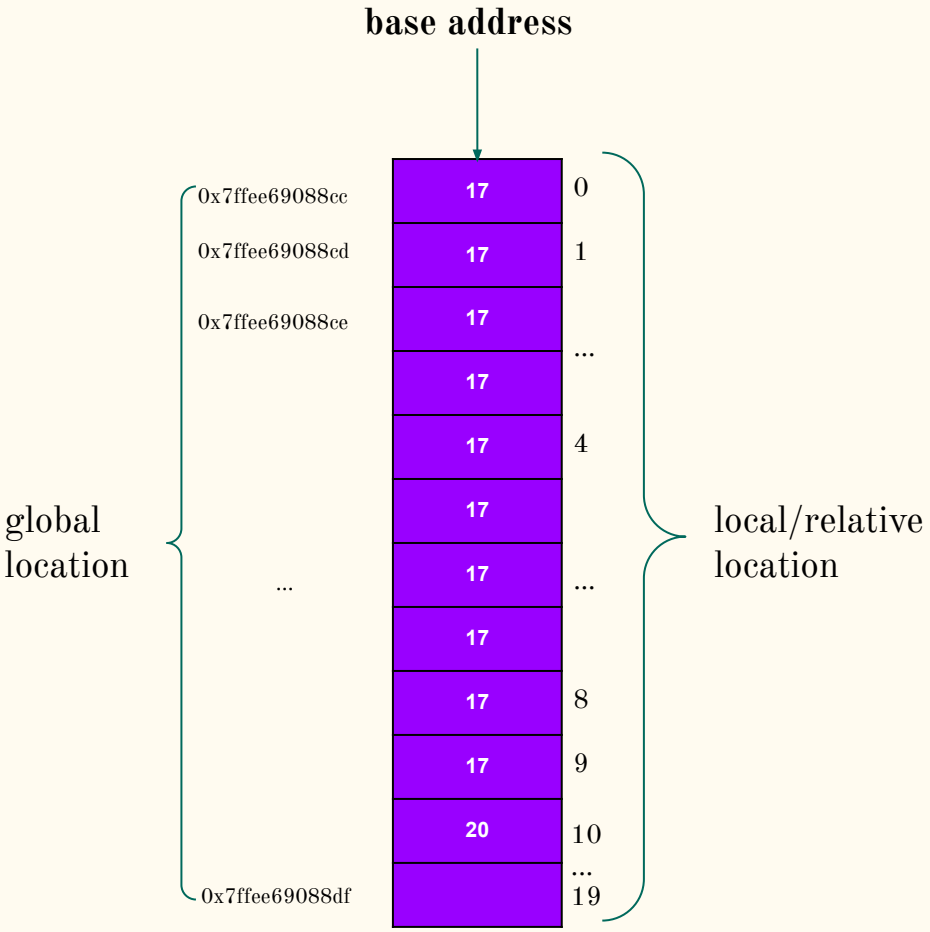
local/relative to what?

Pointer arithmetic

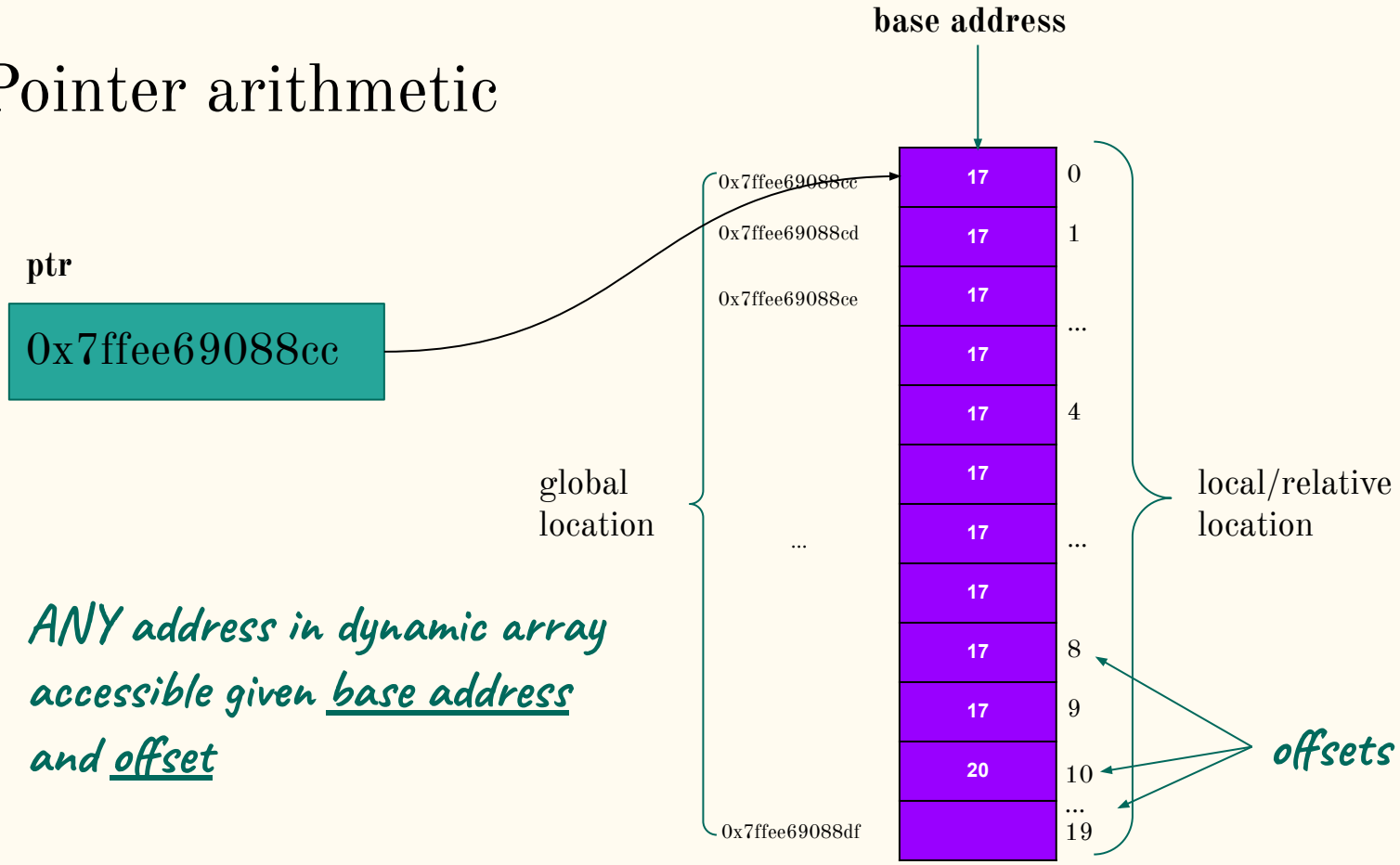


Pointer arithmetic

ptr

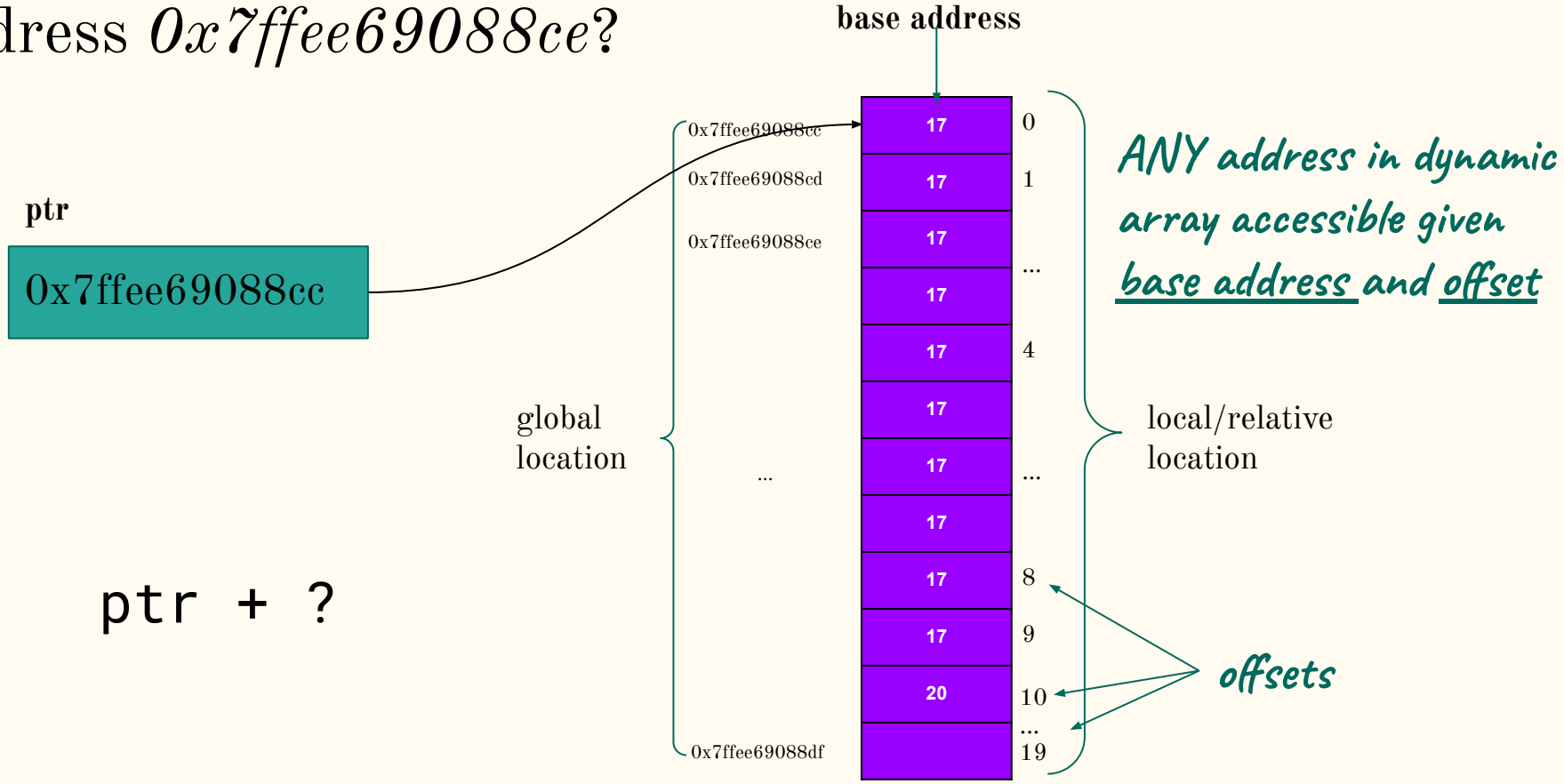


Pointer arithmetic



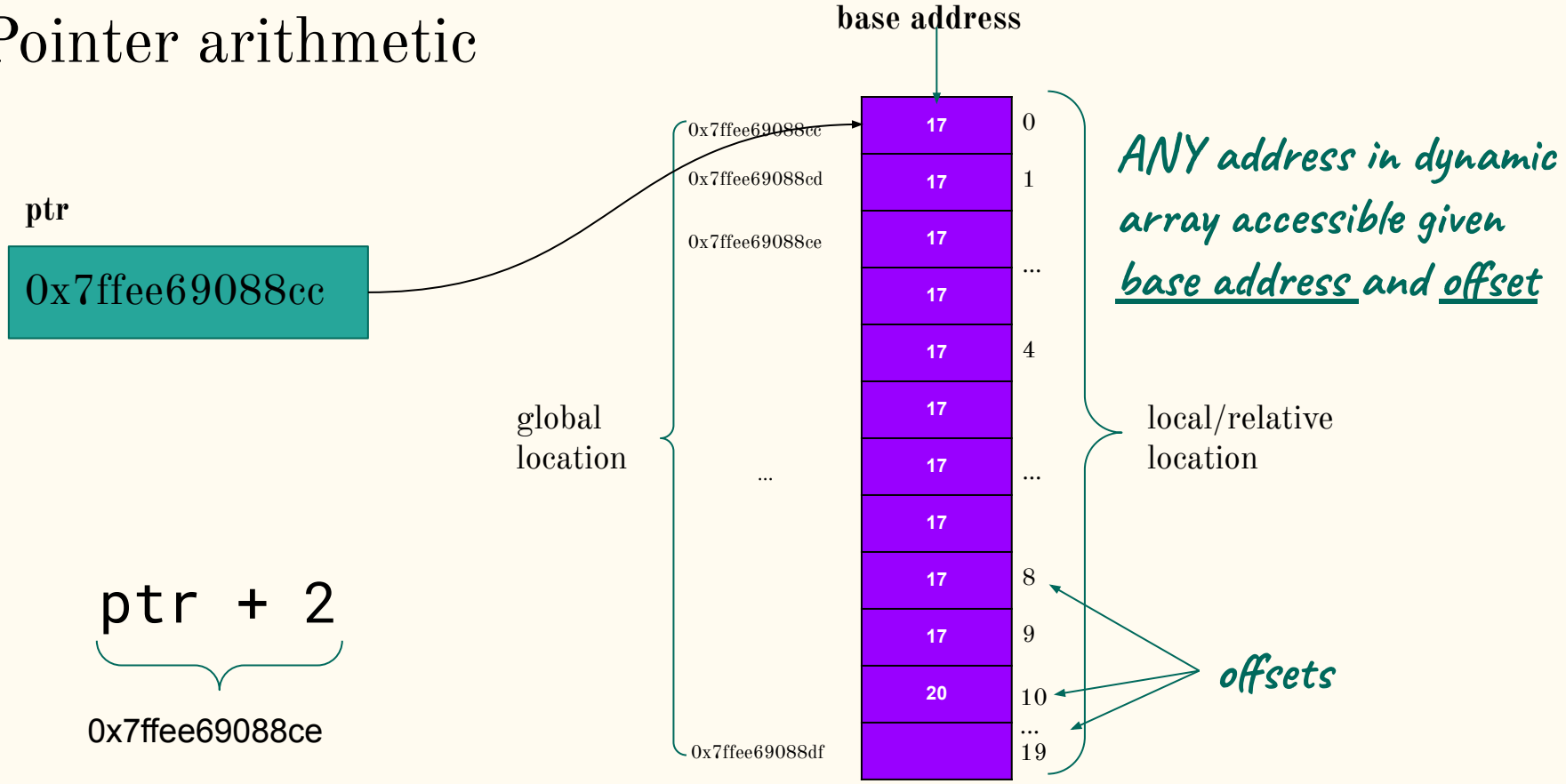
ANY address in dynamic array accessible given <u>base address</u> and <u>offset</u>

Which offset replaces ? so that the expression evaluates to address *0x7ffee69088ce*?

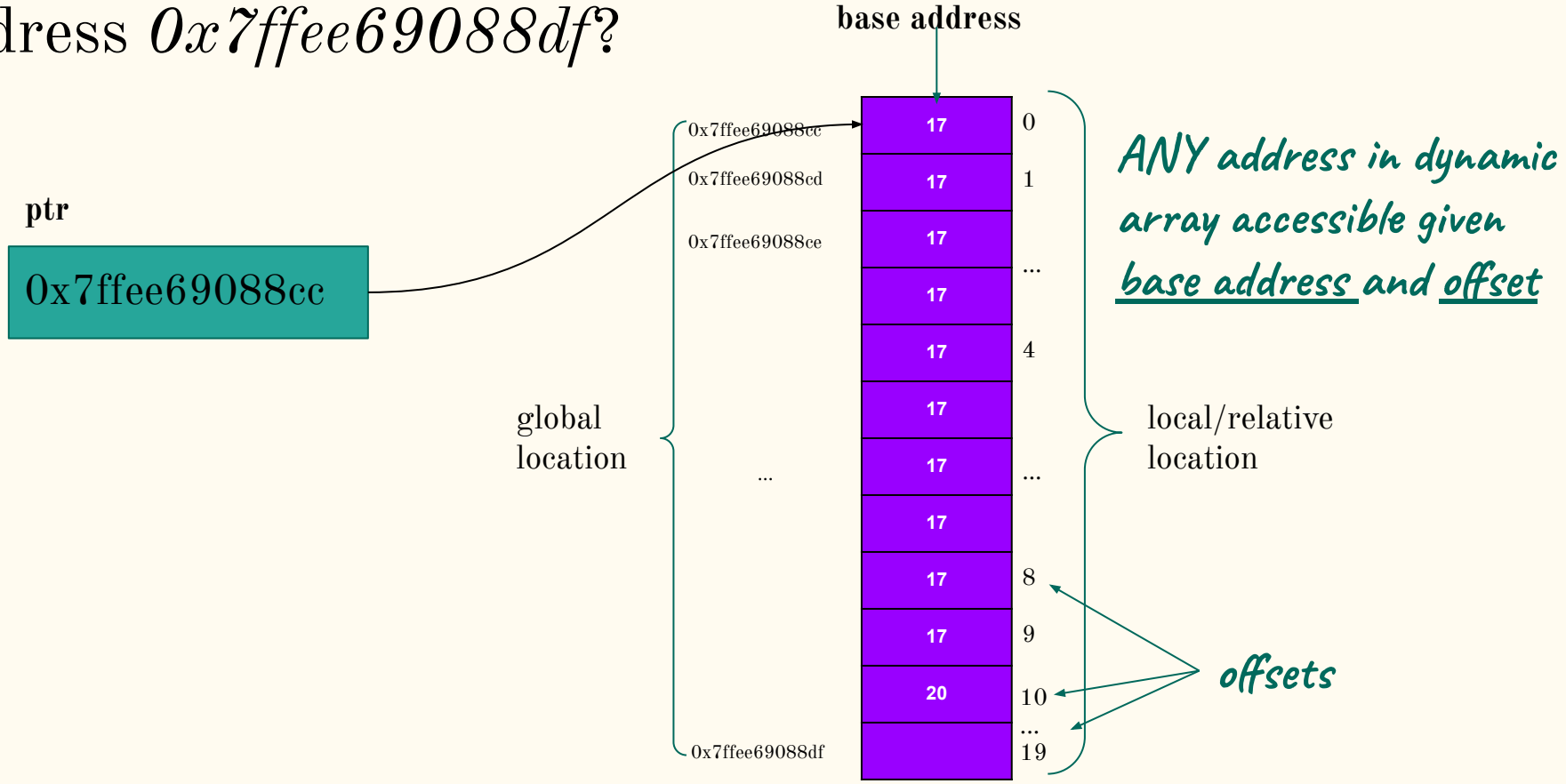



```
ptr + ?
```

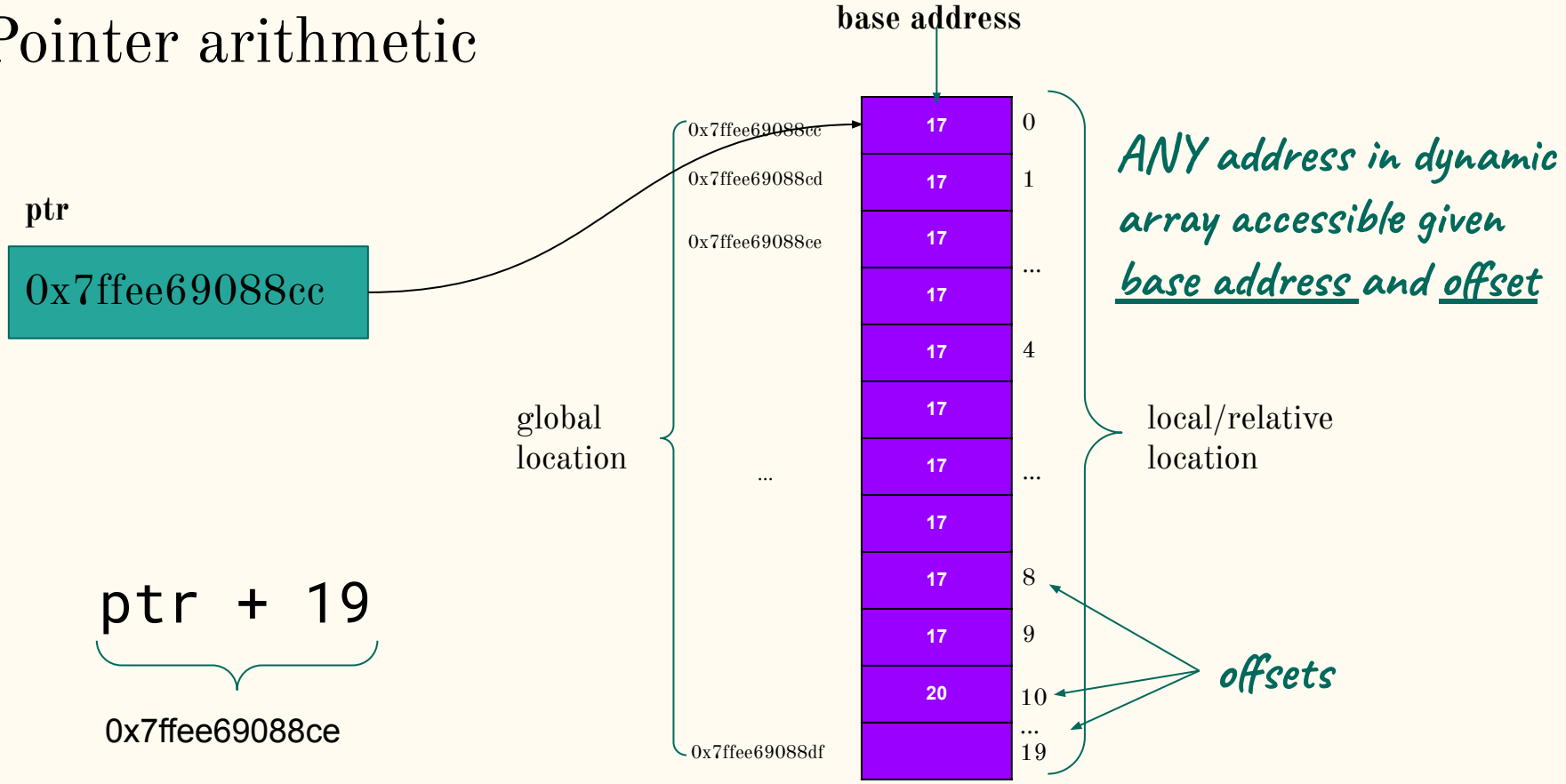
Pointer arithmetic



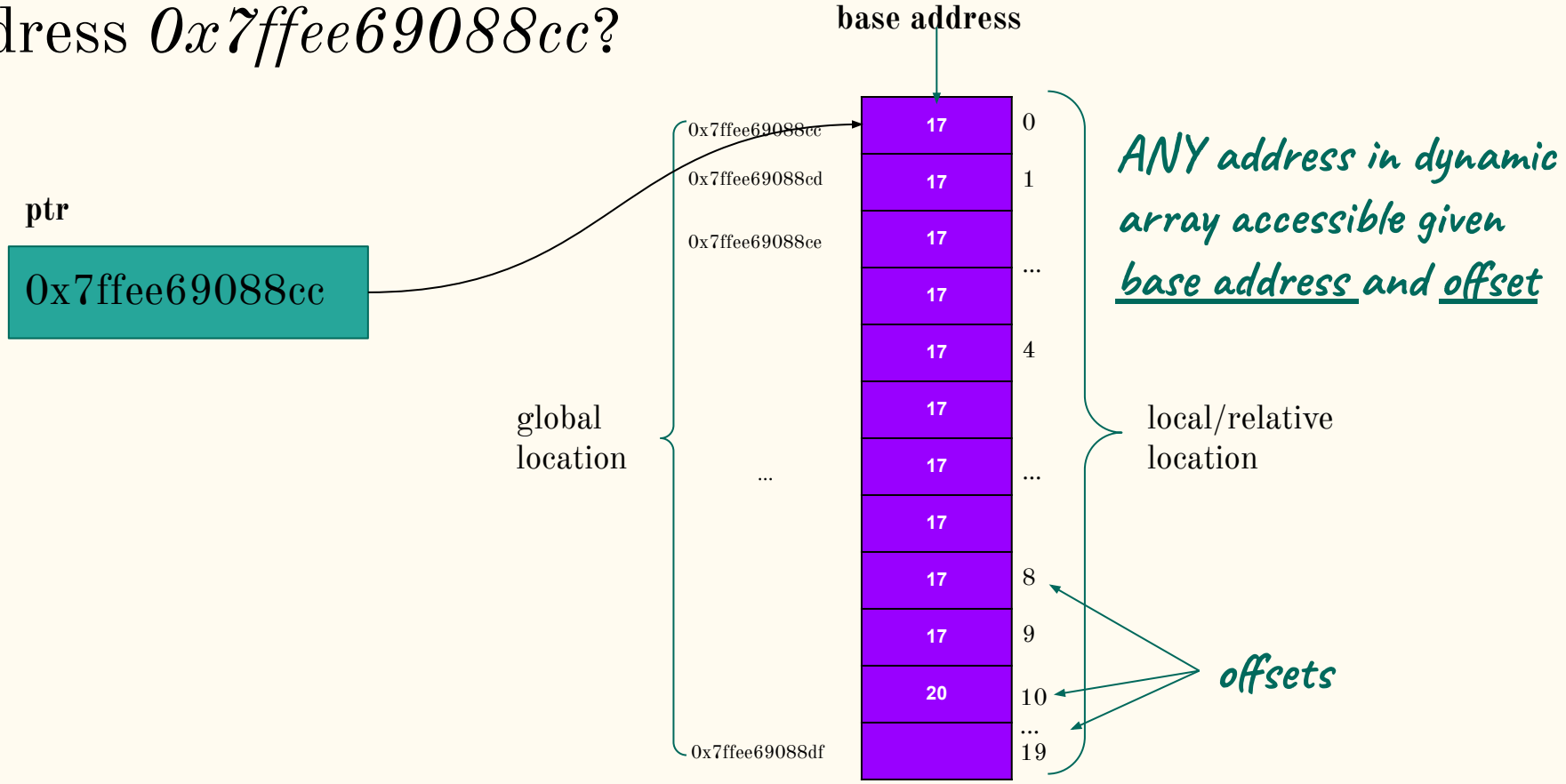
Which expression using `ptr` and an offset evaluates to address *0x7ffee69088df*?



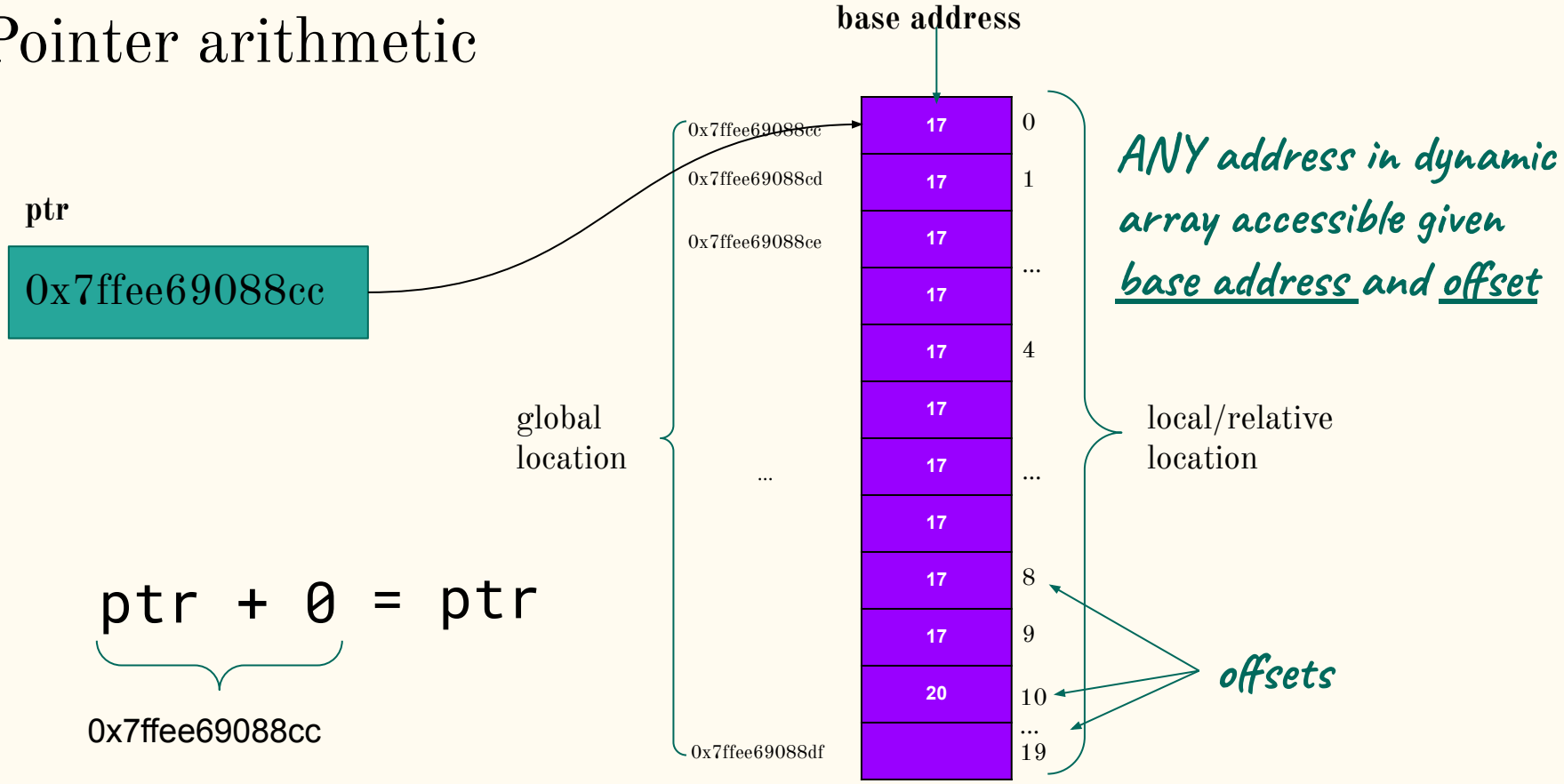
Pointer arithmetic



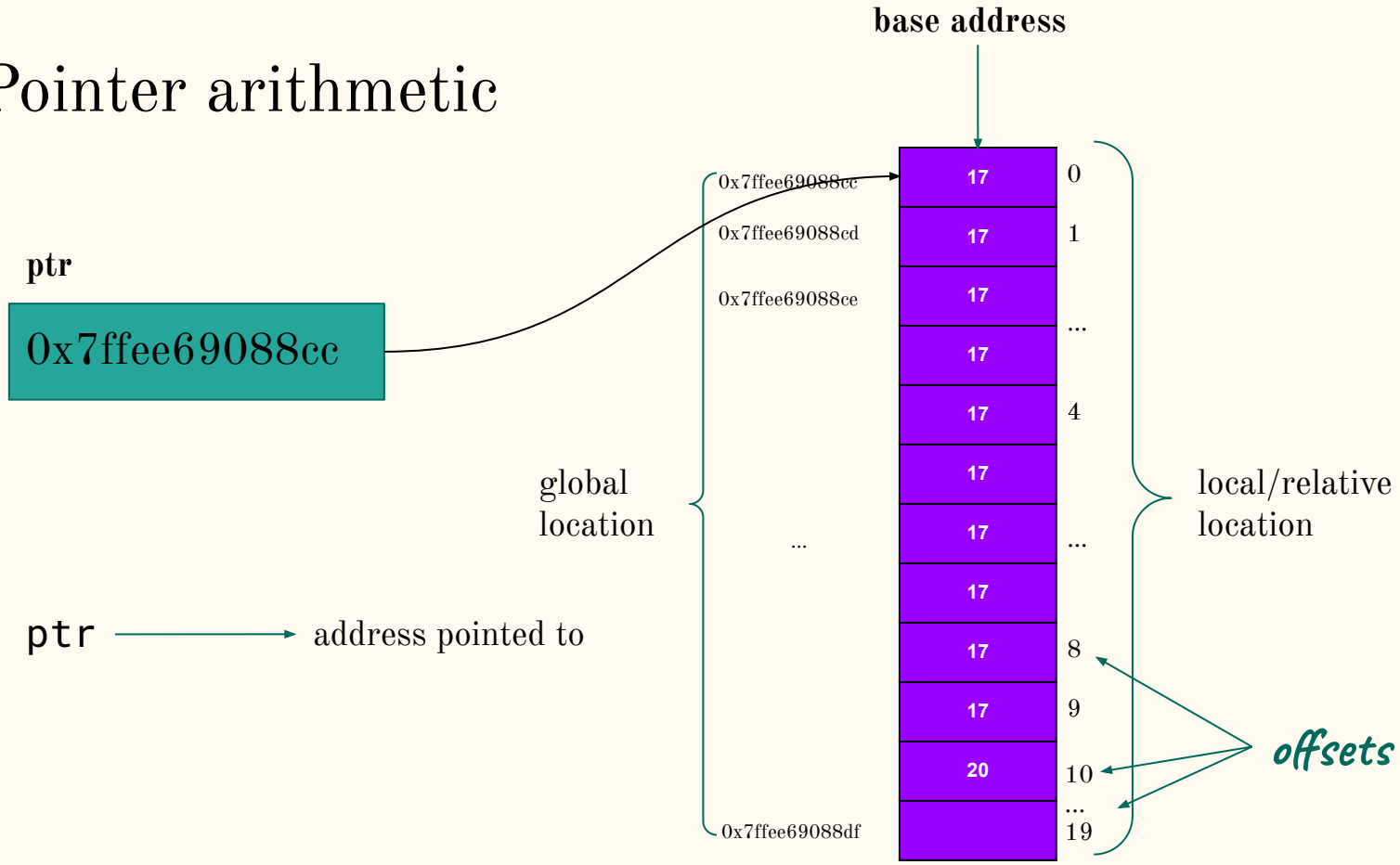
Which expression using `ptr` and an offset evaluates to address *0x7ffee69088cc*?



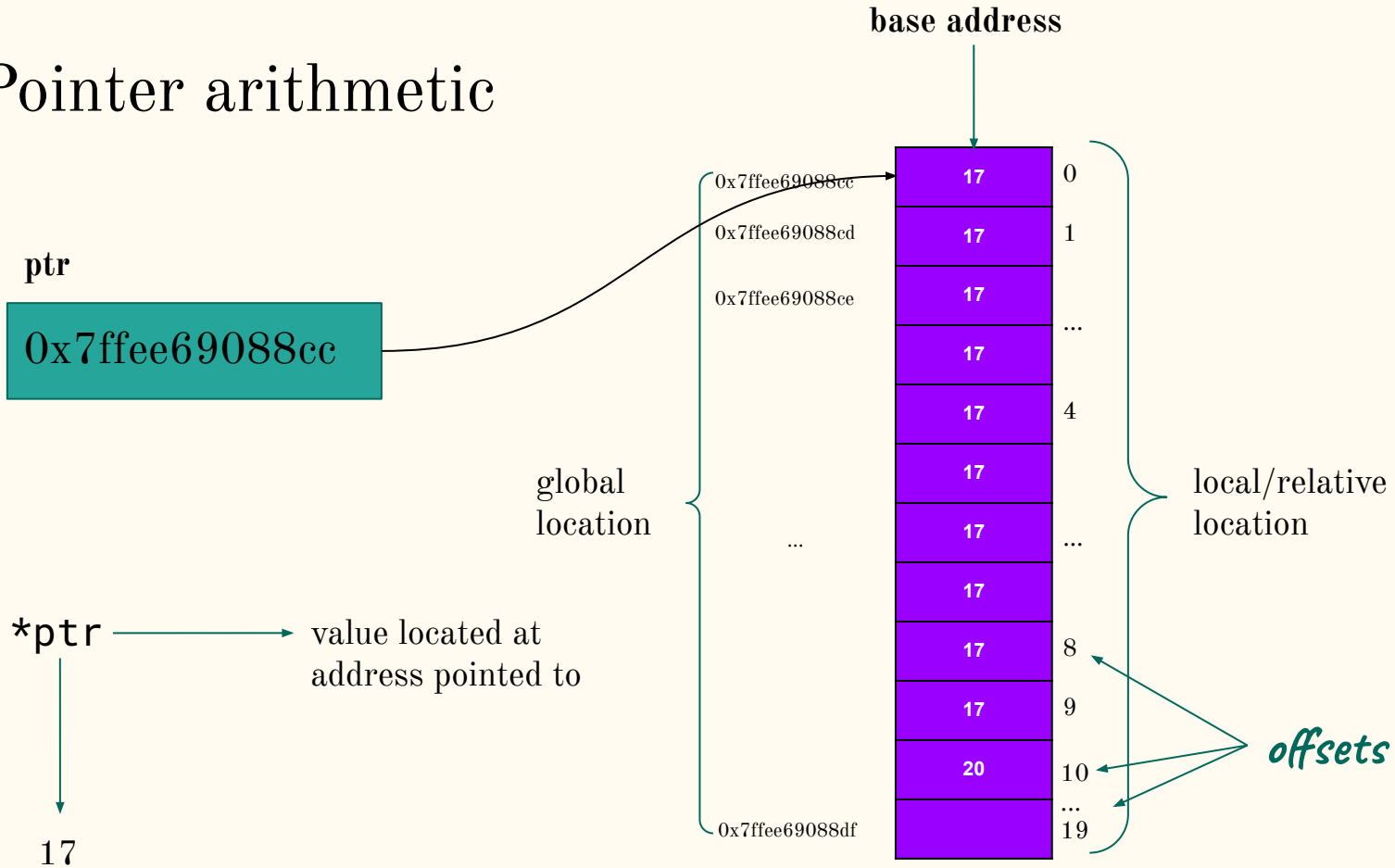
Pointer arithmetic



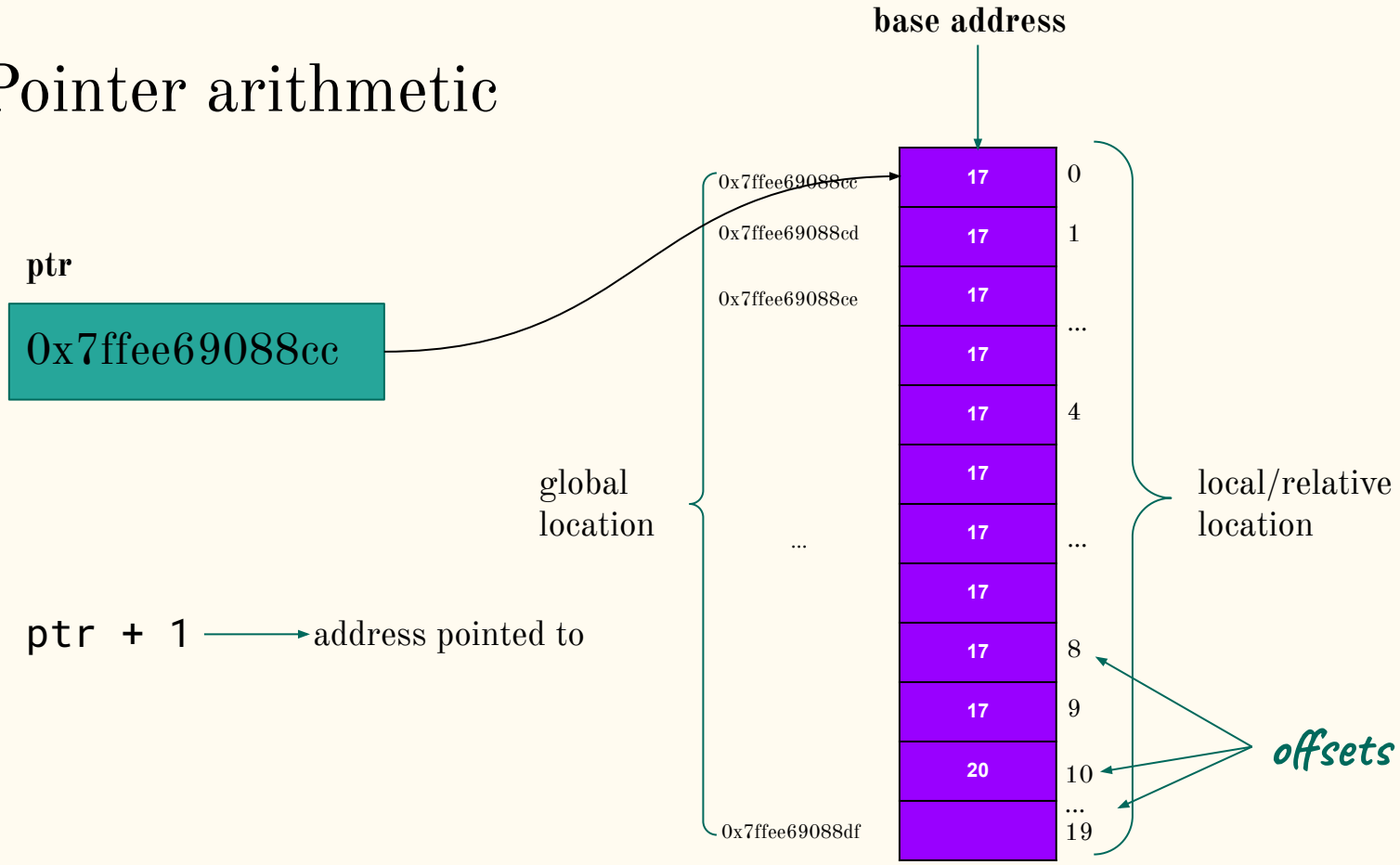
Pointer arithmetic



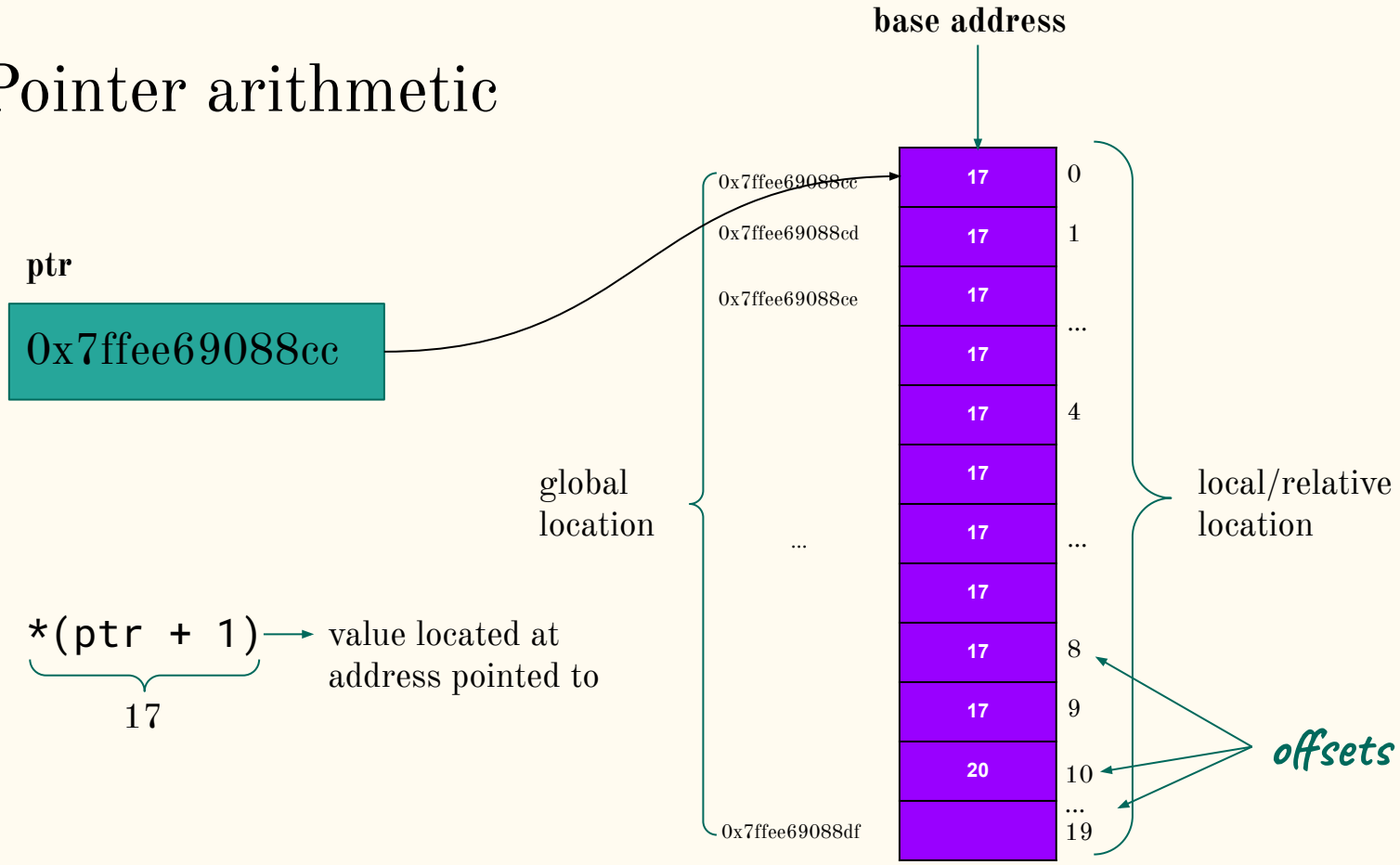
Pointer arithmetic



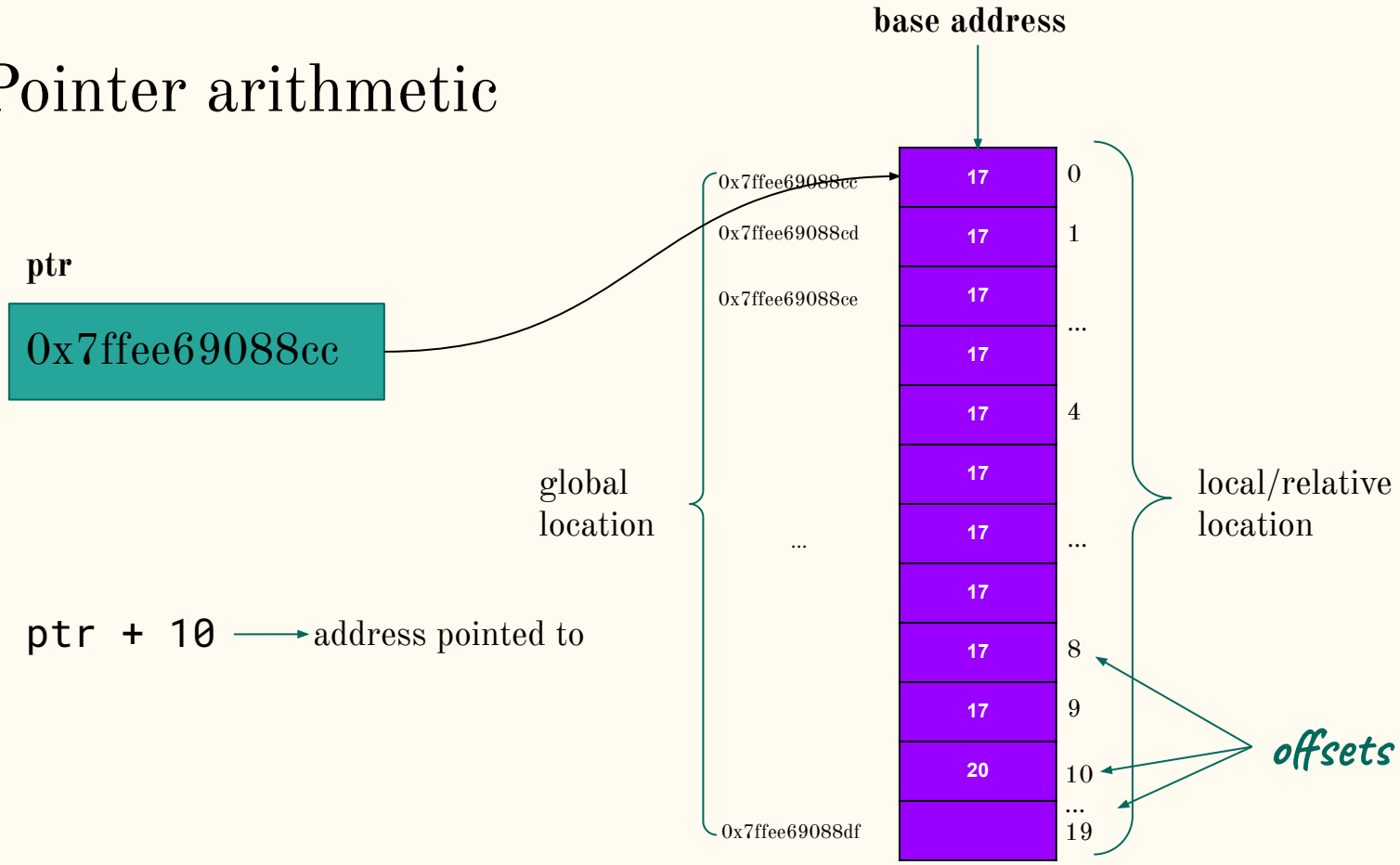
Pointer arithmetic



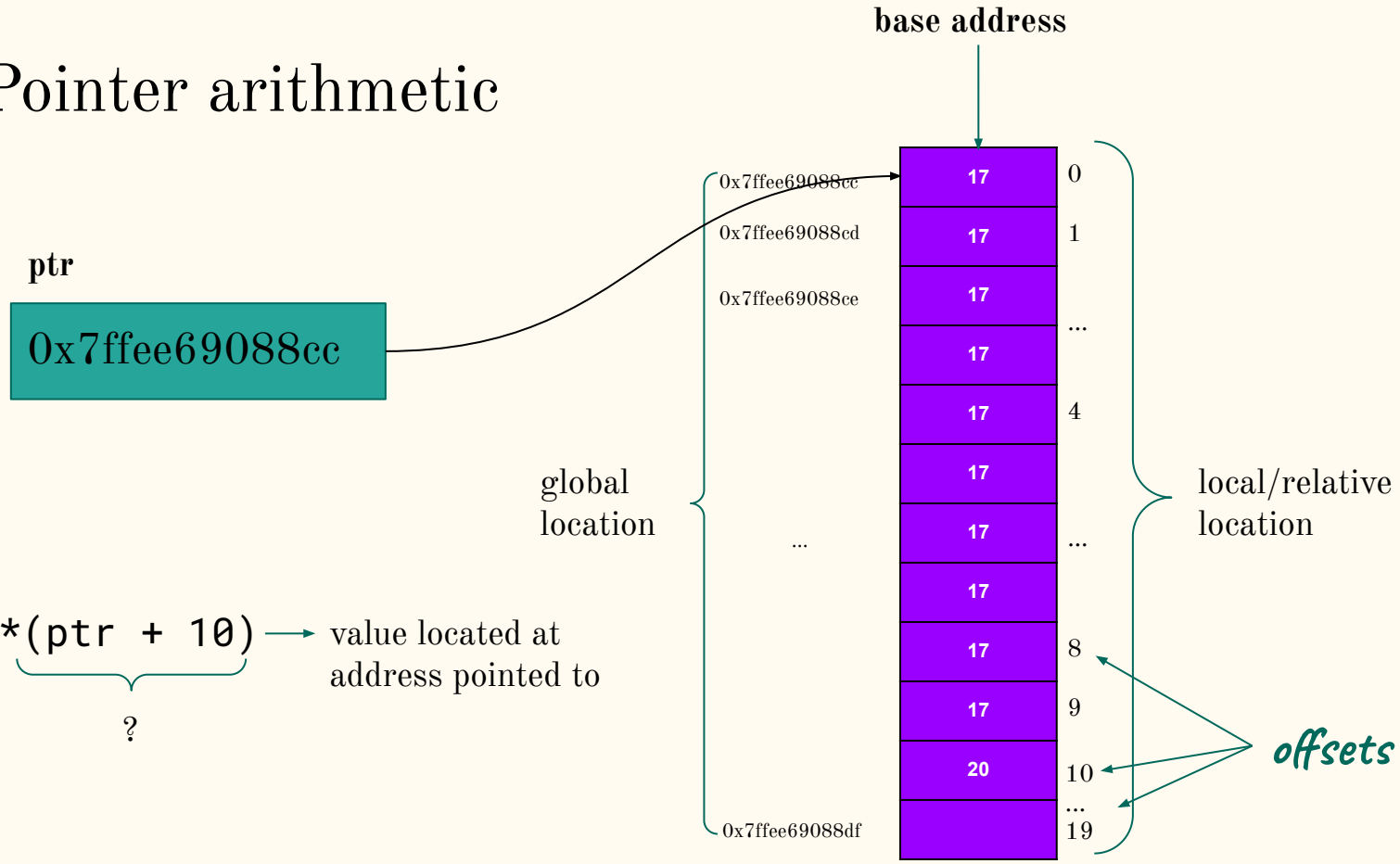
Pointer arithmetic



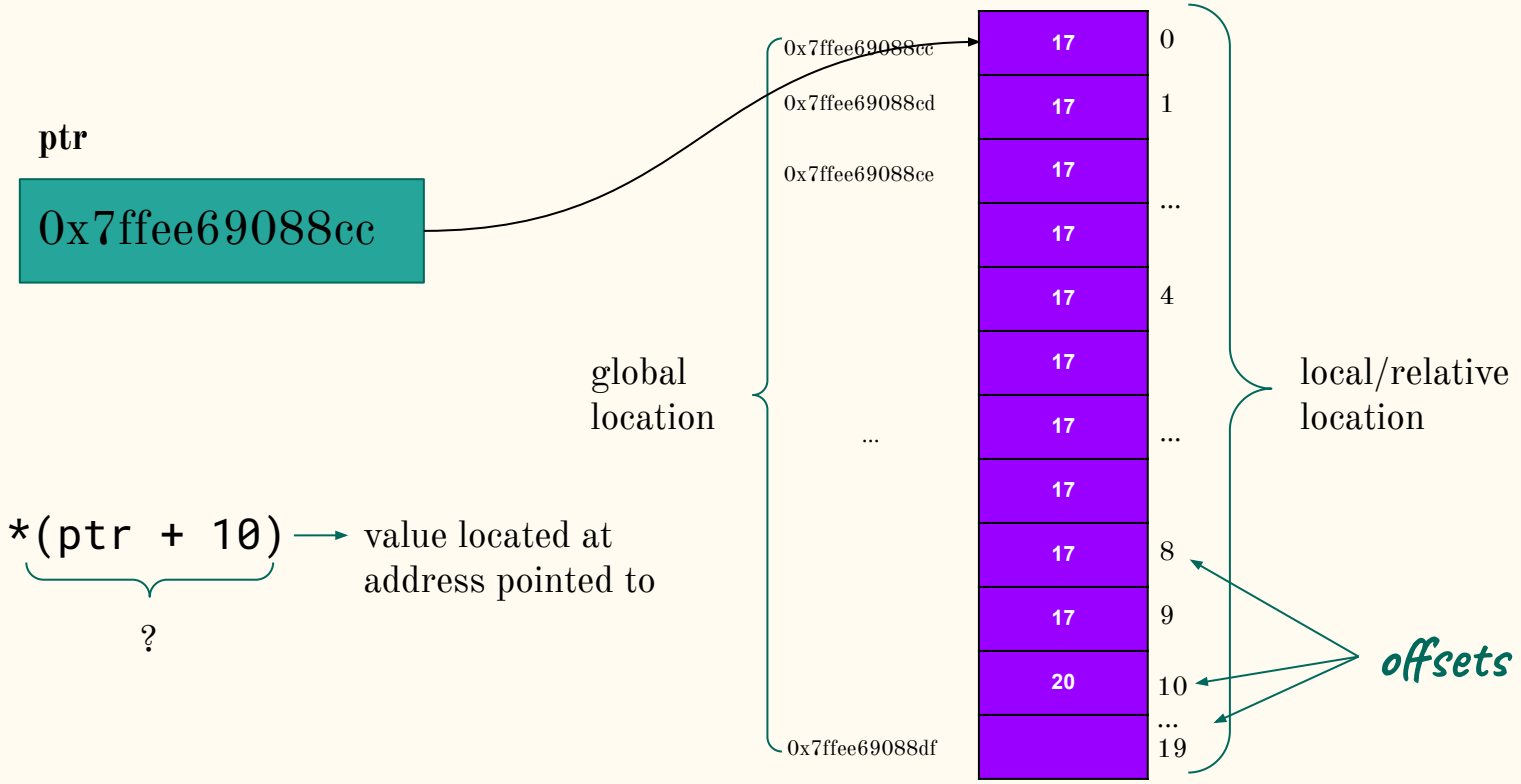
Pointer arithmetic



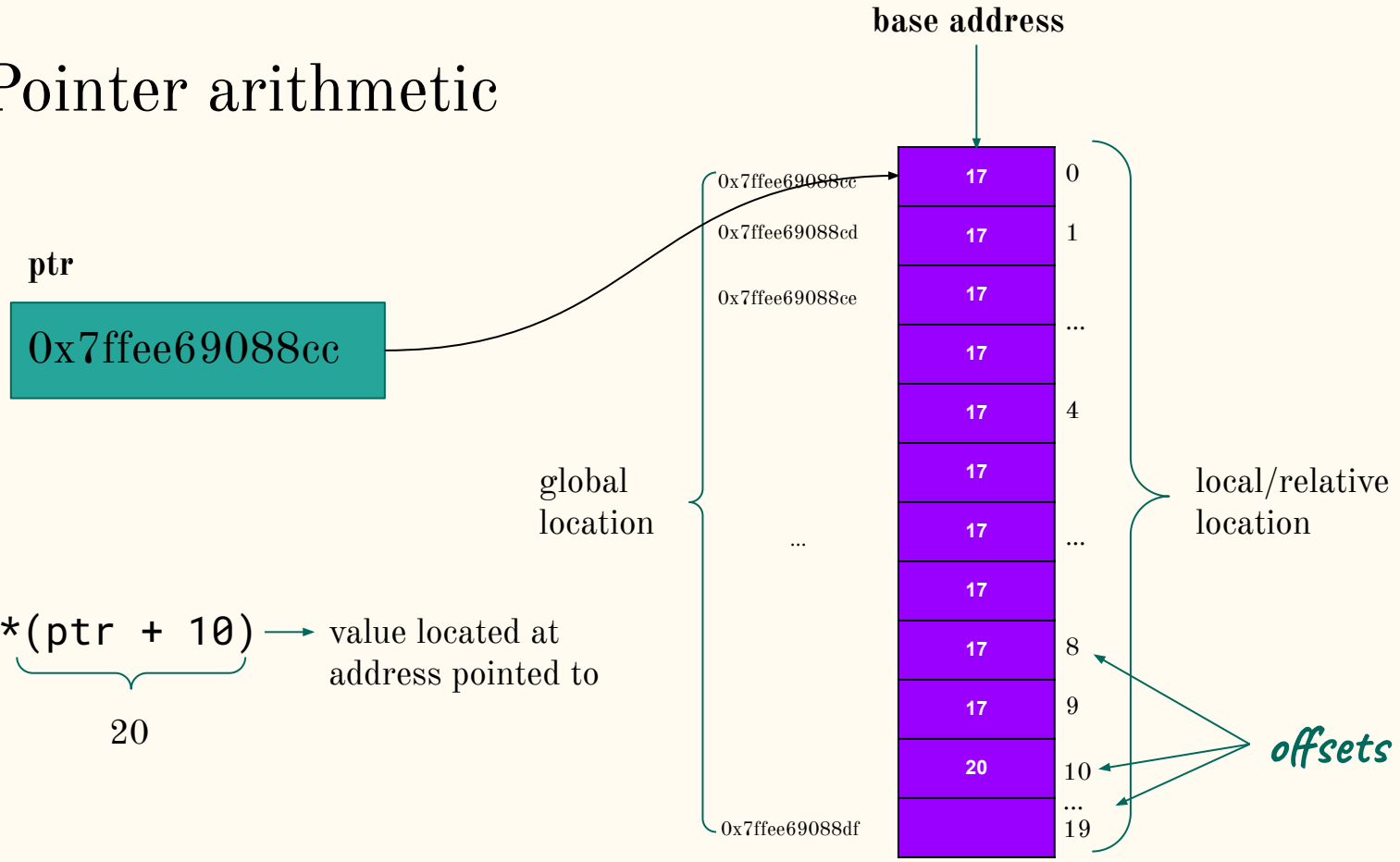
Pointer arithmetic



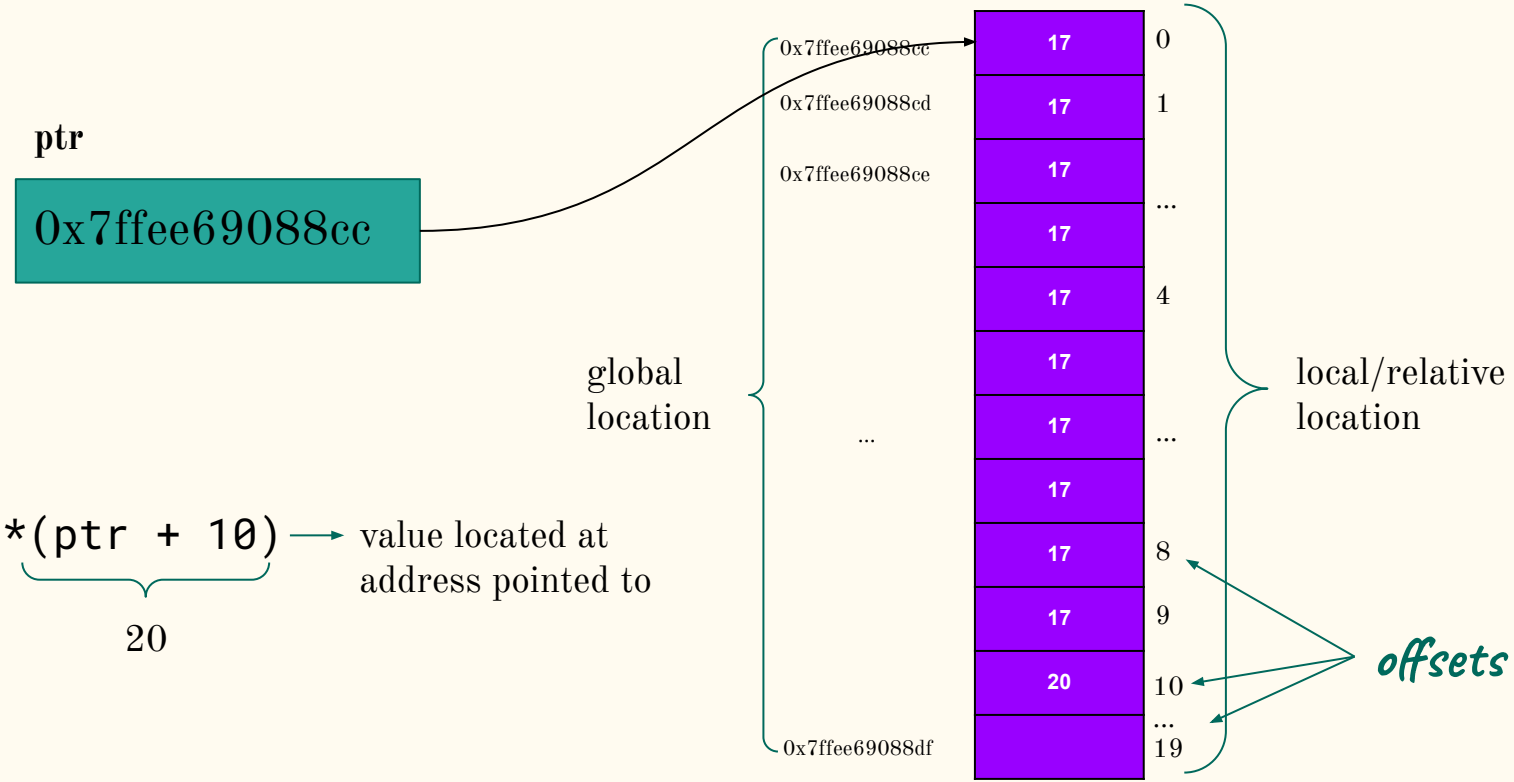
Evaluate `*(ptr + 10)`:



Pointer arithmetic



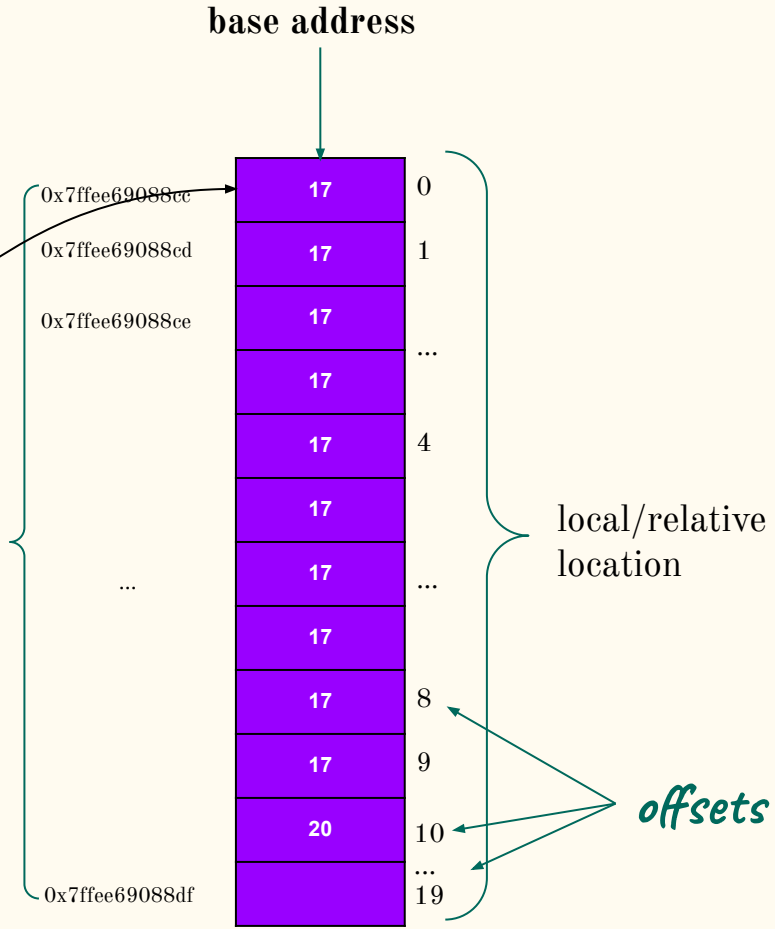
Which other expression using `ptr` and an offset evaluates to 20?



Pointer arithmetic

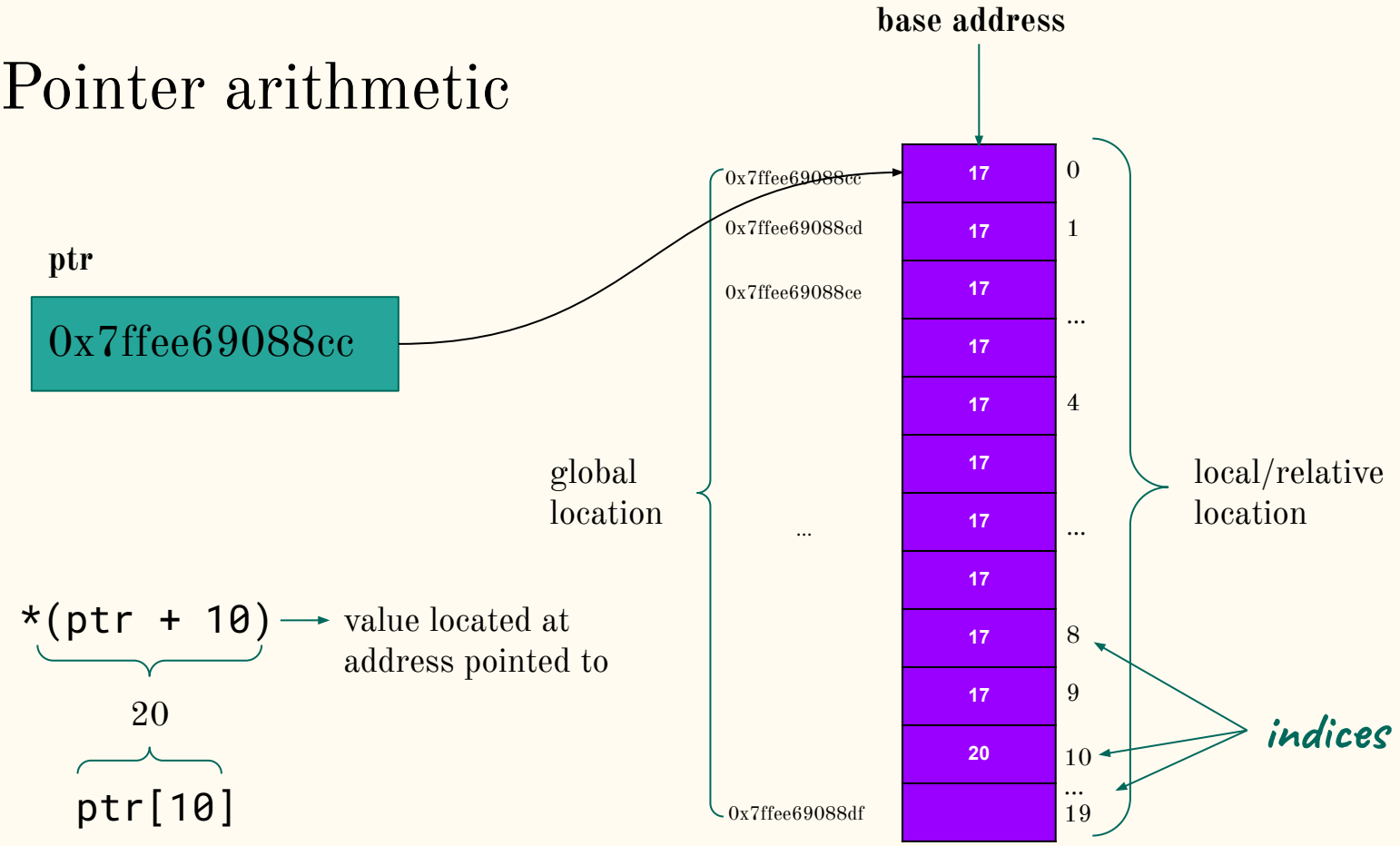
ptr

0x7ffee69088cc

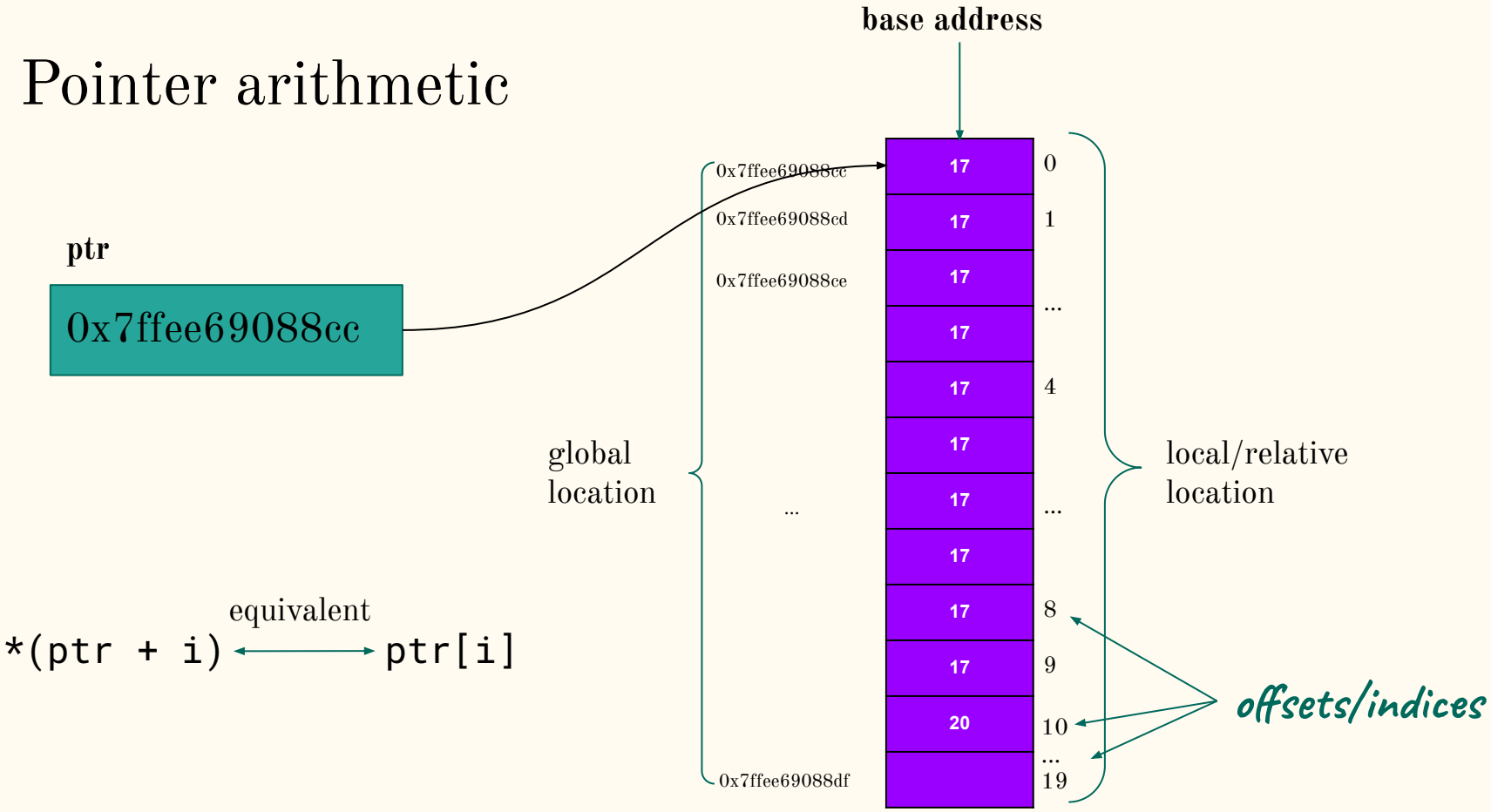


$\underbrace{\underbrace{\texttt{*(ptr + 10)}}_{20}}_{\texttt{ptr[10]}}$ → value located at address pointed to

Pointer arithmetic



Pointer arithmetic



`begin()` and `end()` methods

```
int main() {
    Vector vec(5, -1);

    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() { ___}

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

How to represent the end of the array?

Both `begin()` and `end()` return pointers

`begin()` and `end()` methods

```
int main() {
    Vector vec(5, -1);

        compilation error

    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() { // return address following last vector item }


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

Both `begin()` and `end()` return pointers

`begin()` and `end()` methods

```
int main() {
    Vector vec(5, -1);

    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() {
        // return address following last vector item
        ---
    }


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

Both `begin()` and `end()` return pointers

`begin()` and `end()` methods

```
int main() {
    Vector vec(5, -1);

        compilation error

    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() {
        // return address following last vector item
        return _4_;
    }


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

Both `begin()` and `end()` return pointers

Which expression when replacing blank #4 evaluates to the address directly following the last item in the vector?

```
int main() {
    Vector vec(5, -1);

        compilation error

    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() {
        // return address following last vector item
        return _4_;
    }


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

Both `begin()` and `end()` return pointers

`begin()` and `end()` methods

```
int main() {
    Vector vec(5, -1);

    for (int elem: vec) {
        cout << elem << endl;
    }

}
```

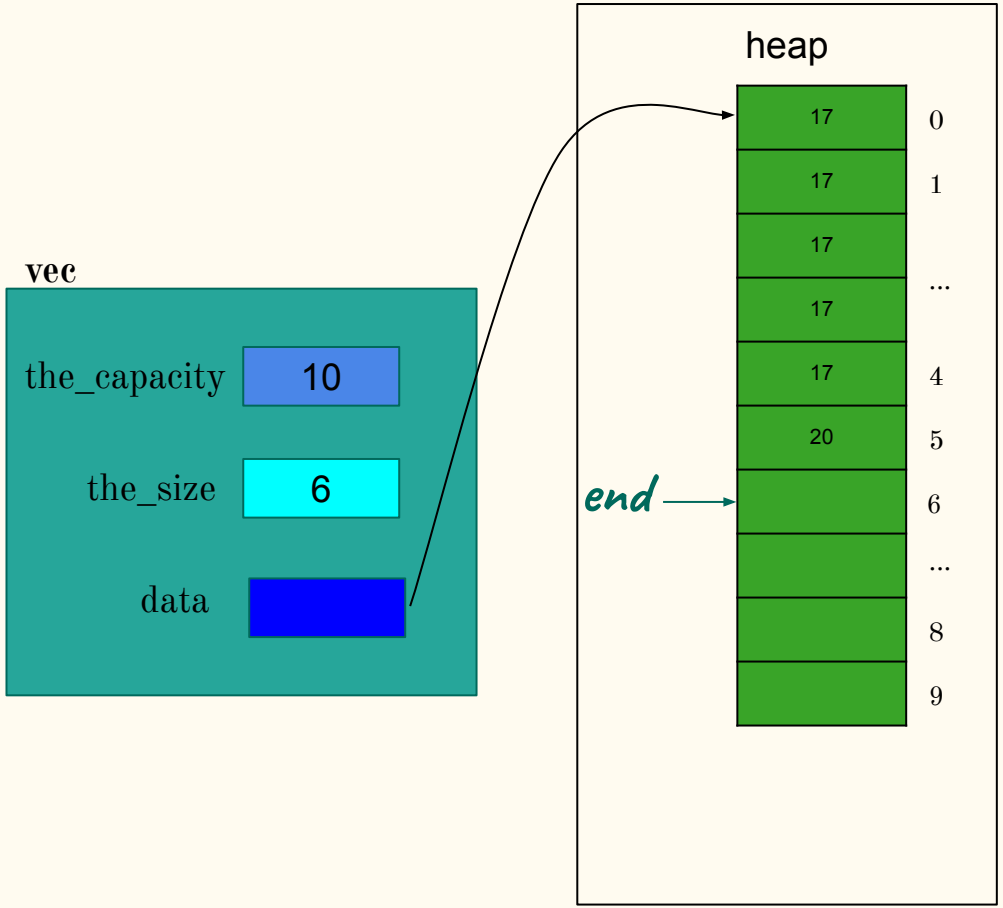
compilation error

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

    size_t size() const { return the_size; }
    int operator[](size_t i) const { return data[i]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() {
        // return address following last vector item
        return data + the_size;
    }


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

Both `begin()` and `end()` return pointers

Pointer arithmetic

$\underbrace{\texttt{data[the\_size - 1]}}_{\text{last } \texttt{int} \text{ in } \texttt{data}}$



`begin()` and `end()` methods

```
int main() {
    Vector vec(5, -1);

        compilation error

    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() { return data + the_size; }


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

`begin()` and `end()` methods

```
int main() {
    Vector vec(5, -1);


    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() { return data + the_size; }


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

`const begin()` and `end()` methods

```
void print_vec(const Vector& vec) {
```

compilation error

```
    for (int elem: vec) {
        cout << elem << endl;
    }
}
```

```
 void add_one(Vector& vec) {

     for (int& elem: vec) {
         elem += 1; ✔
     }
 }
```

```
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]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() { return data + the_size; }

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

non-const
(allow modification)

`const begin()` and `end()` methods

```
void print_vec(const Vector& vec) {
    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() { return data + the_size; }

    // define const begin()
    // define const end()

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

non-const
(allow modification)

`const begin()` and `end()` methods

```
void print_vec(const Vector& vec) {
    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() { return data + the_size; }

    // define const begin()
    ___ begin() ___ { return data; }
    // define const end()

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

non-const
(allow modification)

`const begin()` and `end()` methods

```
void print_vec(const Vector& vec) {
    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() { return data + the_size; }

    // define const begin()
    ___ begin() ___ { return data; }
    // define const end()

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

non-const
(allow modification)

`const begin()` and `end()` methods

```
void print_vec(const Vector& vec) {
    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() { return data + the_size; }

    // define const begin()
    ___ begin() _5_ { return data; }
    // define const end()

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

non-const
(allow modification)

Which keyword replaces blank #5 to indicate that the method will not modify the current object?

```
void print_vec(const Vector& vec) {
    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() { return data + the_size; }

    // define const begin()
    ___ begin() _5_ { return data; }
    // define const end()

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

non-const
(allow modification)

`const begin()` and `end()` methods

```
void print_vec(const Vector& vec) {
    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() { return data + the_size; }

    // define const begin()
    ___ begin() const { return data; }
    // define const end()

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

non-const
(allow modification)

`const begin()` and `end()` methods

```
void print_vec(const Vector& vec) {
    for (int elem: vec) {
        cout << elem << endl;
    }
}
```

compilation error

need to indicate int cannot be modified

```
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]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() { return data + the_size; }

    // define const begin()
    int* begin() const { return data; }
    // define const end()

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

non-const (allow modification)

To indicate that the value of the `int` *pointed at* by the returned `int*`, is the `const` keyword written before or after `int*`?

```
void print_vec(const Vector& vec) {
    for (int elem: vec) {
        cout << elem << endl;
    }
}
```

compilation error

need to indicate int cannot be modified

```
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]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() { return data + the_size; }

    // define const begin()
    int* begin() const { return data; }
    // define const end()

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

non-const (allow modification)

`const begin()` and `end()` methods

```
void print_vec(const Vector& vec) {
    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() { return data + the_size; }

    // define const begin()
    const int* begin() const { return data; }
    // define const end()

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

non-const
(allow modification)

`const begin()` and `end()` methods

```
void print_vec(const Vector& vec) {
    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() { return data + the_size; }

    // define const begin()
    const int* begin() const { return data; }
    // define const end()
    const int* end() const { return data + the_size; }

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

non-const (allow modification)

`const begin()` and `end()` methods

```
void print_vec(const Vector& vec) {
    ~~compilation error~~
    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() { return data + the_size; }

    const int* begin() const { return data; }
    const int* end() const { return data + the_size; }

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

non-const (allow modification)

const (no modification)

`const begin()` and `end()` methods

```
void print_vec(const Vector& vec) {

    for (int elem: vec) {
        cout << elem << 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]; }
    int& operator[](size_t i) { return data[i]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() { return data + the_size; }

    const int* begin() const { return data; }
    const int* end() const { return data + the_size; }

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

non-const (allow modification)

const (no modification)

An alternative index operator implementation

```
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]; }
    void clear() { the_size = 0; }
    void pop_back() { --the_size; }

    int* begin() { return data; }
    int* end() { return data + the_size; }

    const int* begin() const { return data; }
    const int* end() const { return data + theSize; }

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

An alternative index operator implementation

```
class Vector {
public:
    ...
    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;
};
```

An alternative index operator implementation

```
class Vector {
public:
    ...
    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;
};
```

An alternative index operator implementation

```
class Vector {
public:
    ...
    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;
};
```

An alternative index operator implementation

```
class Vector {
public:
    ...
    int operator[](size_t i) const {
        // return data[i];
        return ___;
    }

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

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

An alternative index operator implementation

```
class Vector {
public:
    ...
    int operator[](size_t i) const {
        // return data[i];
        return _6_;
    }

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

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

Which expression using pointer arithmetic can replace blank #6 to return the correct value?

```
class Vector {
public:
    ...
    int operator[](size_t i) const {
        // return data[i];
        return _6_;
    }

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

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

An alternative index operator implementation

```
class Vector {
public:
    ...
    int operator[](size_t i) const {
        // return data[i];
        return *(data + i);
    }

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

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

An alternative index operator implementation

```
class Vector {
public:
    ...
    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;
};
```

Just for fun...

index notation more readable

An alternative index operator implementation

```
class Vector {
public:
    ...
    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;
};
```