TDDD38/726G82: Adv. Programming in C++

Fundamentals III

Christoffer Holm

Department of Computer and information science

LINKÖPING UNIVERSITY

LINKÖPING UNIVERSITY

LINKÖPING UNIVERSITY

Lifetime & Special Member Functions

```
class Array
{
public:
  explicit Array(std::size_t size)                // constructor
    : size { size }, data { new int[size] }
  { }

  int& operator[](std::size_t index)              // non-const access
  { return data[index]; }

  int const& operator[](std::size_t index) const // const access
  { return data[index]; }

  std::size_t size() const                        // get the size
  { return size; }

private:
  std::size_t size;
  int* data;
};
```

Lifetime & Special Member Functions

RAII

```
class Array
{
public:
  explicit Array(std::size_t size)                // constructor
    : size { size }, data { new int[size] }
  { }

  ~Array()
  {
    delete[] data;
  }

  // ...

private:
  std::size_t size;
  int* data;
};
```

Lifetime & Special Member Functions

Copies

```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```

Lifetime & Special Member Functions

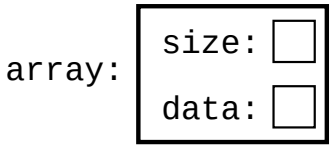
Copies

```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```

Lifetime & Special Member Functions

Copies

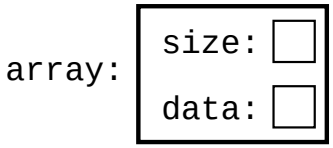
```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```



Lifetime & Special Member Functions

Copies

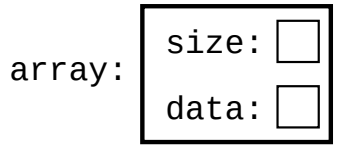
```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```



Lifetime & Special Member Functions

Copies

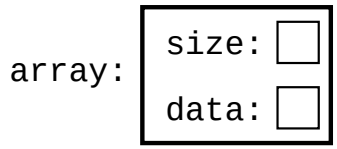
```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```



Lifetime & Special Member Functions

Copies

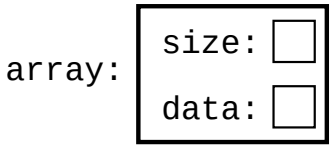
```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```



Lifetime & Special Member Functions

Copies

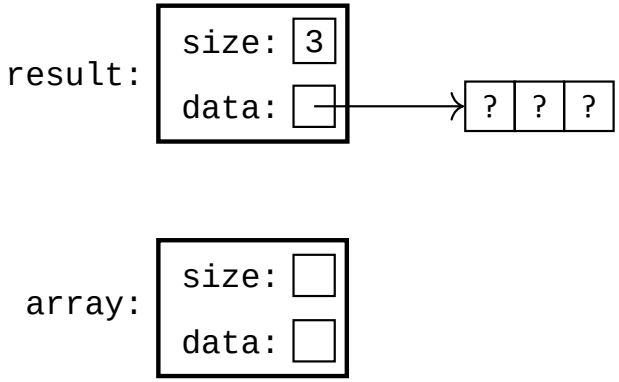
```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```



Lifetime & Special Member Functions

Copies

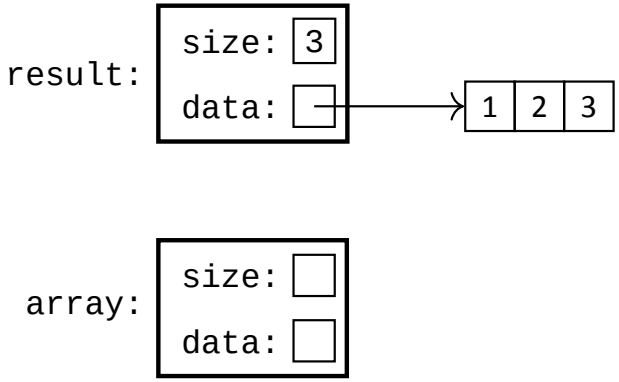
```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```



Lifetime & Special Member Functions

Copies

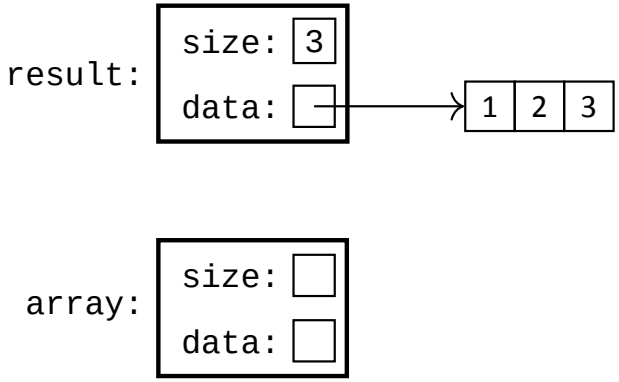
```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```



Lifetime & Special Member Functions

Copies

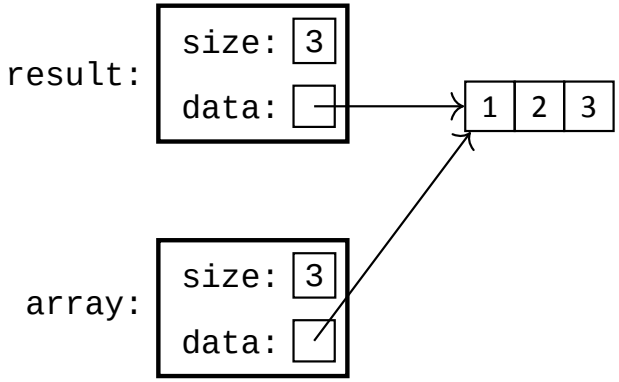
```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```



Lifetime & Special Member Functions

Copies

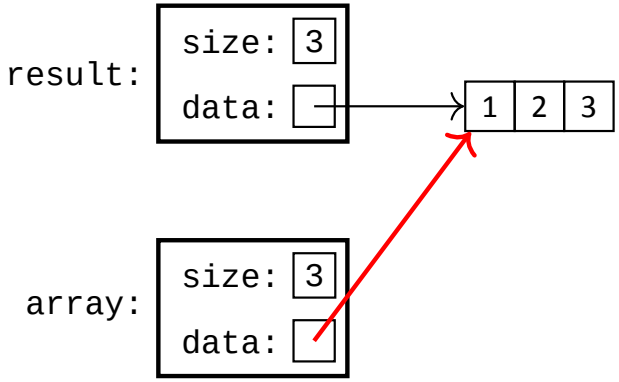
```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```



Lifetime & Special Member Functions

Copies

```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```



Lifetime & Special Member Functions

Copy constructor

```
class Array
{
public:
  // ...

  Array(Array const& other) // copy constructor
    : size { other.size },
      data { new int[size] }
  {
    // copy everything from other into our own allocation
    for (std::size_t i { 0 }; i < size; ++i)
      data[i] = other.data[i];
  }

  // ...
private:
  std::size_t size;
  int* data;
};
```

Lifetime & Special Member Functions

Copies

```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```

Lifetime & Special Member Functions

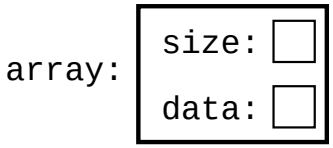
Copies

```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```

Lifetime & Special Member Functions

Copies

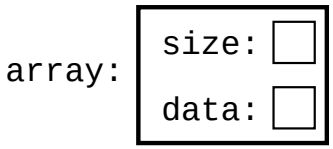
```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```



Lifetime & Special Member Functions

Copies

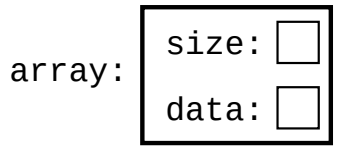
```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```



Lifetime & Special Member Functions

Copies

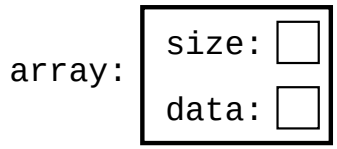
```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```



Lifetime & Special Member Functions

Copies

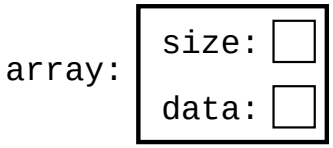
```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```



Lifetime & Special Member Functions

Copies

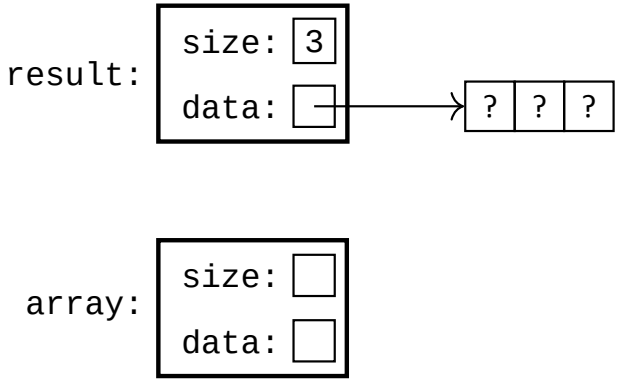
```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```



Lifetime & Special Member Functions

Copies

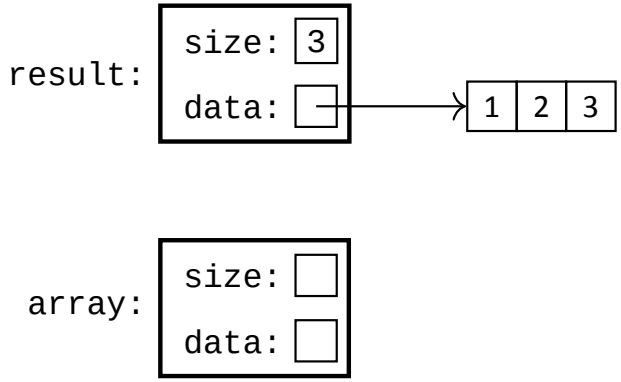
```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```



Lifetime & Special Member Functions

Copies

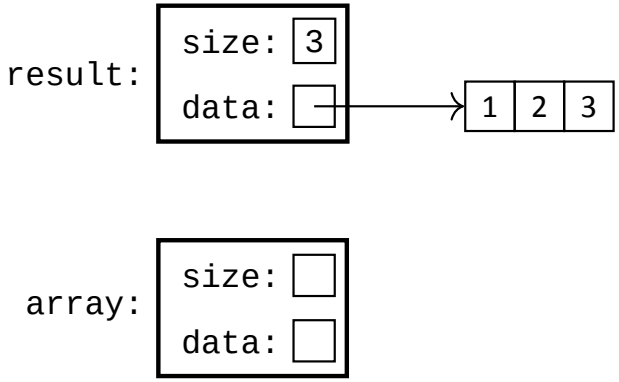
```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```



Lifetime & Special Member Functions

Copies

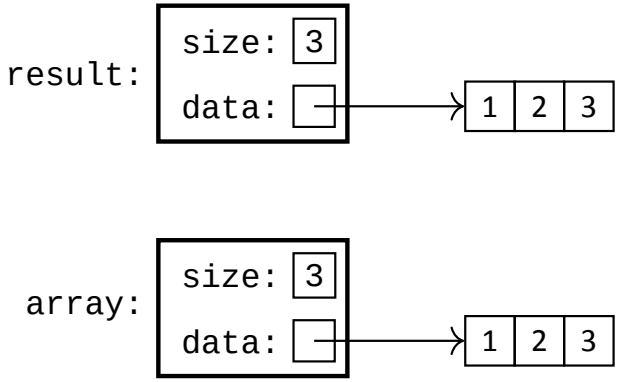
```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```



Lifetime & Special Member Functions

Copies

```
Array create(int a, int b, int c)
{
  Array result { 3 };
  result[0] = a;
  result[1] = b;
  result[2] = c;
  return result;
}

int main()
{
  Array array { create(1, 2, 3) };
  // ...
}
```



Lifetime & Special Member Functions

Copy assignment

```
Array a { create(1, 2, 3) };
Array b { create(2, 3, 4) };

b = a; // copy a into b
```

Lifetime & Special Member Functions

Copy assignment operator

```
class Array
{
public:
  // ...
  Array& operator=(Array const& other)
  {
    if (this != &other)
    {
      delete[] data;
      size = other.size;
      data = new int[size];
      for (std::size_t i { 0 }; i < size; ++i)
        data[i] = other.data[i];
    }
    return *this;
  }
  // ...
private:
  std::size_t size;
  int* data;
};
```

Lifetime & Special Member Functions

Copy-and-swap idiom

```
class Array
{
public:
  // ...
  Array& operator=(Array const& other)
  {
    // reuse copy constructor
    Array copy { other };

    // swap the members
    std::swap(size, other.size);
    std::swap(data, other.data);

    // copy is destroyed which releases the previous data
    return *this;
  }
  // ...
private:
  std::size_t size;
  int* data;
};
```

Lifetime & Special Member Functions

Opportunity for optimization

```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```

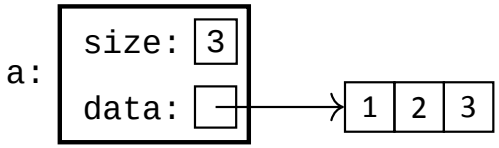
Lifetime & Special Member Functions

Opportunity for optimization

```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```

Lifetime & Special Member Functions

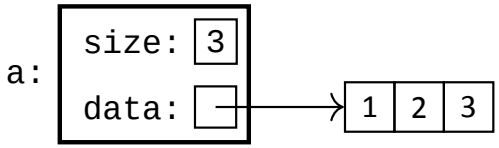
Opportunity for optimization




```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```

Lifetime & Special Member Functions

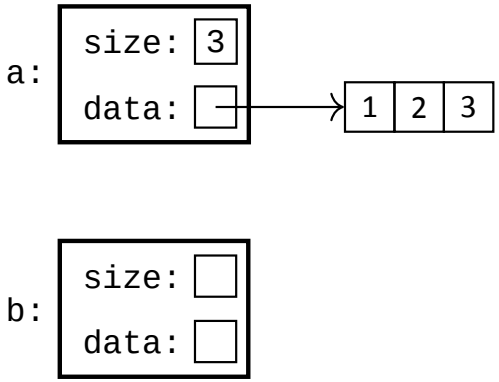
Opportunity for optimization




```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```

Lifetime & Special Member Functions

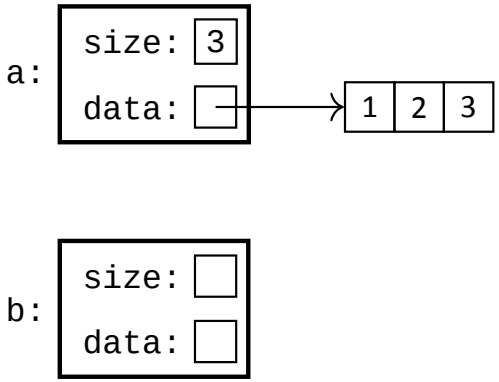
Opportunity for optimization




```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```

Lifetime & Special Member Functions

Opportunity for optimization

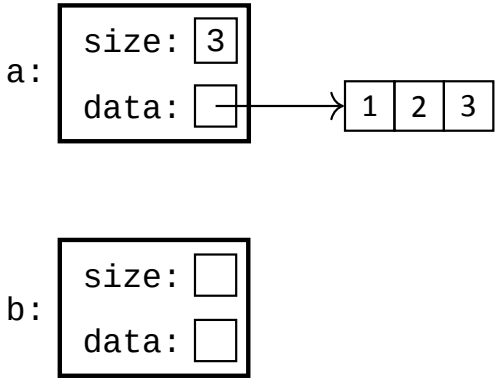



```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```

Lifetime & Special Member Functions

Opportunity for optimization

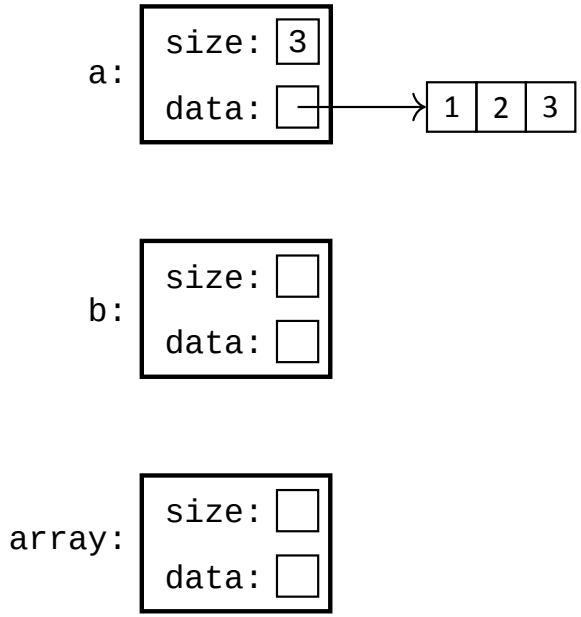
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

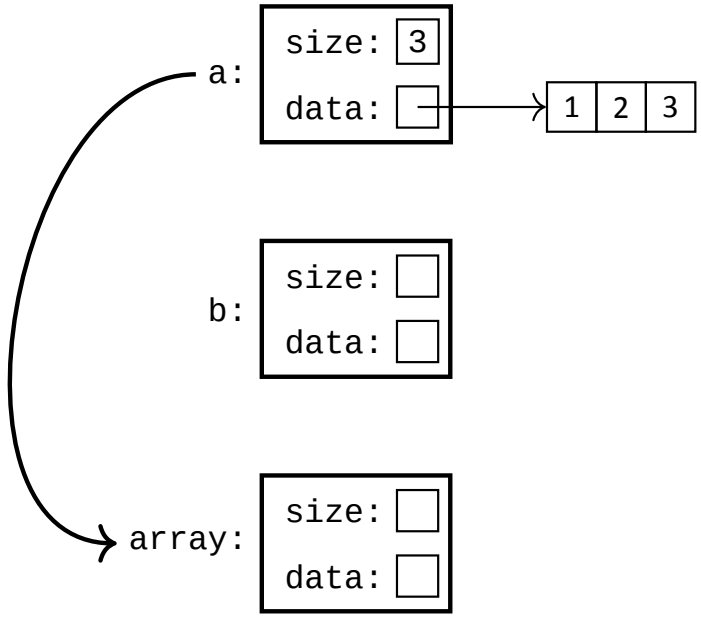
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

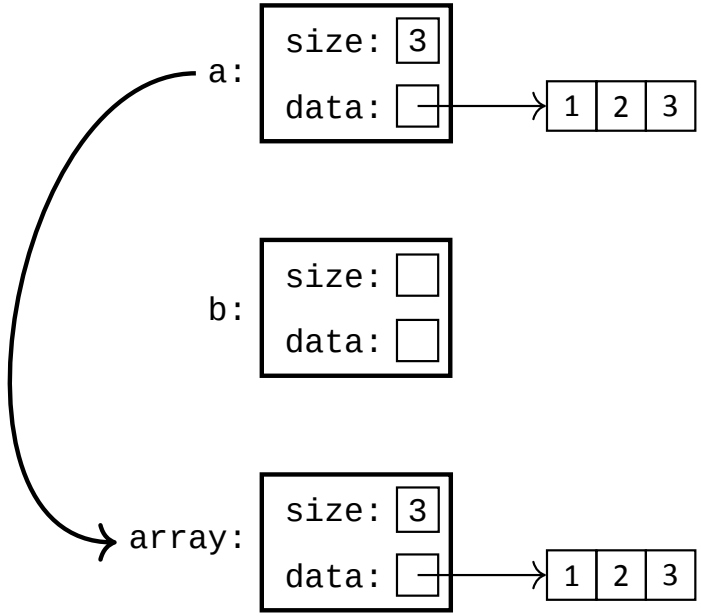
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

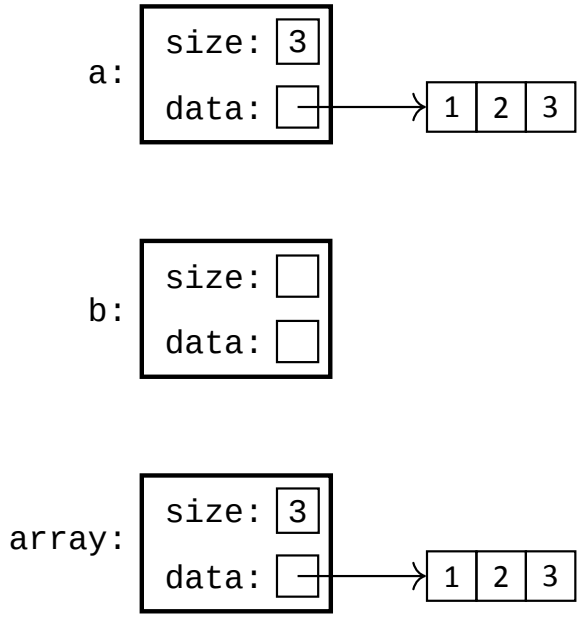
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

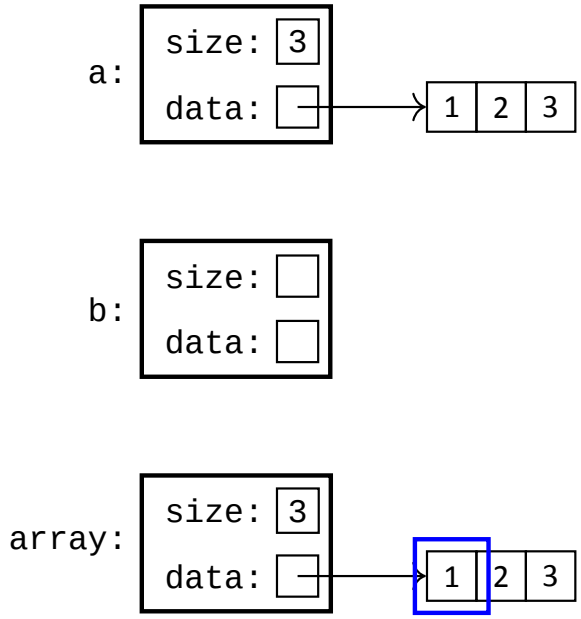
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

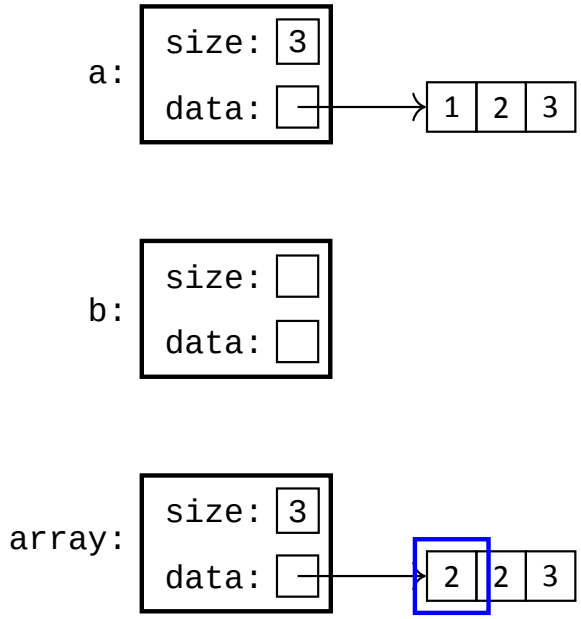
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

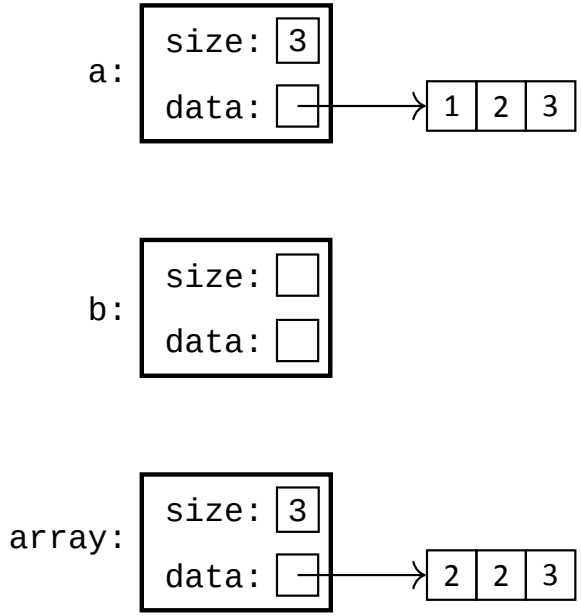
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

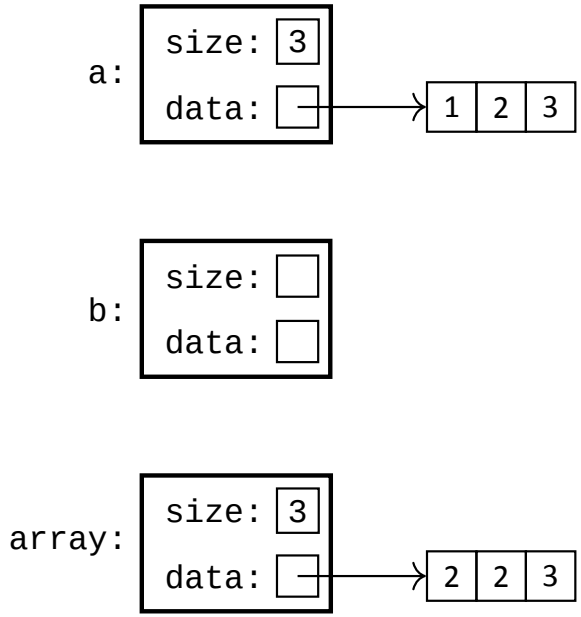
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

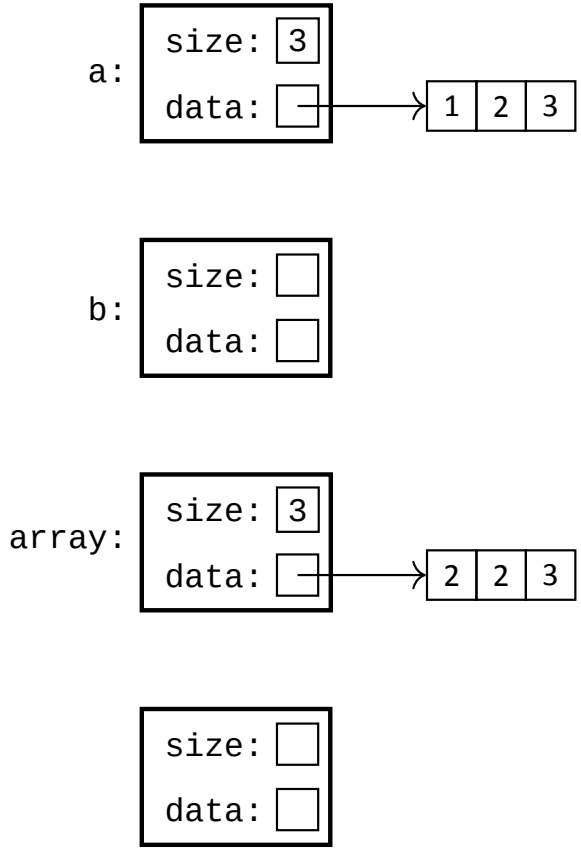
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

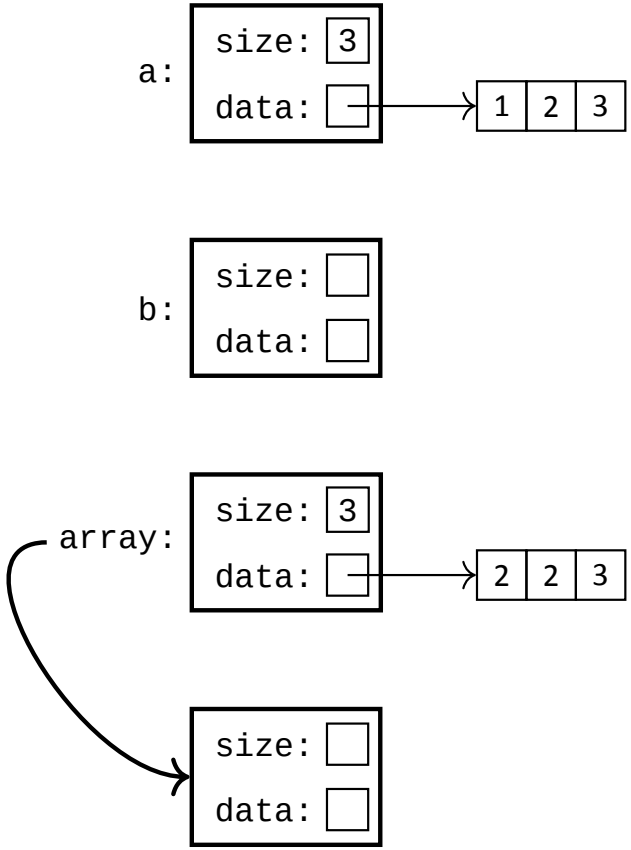
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

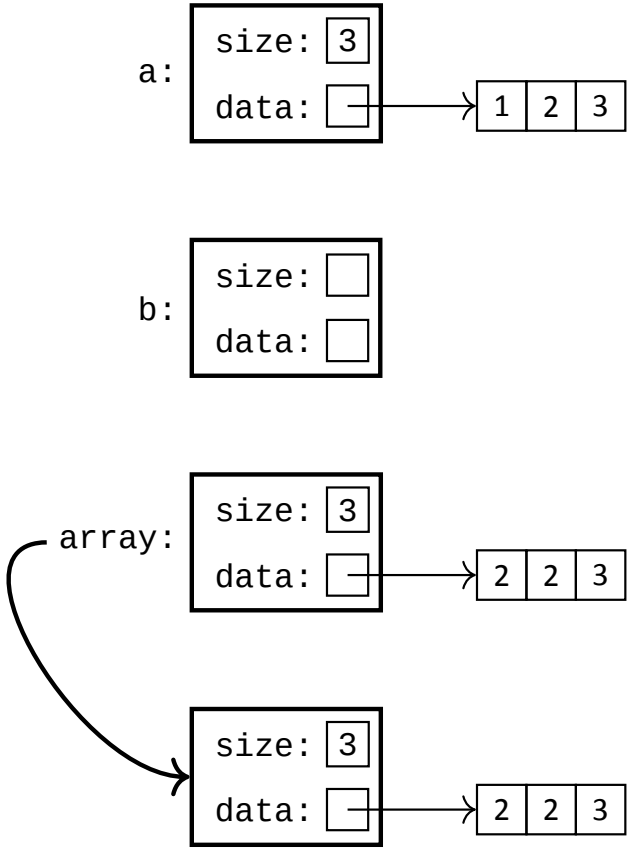
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

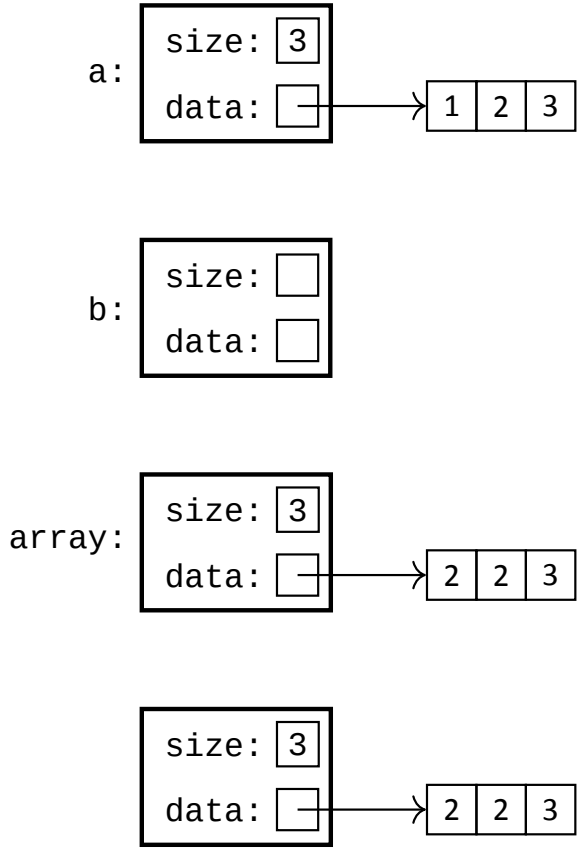
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

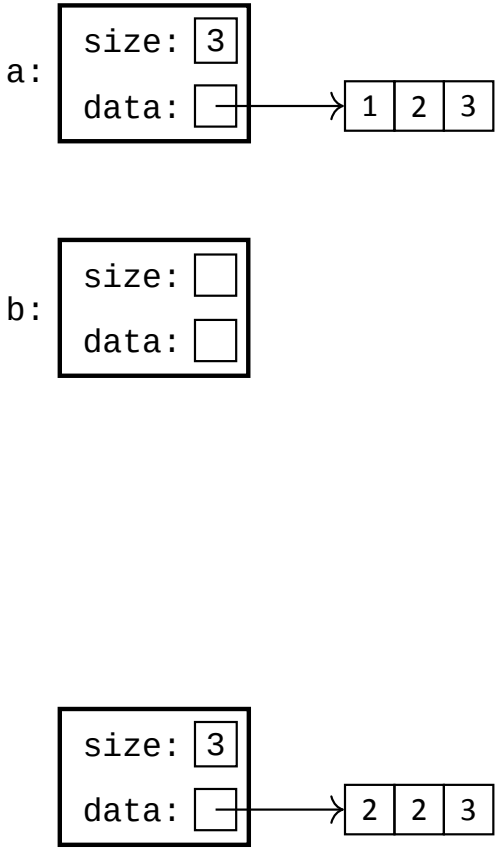
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

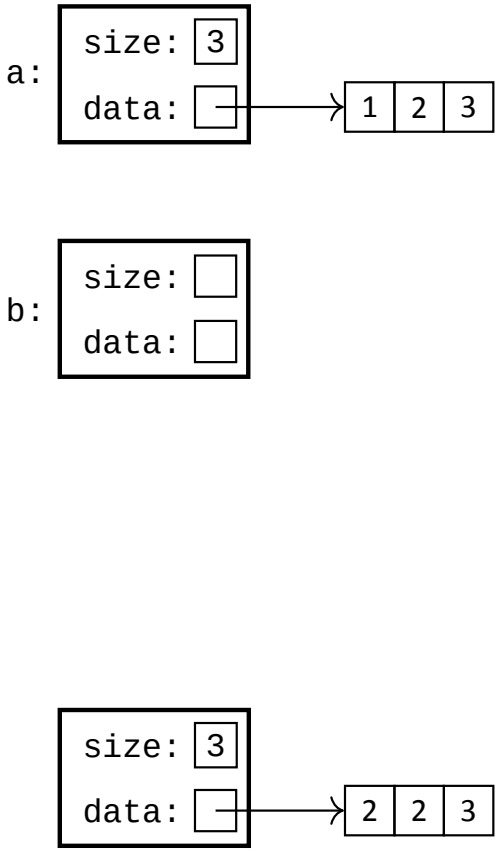
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

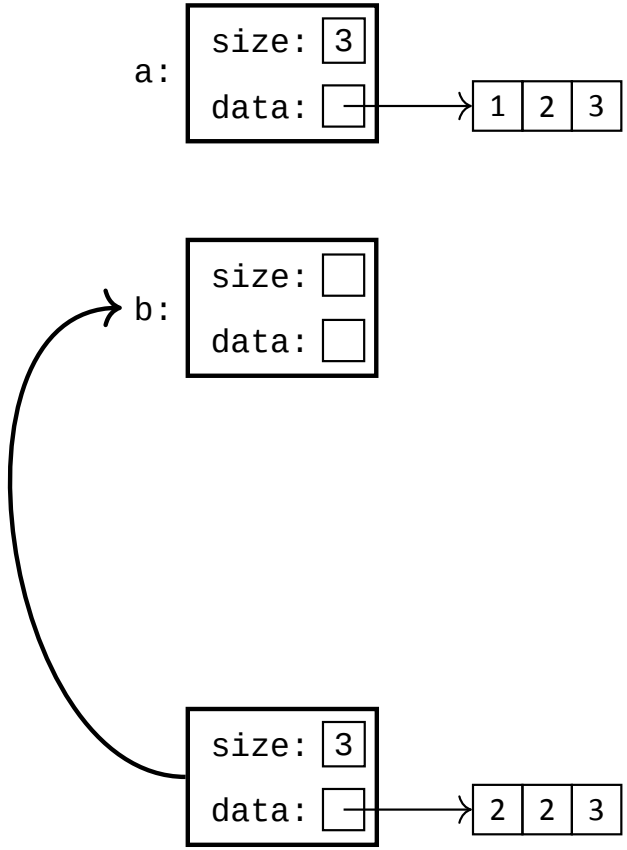
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

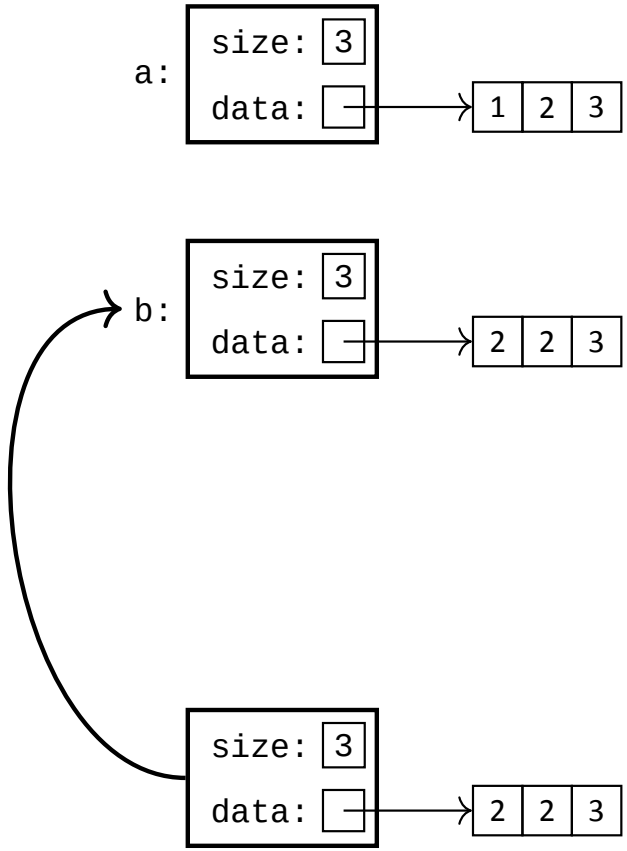
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

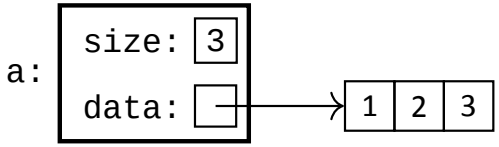
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```

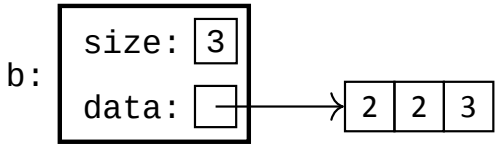


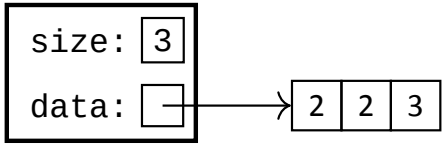
Lifetime & Special Member Functions

Opportunity for optimization

```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



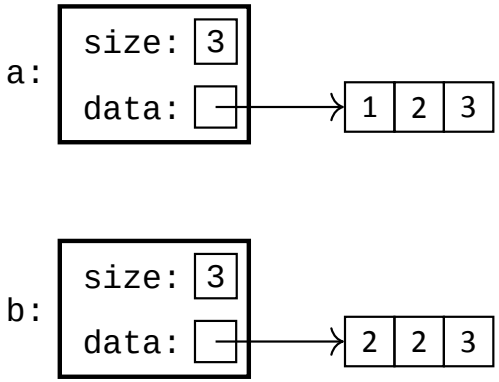




Lifetime & Special Member Functions

Opportunity for optimization

```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

We can do better!

Lifetime & Special Member Functions

Opportunity for optimization

```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```

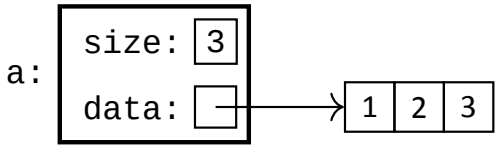
Lifetime & Special Member Functions

Opportunity for optimization

```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```

Lifetime & Special Member Functions

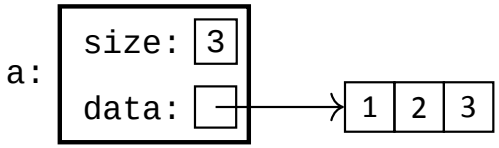
Opportunity for optimization




```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```

Lifetime & Special Member Functions

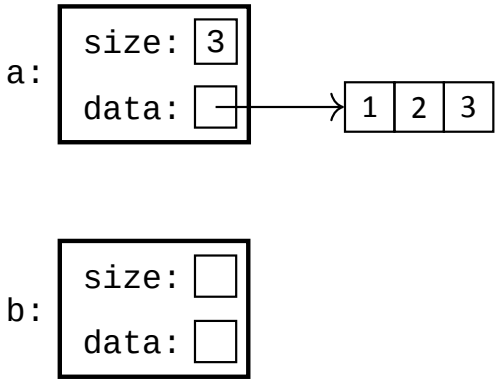
Opportunity for optimization




```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```

Lifetime & Special Member Functions

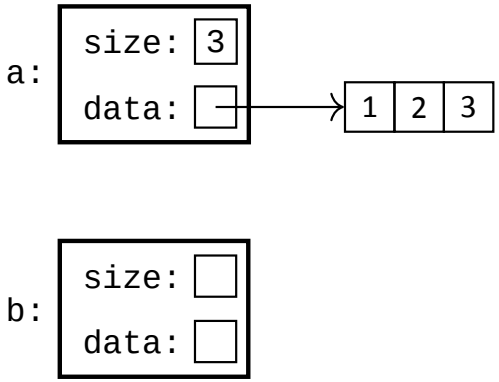
Opportunity for optimization




```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```

Lifetime & Special Member Functions

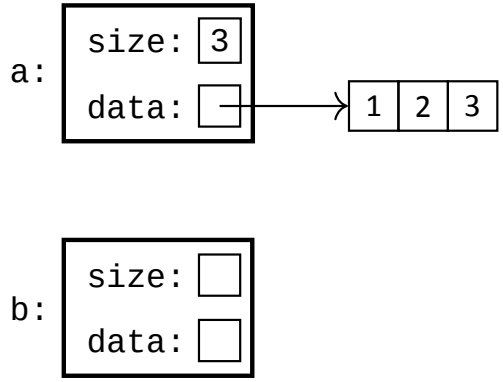
Opportunity for optimization




```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```

Lifetime & Special Member Functions

Opportunity for optimization

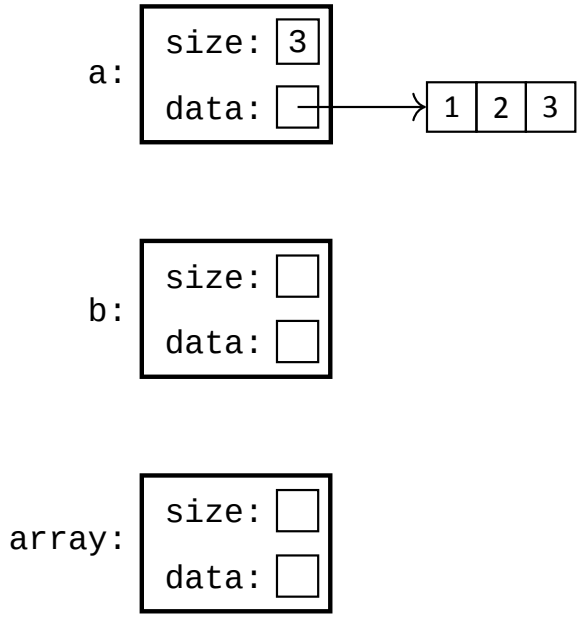



```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```

Lifetime & Special Member Functions

Opportunity for optimization

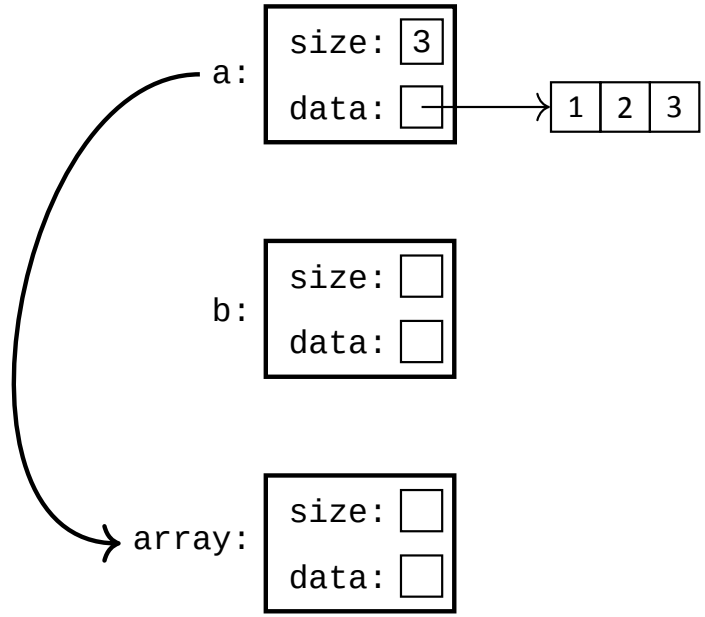
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

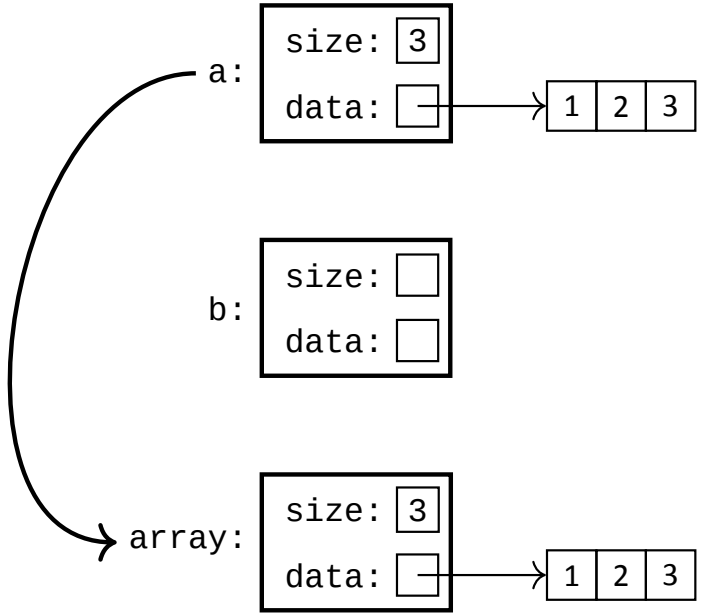
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

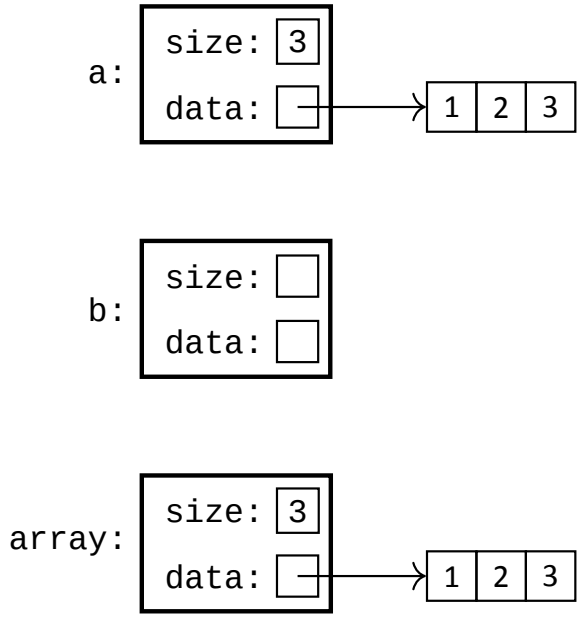
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

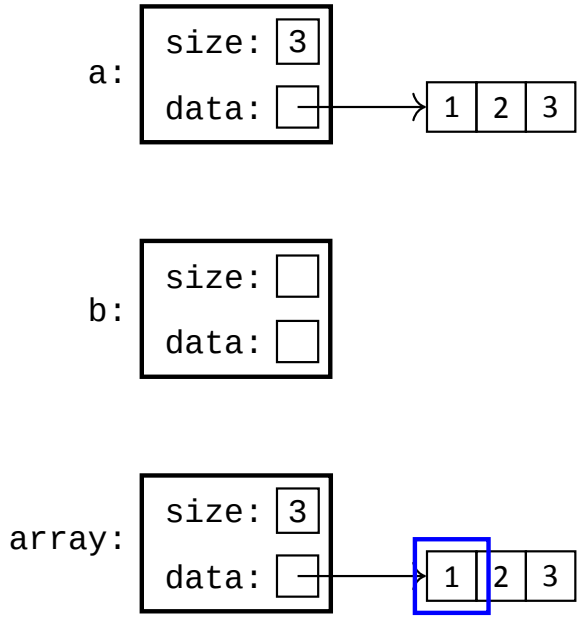
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

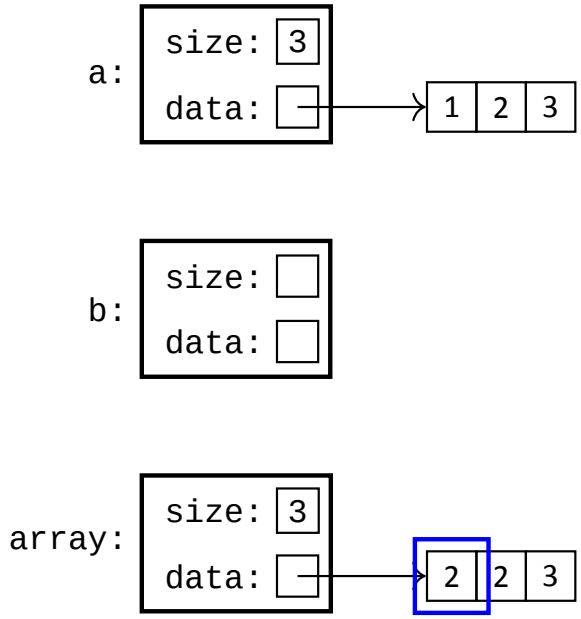
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

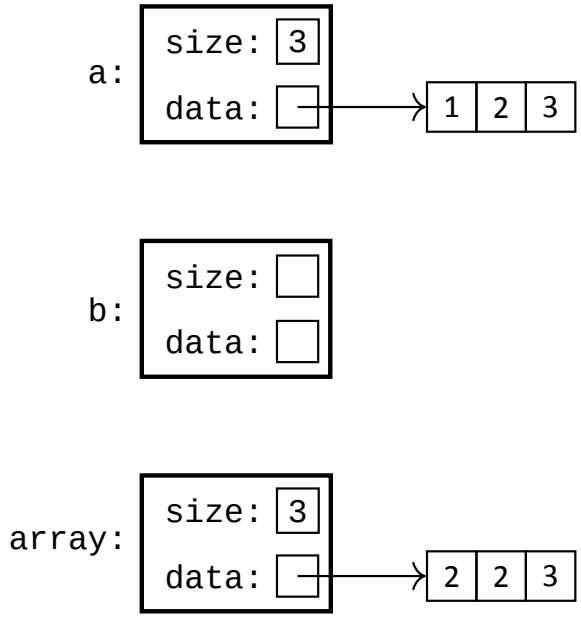
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

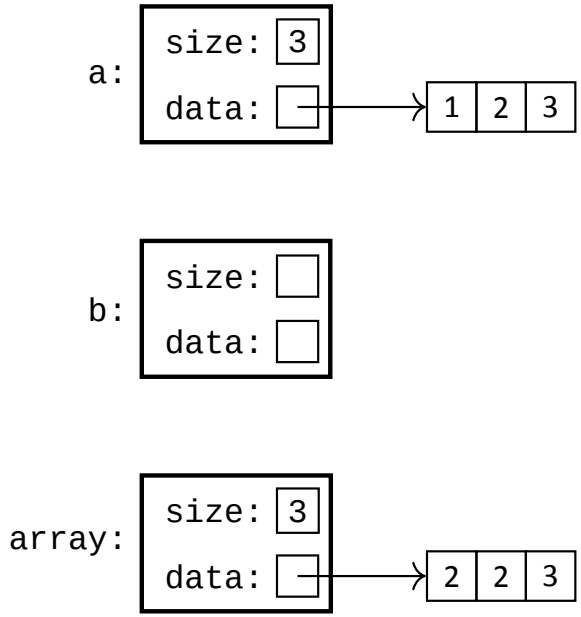
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

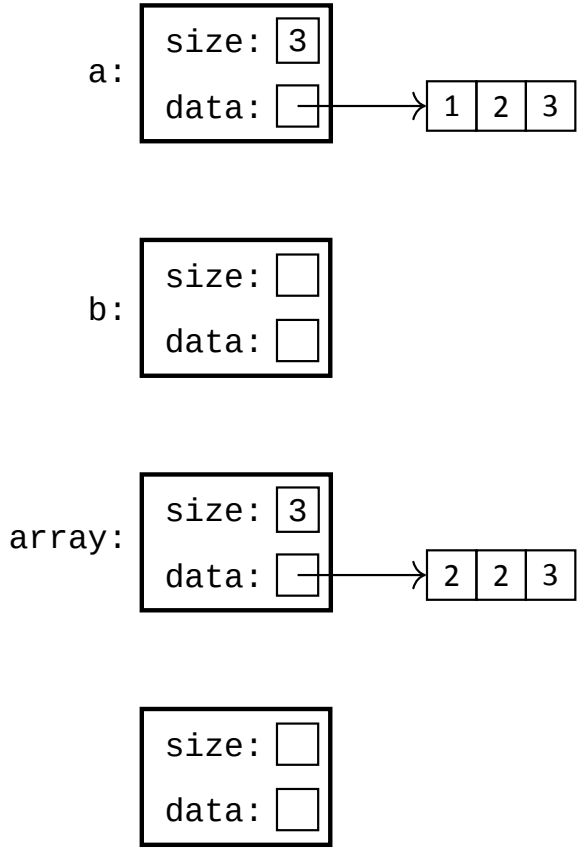
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

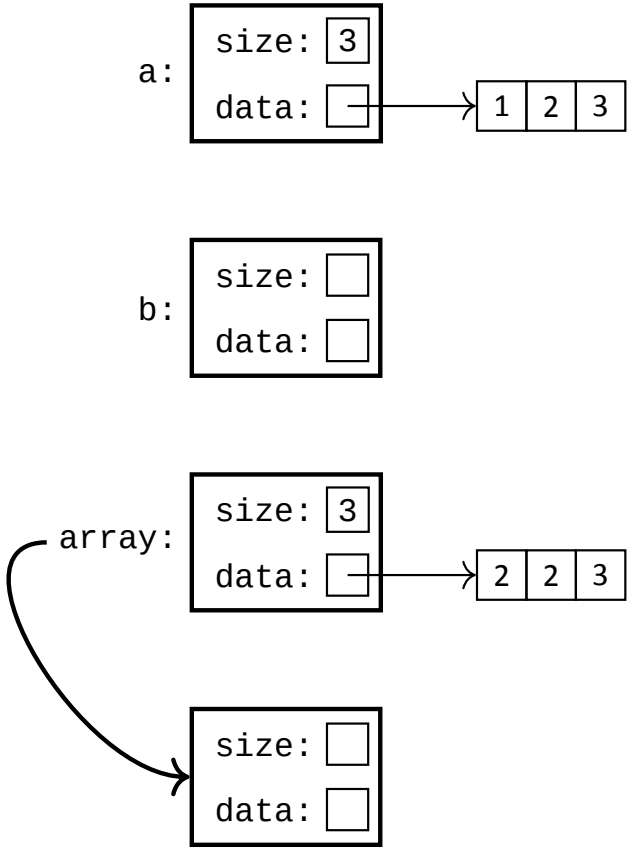
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

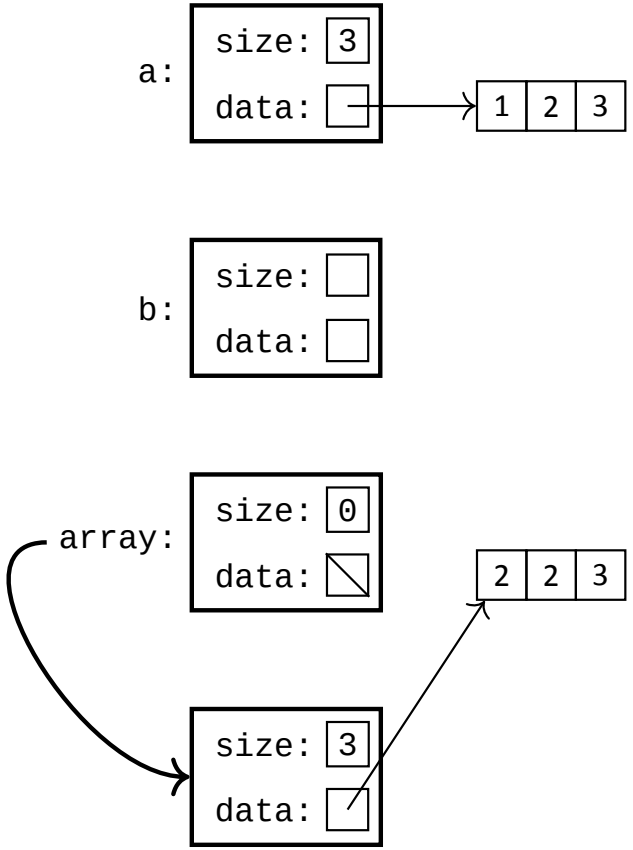
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

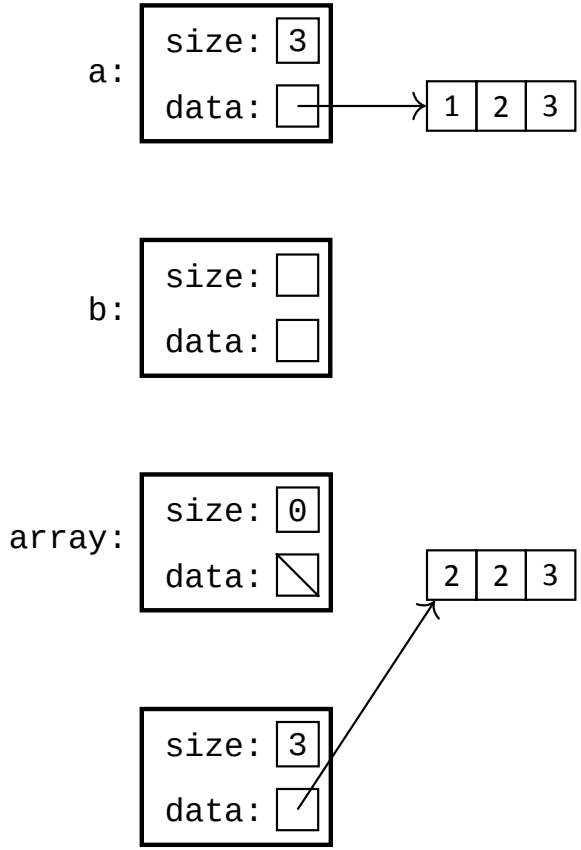
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

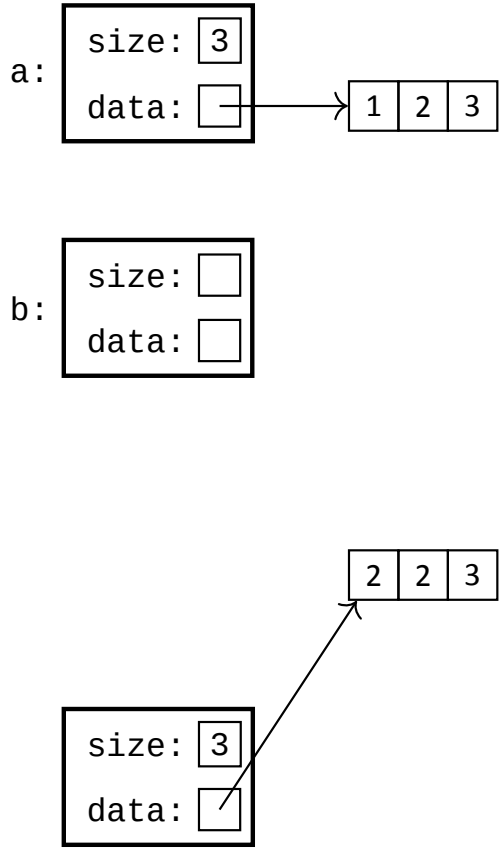
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

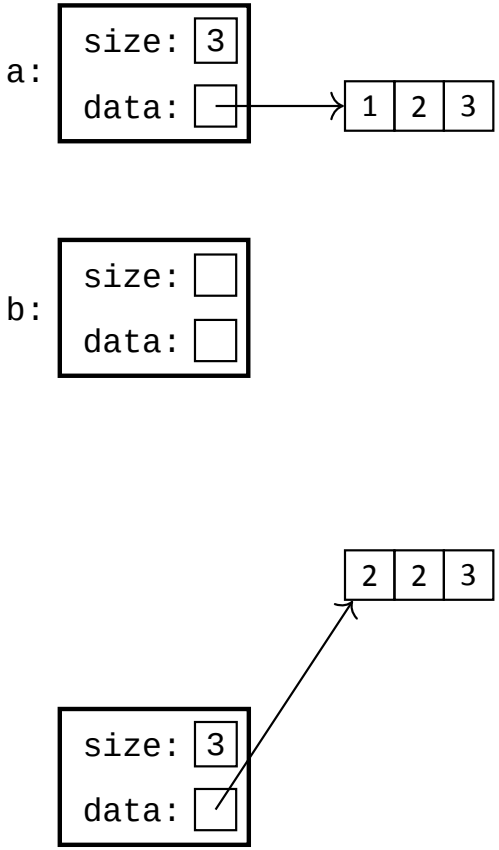
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

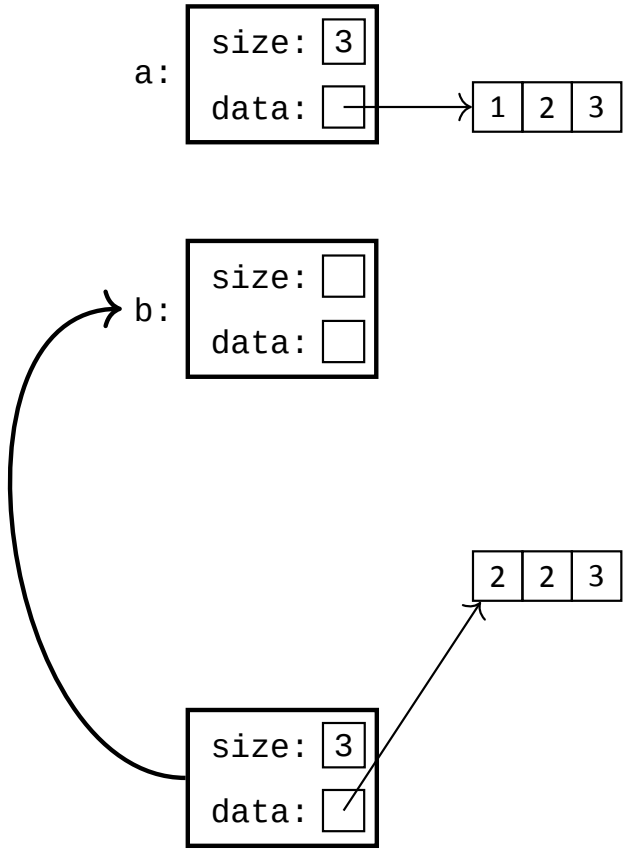
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

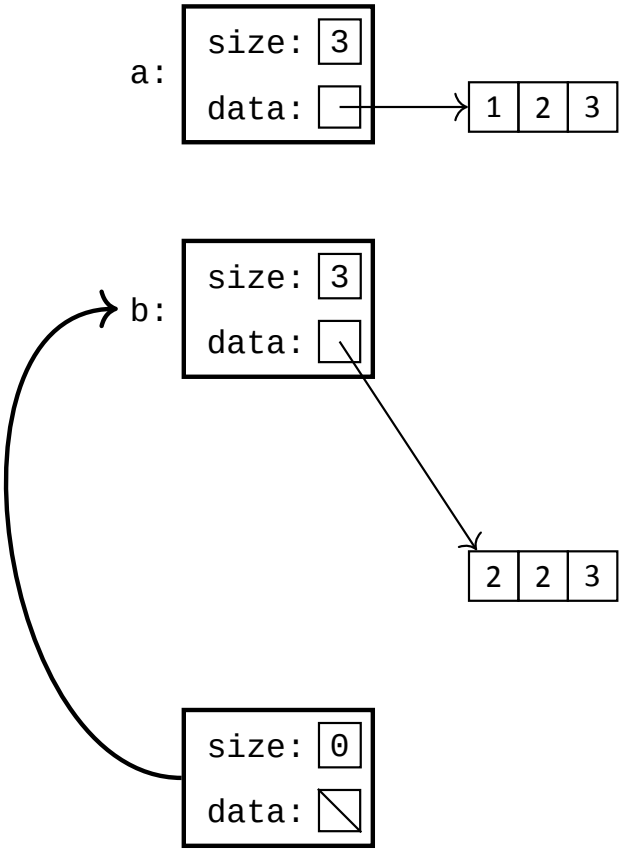
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

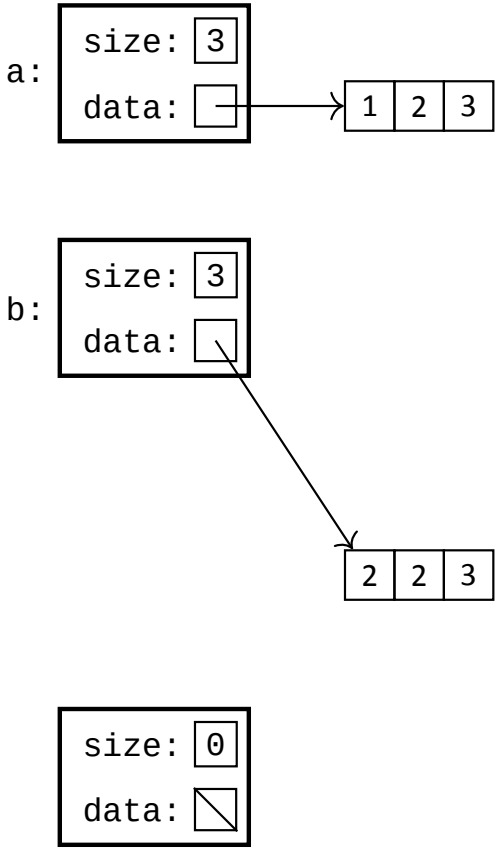
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

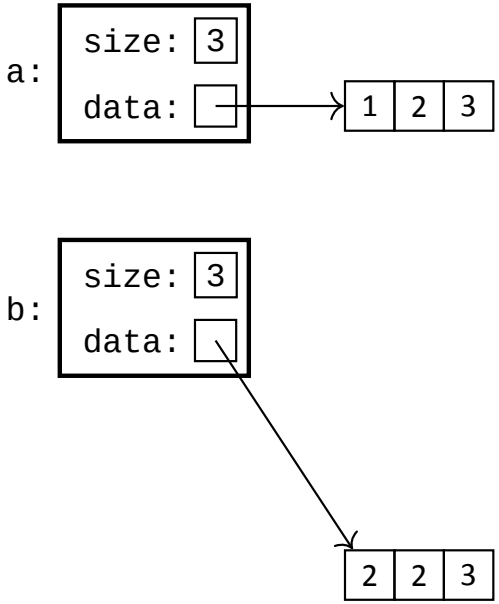
```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Opportunity for optimization

```
Array modify(Array array)
{
  ++array[0];
  return array;
}

int main()
{
  Array a { create(1, 2, 3) };
  Array b { modify(a) };
}
```



Lifetime & Special Member Functions

Let's implement *move* semantics

```
class Array
{
public:
  // ...
  Array(Array&& other)
    : size { other.size }, data { other.data }
  {
    other.size = 0;
    array.data = nullptr;
  }

  Array& operator=(Array&& other)
  {
    std::swap(size, other.size);
    std::swap(data, other.data);
    return *this;
  }
  // ...
};
```

Lifetime & Special Member Functions

Special member functions

```
class Array
{
public:
  // ...
  Array(Array const& other);            // copy constructor
  Array(Array&& other);                 // move constructor
  ~Array();                             // destructor
  Array& operator=(Array const& other); // copy assignment operator
  Array& operator=(Array&& other);      // move assignment operator
  // ...
};
```

Lifetime & Special Member Functions

Rule of N

- rule of three
- rule of five
- rule of zero

LINKÖPING UNIVERSITY

Value categories

rvalue references

```
int  x { 3 };
// lvalue references to x
int& lref { x };

// rvalue reference to x + 1
int&& rref { x + 1 };

++x;
cout << lref << endl; // prints 4
cout << rref << endl; // also prints 4??
```

Value categories

rvalues and function overloads

```
class Array
{
  // ...
  void test(Array const&); // #1
  void test(Array&&);      // #2
  // ...
};

int main()
{
  Array a1 { create(1, 2, 3) };
  Array a2 { create(4, 5, 6) };

  a1.test(a2);              // argument is an lvalue so call #1
  a1.test(create(1, 2, 3)); // argument is an rvalue so call #2
}
```

Value categories

Consequences of rvalue references

```
1  void fun(Array&& array)
2  {
3    // can we move here?
4
5    Array other { array };
6
7    // ...
8  }
9
10 int main()
11 {
12   fun(create(1, 2, 3));
13 }
```

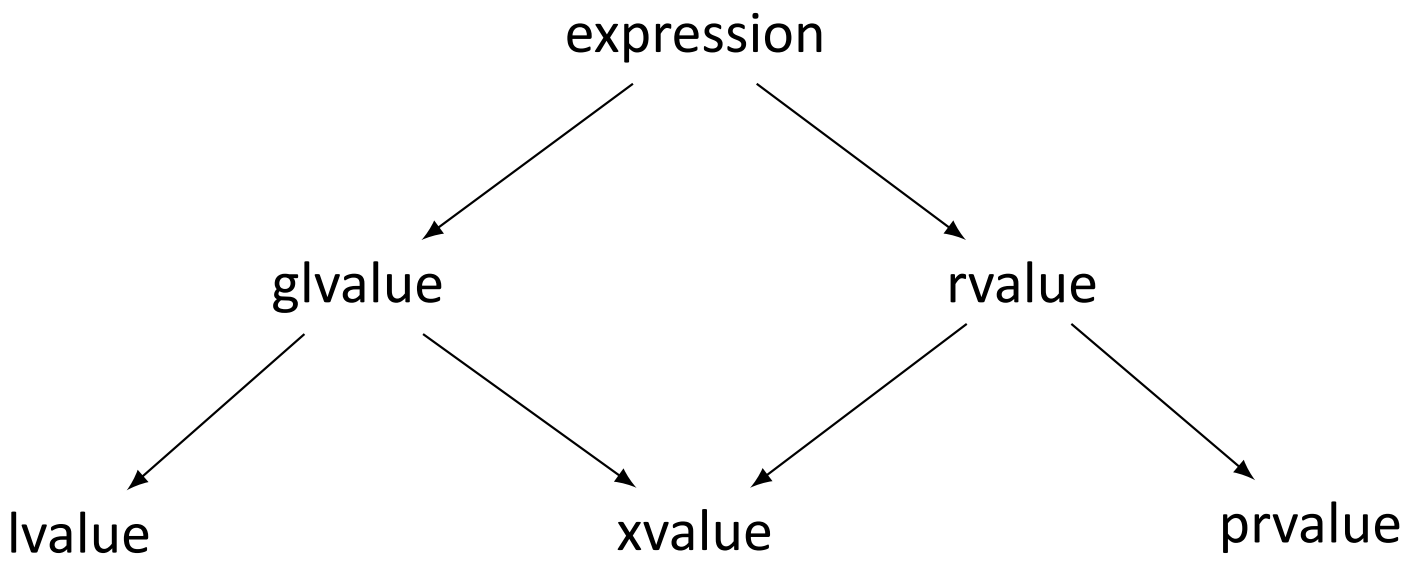
Value categories

Consequences of rvalue references

```
void fun(Array&& array)
{
  // NO! We might reference array later
  // which means we cannot steal its content
  Array other { array };
  // For example:
  cout << array[0] << endl;
}

int main()
{
  fun(create(1, 2, 3));
}
```

Value categories

Value categories: the complete picture



Value categories

Value categories: the complete picture

- We introduce *xvalues* (e**x**piring **values**) which are values that *temporarily* have identity (i.e. if they are bound to rvalue references)
- What we previously called *lvalues* are renamed to *glvalues* (**g**eneralized **lvalues**): expressions which refers to objects with identity
- What we previously called *rvalues* are renamed to *prvalues* (**p**ure **rvalues**): expressions *without* identity
- *lvalues* now refer to all glvalues that are **not** xvalues
- *rvalues* now refer to both xvalues and prvalues

LINKÖPING UNIVERSITY

Inheritance

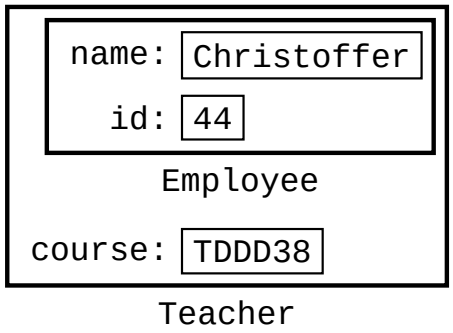
Mental Model

```
class Employee
{
  string name{"Christoffer"};
  int id{44};
};
class Teacher : public Employee
{
  string course{"TDDD38"};
};
Teacher c{};
```

Inheritance

Mental Model

```
class Employee
{
  string name{"Christoffer"};
  int id{44};
};
class Teacher : public Employee
{
  string course{"TDDD38"};
};
Teacher c{};
```



Inheritance

Protected members

```
class Base
{
public:
  Base(int x)
    : x{x} { }

private:
  int x;
};
```

```
struct Derived : Base
{
  Derived(int x)
    : Base{x} { }
  int get()
  {
    return x; // Error!
  }
};
```

Inheritance

Protected members

```
class Base
{
public:
  Base(int x)
    : x{x} { }

protected:
  int x;
};
```

```
struct Derived : Base
{
  Derived(int x)
    : Base{x} { }
  int get()
  {
    return x; // OK!
  }
};
```

Inheritance

Constructors

```
class Base
{
public:
  Base(int x);
private:
  int x;
};

Base::Base(int x)
  : x{x}
{
}
```

```
class Derived : public Base
{
public:
  Derived(int x, double y);
private:
  double y;
};

Derived::Derived(int x, double y)
  : Base{x}, y{y}
{
}
```

Inheritance

Initialization & Destruction

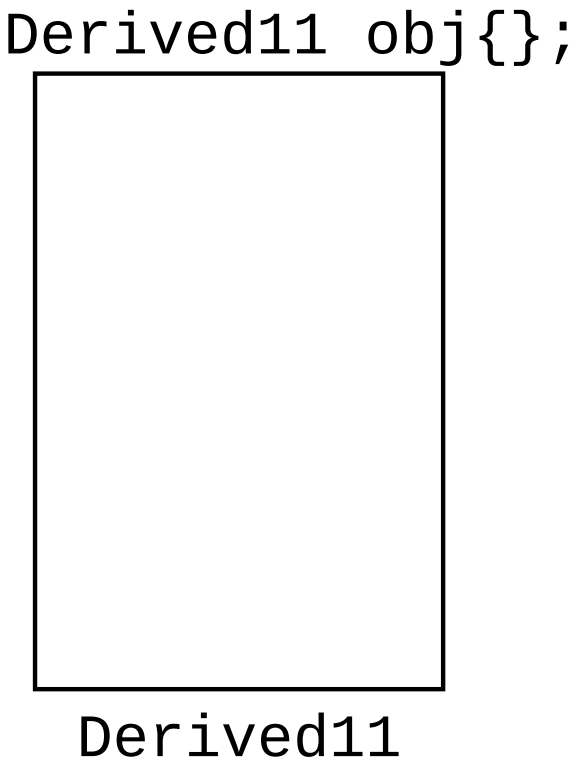
```
class Base
{
  int x{1};
};
class Derived1 : public Base
{
  double y{2.34};
};
class Derived11 final
  : public Derived1
{
  int z{56};
};
```

```
Derived11 obj{};
```

Inheritance

Initialization & Destruction

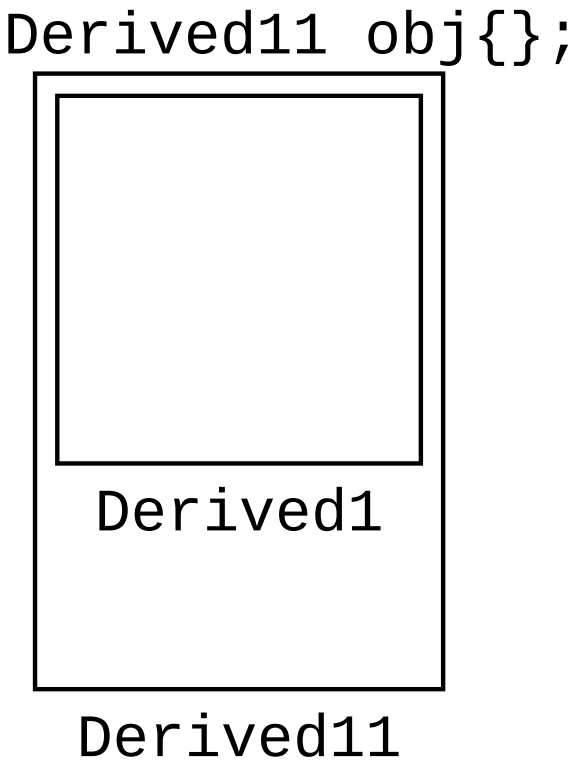
```
class Base
{
  int x{1};
};
class Derived1 : public Base
{
  double y{2.34};
};
class Derived11 final
  : public Derived1
{
  int z{56};
};
```



Inheritance

Initialization & Destruction

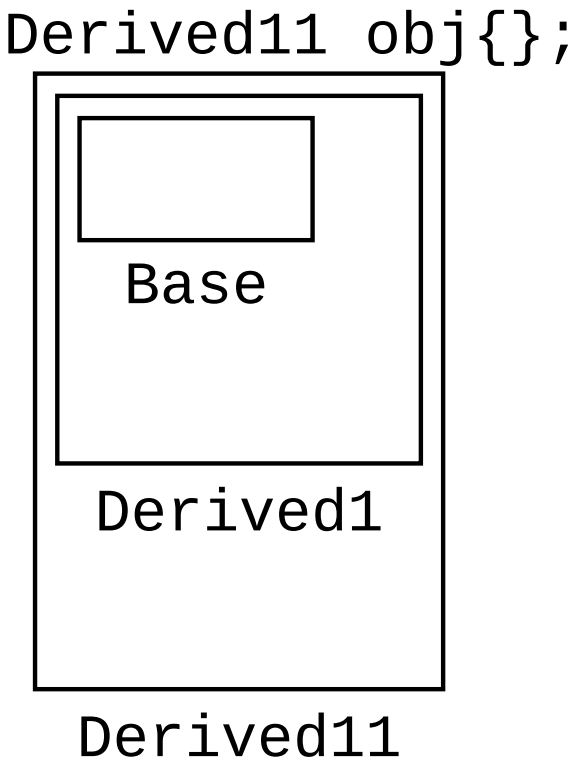
```
class Base
{
  int x{1};
};
class Derived1 : public Base
{
  double y{2.34};
};
class Derived11 final
  : public Derived1
{
  int z{56};
};
```



Inheritance

Initialization & Destruction

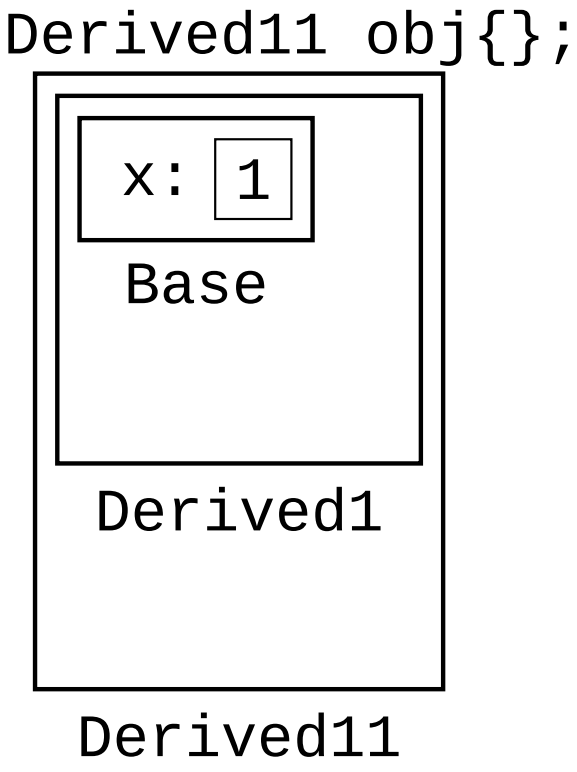
```
class Base
{
  int x{1};
};
class Derived1 : public Base
{
  double y{2.34};
};
class Derived11 final
  : public Derived1
{
  int z{56};
};
```



Inheritance

Initialization & Destruction

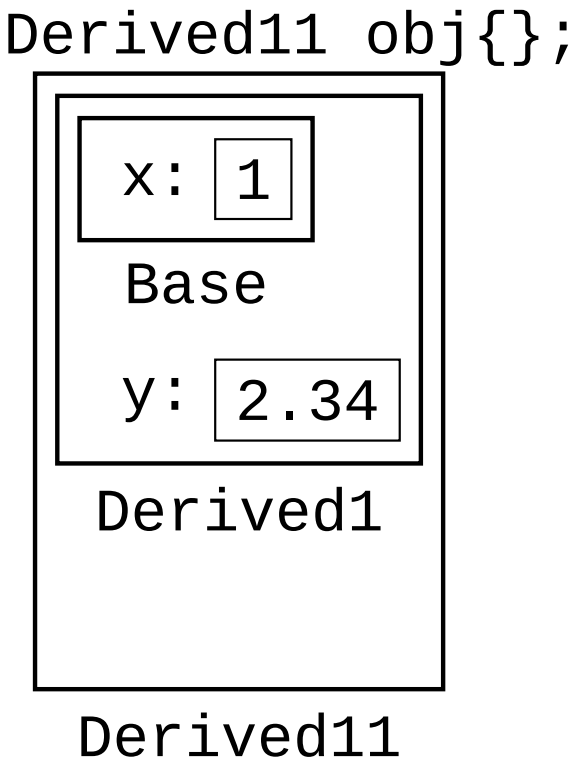
```
class Base
{
  int x{1};
};
class Derived1 : public Base
{
  double y{2.34};
};
class Derived11 final
  : public Derived1
{
  int z{56};
};
```



Inheritance

Initialization & Destruction

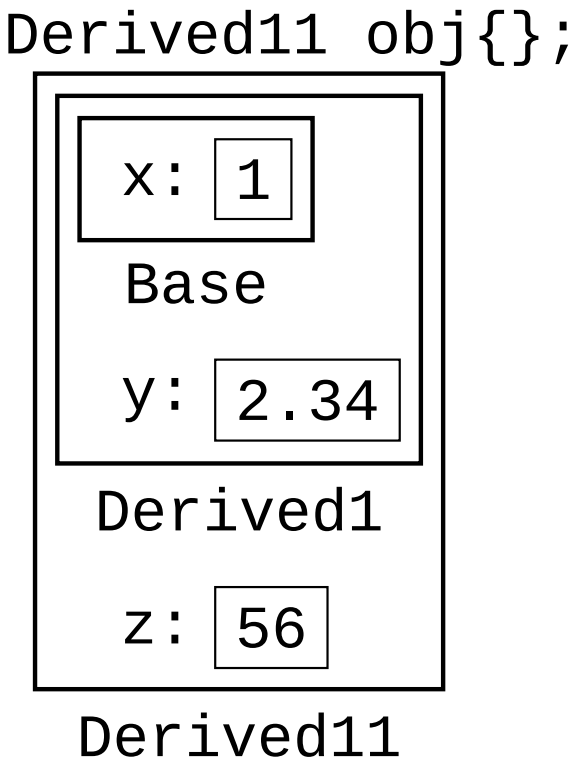
```
class Base
{
  int x{1};
};
class Derived1 : public Base
{
  double y{2.34};
};
class Derived11 final
  : public Derived1
{
  int z{56};
};
```



Inheritance

Initialization & Destruction

```
1  class Base
2  {
3    int x{1};
4  };
5  class Derived1 : public Base
6  {
7    double y{2.34};
8  };
9  class Derived11 final
10   : public Derived1
11 {
12   int z{56};
13 };
```

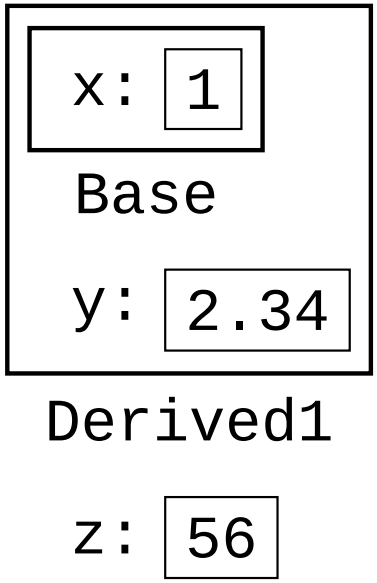


Inheritance

Initialization & Destruction

```
class Base
{
  int x{1};
};
class Derived1 : public Base
{
  double y{2.34};
};
class Derived11 final
  : public Derived1
{
  int z{56};
};
```

```
Derived11 obj{};
```

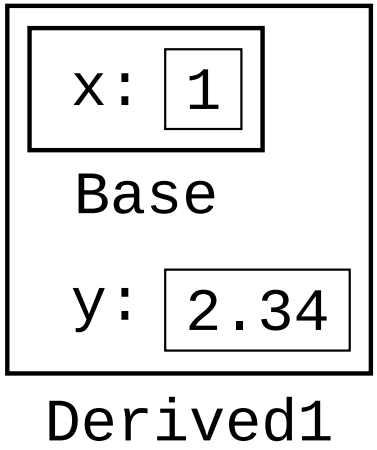


Inheritance

Initialization & Destruction

```
class Base
{
  int x{1};
};
class Derived1 : public Base
{
  double y{2.34};
};
class Derived11 final
  : public Derived1
{
  int z{56};
};
```

```
Derived11 obj{};
```

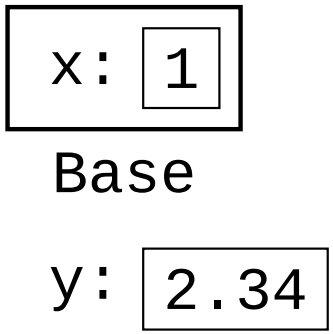


Inheritance

Initialization & Destruction

```
1  class Base
2  {
3    int x{1};
4  };
5  class Derived1 : public Base
6  {
7    double y{2.34};
8  };
9  class Derived11 final
10   : public Derived1
11 {
12   int z{56};
13 };
```

```
Derived11 obj{};
```

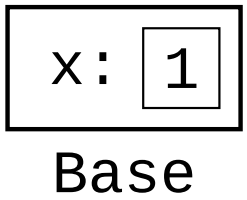


Inheritance

Initialization & Destruction

```
class Base
{
  int x{1};
};
class Derived1 : public Base
{
  double y{2.34};
};
class Derived11 final
  : public Derived1
{
  int z{56};
};
```

```
Derived11 obj{};
```



Inheritance

Initialization & Destruction

```
1  class Base
2  {
3    int x{1};
4  };
5  class Derived1 : public Base
6  {
7    double y{2.34};
8  };
9  class Derived11 final
10   : public Derived1
11 {
12   int z{56};
13 };
```

```
Derived11 obj{};
```

x: 1

Inheritance

Initialization & Destruction

```
class Base
{
  int x{1};
};
class Derived1 : public Base
{
  double y{2.34};
};
class Derived11 final
  : public Derived1
{
  int z{56};
};
```

```
Derived11 obj{};
```

Inheritance

Types of Inheritance

- `public` inheritance
- `protected` inheritance
- `private` inheritance

Inheritance

What will happen? Why?

```
1  struct Base
2  {
3    ~Base() { cout << "Base" << endl; }
4  };
5  struct Derived : public Base
6  {
7    ~Derived() { cout << "Derived" << endl; }
8  };
9  int main()
10 {
11   Derived d{};
12 }
```

LINKÖPING UNIVERSITY

Polymorphism

Dynamic dispatch

```
void print1()
{  cout << "1" << endl; }

struct Base
{
  Base() = default;
  void print()
  {
    foo();
  }

protected:
  using function_t = void (*)();

  Base(function_t foo)
    : foo{foo} { }

private:
  function_t foo{print1};
};
```

```
void print2()
{ cout << "2" << endl; }

struct Derived : public Base
{
  // define default constructor
  Derived()
     // call the base constructor
   : Base{print2} { }
};

int main()
{
  Base* bp {new Base{}};
  bp->print();
  delete bp;

  bp = new Derived{};
  bp->print();
}
```

Polymorphism

Easier dynamic dispatch

```
struct Base
{
  virtual void print()
  {
    cout << "1" << endl;
  }
};

struct Derived : public Base
{
  void print() override
  {
    cout << "2" << endl;
  }
};
```

```
int main()
{
  Base* bp {new Base{}};
  bp->print();
  delete bp;

  bp = new Derived{};
  bp->print();
}
```

Polymorphism

What will happen? Why?

```
1  struct Base
2  {
3    ~Base() { cout << "Base" << endl; }
4  };
5  struct Derived : public Base
6  {
7    ~Derived() { cout << "Derived" << endl; }
8  };
9  int main()
10 {
11   Base* bp{new Derived()};
12   delete bp;
13 }
```

Polymorphism

What will happen? Why?

```
1  struct Base
2  {
3    virtual ~Base() { cout << "Base" << endl; }
4  };
5  struct Derived : public Base
6  {
7    ~Derived() { cout << "Derived" << endl; }
8  };
9  int main()
10 {
11   Base* bp{new Derived()};
12   delete bp;
13 }
```

Polymorphism

Virtual Table

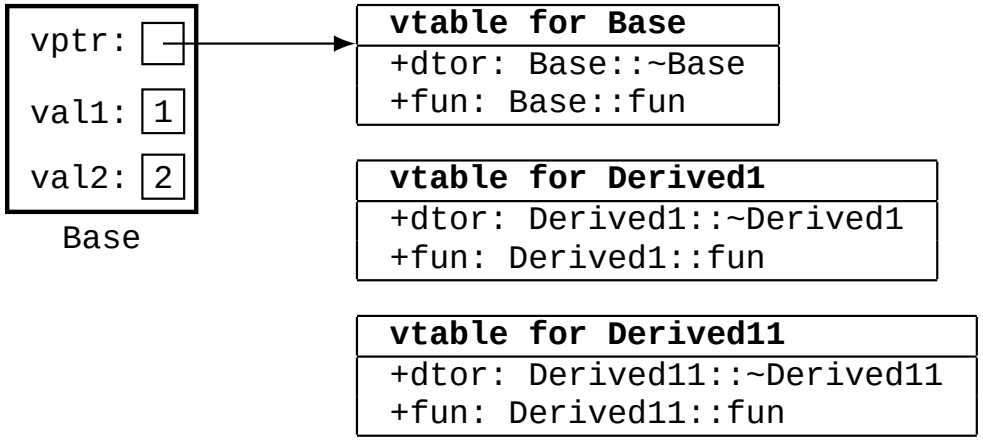
```
struct Base
{
  virtual ~Base();
  virtual void fun();
  int val1{1};
  int val2{2};
};
struct Derived1 : public Base
{
  void fun() override;
  double d{3.4};
};
struct Derived11 : public Derived1
{
  void fun() final;
};
```

```
void Base::fun()
{
  cout << val1 << ' ' << val2;
}

void Derived1::fun()
{
  Base::fun();
  cout << ' ' << d;
}

void Derived11::fun()
{
  cout << "Derived11 ";
  Derived1::fun();
}
```

Polymorphism

Virtual Table

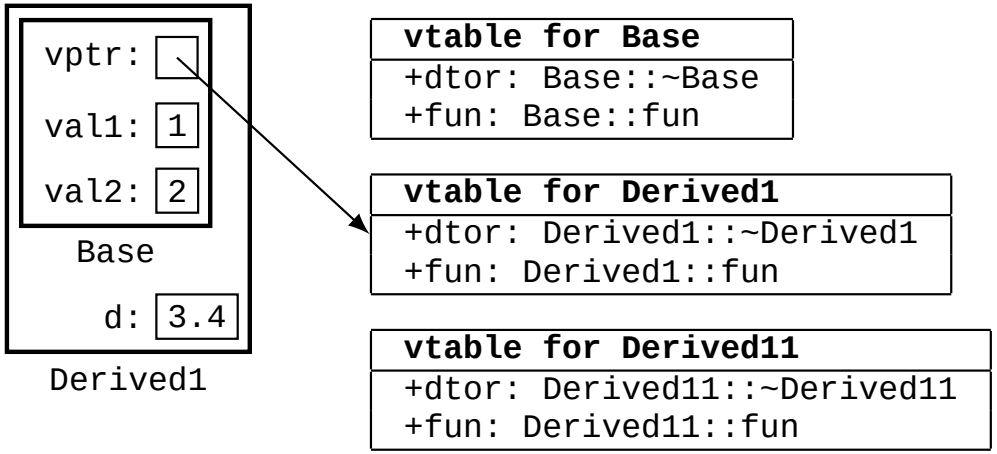
```
Base* bp{new Base{}};
```



Polymorphism

Virtual Table

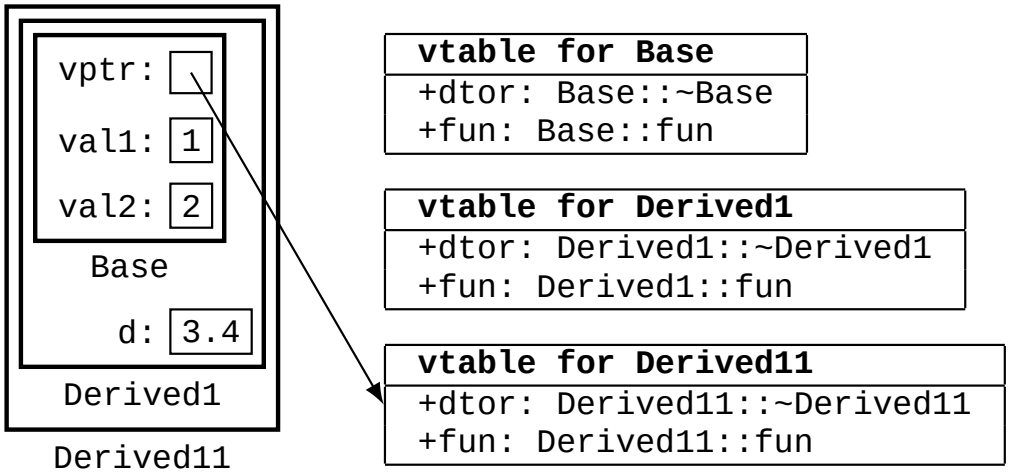
```
Base* bp{new Derived1{}};
```



Polymorphism

Virtual Table

```
Base* bp{new Derived11{}};
```



Polymorphism

Run-time type information (RTTI)

```
struct Base { virtual ~Base() = default; };
struct Derived1 : public Base { };
struct Derived11 : public Derived1 { };
int main()
{
  Base b;
  Derived1 d1, d2;
  Derived11 d11;
  cout << typeid(b).name() << endl;
  cout << typeid(d1).hash_code() << endl;
  cout << (typeid(d1) == typeid(b)) << endl;
  cout << (typeid(d1) == typeid(d2)) << endl;
  cout << (typeid(d1) == typeid(d11)) << endl;
}
```

Polymorphism

Run-time type information (RTTI)

```
struct Base { virtual ~Base() = default; };
struct Derived1 : public Base { };
struct Derived11 : public Derived1 { };
int main()
{
  Base* bp{new Derived1()};
  cout << (dynamic_cast<Base*>(bp) == nullptr) << endl;
  cout << (dynamic_cast<Derived11*>(bp) == nullptr) << endl;
}
```

Polymorphism

Run-time type information (RTTI)

```
struct Base
{
  virtual ~Base() = default;
};
struct Derived1 : public Base
{
  int foo() { return 1; }
};
struct Derived11 : public Derived1 { };
int main()
{
  Base* bp{new Derived1()};
  // won't work, since foo is a non-virtual function in Derived
  cout << bp->foo() << endl;
  // will work, since we converted bp to Derived* which has access to foo
  cout << dynamic_cast<Derived1&>(*bp).foo() << endl;
  // will throw an exception of type std::bad_cast
  cout << dynamic_cast<Derived11&>(*bp).foo() << endl;
}
```

Polymorphism

What will happen? Why?

```
struct Base { virtual ~Base() = default; };
struct Derived1 : public Base { };
struct Derived11 : public Derived1 { };
struct Derived2 : public Base { };
int main()
{
  Base* bp{new Derived1()};
  if (dynamic_cast<Base*>(bp))
    cout << "B ";
  if (dynamic_cast<Derived1*>(bp))
    cout << "D1 ";
  if (dynamic_cast<Derived11*>(bp))
    cout << "D11 ";
  if (dynamic_cast<Derived2*>(bp))
    cout << "D2 ";
}
```

Polymorphism

Slicing

```
struct Base
{
  virtual void print() {cout << x;}
  int x{1};
};
struct Derived : public Base
{
  void print() override {cout << y;}
  int y{2};
};
```

```
void print(Base b)
{
  b.print();
}

int main()
{
  Derived d{};
  print(d);
}
```

www.liu.se

LINKÖPING
UNIVERSITY