CSE 333 Section AB

C++ classes & dynamic memory! (w/ Yifan & Travis)

Logistics

Due TONIGHT:

Homework 2 @ 9 pm

Due Monday:

Exercise 12 @ 11 am

Due Wednesday 10/24:

Exercise 12a @ 11 am

!!!! Midterm next week
Friday November 1st!!!!

C++ continued

C++ Classes
Memory Dynamism

Questions and review

- What do the following modifiers mean?
  - `public`: Member is accessible by anyone
  - `protected`: Member is accessible by this class and any derived classes
  - `private`: Member is only accessible by this class
  - `friend`: Allows access of private/protected members to other functions and/or classes

- What is a struct under this new context?
  - A struct can be thought of as a class where all members are default public instead of default private. In C++, it is also possible to give member functions (such as a constructor) to structs

When we assign a struct variable to another, what happens when the structure contains an array?

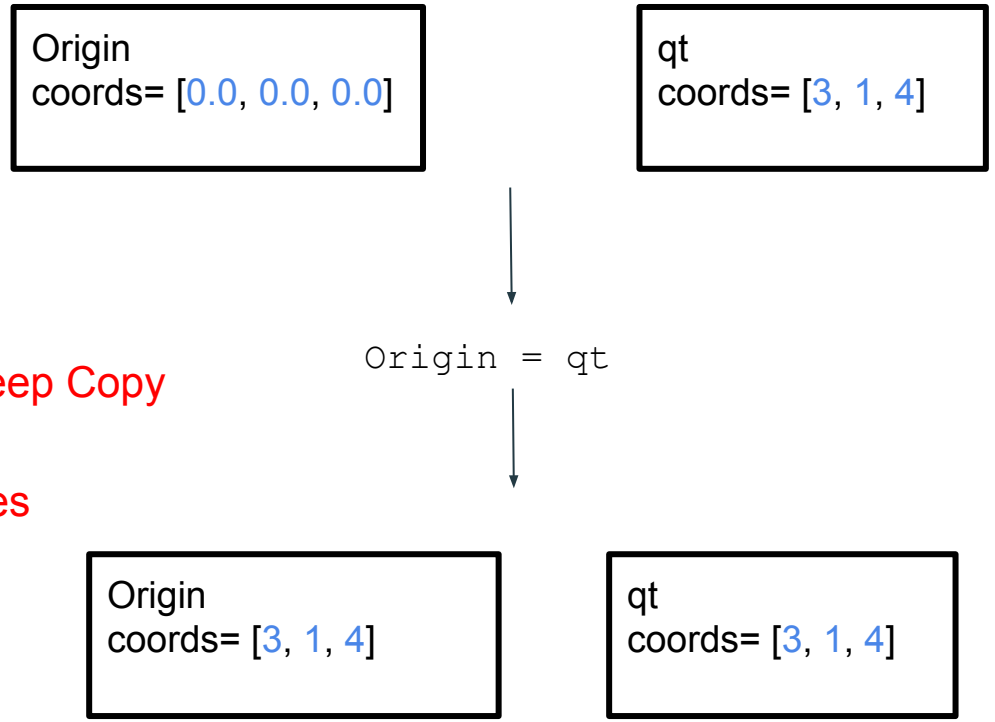
```
struct vector{

  double coords[3];

  int id;

}
```

- Compiler automatically performs Deep Copy for array members
- Same behaviour for arrays in classes



Constructors Revisited

```
class Int {
  public:
    Int() { ival = 17; cout << "default(" << ival << ")" << endl; }
    Int(int n) { ival = n; cout << "ctor(" << ival << ")" << endl; }
    Int(const Int &n) {
      ival = n.ival;
      cout << "cctor(" << ival << ")" << endl;
    }
    ~Int() { cout << "dtor(" << ival << ")" << endl; }
    …
}
```

Constructor (ctor)

Constructor (ctor)

Copy Constructor (cctor)

Destructor (dtor)

- **Copy Constructor (cctor):** Creates a new instance based on another instance (must take a reference!). Invoked when passing/returning a **non-reference** object to/from a function.
- **Constructor (ctor)**: Can define any number as long as they have different parameters. Constructs a new instance of the class.
- **Destructor (dtor):** Cleans up the class instance. Deletes dynamically allocated memory (if any).

Constructors Revisited

```
class Int {
  public:
    Int() { ival = 17; cout << "default(" << ival << ")" << endl; }
    Int(int n) { ival = n; cout << "ctor(" << ival << ")" << endl; }
    Int(const Int &n) {
      ival = n.ival;
      cout << "cctor(" << ival << ")" << endl;
    }
    ~Int() { cout << "dtor(" << ival << ")" << endl; }
    …
}
```

Constructor (ctor)

Constructor (ctor)

Copy Constructor (cctor)

Destructor (dtor)

- **Copy Constructor (cctor):** Creates a new instance based on another instance (must take a reference!). Invoked when passing/returning a **non-reference** object to/from a function.
- **Constructor (ctor)**: Can define any number as long as they have different parameters. Constructs a new instance of the class.
- **Destructor (dtor):** Cleans up the class instance. Deletes dynamically allocated memory (if any).

```
int main(int argc, char **argv) {
  Int p;
  Int q(p);
  Int r(5);
  q.set(p.get()+1);
  return EXIT_SUCCESS;
}
class Int {
 public:
  Int() { ival_ = 17; cout << "default(" << ival_ << ")" << endl; }
  Int(int n) { ival_ = n; cout << "ctor(" << ival_ << ")" << endl; }
  Int(const Int &n) {
    ival_ = n.ival_;
    cout << "cctor(" << ival_ << ")" << endl;
  }
  ~Int() { cout << "dtor(" << ival_ << ")" << endl; }
  int get() const {
    cout << "get(" << ival_ << ")" << endl;
    return ival_;
  }
  void set(int n) {
    ival_ = n;
    cout << "set(" << ival_ << ")" << endl;
  }
 private:
  int ival_;
};
```

```
int main(int argc, char **argv) {
  Int p;
  Int q(p);
  Int r(5);
  q.set(p.get()+1);
  return EXIT_SUCCESS;
}
class Int {
 public:
  Int() { ival_ = 17; cout << "default(" << ival_ << ")" << endl; }
  Int(int n) { ival_ = n; cout << "ctor(" << ival_ << ")" << endl; }
  Int(const Int &n) {
    ival_ = n.ival_;
    cout << "cctor(" << ival_ << ")" << endl;
  }
  ~Int() { cout << "dtor(" << ival_ << ")" << endl; }
  int get() const {
    cout << "get(" << ival_ << ")" << endl;
    return ival_;
  }
  void set(int n) {
    ival_ = n;
    cout << "set(" << ival_ << ")" << endl;
  }
 private:
  int ival_;
};
```

```
1. default(17)
2. cctor(17)
3. ctor(5)
4. get(17)
5. set(18)
```

UwU Looks like we got all the function calls!

Questions and review

```
int main(int argc, char **argv) {
  Int p;
  Int q(p);
  Int r(5);
  q.set(p.get()+1);
  return EXIT_SUCCESS;
}
```

- What is the destruction order?

Destruction order is the reverse of construction order.

```
int main(int argc, char **argv) {
  Int p;
  Int q(p);
  Int r(5);
  q.set(p.get()+1);
  return EXIT_SUCCESS;
}
```

1. default(17)
2. cctor(17)
3. ctor(5)
4. get(17)
5. set(18)
6. dtor(5)
7. dtor(18)
8. dtor(17)

Questions and review

- What happens if you don't define a copy constructor? Or an assignment operator? Or a destructor? Why might this be bad?
- (Hint: What if a member of a class is a pointer to heap-allocated struct?)

In C++, if you don't define any of these, a default one will be synthesized for you.
The default copy constructor does a shallow copy of all fields.
The default assignment operator does a shallow copy of all fields.
The default destructor calls the default destructors of any fields that have them.

- How can you disable the copy constructor/assignment operator/destructor?

Set their prototypes equal to the keyword "delete": ~SomeClass() = delete;

```
class foo {
 public:
  foo()                          { cout << "p"; }                // ctor
  foo(int i)                     { cout << "a"; }                // ctor (1 int)
  foo(int i, int j)              { cout << "h"; }                // ctor (2 ints)
  ~foo()                         { cout << "s"; }                // dtor
};

class bar {
 public:
  bar(): foo_(new foo())         { cout << "g"; }                // ctor
  bar(int i): foo_(new foo(i)) { cout << "p"; }                // ctor (1 int)
  ~bar()                         { cout << "e"; delete foo_; } // dtor
 private:
  foo *foo_;
  foo otherfoo_;
};

class baz {
 public:
  baz(int a,int b,int c) : bar_(a), foo_(b,c)
                                 { cout << "i"; }                // ctor (3 ints)
  ~baz()                         { cout << "n"; }                // dtor
 private:
  foo foo_;
  bar bar_;
};
```

```
int main() {
  baz b(1,2,3);
  return EXIT_SUCCESS;
}
```

Call Stack:

baz(1, 2, 3)

```
class foo {
 public:
  foo()                          { cout << "p"; }                 // ctor
  foo(int i)                     { cout << "a"; }                 // ctor (1 int)
  foo(int i, int j)              { cout << "h"; }                 // ctor (2 ints)
  ~foo()                         { cout << "s"; }                 // dtor
};

class bar {
 public:
  bar(): foo_(new foo())         { cout << "g"; }                 // ctor
  bar(int i): foo_(new foo(i))   { cout << "p"; }                 // ctor (1 int)
  ~bar()                         { cout << "e"; delete foo_; }    // dtor
 private:
  foo *foo_;
  foo otherfoo_;
};

class baz {
 public:
  baz(int a,int b,int c) : bar_(a), foo_(b,c)
                                 { cout << "i"; }                 // ctor (3 ints)
  ~baz()                         { cout << "n"; }                 // dtor
 private:
  foo foo_;
  bar bar_;
};
```

h

```
int main() {
  baz b(1,2,3);
  return EXIT_SUCCESS;
}
```

Call Stack:

baz(1, 2, 3)

foo(2, 3)

```
class foo {
 public:
  foo()                          { cout << "p"; }                 // ctor
  foo(int i)                     { cout << "a"; }                 // ctor (1 int)
  foo(int i, int j)              { cout << "h"; }                 // ctor (2 ints)
  ~foo()                         { cout << "s"; }                 // dtor
};

class bar {
 public:
  bar(): foo_(new foo())         { cout << "g"; }                 // ctor
  bar(int i): foo_(new foo(i))   { cout << "p"; }                 // ctor (1 int)
  ~bar()                         { cout << "e"; delete foo_; }    // dtor
 private:
  foo *foo_;
  foo otherfoo_;
};

class baz {
 public:
  baz(int a,int b,int c) : bar_(a), foo_(b,c)
                                 { cout << "i"; }                 // ctor (3 ints)
  ~baz()                         { cout << "n"; }                 // dtor
 private:
  foo foo_;
  bar bar_;
};
```

h a

```
int main() {
  baz b(1,2,3);
  return EXIT_SUCCESS;
}
```

Call Stack:

baz(1, 2, 3)

bar(1)

foo(1)

```
class foo {
 public:
  foo()                          { cout << "p"; }                 // ctor
  foo(int i)                     { cout << "a"; }                 // ctor (1 int)
  foo(int i, int j)              { cout << "h"; }                 // ctor (2 ints)
  ~foo()                         { cout << "s"; }                 // dtor
};

class bar {
 public:
  bar(): foo_(new foo())         { cout << "g"; }                 // ctor
  bar(int i): foo_(new foo(i))   { cout << "p"; }                 // ctor (1 int)
  ~bar()                         { cout << "e"; delete foo_; }    // dtor
 private:
  foo *foo_;
  foo otherfoo_;
};

class baz {
 public:
  baz(int a,int b,int c) : bar_(a), foo_(b,c)
                                 { cout << "i"; }                 // ctor (3 ints)
  ~baz()                         { cout << "n"; }                 // dtor
 private:
  foo foo_;
  bar bar_;
};
```

h a p

```
int main() {
  baz b(1,2,3);
  return EXIT_SUCCESS;
}
```

Call Stack:

baz(1, 2, 3)

bar(1)

foo()

```
class foo {
 public:
  foo()                       { cout << "p"; }              // ctor
  foo(int i)                  { cout << "a"; }              // ctor (1 int)
  foo(int i, int j)           { cout << "h"; }              // ctor (2 ints)
  ~foo()                      { cout << "s"; }              // dtor
};

class bar {
 public:
  bar(): foo_(new foo())      { cout << "g"; }              // ctor
  bar(int i): foo_(new foo(i)) { cout << "p"; }             // ctor (1 int)
  ~bar()                      { cout << "e"; delete foo_; } // dtor
 private:
  foo *foo_;
  foo otherfoo_;
};

class baz {
 public:
  baz(int a,int b,int c) : bar_(a), foo_(b,c)
                              { cout << "i"; }              // ctor (3 ints)
  ~baz()                      { cout << "n"; }              // dtor
 private:
  foo foo_;
  bar bar_;
};
```

h a p p

```
int main() {
  baz b(1,2,3);
  return EXIT_SUCCESS;
}
```

Call Stack:

baz(1, 2, 3)

bar(1)

```
class foo {
 public:
  foo()                          { cout << "p"; }                 // ctor
  foo(int i)                     { cout << "a"; }                 // ctor (1 int)
  foo(int i, int j)              { cout << "h"; }                 // ctor (2 ints)
  ~foo()                         { cout << "s"; }                 // dtor
};

class bar {
 public:
  bar(): foo_(new foo())         { cout << "g"; }                 // ctor
  bar(int i): foo_(new foo(i))   { cout << "p"; }                 // ctor (1 int)
  ~bar()                         { cout << "e"; delete foo_; }    // dtor
 private:
  foo *foo_;
  foo otherfoo_;
};

class baz {
 public:
  baz(int a,int b,int c) : bar_(a), foo_(b,c)
                                 { cout << "i"; }                 // ctor (3 ints)
  ~baz()                         { cout << "n"; }                 // dtor
 private:
  foo foo_;
  bar bar_;
};
```

h a p p i

```
int main() {
  baz b(1,2,3);
  return EXIT_SUCCESS;
}
```

Call Stack:

baz(1, 2, 3)

```
class foo {
 public:
  foo()                       { cout << "p"; }              // ctor
  foo(int i)                  { cout << "a"; }              // ctor (1 int)
  foo(int i, int j)           { cout << "h"; }              // ctor (2 ints)
  ~foo()                      { cout << "s"; }              // dtor
};

class bar {
 public:
  bar(): foo_(new foo())      { cout << "g"; }              // ctor
  bar(int i): foo_(new foo(i)) { cout << "p"; }             // ctor (1 int)
  ~bar()                      { cout << "e"; delete foo_; } // dtor
 private:
  foo *foo_;
  foo otherfoo_;
};

class baz {
 public:
  baz(int a,int b,int c) : bar_(a), foo_(b,c)
                              { cout << "i"; }              // ctor (3 ints)
  ~baz()                      { cout << "n"; }              // dtor
 private:
  foo foo_;
  bar bar_;
};
```

h a p p i n

```
int main() {
  baz b(1,2,3);
  return EXIT_SUCCESS;
}
```

Call Stack:

~baz()

```
class foo {
 public:
  foo()                          { cout << "p"; }                 // ctor
  foo(int i)                     { cout << "a"; }                 // ctor (1 int)
  foo(int i, int j)              { cout << "h"; }                 // ctor (2 ints)
  ~foo()                         { cout << "s"; }                 // dtor
};

class bar {
 public:
  bar(): foo_(new foo())         { cout << "g"; }                 // ctor
  bar(int i): foo_(new foo(i)) { cout << "p"; }                 // ctor (1 int)
  ~bar()                         { cout << "e"; delete foo_; }  // dtor
 private:
  foo *foo_;
  foo otherfoo_;
};

class baz {
 public:
  baz(int a,int b,int c) : bar_(a), foo_(b,c)
                                 { cout << "i"; }                 // ctor (3 ints)
  ~baz()                         { cout << "n"; }                 // dtor
 private:
  foo foo_;
  bar bar_;
};
```

h a p p i n e

```
int main() {
  baz b(1,2,3);
  return EXIT_SUCCESS;
}
```

Call Stack:

~bar()

```
class foo {
 public:
  foo()                          { cout << "p"; }                // ctor
  foo(int i)                     { cout << "a"; }                // ctor (1 int)
  foo(int i, int j)              { cout << "h"; }                // ctor (2 ints)
  ~foo()                         { cout << "s"; }                // dtor
};

class bar {
 public:
  bar(): foo_(new foo())         { cout << "g"; }                // ctor
  bar(int i): foo_(new foo(i)) { cout << "p"; }                // ctor (1 int)
  ~bar()                         { cout << "e"; delete foo_; } // dtor
 private:
  foo *foo_;
  foo otherfoo_;
};

class baz {
 public:
  baz(int a,int b,int c) : bar_(a), foo_(b,c)
                                 { cout << "i"; }                // ctor (3 ints)
  ~baz()                         { cout << "n"; }                // dtor
 private:
  foo foo_;
  bar bar_;
};
```

h a p p i n e s

```
int main() {
  baz b(1,2,3);
  return EXIT_SUCCESS;
}
```

Call Stack:

~foo()

```
class foo {
 public:
  foo()                       { cout << "p"; }              // ctor
  foo(int i)                  { cout << "a"; }              // ctor (1 int)
  foo(int i, int j)           { cout << "h"; }              // ctor (2 ints)
  ~foo()                      { cout << "s"; }              // dtor
};

class bar {
 public:
  bar(): foo_(new foo())      { cout << "g"; }              // ctor
  bar(int i): foo_(new foo(i)) { cout << "p"; }             // ctor (1 int)
  ~bar()                      { cout << "e"; delete foo_; } // dtor
 private:
  foo *foo_;
  foo otherfoo_;
};

class baz {
 public:
  baz(int a,int b,int c) : bar_(a), foo_(b,c)
                              { cout << "i"; }              // ctor (3 ints)
  ~baz()                      { cout << "n"; }              // dtor
 private:
  foo foo_;
  bar bar_;
};
```

h a p p i n e s s

```
int main() {
  baz b(1,2,3);
  return EXIT_SUCCESS;
}
```

Call Stack:

~foo()

```
class foo {
 public:
  foo()                          { cout << "p"; }                 // ctor
  foo(int i)                     { cout << "a"; }                 // ctor (1 int)
  foo(int i, int j)              { cout << "h"; }                 // ctor (2 ints)
  ~foo()                         { cout << "s"; }                 // dtor
};

class bar {
 public:
  bar(): foo_(new foo())         { cout << "g"; }                 // ctor
  bar(int i): foo_(new foo(i))   { cout << "p"; }                 // ctor (1 int)
  ~bar()                         { cout << "e"; delete foo_; }    // dtor
 private:
  foo *foo_;
  foo otherfoo_;
};

class baz {
 public:
  baz(int a,int b,int c) : bar_(a), foo_(b,c)
                                 { cout << "i"; }                 // ctor (3 ints)
  ~baz()                         { cout << "n"; }                 // dtor
 private:
  foo foo_;
  bar bar_;
};
```

h a p p i n e s s s

```
int main() {
  baz b(1,2,3);
  return EXIT_SUCCESS;
}
```

Call Stack:

~foo()

```
#include <cstdlib>

class Leaky {
 public:
  Leaky() { x_ = new int(5); }
 private:
  int* x_;
};

int main(int argc, char** argv) {
  Leaky** lkyptr = new Leaky*;
  Leaky* lky = new Leaky();
  *lkyptr = lky;
  delete lkyptr;
  return EXIT_SUCCESS;
}
```

How many bytes of memory are leaked by this program?

```
#include <cstdlib>

class Leaky {
 public:
  Leaky() { x_ = new int(5); }
 private:
  int* x_;
};

int main(int argc, char** argv) {
  Leaky** lkyptr = new Leaky*;
  Leaky* lky = new Leaky();
  *lkyptr = lky;
  delete lkyptr;
  return EXIT_SUCCESS;
}
```

How many bytes of memory are leaked by this program?

12 bytes

```
#include <cstdlib>

class Leaky {
 public:
  Leaky() { x_ = new int(5); }
 private:
  int* x_;
};

int main(int argc, char** argv) {
  Leaky** lkyptr = new Leaky*;
  Leaky* lky = new Leaky();
  *lkyptr = lky;
  delete lkyptr;
  return EXIT_SUCCESS;
}
```

How can we fix these memory leaks?

```
#include <cstdlib>
class Leaky {
 public:
  Leaky() { x_ = new int(5); }
  ~Leaky() { delete x_; }  // Delete the allocated int
 private:
  int* x_;
};
int main(int argc, char** argv) {
  Leaky** lkyptr = new Leaky*;
  Leaky* lky = new Leaky();
  *lkyptr = lky;
  delete lkyptr;
  delete lky;  // Delete of lkyptr doesn't delete what lky points to
  return EXIT_SUCCESS;
}
```

How can we fix these memory leaks?

Identify the memory error with the following code. Then fix it!

```
class BadCopy {
 public:
  BadCopy()  { arr_ = new int[5]; }
  ~BadCopy() { delete [] arr_; }
 private:
  int *arr_;
};

int main(int argc, char** argv) {
  BadCopy *bc1 = new BadCopy;
  BadCopy *bc2 = new BadCopy(*bc1);  // BadCopy's cctor

  delete bc1;
  delete bc2;

  return EXIT_SUCCESS;
}
```

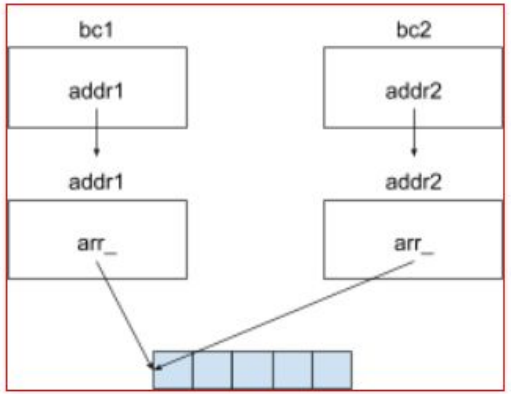
Identify the memory error with the following code. Then fix it!

```
class BadCopy {
 public:
  BadCopy()  { arr_ = new int[5]; }
  ~BadCopy() { delete [] arr_; }
 private:
  int *arr_;
};

int main(int argc, char** argv) {
  BadCopy *bc1 = new BadCopy;
  BadCopy *bc2 = new BadCopy(*bc1);  // BadCopy's cctor

  delete bc1;
  delete bc2;

  return EXIT_SUCCESS;
}
```



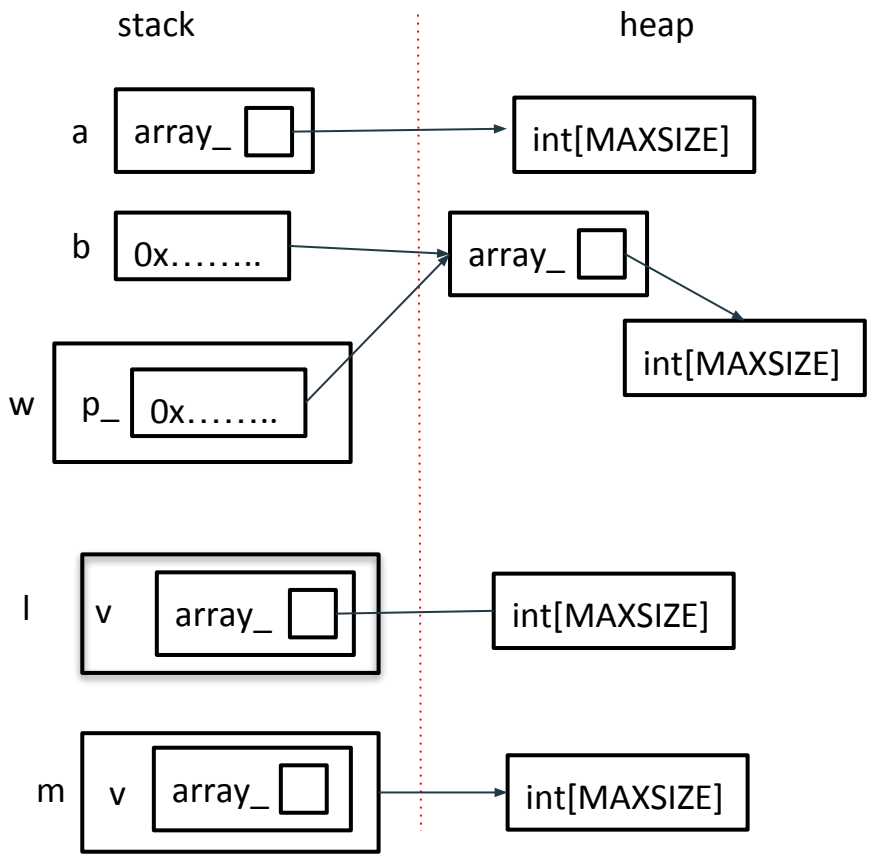
When ~BadCopy() is invoked for bc2, we will try to delete already deleted memory

Question 5

```
int main(int argc, char** argv) {
  IntArrayList a;
  IntArrayList* b = new IntArrayList();
  struct List l { a };
  struct List m { *b };
  Wrap w(b);
  delete b;
  return EXIT_SUCCESS;
}
```

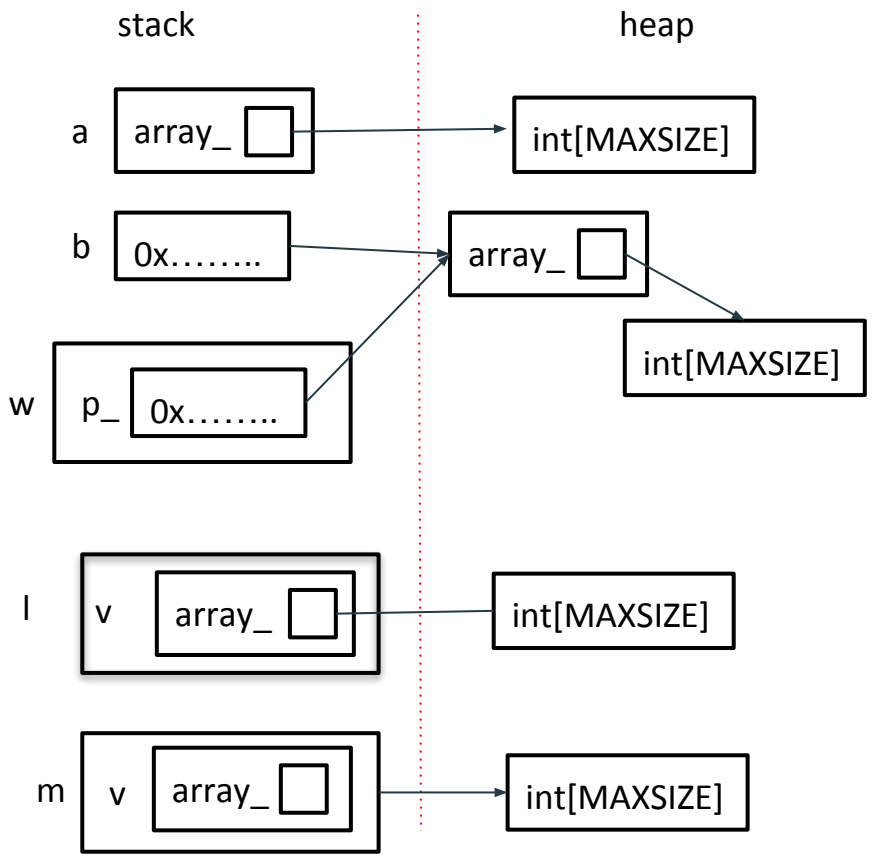
Question 5

```
int main(int argc, char** argv) {
  IntArrayList a;
  IntArrayList* b = new IntArrayList();
  struct List l { a };
  struct List m { *b };
  Wrap w(b);
  delete b;
  return EXIT_SUCCESS;
}
```



Question 5

```
int main(int argc, char** argv) {
  IntArrayList a;
  IntArrayList* b = new IntArrayList();
  struct List l { a };
  struct List m { *b };
  Wrap w(b);
  delete b;
  return EXIT_SUCCESS;
}
```



Question 5

```
int main(int argc, char** argv) {
  IntArrayList a;
  IntArrayList* b = new IntArrayList();
  struct List l { a };
  struct List m { *b };
  Wrap w(b);
  delete b;
  return EXIT_SUCCESS;
}
```

Implement the destructor:

```
IntArrayList::~IntArrayList() { delete[] array_; }
```