CSE 333 Section AA

C++ Classes & Dynamic Memory

Logistics

Due Today:

- Exercise 11 @ 11am
- Homework 2 @ 9 pm

Next Week:

- Exercise 12
- Exercise 12a
- Midterm Exam

Access Specifiers

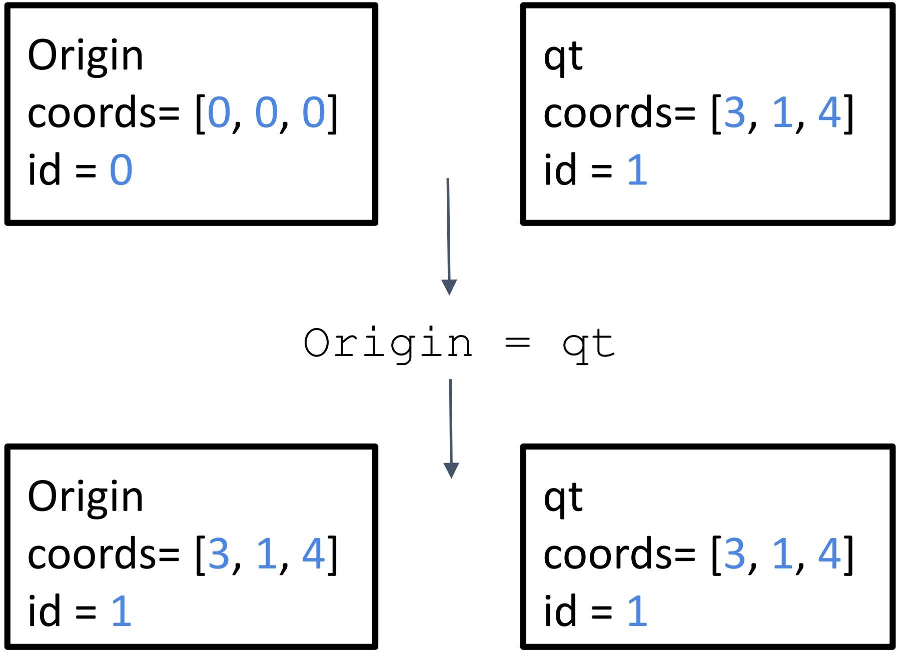
- What do the following modifiers mean?
  - public: Member is accessible by anyone
  - protected: Member is accessible by this class and any derived classes
  - private: Member is only accessible by this class
  - friend: Allows access of private/protected members to other functions and/or classes

- 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



Origin
coords= [0, 0, 0]
id = 0

qt
coords= [3, 1, 4]
id = 1

```
Origin = qt
```

Origin
coords= [3, 1, 4]
id = 1

qt
coords= [3, 1, 4]
id = 1

Constructors Revisited

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

- **Constructor (ctor)**: Can define any number as long as they have different parameters. Constructs a new instance of the class.
- **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.
- **Destructor (dtor):** Cleans up the class instance. Deletes dynamically allocated memory (if any).

- 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?)

A default one will be synthesized for you.

The default copy constructor copies of all fields.

The default assignment operator copies of all fields.

The default destructor calls the destructors of any fields that have them.

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

```
SomeClass(const SomeClass&) = delete;
```

Worksheet Questions

explicit

```
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)

Int p = 5;

Int p = Int(5);

```
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_;
};
```

happin

```
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_;
};
```

happine

```
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_;
};
```

happines

```
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_;
};
```

happiness

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

Call Stack:

~foo()

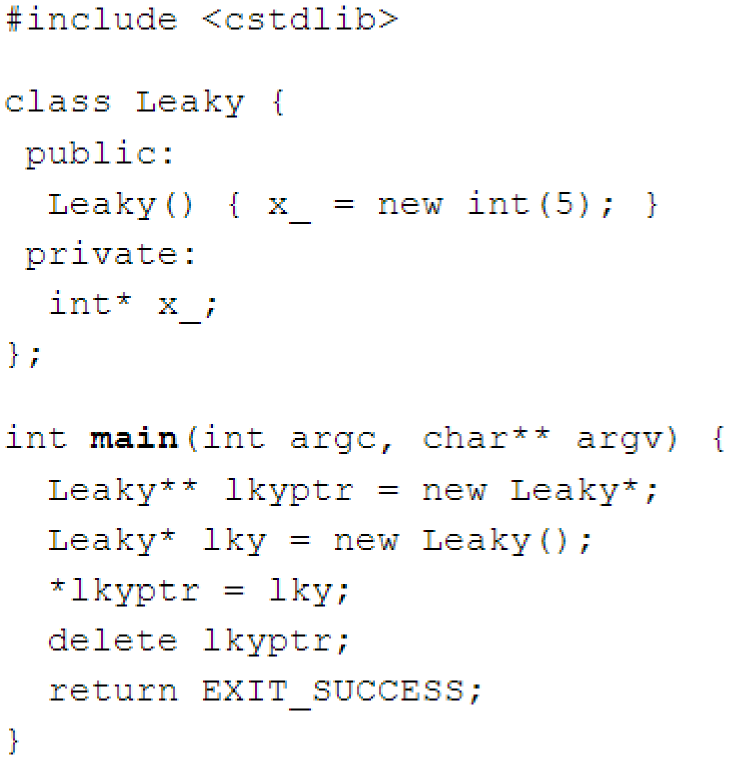
```
class foo {
 public:
  foo()                       { cout << "p"; }            // ctor
  foo(int i)                  { cout << "a"; }            // ctor (1 int)
  foo(int i, int j)           { cout << "h"; }            // ctor (2 ints)
  ~foo()                      { cout << "s"; }            //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

happinesss

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

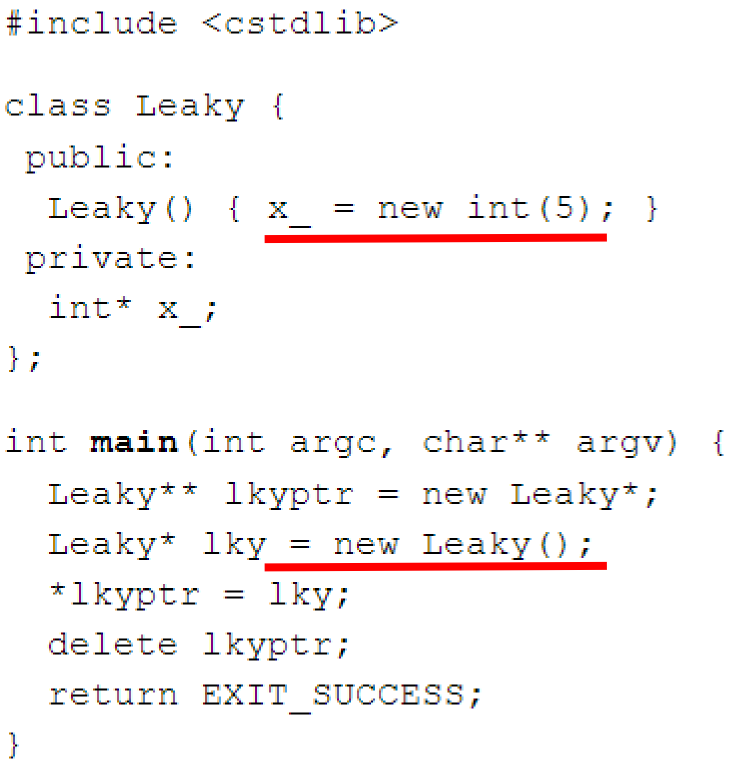
Call Stack:

~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;
}
```

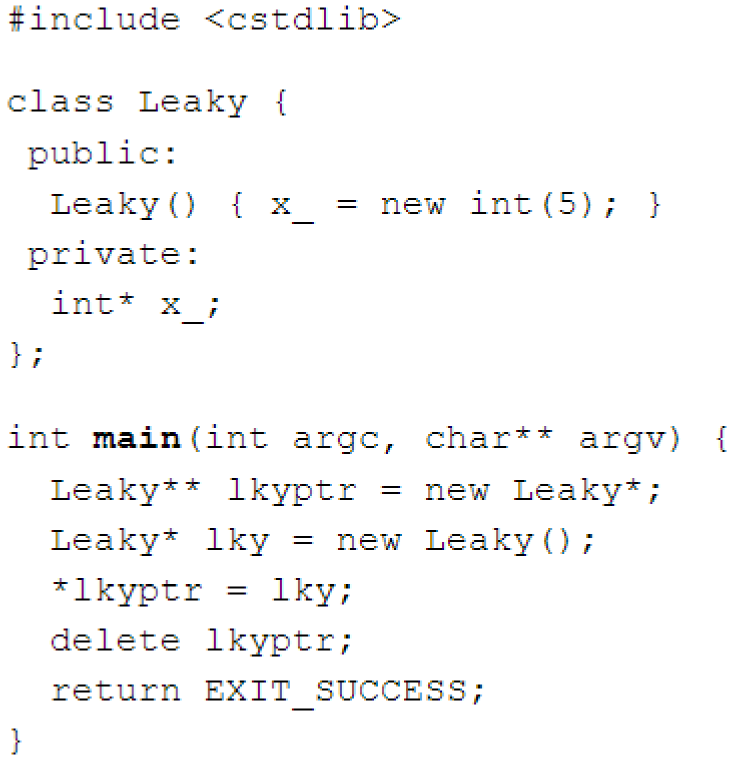
How many bytes of memory are leaked by this program?



```
#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;
}
```

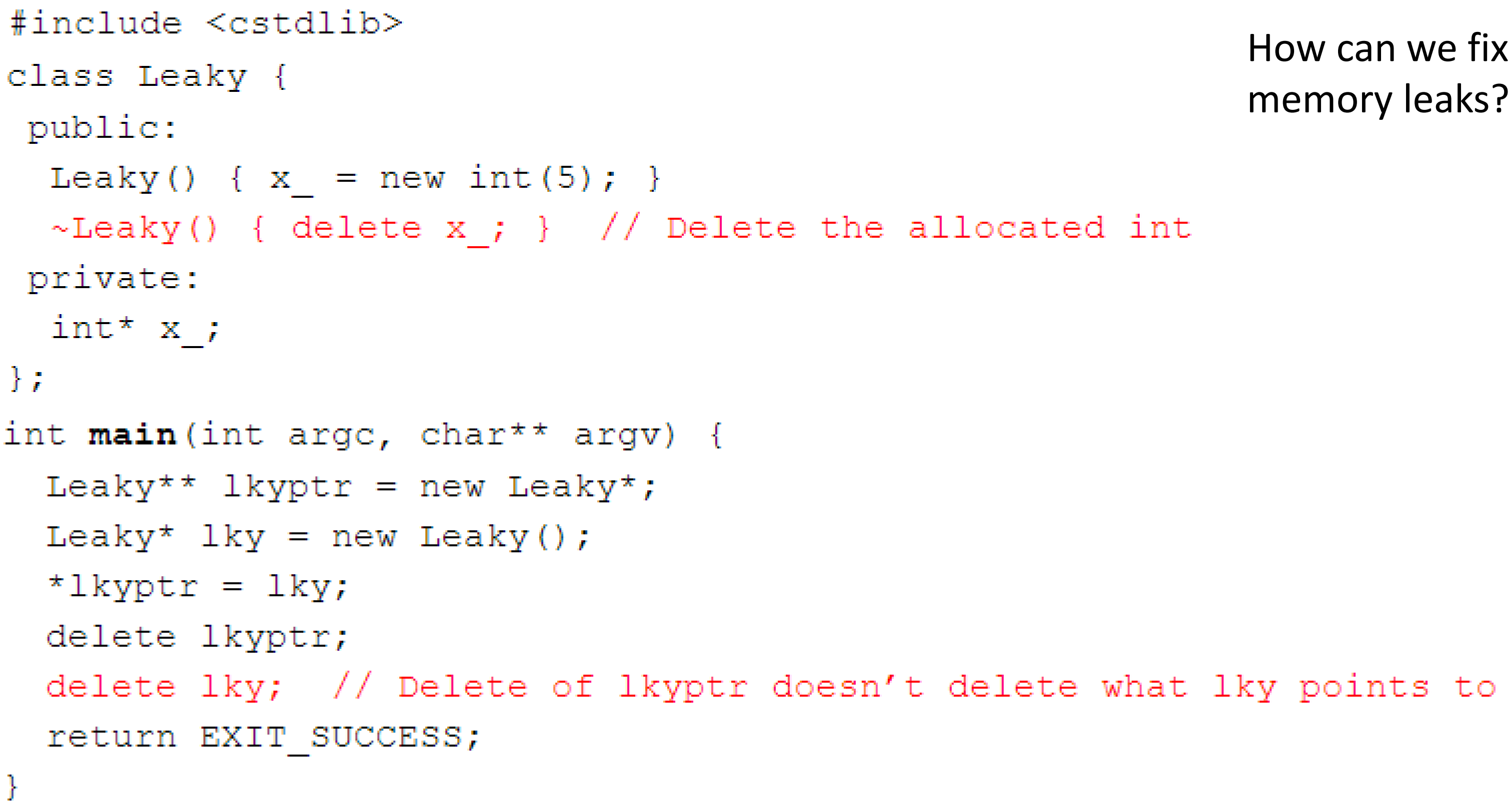
How many bytes of memory are leaked by this program?

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;
}
```

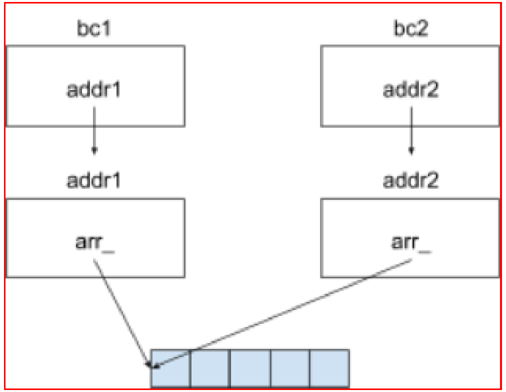
How can we fix these memory leaks?

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