UNIVERSITY OF OREGON

CIS 330:

```
   __  __      _                           __   ______    __ ______
  / / / /___  (_)  __   ____ _____  ____/ /  / ____/  _/_/ ____/__    __
 / / / / __ \/ / |/_/  / __ `/ __ \/ __  /  / /     _/_// /  __/ /___/ /_
/ /_/ / / / / />  <   / /_/ / / / / /_/ /  / /___ _/_/ / /__/_  __/_  __/
\____/_/ /_/_/_/|_|   \__,_/_/ /_/\__,_/   \____//_/   \____//_/   /_/
```

Lecture 20: templates

June 1, 2018

Hank Childs, University of Oregon

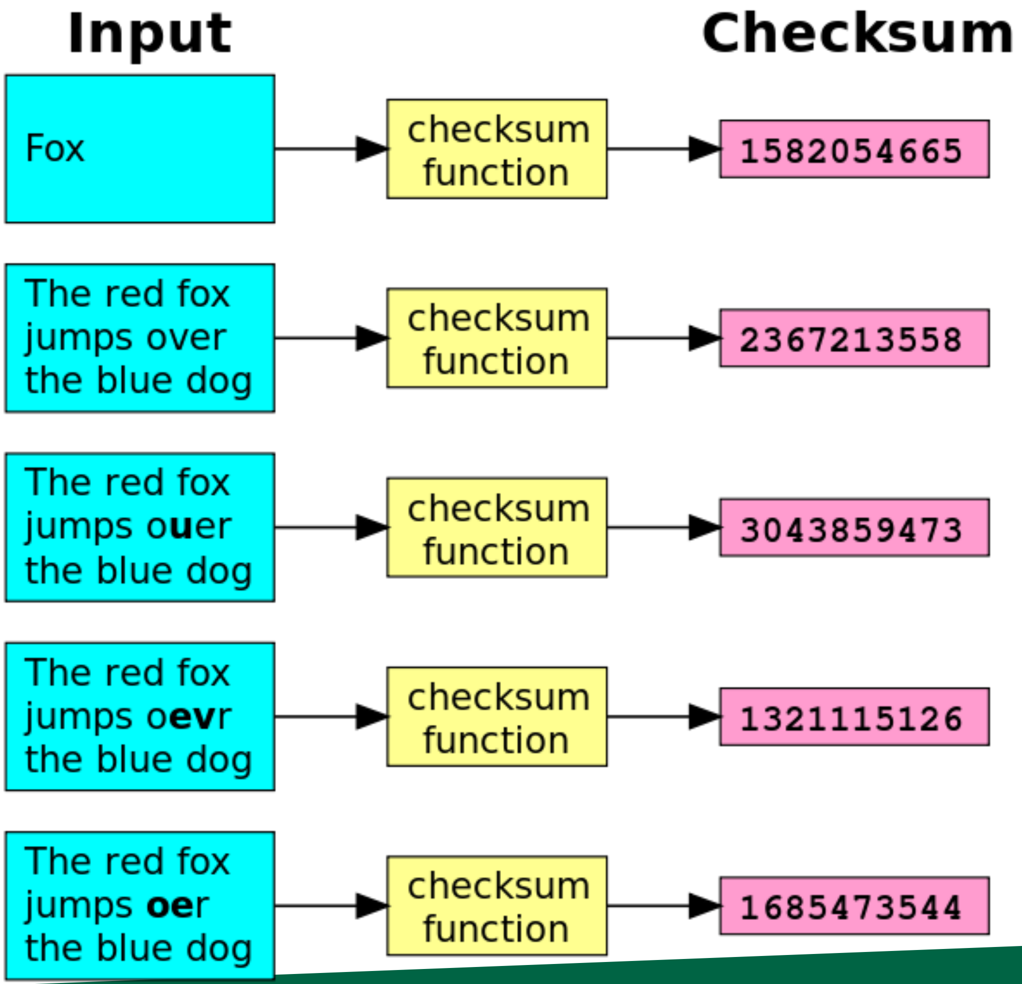
Announcements

- 3T: due in 2 hours!
  - No late work accepted
- 3H, 4B, and 4C: assigned Saturday AM
  - "Due" June 9th
  - But full credit will be awarded until 6am June 14th
- IMPORTANT: all work must be submitted by 6am Weds June 14th
  - No work accepted starting at 601am June 14th

Stress Test Project (3H)

- We will have ~60 stress tests
- We can't check in 60 baseline images and difference them all
  - Will slow ix to a halt
- Solution:
  - We commit "essence of the solution"
  - We also complement that all images posted if needed.

Checksums



Most useful when input is very large and checksum is very small

From Wikipedia

Our “checksum”

- Three integers:
  - Sum of red channel
  - Sum of green channel
  - Sum of blue channel
- When you create a stress test, you register these three integers
- When you test against others stress tests, you compare against their integers
  - If they match, you got it right

This will be done with a derived type of Sink.

Should Checksums Match?

- On ix, everything should match
- On different architectures, floating point math won’t match
- Blender: has floating point math
- → no blender

Assignment:
Make your 3E project run memory error and leak free

4E

Tasks:

1) start with your 3E source code. (Don't use 3F, since it has exceptions and

ed to use ix to do this ... esp.

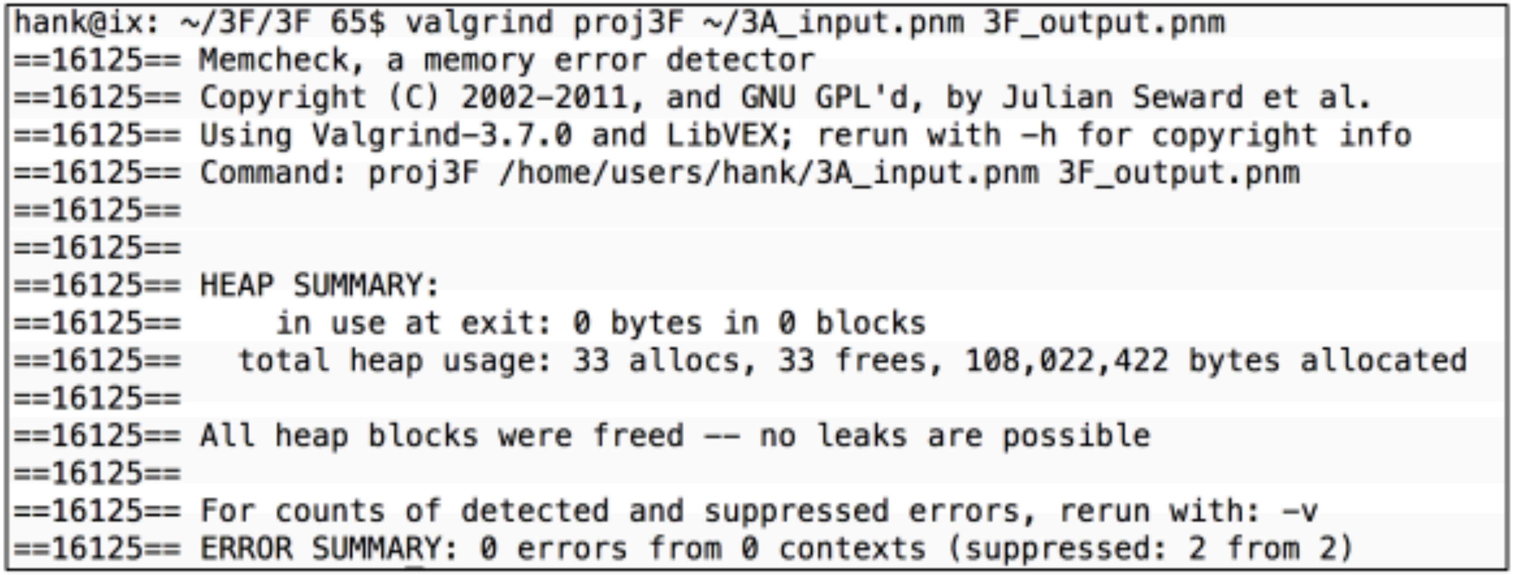
clares your program leak free

Notes:

1) if I had memory errors, they would have occurred after "Command: proj3F" and before "HEAP SUMMARY". None there, so I'm OK.
   a. Don't forget to use your 3E code ... not 3F code.
2) Re memory leaks: "All heap blocks were freed" is the magic statement ... that means no memory leaks.

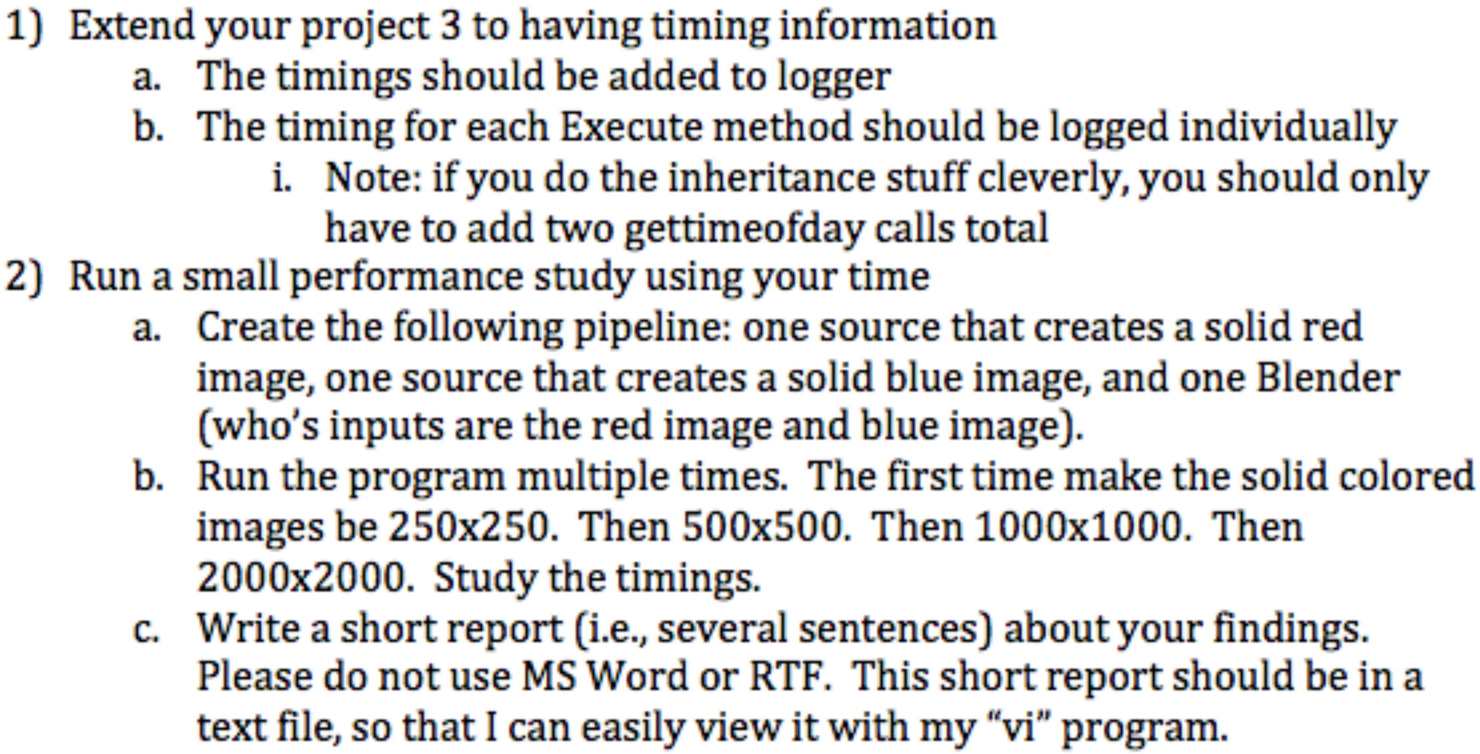
Submit:

1) a screenshot of the valgrind output (see mine below)
2) your source code



```
hank@ix: ~/3F/3F 65$ valgrind proj3F ~/3A_input.pnm 3F_output.pnm
==16125== Memcheck, a memory error detector
==16125== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==16125== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==16125== Command: proj3F /home/users/hank/3A_input.pnm 3F_output.pnm
==16125==
==16125==
==16125== HEAP SUMMARY:
==16125==     in use at exit: 0 bytes in 0 blocks
==16125==   total heap usage: 33 allocs, 33 frees, 108,022,422 bytes allocated
==16125==
==16125== All heap blocks were freed -- no leaks are possible
==16125==
==16125== For counts of detected and suppressed errors, rerun with: -v
==16125== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 2 from 2)
```

4C



1) Extend your project 3 to having timing information
    a. The timings should be added to logger
    b. The timing for each Execute method should be logged individually
        i. Note: if you do the inheritance stuff cleverly, you should only have to add two gettimeofday calls total
2) Run a small performance study using your time
    a. Create the following pipeline: one source that creates a solid red image, one source that creates a solid blue image, and one Blender (who's inputs are the red image and blue image).
    b. Run the program multiple times. The first time make the solid colored images be 250x250. Then 500x500. Then 1000x1000. Then 2000x2000. Study the timings.
    c. Write a short report (i.e., several sentences) about your findings. Please do not use MS Word or RTF. This short report should be in a text file, so that I can easily view it with my "vi" program.

The plan

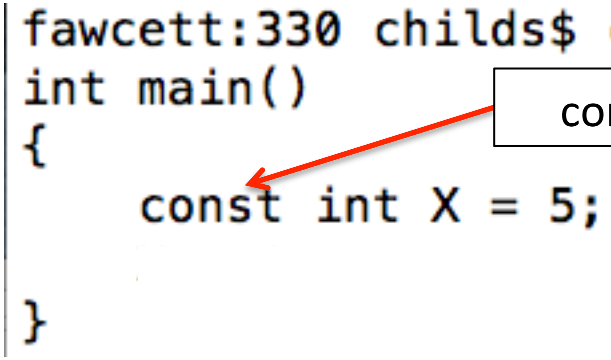
- Monday June 5: No class
- Weds, June 7: No class
  - Remember: you will tell me about late passes on the final
- Fri, June 9: Finals Review
  - This room, this time

Review

const

- const:
  - is a keyword in C and C++
  - qualifies variables
  - is a mechanism for preventing write access to variables

const example



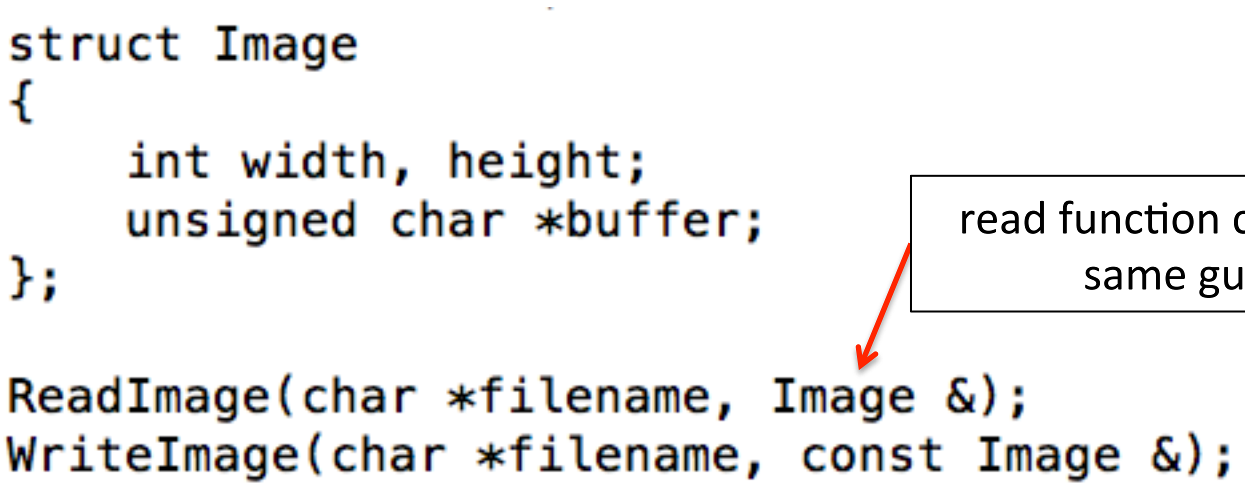
```
fawcett:330 childs$ cat const1.C
int main()
{
    const int X = 5;

}
```

const keyword modifies int

The compiler enforces const … just like public/ private access controls

const arguments to functions

- Functions can use const to guarantee to the calling function that they won’t modify the arguments passed in.



```
struct Image
{
    int width, height;
    unsigned char *buffer;
};

ReadImage(char *filename, Image &);
WriteImage(char *filename, const Image &);
```

read function can’t make the same guarantee

guarantees function won’t modify the Image

const pointers

- Assume a pointer named “P”
- Two distinct ideas:
  - P points to something that is constant
    - P may change, but you cannot modify what it points to via P
  - P must always point to the same thing, but the thing P points to may change.

const pointe

- Assume a pointer named "P"
- Two distinct ideas:
  - P points to something that is constant
    - P may change, but you cannot modify what it points to via P
  - P must always point to the same thing, but the thing P points to may change.

onst pointer

```
int X = 4;
int *P = &X;
```

Idea #1:

violates const:

“*P = 3;”

OK:

“int Y = 5; P = &Y;”

pointer can change, but you can’t modify the thing it points to

Idea #2:

violates const:

“int Y = 5; P = &Y;”

OK:

“*P = 3;”

pointer can’t change, but you can modify the thing it points to

const p

int X

int *P

Idea #3:

violates const:

“*P = 3;”

“int Y = 5; P = &Y;”

OK:

none

pointer can’t change, and you can’t modify the thing it points to

const pointers

int X = 4;

int *P = &X;

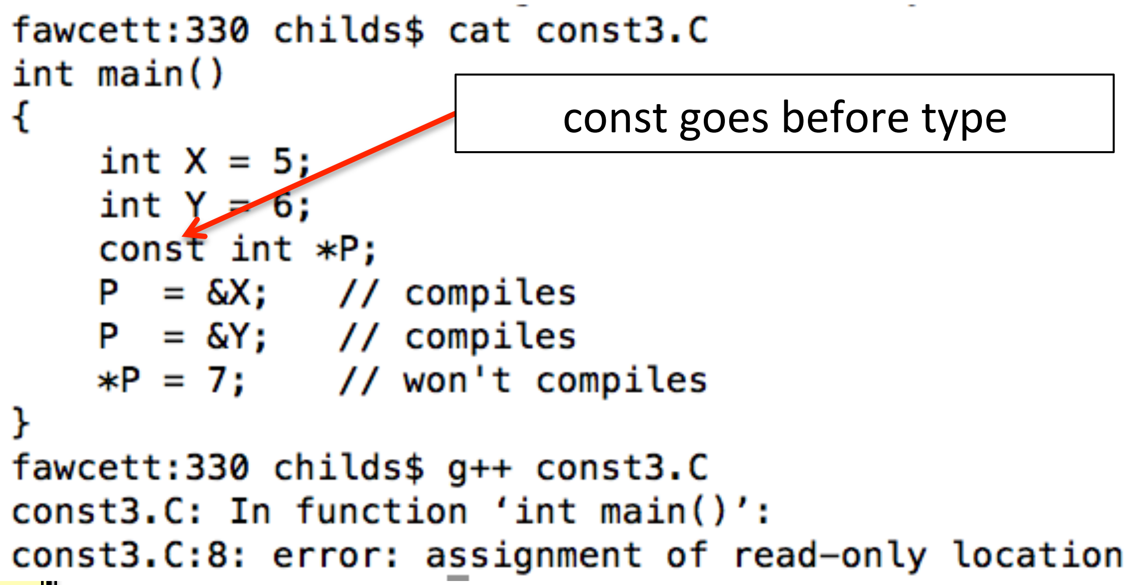
Idea #1:

violates const:

"*P = 3;"

OK:

"int Y = 5; P = &Y;"



```
fawcett:330 childs$ cat const3.C
int main()
{
        int X = 5;
        int Y = 6;
        const int *P;
        P   = &X;    // compiles
        P   = &Y;    // compiles
        *P  = 7;     // won't compiles
}
fawcett:330 childs$ g++ const3.C
const3.C: In function ‘int main()’:
const3.C:8: error: assignment of read-only location
```

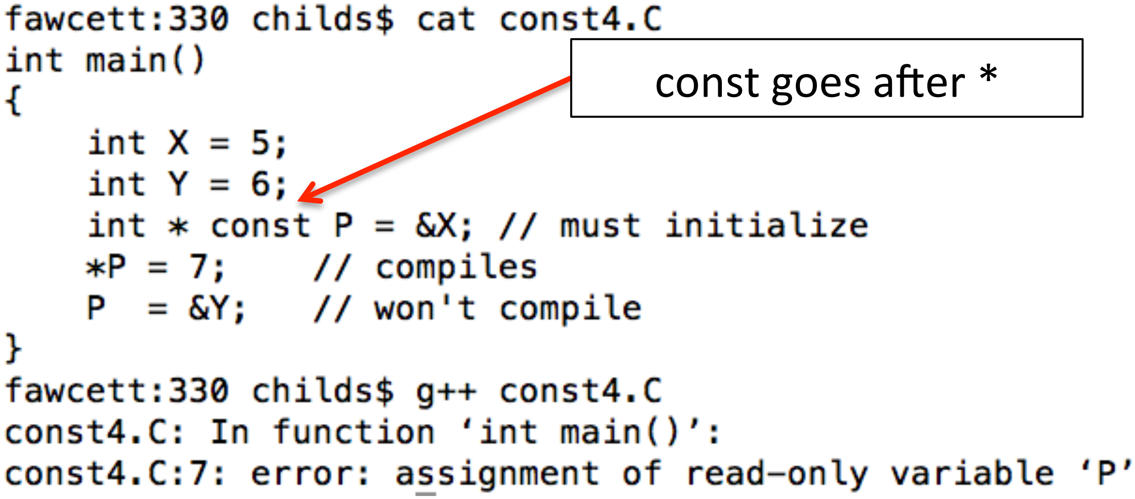
const goes before type

pointer can change, but you can't modify the thing it points to

const pointers

int X = 4;

int *P = &X;



```
fawcett:330 childs$ cat const4.C
int main()
{
    int X = 5;
    int Y = 6;
    int * const P = &X; // must initialize
    *P = 7;     // compiles
    P  = &Y;    // won't compile
}
fawcett:330 childs$ g++ const4.C
const4.C: In function ‘int main()’:
const4.C:7: error: assignment of read-only variable ‘P’
```

const goes after *

Idea #2:

violates const:

“int Y = 5; P = &Y;”

OK:

“*P = 3;”

pointer can’t change, but you can modify the thing it points to

const pointers

```
int X = 4;
int *P = &X;
```

<u>Idea #3:</u>

violates const:

“*P = 3;”

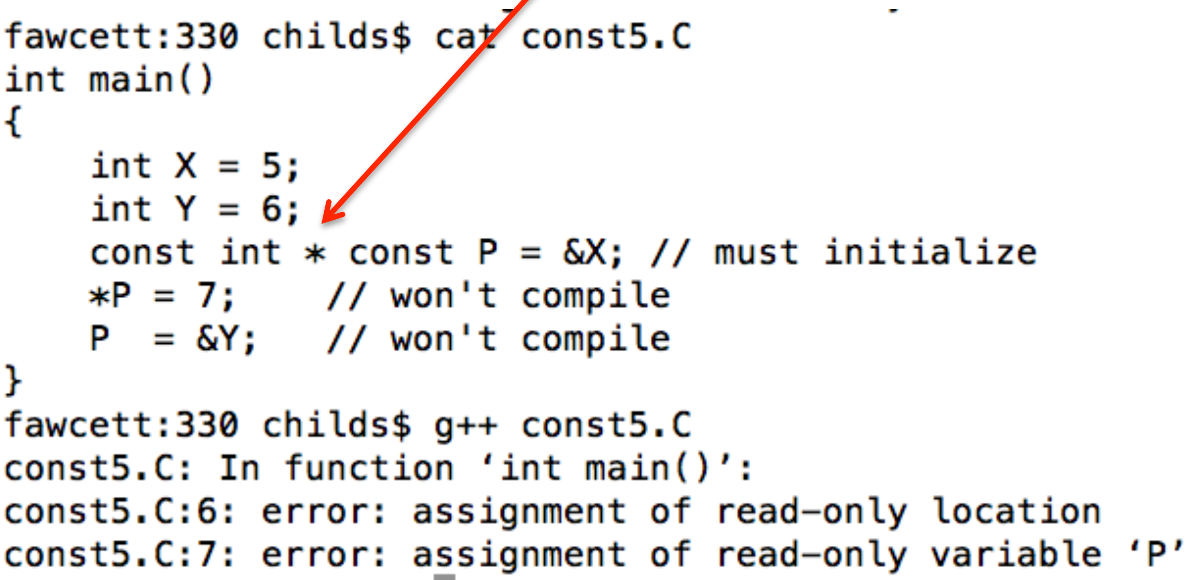
“int Y = 5; P = &Y;”

OK:

none

pointer <u>can’t</u> change, and you <u>can’t</u> modify the thing it points to

const in both places

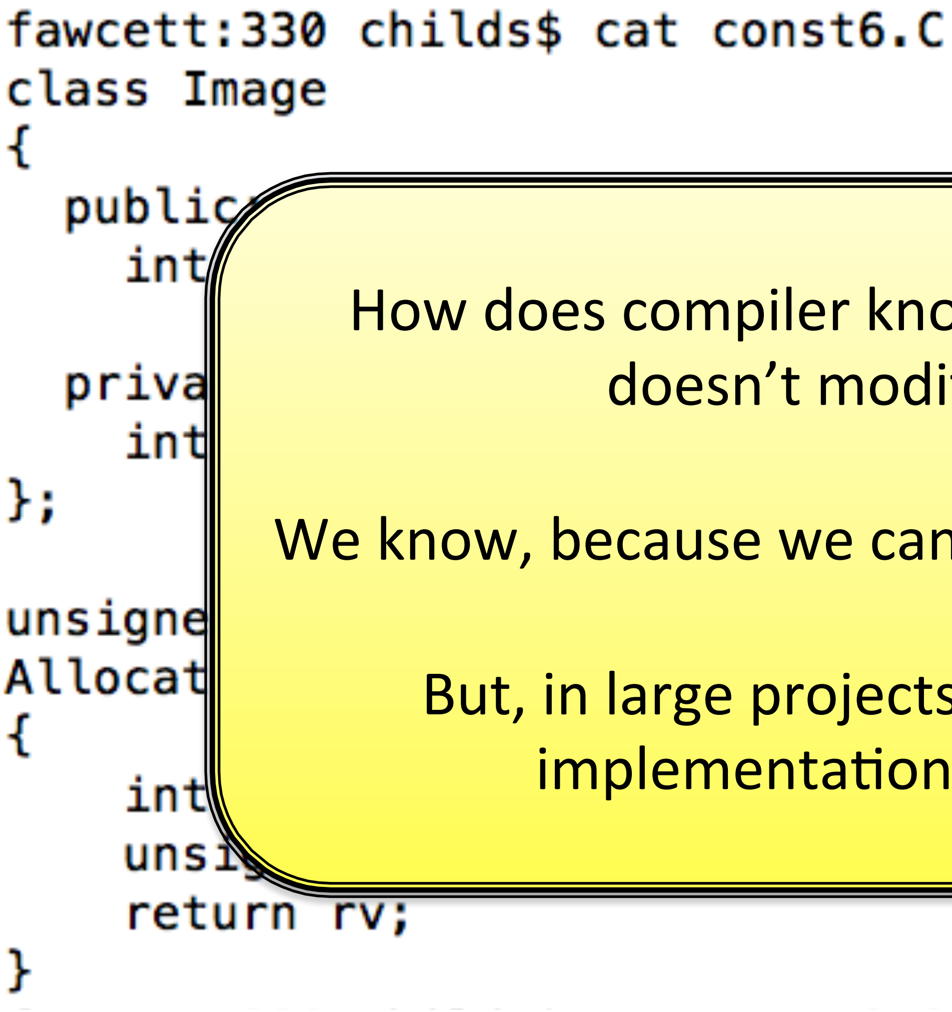


```
fawcett:330 childs$ cat const5.C
int main()
{
    int X = 5;
    int Y = 6;
    const int * const P = &X; // must initialize
    *P = 7;     // won't compile
    P  = &Y;    // won't compile
}
fawcett:330 childs$ g++ const5.C
const5.C: In function ‘int main()’:
const5.C:6: error: assignment of read-only location
const5.C:7: error: assignment of read-only variable ‘P’
```

const usage

- class Image;
- const Image *ptr;
  - – Used a lot: offering the guarantee that the function won't change the Image ptr points to
- Image * const ptr;
  - – Helps with efficiency. Rarely need to worry about this.
- const Image * const ptr;
  - – Interview question!!

Very common issue with const and objects



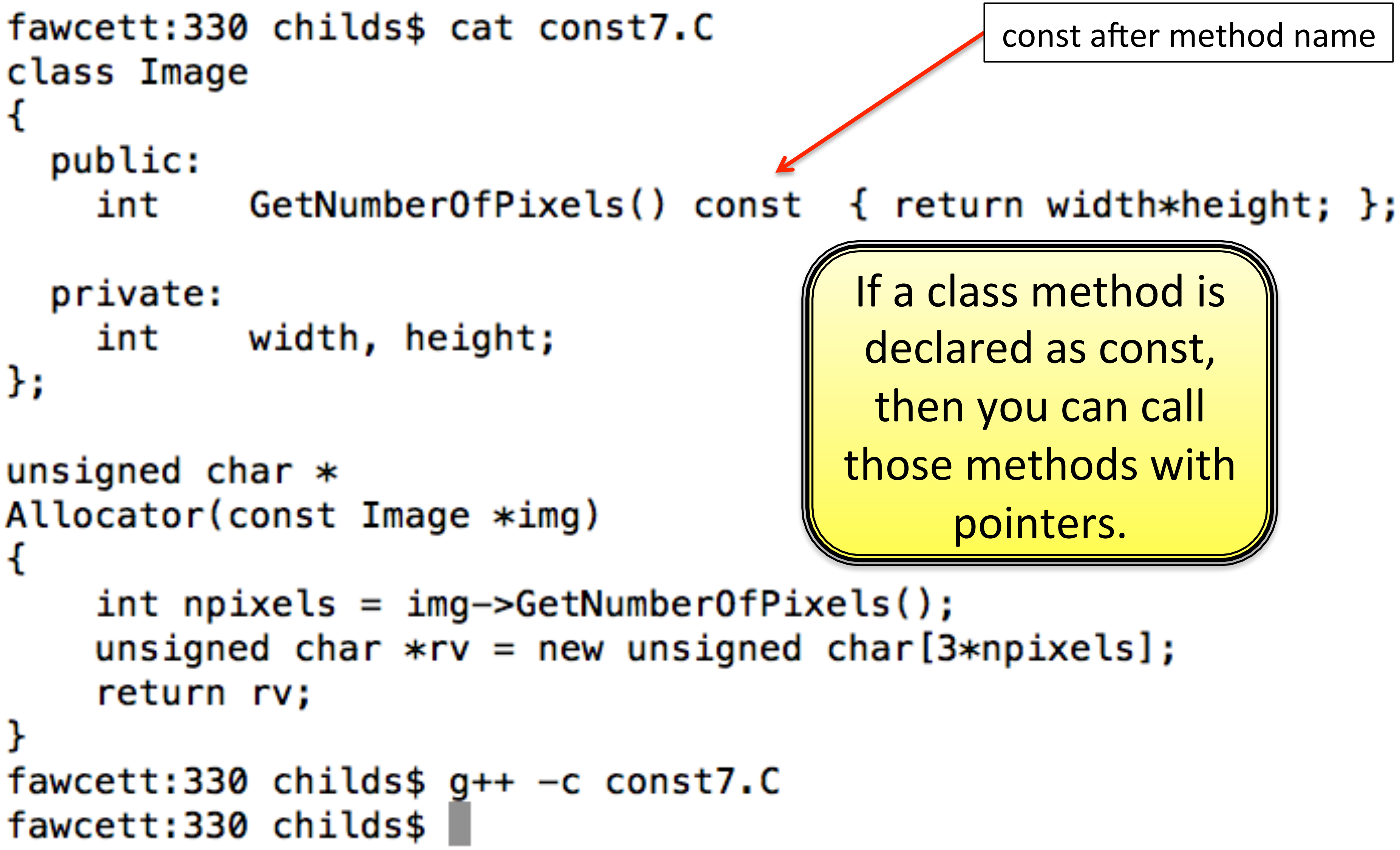
```
fawcett:330 childs$ cat const6.C
class Image
{
   public
      int
      
   priva
      int
};

unsigne
Allocat
{
      int
      unsi
      return rv;
}
```

How does compiler know GetNumberOfPixels doesn't modify an Image?

We know, because we can see the implementation.

But, in large projects, compiler can't see implementation for everything.

const functions with objects



```
fawcett:330 childs$ cat const7.C
class Image
{
  public:
    int    GetNumberOfPixels() const  { return width*height; };

  private:
    int    width, height;
};

unsigned char *
Allocator(const Image *img)
{
    int npixels = img->GetNumberOfPixels();
    unsigned char *rv = new unsigned char[3*npixels];
    return rv;
}
fawcett:330 childs$ g++ -c const7.C
fawcett:330 childs$
```

const after method name

If a class method is declared as const, then you can call those methods with pointers.

mutable

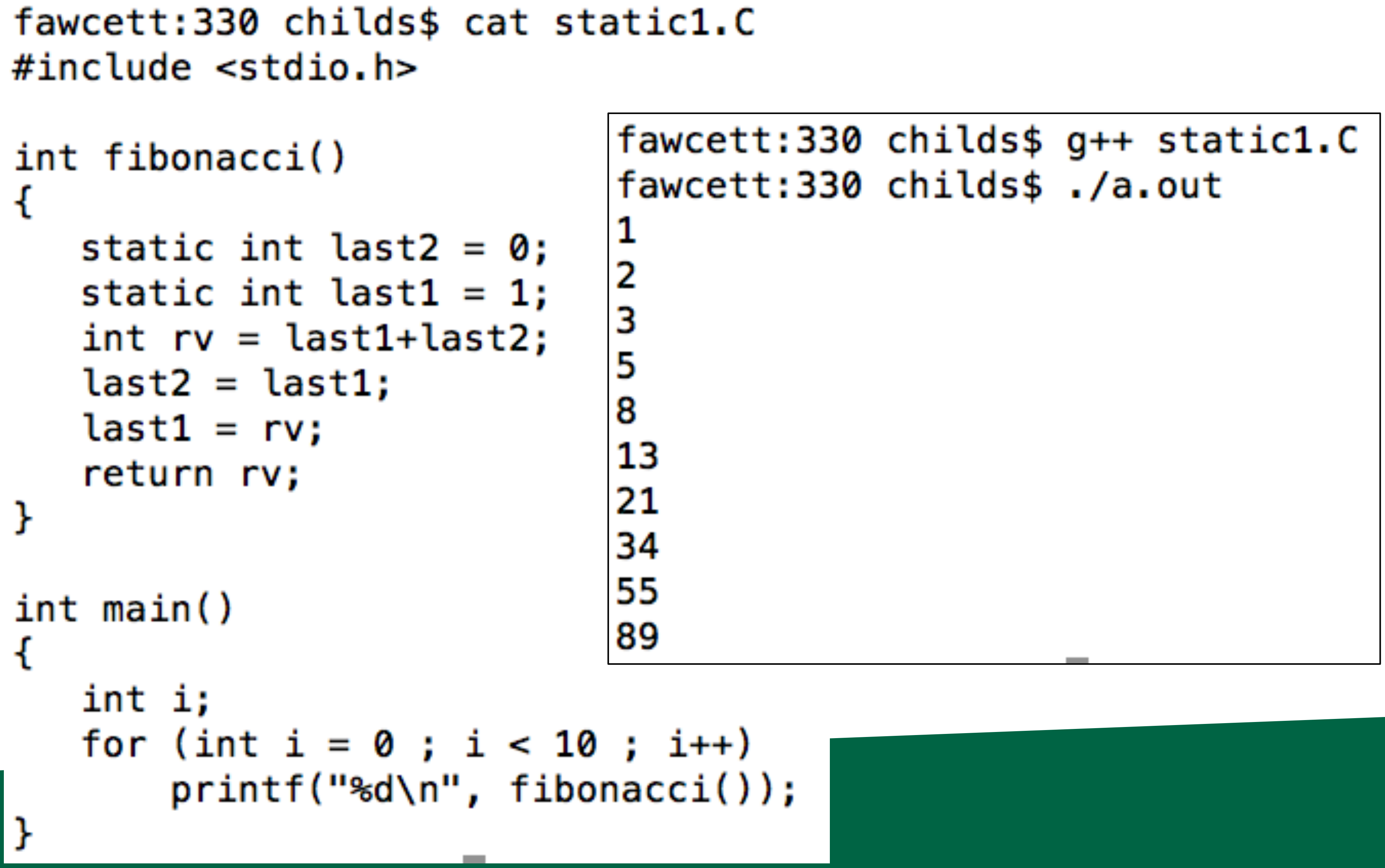
- mutable: special keyword for modifying data members of a class
  - If a data member is mutable, then it can be modified in a const method of the class.
  - Comes up rarely in practice.

static

- static memory: third kind of memory allocation
  - reserved at compile time
- contrasts with dynamic (heap) and automatic (stack) memory allocations
- accomplished via keyword that modifies variables

There are three distinct usages of statics

static usage #1: persistency within a function

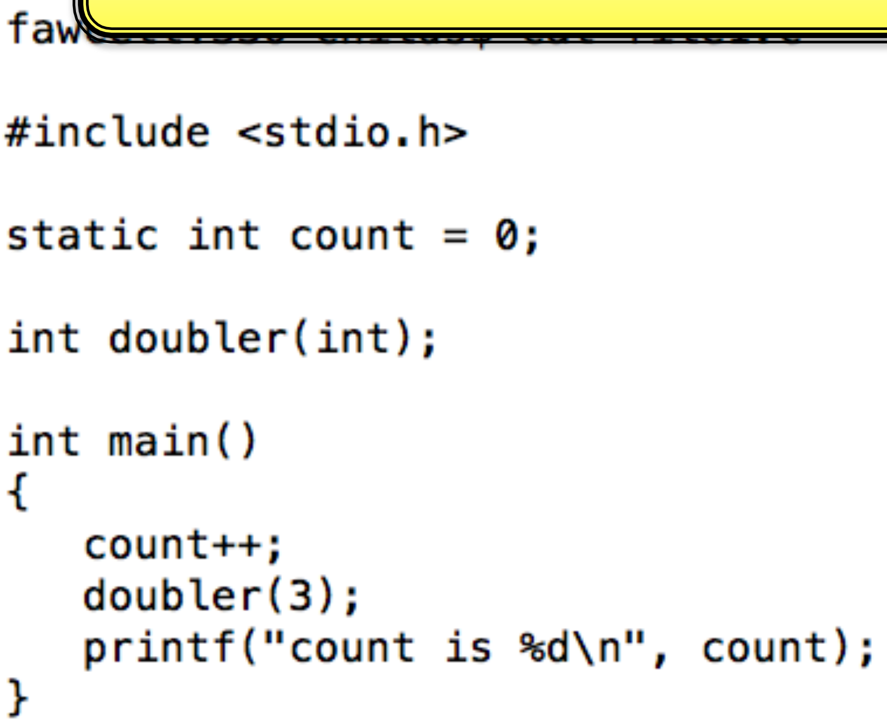


```
fawcett:330 childs$ cat static1.C
#include <stdio.h>

int fibonacci()
{
   static int last2 = 0;
   static int last1 = 1;
   int rv = last1+last2;
   last2 = last1;
   last1 = rv;
   return rv;
}

int main()
{
   int i;
   for (int i = 0 ; i < 10 ; i++)
       printf("%d\n", fibonacci());
}
```

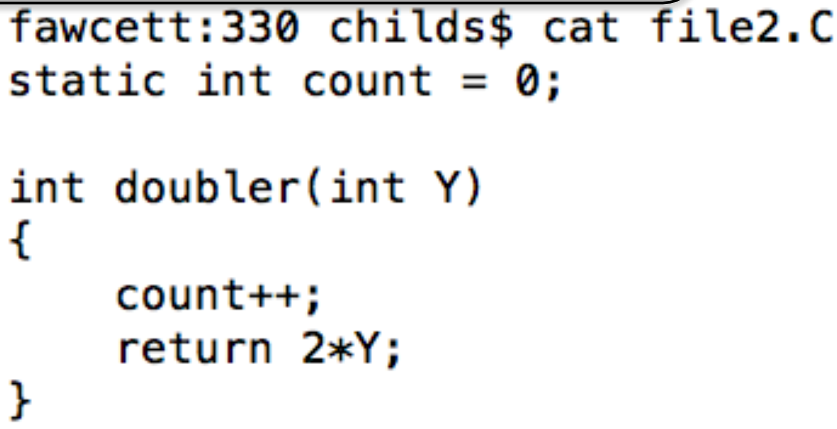
```
fawcett:330 childs$ g++ static1.C
fawcett:330 childs$ ./a.out
1
2
3
5
8
13
21
34
55
89
```

static usage #2: making global variables be local to a file

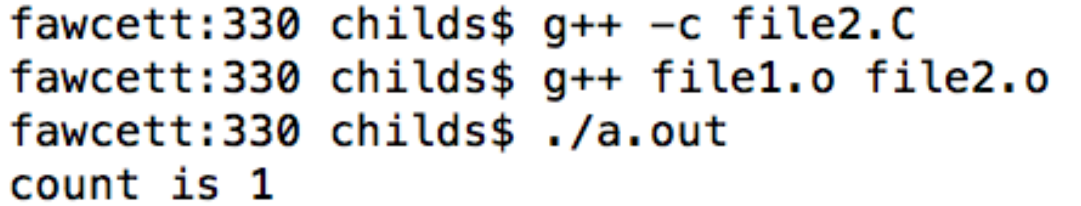
I have no idea why the static keyword is used in this way.



```
fawcett:330 childs$ cat file1.C

#include <stdio.h>

static int count = 0;

int doubler(int);

int main()
{
    count++;
    doubler(3);
    printf("count is %d\n", count);
}
```

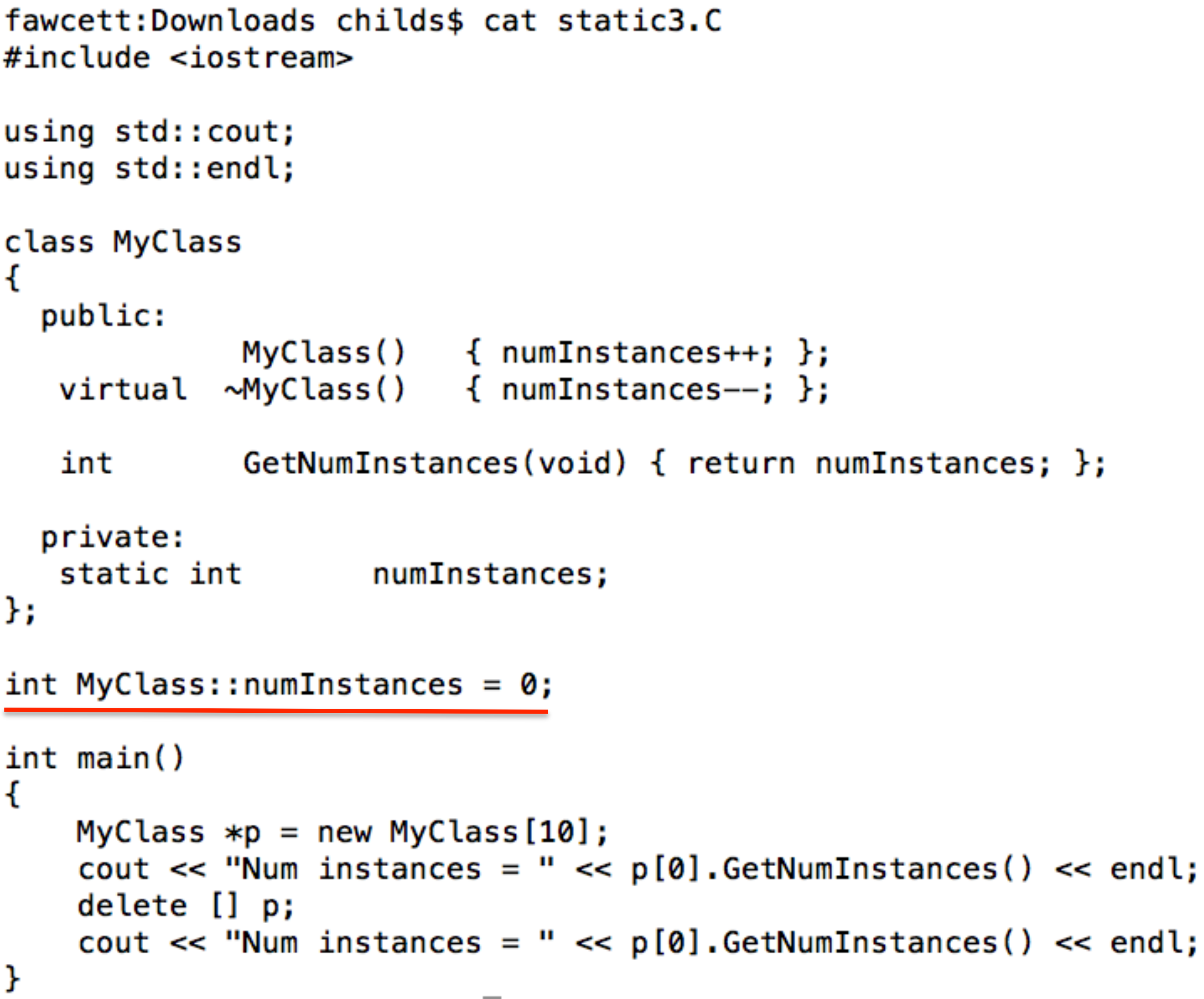


```
fawcett:330 childs$ cat file2.C
static int count = 0;

int doubler(int Y)
{
    count++;
    return 2*Y;
}
```



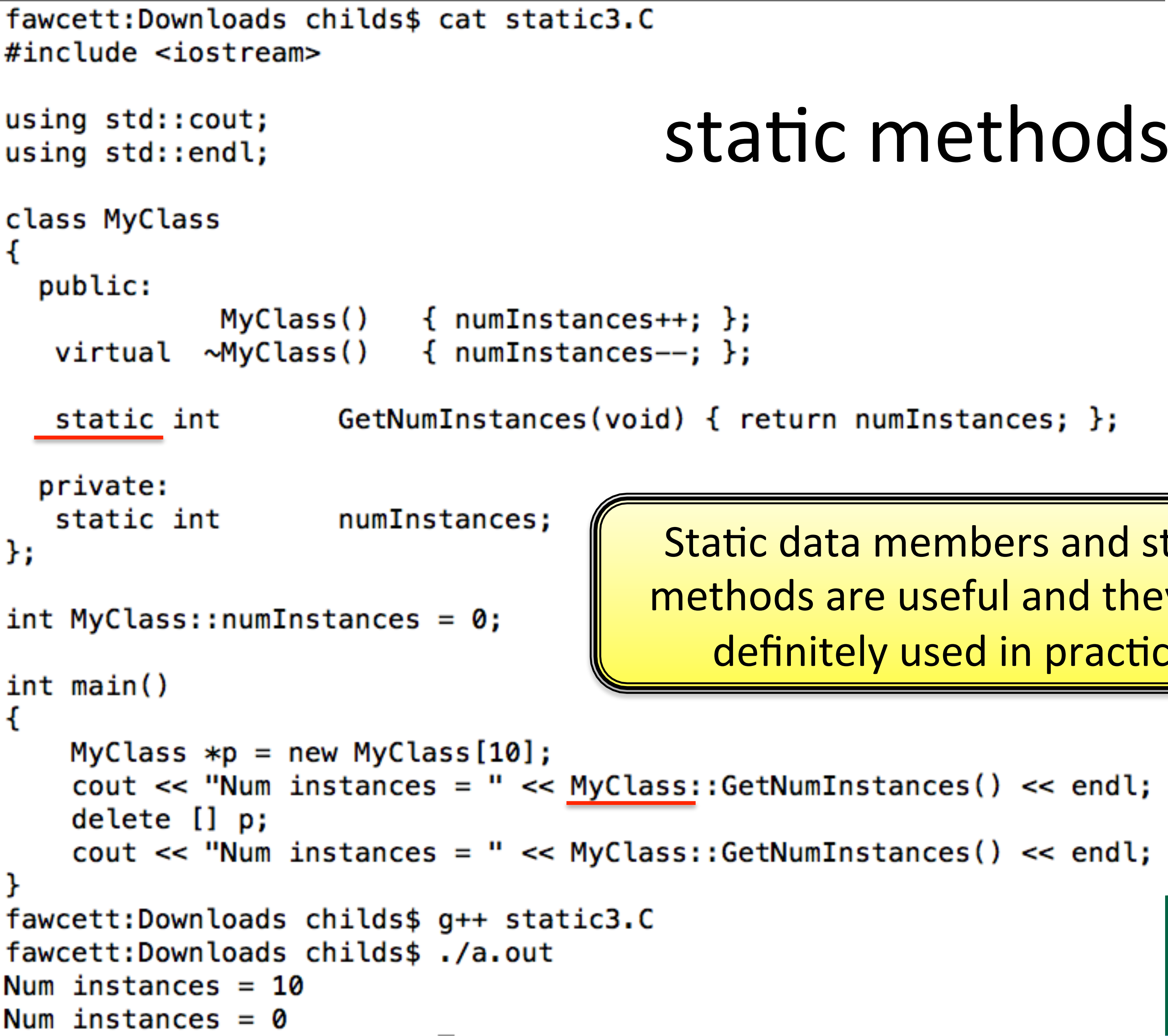
```
fawcett:330 childs$ g++ -c file2.C
fawcett:330 childs$ g++ file1.o file2.o
fawcett:330 childs$ ./a.out
count is 1
```

static usage #3: making a singleton for a class



```
fawcett:Downloads childs$ cat static3.C
#include <iostream>

using std::cout;
using std::endl;

class MyClass
{
  public:
             MyClass()   { numInstances++; };
   virtual  ~MyClass()   { numInstances--; };

   int       GetNumInstances(void) { return numInstances; };

  private:
   static int       numInstances;
};

int MyClass::numInstances = 0;

int main()
{
    MyClass *p = new MyClass[10];
    cout << "Num instances = " << p[0].GetNumInstances() << endl;
    delete [] p;
    cout << "Num instances = " << p[0].GetNumInstances() << endl;
}
```

static methods

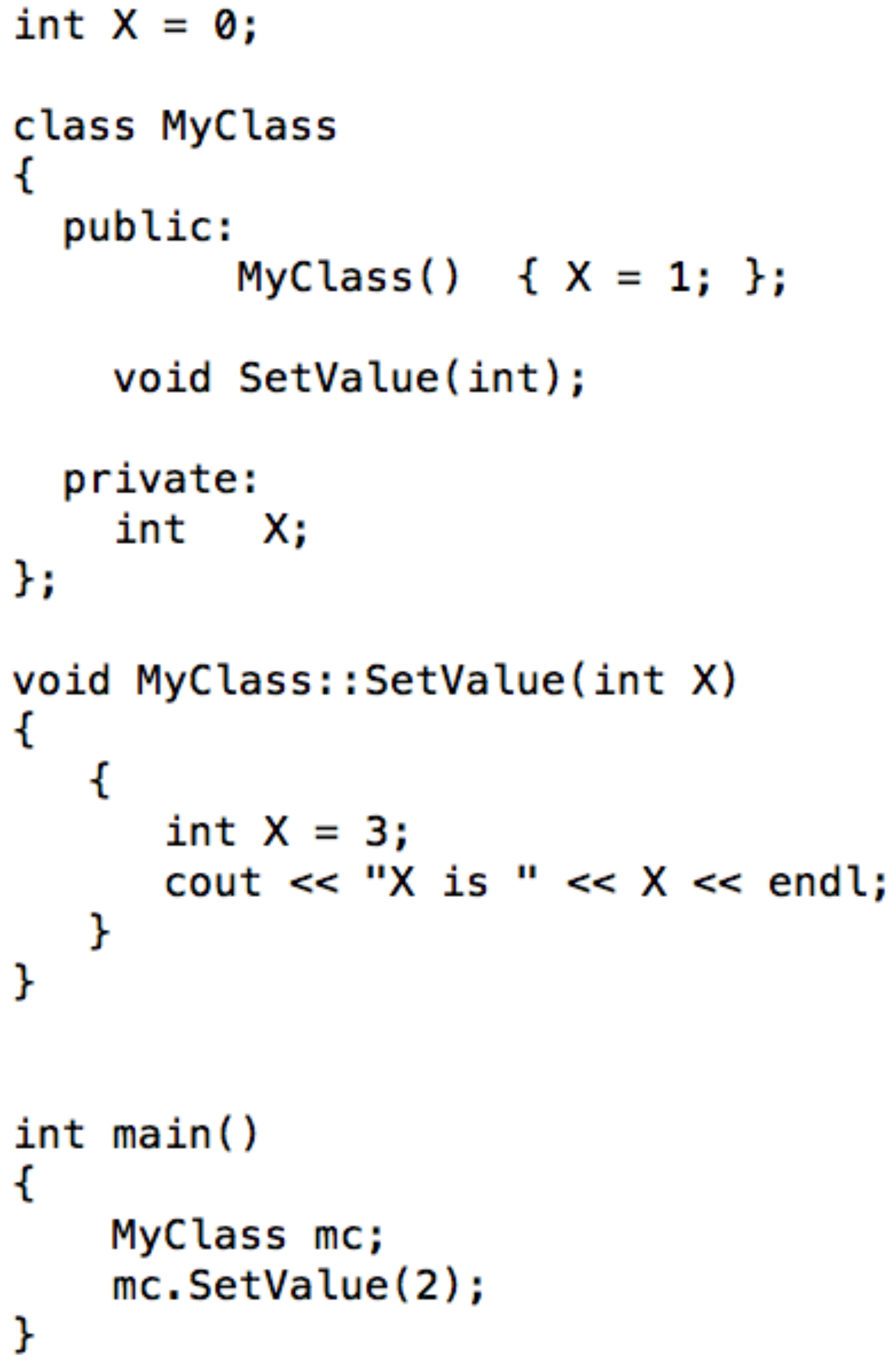


```
fawcett:Downloads childs$ cat static3.C
#include <iostream>

using std::cout;
using std::endl;

class MyClass
{
  public:
              MyClass()   { numInstances++; };
   virtual  ~MyClass()   { numInstances--; };

   static int        GetNumInstances(void) { return numInstances; };

  private:
   static int        numInstances;
};

int MyClass::numInstances = 0;

int main()
{
    MyClass *p = new MyClass[10];
    cout << "Num instances = " << MyClass::GetNumInstances() << endl;
    delete [] p;
    cout << "Num instances = " << MyClass::GetNumInstances() << endl;
}
fawcett:Downloads childs$ g++ static3.C
fawcett:Downloads childs$ ./a.out
Num instances = 10
Num instances = 0
```

Static data members and static methods are useful and they are definitely used in practice

Scope

scope



```
int X = 0;

class MyClass
{
  public:
         MyClass()  { X = 1; };

    void SetValue(int);

  private:
    int   X;
};

void MyClass::SetValue(int X)
{
   {
      int X = 3;
      cout << "X is " << X << endl;
   }
}


int main()
{
    MyClass mc;
    mc.SetValue(2);
}
```

This one will compile … the compiler thinks that you made a new scope on purpose.

So what does it print?

Answer: 3

Scope Rules

- The compiler looks for variables:
  - inside a function or block
  - function arguments
  - data members (methods only)
  - globals

Sort of new / sort of review

Overloading Operators

- NOTE: I lectured on this some, but it was informal. These slides formally capture the ideas we discussed.

C++ lets you define operators

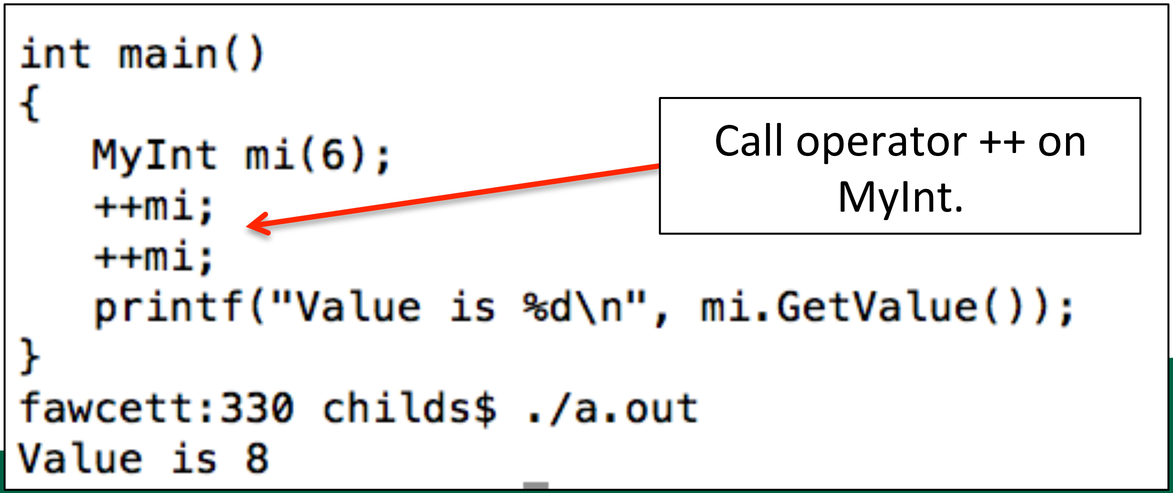
- You declare a method that uses an operator in conjunction with a class
  - +, -, /, !, ++, etc.
- You can then use operator in your code, since the compiler now understands how to use the operator with your class
- This is called “operator overloading”
  - … we are overloading the use of the operator for more than just the simple types.

Example of operator overloading

```
class MyInt
{
  public:
    MyInt(int x) { myInt = x; };

    MyInt& operator++();

    int     GetValue(void) { return myInt; };

  protected:
    int     myInt;
};

MyInt &
MyInt::operator++()
{
    myInt++;
    return *this;
}
```

Declare operator ++ will be overloaded for MyInt

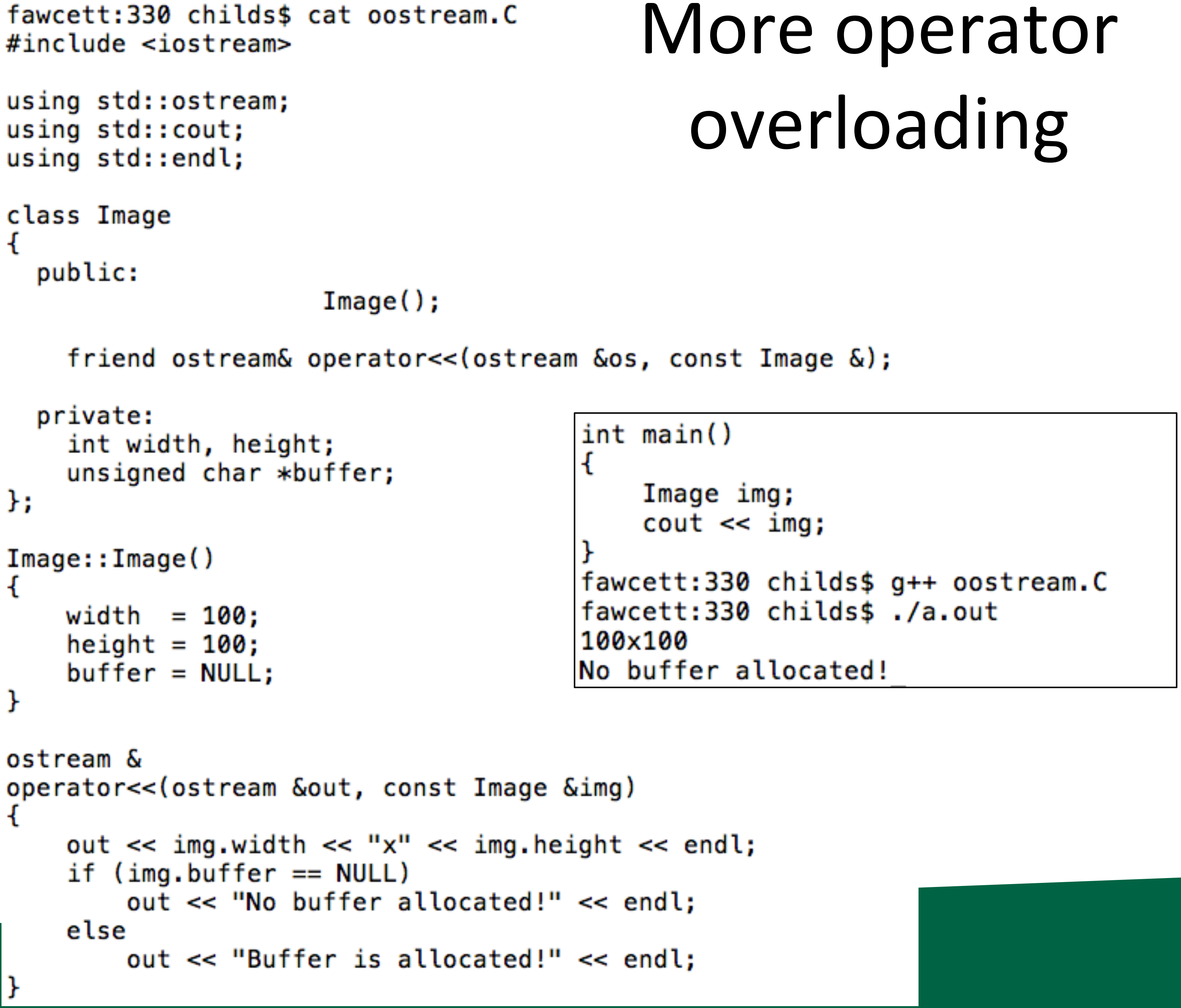
Define operator ++ for MyInt



```
int main()
{
   MyInt mi(6);
   ++mi;
   ++mi;
   printf("Value is %d\n", mi.GetValue());
}
fawcett:330 childs$ ./a.out
Value is 8
```

Call operator ++ on MyInt.

More operator overloading

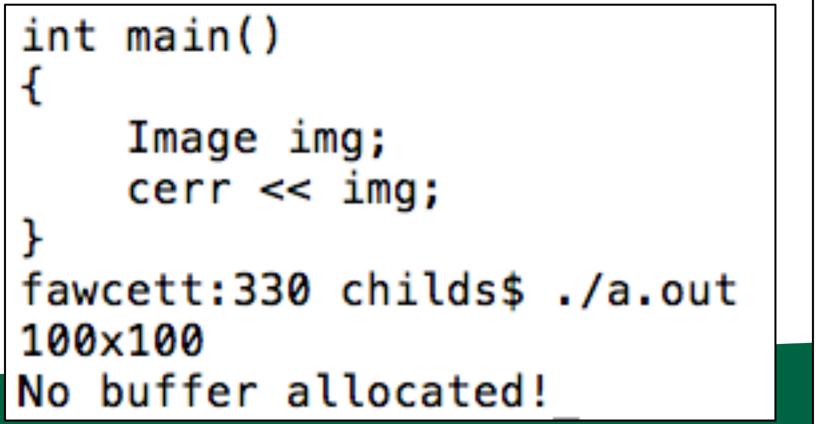


```
fawcett:330 childs$ cat oostream.C
#include <iostream>

using std::ostream;
using std::cout;
using std::endl;

class Image
{
  public:
                   Image();

    friend ostream& operator<<(ostream &os, const Image &);

  private:
    int width, height;
    unsigned char *buffer;
};

Image::Image()
{
    width  = 100;
    height = 100;
    buffer = NULL;
}

ostream &
operator<<(ostream &out, const Image &img)
{
    out << img.width << "x" << img.height << endl;
    if (img.buffer == NULL)
        out << "No buffer allocated!" << endl;
    else
        out << "Buffer is allocated!" << endl;
}
```

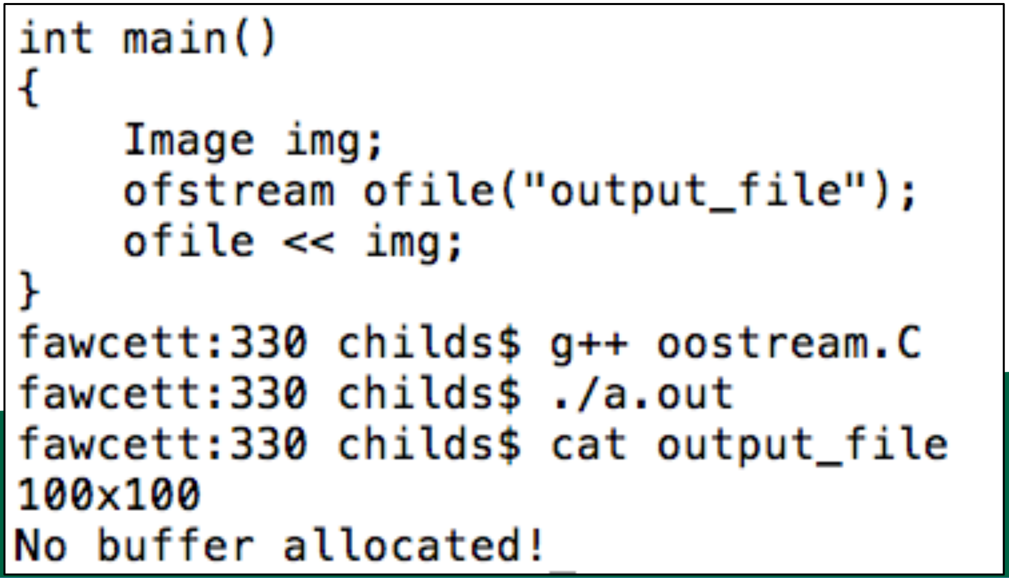
```
int main()
{
    Image img;
    cout << img;
}
fawcett:330 childs$ g++ oostream.C
fawcett:330 childs$ ./a.out
100x100
No buffer allocated!
```

Beauty of inheritance

- ostream provides an abstraction
  - That's all Image needs to know
    - it is a stream that is an output
  - You code to that interface
  - All ostream's work with it

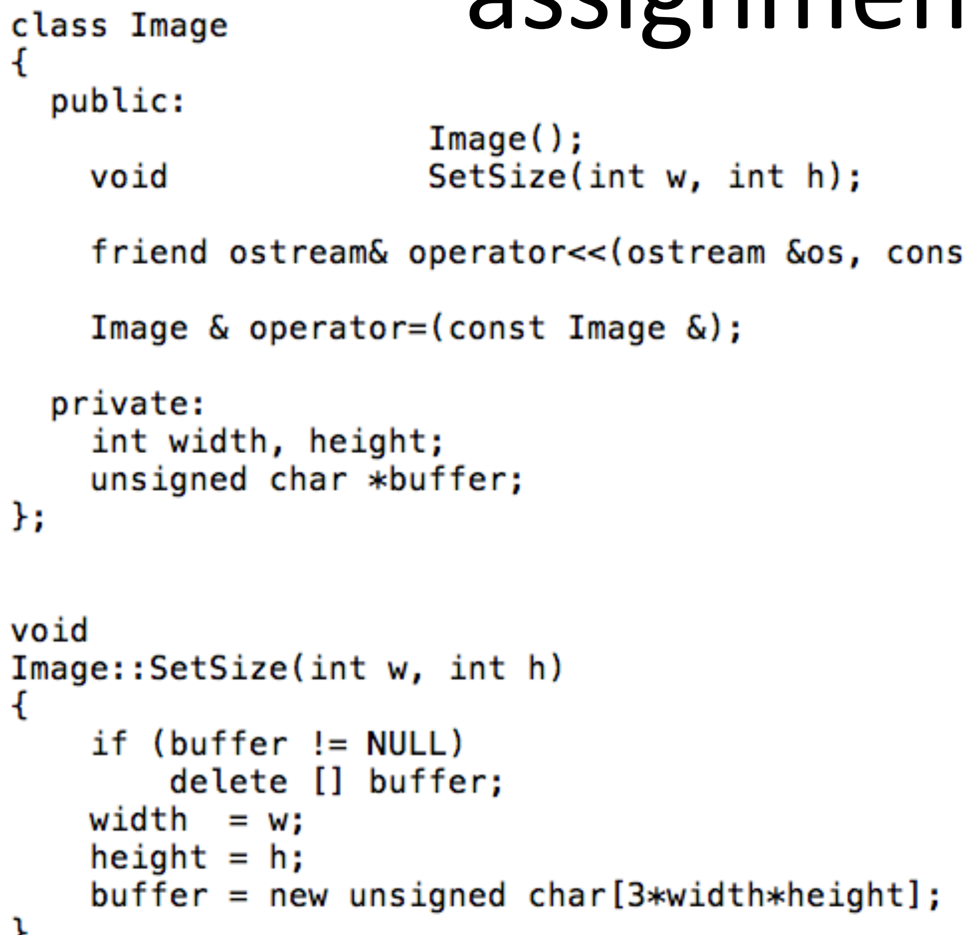


```
int main()
{
    Image img;
    cerr << img;
}
fawcett:330 childs$ ./a.out
100x100
No buffer allocated!
```

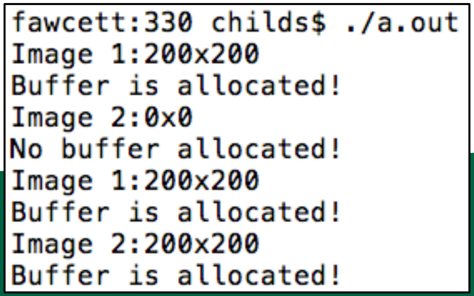


```
int main()
{
    Image img;
    ofstream ofile("output_file");
    ofile << img;
}
fawcett:330 childs$ g++ oostream.C
fawcett:330 childs$ ./a.out
fawcett:330 childs$ cat output_file
100x100
No buffer allocated!
```

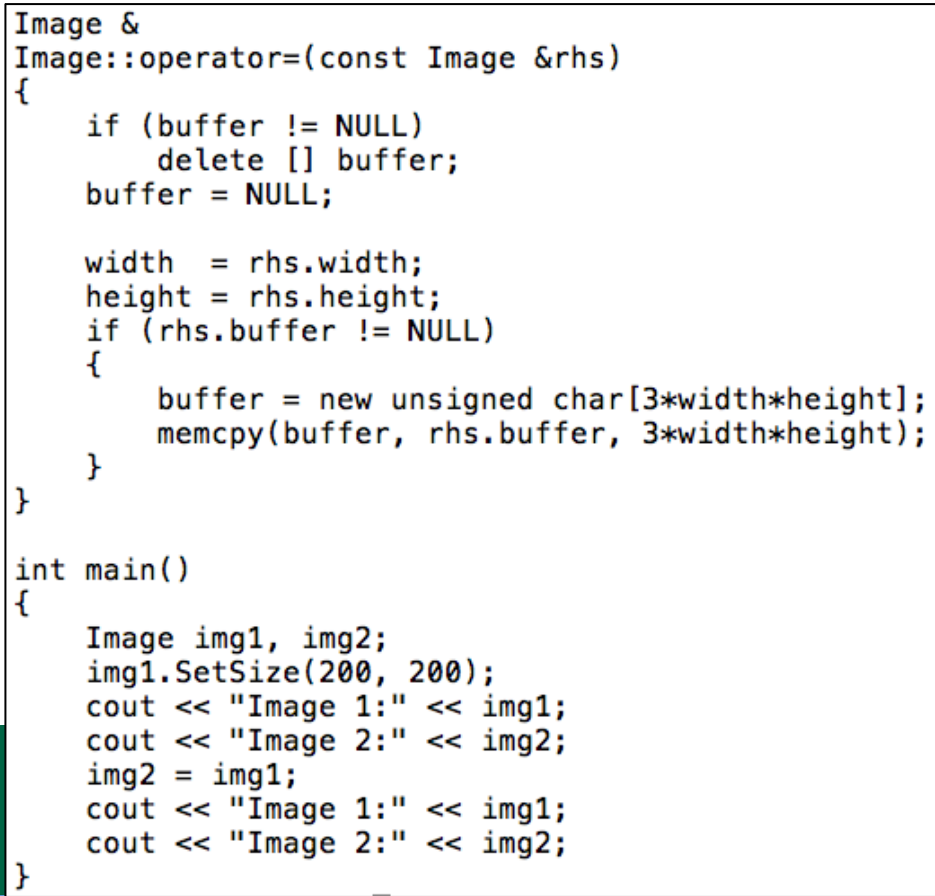
assignment operator



```
class Image
{
  public:
                         Image();
    void                 SetSize(int w, int h);

    friend ostream& operator<<(ostream &os, const Image &);

    Image & operator=(const Image &);

  private:
    int width, height;
    unsigned char *buffer;
};


void
Image::SetSize(int w, int h)
{
    if (buffer != NULL)
        delete [] buffer;
    width  = w;
    height = h;
    buffer = new unsigned char[3*width*height];
}
```

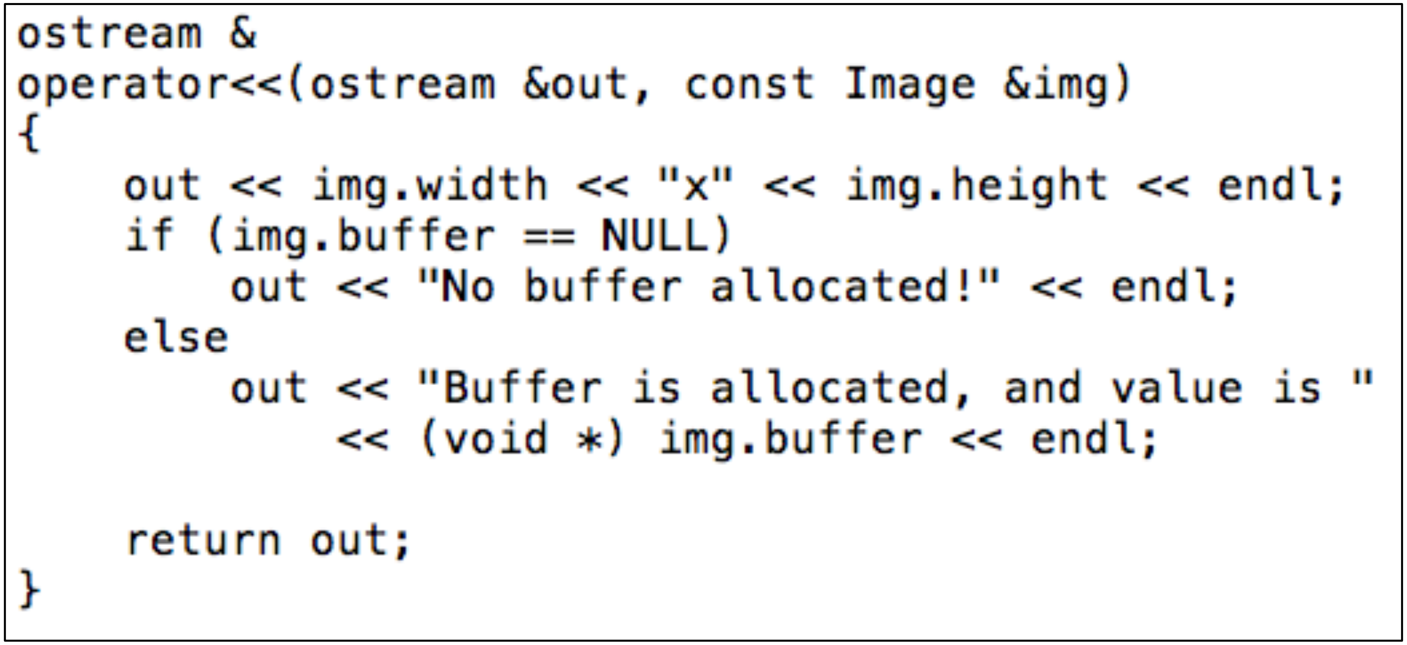


```
fawcett:330 childs$ ./a.out
Image 1:200x200
Buffer is allocated!
Image 2:0x0
No buffer allocated!
Image 1:200x200
Buffer is allocated!
Image 2:200x200
Buffer is allocated!
```

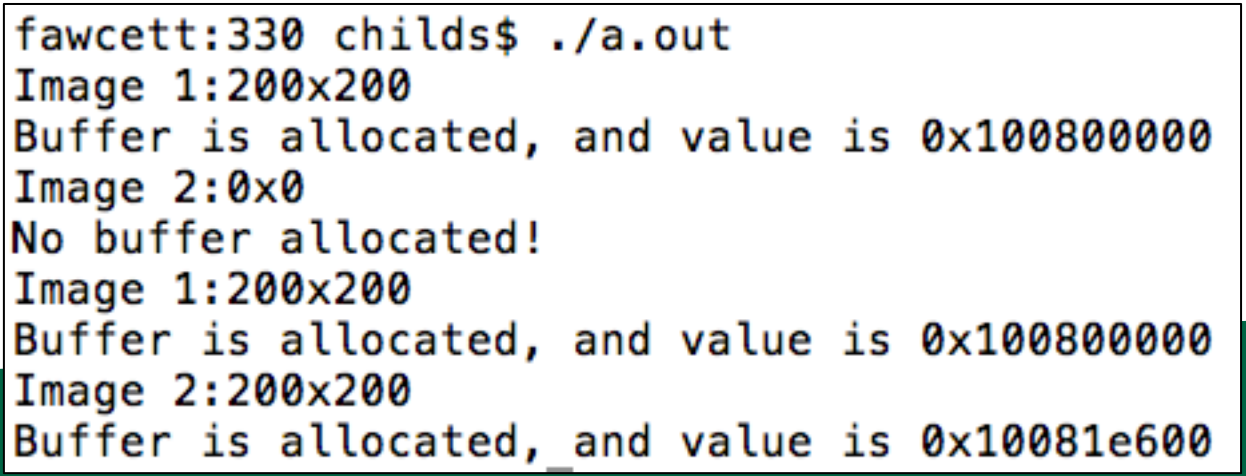


```
Image &
Image::operator=(const Image &rhs)
{
    if (buffer != NULL)
        delete [] buffer;
    buffer = NULL;

    width  = rhs.width;
    height = rhs.height;
    if (rhs.buffer != NULL)
    {
        buffer = new unsigned char[3*width*height];
        memcpy(buffer, rhs.buffer, 3*width*height);
    }
}

int main()
{
    Image img1, img2;
    img1.SetSize(200, 200);
    cout << "Image 1:" << img1;
    cout << "Image 2:" << img2;
    img2 = img1;
    cout << "Image 1:" << img1;
    cout << "Image 2:" << img2;
}
```

let’s do this again…



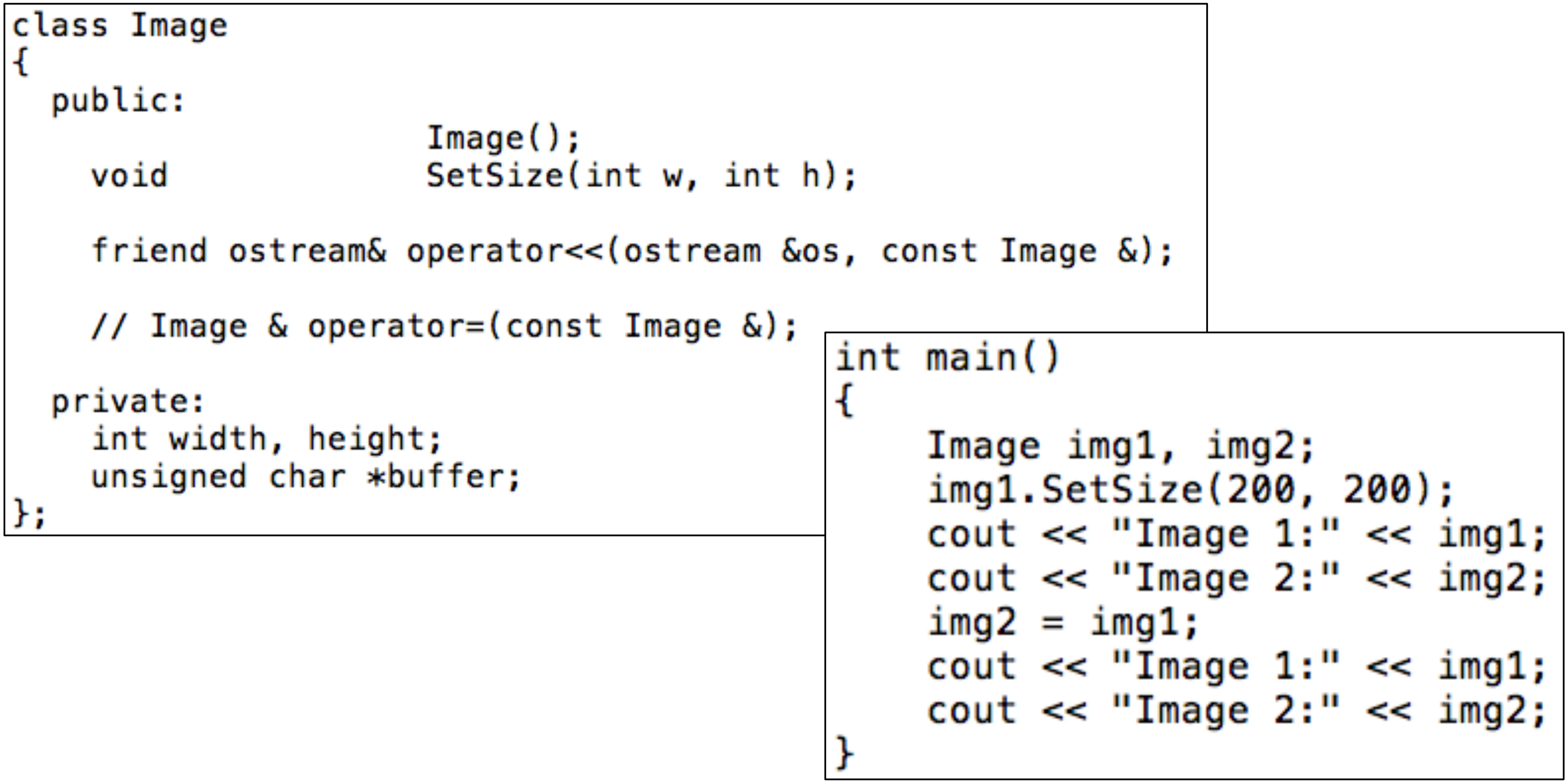
```
ostream &
operator<<(ostream &out, const Image &img)
{
    out << img.width << "x" << img.height << endl;
    if (img.buffer == NULL)
        out << "No buffer allocated!" << endl;
    else
        out << "Buffer is allocated, and value is "
            << (void *) img.buffer << endl;

    return out;
}
```



```
fawcett:330 childs$ ./a.out
Image 1:200x200
Buffer is allocated, and value is 0x100800000
Image 2:0x0
No buffer allocated!
Image 1:200x200
Buffer is allocated, and value is 0x100800000
Image 2:200x200
Buffer is allocated, and value is 0x10081e600
```

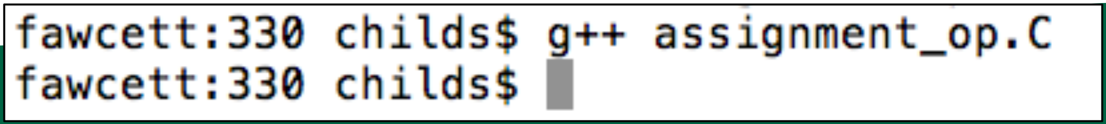
(ok, fine)

let’s do this again…



```
class Image
{
  public:
                     Image();
    void             SetSize(int w, int h);

    friend ostream& operator<<(ostream &os, const Image &);

    // Image & operator=(const Image &);

  private:
    int width, height;
    unsigned char *buffer;
};
```

```
int main()
{
    Image img1, img2;
    img1.SetSize(200, 200);
    cout << "Image 1:" << img1;
    cout << "Image 2:" << img2;
    img2 = img1;
    cout << "Image 1:" << img1;
    cout << "Image 2:" << img2;
}
```

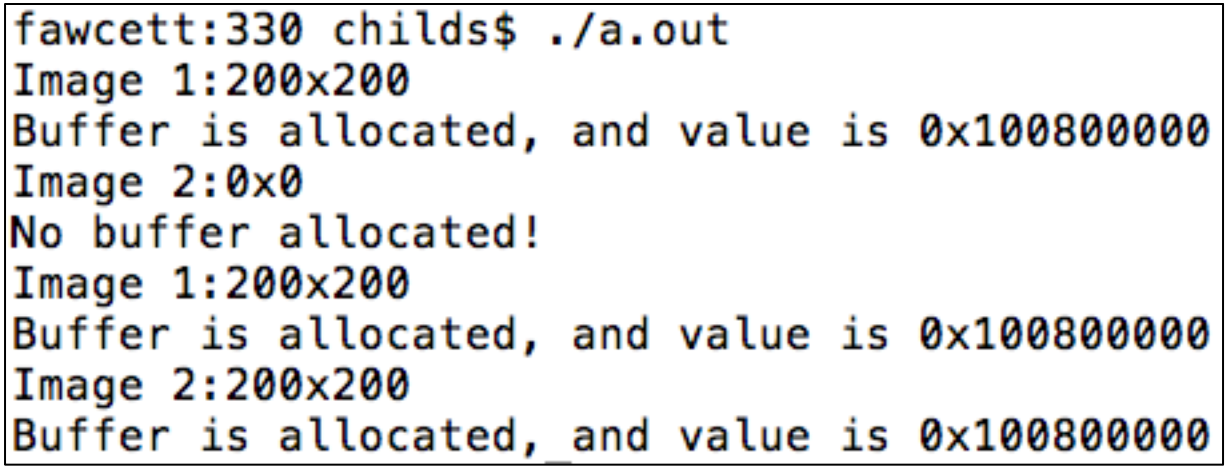


```
fawcett:330 childs$ g++ assignment_op.C
fawcett:330 childs$ █
```

it still compiled …
why?

C++ defines a default assignment operator for you

- This assignment operator does a bitwise copy from one object to the other.
- Does anyone see a problem with this?



```
fawcett:330 childs$ ./a.out
Image 1:200x200
Buffer is allocated, and value is 0x100800000
Image 2:0x0
No buffer allocated!
Image 1:200x200
Buffer is allocated, and value is 0x100800000
Image 2:200x200
Buffer is allocated, and value is 0x100800000
```

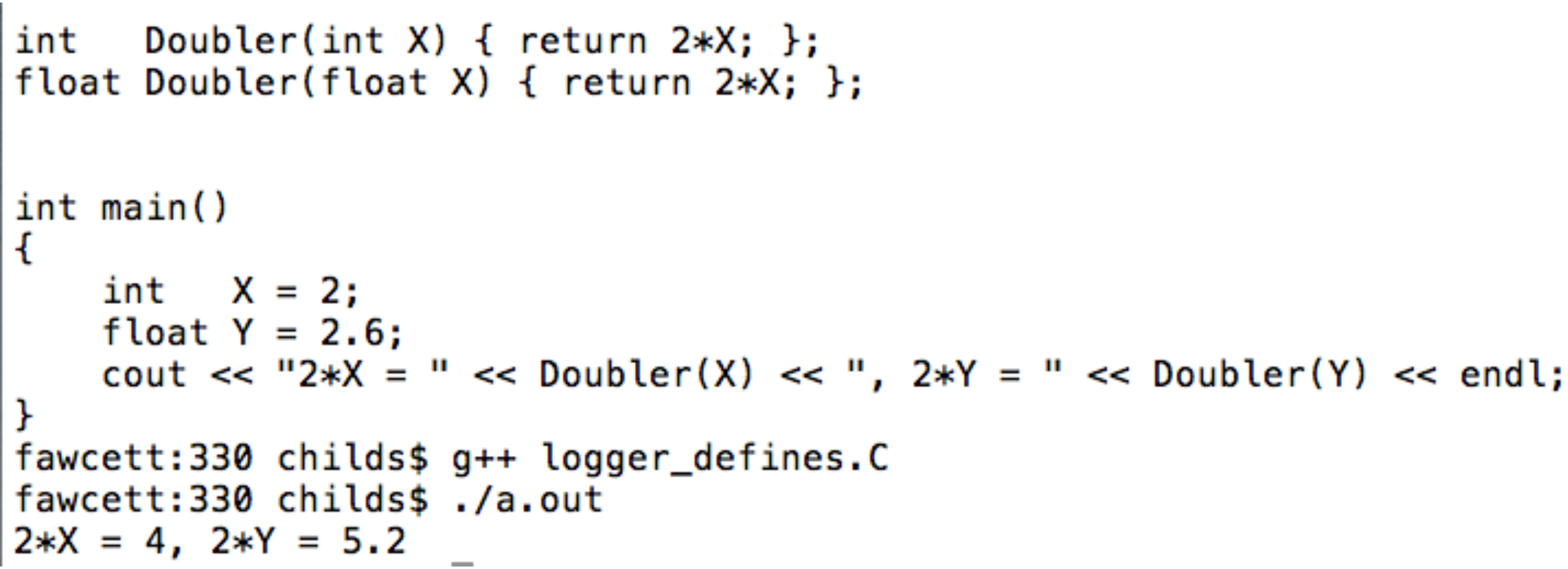
This behavior is sometimes OK and sometimes disastrous.

Copy constructors: same deal

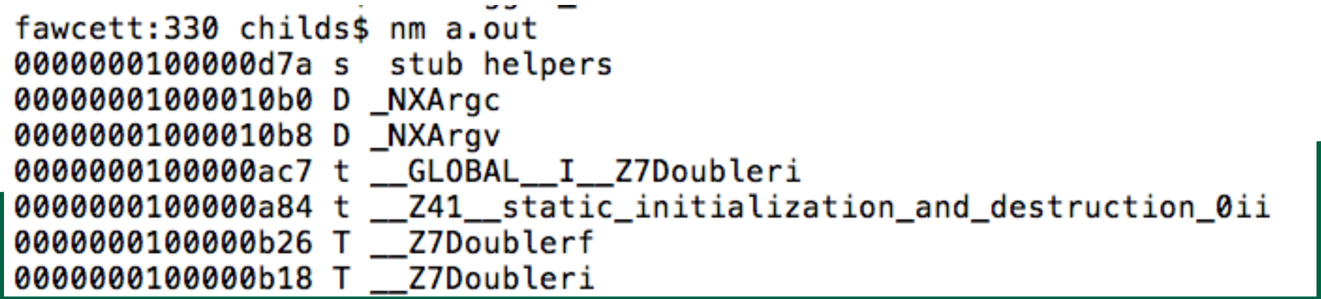
- C++ automatically defines a copy constructor that does bitwise copying.
- Solutions for copy constructor and assignment operators:
  - Re-define them yourself to do “the right thing”
  - Re-define them yourself to throw exceptions
  - Make them private so they can’t be called

Templates

Motivation

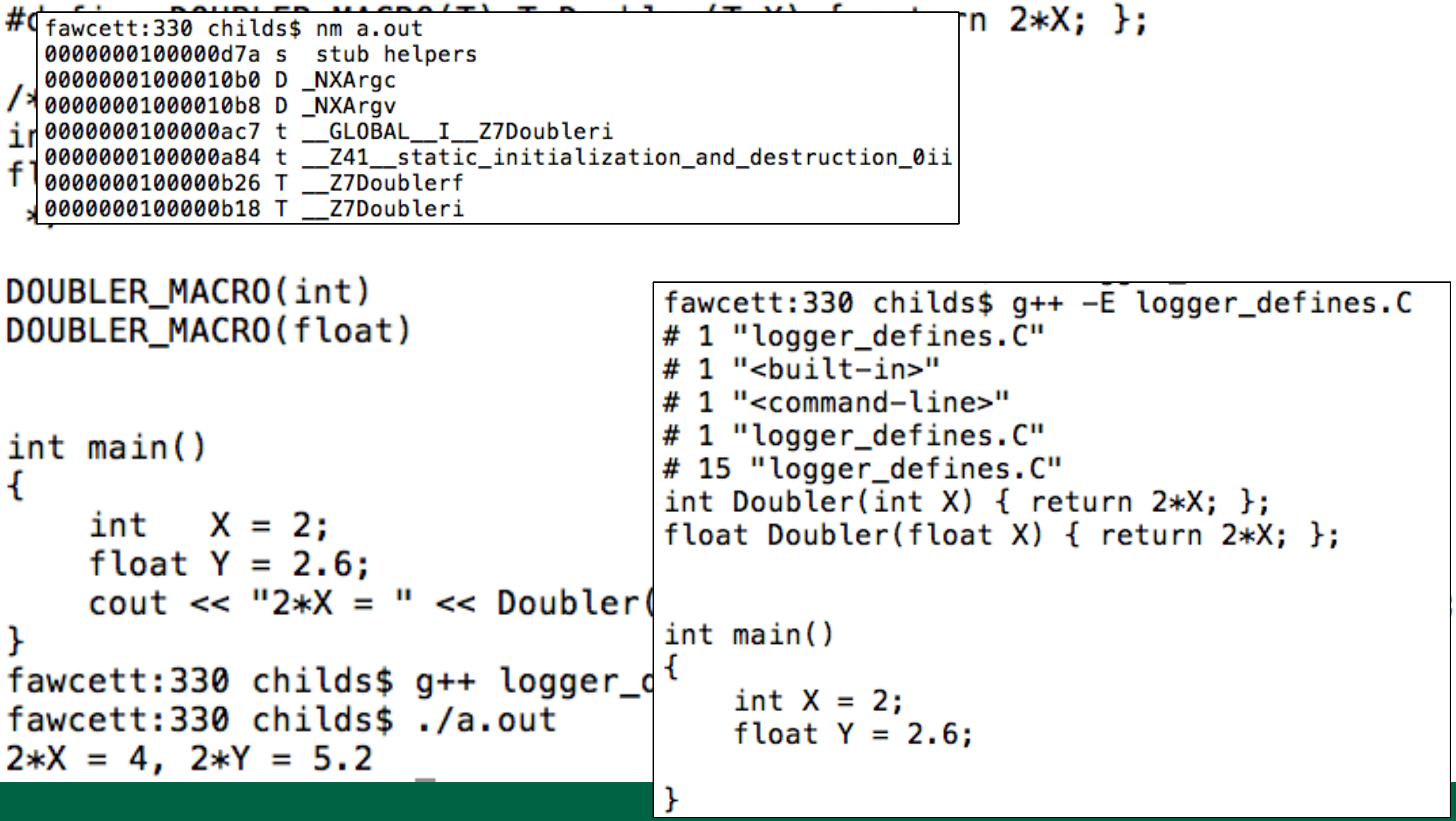


```
int   Doubler(int X) { return 2*X; };
float Doubler(float X) { return 2*X; };


int main()
{
    int   X = 2;
    float Y = 2.6;
    cout << "2*X = " << Doubler(X) << ", 2*Y = " << Doubler(Y) << endl;
}
fawcett:330 childs$ g++ logger_defines.C
fawcett:330 childs$ ./a.out
2*X = 4, 2*Y = 5.2
```



```
fawcett:330 childs$ nm a.out
0000000100000d7a s  stub helpers
00000001000010b0 D _NXArgc
00000001000010b8 D _NXArgv
0000000100000ac7 t __GLOBAL__I__Z7Doubleri
0000000100000a84 t __Z41__static_initialization_and_destruction_0ii
0000000100000b26 T __Z7Doublerf
0000000100000b18 T __Z7Doubleri
```

Motivation

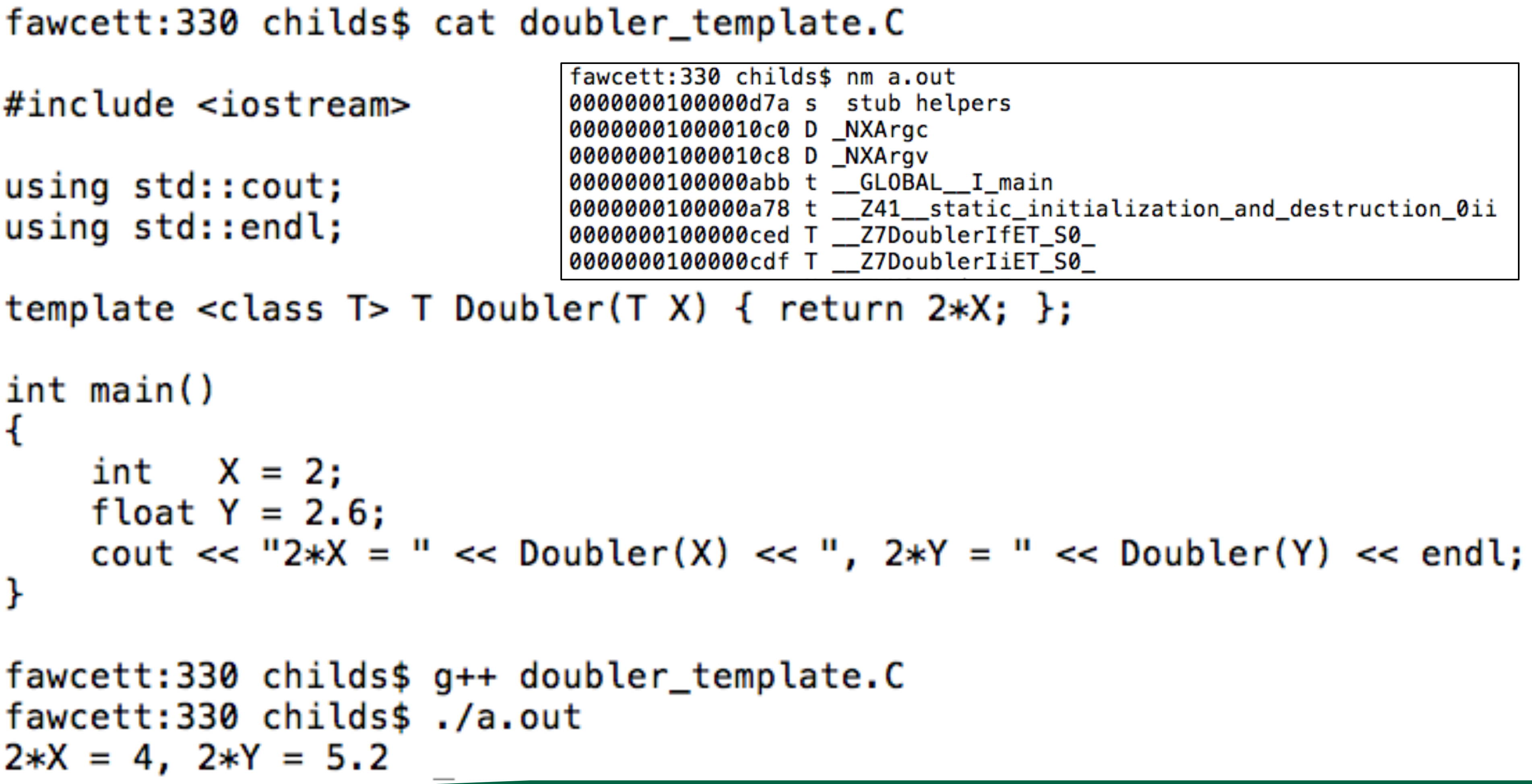


```
#d                                                                   rn 2*X; };

/*
in
fl
 */

DOUBLER_MACRO(int)
DOUBLER_MACRO(float)


int main()
{
    int   X = 2;
    float Y = 2.6;
    cout << "2*X = " << Doubler(
}
fawcett:330 childs$ g++ logger_d
fawcett:330 childs$ ./a.out
2*X = 4, 2*Y = 5.2
```

```
fawcett:330 childs$ nm a.out
0000000100000d7a s  stub helpers
00000001000010b0 D _NXArgc
00000001000010b8 D _NXArgv
0000000100000ac7 t __GLOBAL__I__Z7Doubleri
0000000100000a84 t __Z41__static_initialization_and_destruction_0ii
0000000100000b26 T __Z7Doublerf
0000000100000b18 T __Z7Doubleri
```

```
fawcett:330 childs$ g++ -E logger_defines.C
# 1 "logger_defines.C"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "logger_defines.C"
# 15 "logger_defines.C"
int Doubler(int X) { return 2*X; };
float Doubler(float X) { return 2*X; };


int main()
{
    int X = 2;
    float Y = 2.6;

}
```

First Template



```
fawcett:330 childs$ cat doubler_template.C

#include <iostream>

using std::cout;
using std::endl;

template <class T> T Doubler(T X) { return 2*X; };

int main()
{
    int   X = 2;
    float Y = 2.6;
    cout << "2*X = " << Doubler(X) << ", 2*Y = " << Doubler(Y) << endl;
}

fawcett:330 childs$ g++ doubler_template.C
fawcett:330 childs$ ./a.out
2*X = 4, 2*Y = 5.2
```

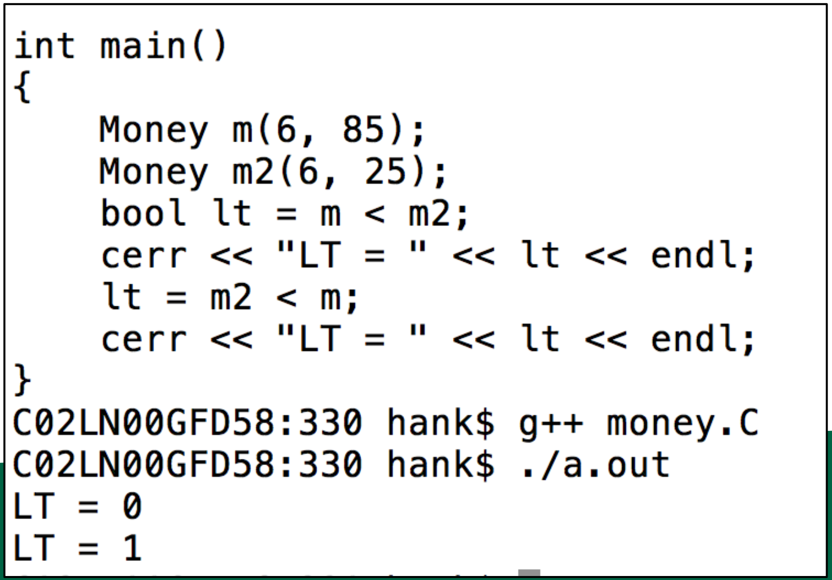
```
fawcett:330 childs$ nm a.out
0000000100000d7a s  stub helpers
00000001000010c0 D _NXArgc
00000001000010c8 D _NXArgv
0000000100000abb t __GLOBAL__I_main
0000000100000a78 t __Z41__static_initialization_and_destruction_0ii
0000000100000ced T __Z7DoublerIfET_S0_
0000000100000cdf T __Z7DoublerIiET_S0_
```

Will now do an example to compare templates and virtual functions

- Will take some buildup…

Money Class

```
class Money
{
  public:
                 Money(int d, int c) { dollars = d; cents = c; };
    bool         operator<(const Money &m);

  private:
    int dollars;
    int cents;
};

bool Money::operator<(const Money &m)
{
    if (dollars < m.dollars)
        return true;
    if (dollars == m.dollars)
        return (cents < m.cents);

    return false;
}
```



```
int main()
{
    Money m(6, 85);
    Money m2(6, 25);
    bool lt = m < m2;
    cerr << "LT = " << lt << endl;
    lt = m2 < m;
    cerr << "LT = " << lt << endl;
}
C02LN00GFD58:330 hank$ g++ money.C
C02LN00GFD58:330 hank$ ./a.out
LT = 0
LT = 1
```

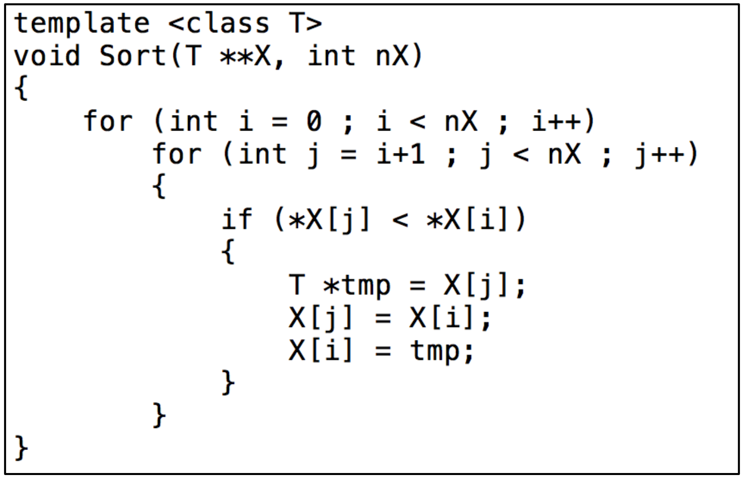
License Plate Class

```
class LicensePlate
{
  public:
               LicensePlate(char c1, char c2, char c3,
                            int i1, int i2, int i3)
                  {
                     letters[0] = c1;
                     letters[1] = c2;
                     letters[2] = c3;
                     numbers[0] = i1;
                     numbers[1] = i2;
                     numbers[2] = i3;
                  }

    bool       operator<(const LicensePlate &);

  private:
    char       letters[3];
    int        numbers[3];
};
```

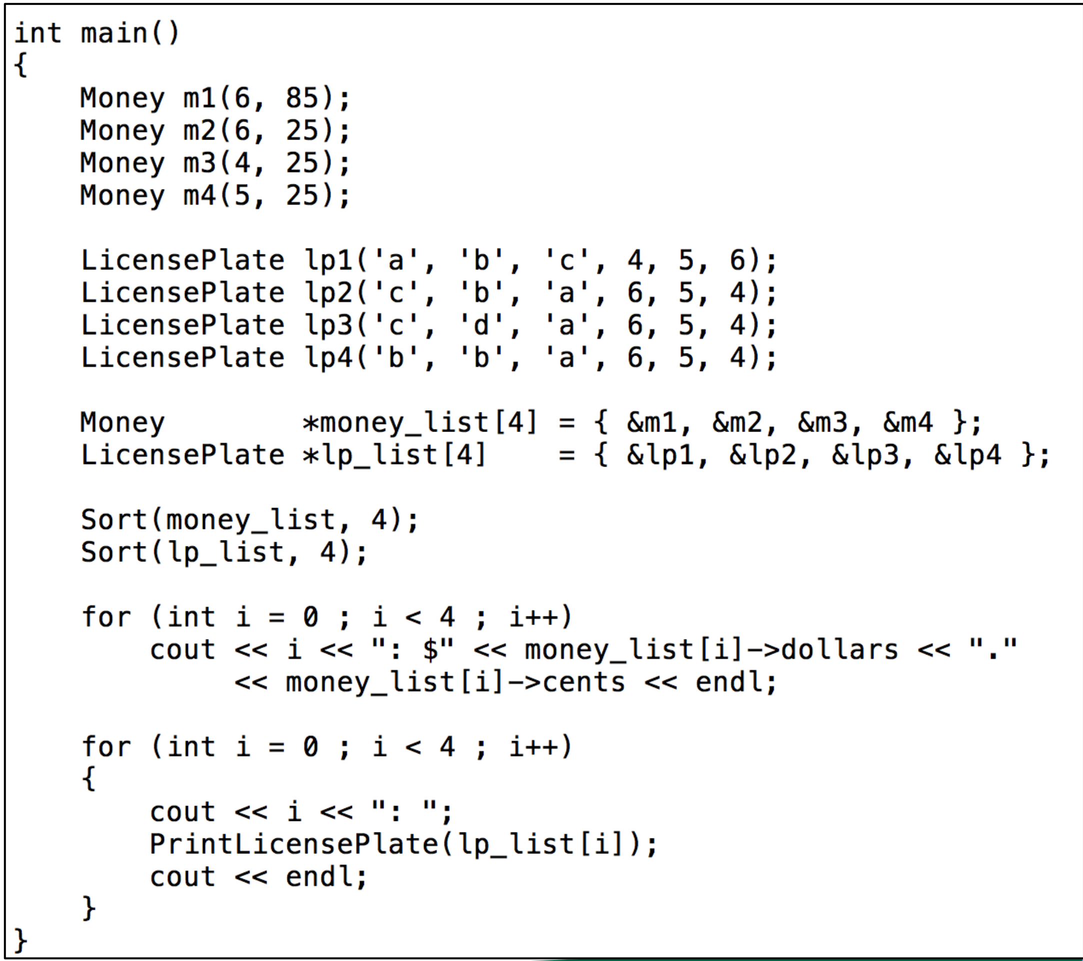
```
bool LicensePlate::operator<(const LicensePlate &rhs)
{
    for (int i = 0 ; i < 3 ; i++)
    {
        if (letters[i] < rhs.letters[i])
            return true;
        if (letters[i] > rhs.letters[i])
            return false;
    }
    for (int i = 0 ; i < 3 ; i++)
    {
        if (numbers[i] < rhs.numbers[i])
            return true;
        if (numbers[i] > rhs.numbers[i])
            return false;
    }

    // equal
    return false;
}
```

```
int main()
{
    LicensePlate lp1('a', 'b', 'c', 4, 5, 6);
    LicensePlate lp2('c', 'b', 'a', 6, 5, 4);
    bool lt = lp1 < lp2;
    cerr << "LT = " << lt << endl;
    lt = lp2 < lp1;
    cerr << "LT = " << lt << endl;
}
```

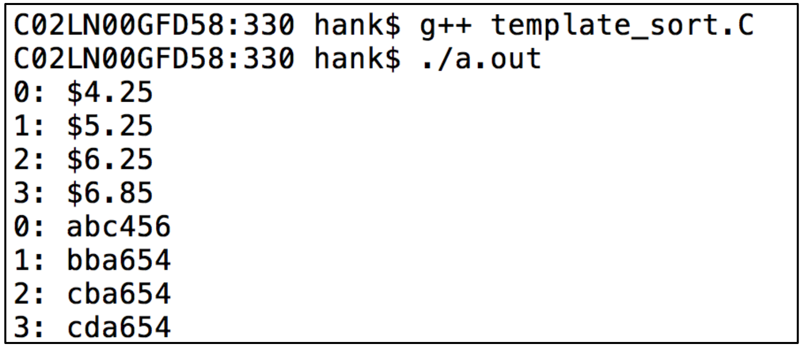
Sorting With Templates



```
template <class T>
void Sort(T **X, int nX)
{
    for (int i = 0 ; i < nX ; i++)
        for (int j = i+1 ; j < nX ; j++)
        {
            if (*X[j] < *X[i])
            {
                T *tmp = X[j];
                X[j] = X[i];
                X[i] = tmp;
            }
        }
}
```

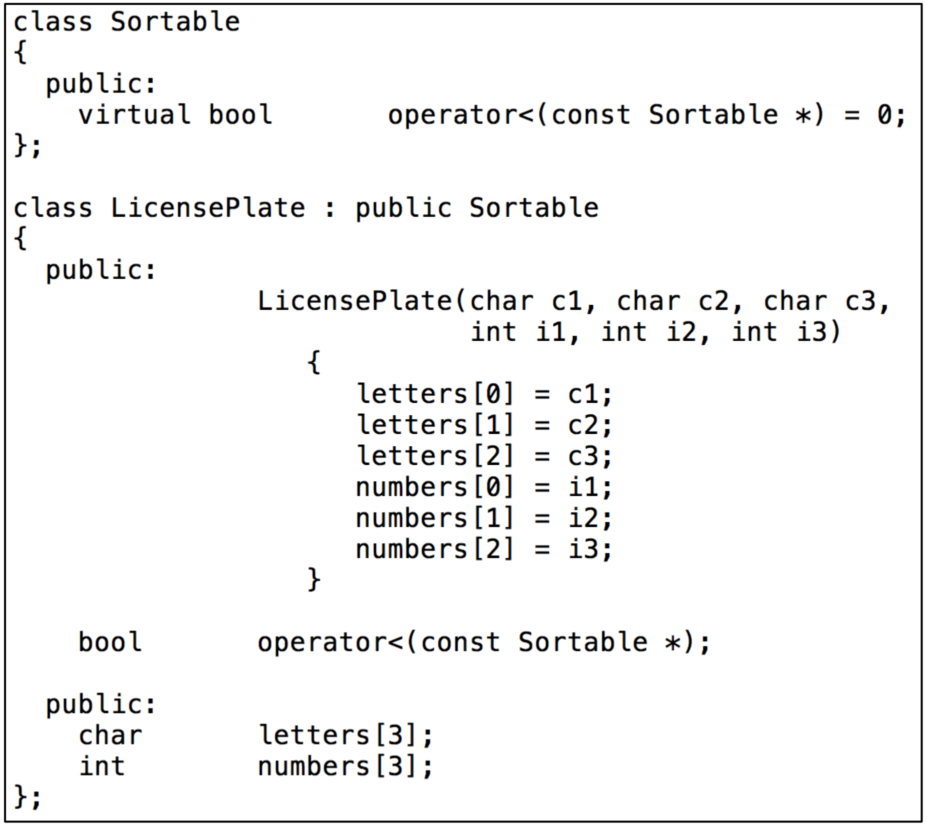


```
int main()
{
    Money m1(6, 85);
    Money m2(6, 25);
    Money m3(4, 25);
    Money m4(5, 25);

    LicensePlate lp1('a', 'b', 'c', 4, 5, 6);
    LicensePlate lp2('c', 'b', 'a', 6, 5, 4);
    LicensePlate lp3('c', 'd', 'a', 6, 5, 4);
    LicensePlate lp4('b', 'b', 'a', 6, 5, 4);

    Money        *money_list[4] = { &m1, &m2, &m3, &m4 };
    LicensePlate *lp_list[4]    = { &lp1, &lp2, &lp3, &lp4 };

    Sort(money_list, 4);
    Sort(lp_list, 4);

    for (int i = 0 ; i < 4 ; i++)
        cout << i << ": $" << money_list[i]->dollars << "."
             << money_list[i]->cents << endl;

    for (int i = 0 ; i < 4 ; i++)
    {
        cout << i << ": ";
        PrintLicensePlate(lp_list[i]);
        cout << endl;
    }
}
```

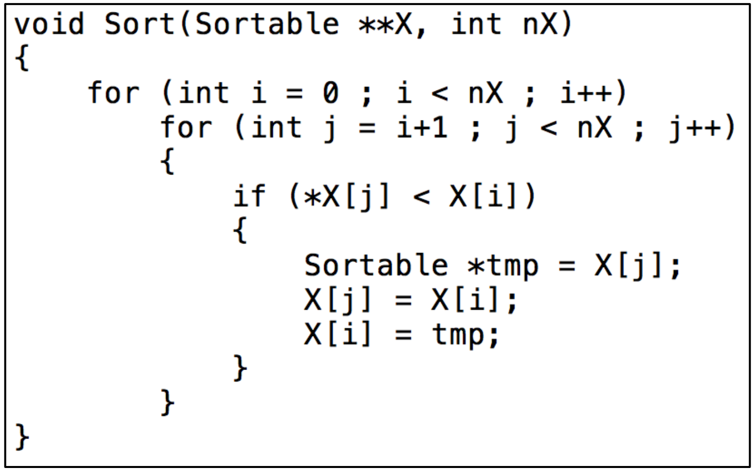


```
C02LN00GFD58:330 hank$ g++ template_sort.C
C02LN00GFD58:330 hank$ ./a.out
0:  $4.25
1:  $5.25
2:  $6.25
3:  $6.85
0:  abc456
1:  bba654
2:  cba654
3:  cda654
```

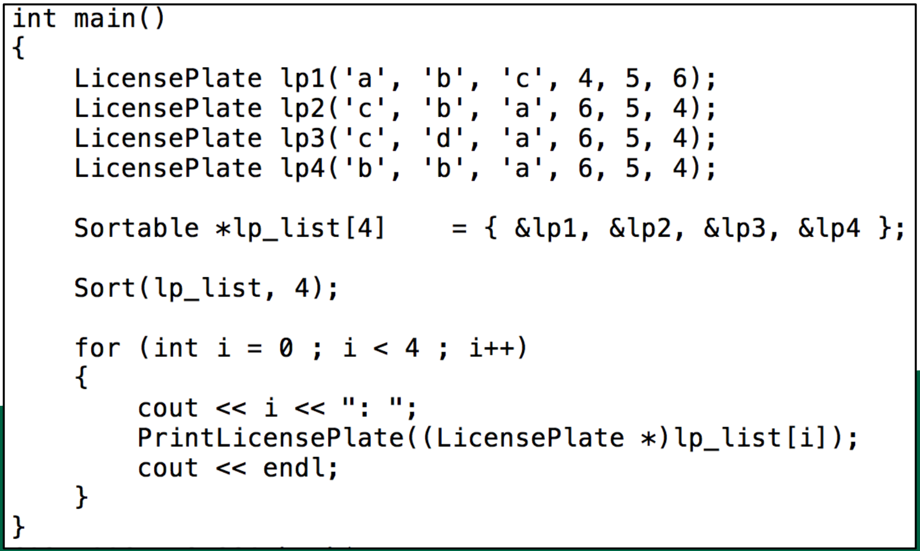
Doing the same with inheritance



```
class Sortable
{
  public:
    virtual bool       operator<(const Sortable *) = 0;
};

class LicensePlate : public Sortable
{
  public:
               LicensePlate(char c1, char c2, char c3,
                            int i1, int i2, int i3)
                  {
                     letters[0] = c1;
                     letters[1] = c2;
                     letters[2] = c3;
                     numbers[0] = i1;
                     numbers[1] = i2;
                     numbers[2] = i3;
                  }

    bool       operator<(const Sortable *);

  public:
    char       letters[3];
    int        numbers[3];
};
```



```
void Sort(Sortable **X, int nX)
{
    for (int i = 0 ; i < nX ; i++)
        for (int j = i+1 ; j < nX ; j++)
        {
            if (*X[j] < X[i])
            {
                Sortable *tmp = X[j];
                X[j] = X[i];
                X[i] = tmp;
            }
        }
}
```



```
int main()
{
    LicensePlate lp1('a', 'b', 'c', 4, 5, 6);
    LicensePlate lp2('c', 'b', 'a', 6, 5, 4);
    LicensePlate lp3('c', 'd', 'a', 6, 5, 4);
    LicensePlate lp4('b', 'b', 'a', 6, 5, 4);

    Sortable *lp_list[4]    = { &lp1, &lp2, &lp3, &lp4 };

    Sort(lp_list, 4);

    for (int i = 0 ; i < 4 ; i++)
    {
        cout << i << ": ";
        PrintLicensePlate((LicensePlate *)lp_list[i]);
        cout << endl;
    }
}
```

Templates vs Virtual Functions

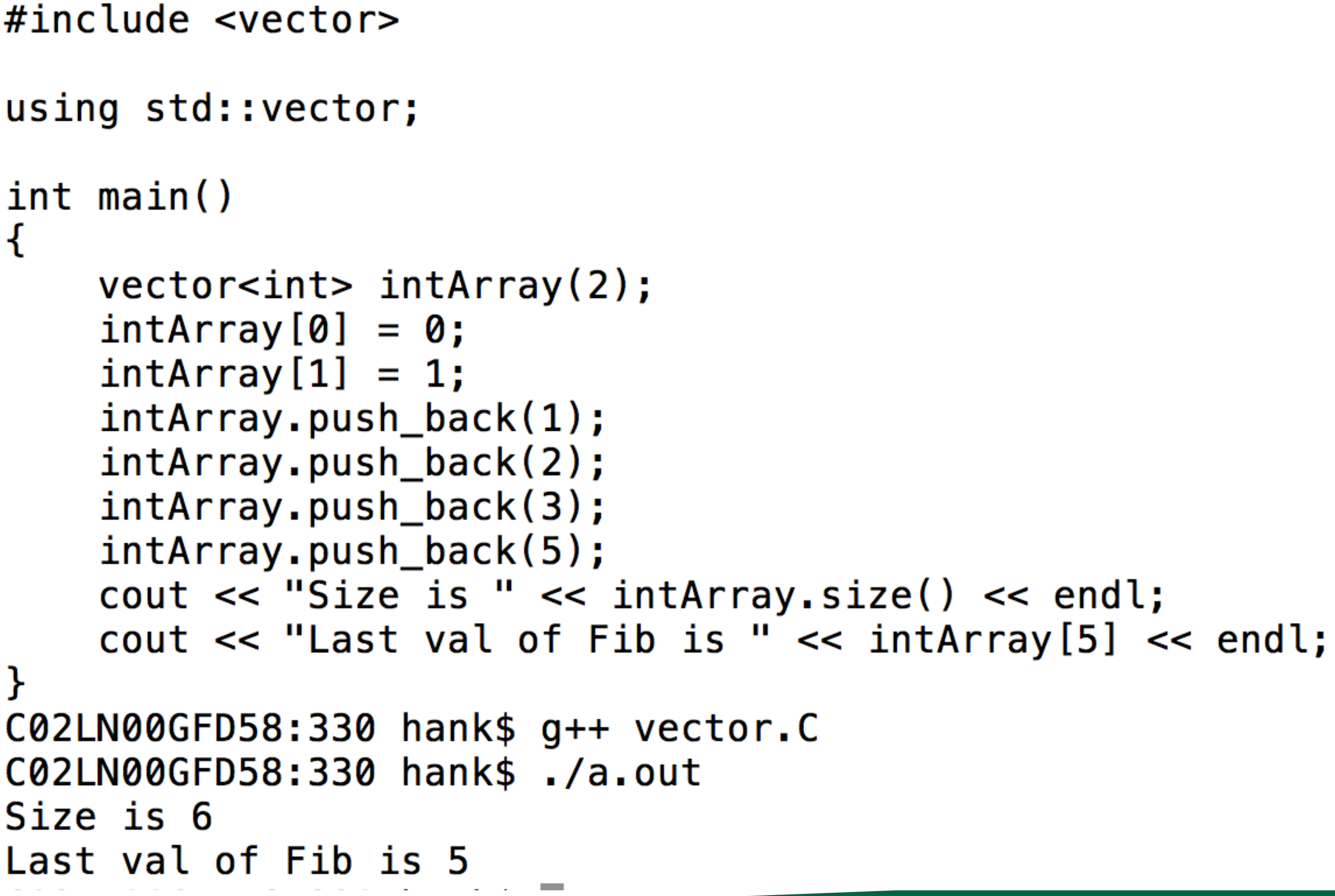
- Virtual functions:
  - Had to affect inheritance hierarchy
  - Overhead in function call (virtual function table)
- Templates:
  - Did not need to affect inheritance hierarchy, although function names had to coincide
  - No additional overhead (resolved at compile time)

Standard Template Library

Standard Template Library

- Standard Template Library: STL
- Many, many templated types
- Can ease your programming burden
- Can also hide significant performance issues
  - And you use C/C++ for performance
- My recommendation: use with caution for code that needs to be performant

Vector

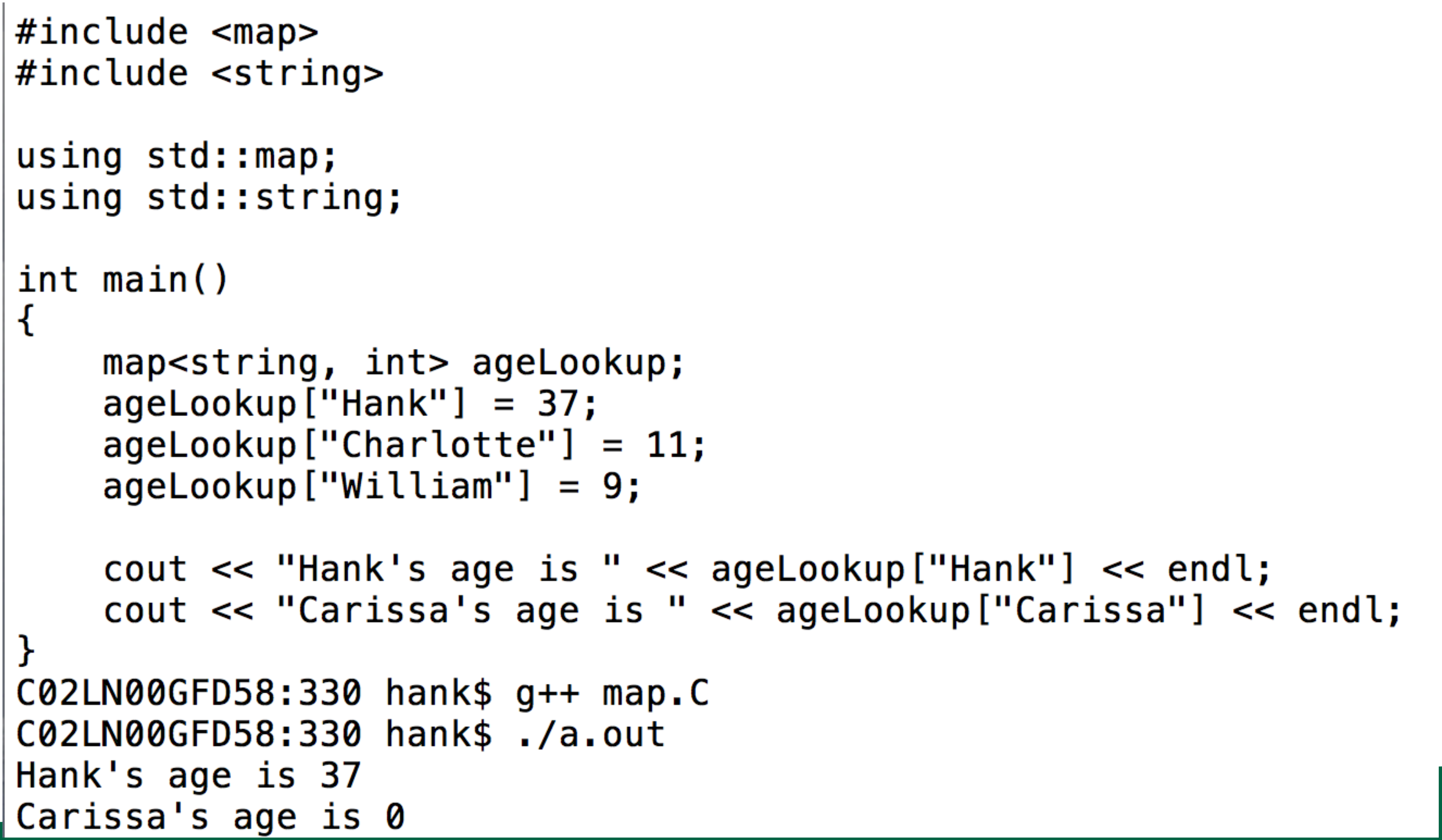


```
#include <vector>

using std::vector;

int main()
{
    vector<int> intArray(2);
    intArray[0] = 0;
    intArray[1] = 1;
    intArray.push_back(1);
    intArray.push_back(2);
    intArray.push_back(3);
    intArray.push_back(5);
    cout << "Size is " << intArray.size() << endl;
    cout << "Last val of Fib is " << intArray[5] << endl;
}
C02LN00GFD58:330 hank$ g++ vector.C
C02LN00GFD58:330 hank$ ./a.out
Size is 6
Last val of Fib is 5
```

std::vector

- Always has the amount of memory you need
- Many STL algorithms work on it
- Memory allocation:
  - If you don't have enough, double it
    - (can be a big overestimation)
- Overhead in access
  - Maybe not a big deal if data-intensive?

STL: map

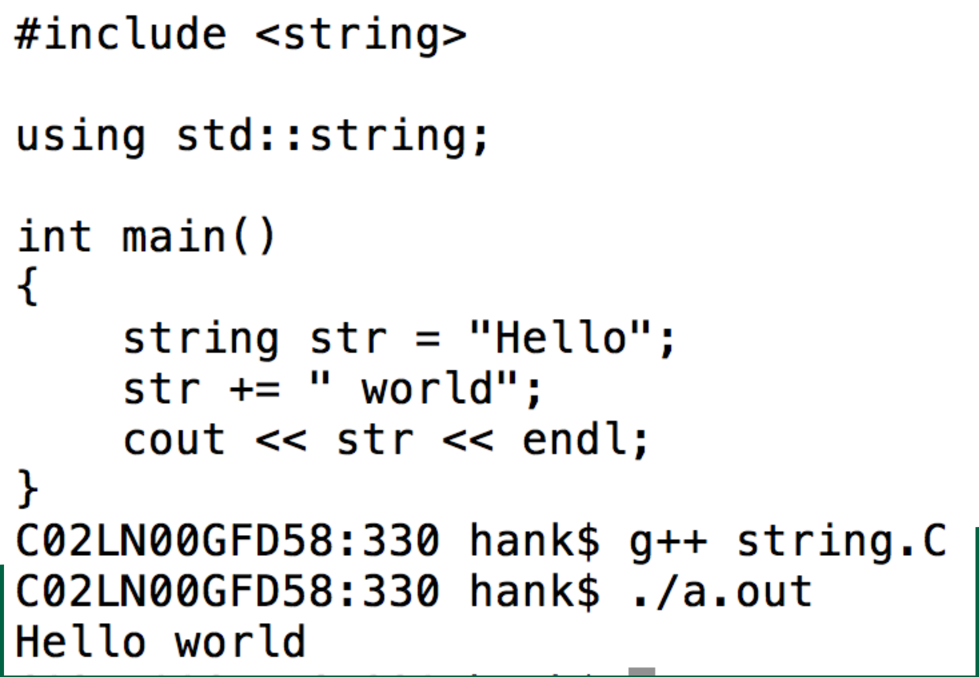


```
#include <map>
#include <string>

using std::map;
using std::string;

int main()
{
    map<string, int> ageLookup;
    ageLookup["Hank"] = 37;
    ageLookup["Charlotte"] = 11;
    ageLookup["William"] = 9;

    cout << "Hank's age is " << ageLookup["Hank"] << endl;
    cout << "Carissa's age is " << ageLookup["Carissa"] << endl;
}
C02LN00GFD58:330 hank$ g++ map.C
C02LN00GFD58:330 hank$ ./a.out
Hank's age is 37
Carissa's age is 0
```

C++ Strings

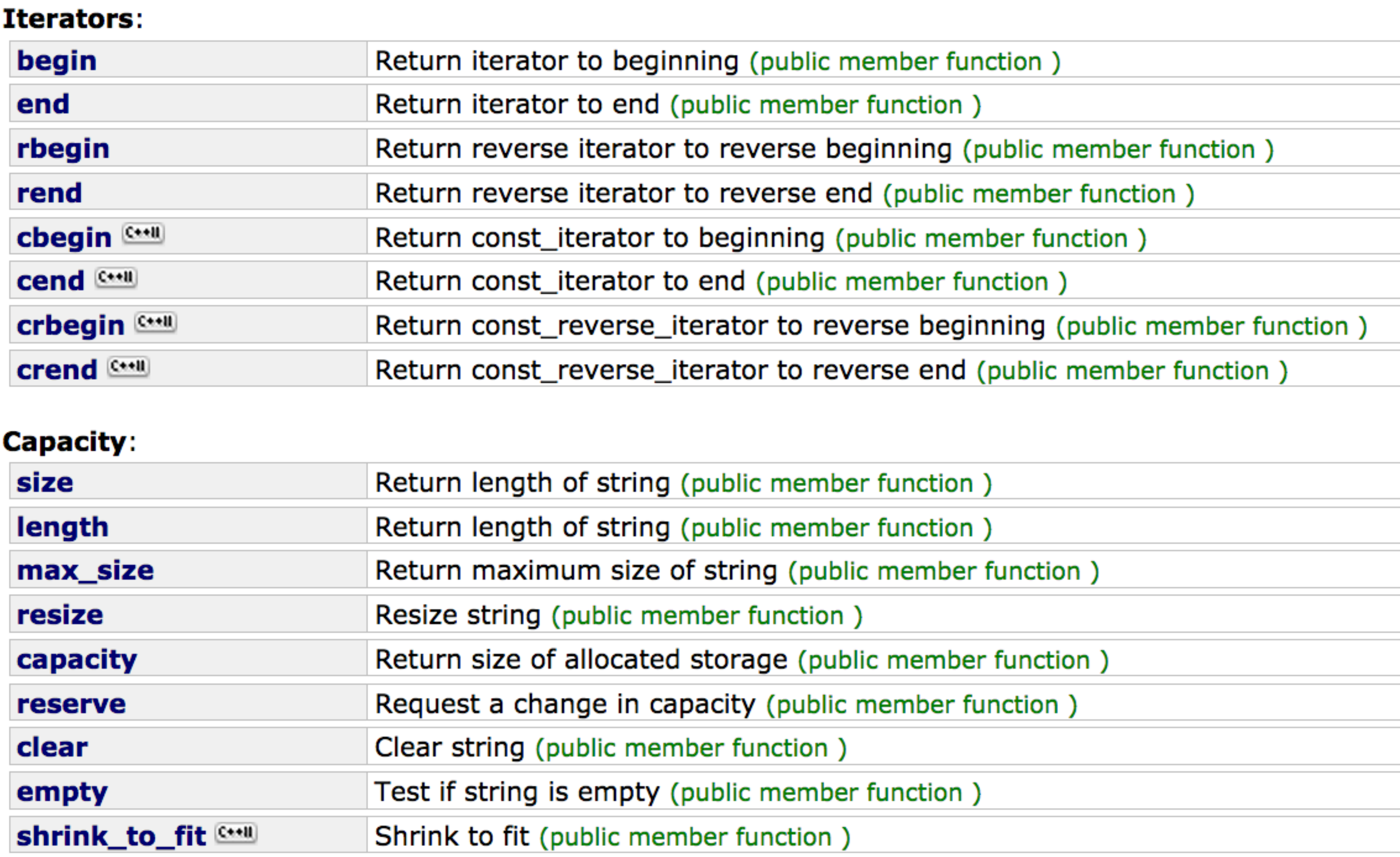
(not a template thing): String

- C++ string class is very useful
- Great implementation of a class that encapsulates a string



```
#include <string>

using std::string;

int main()
{
    string str = "Hello";
    str += " world";
    cout << str << endl;
}
C02LN00GFD58:330 hank$ g++ string.C
C02LN00GFD58:330 hank$ ./a.out
Hello world
```

String methods



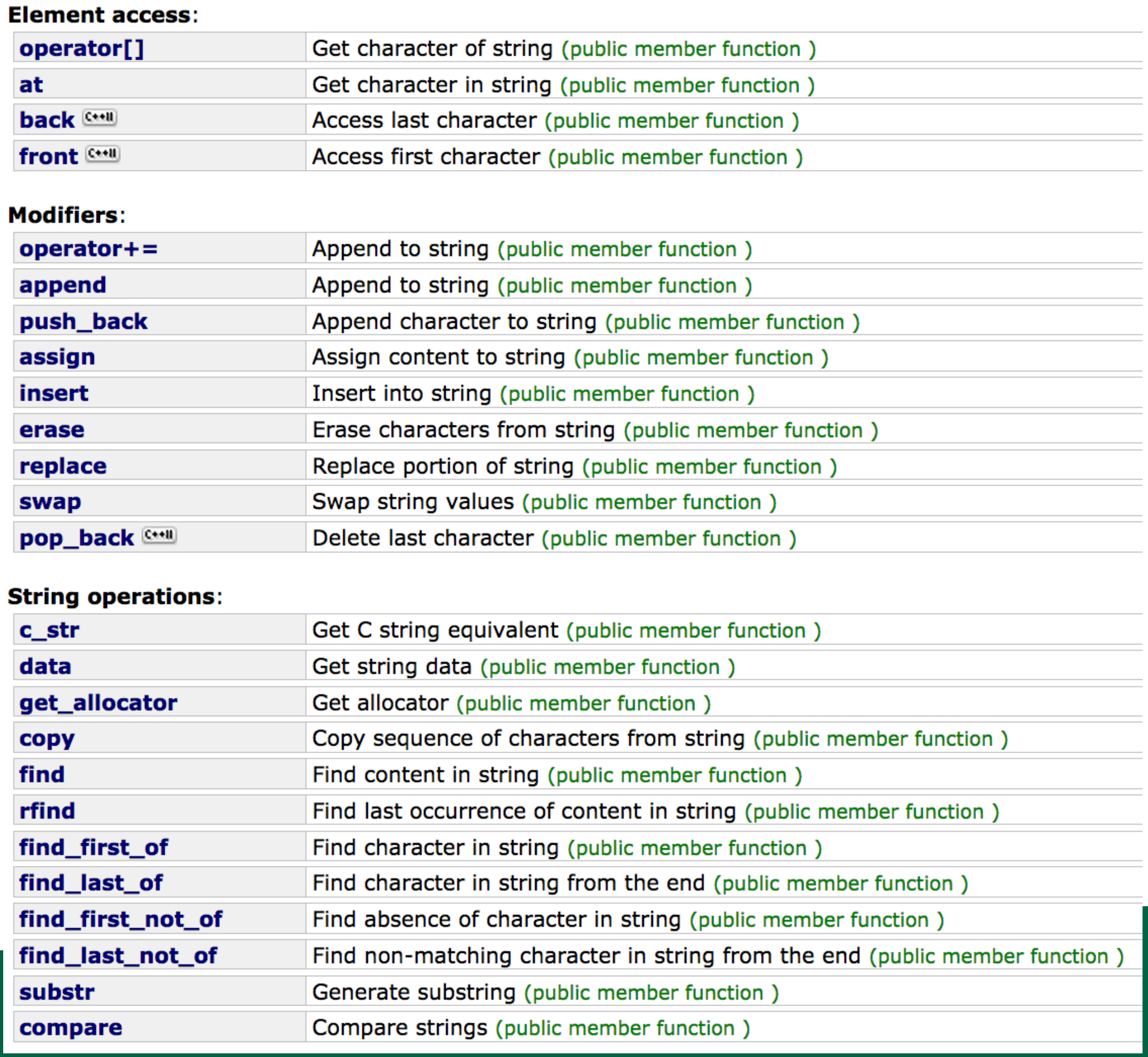
Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin C++11	Return const_iterator to beginning (public member function)
cend C++11	Return const_iterator to end (public member function)
crbegin C++11	Return const_reverse_iterator to reverse beginning (public member function)
crend C++11	Return const_reverse_iterator to reverse end (public member function)

Capacity:

size	Return length of string (public member function)
length	Return length of string (public member function)
max_size	Return maximum size of string (public member function)
resize	Resize string (public member function)
capacity	Return size of allocated storage (public member function)
reserve	Request a change in capacity (public member function)
clear	Clear string (public member function)
empty	Test if string is empty (public member function)
shrink_to_fit C++11	Shrink to fit (public member function)

String methods



Element access:

operator[]	Get character of string (public member function)
at	Get character in string (public member function)
back C++11	Access last character (public member function)
front C++11	Access first character (public member function)

Modifiers:

operator+=	Append to string (public member function)
append	Append to string (public member function)
push_back	Append character to string (public member function)
assign	Assign content to string (public member function)
insert	Insert into string (public member function)
erase	Erase characters from string (public member function)
replace	Replace portion of string (public member function)
swap	Swap string values (public member function)
pop_back C++11	Delete last character (public member function)

String operations:

c_str	Get C string equivalent (public member function)
data	Get string data (public member function)
get_allocator	Get allocator (public member function)
copy	Copy sequence of characters from string (public member function)
find	Find content in string (public member function)
rfind	Find last occurrence of content in string (public member function)
find_first_of	Find character in string (public member function)
find_last_of	Find character in string from the end (public member function)
find_first_not_of	Find absence of character in string (public member function)
find_last_not_of	Find non-matching character in string from the end (public member function)
substr	Generate substring (public member function)
compare	Compare strings (public member function)

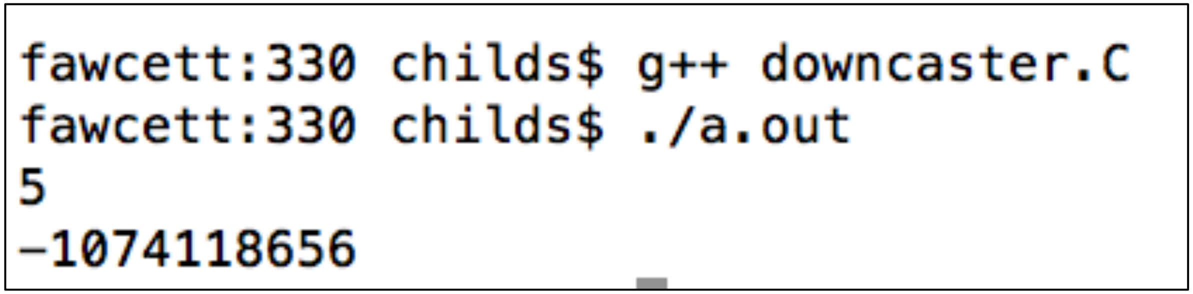
Bonus Topics

Upcasting and Downcasting

- Upcast: treat an object as the base type
  - We do this all the time!
  - Treat a Rectangle as a Shape
- Downcast: treat a base type as its derived type
  - We don't do this one often
  - Treat a Shape as a Rectangle
    - You better know that Shape really is a Rectangle!!

Upcasting and Downcasting

```
class A
{
};

class B : public A
{
  public:
               B() { myInt = 5; };
    void       Printer(void) { cout << myInt << endl; };

  private:
    int        myInt;
};

void Downcaster(A *a)
{
    B *b = (B *) a;
    b->Printer();
}

int main()
{
    A a;
    B b;

    Downcaster(&b);   // no problem
    Downcaster(&a);   // no good
}
```



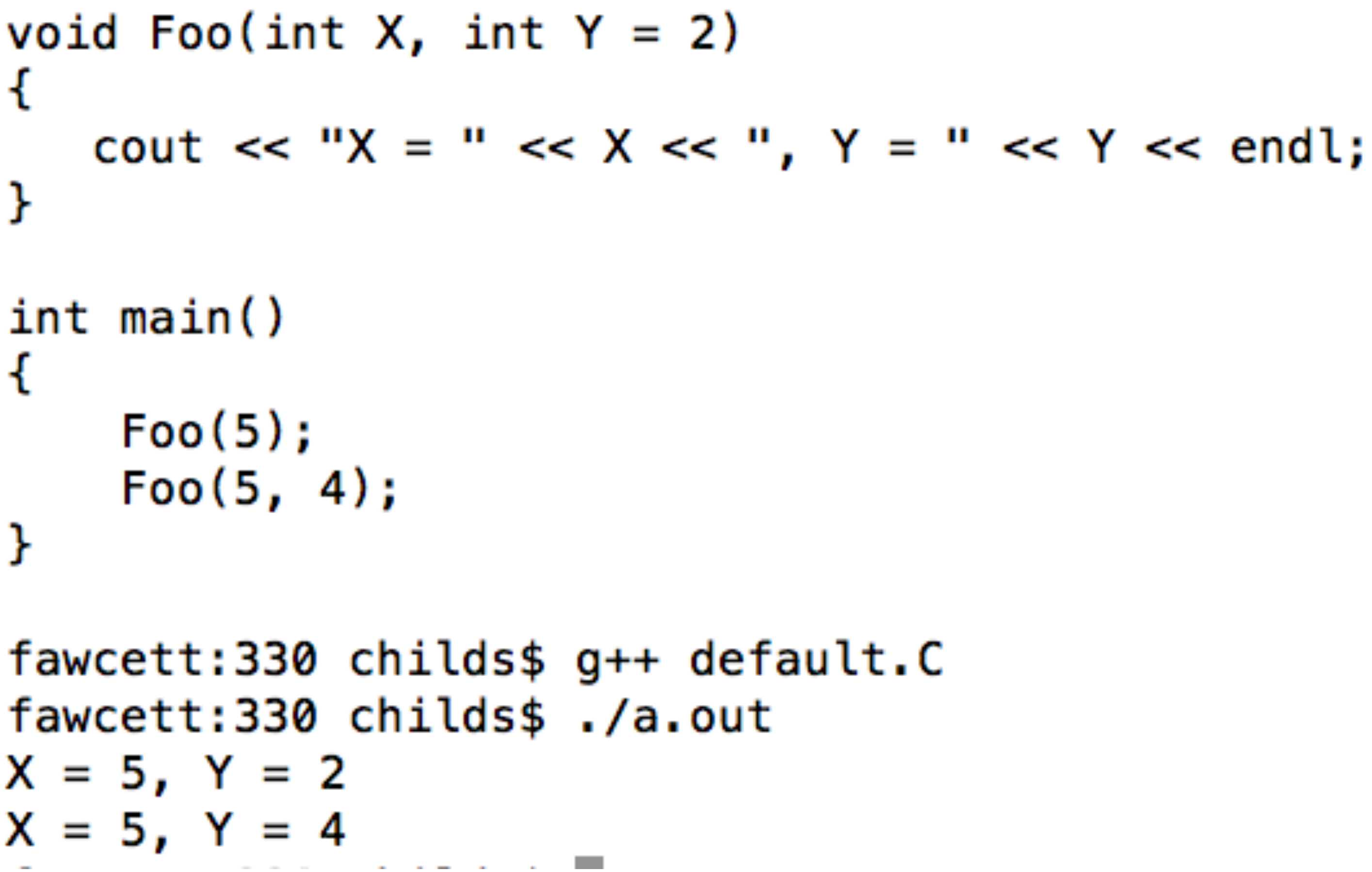
```
fawcett:330 childs$ g++ downcaster.C
fawcett:330 childs$ ./a.out
5
-1074118656
```

what do we get?

Upcasting and Downcasting

- C++ has a built in facility to assist with downcasting: dynamic_cast
- I personally haven’t used it a lot, but it is used in practice
- Ties in to std::exception and polymorphism
- Others: reinterpret_cast, static_cast, const_cast

Default Arguments



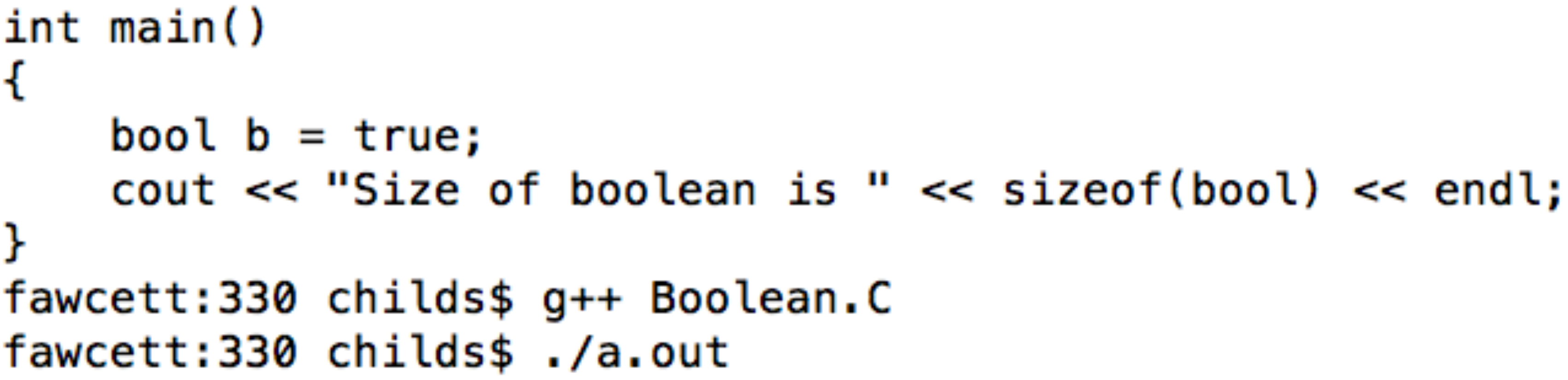
```
void Foo(int X, int Y = 2)
{
   cout << "X = " << X << ", Y = " << Y << endl;
}

int main()
{
    Foo(5);
    Foo(5, 4);
}
```

```
fawcett:330 childs$ g++ default.C
fawcett:330 childs$ ./a.out
X = 5, Y = 2
X = 5, Y = 4
```

default arguments: compiler pushes values on the stack for you if you choose not to enter them

Booleans

- New simple data type: bool (Boolean)
- New keywords: true and false



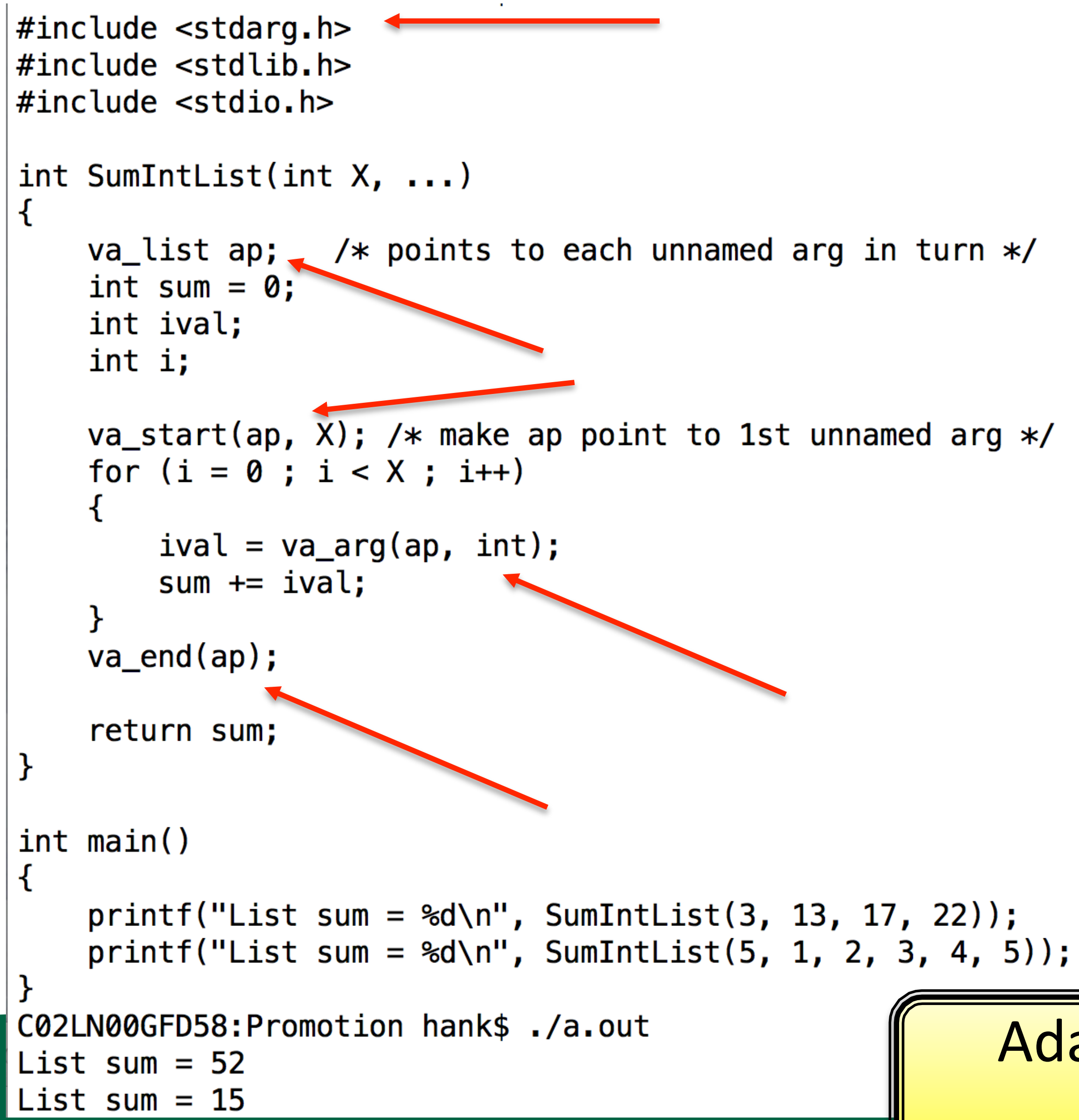
```
int main()
{
    bool b = true;
    cout << "Size of boolean is " << sizeof(bool) << endl;
}
fawcett:330 childs$ g++ Boolean.C
fawcett:330 childs$ ./a.out
```

printf

- Print to stdout
  - printf(“hello world\n”);
  - printf(“Integers are like this %d\n”, 6);
  - printf(“Two floats: %f, %f”, 3.5, 7.0);

Have you ever wondered how printf takes a variable number of arguments?

Variable-Length Argument Lists



```
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>

int SumIntList(int X, ...)
{
    va_list ap;   /* points to each unnamed arg in turn */
    int sum = 0;
    int ival;
    int i;

    va_start(ap, X); /* make ap point to 1st unnamed arg */
    for (i = 0 ; i < X ; i++)
    {
        ival = va_arg(ap, int);
        sum += ival;
    }
    va_end(ap);

    return sum;
}

int main()
{
    printf("List sum = %d\n", SumIntList(3, 13, 17, 22));
    printf("List sum = %d\n", SumIntList(5, 1, 2, 3, 4, 5));
}
C02LN00GFD58:Promotion hank$ ./a.out
List sum = 52
List sum = 15
```

Adapted from Kernigan & Ritchie C book