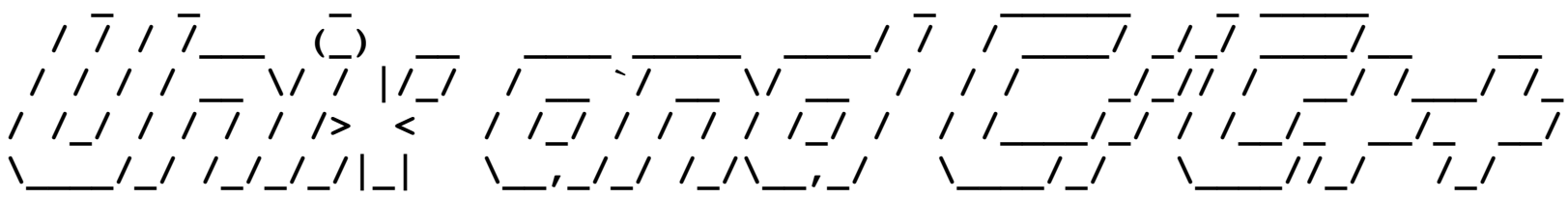
UNIVERSITY OF OREGON

CIS 330:



Lecture 16: potpourri

May 20th, 2016

Hank Childs, University of Oregon

Announcements

- Weekend OH?
- No OH on Tuesday!!
- Extra Credit

Announcements: Rest of Term

- 3G: assigned today, due next Sunday
- 3H, 4A, 4B: assigned next week, nominally due on Friday June 3rd
  - But: you can take until the Final to get them (3H, 4A, 4B) wrapped up (& they will not be late)

Project 3E

- You will need to think about how to accomplish the data flow execution pattern and think about how to extend your implementation to make it work.
- This prompt is vaguer than some previous ones
  - … not all of the details are there on how to do it

Project 3E

```
    blender.SetInput(tbconcat2.GetOutput());
    blender.SetInput2(reader.GetOutput());

    writer.SetInput(blender.GetOutput());

    reader.Execute();
    shrinker1.Execute();
    lrconcat1.Execute();
    tbconcat1.Execute();
    shrinker2.Execute();
    lrconcat2.Execute();
    tbconcat2.Execute();
    blender.Execute();

    writer.Write(argv[2]);
}
```

```
    blender.SetInput(tbconcat2.GetOutput());
    blender.SetInput2(reader.GetOutput());

    writer.SetInput(blender.GetOutput());

    blender.GetOutput()->Update();
    writer.Write(argv[2]);
}
```

Project 3E

- Worth 5% of your grade
- Assigned May 13, due ~~May 20th~~ 21st

Exceptions

- C++ mechanism for handling error conditions
- Three new keywords for exceptions
  - try: code that you "try" to execute and hope there is no exception
  - throw: how you invoke an exception
  - catch: catch an exception … handle the exception and resume normal execution

Exceptions

```
fawcett:330 childs$ cat exceptions.C
#include <iostream>
using std::cout;
using std::endl;

int main()
{
    try
    {
        cout << "About to throw 105" << endl;
        throw 105;
        cout << "Done throwing 105" << endl;
    }
    catch (int &theInt)
    {
        cout << "Caught an int: " << theInt << endl;
    }
}
fawcett:330 childs$ g++ exceptions.C
```

Exceptions: catching multiple types

```
fawcett:330 childs$ cat exceptions2.C
#include <iostream>
using std::cout;
using std::endl;

int main()
{
    try
    {
        cout << "About to throw 105" << endl;
        throw 105;
        cout << "Done throwing 105" << endl;
    }
    catch (int &theInt)
    {
        cout << "Caught an int: " << theInt << endl;
    }
    catch (float &theFloat)
    {
        cout << "Caught a float: " << theFloat << endl;
    }
}
fawcett:330 childs$ g++ exceptions2.C
fawcett:330 childs$ ./a.out
About to throw 105
Caught an int: 105
```

Exceptions: catching multiple types

```
fawcett:330 childs$ cat exceptions3.C
#include <iostream>
using std::cout;
using std::endl;

int main()
{
    try
    {
        cout << "About to throw 10.5" << endl;
        throw 10.5;
        cout << "Done throwing 10.5" << endl;
    }
    catch (int &theInt)
    {
        cout << "Caught an int: " << theInt << endl;
    }
    catch (float &theFloat)
    {
        cout << "Caught a float: " << theFloat << endl;
    }
}
fawcett:330 childs$ g++ exceptions3.C
fawcett:330 childs$ ./a.out
About to throw 10.5
terminate called after throwing an instance of 'double'
Abort trap
```

Exceptions: throwing/catching complex types

```
class MyExceptionType  { };

class MemoryException : public MyExceptionType {};
class FailedAllocationException : public MemoryException {};
class NULLPointerException : public MemoryException {};

class FloatingPointException : public MyExceptionType {};
class DivideByZeroException : public FloatingPointException {};
class OverflowException : public FloatingPointException {};
```

```
void Foo();

int main()
{
    try
    {
        Foo();
    }
    catch (MemoryException &e)
    {
         cout << "I give up" << endl;
    }
    catch (OverflowException &e)
    {
         cout << "I think it is OK" << endl;
    }
    catch (DivideByZeroException &e)
    {
         cout << "The answer is bogus" << endl;
    }
}
```

Exceptions: cleaning up before you return

```
void Foo(int *arr);

int *
Foo2(void)
{
    int  *arr = new int[1000];
    try
    {
        Foo(arr);
    }
    catch (MyExceptionType &e)
    {
         delete [] arr;
         return NULL;
    }

    return arr;
}
```

Exceptions: re-throwing

```
void Foo(int *arr);

int *
Foo2(void)
{
    int  *arr = new int[1000];
    try
    {
        Foo(arr);
    }
    catch (MyExceptionType &e)
    {
         delete [] arr;
         throw e;
    }

    return arr;
}
```

Exceptions: catch and re-throw anything

```
void Foo(int *arr);

int *
Foo2(void)
{
    int  *arr = new int[1000];
    try
    {
        Foo(arr);
    }
    catch (...)
    {
        delete [] arr;
        throw;
    }

    return arr;
}
```

3F

Project 3F in a nutshell

- Logging:
  - infrastructure for logging
  - making your data flow code use that infrastructure
- Exceptions:
  - infrastructure for exceptions
  - making your data flow code use that infrastructure

The webpage has a head start at the infrastructure pieces for you.

Warning about 3F

- My driver program only tests a few exception conditions
- Your stress tests later will test a lot more.
  - Be thorough, even if I'm not testing it

3F timeline

- Assigned today, due Weds

const

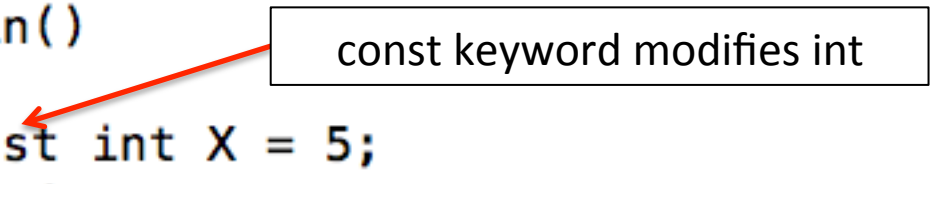
const

- 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;

}
```



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

Efficiency

```
int  NumIterations() { return 10; }

int main()
{
    int         count = 0;
    int         i;
    const int X = 10;
    int         Y = 10;
    for (i = 0 ; i < X ; i++)
        count++;
    for (i = 0 ; i < Y ; i++)
        count++;
    for (i = 0 ; i < NumIterations() ; i++)
        count++;
}
```

Are any of the three for loops faster than the others? Why or why not?

Answer: NumIterations is slowest … overhead for function calls.

Answer: X is probably faster than Y … compiler can do optimizations where it doesn't have to do "i < X" comparisons (loop unrolling)

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.

nst pointer

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

Idea #1:

violates const:

“*P = 3;”

OK:

“int Y = 5; P = &Y;”

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

Idea #2:

violates const:

“int Y = 5; P = &Y;”

OK:

“*P = 3;”

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

const pointers

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

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;"

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

```
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

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;
```

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 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.

globals

globals

- You can create global variables that exist outside functions.

```
fawcett:Documents childs$ cat global1.C

#include <stdio.h>
int X = 5;

int main()
{
    printf("X is %d\n", X);
}

fawcett:Documents childs$ g++ global1.C
fawcett:Documents childs$ ./a.out
X is 5
fawcett:Documents childs$
```

global variables

- global variables are initialized before you enter main

```
fawcett:Documents childs$ cat global2.C

#include <stdio.h>

int Initializer()
{
    printf("In initializer\n");
    return 6;
};

int X = Initializer();

int main()
{
    printf("In main\n");
    printf("X is %d\n", X);
}

fawcett:Documents childs$ g++ global2.C
fawcett:Documents childs$ ./a.out
In initializer
In main
X is 6
```

Storage of global variables…

- global variables are stored in a special part of memory
  - "data segment" (not heap, not stack)
- If you re-use global names, you can have collisions

```
fawcett:Documents childs$ cat file1.C
int X = 6;

int main()
{
}
fawcett:Documents childs$ g++ -c file1.C
fawcett:Documents childs$ cat file2.C
int X = 7;

int doubler(int Y)
{
   return 2*Y;
}
fawcett:Documents childs$ g++ -c file2.C
fawcett:Documents childs$ g++ file1.o file2.o
ld: duplicate symbol _X in file2.o and file1.o
collect2: ld returned 1 exit status
```

Externs: mechanism for unifying global variables across multiple files

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

#include <stdio.h>

int count = 0;

int doubler(int);

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

```
fawcett:330 childs$ cat file2.C
extern int count;

int doubler(int Y)
{
    count++;
    return 2*Y;
}
fawcett:330 childs$ g++ -c file1.C
fawcett:330 childs$ g++ -c file2.C
fawcett:330 childs$ g++ file1.o file2.o
fawcett:330 childs$ ./a.out
count is 2
```

extern: there's a global variable, and it lives in a different file.

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.

```
faw
#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:
    int       numInstances;
};

int main()
{
    MyClass *p = new MyClass[10];
    cout << "Num instances = " << p[0].GetNumInstances() << endl;
    delete [] p;
    cout << "Num instances = " << p[0].GetNumInstances() << endl;
}
fawcett:Downloads childs$ g++ static3.C
fawcett:Downloads childs$ ./a.out
Num instances = 1
Num instances = 0
fawcett:Downloads childs$
```

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()
   virtual  ~MyClass()

   int       GetNumInstan

  private:
   static int       numInstances;
};

i
{
    delete [] p;
    cout << "Num instances = " << p[0].GetNumInstances() << endl;
}
```

```
fawcett:Downloads childs$ g++ static3.C
Undefined symbols:
  "MyClass::numInstances", referenced from:
      MyClass::MyClass()in ccoao8Hf.o
      MyClass::MyClass()in ccoao8Hf.o
      MyClass::GetNumInstances()      in ccoao8Hf.o
      MyClass::~MyClass()in ccoao8Hf.o
      MyClass::~MyClass()in ccoao8Hf.o
      MyClass::~MyClass()in ccoao8Hf.o
      MyClass::~MyClass()in ccoao8Hf.o
ld: symbol(s) not found
collect2: ld returned 1 exit status
```

We have to tell the compiler where to store this static.

What do we get?

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

- I saw this bug quite a few times…

The compiler will sometimes have multiple choices as to which variable you mean.

It has rules to make a decision about which one to use.

This topic is referred to as “scope”.

```
class MyClass
{
  public:
    void SetValue(int);

  private:
    int   X;
};

void MyClass::SetValue(int X)
{
   X = X;
}
```

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 won't compile.

The compiler notices that you have a variable called X that "shadows" the argument called X.

```
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);
}
```

scope

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

So what does it print?

Answer: 3

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);
}
```

What does this one print?

Answer: 2

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);
}
```

What does this one print?

Answer: 1

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);
}
```

What does this one print?

Answer: 0

Scope Rules

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

Pitfall #8

```
#include <stdlib.h>

class Image
{
  public:
                    Image() { width = 0; height = 0; buffer = NULL; };
    virtual        ~Image() { delete [] buffer; };

    void            ResetSize(int width, int height);
    unsigned char  *GetBuffer(void) { return buffer; };

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

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

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

```
int main()
{
    Image img;
    unsigned char *buffer = img.GetBuffer();
    img.ResetSize(1000, 1000);
    for (int i = 0 ; i < 1000 ; i++)
        for (int j = 0 ; j < 1000 ; j++)
            for (int k = 0 ; k < 1000 ; k++)
             buffer[3*(i*1000+j)+k] = 0;
}
```

Shadowing

- Shadowing is a term used to describe a "subtle" scope issue.
  - … i.e., you have created a situation where it is confusing which variable you are referring to

```
class Sink
{
  public:
    void SetInput(Image *i) { input = i; };
  protected:
    Image *input;
};

class Writer : public Sink
{
  public:
    void Write(void)  { /* write input */ };
  protected:
    Image *input;
};

int main()
{
   Writer writer;
   writer.SetInput(image);
   writer.Write();
}
```

Overloading Operators

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

Project 3G

Please read this entire prompt!

Add 4 new filters:

1) Crop
2) Transpose
3) Invert
4) Checkerboard

Add 1 new source:

1) Constant color

Add 1 new sink:

1) Checksum

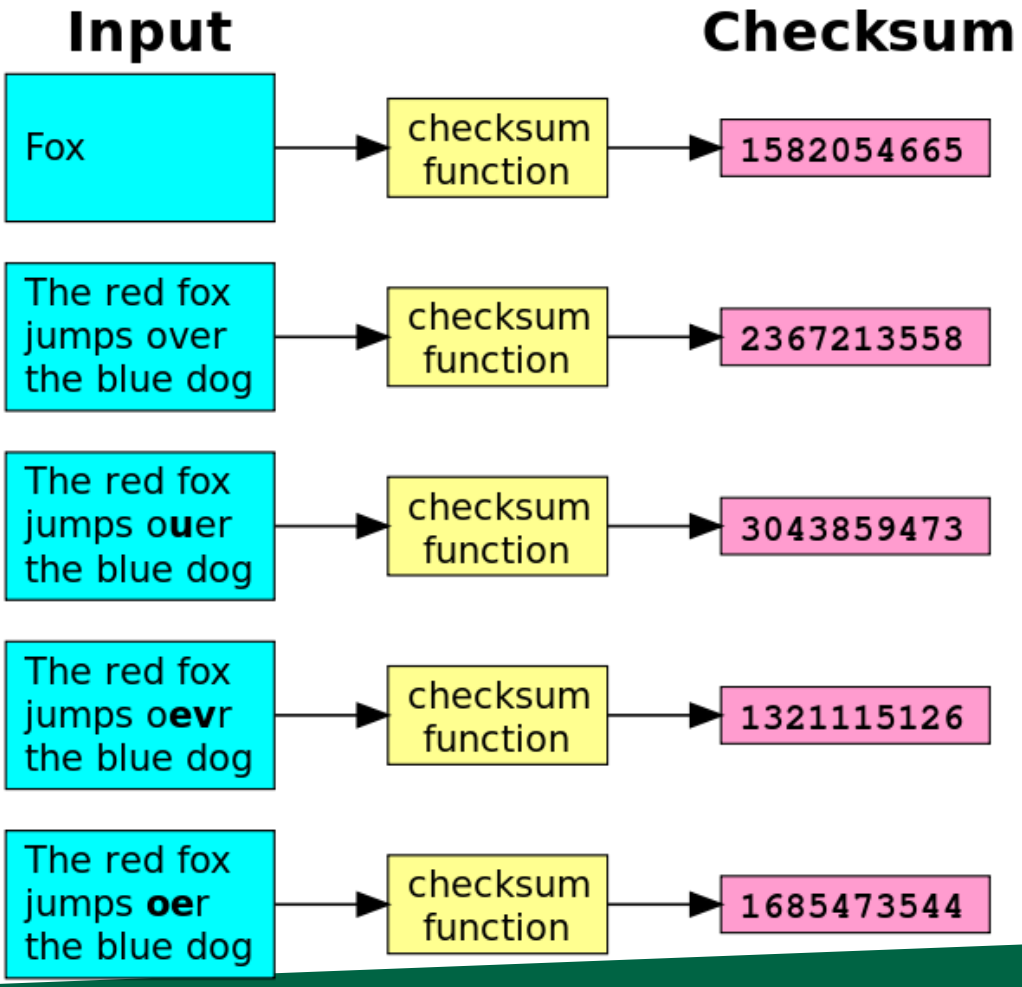
Plus: make the two image inputs in Sink be const pointers.

Assigned today, due next Sunday

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 grind
- 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

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

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
```

Backgrounding

- "&": tell shell to run a job in the background
  - Background means that the shell acts as normal, but the command you invoke is running at the same time.
- "sleep 60" vs "sleep 60 &"

When would backgrounding be useful?

Suspending Jobs

- You can suspend a job that is running

  Press “Ctrl-Z”
- The OS will then stop job from running and not schedule it to run.
- You can then:
  - make the job run in the background.
    - Type “bg”
  - make the job run in the foreground.
    - Type “fg”
      - like you never suspended it at all!!

Unix and Windows difference

- Unix:
  - "\n": goes to next line, and sets cursor to far left
- Windows:
  - "\n": goes to next line (cursor does not go to left)
  - "\m": sets cursor to far left

- Text files written in Windows often don't run well on Unix, and vice-versa
  - There are more differences than just newlines

vi: "set ff=unix" solves this