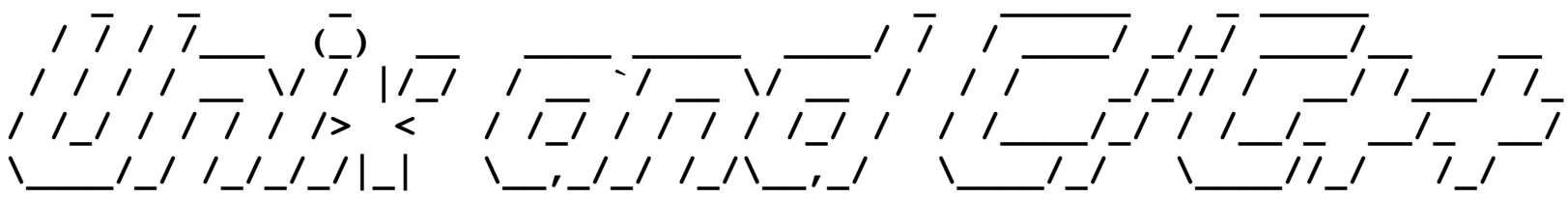
UNIVERSITY OF OREGON

# CIS 330:



# Lecture 14: methods invocations & virtual function tables

May 13th, 2016

Hank Childs, University of Oregon

# Announcements

- Weekend OH?
- Extra Credit
- 441: http://ix.cs.uoregon.edu/~hank/441/proj1F/proj1F.mp4

# Review

# Pure Virtual Functions

- Pure Virtual Function: define a function to be part of the interface for a class, but do not provide a definition.
- Syntax: add "=0" after the function definition.
- This makes the class be "abstract"
  - It cannot be instantiated
- When derived types define the function, then are "concrete"
  - They can be instantiated

# Pure Virtual Functions Example

```
class Shape
{
  public:
    virtual double GetArea(void) = 0;
};

class Rectangle : public Shape
{
  public:
    virtual double GetArea() { return 4; };
};

int main()
{
    Shape s;
    Rectangle r;
}
```

```
fawcett:330 childs$ g++ pure_virtual.C
pure_virtual.C: In function 'int main()':
pure_virtual.C:15: error: cannot declare variable 's' to be of abstract type 'Shape'
pure_virtual.C:2: note:   because the following virtual functions are pure within 'Shape':
pure_virtual.C:4: note:         virtual double Shape::GetArea()
```

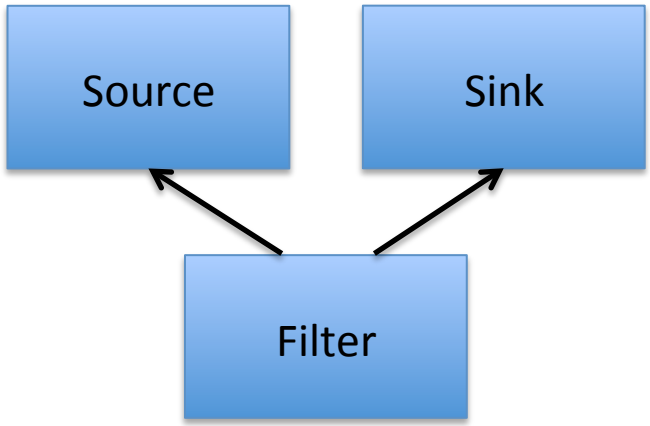
# Data Flow Networks

# Data Flow Overview

- Basic idea:
  - You have many modules
    - Hundreds!!
  - You compose modules together to perform some desired functionality
- Advantages:
  - Customizability
  - Design fosters interoperability between modules to the extent possible

# Data Flow Overview

- Participants:
  - Source: a module that produces data
    - It creates an output
  - Sink: a module that consumes data
    - It operates on an input
  - Filter: a module that transforms input data to create output data

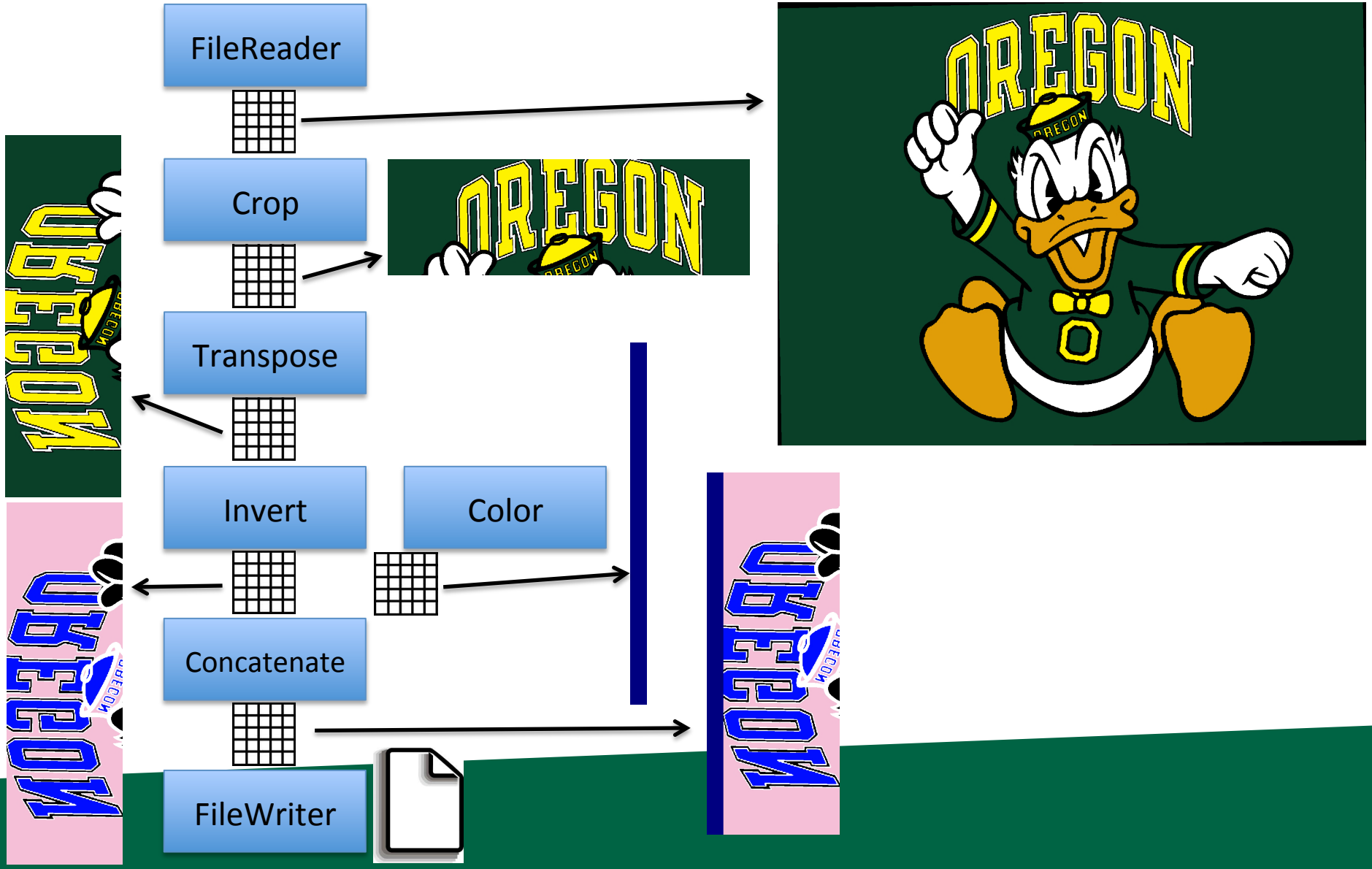



- Nominal inheritance hierarchy:
  - A filter "is a" source
  - A filter "is a" sink

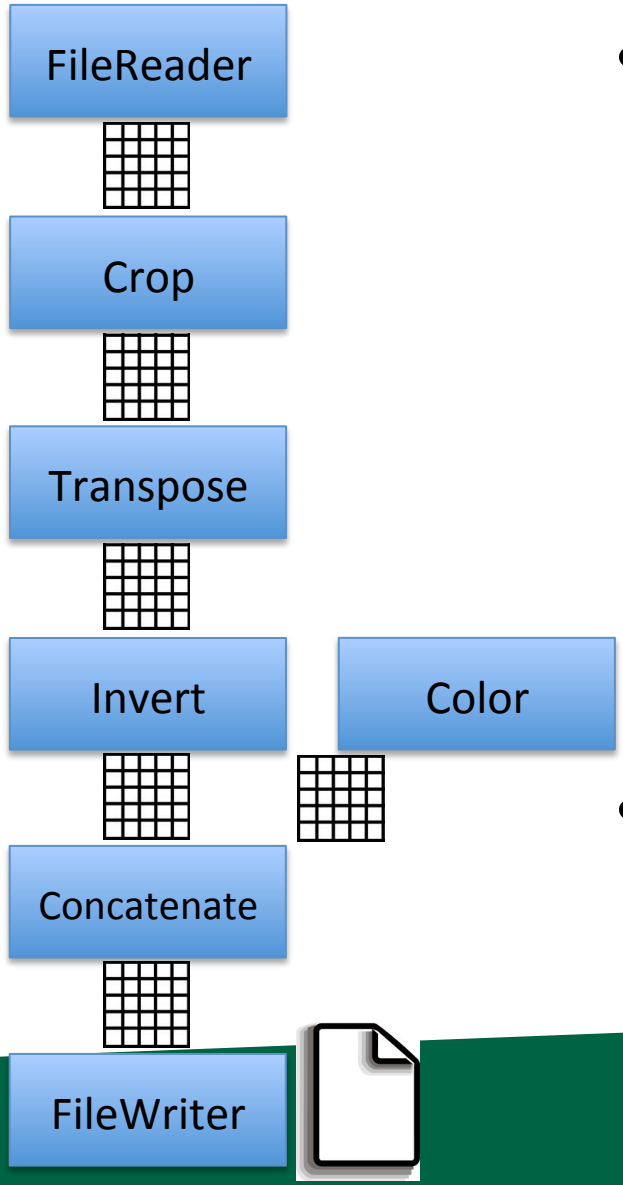
# Example of data flow (image processing)

- Sources:
  - FileReader: reader from file
  - Color: generate image with one color
- Filters:
  - Crop: crop image, leaving only a sub-portion
  - Transpose: view image as a 2D matrix and transpose it
  - Invert: invert colors
  - Concatenate: paste two images together
- Sinks:
  - FileWriter: write to file

# Example of data flow (image processing)



# Example of data flow (image processing)




- Participants:
  - Source: a module that produces data
    - It creates an output
  - Sink: a module that consumes data
    - It operates on an input
  - Filter: a module that transforms input data to create output data
- Pipeline: a collection of sources, filters, and sinks connected together

# Project 3C

# Project 3C

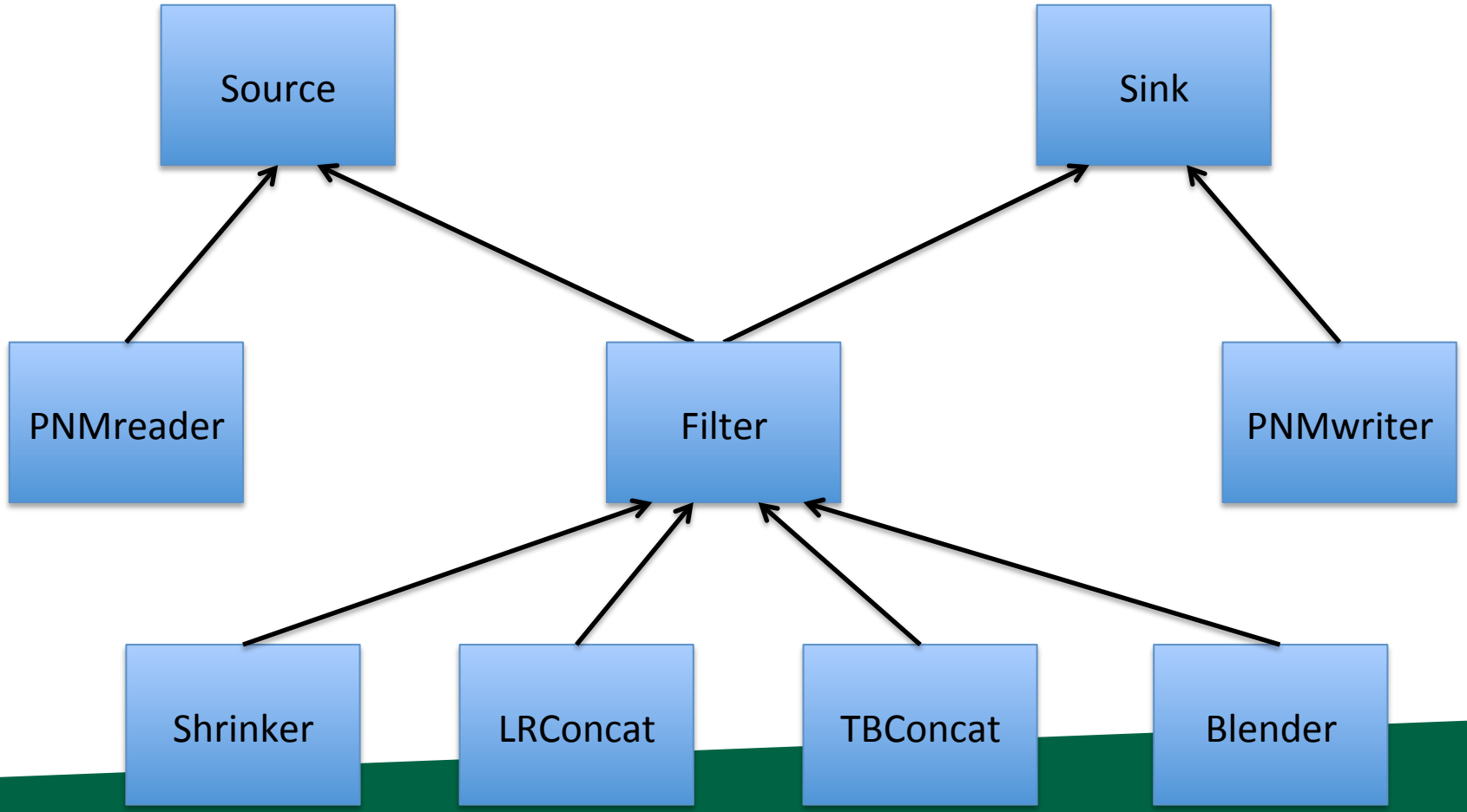
CIS 330: Project #3C
Assigned: May 7th, 2016
Due May 17th, 2016
(which means submitted by 6am on May 18th, 2016)
Worth 7% of your grade

*Please read this entire prompt!*

Assignment: Change your 3B project to be object-oriented.

3D will be due on May 17 as well.
BUT: you can skip 3D.
You get 0/3 points.
But you don't need 3D to do 3E-3I.

# Assignment: make your code base be data flow networks with OOP



# More C++

# C++ lets you define operators

- You declare a method that uses an operator in conjunction with a class
  - +, -, /, !, ++, etc.
- You can then use your 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.

You can also do this with functions.

# Example of operator overloading

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

    MyInt& operator++();

    int     Ge

  protected:
    int     my
};

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

Declare operator ++ will be overloaded for MyInt

We will learn more about operator overloading later in the quarter.

```
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.

Define operator ++ for MyInt

# New operators: << and >>

- "<<": Insertion operator
- ">>": Extraction operator
  - Operator overloading: you can define what it means to insert or extract your object.

- Often used in conjunction with "streams"
  - Recall our earlier experience with C streams
    - stderr, stdout, stdin
  - Streams are communication channels

# Putting it all together

```
fawcett:330 childs$ cat print.c
#include <stdio.h>

int main()
{
    printf("The answer is: ");
    printf("%d", 8);
    printf("\n");
}
fawcett:330 childs$ gcc print.c
fawcett:330 childs$ ./a.out
The answer is: 8
```

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

int main()
{
    std::cout << "The answer is: ";
    std::cout << 8;
    std::cout << "\n";
}
fawcett:330 childs$ g++ printCPP.C
fawcett:330 childs$ ./a.out
The answer is: 8
```

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

int main()
{
    printf("The answer is: %d\n", 8);
}
fawcett:330 childs$ g++ print.C
fawcett:330 childs$
```

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

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

int main()
{
    cout << "The answer is: " << 8 << endl;
}
fawcett:330 childs$ g++ printCPP.C
fawcett:330 childs$
```

# Three pre-defined streams

- cout <= => fprintf(stdout, …
- cerr <= => fprintf(stderr, …
- cin <= => fscanf(stdin, …

# fstream

- ifstream: input stream that does file I/O
- ofstream: output stream that does file I/O

- Not lecturing on this, since it follows from:
  - C file I/O
  - C++ streams

http://www.tutorialspoint.com/cplusplus/cpp_files_streams.htm

# Project 3D

- Assigned: today, 5/11
- Due: Tuesday, 5/17
- Important: if you skip this project, you will still be able to do future projects (3E, 3F, etc)
- Assignment:
  - Write PNMreaderCPP and PNMwriterCPP … new version of the file reader and writer that use fstream.

# Inline function

- inlined functions:
  - hint to a compiler that can improve performance
  - basic idea: don't actually make this be a separate function that is called
    - Instead, just pull the code out of it and place it inside the current function
  - new keyword: inline

```
inline int doubler(int X)
{
    return 2*X;
}

int main()
{
    int Y = 4;
    int Z = doubler(Y);
}
```

The compiler sometimes refuses your inline request (when it thinks inlining won't improve performance), but it does it silently.

# Inlines can be automatically done within class definitions

- Even though you don't declare this as inline, the compiler treats it as an inline

```
class MyDoublerClass
{
    int doubler(int X) { return 2*X; };
};
```

# You should only do inlines within header files

```
fawcett:330 childs$ cat mydoubler.h
#ifndef MY_DOUBLER_H
#define MY_DOUBLER_H

class MyDoubler
{
  public:
    int Doubler(int X) { return 2*X; };
};


#endif
```

```
fawcett:330 childs$ cat mydoubler2.h
#ifndef MY_DOUBLER_H
#define MY_DOUBLER_H

class MyDoubler
{
  public:
};

int
MyDoubler::Doubler(int X)
{
   return 2*X;
}

#endif
```

Left: function is inlined in every .C that includes it … no problem

Right: function is defined in every .C that includes it … duplicate symbols

# New Content

# How C++ Does Methods

# “this”: pointer to current object

- From within any struct’s method, you can refer to the current object using “this”

```
TallyCounter::TallyCounter(int c)
{
    count = c;
}

     <------>

TallyCounter::TallyCounter(int c)
{
    this->count = c;
}
```

# How methods work under the covers (1/4)

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

    friend void   FriendIncrementFunction(MyIntClass *);
    int           GetMyInt() { return myInt; };

  protected:
    int           myInt;
};

void
FriendIncrementFunction(MyIntClass *mic)
{
    mic->myInt++;
}

int main()
{
    MyIntClass MIC(12);
    FriendIncrementFunction(&MIC);
    FriendIncrementFunction(&MIC);
    cout << "My int is " << MIC.GetMyInt() << endl;
}
```

```
fawcett:330 childs$ g++ this.C
fawcett:330 childs$ ./a.out
My int is 14
fawcett:330 childs$
```

# How methods work under the covers (2/4)

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

    friend void     FriendIncrementFunction(MyIntClass *);
    int             GetMyInt() { return myInt; };

  protected:
    int             myInt;
};

void
FriendIncrementFunction(MyIntClass *mic)
{
    mic->myInt++;
}

int main()
{
    MyIntClass MIC(12);
    FriendIncrementFunction(&MIC);
    FriendIncrementFunction(&MIC);
    cout << "My int is " << MIC.GetMyInt() << endl;
}
```

| Addr. | Variable | Value |
|---|---|---|
| 0x8000 | MIC/ myInt | 12 |

| Addr. | Variable | Value |
|---|---|---|
| 0x8000 | MIC/ myInt | 12 |
| 0x8004 | mic | 0x8000 |

# How methods work under the covers (3/4)

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

    friend void   FriendIncrementFunction(MyIntClass *);
    void          IncrementMethod(void);
    int           GetMyInt() { return myInt; };

  protected:
    int           myInt;
};

void
FriendIncrementFunction(MyIntClass *mic)
{
    mic->myInt++;
}

void
MyIntClass::IncrementMethod(void)
{
    this->myInt++;
}

int main()
{
    MyIntClass MIC(12);
    FriendIncrementFunction(&MIC);
    MIC.IncrementMethod();
    cout << "My int is " << MIC.GetMyInt() << endl;
}
```

```
fawcett:330 childs$ g++ this.C
fawcett:330 childs$ ./a.out
My int is 14
fawcett:330 childs$
```

# How methods work under the covers (4/4)

The compiler secretly slips "this" onto the stack whenever you make a method call.

It also automatically changes "myInt" to this->myInt in methods.

```
class  MyIntClass
{
...
Fri
{
    mic->myInt++;
}

void
MyIntClass::IncrementMethod(void)
{
    this->myInt++;
}

int main()
{
    MyIntClass MIC(12);
    FriendIncrementFunction(&MIC);
    MIC.IncrementMethod();
    cout << "My int is " << MIC.GetMyInt() << endl;
}
```

| Addr. | Variable | Value |
|---|---|---|
| 0x8000 | MIC/ myInt | 12 |

| Addr. | Variable | Value |
|---|---|---|
| 0x8000 | MIC/ myInt | 12 |
| 0x8004 | mic | 0x8000 |

| Addr. | Variable | Value |
|---|---|---|
| 0x8000 | MIC/ myInt | 12 |
| 0x8004 | this | 0x8000 |

# Virtual Function Tables

# Virtual functions

- Virtual function: function defined in the base type, but can be re-defined in derived type.
- When you call a virtual function, you get the version defined by the derived type

# Virtual functions: example

```
128-223-223-72-wireless:330 hank$ cat virtual.C
#include <stdio.h>

struct SimpleID
{
   int id;
   virtual int GetIdentifier() { return id; };
};

struct ComplexID : SimpleID
{
   int extraId;
   virtual int GetIdentifier() { return extraId*128+id; };
};

int main()
{
   ComplexID cid;
   cid.id = 3;
   cid.extraId = 3;
   printf("ID = %d\n", cid.GetIdentifier());
}
128-223-223-72-wireless:330 hank$ g++ virtual.C
128-223-223-72-wireless:330 hank$ ./a.out
ID = 387
```

# Picking the right virtual function

```
class A
{
  public:
    virtual const char *GetType() { return "A"; };
};

class B : public A
{
  public:
    virtual const char *GetType() { return "B"; };
};

int main()
{
    A a;
    B b;

    cout << "a is " << a.GetType() << endl;
    cout << "b is " << b.GetType() << endl;
}
```

```
fawcett:330 childs$ g++ virtual.C
fawcett:330 childs$ ./a.out
```

??????

It seems like the compiler should be able to figure this out ...
it knows that a is of type A
and
it knows that b is of type B

# Picking the right virtual function

```
class A
{
  public:
    virtual const char *GetType() { return "A"; };
};

class B : public A
{
  public:
    virtual const char *GetType() { return "B"; };
};

void
ClassPrinter(A *ptrToA)
{
    cout << "ptr points to a " << ptrToA->GetType() << endl;
}

int main()
{
    A a;
    B b;

    ClassPrinter(&a);
    ClassPrinter(&b);
}

fawcett:330 childs$ g++ virtual2.C
fawcett:330 childs$ ./a.out
                ??????
```

So how to does the compiler know?

How does it get “B” for “b” and “A” for “a”?

# Virtual Function Table

- Let C be a class and X be an instance of C.
- Let C have 3 virtual functions & 4 non-virtual functions
- C has a hidden data member called the “virtual function table”
- This table has 3 rows
  - Each row has the correct definition of the virtual function to call for a “C”.
- When you call a virtual function, this table is consulted to locate the correct definition.

# Showing the existence of the virtual function pointer with sizeof()

```
class A
{
  public:
    virtual
};

class B : pub
{
  public:
    virtual
};

class C
{
  public:
    const char *GetType() { return "C"; };
};

int main()
{
    A a;
    B b;

    cout << "Size of A is " << sizeof(A) << endl;
    cout << "Size of a pointer is " << sizeof(int *) << endl;
    cout << "Size of C is " << sizeof(C) << endl;
}
```

empty objects have size of 1? why?!?

Answer: so every object has a unique address.

```
fawcett:330 childs$ ./a.out
Size of A is 8
Size of a pointer is 8
Size of C is 1
```

what will this print?

# Virtual Function Table

- Let C be a class and X be an instance of C.
- Let C have 3 virtual functions & 4 non-virtual functions
- Let D be a class that inherits from C and Y be an instance of D.
  - Let D add a new virtual function
- D's virtual function table has 4 rows
  - Each row has the correct definition of the virtual function to call for a "D".

# More notes on virtual function tables

- There is one instance of a virtual function table for each class
  - Each instance of a class shares the same virtual function table
- Easy to overwrite (i.e., with a memory error)
  - And then all your virtual function calls will be corrected
  - Don't do this! ;)

# Virtual function table: example

CIS 330: Project #2C
Assigned: April 17th, 2014
Due April 24th, 2014
(which means submitted by 6am on April 25th, 2014)
Worth 6% of your grade

*Please read this entire prompt!*

Assignment: You will implement subtypes with C.

1) Make a union called ShapeUnion with the three types (Circle, Rectangle, Triangle).
2) Make a struct called FunctionTable that has pointers to functions.
3) Make an enum called ShapeType that identifies the three types.
4) Make a struct called Shape that has a ShapeUnion, a ShapeType, and a FunctionTable.
5) Modify your 9 functions to deal with Shapes.
6) Integrate with the new driver function. Test that it produces the correct output.

# Virtual function table: example

```
class Shape
{
    virtual double GetArea() = 0;
    virtual void   GetBoundingBox(double *) = 0;
};

class Rectangle : public Shape
{
  public:
                   Rectangle(double, double, double, double);
    virtual double GetArea();
    virtual void   GetBoundingBox(double *);
  protected:
    double minX, maxX, minY, maxY;
};

class Triangle : public Shape
{
  public:
                   Triangle(double, double, double, double);
    virtual double GetArea();
    virtual void   GetBoundingBox(double *);
  protected:
    double pt1X, pt2X, minY, maxY;
};
```

# Questions

- What does the virtual function table look like for a Shape?

```
typedef struct
{
    double (*GetArea)(Shape *);
    void   (*GetBoundingBox)(Shape *, double *);
} VirtualFunctionTable;
```

- What does Shape's virtual function table look like?
  - Trick question: Shape can't be instantiated, precisely because you can't make a virtual function table
    - abstract type due to pure virtual functions

# Questions

- What is the virtual function table for Rectangle?

```
c->ft.GetArea = GetRectangleArea;
c->ft.GetBoundingBox = GetRectangleBoundingBox;
```

- (this is a code fragment from my 2C solution)

# Calling a virtual function

- Let X be an instance of class C.
- Let the virtual function be the 4th function
- Let the arguments to the virtual function be an integer Y and a float Z.
- Then call:

```
(X.vptr[3])(&X, Y, Z);
```

The 4th virtual function has index 3 (0-indexing)

The pointer to the virtual function pointer (often called a vptr) is a data member of X

Secretly pass "this" as first argument to method

# Inheritance and Virtual Function Tables

```
class A
{
  public:
    virtual void Foo1();
    virtual void Foo2();
};

class C
{
  public:
    virtual void Foo1();
    virtual void Foo2();
    virtual void Foo3();
};
```

This whole scheme gets much harder with virtual inheritance, and you have to carry around multiple virtual function tables.

| A | |
|---|---|
| | Location of Foo1 |
| Foo2 | Location of Foo2 |

| | |
|---|---|
| Foo1 | Location of Foo1 |
| Foo2 | Location of Foo2 |

| C | |
|---|---|
| Foo1 | Location of Foo1 |
| Foo2 | Location of Foo2 |
| Foo3 | Location of Foo3 |

Same as B's
This is how you can treat a C as a B

# Virtual Function Table: Summary

- Virtual functions require machinery to ensure the correct form of a virtual function is called
- This is implemented through a virtual function table
- Every instance of a class that has virtual functions has a pointer to its class's virtual function table
- The virtual function is called via following pointers
  - Performance issue

# Now show Project 2D in C++

- Comment:
  - C/C++ great because of performance
  - Performance partially comes because of a philosophy of not adding “magic” to make programmer’s life easier
  - C has very little pixie dust sprinkled in
    - Exception: ‘\0’ to terminate strings
  - C++ has more
    - Hopefully this will demystify one of those things (virtual functions)

# vptr.C

```
fawcett:vptr childs$ cat vptr.C
#include <iostream>
using std::cerr;
using std::endl;

class Shape
{
  public:
     int s;
     virtual double GetArea() = 0;
     virtual void   GetBoundingBox(double *) = 0;
};

class Triangle : public Shape
{
  public:
     virtual double GetArea() { cerr << "In GetArea for Triangle" << endl; return 1;};
     virtual void GetBoundingBox(double *) { cerr << "In GetBBox for Triangle" << endl; };
};

class Rectangle : public Shape
{
  public:
     virtual double GetArea() { cerr << "In GetArea for Rectangle" << endl; return 2; };
     virtual void GetBoundingBox(double *) { cerr << "In GetBBox for Rectangle" << endl; };
};

struct VirtualFunctionTable
{
     double (*GetArea)(Shape *);
     void (*GetBoundingBox)(Shape *, double *);
};


int main()
{
    Rectangle r;
    cerr << "Size of rectangle is " << sizeof(r) << endl;

    VirtualFunctionTable *vft = *((VirtualFunctionTable**)&r);
    cerr << "Vptr = " << vft << endl;
    double d = vft->GetArea(&r);
    cerr << "Value = " << d << endl;

    double bbox[4];
    vft->GetBoundingBox(&r, bbox);
}
```

# 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

- Worth 5% of your grade
- Assigned today, due May 20th

# Pitfalls

# Pitfall #1

```
void AllocateBuffer(int w, int h, unsigned char **buffer)
{
    *buffer = new unsigned char[3*w*h];
}

int main()
{
    int w = 1000, h = 1000;
    unsigned char *buffer = NULL;
    AllocateBuffer(w, h, &buffer);
}
```

This is using call-by-value, not call-by-reference.

# Pitfall #2

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

Image *ReadFromFile(char *filename)
{
    Image *rv = NULL;

    /*  OPEN FILE, descriptor = f */
    /*    ...     */
    /*   set up width w,  and height h */
    /*    ...     */

    rv = malloc(sizeof(Image));
    rv->width  = w;
    rv->height = h;
    fread(rv->buffer, sizeof(unsigned char), w*h, f);
}
```

# Pitfall #3

- int *s = new int[6*sizeof(int)];

# Pitfall #4

```
int main()
{
    // new black image
    int height = 1000, width = 1000;
    unsigned char *buffer = new unsigned char[3*width*height];
    for (int i = 0 ; i < sizeof(buffer) ; i++)
    {
        buffer[i] = 0;
    }
}
```

- Assume:

  int *X = new int[100];
- What is sizeof(X)?
- What is sizeof(*X)?

# Pitfall #5

```
/* struct definition */
struct Image
{
   /* data members */
};

/* prototypes */
void WriteImage(Image *, const char *);


/* main */
int main()
{
    Image *img = NULL;
    /* set up Image */
    const char *filename = "out.pnm";
    WriteImage(img, filename);
}

/* WriteImage function */
void WriteImage(char *filename, Image *img)
{
   /* code to write img to filename */
}
```

```
fawcett:330 childs$ g++ write_image.c
Undefined symbols:
  "WriteImage(Image*, char const*)", referenced from:
      _main in ccSjC6w2.o
ld: symbol(s) not found
collect2: ld returned 1 exit status
```

# (not-a-)Pitfall #6

```
unsigned char* Image::getPixel(int i, int j) {
    int pixStart = 3*i*this->width+3+j;
    unsigned char *pixel = new unsigned char[3];
    pixel[0] = this->data[pixStart];
    pixel[1] = this->data[pixStart + 1];
    pixel[2] = this->data[pixStart + 2];
    return pixel;
}
```

```
unsigned char* Image::getPixel(int i, int j) {
    int pixStart = 3*i*this->width+3+j;
    return this->data+pixStart;
}
```

Top requires memory allocation / deletion, and does extra copy.

# Pitfall #7

- For objects on the stack, the destructors are called when a function goes out of scope
  - You may have a perfectly good function, but it seg-faults on return
- Especially tricky for main
  - program ran to completion, and crashed at the very end

# 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];
}
```

```
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;
}
```

# Bonus Topics

# 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!!

# Web pages

- ssh –l <user name> ix.cs.uoregon.edu
- cd public_html
- put something in index.html
- → it will show up as

  http://ix.cs.uoregon.edu/~<username>

# Web pages

- You can also exchange files this way
  - scp file.pdf <username>@ix.cs.uoregon.edu:~/public_html
  - point people to http://ix.cs.uoregon.edu/~<username>/file.pdf

Note that ~/public_html/dir1 shows up as http://ix.cs.uoregon.edu/~<username>/dir1

("~/dir1" is not accessible via web)

# 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