UNIVERSITY OF OREGON

CIS 330:

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

Lecture 21: Review

June 8th, 2018

Hank Childs, University of Oregon

Your Grade

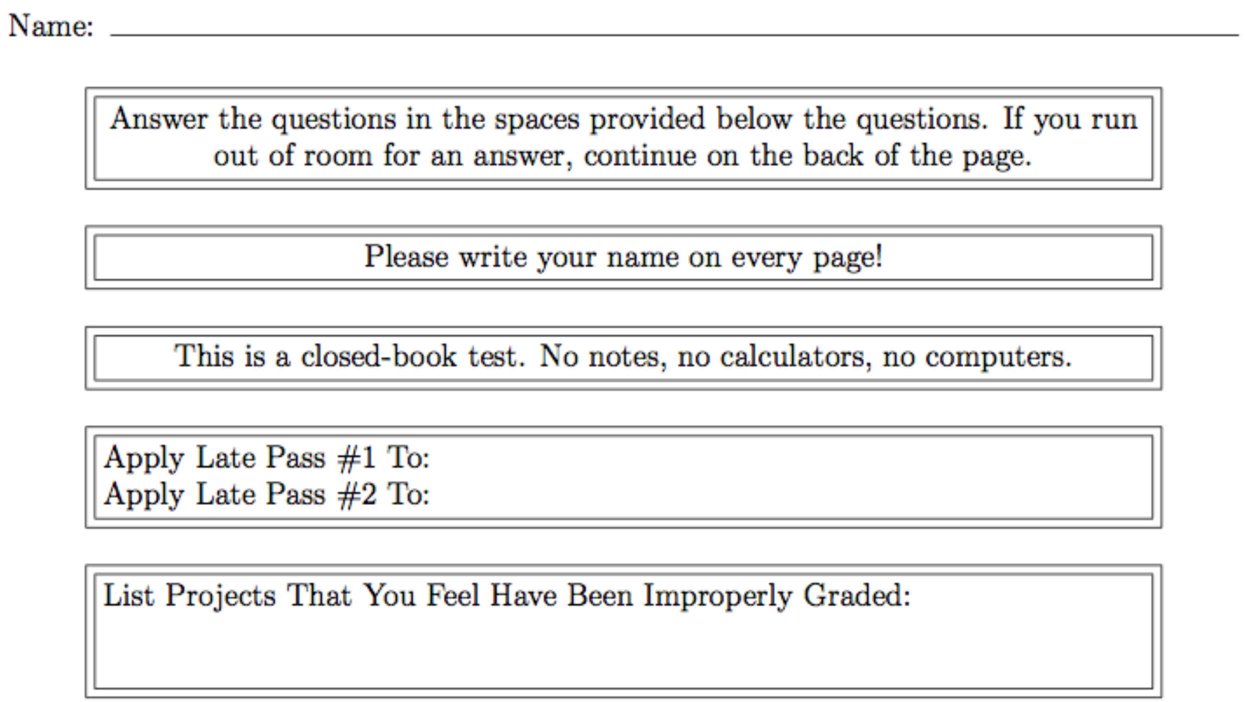
- Projects: 70%
- Final: 30%
- If you have 70% right now, you have already passed the class.
  - Although if you bomb the final, I may meet with you and ask you to code

Late Pass Policy

- You have 2 "late passes."
- Late passes allow you to turn in your project (specifically a sub-project, i.e., project 1A) two days after the due date for full credit.
- You may also use two late passes on one assignment and get a four day extension.
  - For example, you submit a project that was due on a Wednesday on Friday (i.e., two days later) and get full credit if you use one late pass.

Late Pass Decisions

- There will be a spot on the final



Name: ________________________________

Answer the questions in the spaces provided below the questions. If you run out of room for an answer, continue on the back of the page.

Please write your name on every page!

This is a closed-book test. No notes, no calculators, no computers.

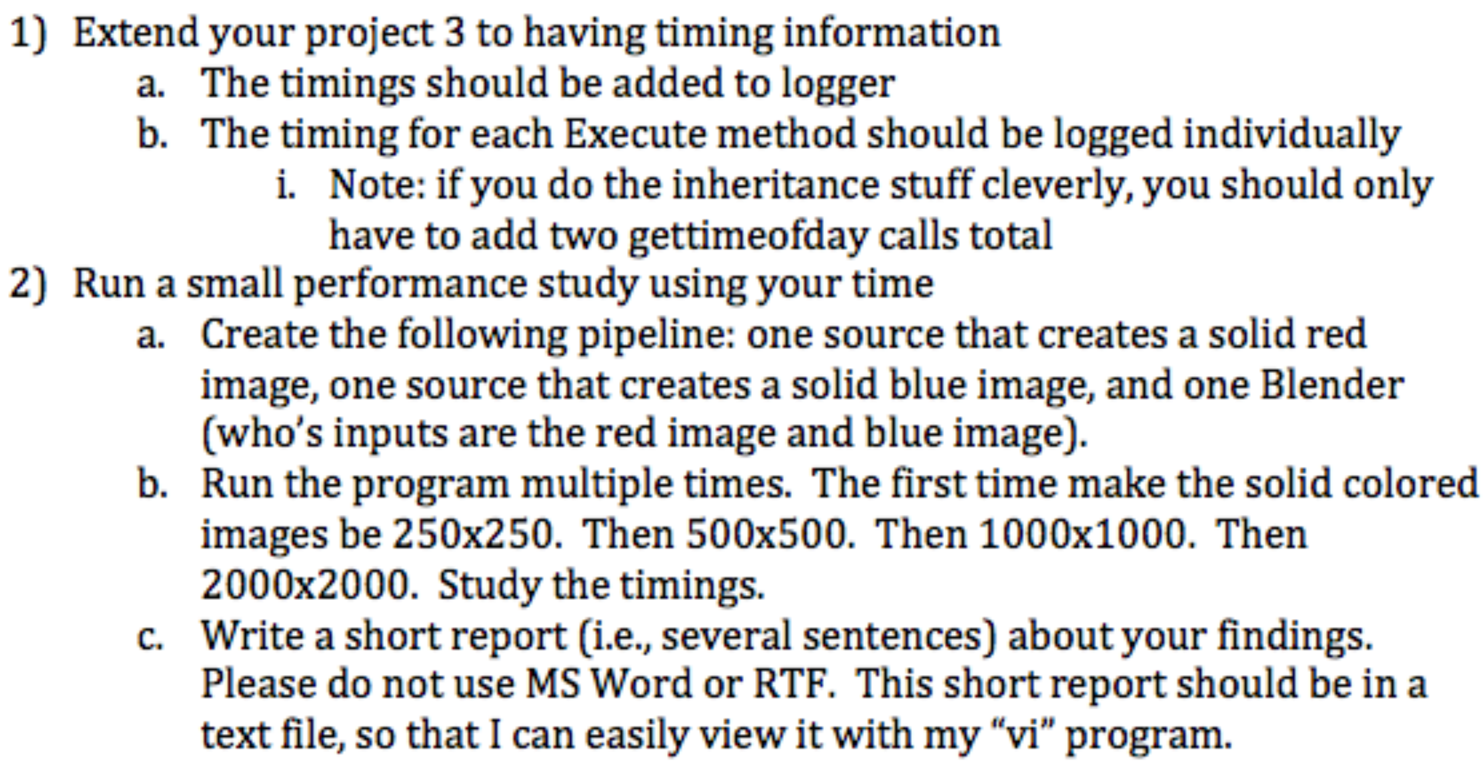
Apply Late Pass #1 To:
Apply Late Pass #2 To:

List Projects That You Feel Have Been Improperly Graded:

Homeworks

- 3H, 4B
  - due Wednesday June 13th
    - not late until 6am Thurs June 14th
- And: no work accepted after 6am Thurs June 14th
  - Late passes don't help after this time
  - And this goes for all projects, not just 3H/4B

4C: canceled



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.

What’s on the final?

- HW 2A is likely
- “What does this print?”
  - Or related questions like “what is the value of this?”
- What is a dangling pointer?
  - (i.e., true/false or vocab)
- Write code!
- … this year I will likely have a question on debuggers

What Will NOT Be On The Final

- Makefiles
  - I don’t expect you to write one
  - I may have a question where I write one for you
  - and other aspects of compilation are fair game
- vi
- templates / STL
- performance analysis
- specific filters from Proj3 (i.e., no “implement blur”)

Notes

- No notes
- Closed book

Will the Final Be Hard?

- I used to say “no”…

  but now I say “yes”…

I want to understand what you have learned about C/C++/Unix

Everything in lecture is fair game

But: almost all parts of lecture are reinforced in the homeworks

Is This Review Exhaustive?

- No.
- Things not in the review are fair game

Review

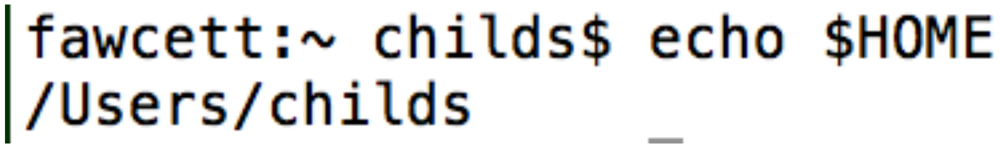
Directories are hierarchical

- Directories can be placed within other directories
- "/" -- The root directory
  - Note "/", where Windows uses "\"
- "/dir1/dir2/file1"
  - What does this mean?

File file1 is contained in directory dir2,
which is contained in directory dir1,
which is in the root directory

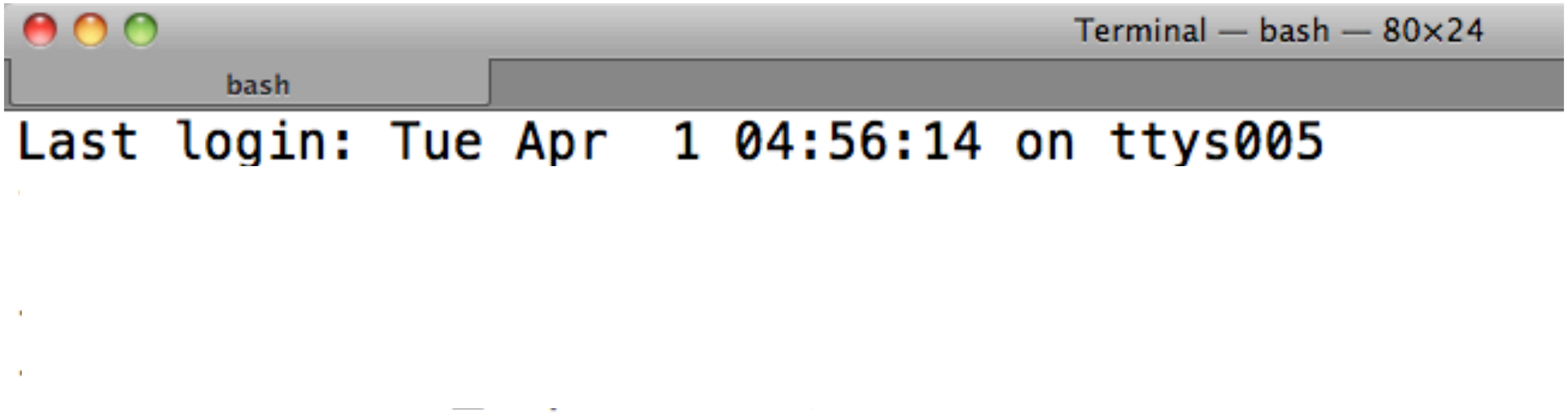
Home directory

- Unix supports multiple users
- Each user has their own directory that they control
- Location varies over Unix implementation, but typically something like “/home/username”
- Stored in environment variables



```
fawcett:~ childs$ echo $HOME
/Users/childs
```

File manipulation



```
Last login: Tue Apr  1 04:56:14 on ttys005
```

New commands: mkdir, cd, touch, ls, rmdir, rm

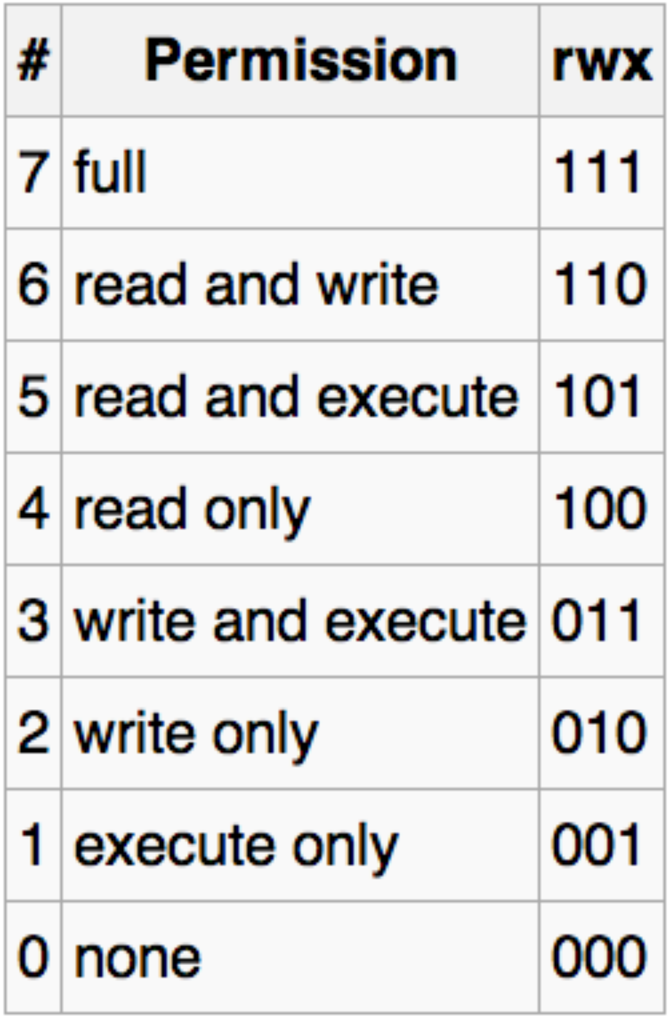
There are 9 file permission attributes

- Can user read?
- Can user write?
- Can user execute?
- Can group read?
- Can group write?
- Can group execute?
- Can other read?
- Can other write?
- Can other execute?

User = "owner"
Other = "not owner, not group"

A bunch of bits … we could represent this with binary

Translating R/W/E permissions to binary



#	Permission	rwx
7	full	111
6	read and write	110
5	read and execute	101
4	read only	100
3	write and execute	011
2	write only	010
1	execute only	001
0	none	000

Image from wikipedia

Which of these modes make sense? Which don't?

We can have separate values (0-7) for user, group, and other

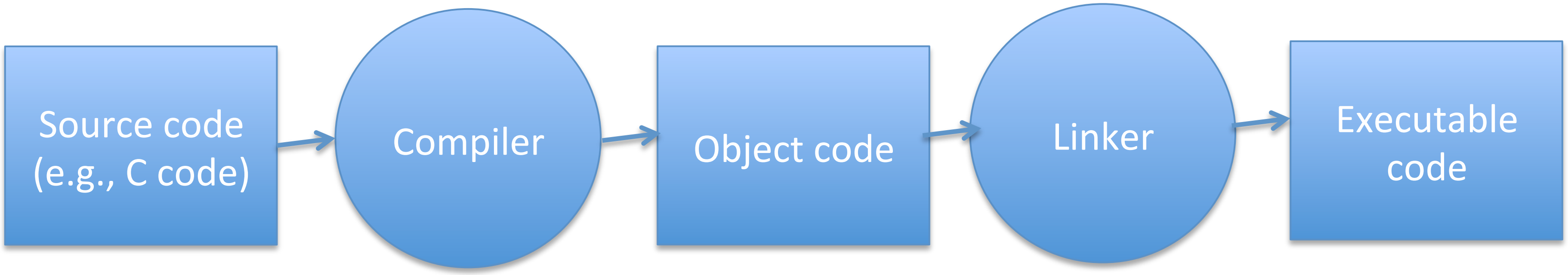
Unix command: chmod

- chmod: change file mode
- chmod 750 <filename>
  - User gets 7 (rwx)
  - Group gets 5 (rx)
  - Other gets 0 (no access)

Lots of options to chmod
(usage shown here is most common)

Build: The Actors

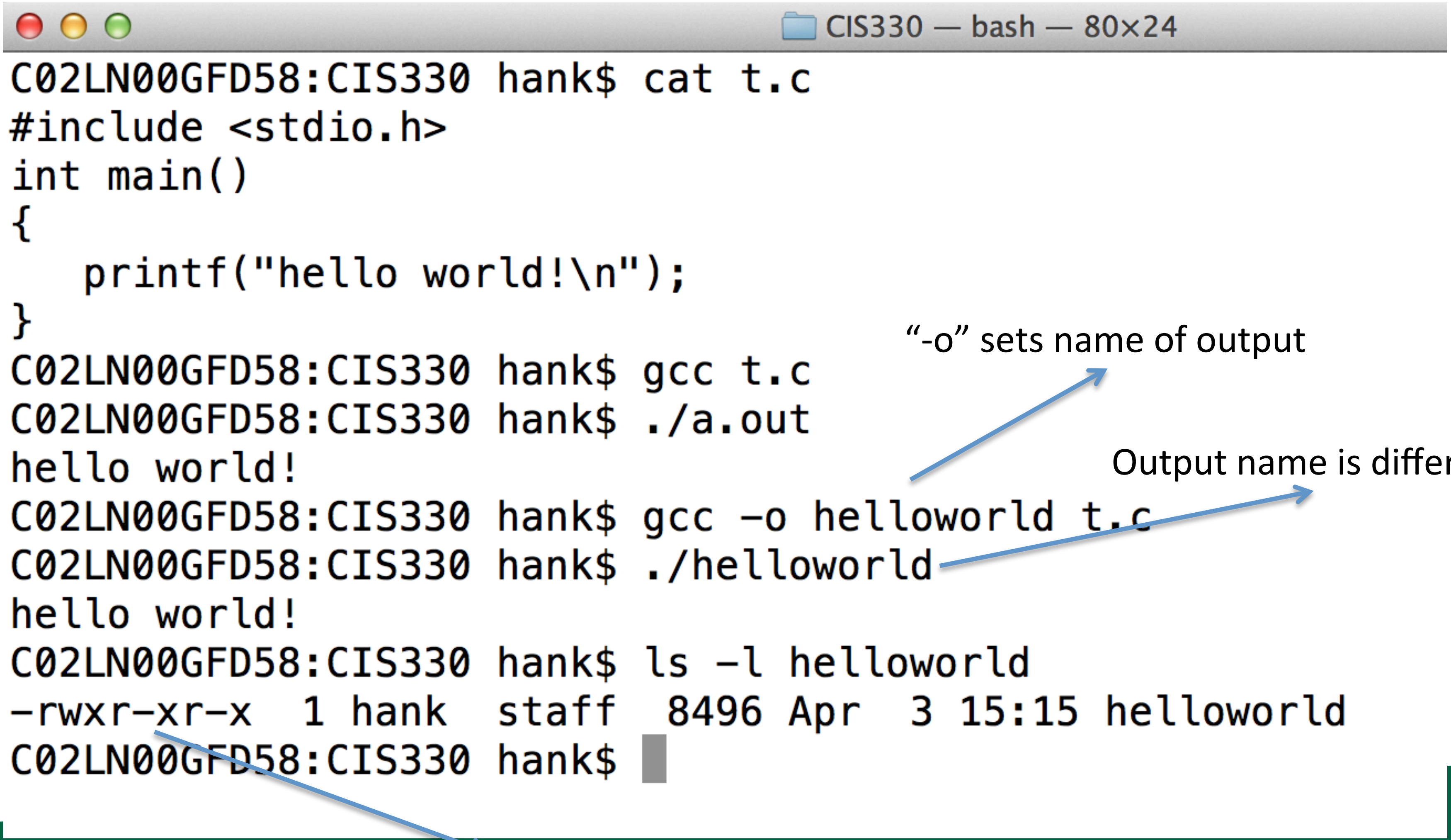
- File types
  - Source code
  - Object code
  - Executable code
- Programs
  - Compiler
  - Linker



Compilers, Object Code, and Linkers

- Compilers transform source code to object code
  - Confusing: most compilers also secretly have access to linkers and apply the linker for you.
- Object code: statements in machine code
  - not executable
  - intended to be part of a program
- Linker: turns object code into executable programs

Our first gcc program: named output



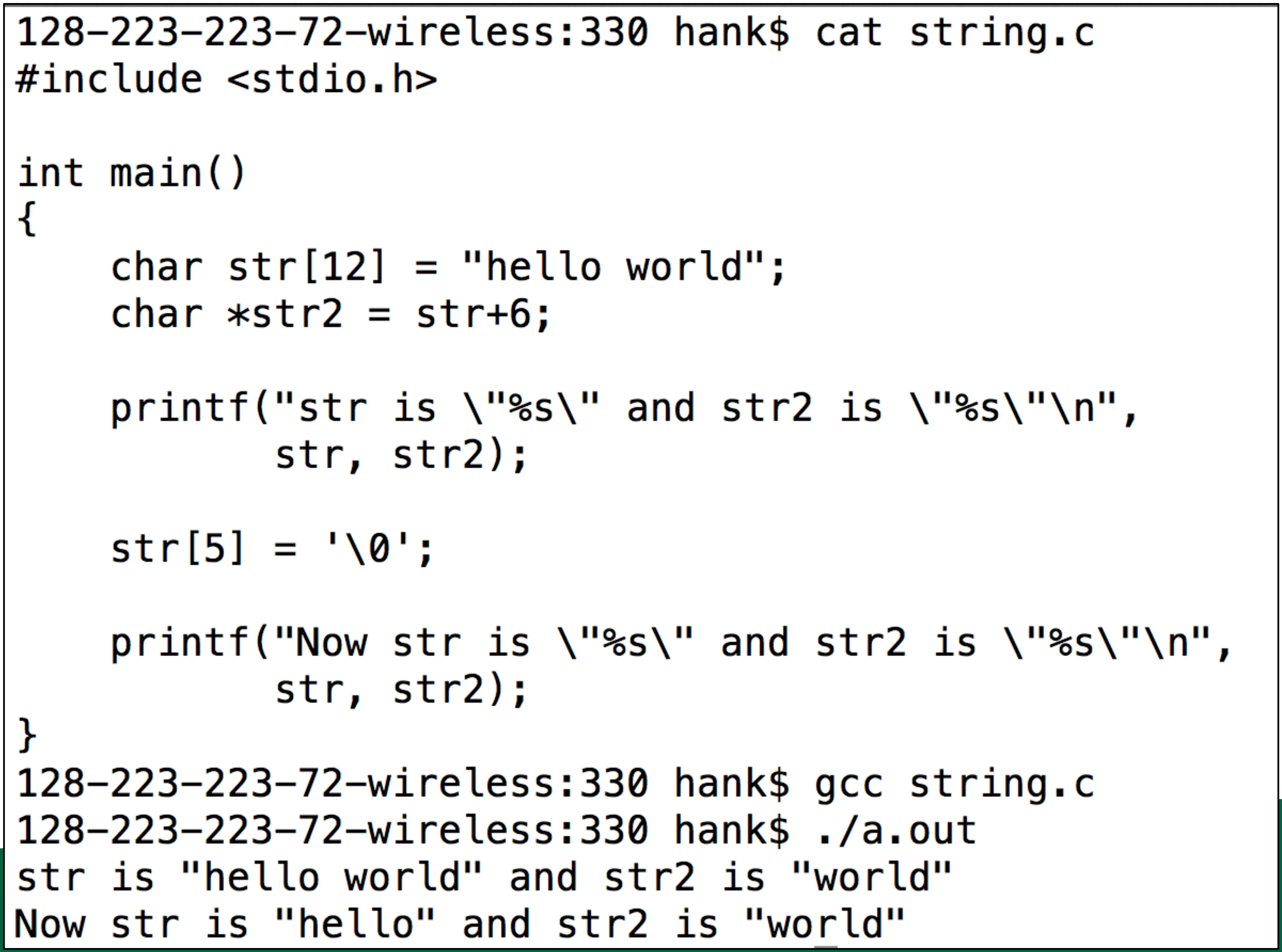
```
C02LN00GFD58:CIS330 hank$ cat t.c
#include <stdio.h>
int main()
{
   printf("hello world!\n");
}
C02LN00GFD58:CIS330 hank$ gcc t.c
C02LN00GFD58:CIS330 hank$ ./a.out
hello world!
C02LN00GFD58:CIS330 hank$ gcc -o helloworld t.c
C02LN00GFD58:CIS330 hank$ ./helloworld
hello world!
C02LN00GFD58:CIS330 hank$ ls -l helloworld
-rwxr-xr-x  1 hank  staff  8496 Apr  3 15:15 helloworld
C02LN00GFD58:CIS330 hank$ 
```

"-o" sets name of output

Output name is different

Output has execute permissions

Character strings example



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

int main()
{
    char str[12] = "hello world";
    char *str2 = str+6;

    printf("str is \"%s\" and str2 is \"%s\"\n",
           str, str2);

    str[5] = '\0';

    printf("Now str is \"%s\" and str2 is \"%s\"\n",
           str, str2);
}
128-223-223-72-wireless:330 hank$ gcc string.c
128-223-223-72-wireless:330 hank$ ./a.out
str is "hello world" and str2 is "world"
Now str is "hello" and str2 is "world"
```

```c
#include <stdio.h>
#include <printf.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    FILE *f_in, *f_out;
    int buff_size;
    char *buffer;

    if (argc != 3)
    {
        printf("Usage: %s <file1> <file2>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    f_in = fopen(argv[1], "r");
    fseek(f_in, 0, SEEK_END);
    buff_size = ftell(f_in);
    fseek(f_in, 0, SEEK_SET);

    buffer = malloc(buff_size);
    fread(buffer, sizeof(char), buff_size, f_in);

    printf("Copying %d bytes from %s to %s\n", buff_size, argv[1], argv[2]);

    f_out = fopen(argv[2], "w");
    fwrite(buffer, sizeof(char), buff_size, f_out);

    fclose(f_in);
    fclose(f_out);

    return 0;
}
```

fprintf

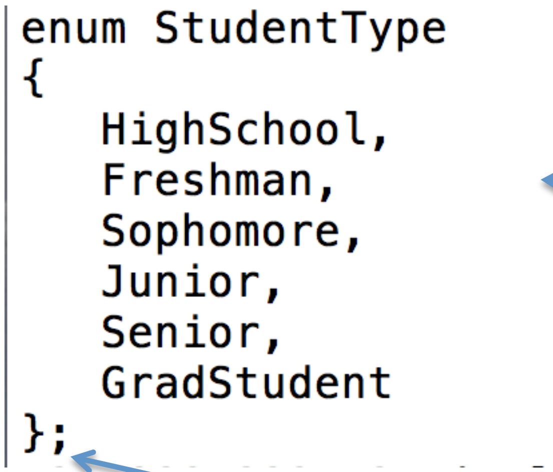
- Just like printf, but to streams
- fprintf(stdout, “helloworld\n”);
  - → same as printf
- fprintf(stderr, “helloworld\n”);
  - prints to “standard error”
- fprintf(f_out, “helloworld\n”);
  - prints to the file pointed to by FILE *f_out.

Enums

- Enums make your own type
  - Type is “list of key words”
- Enums are useful for code clarity
  - Always possible to do the same thing with integers
- Be careful with enums
  - … you can “contaminate” a bunch of useful words

enum example

C keyword "enum" – means enum definition is coming



```
enum StudentType
{
    HighSchool,
    Freshman,
    Sophomore,
    Junior,
    Senior,
    GradStudent
};
```

This enum contains 6 different student types

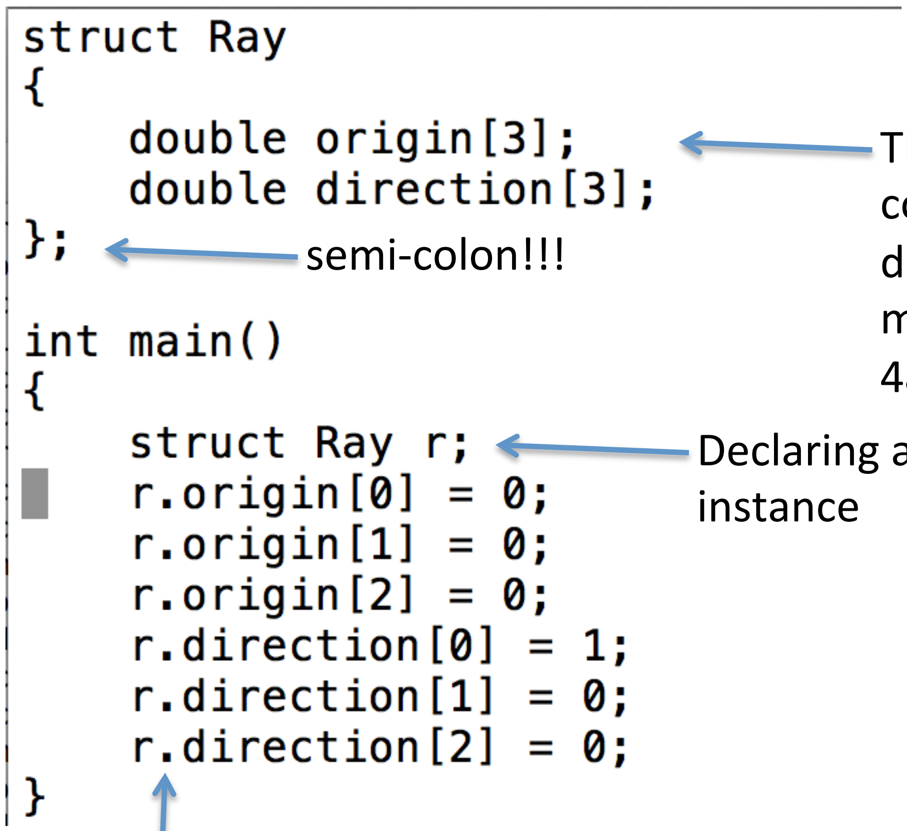
semi-colon!!!

Structs: a complex data type

- Construct that defines a group of variables
  - Variables must be grouped together in contiguous memory
- Also makes accessing variables easier … they are all part of the same grouping (the struct)

struct syntax

C keyword "struct" – means struct definition is coming



```
struct Ray
{
    double origin[3];
    double direction[3];
};

int main()
{
    struct Ray r;
    r.origin[0] = 0;
    r.origin[1] = 0;
    r.origin[2] = 0;
    r.direction[0] = 1;
    r.direction[1] = 0;
    r.direction[2] = 0;
}
```

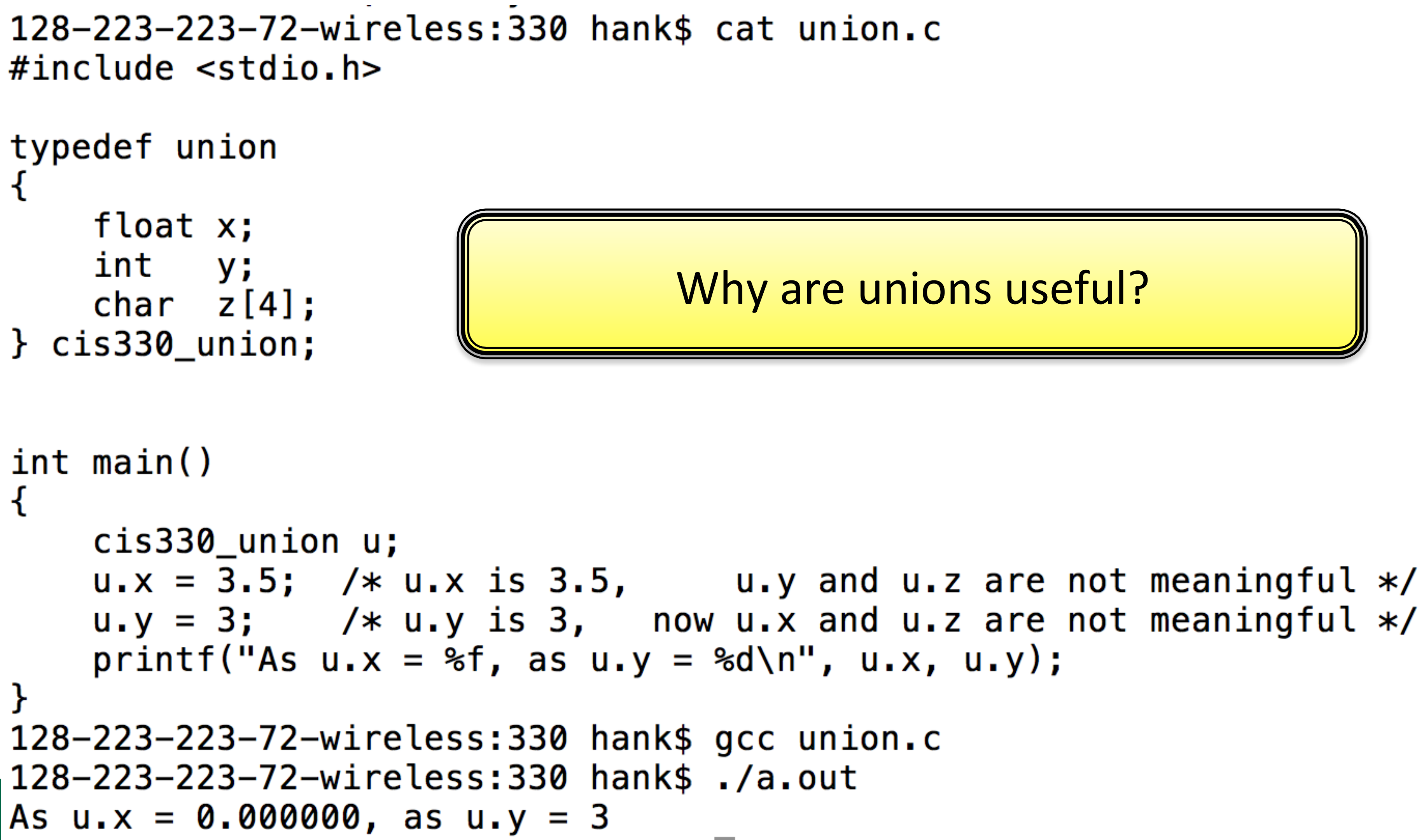
This struct contains 6 doubles, meaning it is 48 bytes

semi-colon!!!

Declaring an instance

"." accesses data members for a struct

Unions



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

typedef union
{
    float x;
    int   y;
    char  z[4];
} cis330_union;


int main()
{
    cis330_union u;
    u.x = 3.5;  /* u.x is 3.5,     u.y and u.z are not meaningful */
    u.y = 3;    /* u.y is 3,   now u.x and u.z are not meaningful */
    printf("As u.x = %f, as u.y = %d\n", u.x, u.y);
}
128-223-223-72-wireless:330 hank$ gcc union.c
128-223-223-72-wireless:330 hank$ ./a.out
As u.x = 0.000000, as u.y = 3
```

Why are unions useful?

Why are Unions useful?

- Allows you to represent multiple data types simultaneously
  - But only if you know you want exactly one of them
- Benefit is space efficiency, which leads to performance efficiency

Unions are also useful for abstracting type. We will re-visit this when we talk about C++'s templates.

Function Pointer Example

```
128-223-223-72-wireless:cli hank$ cat function_ptr.c
#include <stdio.h>
int doubler(int x) { return 2*x; }
int tripler(int x) { return 3*x; }
int main()
{
    int (*multiplier)(int);
    multiplier = doubler;
    printf("Multiplier of 3 = %d\n", multiplier(3));
    multiplier = tripler;
    printf("Multiplier of 3 = %d\n", multiplier(3));
}
128-223-223-72-wireless:cli hank$ gcc function_ptr.c
128-223-223-72-wireless:cli hank$ ./a.out
Multiplier of 3 = 6
Multiplier of 3 = 9
```

Function Pointer Example #2

```
128-223-223-72-wireless:cli hank$ cat array_fp.c
#include <stdio.h>
void doubler(int *X) { X[0]*=2; X[1]*=2; };
void tripler(int *X) { X[0]*=3; X[1]*=3; };
int main()
{
    void (*multiplier)(int *);
    int A[2] = { 2, 3 };
    multiplier = doubler;
    multiplier(A);
    printf("Multiplier of 3 = %d, %d\n", A[0], A[1]);
    multiplier = tripler;
    multiplier(A);
    printf("Multiplier of 3 = %d, %d\n", A[0], A[1]);
}
128-223-223-72-wireless:cli hank$ gcc array_fp.c
128-223-223-72-wireless:cli hank$ ./a.out
```

Function pointer

Part of function signature

Don’t be scared of extra ‘*’s … they just come about because of pointers in the arguments or return values.

Simple-to-Exotic Function Pointer Declarations

```
void (*foo)(void);
void (*foo)(int **, char ***);
char ** (*foo)(int **, void (*)(int));
```

These sometimes come up on interviews.

Definition of Rectangle in rectangle.c Why is this a problem?

prototypes.h

```
struct Rectangle;
void IntializeRectangle(struct Rectangle *r, double v1, double v2, double v3, double v4);
```

rectangle.c

```
struct Rectangle
{
{
   r->minX = v1;    r->maxX = v2;    r->minY = v3;    r->maxY = v4;
}
```

"gcc –c driver.c" needs to make an object file.
It needs info about Rectangle then, not later.

driver.c

```
#include <prototypes.h>

int main()
{
   struct Rectangle r;
   InitializeRectangle(r, 0, 1, 0, 1.5);
}
```

ion: “gcc –E”

The fix is to make sure driver.c has access to the Rectangle struct definition.

er.c

```
{
    struct Rectangle r;
    InitializeRectangle(r, 0, 1, 0, 1.5);
}
```

gcc –E –I.

```
# 1 "driver.c"
# 1 "<built-in>" 1
# 1 "<built-in>" 3
# 162 "<built-in>" 3
# 1 "<command line>" 1
# 1 "<built-in>" 2
# 1 "driver.c" 2
# 1 "./prototypes.h" 1

struct Rectangle;

void InitializeRectangle(struct Rectangle *r, double v1, double v2,
# 2 "driver.c" 2

int main()
{
    struct Rectangle r;
    InitializeRectangle(r, 0, 1, 0, 1.5);
}
```

gcc –E shows what the compiler sees after satisfying “preprocessing”, which includes steps like “#include”.

This is it. If the compiler can’t figure out how to make object file with this, then it has to give up.

4 files: struct.h, prototypes.h, rectangle.c, driver.c

struct.h

```
struct Rectangle
{
   double minX, maxX, minY, maxY;
};
```

prototypes.h

```
#include <struct.h>
void InitializeRectangle(struct Rectangle *r, double v1, double v2, double v3, double v4);
```

rectangle.c

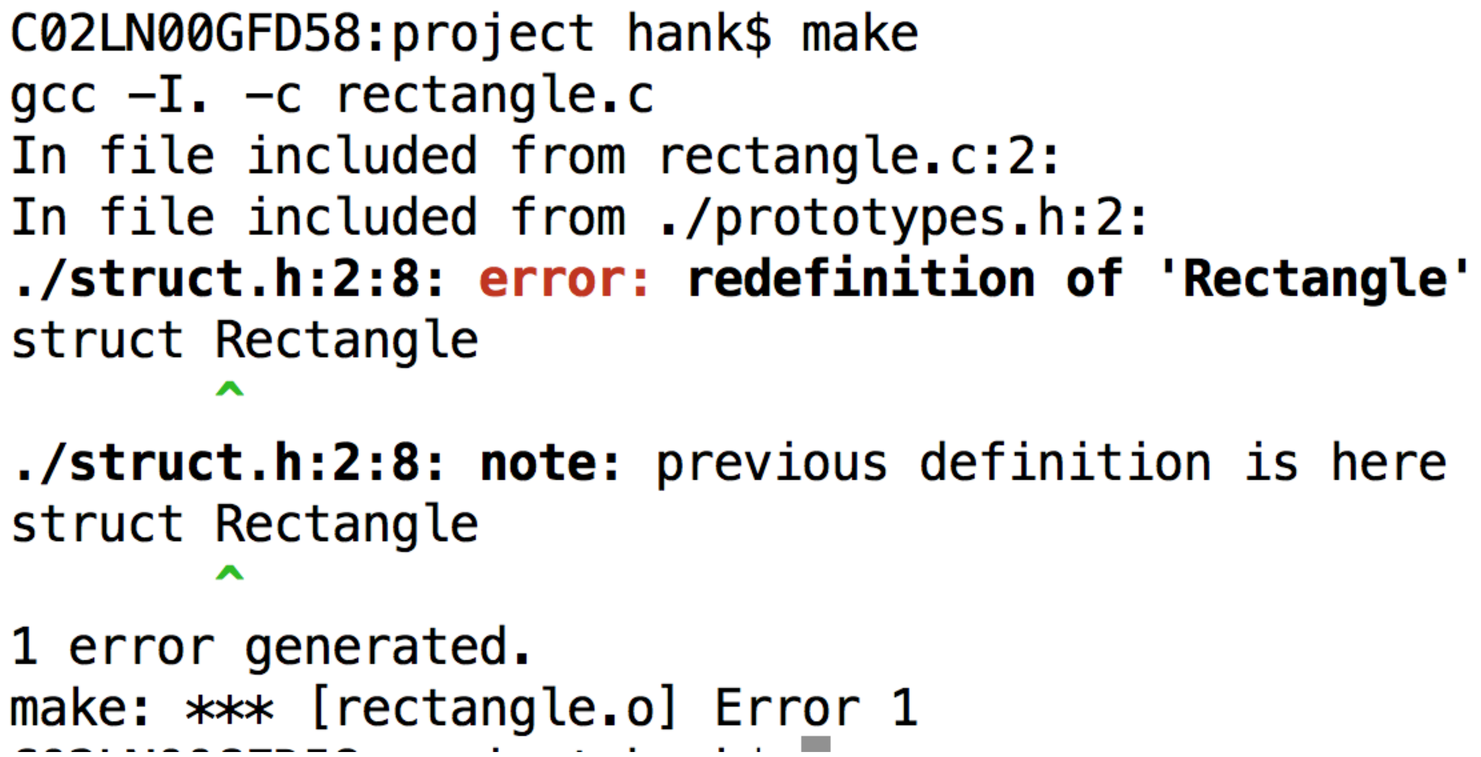
```
#include <struct.h>
#include <prototypes.h>
void IntializeRectangle(struct Rectangle *r, double v1, double v2, double v3, double v4)
{
    r->minX = v1;    r->maxX = v2;    r->minY = v3;    r->maxY = v4;
}
```

driver.c

```
#include <struct.h>
#include <prototypes.h>
int main()
{
    str
    Ini
}
```

What is the problem with this configuration?

Compilation error



```
C02LN00GFD58:project hank$ make
gcc -I. -c rectangle.c
In file included from rectangle.c:2:
In file included from ./prototypes.h:2:
./struct.h:2:8: error: redefinition of 'Rectangle'
struct Rectangle
       ^

./struct.h:2:8: note: previous definition is here
struct Rectangle
       ^

1 error generated.
make: *** [rectangle.o] Error 1
```

#ifndef / #define to the rescue

struct.h

```
#ifndef RECTANGLE_330
#define RECTANGLE_330

struct Rectangle
{
  double minX, maxX, minY, maxY;
};

#endif
```

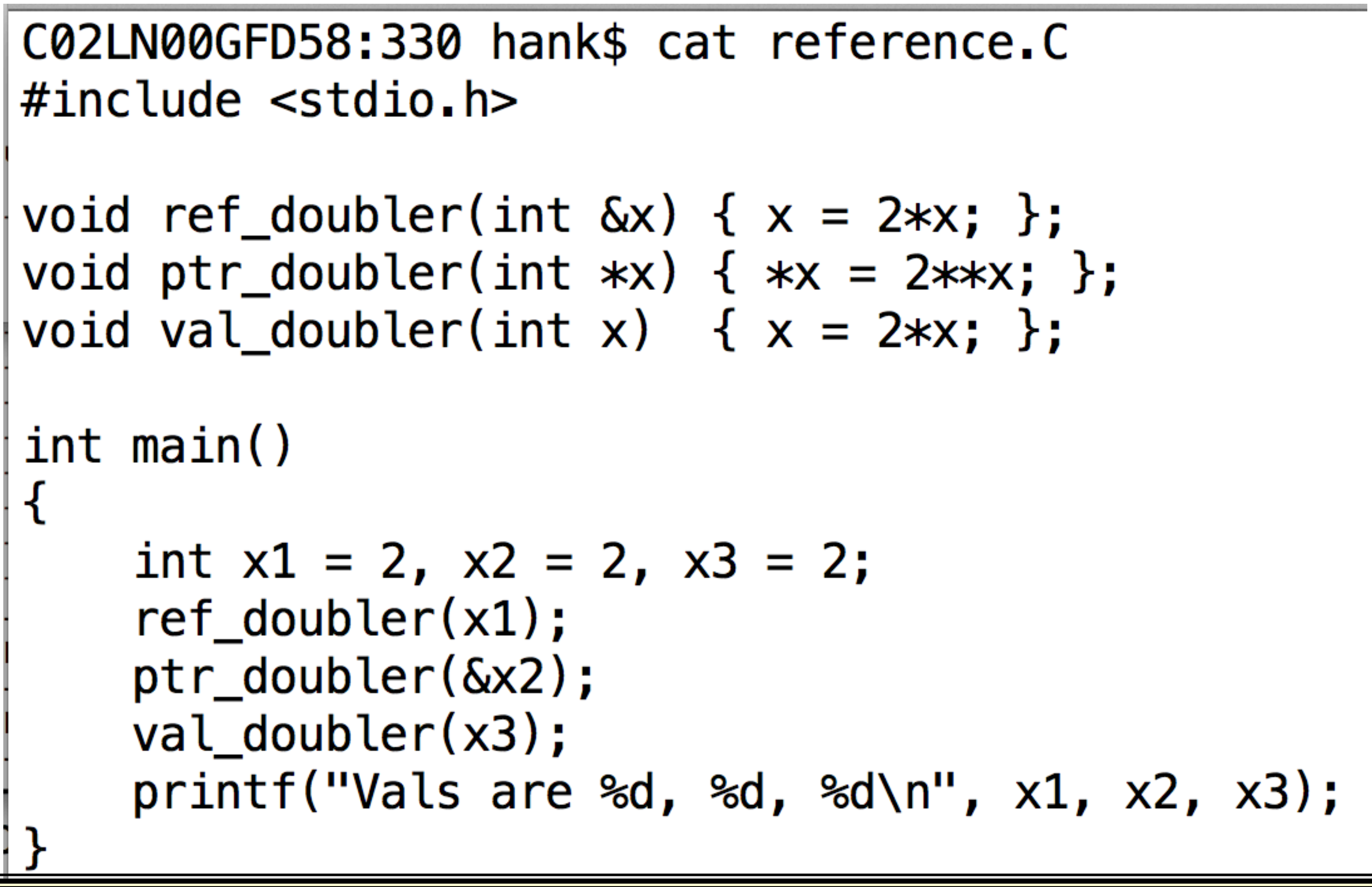
Why does this work?

This problem comes up a lot with big projects, and especially with C++.

References

- Simplified version of a pointer.
- Key differences:
  - You cannot manipulate it
    - Meaning: you are given a reference to exactly one instance … you can't do pointer arithmetic to skip forward in an array to find another object
  - A reference is always valid
    - No equivalent of a NULL pointer … must be a valid instance

References vs Pointers vs Call-By-Value



```
C02LN00GFD58:330 hank$ cat reference.C
#include <stdio.h>

void ref_doubler(int &x) { x = 2*x; };
void ptr_doubler(int *x) { *x = 2**x; };
void val_doubler(int x)  { x = 2*x; };

int main()
{
    int x1 = 2, x2 = 2, x3 = 2;
    ref_doubler(x1);
    ptr_doubler(&x2);
    val_doubler(x3);
    printf("Vals are %d, %d, %d\n", x1, x2, x3);
}
```

ref_doubler and ptr_doubler are both examples of call-by-reference.
val_doubler is an example of call-by-value.

Constructor Types

```
struct TallyCounter
{
   int     count;

           TallyCounter(void);
           TallyCounter(int c);
           TallyCounter(TallyCounter &);
   void    Reset();
   int     GetCount();
   void    IncrementCount();
};
```

Default constructor

Parameterized constructor

Copy constructor

Access Control

- New keywords: public and private
  - public: accessible outside the struct
  - private: accessible only inside the struct
    - Also "protected" … we will talk about that later

```
struct TallyCounter
{
  private:
    int     count;

  public:
            TallyCounter(void);
            TallyCounter(int c);
            TallyCounter(TallyCounter &);
    void    Reset();
    int     GetCount();
    void    IncrementCount();
};
```

Everything following is private. Only will change when new access control keyword is encountered.

Everything following is now public. Only will change when new access control keyword is encountered.

Public, private, protected

	Accessed by derived types*	Accessed outside object
Public	Yes	Yes
Protected	Yes	No
Private	No	No

* = with public inheritance

3 big changes to structs in C++

1) You can associate “methods” (functions) with structs
2) You can control access to data members and methods
3) Inheritance

class vs struct

- class is new keyword in C++
- classes are very similar to structs
  - the only differences are in access control
    - primary difference: struct has public access by default, class has private access by default
- Almost all C++ developers use classes and not structs
  - C++ developers tend to use structs when they want to collect data types together (i.e., C-style usage)
  - C++ developers use classes for objects … which is most of the time

You should use classes!
Even though there isn't much difference …

Simple inheritance example

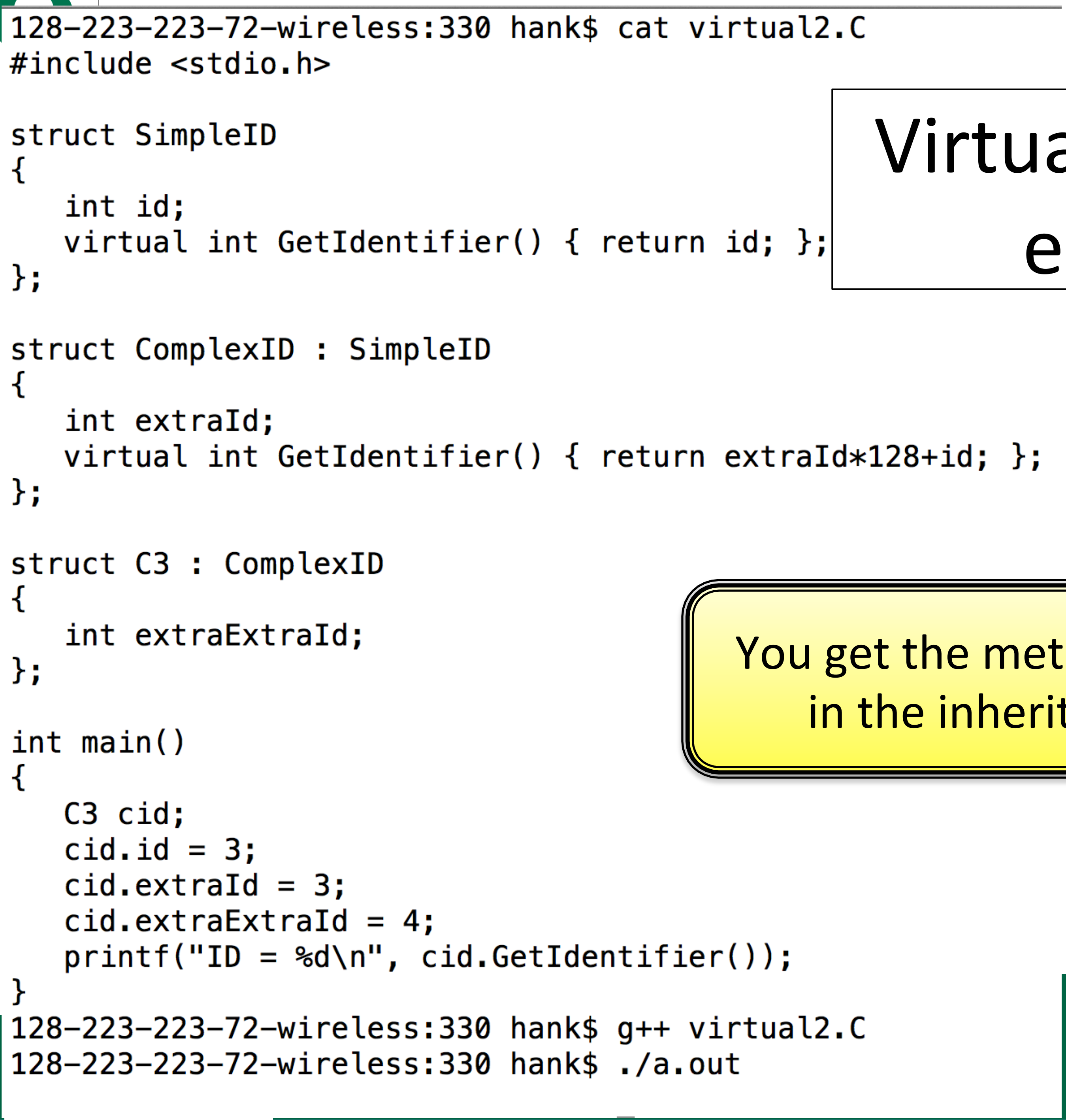
```
struct A
{
    int x;
};

struct B : A
{
    int y;
};

int main()
{
    B b;
    b.x = 3;
    b.y = 4;
}
```

- Terminology
  - B inherits from A
  - A is a base type for B
  - B is a derived type of A
- Noteworthy
  - ":" (during struct definition) → inherits from
    - Everything from A is accessible in B
      - (b.x is valid!!)

Virtual functions: example


```
128-223-223-72-wireless:330 hank$ cat virtual2.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; };
};

struct C3 : ComplexID
{
   int extraExtraId;
};

int main()
{
   C3 cid;
   cid.id = 3;
   cid.extraId = 3;
   cid.extraExtraId = 4;
   printf("ID = %d\n", cid.GetIdentifier());
}
128-223-223-72-wireless:330 hank$ g++ virtual2.C
128-223-223-72-wireless:330 hank$ ./a.out
```

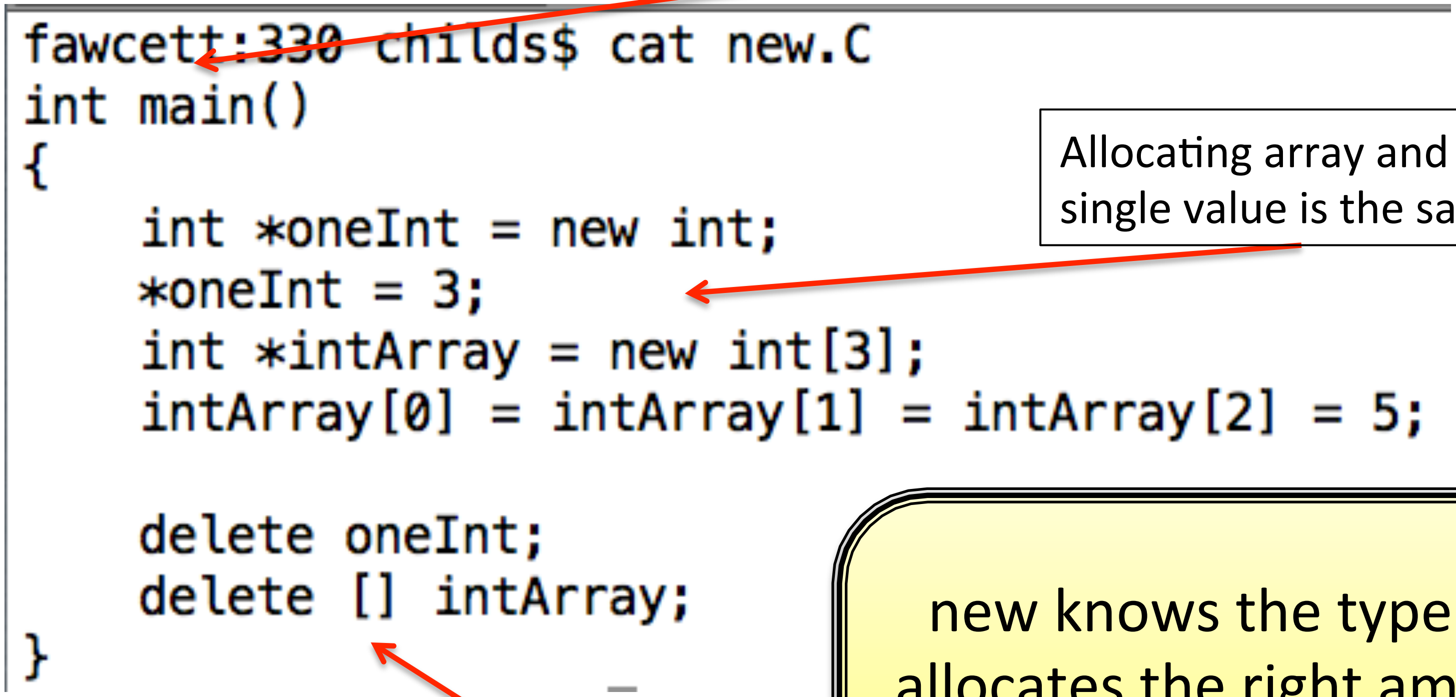
You get the method furthest down in the inheritance hierarchy

C++ memory management

- C++ provides new constructs for requesting heap memory from the memory manager
  - stack memory management is not changed
    - (automatic before, automatic now)
- Allocate memory: “new”
- Deallocate memory: “delete”

new / delete syntax

No header necessary



```
fawcett:330 childs$ cat new.C
int main()
{
     int *oneInt = new int;
     *oneInt = 3;
     int *intArray = new int[3];
     intArray[0] = intArray[1] = intArray[2] = 5;

     delete oneInt;
     delete [] intArray;
}
```

Allocating array and single value is the same.

Deleting array takes [], deleting single value doesn't.

new knows the type and allocates the right amount.

new int → 4 bytes
new int[3] → 12 bytes

new calls constructors for your classes

- Declare variable in the stack: constructor called
- Declare variable with “malloc”: constructor not called
  - C knows nothing about C++!
- Declare variable with “new”: constructor called

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()
```

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
```

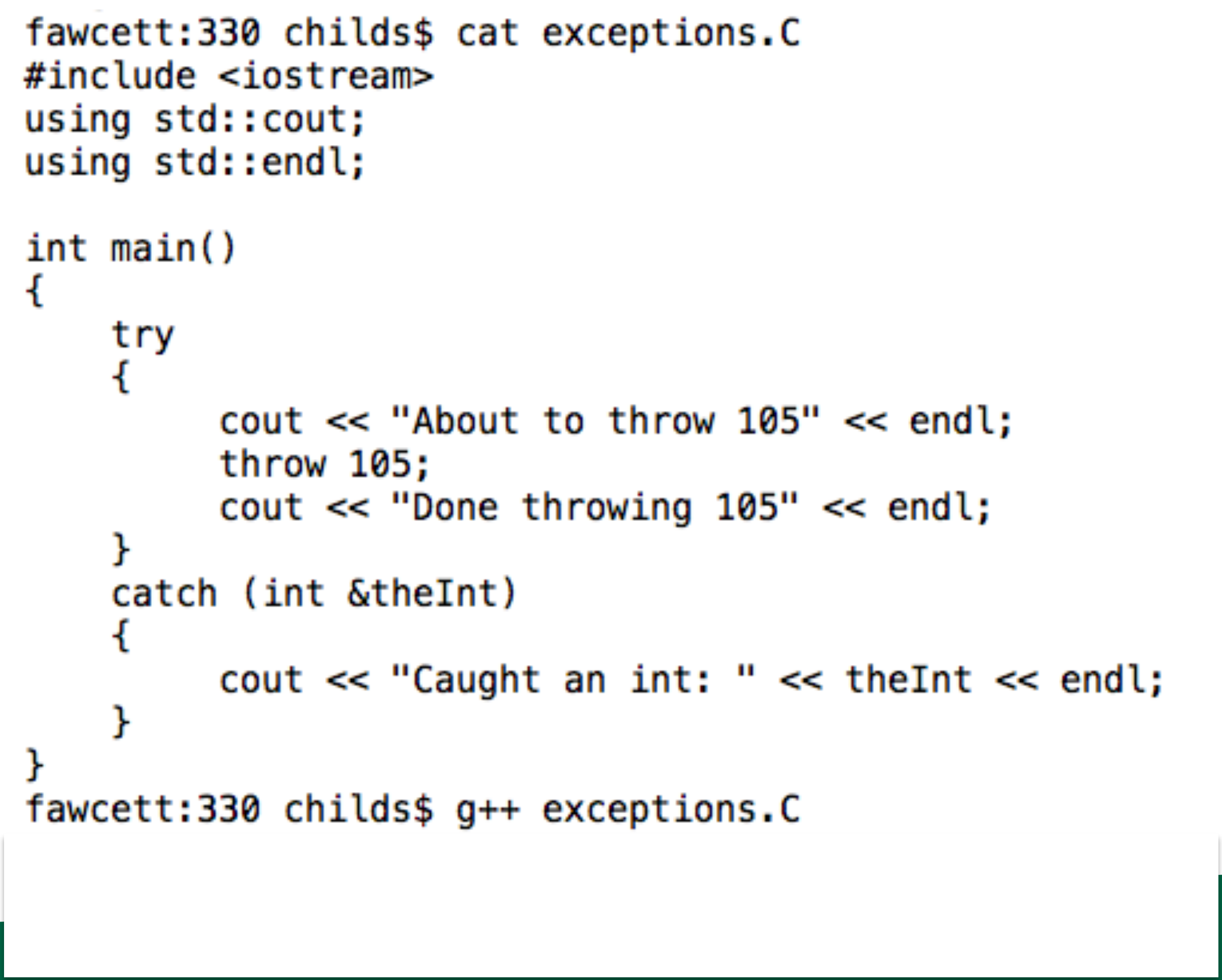
Call operator ++ on MyInt.

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".

UNIVERSITY OF OREGON

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
```

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 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 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 pointers

int X = 4;

int *P = &X;

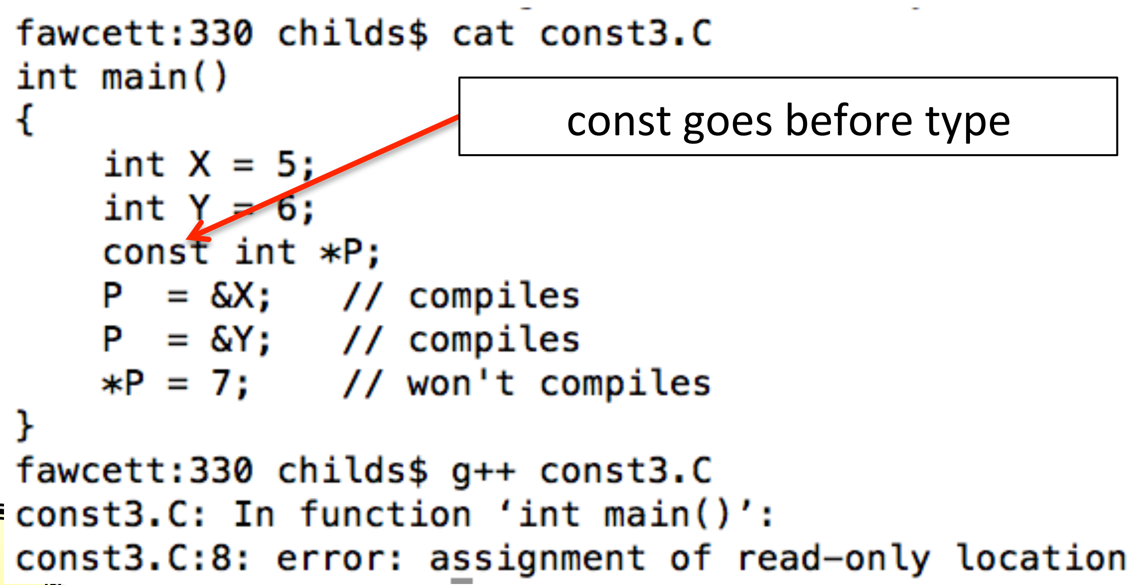
<u>Idea #1:</u>

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
```

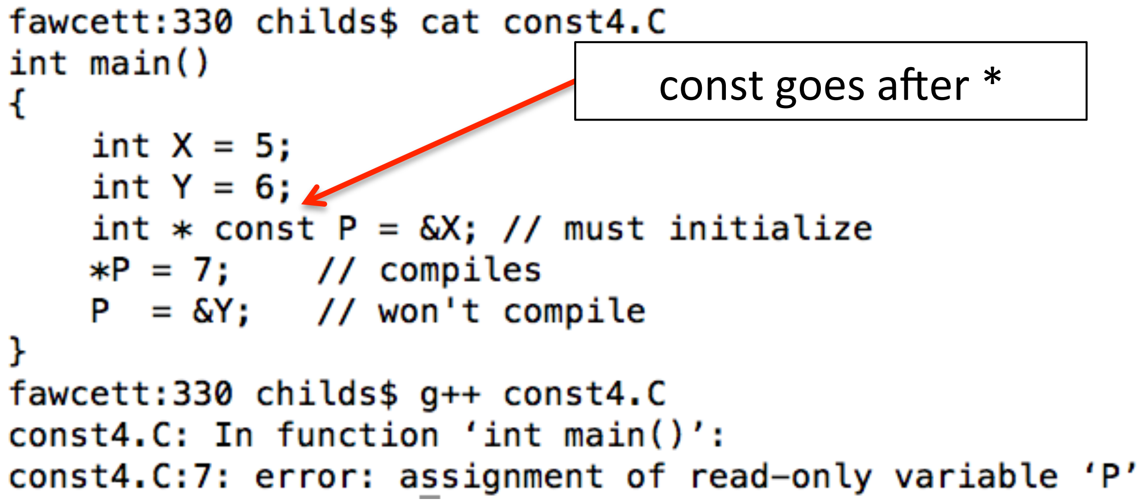
const goes before type

pointer <u>can</u> change, but you <u>can’t</u> 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 *

<u>Idea #2:</u>

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;”

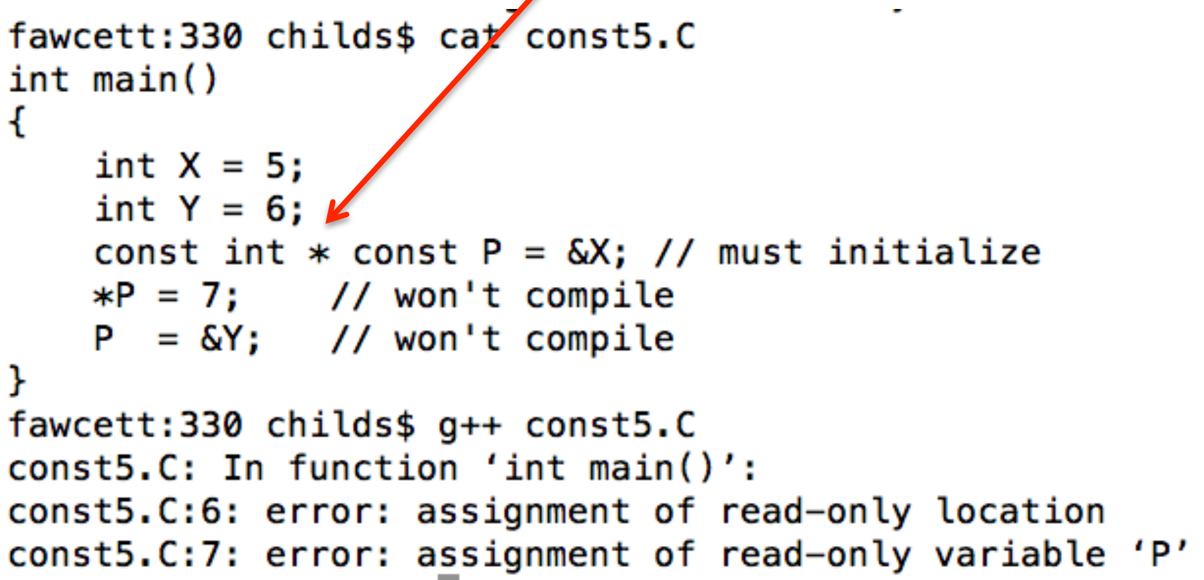
“int Y = 5; P = &Y;”

OK:

none

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

const 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’
```

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.

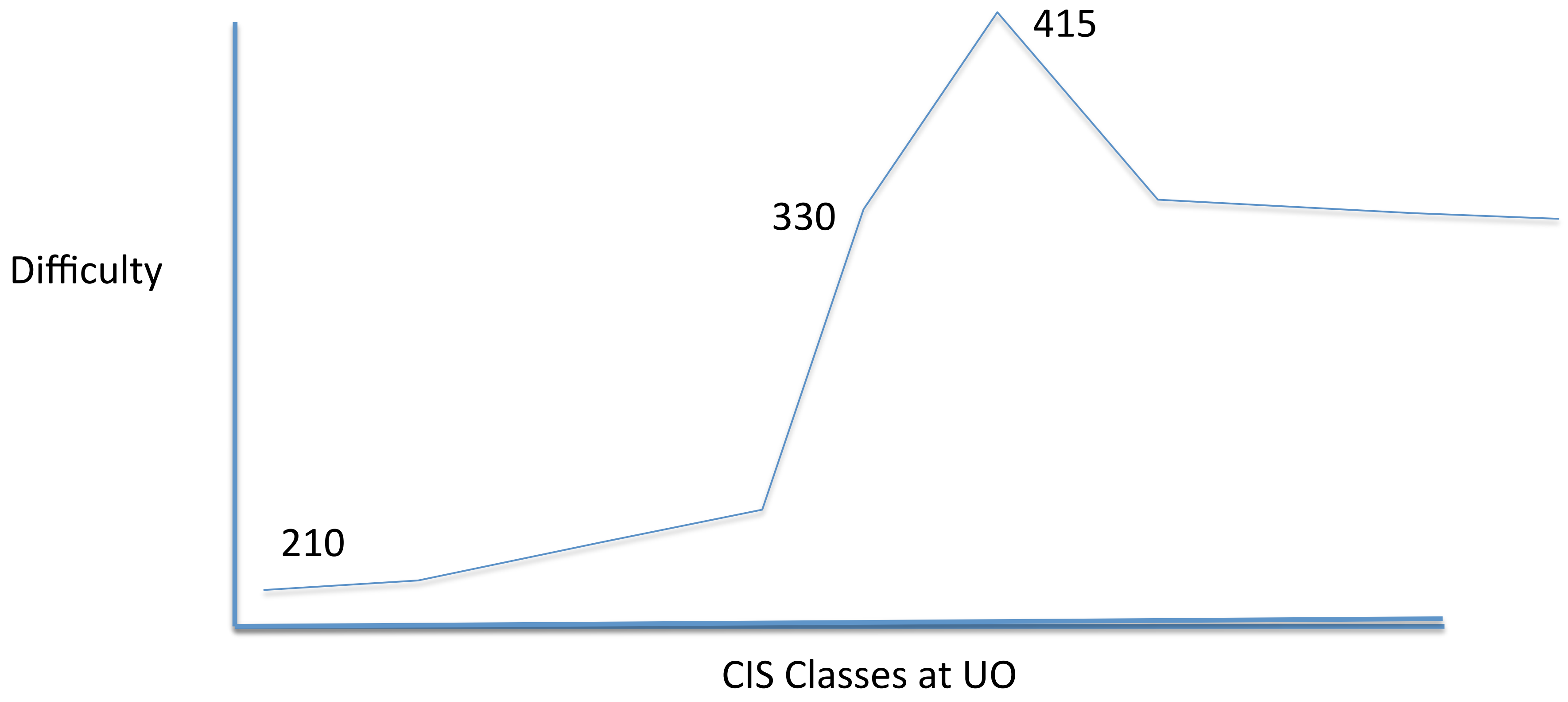
But, in large projects, compiler can't see implementation for everything.

Chance to Discuss Homeworks

- Your questions?

Final Thoughts

Goal: Getting Ready for 415



Final Comments

- This class is hard
- It is hard for the instructors too
  - I set it up this way, since I think it is crucial that you learn these things
  - Remember: I was in industry for 15 years
- Project 3: you have done something significant and you should be proud.
  - Github

This class

- If you liked this class
  - You will love being a professional programmer
    - One of my favorite periods of my career
  - Aim for a development job (write code)
- If you didn't like this class
  - Give it another chance
  - This class also challenged you in other ways (time management, hours per week, lots of material)
- Thank you for a great term

UNIVERSIT

4B

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

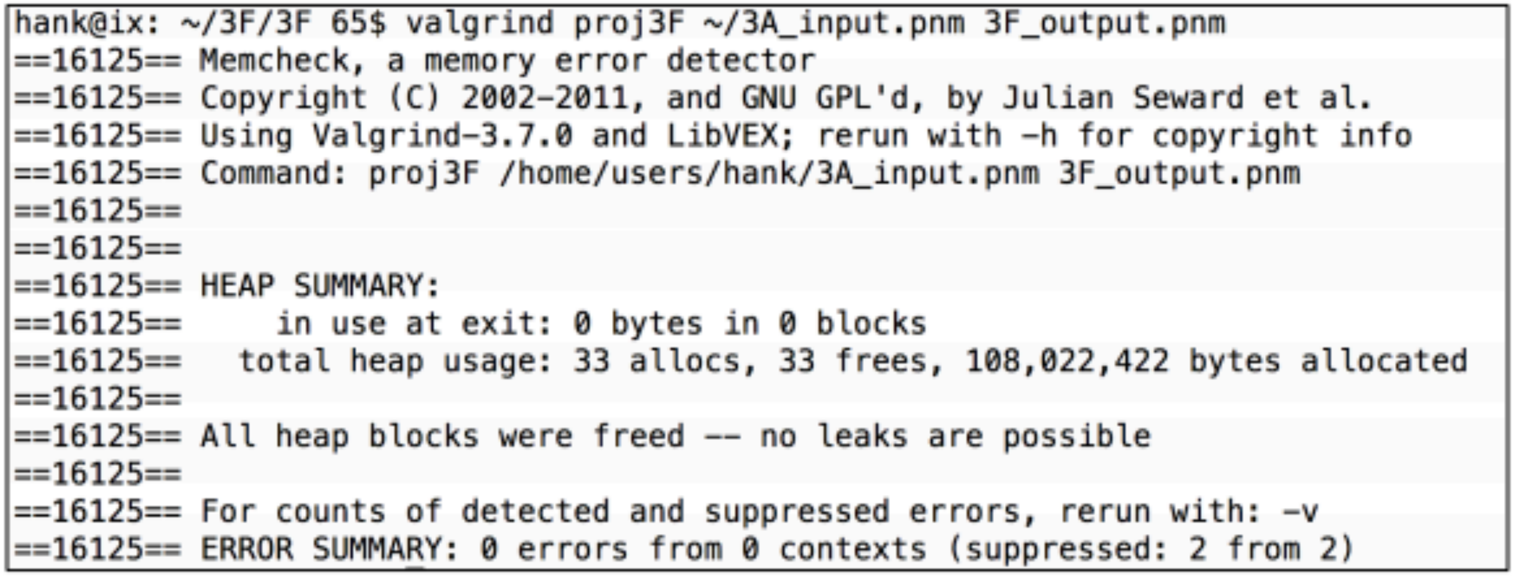
Tasks:
1) start with your 3E source code. (Don't use 3F, since it has exceptions and

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)
```

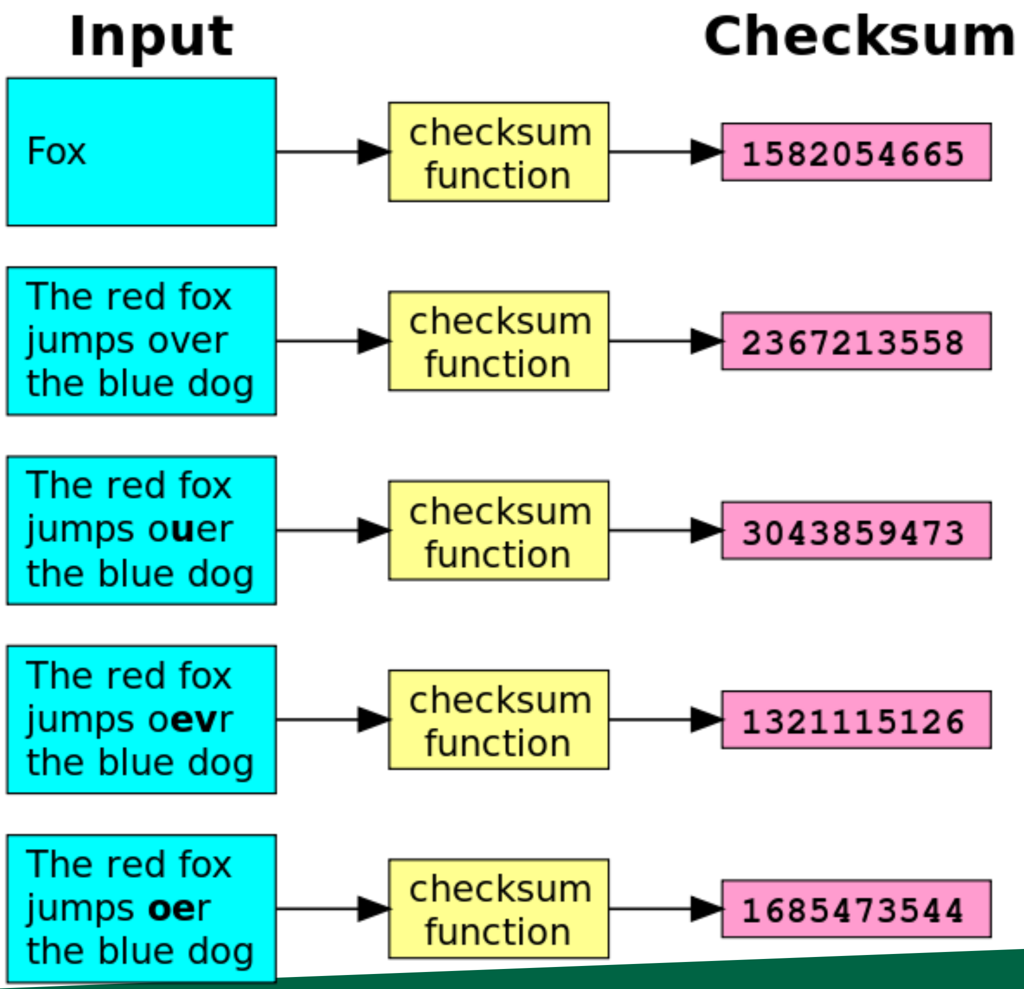
3H

- Will look at this together now

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