CIS 330:

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

Lecture 8: enum, structs, and unions

April 23rd, 2018

Hank Childs, University of Oregon

Announcements: Plan This Week (1/2)

- Hank out of town: Monday night->Thursday night
- Brent covers Hank's OH on Tuesday
- Wednesday class is ON.
  - Brent does lab #2, which includes parts of lecture 7
- Hank does lecture on Friday

Announcements: Plan This Week (2/2)

- 2B: due today
- 2C: assigned today, due Saturday, 4/28/18
- 3A: assigned today due Tuesday, 5/1/18
  - NOTE: Lecture 9 is online
  - Corrections on Piazza

Outline

- Review
- Project 2B
- Enum
- Struct
- Unions
- Project 2C

File I/O: streams and file descriptors

- Two ways to access files:
  - File descriptors:
    - Lower level interface to files and devices
      - Provides controls to specific devices
    - Type: small integers (typically 20 total)
  - Streams:
    - Higher level interface to files and devices
      - Provides uniform interface; easy to deal with, but less powerful
    - Type: FILE *

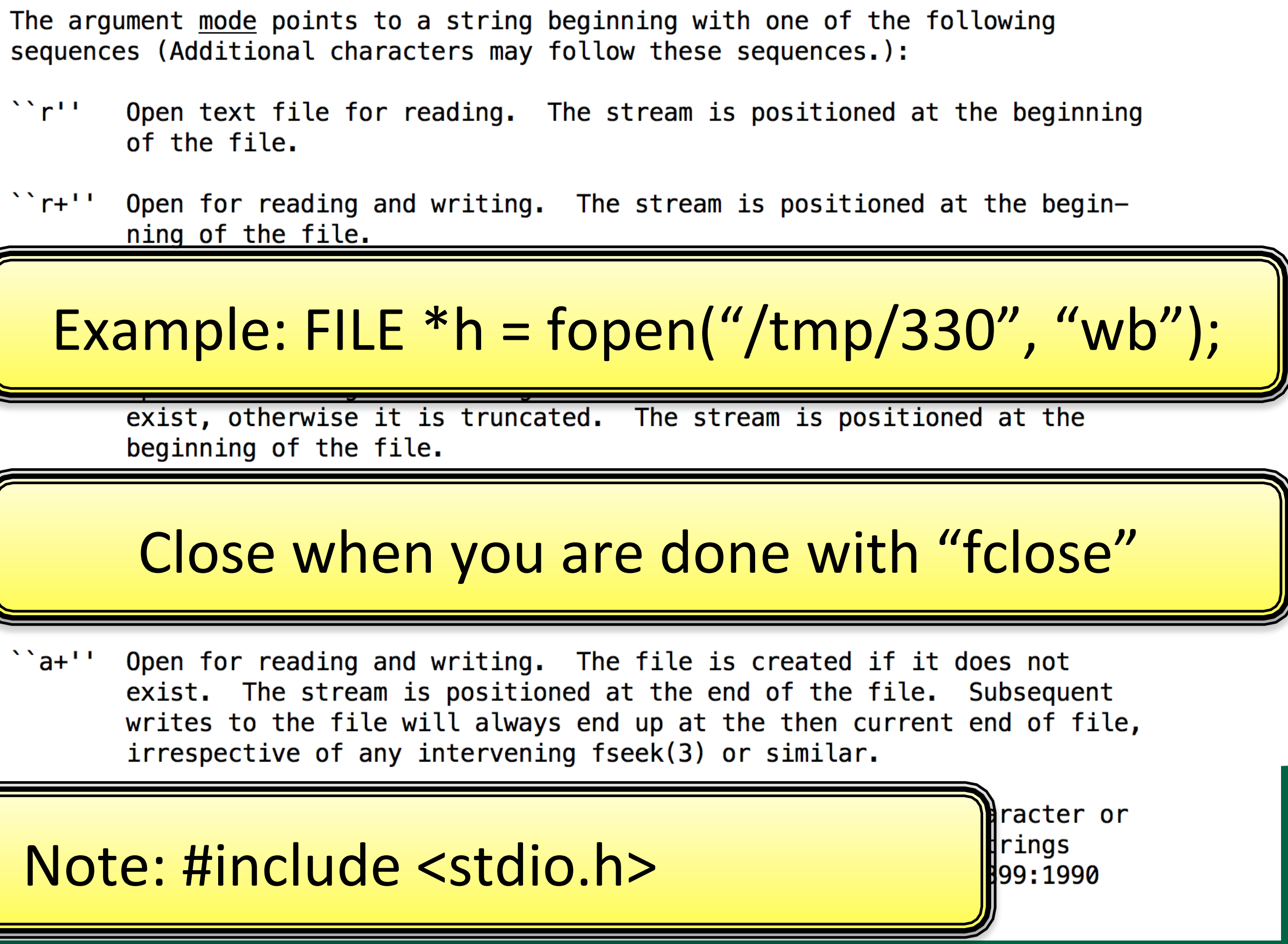
Streams are more portable, and more accessible to beginning programmers. (I teach streams here.)

File I/O

- Process for reading or writing
  - Open a file
    - Tells Unix you intend to do file I/O
    - Function returns a "FILE *
      - Used to identify the file from this point forward
    - Checks to see if permissions are valid
  - Read from the file / write to the file
  - Close the file

Opening a file

- FILE *handle = fopen(filename, mode);



```
The argument mode points to a string beginning with one of the following
sequences (Additional characters may follow these sequences.):

``r''   Open text file for reading.  The stream is positioned at the beginning
        of the file.

``r+''  Open for reading and writing.  The stream is positioned at the begin-
        ning of the file.

        exist, otherwise it is truncated.  The stream is positioned at the
        beginning of the file.

``a+''  Open for reading and writing.  The file is created if it does not
        exist.  The stream is positioned at the end of the file.  Subsequent
        writes to the file will always end up at the then current end of file,
        irrespective of any intervening fseek(3) or similar.

                                                      racter or
                                                      rings
                                                      99:1990
```

Example: FILE *h = fopen("/tmp/330", "wb");

Close when you are done with "fclose"

Note: #include <stdio.h>

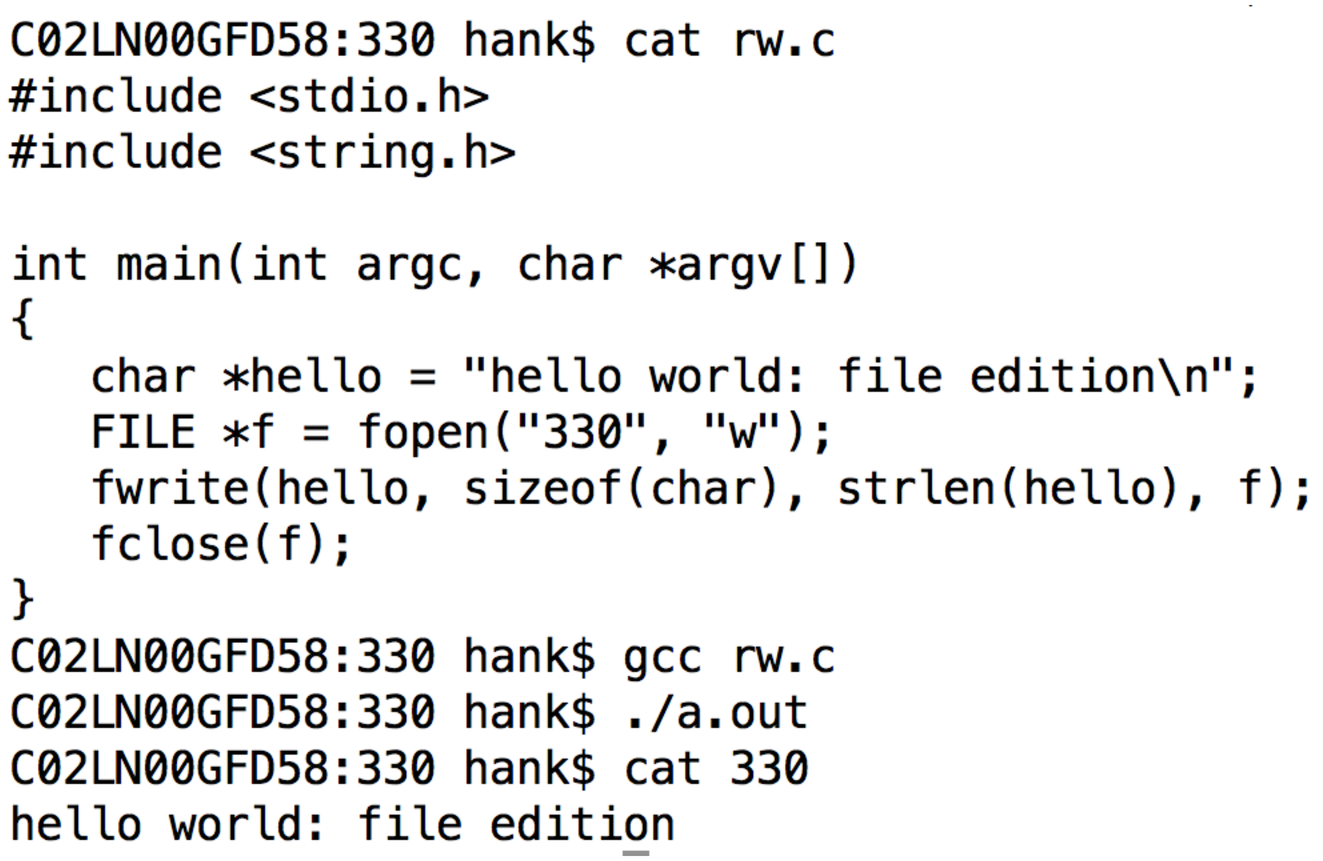
Unix and Windows difference

- 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

Example



```
C02LN00GFD58:330 hank$ cat rw.c
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
   char *hello = "hello world: file edition\n";
   FILE *f = fopen("330", "w");
   fwrite(hello, sizeof(char), strlen(hello), f);
   fclose(f);
}
C02LN00GFD58:330 hank$ gcc rw.c
C02LN00GFD58:330 hank$ ./a.out
C02LN00GFD58:330 hank$ cat 330
hello world: file edition
```

```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;
}
```

Printing to terminal and reading from terminal

- In Unix, printing to terminal and reading from terminal is done with file I/O
- Keyboard and screen are files in the file system!
  - (at least they were …)

Standard Streams

- Wikipedia: “preconnected input and output channels between a computer program and its environment (typically a text terminal) when it begins execution”
- Three standard streams:
  - stdin (standard input)
  - stdout (standard output)
  - stderr (standard error)

What mechanisms in C allow you to access standard streams?

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);

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.

buffering and printf

- Important: printf is buffered
- So:
  - printf puts string in buffer
  - other things happen
  - buffer is eventually printed
- But what about a crash?
  - printf puts string in buffer
  - other things happen … including a crash
  - buffer is never printed!

Solutions: (1) fflush, (2) fprintf(stderr) always flushed

Outline

- Review
- Project 2B
- Enum
- Struct
- Unions
- Project 2C

Project 2B

Worth 4% of your grade

Assignment: Write a program that reads the file “2B_binary_file”. This file contains a two-dimensional array of 100 integers (indices 0 to 99); that is, 10x10. You are to read in the 5x5 top left corner of the array. That is, the values at indices 0-4, 10-14, 20-24, 30-34, and 40-44. You may only read 25 integers total. Do not read all 100 and throw some out. You will then write out the new 5x5 array to the output file, which is one of the command line arguments. Please write one integer per line (25 lines total). You should be able to “cat” the file afterwards and see the values.

Use only C file stream functions for file reading and writing in this project: fopen, fread, fseek, fprintf, fclose (consider ftell for debugging). Each of these functions needs a “FILE *” pointer as input. <u>Your program will be checked for good programming practices.</u> (Close your file streams, use memory correctly, variable initialization, etc.)

Also, add support for command line arguments (argc and argv) in the main function.

Your program should run as:

```
./proj2B <input_name> <output_name>
```

Outline

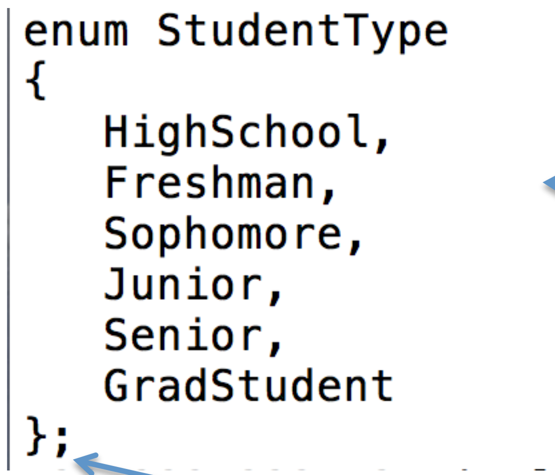
- Review
- Project 2B
- Enum
- Struct
- Unions
- Project 2C

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
};
```

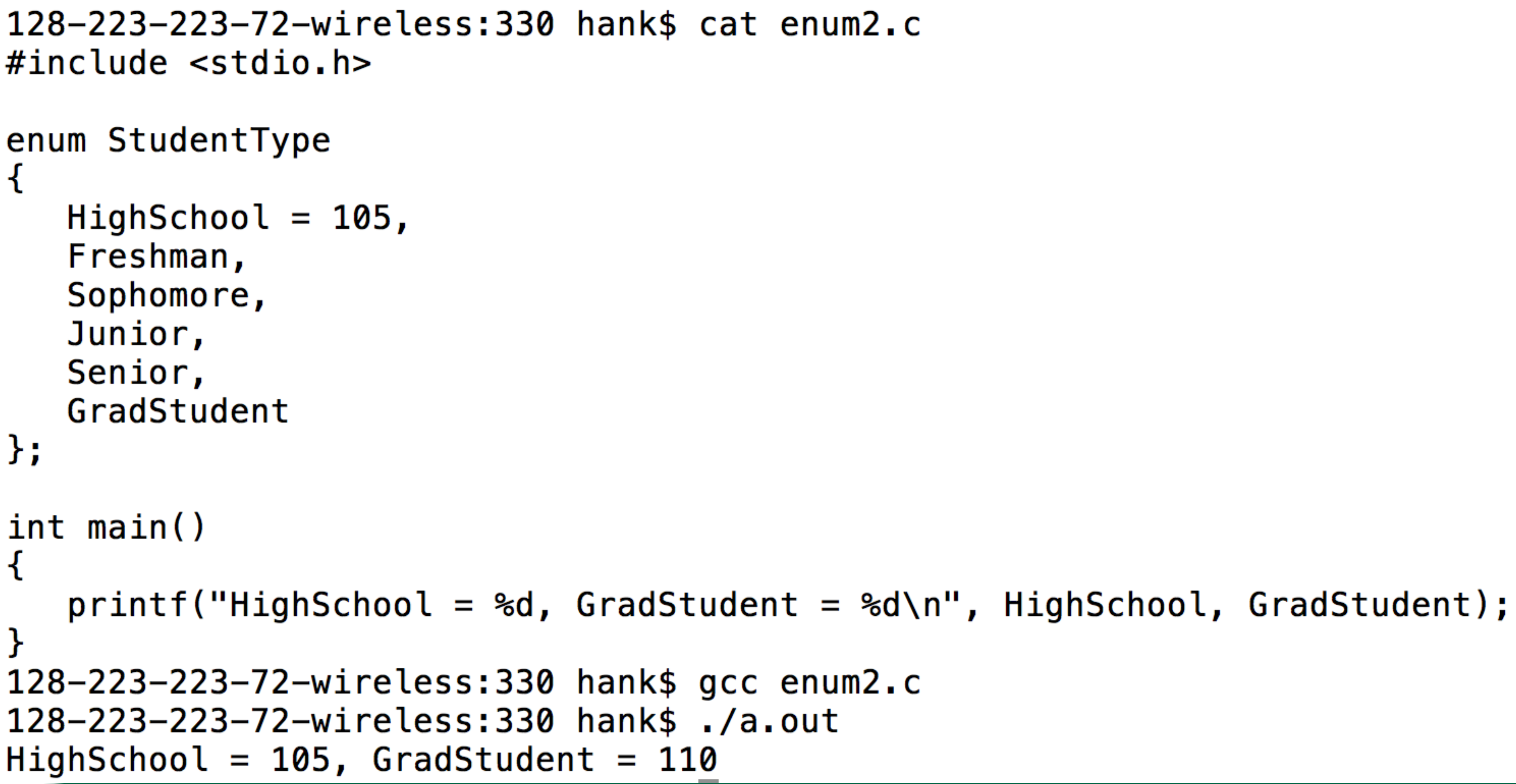
This enum contains 6 different student types

semi-colon!!!

enum example

```
int AverageAge(enum StudentType st)
{
    if (st == HighSchool)
       return 16;
    if (st == Freshman)
       return 18;
    if (st == Sophomore)
       return 19;
    if (st == Junior)
       return 21;
    if (st == Senior)
       return 23;
    if (st == GradStudent)
       return 26;

    return -1;
}
```

enums translate to integers … and you can set their range



```
128-223-223-72-wireless:330 hank$ cat enum2.c
#include <stdio.h>

enum StudentType
{
   HighSchool = 105,
   Freshman,
   Sophomore,
   Junior,
   Senior,
   GradStudent
};

int main()
{
   printf("HighSchool = %d, GradStudent = %d\n", HighSchool, GradStudent);
}
128-223-223-72-wireless:330 hank$ gcc enum2.c
128-223-223-72-wireless:330 hank$ ./a.out
HighSchool = 105, GradStudent = 110
```

But enums can be easier to maintain than integers

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

```
int AverageAge(enum StudentType st)
{
    if (st == HighSchool)
        return 16;
    if (st == Freshman)
        return 18;
    if (st == Sophomore)
        return 19;
    if (st == Junior)
        return 21;
    if (st == Senior)
        return 23;
    if (st == PostBac)
        return 24;
    if (st == GradStudent)
        return 26;

    return -1;
```

If you had used integers, then this is a bigger change and likely to lead to bugs.

Outline

- Grade 4A
- Review
- Project 2B
- Enum
- Struct
- Unions
- Project 2C

Data types

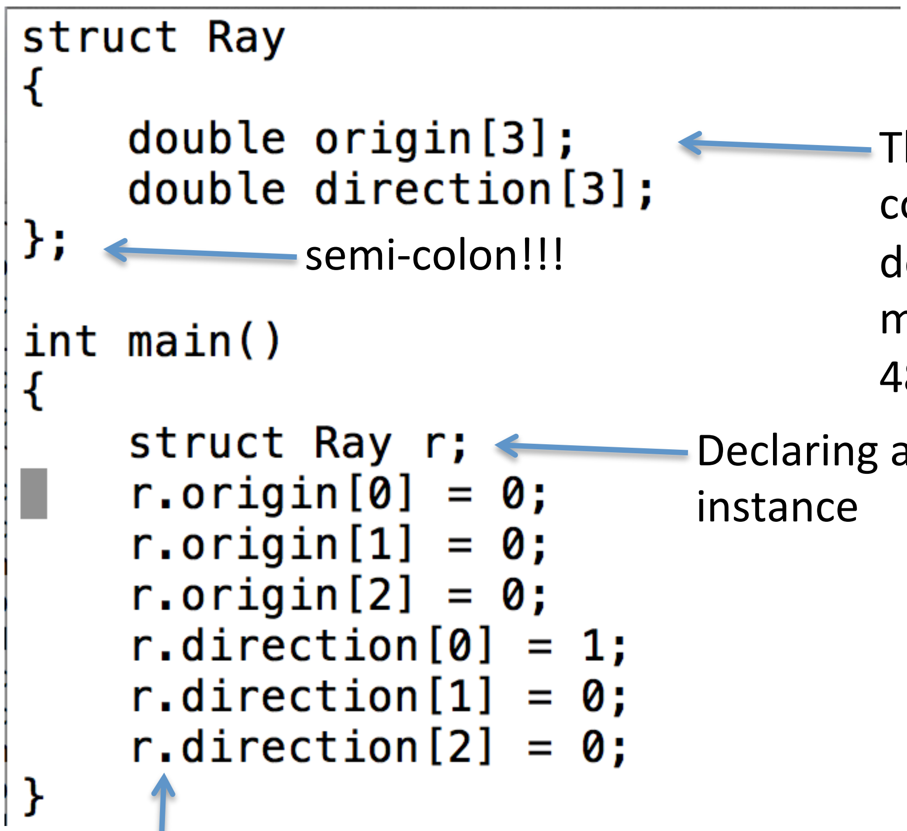
- float
- double
- int
- char
- unsigned char

All of these are simple data types

Structs: a complex data type

- Structs: mechanism provided by C programming language to define 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



```
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;
}
```

C keyword "struct" – means struct definition is coming

This struct contains 6 doubles, meaning it is 48 bytes

semi-colon!!!

Declaring an instance

"." accesses data members for a struct

Nested structs

```
struct Origin
{
    double originX;
    double originY;
    double originZ;
};

struct Direction
{
    double directionX;
    double directionY;
    double directionZ;
};

struct Ray
{
    struct Origin ori;
    struct Direction dir;
};
```

```
int main()
{
    struct Ray r;
    r.ori.originX = 0;
    r.ori.originY = 0;
    r.ori.originZ = 0;
    r.dir.directionX = 0;
    r.dir.directionY = 0;
    r.dir.directionZ = 0;
}
```

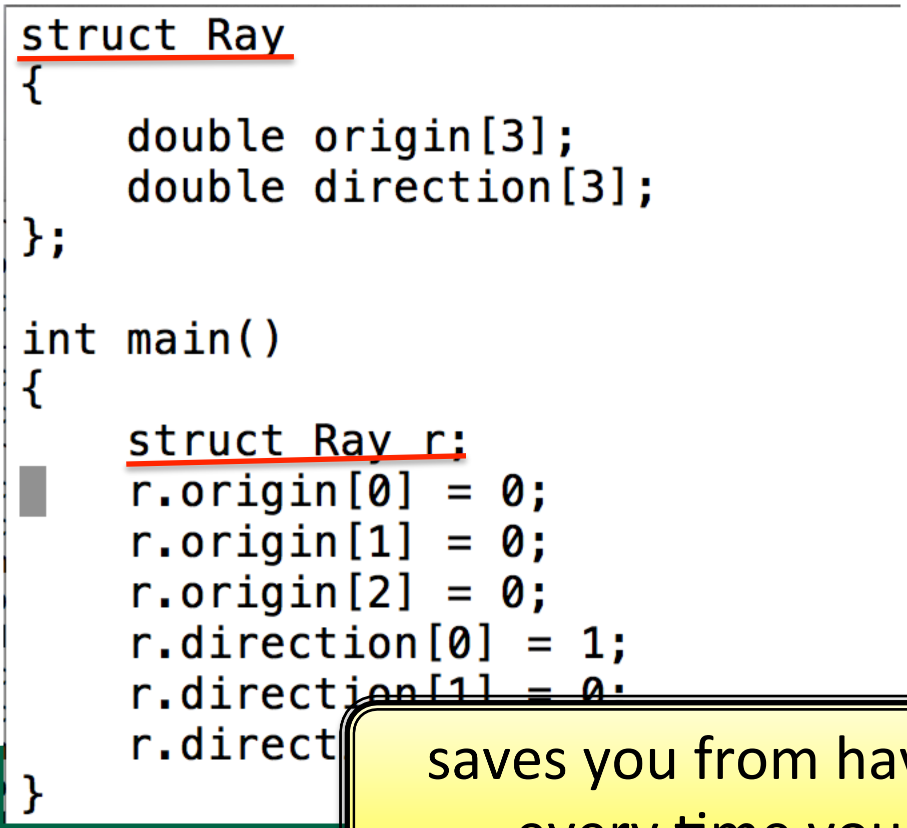
accesses dir part of Ray

accesses directionZ part of Direction (part of Ray)

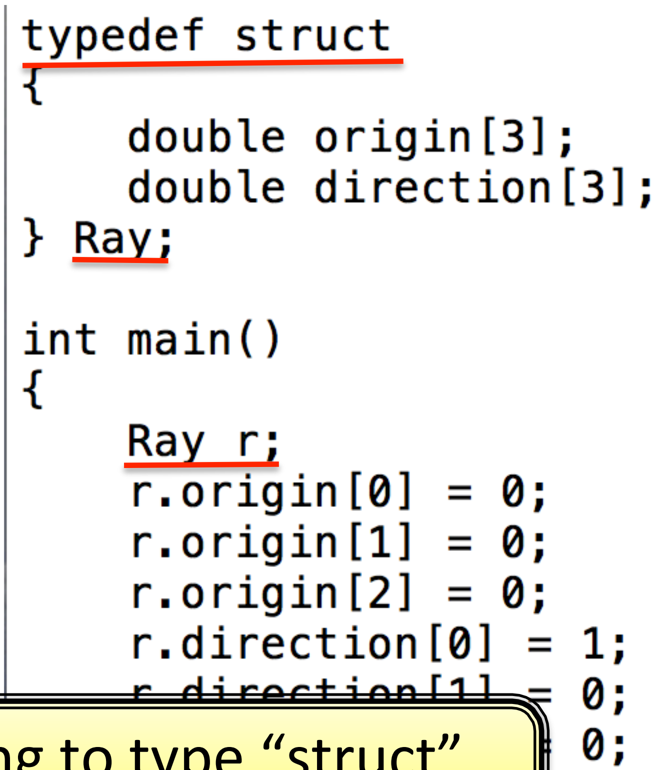
UNIVERSITY OF OREGON

typedef

- typedef: tell compiler you want to define a new type



```
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.direct
}
```



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

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

saves you from having to type "struct" every time you declare a struct.

Other uses for typedef

- Declare a new type for code clarity
  - typedef int MilesPerHour;
    - Makes a new type called MilesPerHour.
    - MilesPerHour works exactly like an int.
- Also used for enums & unions
  - same trick as for structs … typedef saves you a word

Outline

- Grade 4A
- Review
- Project 2B
- Enum
- Struct
- Union
- Project 2C

Unions

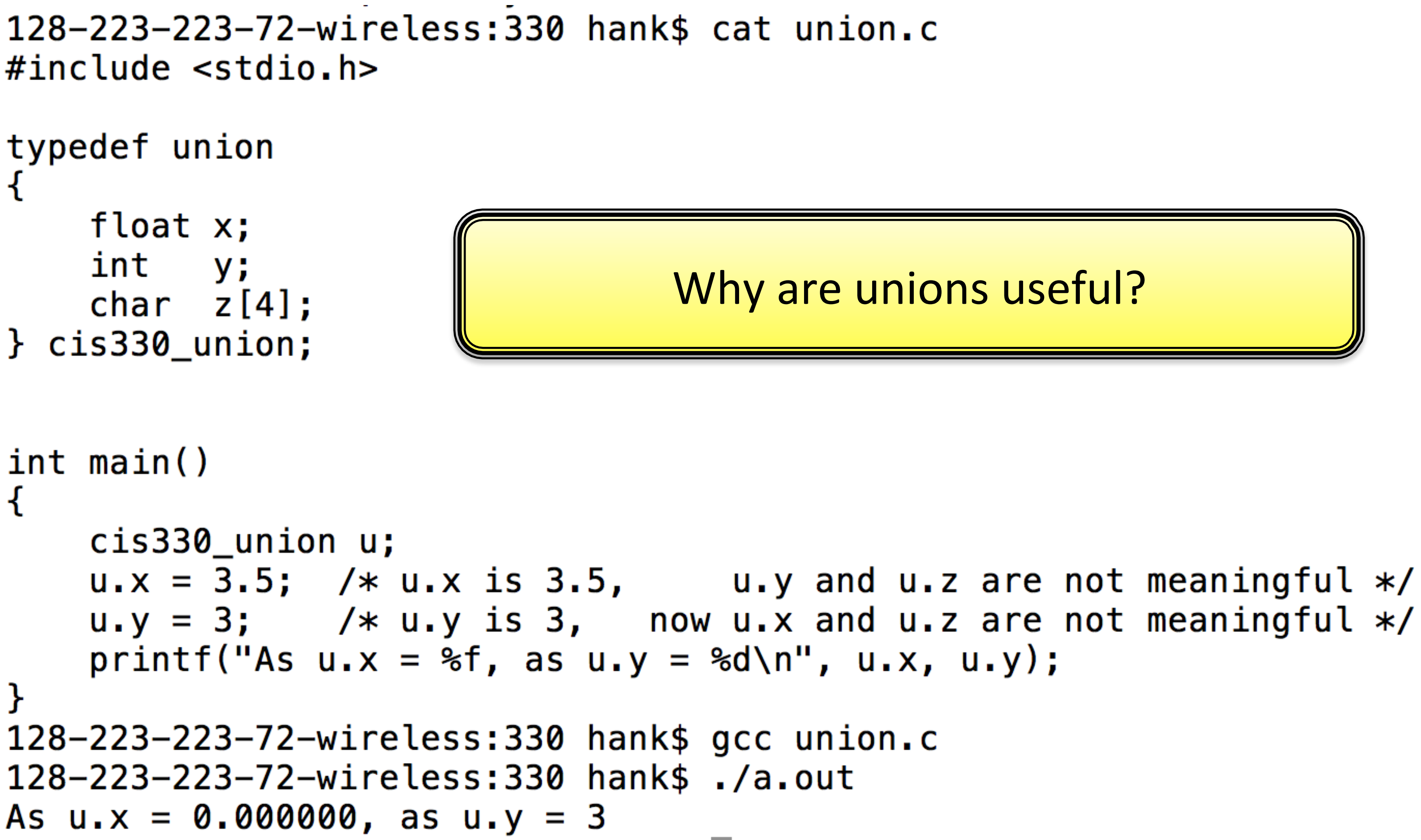
- Union: special data type
  - store many different memory types in one memory location

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

When dealing with this union, you can treat it as a float, as an int, or as 4 characters.

This data structure has 4 bytes

Unions

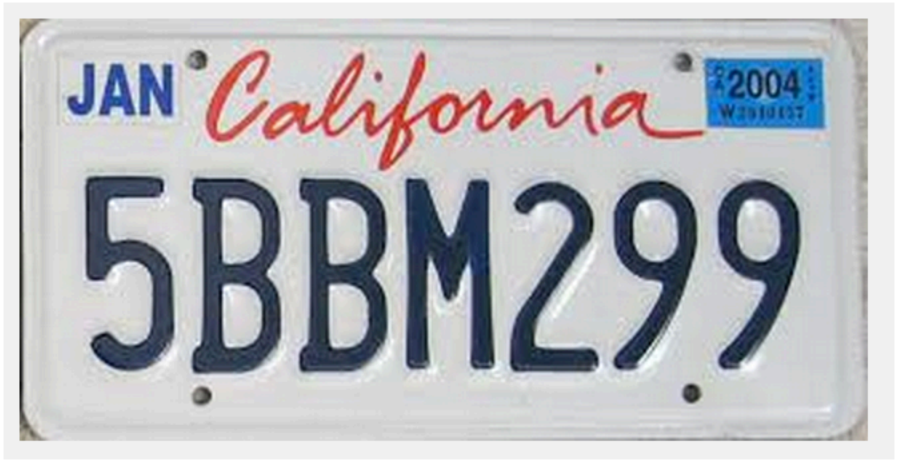


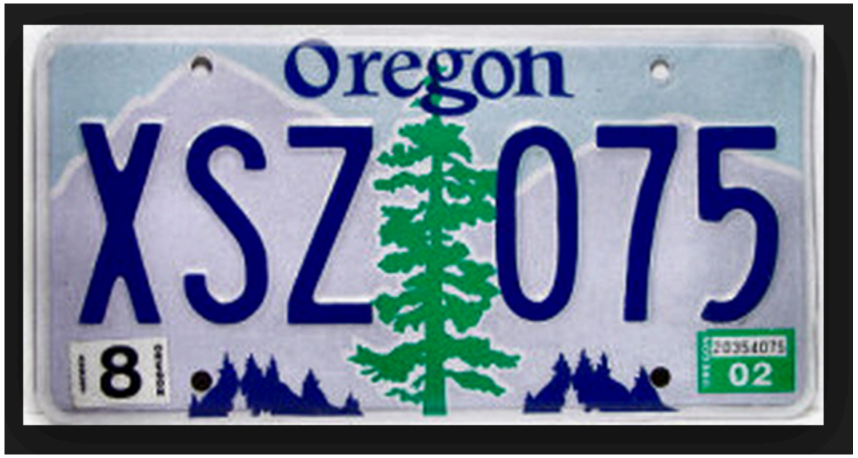
```
128-223-223-72-wireless:330 hank$ cat union.c
#include <stdio.h>

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
```

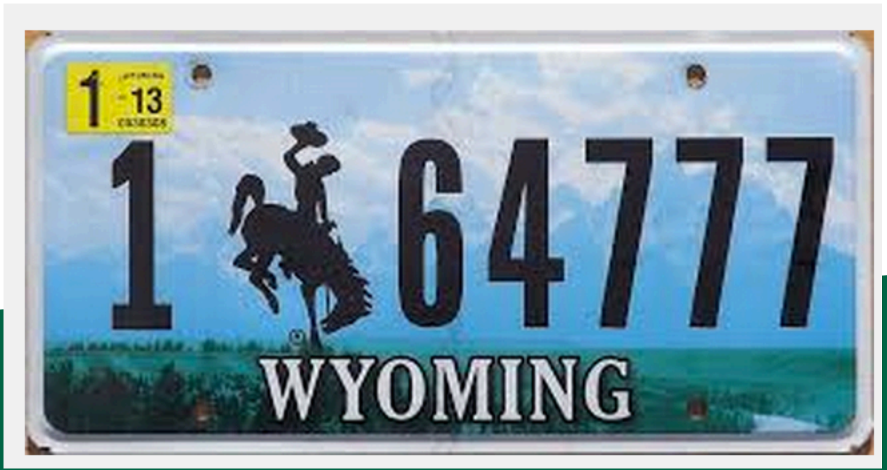
Why are unions useful?

Unions Example

```
typedef struct
{
    int firstNum;
    char letters[3];
    int endNums[3];
} CA_LICENSE_PLATE;

typedef struct
{
    char  letters[3];
    int   nums[3];
} OR_LICENSE_PLATE;

typedef struct
{
    int   nums[6];
} WY_LICENSE_PLATE;

typedef union
{
    CA_LICENSE_PLATE ca;
    OR_LICENSE_PLATE or;
    WY_LICENSE_PLATE wy;
} LicensePlate;
```







Unions Example

```
typedef struct
{
   int firstNum;
   char letters[3];
   int endNums[3];
} CA_LICENSE_PLATE;

typedef struct
{
   char  letters[3];
   int   nums[3];
} OR_LICENSE_PLATE;

typedef struct
{
   int   nums[6];
} WY_LICENSE_PLATE;

typedef union
{
   CA_LICENSE_PLATE ca;
   OR_LICENSE_PLATE or;
   WY_LICENSE_PLATE wy;
} LicensePlate;
```

```
typedef enum
{
   CA,
   OR,
   WY
} US_State;

typedef struct
{
   char *carMake;
   char *carModel;
   US_State  state;
   LicensePlate lp;
} CarInfo;

int main()
{
    CarInfo c;
    c.carMake = "Chevrolet";
    c.carModel = "Camaro";
    c.state = OR;
    c.lp.or.letters[0] = 'X';
    c.lp.or.letters[1] = 'S';
    c.lp.or.letters[2] = 'Z';
    c.lp.or.nums[0] = 0;
    c.lp.or.nums[1] = 7;
    c.lp.or.nums[2] = 5;
}
```

Outline

- Grade 4A
- Review
- Project 2B
- Enum
- Struct
- Unions
- Project 2C

Project 2C

Worth 4% of your grade

Assignment: You will implement 3 structs and 9 functions. The prototypes for the functions are located in the file prototypes.h (available on the website).

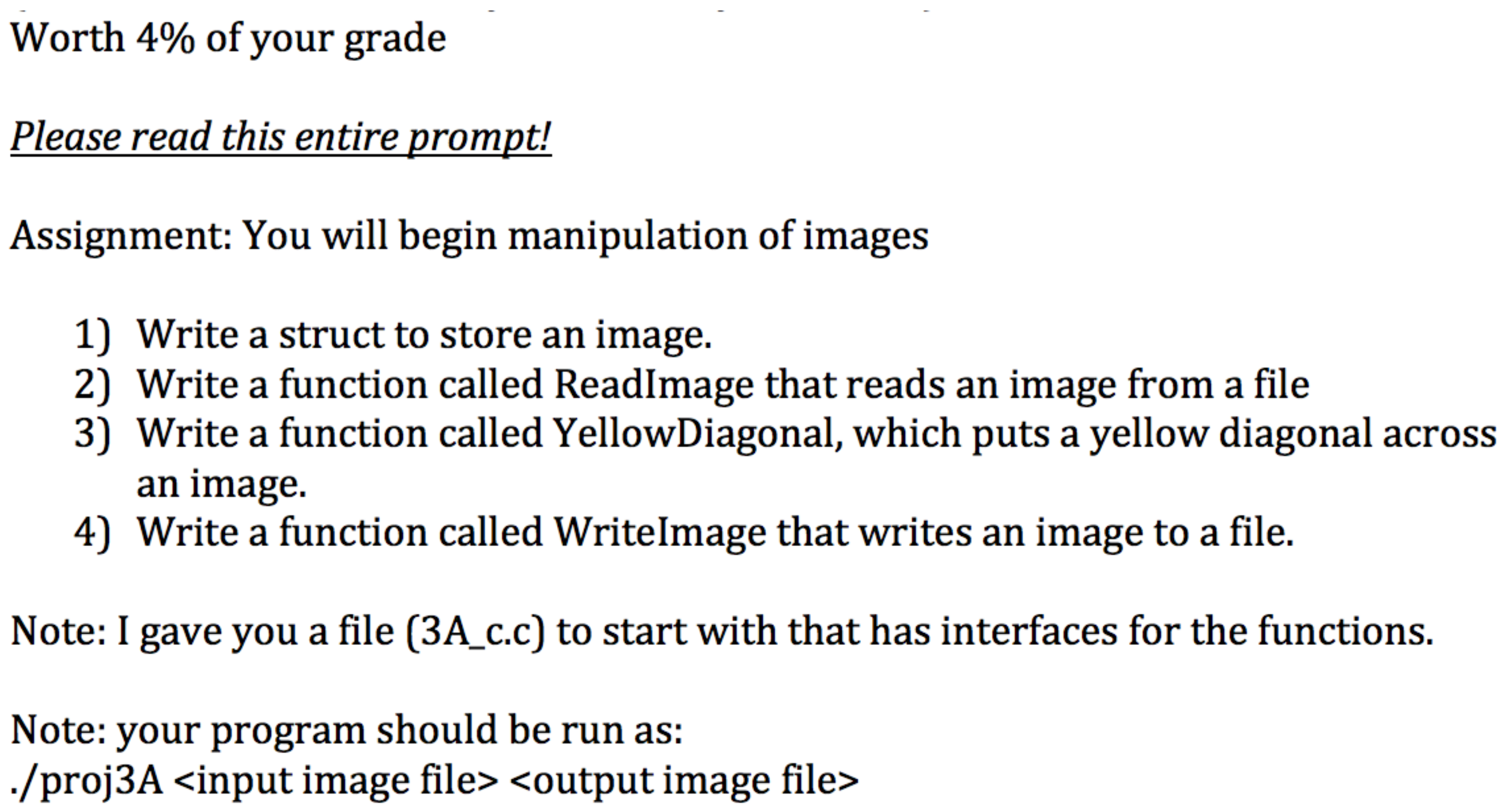
The three structs are Rectangle, Circle, and Triangle, and are described below.

The 3 structs refer to 3 different shapes: Triangle, Circle, and Rectangle.
For each shape, there are 3 functions: Initialize, GetArea, and GetBoundingBox.
You must implement 9 functions total (3*3).

The prototypes for these 9 functions are available in the file prototypes.h

There is also a driver program, and correct output for the driver program.

Project 3A



Worth 4% of your grade

__Please read this entire prompt!__

Assignment: You will begin manipulation of images

1) Write a struct to store an image.
2) Write a function called ReadImage that reads an image from a file
3) Write a function called YellowDiagonal, which puts a yellow diagonal across an image.
4) Write a function called WriteImage that writes an image to a file.

Note: I gave you a file (3A_c.c) to start with that has interfaces for the functions.

Note: your program should be run as:
./proj3A <input image file> <output image file>

Function Pointers

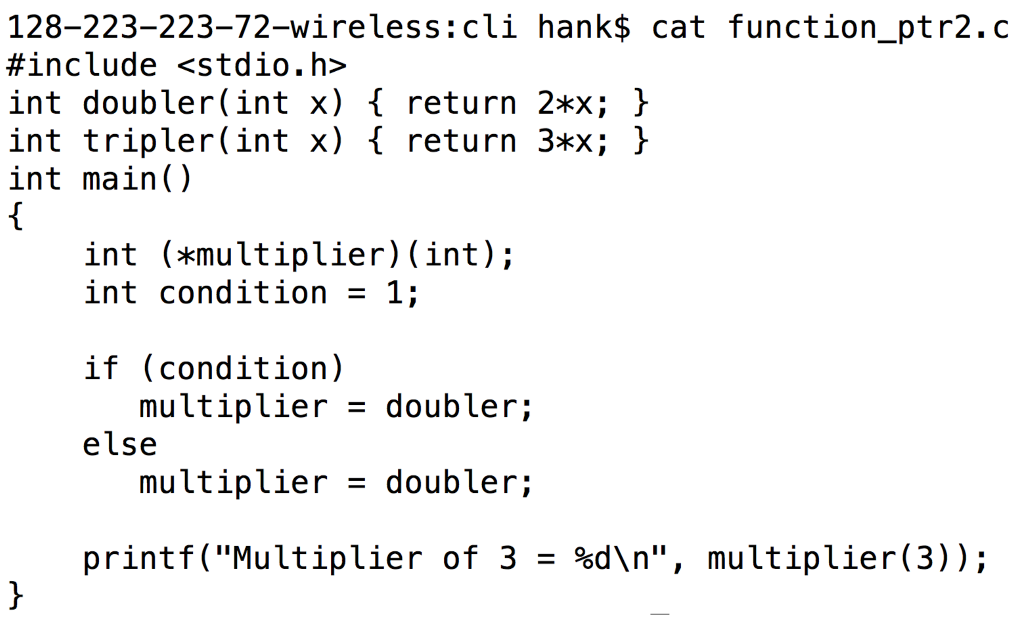
Function Pointers

- Idea:
  - You have a pointer to a function
  - This pointer can change based on circumstance
  - When you call the function pointer, it is like calling a known function

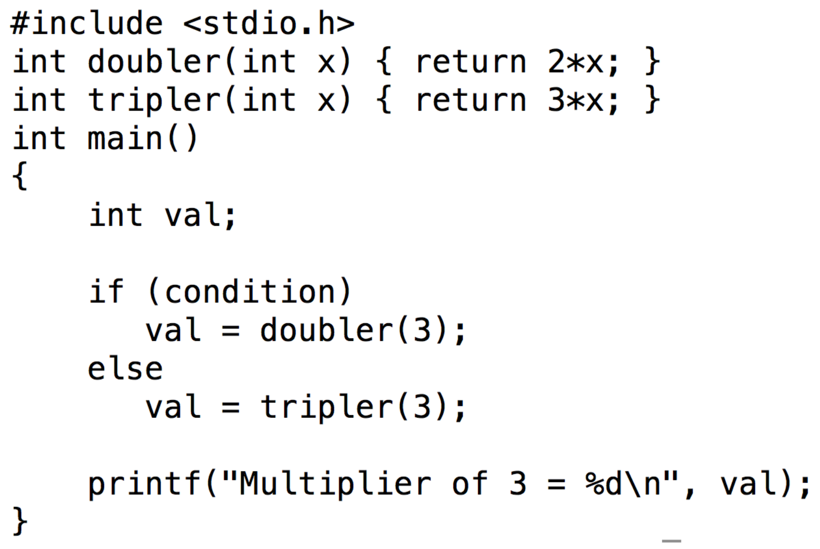
Function Pointer Example

```
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 Pointers vs Conditionals



```
128-223-223-72-wireless:cli hank$ cat function_ptr2.c
#include <stdio.h>
int doubler(int x) { return 2*x; }
int tripler(int x) { return 3*x; }
int main()
{
    int (*multiplier)(int);
    int condition = 1;

    if (condition)
       multiplier = doubler;
    else
       multiplier = doubler;

    printf("Multiplier of 3 = %d\n", multiplier(3));
}
```



```
#include <stdio.h>
int doubler(int x) { return 2*x; }
int tripler(int x) { return 3*x; }
int main()
{
    int val;

    if (condition)
       val = doubler(3);
    else
       val = tripler(3);

    printf("Multiplier of 3 = %d\n", val);
}
```

What are the pros and cons of each approach?

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

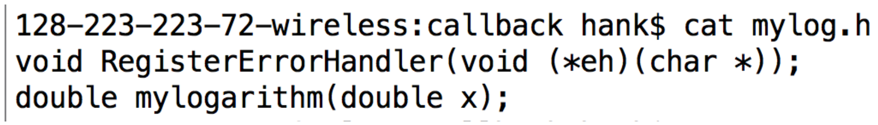
```
void (*foo)(void);
void (*foo)(int **, char ***);
char ** (*foo)(int **, void (*)(int));
```

These sometimes come up on interviews.

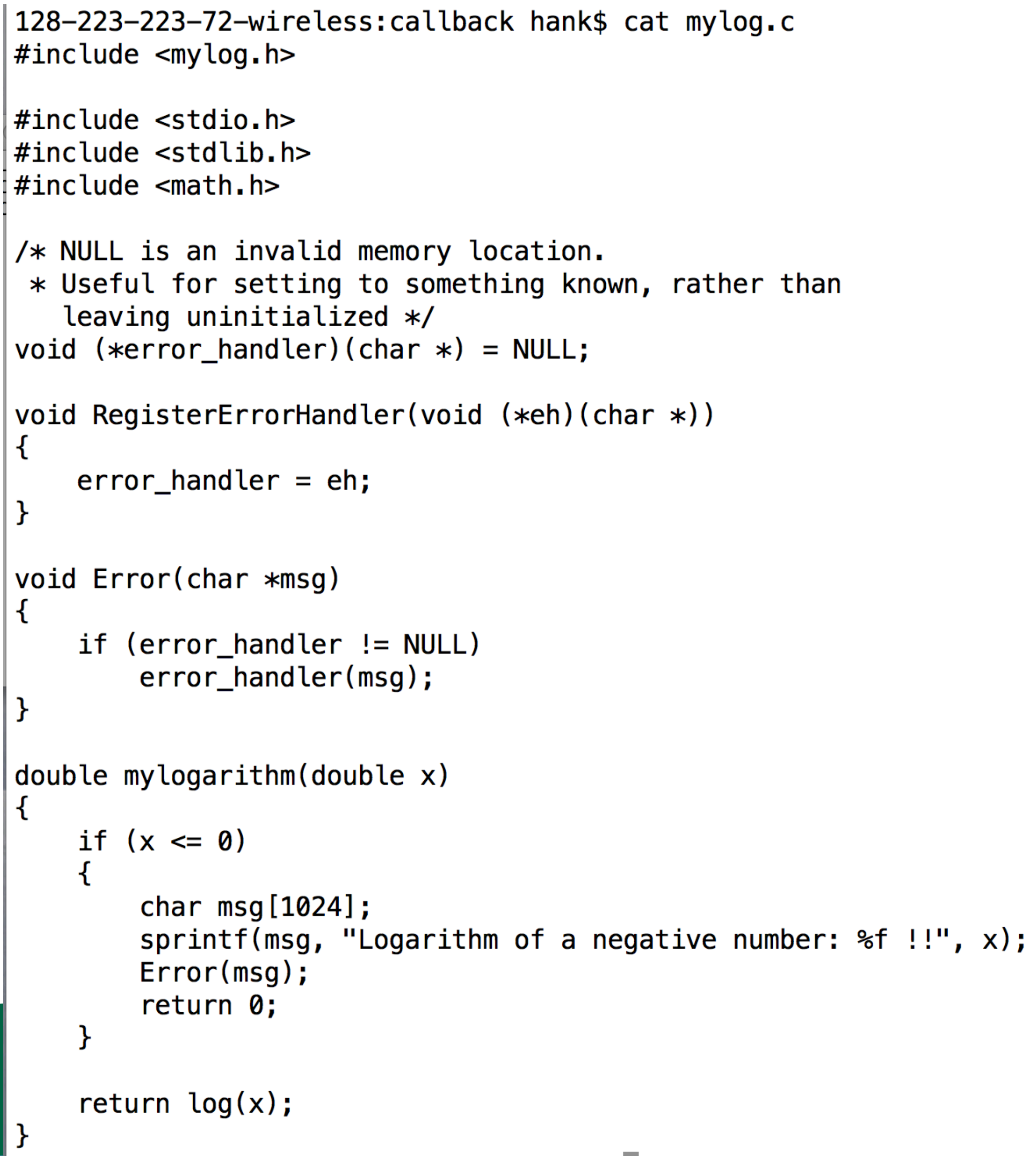
Callbacks

- Callbacks: function that is called when a condition is met
  - Commonly used when interfacing between modules that were developed separately.
  - … libraries use callbacks and developers who use the libraries “register” callbacks.

Callback example



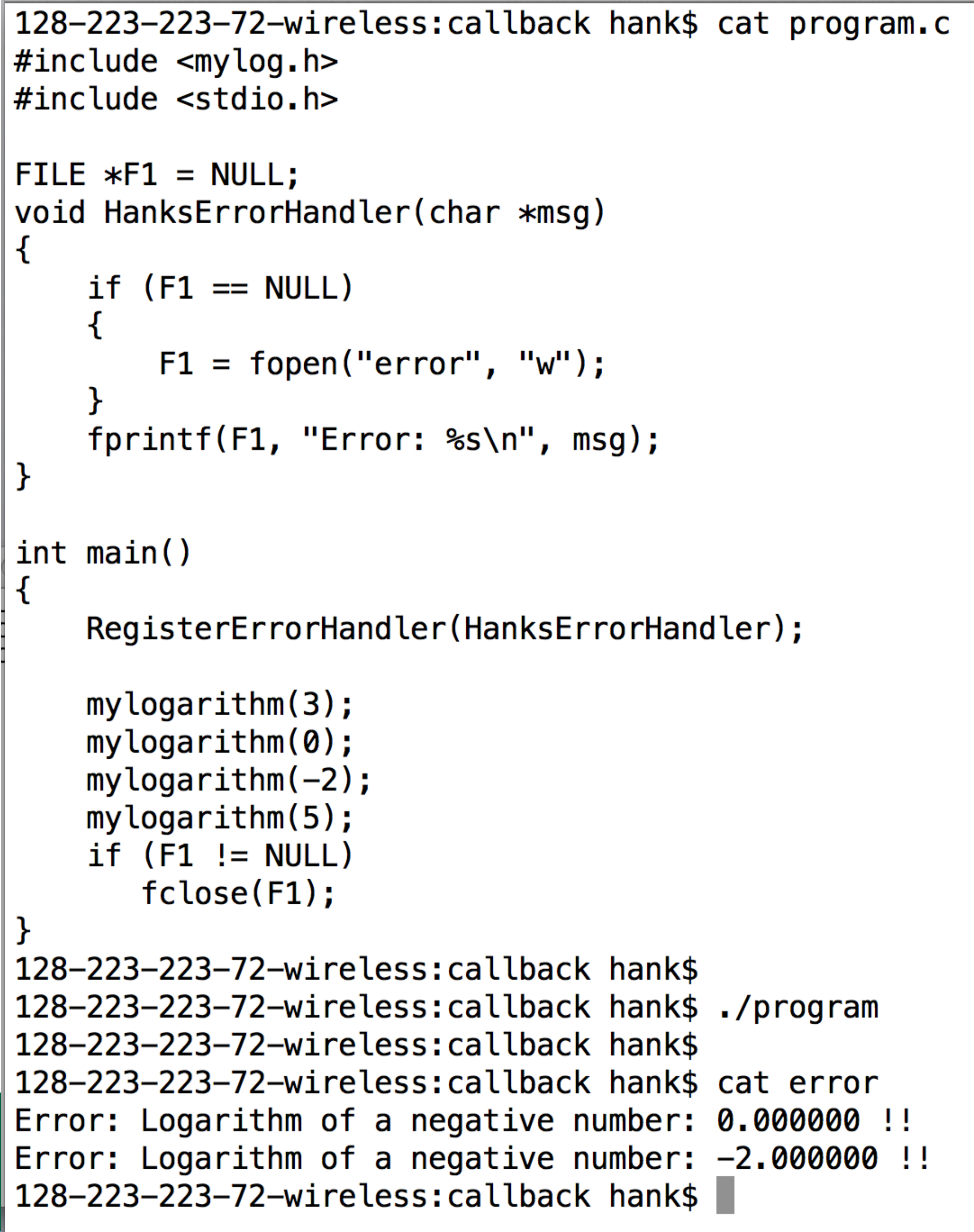
```
128-223-223-72-wireless:callback hank$ cat mylog.h
void RegisterErrorHandler(void (*eh)(char *));
double mylogarithm(double x);
```



```
128-223-223-72-wireless:callback hank$ cat mylog.c
#include <mylog.h>

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* NULL is an invalid memory location.
 * Useful for setting to something known, rather than
   leaving uninitialized */
void (*error_handler)(char *) = NULL;

void RegisterErrorHandler(void (*eh)(char *))
{
    error_handler = eh;
}

void Error(char *msg)
{
    if (error_handler != NULL)
        error_handler(msg);
}

double mylogarithm(double x)
{
    if (x <= 0)
    {
        char msg[1024];
        sprintf(msg, "Logarithm of a negative number: %f !!", x);
        Error(msg);
        return 0;
    }

    return log(x);
}
```

UNIVERSITY OF OREGON

Callback example



```
128-223-223-72-wireless:callback hank$ cat program.c
#include <mylog.h>
#include <stdio.h>

FILE *F1 = NULL;
void HanksErrorHandler(char *msg)
{
    if (F1 == NULL)
    {
        F1 = fopen("error", "w");
    }
    fprintf(F1, "Error: %s\n", msg);
}

int main()
{
    RegisterErrorHandler(HanksErrorHandler);

    mylogarithm(3);
    mylogarithm(0);
    mylogarithm(-2);
    mylogarithm(5);
    if (F1 != NULL)
       fclose(F1);
}
128-223-223-72-wireless:callback hank$
128-223-223-72-wireless:callback hank$ ./program
128-223-223-72-wireless:callback hank$
128-223-223-72-wireless:callback hank$ cat error
Error: Logarithm of a negative number: 0.000000 !!
Error: Logarithm of a negative number: -2.000000 !!
128-223-223-72-wireless:callback hank$
```

Function Pointers

- We are going to use function pointers to accomplish “sub-typing” in Project 2D.

Subtyping

Subtyping

- Type: a data type (int, float, structs)
- Subtype / supertype:
  - Supertype: the abstraction of a type
    - (not specific)
  - Subtype: a concrete implementation of the supertype
    - (specific)

The fancy term for this is “subtype polymorphism”

Subtyping: example

- Supertype: Shape
- Subtypes:
  - Circle
  - Rectangle
  - Triangle

Subtyping works via interfaces

- Must define an interface for supertype/ subtypes
  - Interfaces are the functions you can call on the supertype/subtypes

- The set of functions is fixed
  - Every subtype must define all functions

Subtyping

- I write my routines to the supertype interface
- All subtypes can automatically use this code
  - Don't have to modify code when new supertypes are added

- Example:
  - I wrote code about Shapes.
  - I don't care about details of subtypes (Triangle, Rectangle, Circle)
  - When new subtypes are added (Square), my code doesn't change

More Unix

"." and ".."

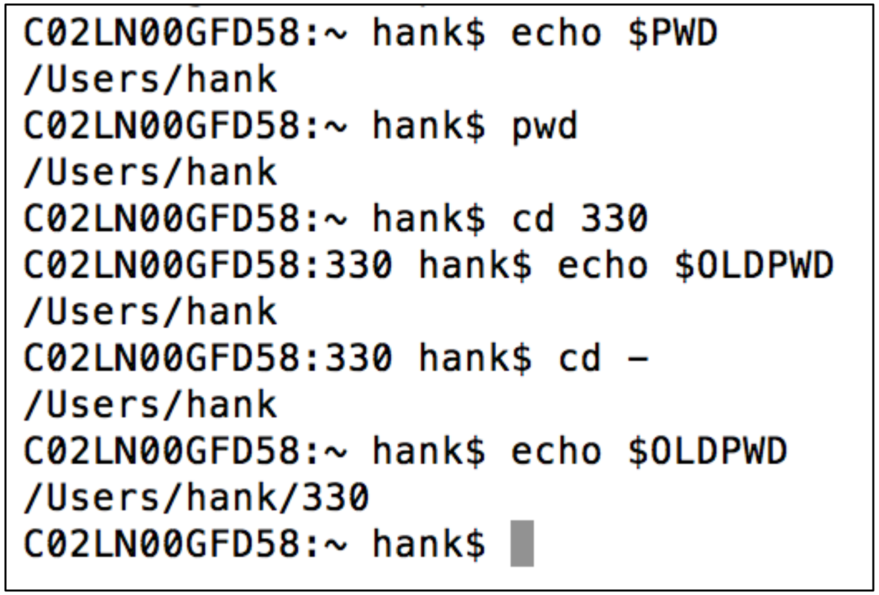
- Unix convention:
  - "." : the current directory
  - ".." : the parent directory

Quiz: you in /path/to/dir and issue "cd ./.././..". Where do you end up?

Answer: "/path"

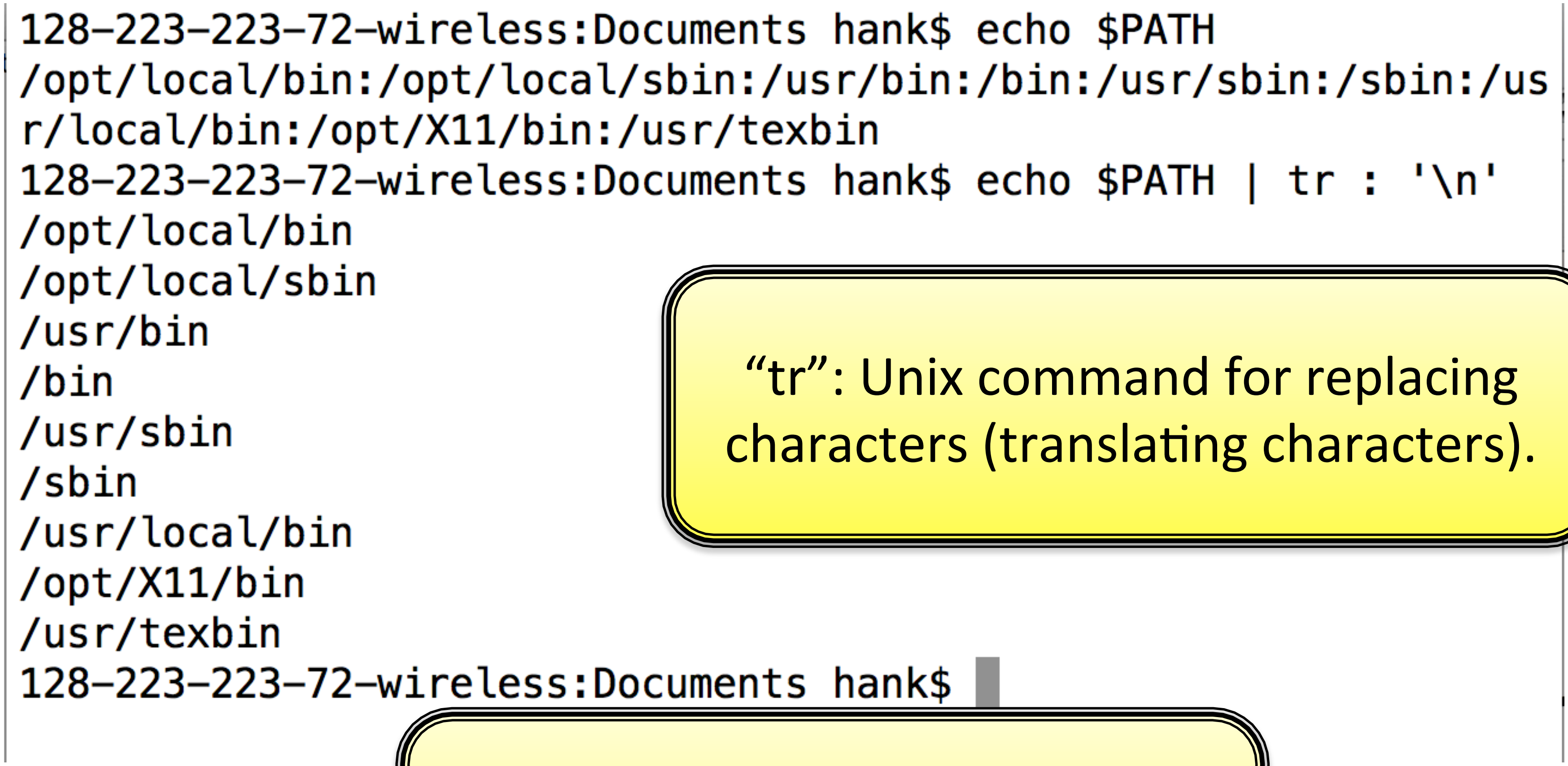
pwd and $PWD

- pwd: unix command that returns the "present working directory"
- $PWD : environment variable that contains the present working directory
- $OLDPWD : environment variable that contains the previous present working directory
- "-" : shortcut for the previous PWD



```
C02LN00GFD58:~ hank$ echo $PWD
/Users/hank
C02LN00GFD58:~ hank$ pwd
/Users/hank
C02LN00GFD58:~ hank$ cd 330
C02LN00GFD58:330 hank$ echo $OLDPWD
/Users/hank
C02LN00GFD58:330 hank$ cd -
/Users/hank
C02LN00GFD58:~ hank$ echo $OLDPWD
/Users/hank/330
C02LN00GFD58:~ hank$ 
```

PATH environment variable

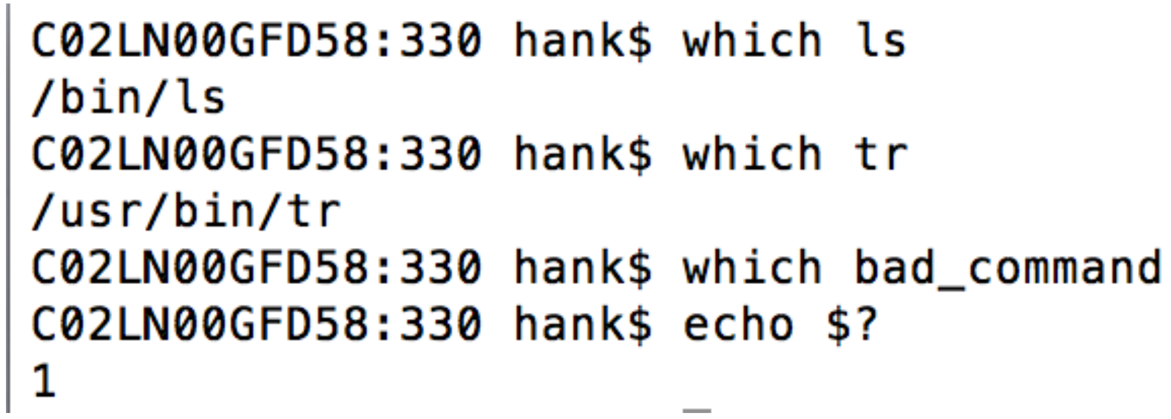


```
128-223-223-72-wireless:Documents hank$ echo $PATH
/opt/local/bin:/opt/local/sbin:/usr/bin:/bin:/usr/sbin:/sbin:/us
r/local/bin:/opt/X11/bin:/usr/texbin
128-223-223-72-wireless:Documents hank$ echo $PATH | tr : '\n'
/opt/local/bin
/opt/local/sbin
/usr/bin
/bin
/usr/sbin
/sbin
/usr/local/bin
/opt/X11/bin
/usr/texbin
128-223-223-72-wireless:Documents hank$
```

"tr": Unix command for replacing characters (translating characters).

When the shell wants to invoke a command, it searches for the command in the path

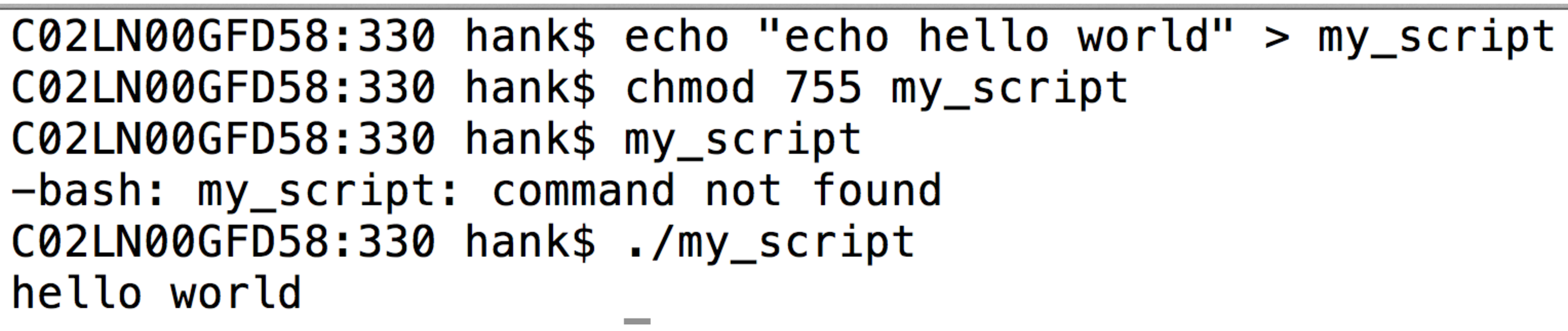
which



```
C02LN00GFD58:330 hank$ which ls
/bin/ls
C02LN00GFD58:330 hank$ which tr
/usr/bin/tr
C02LN00GFD58:330 hank$ which bad_command
C02LN00GFD58:330 hank$ echo $?
1
```

which: tells you the directory the shell is finding a command in.

Invoking programs in current directory

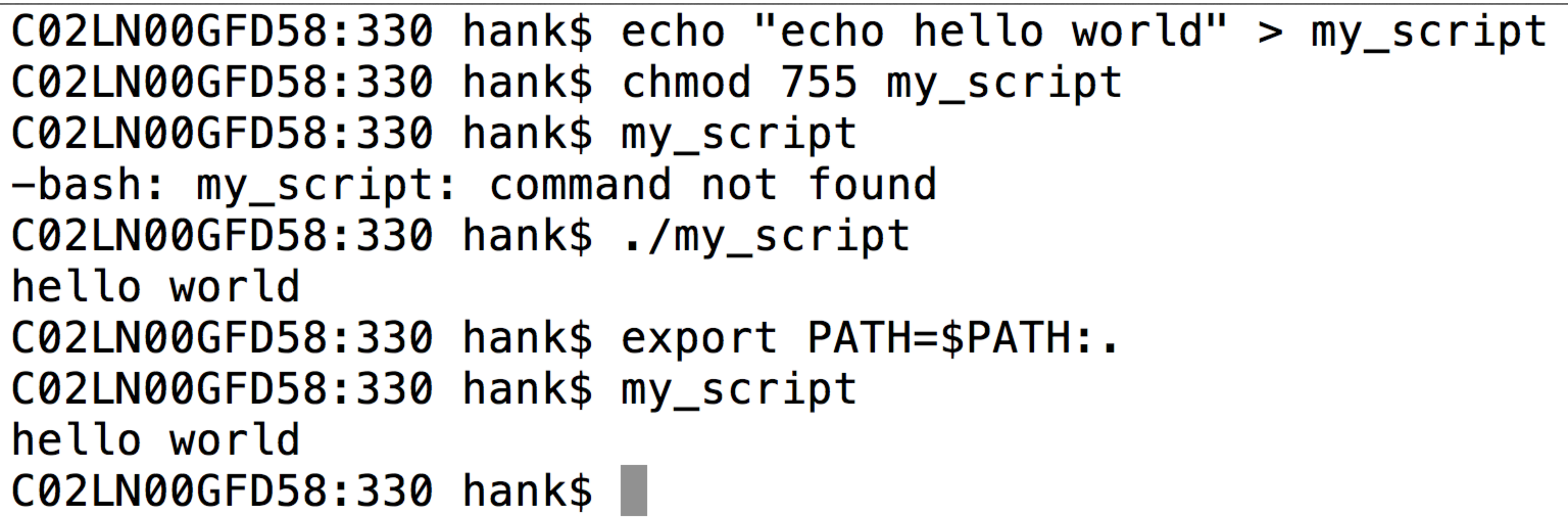


```
C02LN00GFD58:330 hank$ echo "echo hello world" > my_script
C02LN00GFD58:330 hank$ chmod 755 my_script
C02LN00GFD58:330 hank$ my_script
-bash: my_script: command not found
C02LN00GFD58:330 hank$ ./my_script
hello world
```

shell works with ./prog_name since it views this as a path. Hence $PATH is ignored.

UNIVERSITY OF OREGON

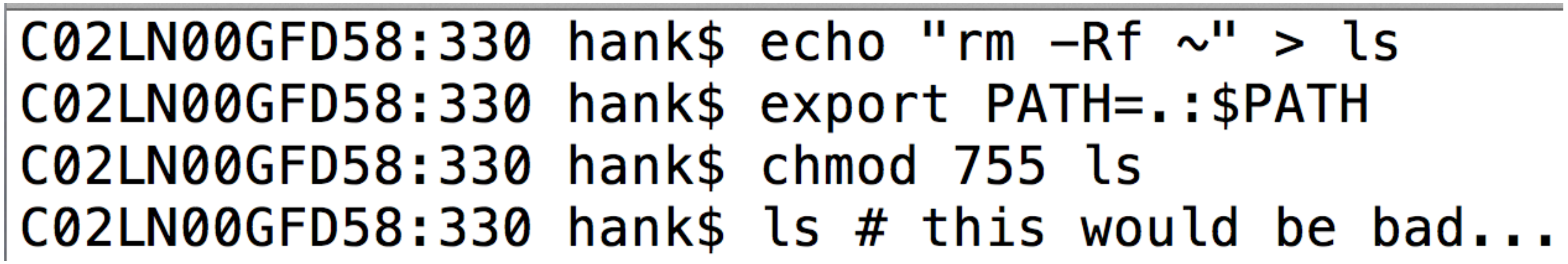
Invoking programs in current directory



```
C02LN00GFD58:330 hank$ echo "echo hello world" > my_script
C02LN00GFD58:330 hank$ chmod 755 my_script
C02LN00GFD58:330 hank$ my_script
-bash: my_script: command not found
C02LN00GFD58:330 hank$ ./my_script
hello world
C02LN00GFD58:330 hank$ export PATH=$PATH:.
C02LN00GFD58:330 hank$ my_script
hello world
C02LN00GFD58:330 hank$
```

Trojan Horse Attack

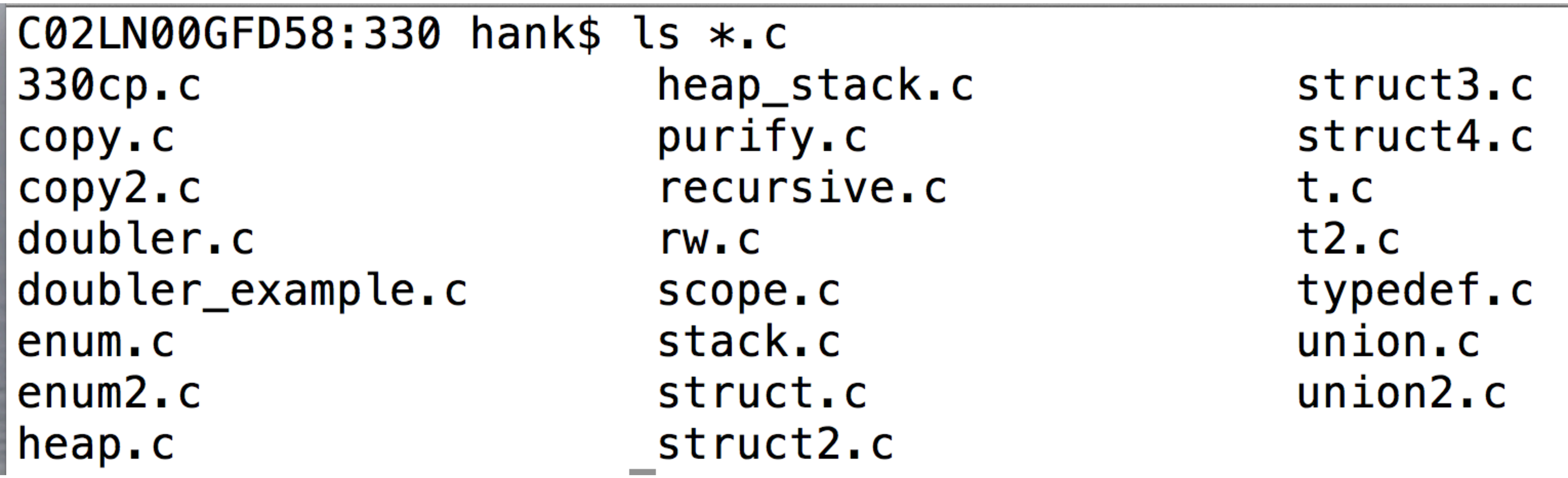
- export PATH=.:$PATH
  - why is this a terrible idea?



```
C02LN00GFD58:330 hank$ echo "rm -Rf ~" > ls
C02LN00GFD58:330 hank$ export PATH=.:$PATH
C02LN00GFD58:330 hank$ chmod 755 ls
C02LN00GFD58:330 hank$ ls # this would be bad...
```

Wild Cards

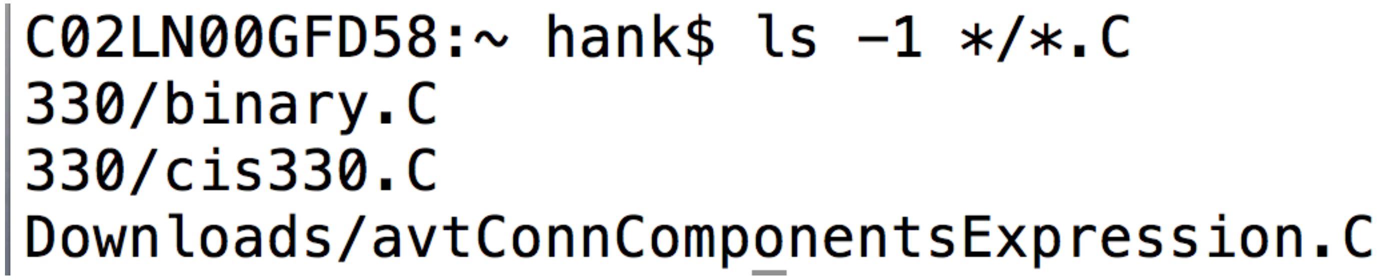
- '*' (asterisk) serves as a wild card that does pattern matching



```
C02LN00GFD58:330 hank$ ls *.c
330cp.c                 heap_stack.c            struct3.c
copy.c                  purify.c                struct4.c
copy2.c                 recursive.c             t.c
doubler.c               rw.c                    t2.c
doubler_example.c       scope.c                 typedef.c
enum.c                  stack.c                 union.c
enum2.c                 struct.c                union2.c
heap.c                  struct2.c
```

Wild Cards

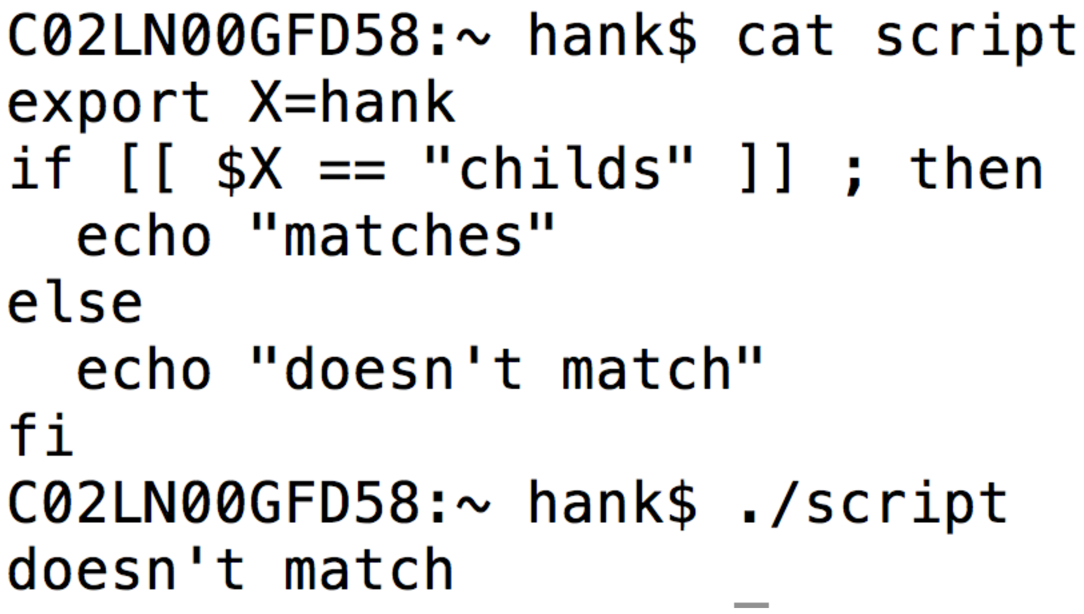
- You can use multiple asterisks for complex patterns



```
C02LN00GFD58:~ hank$ ls -1 */*.C
330/binary.C
330/cis330.C
Downloads/avtConnComponentsExpression.C
```

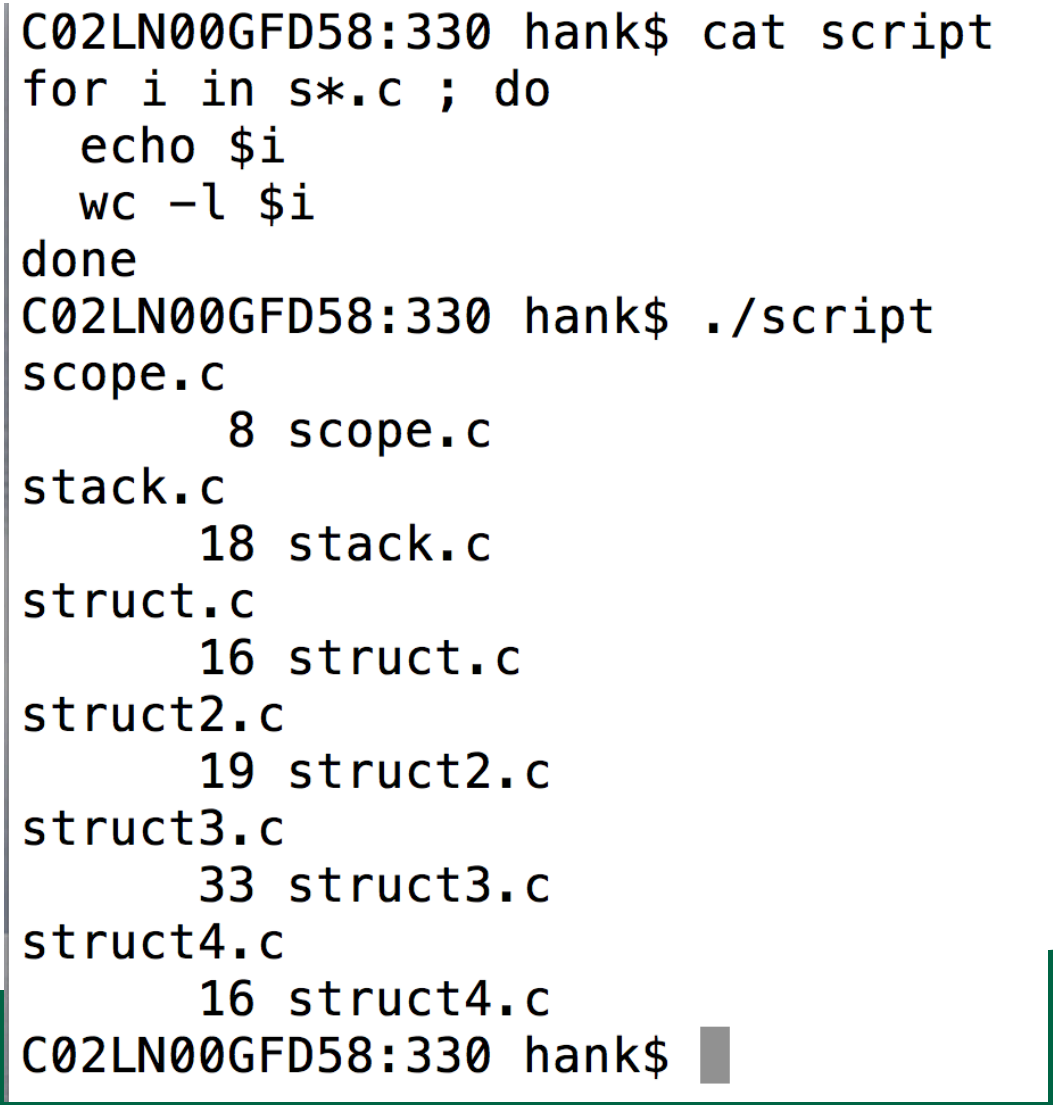
if / then / else / fi

- Advanced constructs:



```
C02LN00GFD58:~ hank$ cat script
export X=hank
if [[ $X == "childs" ]] ; then
  echo "matches"
else
  echo "doesn't match"
fi
C02LN00GFD58:~ hank$ ./script
doesn't match
```

for / do / done



```
C02LN00GFD58:330 hank$ cat script
for i in s*.c ; do
  echo $i
  wc -l $i
done
C02LN00GFD58:330 hank$ ./script
scope.c
       8 scope.c
stack.c
      18 stack.c
struct.c
      16 struct.c
struct2.c
      19 struct2.c
struct3.c
      33 struct3.c
struct4.c
      16 struct4.c
C02LN00GFD58:330 hank$ 
```

-f and -d

- -f : does a file exist?
- -d : does a directory exist?

example:

```
if [[ ! -d include ]] ; then mkdir include ; fi
```

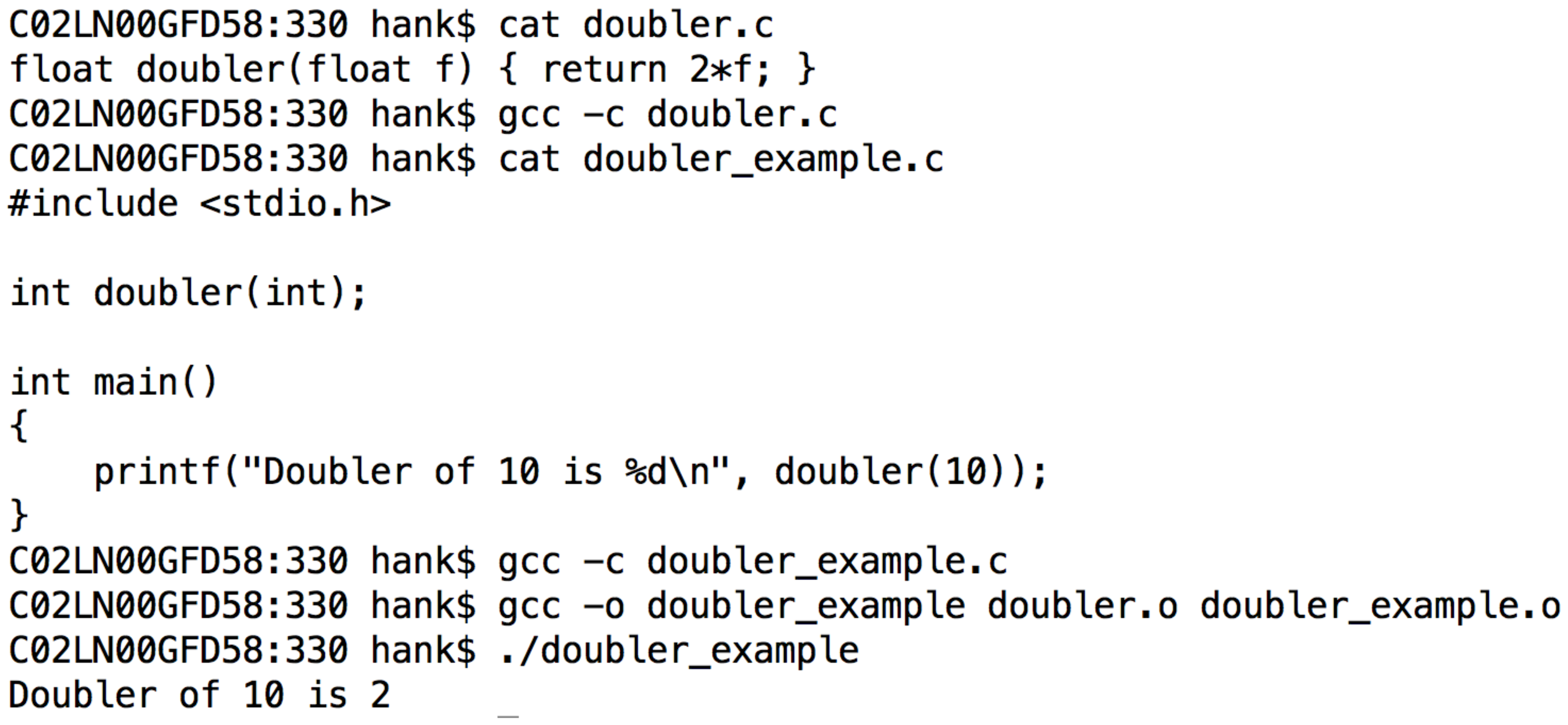
Why are Unions useful?

- 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.

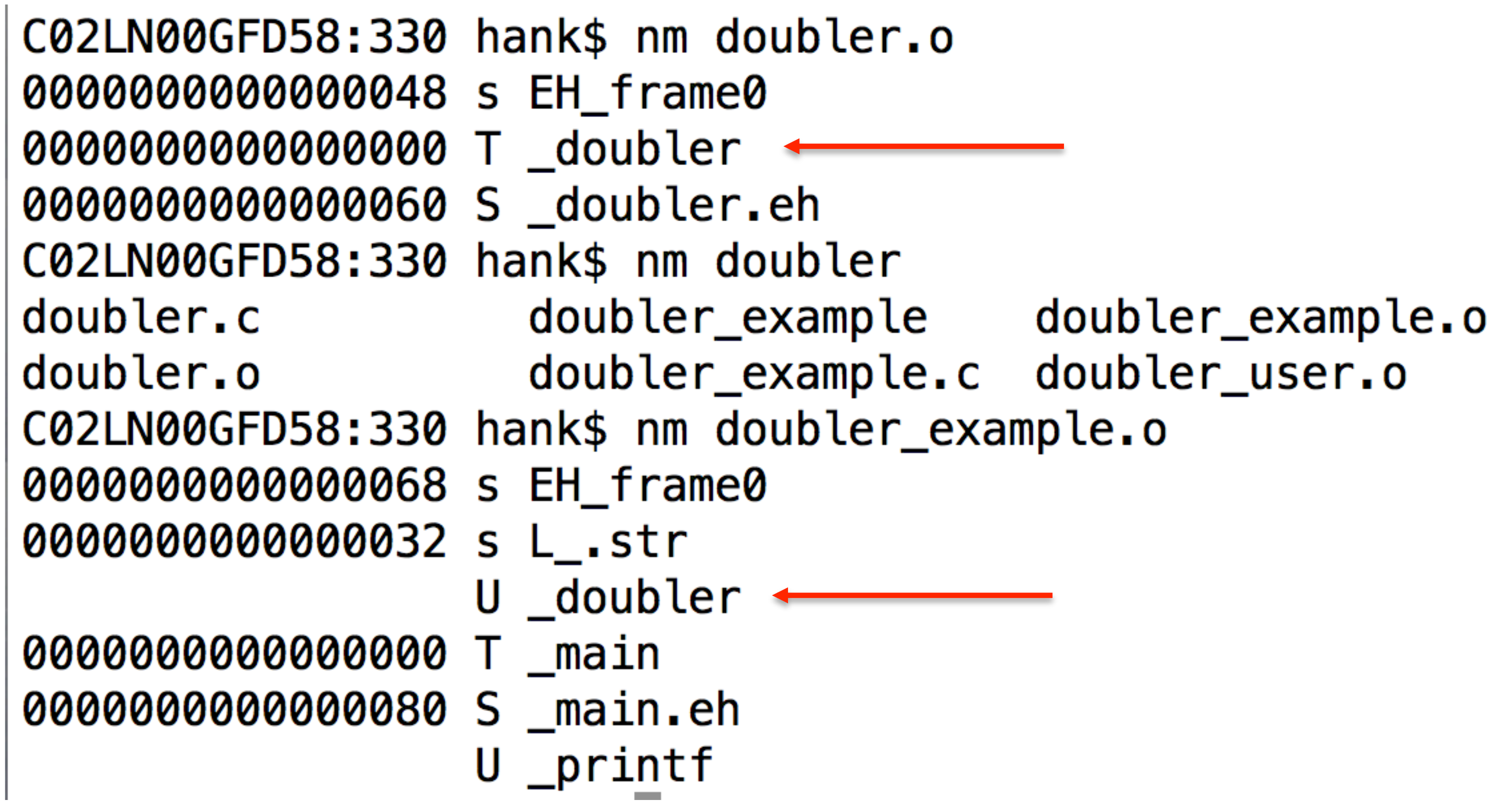
Bonus Material

Problem with C…



```
C02LN00GFD58:330 hank$ cat doubler.c
float doubler(float f) { return 2*f; }
C02LN00GFD58:330 hank$ gcc -c doubler.c
C02LN00GFD58:330 hank$ cat doubler_example.c
#include <stdio.h>

int doubler(int);

int main()
{
    printf("Doubler of 10 is %d\n", doubler(10));
}
C02LN00GFD58:330 hank$ gcc -c doubler_example.c
C02LN00GFD58:330 hank$ gcc -o doubler_example doubler.o doubler_example.o
C02LN00GFD58:330 hank$ ./doubler_example
Doubler of 10 is 2
```

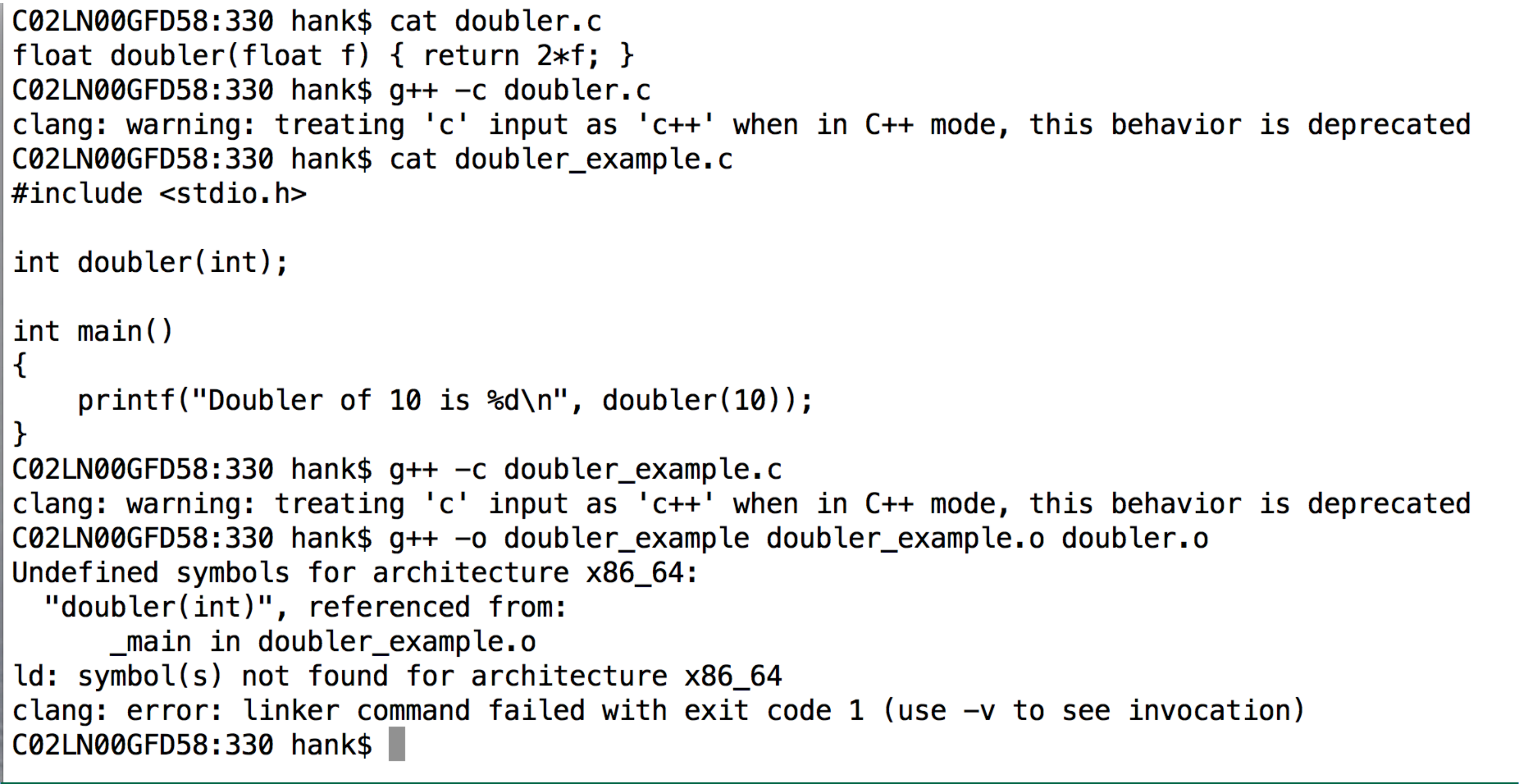
Problem with C…



```
C02LN00GFD58:330 hank$ nm doubler.o
0000000000000048 s EH_frame0
0000000000000000 T _doubler
0000000000000060 S _doubler.eh
C02LN00GFD58:330 hank$ nm doubler
doubler.c           doubler_example     doubler_example.o
doubler.o           doubler_example.c   doubler_user.o
C02LN00GFD58:330 hank$ nm doubler_example.o
0000000000000068 s EH_frame0
0000000000000032 s L_.str
                 U _doubler
0000000000000000 T _main
0000000000000080 S _main.eh
                 U _printf
```

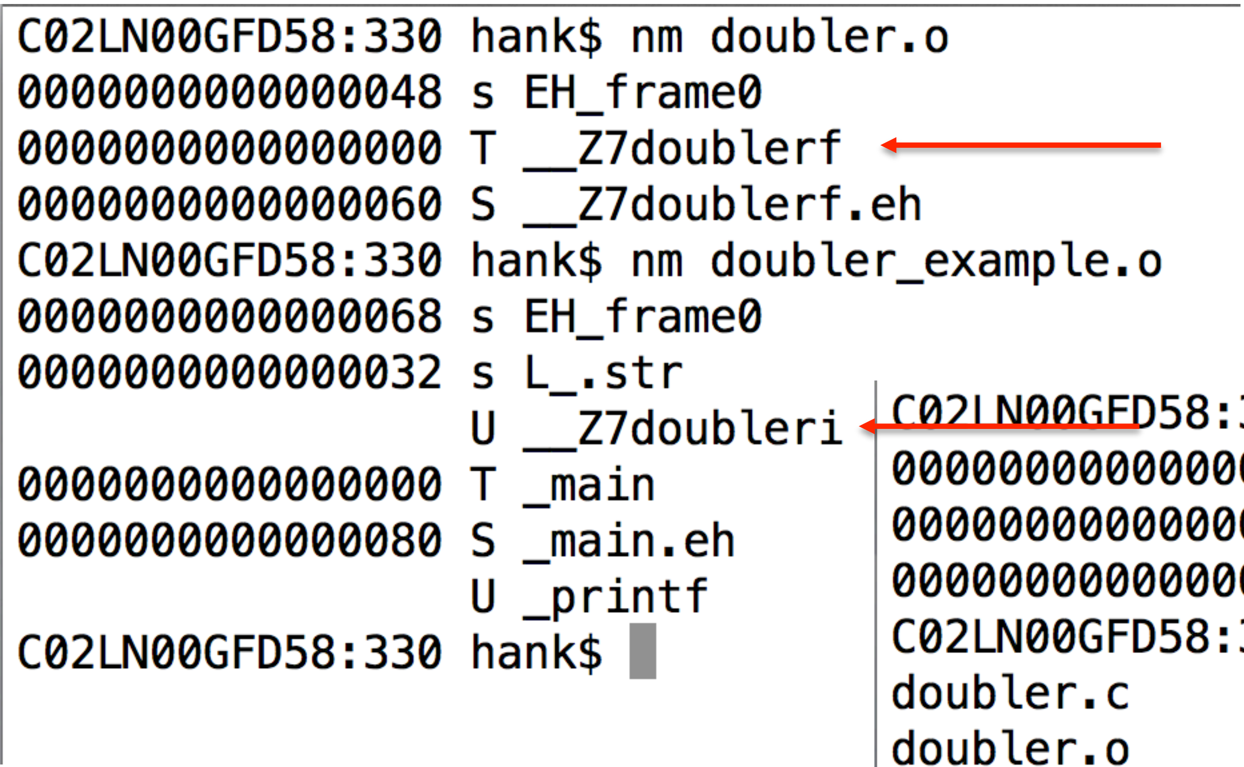
No checking of type…

Problem is fixed with C++…

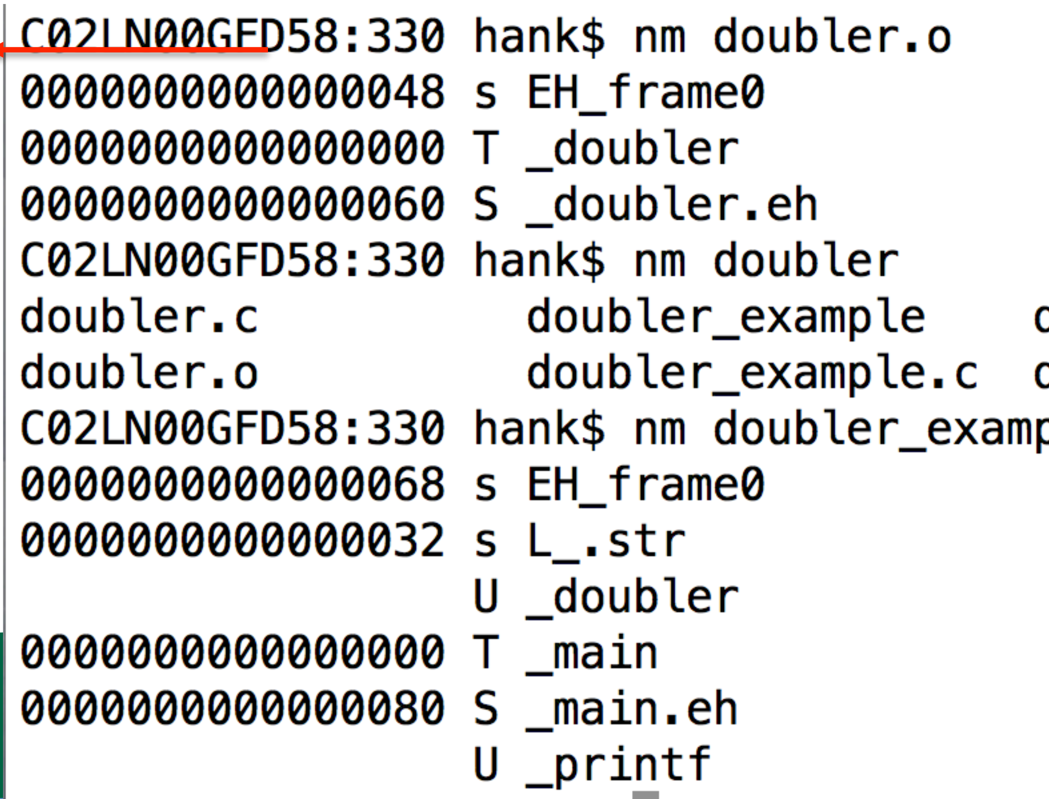


```
C02LN00GFD58:330 hank$ cat doubler.c
float doubler(float f) { return 2*f; }
C02LN00GFD58:330 hank$ g++ -c doubler.c
clang: warning: treating 'c' input as 'c++' when in C++ mode, this behavior is deprecated
C02LN00GFD58:330 hank$ cat doubler_example.c
#include <stdio.h>

int doubler(int);

int main()
{
    printf("Doubler of 10 is %d\n", doubler(10));
}
C02LN00GFD58:330 hank$ g++ -c doubler_example.c
clang: warning: treating 'c' input as 'c++' when in C++ mode, this behavior is deprecated
C02LN00GFD58:330 hank$ g++ -o doubler_example doubler_example.o doubler.o
Undefined symbols for architecture x86_64:
  "doubler(int)", referenced from:
      _main in doubler_example.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
C02LN00GFD58:330 hank$
```

Problem is fixed with C++…



```
C02LN00GFD58:330 hank$ nm doubler.o
0000000000000048 s EH_frame0
0000000000000000 T __Z7doublerf
0000000000000060 S __Z7doublerf.eh
C02LN00GFD58:330 hank$ nm doubler_example.o
0000000000000068 s EH_frame0
0000000000000032 s L_.str
                 U __Z7doubleri
0000000000000000 T _main
0000000000000080 S _main.eh
                 U _printf
C02LN00GFD58:330 hank$
```

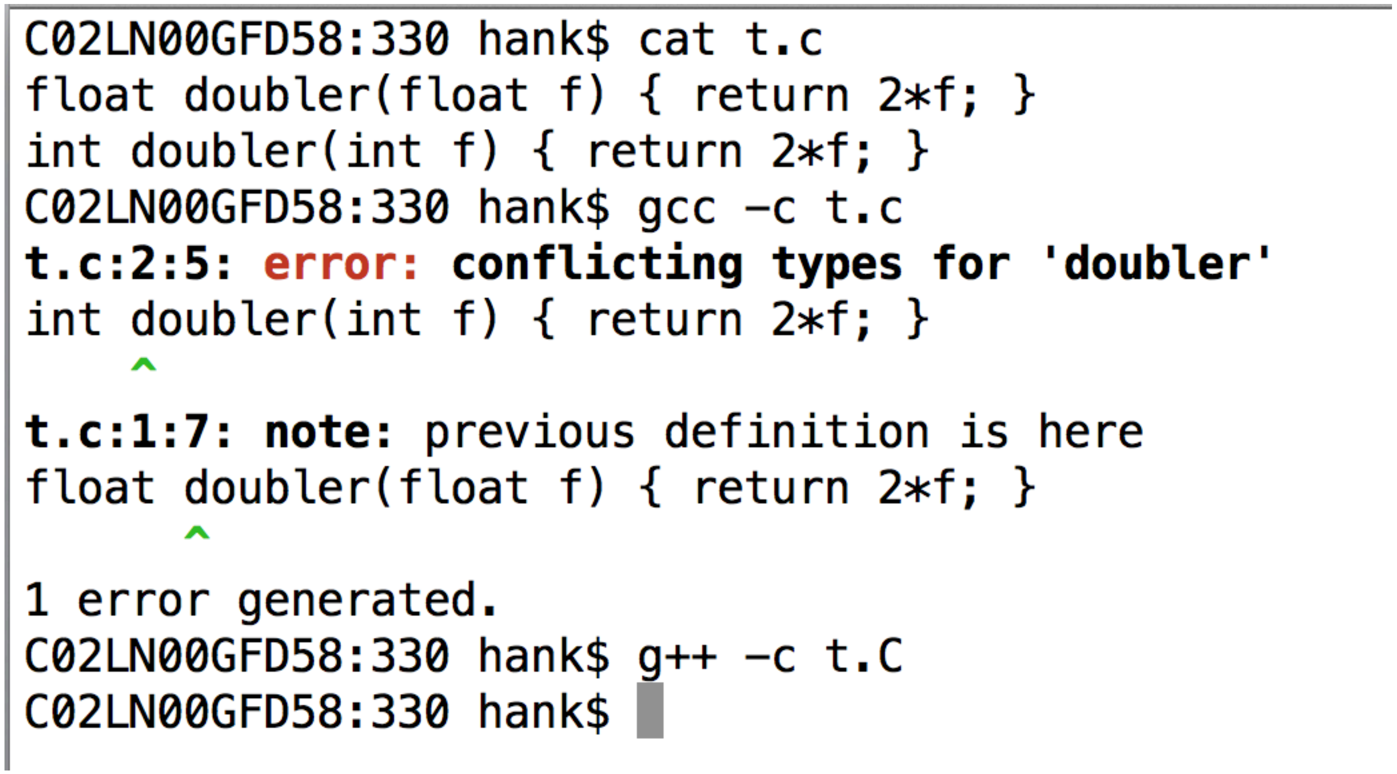


```
C02LN00GFD58:330 hank$ nm doubler.o
0000000000000048 s EH_frame0
0000000000000000 T _doubler
0000000000000060 S _doubler.eh
C02LN00GFD58:330 hank$ nm doubler
doubler.c                  doubler_example
doubler.o                  doubler_example.c
C02LN00GFD58:330 hank$ nm doubler_examp
0000000000000068 s EH_frame0
0000000000000032 s L_.str
                 U _doubler
0000000000000000 T _main
0000000000000080 S _main.eh
                 U _printf
```

Mangling

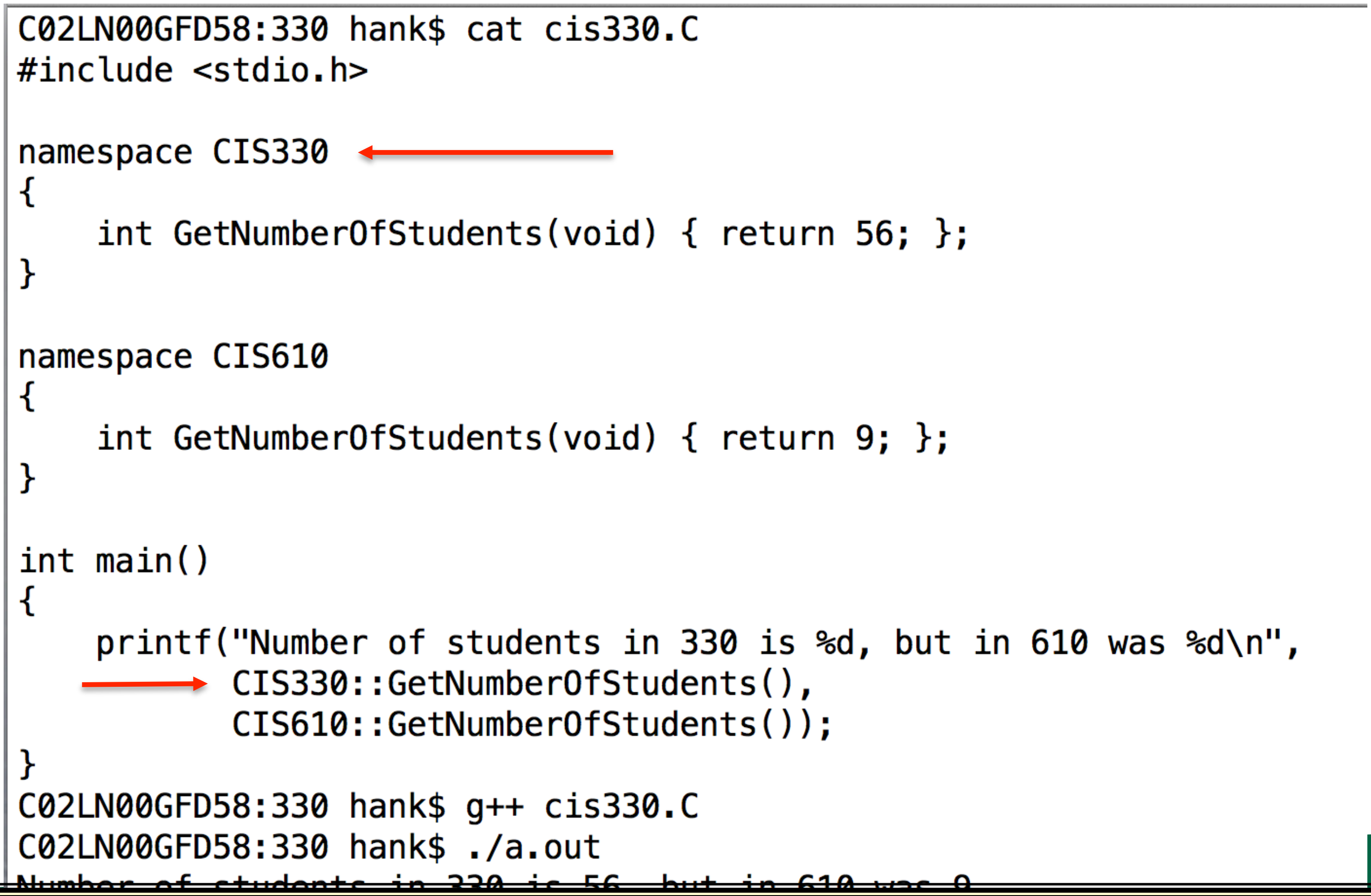
- Mangling refers to combing information about the return type and arguments and “mangling” it with function name.
  - Way of ensuring that you don’t mix up functions.
- Causes problems with compiler mismatches
  - C++ compilers haven’t standardized.
  - Can’t take library from icpc and combine it with g++.

C++ will let you overload functions with different types



```
C02LN00GFD58:330 hank$ cat t.c
float doubler(float f) { return 2*f; }
int doubler(int f) { return 2*f; }
C02LN00GFD58:330 hank$ gcc -c t.c
t.c:2:5: error: conflicting types for 'doubler'
int doubler(int f) { return 2*f; }
    ^

t.c:1:7: note: previous definition is here
float doubler(float f) { return 2*f; }
      ^

1 error generated.
C02LN00GFD58:330 hank$ g++ -c t.C
C02LN00GFD58:330 hank$
```

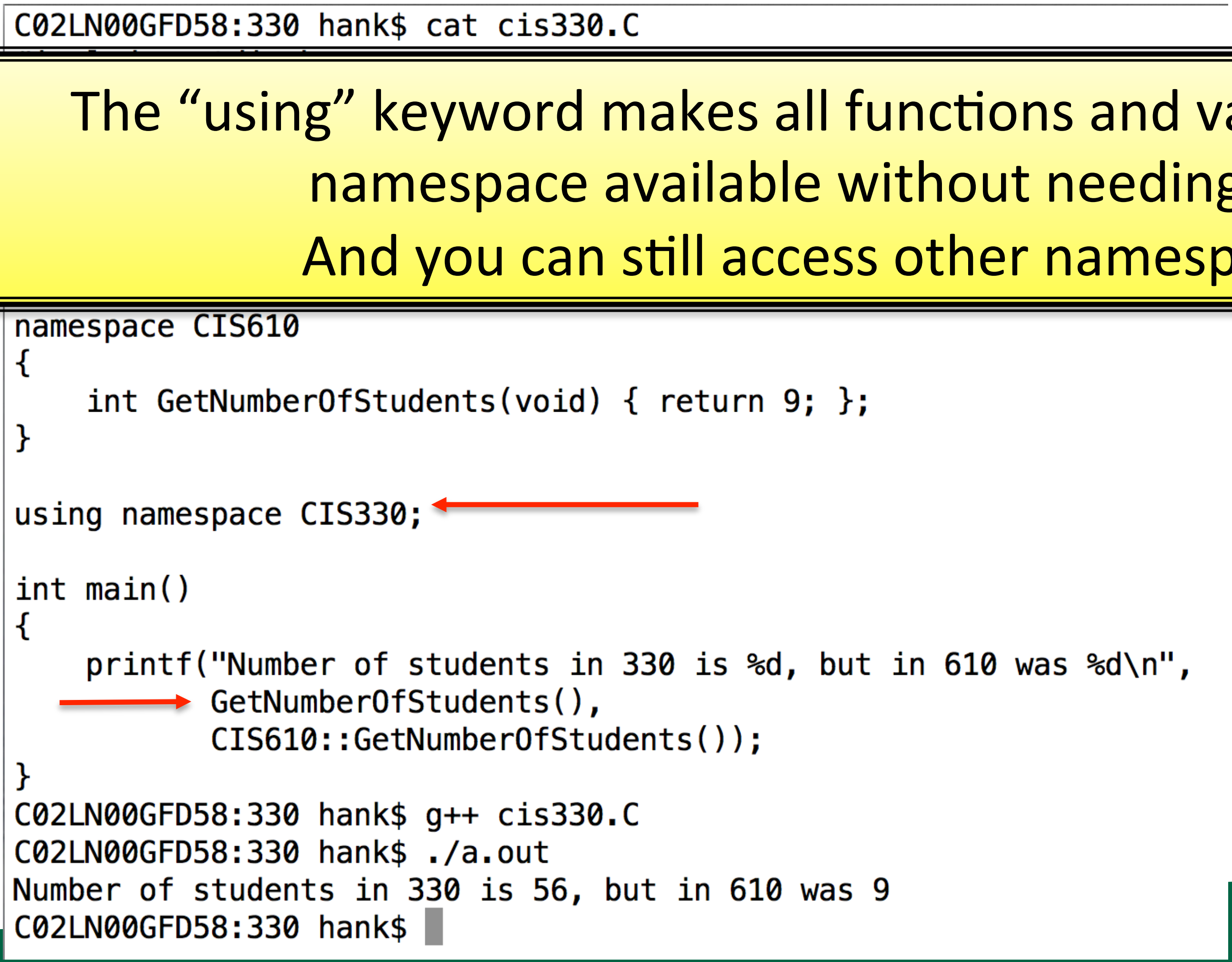
C++ also gives you access to mangling via “namespaces”



```
C02LN00GFD58:330 hank$ cat cis330.C
#include <stdio.h>

namespace CIS330
{
    int GetNumberOfStudents(void) { return 56; };
}

namespace CIS610
{
    int GetNumberOfStudents(void) { return 9; };
}

int main()
{
    printf("Number of students in 330 is %d, but in 610 was %d\n",
           CIS330::GetNumberOfStudents(),
           CIS610::GetNumberOfStudents());
}
C02LN00GFD58:330 hank$ g++ cis330.C
C02LN00GFD58:330 hank$ ./a.out
Number of students in 330 is 56, but in 610 was 9
```

Functions or variables within a namespace are accessed with “::”

C++ also gives you access to mangling via "namespaces"



```
C02LN00GFD58:330 hank$ cat cis330.C
```

The "using" keyword makes all functions and variables from a namespace available without needing "::". And you can still access other namespaces.

```
namespace CIS610
{
    int GetNumberOfStudents(void) { return 9; };
}

using namespace CIS330;

int main()
{
    printf("Number of students in 330 is %d, but in 610 was %d\n",
           GetNumberOfStudents(),
           CIS610::GetNumberOfStudents());
}
C02LN00GFD58:330 hank$ g++ cis330.C
C02LN00GFD58:330 hank$ ./a.out
Number of students in 330 is 56, but in 610 was 9
C02LN00GFD58:330 hank$ 
```

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)