Lecture 7:
Enums, Structs, and Unions
Reminder

• No class one week from today
Grade 4A

• (My apologies for all the problems)
Outline

• Review
• Project 2E
• Enums
• Structs
• Unions
• Project 2F
• Bonus Material
Outline

• Review
• Project 2E
• Enums
• Structs
• Unions
• Project 2F
• Bonus Material
ASCII Character Set

There have been various extensions to ASCII ...
now more than 128 characters
Many special characters are handled outside this convention

image source: granneman.com
signed vs unsigned chars

• signed char ("char"):  
  – valid values: -128 to 127  
  – size: 1 byte  
  – used to represent characters with ASCII  
    • values -128 to -1 are not valid  

• unsigned char:  
  – valid values: 0 to 255  
  – size: 1 byte  
  – used to represent data
character strings

• A character “string” is:
  – an array of type “char”
  – that is terminated by the NULL character

• Example:
  char str[12] = “hello world”;  
  – str[11] = ‘\0’ (the compiler did this automatically)

• The C library has multiple functions for handling strings
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"
Useful C library string functions

- strcpy: string copy
- strncpy: string copy, but just first N characters
- strlen: length of a string

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char str[12] = "hello world";
    char str2[6], str3[7];
    strcpy(str2, str+strlen("hello "));
    strncpy(str3, str, strlen("hello "));
    printf("%s,%s\n", str2, str3);
}
```

```
128-223-223-72-wireless:330 hank$ cat strcpy.c
128-223-223-72-wireless:330 hank$ gcc strcpy.c
128-223-223-72-wireless:330 hank$ ./a.out
world,hello
```
# More useful C library string functions

## Functions

### Copying:
- **`memcpy`**: Copy block of memory ([function](https://cplusplus.com))
- **`memmove`**: Move block of memory ([function](https://cplusplus.com))
- **`strcpy`**: Copy string ([function](https://cplusplus.com))
- **`strncpy`**: Copy characters from string ([function](https://cplusplus.com))

### Concatenation:
- **`strcat`**: Concatenate strings ([function](https://cplusplus.com))
- **`strncat`**: Append characters from string ([function](https://cplusplus.com))

### Comparison:
- **`memcmp`**: Compare two blocks of memory ([function](https://cplusplus.com))
- **`strcmp`**: Compare two strings ([function](https://cplusplus.com))
- **`strcoll`**: Compare two strings using locale ([function](https://cplusplus.com))
- **`strncpy`**: Compare characters of two strings ([function](https://cplusplus.com))
- **`strxfrm`**: Transform string using locale ([function](https://cplusplus.com))

### Searching:
- **`memchr`**: Locate character in block of memory ([function](https://cplusplus.com))
- **`strchr`**: Locate first occurrence of character in string ([function](https://cplusplus.com))
- **`strcspn`**: Get span until character in string ([function](https://cplusplus.com))
- **`strpbrk`**: Locate characters in string ([function](https://cplusplus.com))
- **`strrchr`**: Locate last occurrence of character in string ([function](https://cplusplus.com))
- **`strspn`**: Get span of character set in string ([function](https://cplusplus.com))
- **`strstr`**: Locate substring ([function](https://cplusplus.com))
- **`strtok`**: Split string into tokens ([function](https://cplusplus.com))

### Other:
- **`memset`**: Fill block of memory ([function](https://cplusplus.com))
- **`strerror`**: Get pointer to error message string ([function](https://cplusplus.com))
- **`strlen`**: Get string length ([function](https://cplusplus.com))

### Macros
- **NULL**: Null pointer ([macro](https://cplusplus.com))

### Types
- **size_t**: Unsigned integral type ([type](https://cplusplus.com))

**source: cplusplus.com**
if-then-else

int val = (X < 2 ? X : 2);

if (X<2)
{
    val = X;
}
else
{
    val = 2;
}
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 beginning of the file.

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

Close when you are done with "fclose"

`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.

Note: #include <stdio.h>
NAME
fread, fwrite — binary stream input/output

LIBRARY
Standard C Library (libc, -lc)

SYNOPSIS
#include <stdio.h>

size_t
fread(void *restrict ptr, size_t size, size_t nitems, FILE *restrict stream);

size_t
fwrite(const void *restrict ptr, size_t size, size_t nitems,
      FILE *restrict stream);

DESCRIPTION
The function fread() reads nitems objects, each size bytes long, from the stream pointed to by stream, storing them at the location given by ptr.

The function fwrite() writes nitems objects, each size bytes long, to the stream pointed to by stream, obtaining them from the location given by ptr.

RETURN VALUES
The functions fread() and fwrite() advance the file position indicator for the stream by the number of bytes read or written. They return the number of objects read or written. If an error occurs, or the end-of-file is reached, the return value is a short object count (or zero).
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
The `fseek()` function sets the file position indicator for the stream pointed to by `stream`. The new position, measured in bytes, is obtained by adding `offset` bytes to the position specified by `whence`. If `whence` is set to SEEK_SET, SEEK_CUR, or SEEK_END, the offset is relative to the start of the file, the current position indicator, or end-of-file, respectively. A successful call to the `fseek()` function clears the end-of-file indicator for the stream and undoes any effects of the `ungetc(3)` and `ungetwc(3)` functions on the same stream.
The `ftell()` function obtains the current value of the file position indicator for the stream pointed to by `stream`. 
We have everything we need to make a copy command…

- fopen
- fread
- fwrite
- fseek
- ftell

Can we do this together as a class?
argc & argv

• two arguments to every C program
• argc: how many command line arguments
• argv: an array containing each of the arguments
• “./a.out hank childs”
• → argc == 3
#include <stdio.h>
#include <stdlib.h>
#include <string.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;
}
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.
Unix shells allows you to manipulate standard streams.

- “>” redirect output of program to a file
- Example:
  - `ls > output`
  - `echo "this is a file" > output2`
  - `cat file1 file2 > file3`
Unix shells allows you to manipulate standard streams.

- “<” redirect file to input of program

  Example:
  - python < myscript.py

    - Note: python quits when it reads a special character called EOF (End of File)
    - You can type this character by typing Ctrl-D
    - This is why Python quits when you type Ctrl-D
      - (many other programs too)
Unix shells allows you to manipulate standard streams.

• “>>” concatenate output of program to end of existing file
  – (or create file if it doesn’t exist)

• Example:
  – echo “I am starting the file” > file1
  – echo “I am adding to the file” >> file1
  – cat file1
    I am starting the file
    I am adding to the file
What’s happening here?

ls is outputting its error messages to stderr

```
C02LN00GFD58:330 hank$ mkdir tmp
C02LN00GFD58:330 hank$ cd tmp
C02LN00GFD58:tmp hank$ touch f1
C02LN00GFD58:tmp hank$ ls f1 f2 > out
ls: f2: No such file or directory
C02LN00GFD58:tmp hank$ cat out
f1
```
Redirecting stderr in a shell

```
C02LN00GFD58:Documents hank$ cd ~/330
C02LN00GFD58:330 hank$ mkdir tmp
C02LN00GFD58:330 hank$ cd tmp
C02LN00GFD58:tmp hank$ touch f1
C02LN00GFD58:tmp hank$ ls f1 f2 > out
ls: f2: No such file or directory
C02LN00GFD58:tmp hank$ cat out
f1
C02LN00GFD58:tmp hank$ ls f1 f2 > out 2>out_error
C02LN00GFD58:tmp hank$ cat out_error
ls: f2: No such file or directory
```
Redirecting stderr to stdout

Convenient when you want both to go to the same stream
c functions: fork and pipe

• fork: duplicates current program into a separate instance
  – Two running programs!
  – Only differentiated by return value of fork (which is original and which is new)

• pipe: mechanism for connecting file descriptors between two forked programs

Through fork and pipe, you can connect two running programs. One writes to a file descriptor, and the other reads the output from its file descriptor.

Only used on special occasions. (And one of those occasions is with the shell.)
pipes in Unix shells

- represented with “|”
- output of one program becomes input to another program

```c
C02LN00GFD58:tmp hank$ cat printer.c
#include <stdio.h>
int main() { printf("Hello world\n"); }
C02LN00GFD58:tmp hank$ cat doubler.c
#include <stdio.h>
int main()
{
    int ch = getc(stdin);
    while (ch != EOF)
    {
        printf("%c%c", ch, ch);
        ch = getc(stdin);
    }
}
C02LN00GFD58:tmp hank$ gcc -o printer printer.c
C02LN00GFD58:tmp hank$ gcc -o doubler doubler.c
C02LN00GFD58:tmp hank$ ./printer | ./doubler
HHeelllllo  wwoorrlllddd
```
Very useful programs

- grep: keep lines that match pattern, discard lines that don’t match pattern

```bash
CO2LN00GFD58:Documents hank$ ls -l | grep ppt
-rw-r--r-- 1 hank staff 3278589 Apr  5 11:40 CIS330_Lec2.pptx
-rw-r--r-- 1 hank staff 2220104 Apr  8 20:57 CIS330_Lec3.pptx
-rw-r--r-- 1 hank staff 3899863 Jan 21 09:26 CIS610_Lec2.pptx
-rw-r--r-- 1 hank staff 4629257 Jan 30 10:24 CIS610_Lec3.pptx
-rw-r--r-- 1 hank staff 21382185 Mar 25 12:40 CIS_colloquium2013.pptx
-rw-r--r-- 1 hank staff 21382185 Jan  7 12:21 CIS_colloquium_2013.pptx
-rw-r--r-- 1 hank staff 2172179 Dec 20 15:24 ICS_results.pptx
-rw-r--r-- 1 hank staff 4841050 Nov 13 10:10 MBTI.pptx
-rw-r--r-- 1 hank staff 2031749 Apr  5 16:20 SC14_flow.pptx
-rw-r--r-- 1 hank staff 17972476 Mar 25 12:43 VMV_2013.pptx
-rw-r--r-- 1 hank staff 98149068 Apr  1 10:25 aachen.pptx
-rw-r--r-- 1 hank staff 9815146 Feb 24 07:00 childs_poster_SDAV_AHM_2014.pptx
-rw-r--r-- 1 hank staff 592243 Feb 26 04:09 childs_sdav_slides.pptx
-rw-r--r-- 1 hank staff 15765504 Feb 13 14:57 cig_exascale.ppt
-rw-r--r-- 1 hank staff 16699392 Jan  7 12:14 cis610_Lec1.ppt
-rw-r--r-- 1 hank staff 3159872 Jan  7 11:15 egpgv_cgf.pptx
-rw-r--r-- 1 hank staff 15767552 Mar 23 02:48 eu_regional_school.ppt
-rw-r--r-- 1 hank staff 35099136 Mar 25 00:42 eu_regional_school_part1.ppt
-rw-r--r-- 1 hank staff 10775552 Mar 25 04:49 eu_regional_school_part1B.ppt
-rw-r--r-- 1 hank staff 72966144 Mar 26 08:43 eu_regional_school_part2.ppt
-rw-r--r-- 1 hank staff 7571317 Mar 25 12:53 ilm_booth_talk.pptx
```
Very useful programs

• **sed**: replace pattern 1 with pattern 2
  – `sed s/pattern1/pattern2/g`
    • `s` means substitute
    • `g` means “global” ... every instance on the line

**sed** is also available in “vi”
: `%s/pattern1/pattern2/g` (% means all lines)
: `103,133s/p1/p2/g` (lines 103-133)
Outline

• Review
• Project 2E
• Enums
• Structs
• Unions
• Project 2F
• Bonus Material
CIS 330: Project #2E
Assigned: April 17th, 2015
Due April 23rd, 2015
(which means submitted by 6am on April 24th, 2015)
Worth 4% of your grade

Assignment: Write a program that reads the file “2E_binary_file”. This file contains a
two-dimensional array of integers, that is 10x10. You are to read in the 5x5 bottom
left corner of the array. That is, the values 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. Please write this as strings, one integer per line
(25 lines total). You should be able to “cat” the file afterwards and see the values.

Use Unix file streams for this project (i.e., fopen, fread, fseek, fprintf). Your program
will be checked for good programming practices. (Close your file streams, use
memory correctly, etc. I am not referring to style, variable initialization, etc.)

Also, add support for command line arguments (argc and argv).

Your program should run as:
./<prog_name> <input_name> <output_name>

(The input_name will be 2E_binary_file, unless you change it.)

Finally, note that I am handing you a binary file. I think we are all little endian, and
so it will be fine. But, if it is big endian, then we will have a problem. You can check
if it is little endian by printing the first two values of the file. They should be “0” and
“1”.

Please submit a tarball with (1) a Makefile (should be simple), (2) your source code,
and (3) the output ASCII file from running your program, with the name
“ASCII_output”.
Outline

• Review
• Project 2E
• Enums
• Structs
• Unions
• Project 2F
• Bonus Material
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

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

This enum contains 6 different student types

semi-colon!!!
enum example

```c
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

```
#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

```java
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 == PostBacc) 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

• Review
• Project 2E
• Enums
• Structs
• Unions
• Project 2F
• Bonus Material
Data types

- float
- double
- int
- char
- unsigned char

All of these are simple data types
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
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
- Declaring an instance
- "." accesses data members for a struct
Nested structs

```c
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)
typedef

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

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

typedef 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;
}
```

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

• Review
• Project 2E
• Enums
• Structs
• Unions
• Project 2F
• Bonus Material
Unions

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

```c
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

Why are unions useful?

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
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;
}
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.
Outline

• Review
• Project 2E
• Enums
• Structs
• Unions
• Project 2F
• Bonus Material
Notification: Project 2 going up in value

• Slotted at 10%
• Now 15%
  – 2A/2D: 3%
  – 2E: 4%
  – 2F: 4%
  – 2G: 4%
• Project 3 and 4 will be worth 5% less in some way
CIS 330: Project #2F
Assigned: April 22nd, 2015
Due April 29th, 2015
(which means submitted by 6am on April 30th, 2015)
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.

Again, your job is to define 3 structs and 9 functions. The comments below clarify
the format of the Rectangle, Circle, and Triangle, as well as the convention for
GetBoundingBox, and an example of accessing data members for pointers to structs.

== Rectangle ==

The rectangle has corners (minX, minY), (maxX, minY), (minX, maxY), (maxX, maxY).
Its area is (maxX-minX)*(maxY-minY).
Its bounding box is from minX to maxX in X, and minY to maxY in Y.

== Circle ==

The circle has an origin (x and y) and a radius.

Its area is 3.14159*radius*radius.
Its bounding box is from (x-radius) to (x+radius) in X, and (y-radius) to (y+radius)
in Y.

== Triangle ==

The triangle always has two points at the minimum Y-value. The third point's Y-
value is at the maximum Y-value, and its X-value is at the average of the X's of the
other two points. Saying it another way, the first two points form the "base", and the
third point is "height" above it.

Thus, the height of the triangle is (pt2X-p1X)*(maxY-minY)/2;
And the bounding box is from p1X to pt2X in X, and from minY to maxY in Y.

== GetBoundingBox ==

The GetBoundingBox calls take a double * as an argument. If a shape has its
minimum X at "a", its maximum X at "b", its minimum Y at "c", and its maximum Y at
"d", then it should do something like:

void GetCircleBoundingBox(Circle *, double *bbox)
{
    bbox[0] = a;
    bbox[1] = b;
    bbox[2] = c;
    bbox[3] = d;
}

== Working with pointers to structs ==

We reviewed the way to access struct data members in class, which was with the ".
" operator. We did not review the way to access struct data members when you have
a pointer to a struct. And the 9 function prototypes all use pointers to structs. It is
done with the ->.

So:
typedef struct
{
    int X;
} Y;
int main()
{
    Y y;
    Y *y2;
    y2 = &y;
    y.x = 0;
    y2->x = 1;
}
Outline

• Review
• Project 2E
• Enums
• Structs
• Unions
• Project 2F
• Bonus Material
Problem with C...

```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
00000000000000048  s  EH_frame0
00000000000000000  T  _doubler
00000000000000060  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
00000000000000068  s  EH_frame0
00000000000000032  s  L_.str
    U  _doubler
00000000000000000  T  _main
00000000000000080  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
00000000000000048 s EH_frame0
000000000000000000 T __Z7doublerf
00000000000000060 S __Z7doublerf.eh
C02LN00GFD58:330 hank$ nm doubler_example.o
00000000000000068 s EH_frame0
00000000000000032 s L_.str
U __Z7doubleri
000000000000000000 T _main
00000000000000080 S _main.eh
U _printf
C02LN00GFD58:330 hank$ nm doubler
C02LN00GFD58:330 hank$ nm doubler_example
C02LN00GFD58:330 hank$ nm doubler_example.o
00000000000000068 s EH_frame0
00000000000000032 s L_.str
U _doubler
000000000000000000 T _main
00000000000000080 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

```c
float doubler(float f) { return 2*f; }
int doubler(int f) { return 2*f; }
```

```bash
C02LN00GFD58:330 hank$ cat t.c
C02LN00GFD58:330 hank$ gcc -c t.c
C02LN00GFD58:330 hank$ g++ -c t.C
C02LN00GFD58:330 hank$
```

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

```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());
}
```

Functions or variables within a namespace are accessed with “::”
C++ also gives you access to mangling via “namespaces”

```c
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", 
           CIS610::GetNumberOfStudents(),
           C::GetNumberOfStudents());
}
```

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

NAME
    memcpy -- copy memory area

LIBRARY
    Standard C Library (libc, -lc)

SYNOPSIS
    #include <string.h>

    void *
    memcpy(void *restrict dst, const void *restrict src, size_t n);

DESCRIPTION
    The memcpy() function copies n bytes from memory area src to memory area dst. If dst and src overlap, behavior is undefined. Applications in which dst and src might overlap should use memmove(3) instead.

RETURN VALUES
    The memcpy() function returns the original value of dst.

I mostly use C++, and I still use memcpy all the time
sscanf

- like printf, but it parses from a string
  
  ```
  sscanf(str, "%s\n%d %d\n%d\n", magicNum, &width, &height, &maxval);
  ```
  
on:
  ```
  str="P6\n1000 1000\n255\n";
  ```
gives:
```text
magicNum = “P6”, width = 1000, 
height = 1000, maxval = 255
```