Info Session
- Wednesday, Oct. 30
- 6 - 7 PM in Lillis 262

Who we are
- A diverse group of hardworking, creative, and collaborative students who work on fun, supportive, tight-knit teams to solve challenging problems.

What we do
- Work on 10-week engagements in 6-person teams which include 4 Consultants, 1 Project Manager, and 1 Senior Manager. These high-performing consultants contribute ~10 hours/week to deliver quality final projects.

Application
- Oct. 14 - Nov. 1 @ 11:59 PM
- business.uoregon.edu/ocg
- Questions? Contact lcbocg@uoregon.edu
Let’s Grade 2D!
Dear Hank,

An e-mail has been sent to your students letting them know the Midway Student Experience Survey is now open and will close at 06:00 PM on Fri, Oct 25, 2019 PDT.

Students are aware that instructors may select to provide class time for face-to-face classes when feasible. For this reason please clarify with your students whether or not they should expect class time, or if they should complete the survey on their own.

You can view the feedback from your students beginning October 28th at noon.

Resources:
In-class protocol for Survey completion
Office of the Provost: Revising UO's Teaching Evaluations
Teaching Engagement Program: Student Feedback
Office of the Registrar: Student Experience Survey FAQ

Reminder regarding additional ways to increase response rates and quality feedback:

1. Make it an assignment (you don’t have to give points or extra credit or even keep track).
2. Tell your students that their feedback is valuable to you.
3. Provide students with examples of useful and actionable comments, in contrast to non-actionable comments.

How do students find the Student Experience Surveys?
Students login to DuckWeb and select Course Surveys on the Main Menu page (it is a link, not a tab) and then select a link that says "Open the Course Surveys site". After being redirected into the CollegeNET system, their list of courses will appear. Students can click on Evaluate in the Action column on the right side of the page to fill out their survey.

For questions, email the Office of the Provost at otp@uoregon.edu.

Thank you!
Office of the Provost
Structs, typedef, union
Simple 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)
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;
};
```

```c
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];
} 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;
    r.direction[2] = 0;
}
```
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
  – Note: enums discussed in lab, unions discussed next
So important: struct data member access is different with pointers

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;
    r.direction[2] = 0;
}

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

int main()
{
    Ray *r = malloc(sizeof(Ray));
    r->origin[0] = 0;
    r->origin[1] = 0;
    r->origin[2] = 0;
    r->direction[0] = 1;
    r->direction[1] = 1;
    r->direction[2] = 1;
}
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
#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);
}

As u.x = 0.000000, as u.y = 3
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;
}
File I/O
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”

Note: #include <stdio.h>
Reading / Writing

FREAD(3) BSD Library Functions Manual FREAD(3)

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

```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);
}
```

```bash
C02LN00GFD58:330 hank$ gcc rw.c
C02LN00GFD58:330 hank$ ./a.out
C02LN00GFD58:330 hank$ cat 330
hello world: file edition
```
File Position Indicator

• File position indicator: the current location in the file
• If I read one byte, the one byte you get is where the file position indicator is pointing.
  – And the file position indicator updates to point at the next byte
  – But it can be changed...
```c
int fseek(FILE *stream, long offset, int whence);
```

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.
ftell

```c
long ftell(FILE *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?
#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;
}
Return values in shells

$? is the return value of the last executed command

C02LN00GFD58:330 hank$ ./a.out copy.c copy2.c
Copying 697 bytes from copy.c to copy2.c
C02LN00GFD58:330 hank$ echo $?
0
C02LN00GFD58:330 hank$ ./a.out
Usage: ./a.out <file1> <file2>
C02LN00GFD58:330 hank$ echo $?
1
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)
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
2E

CIS 212: Project #2E
Assigned: October 24, 2019
Due: October 30, 2019
(which means submitted by 6am on October 31, 2019)
Worth 9% of your grade

Assignment:
1) Write a C program that reads from a file and provides a summary of how many words occur and how many times a specified set of words occurs.
2) Important: words are separated by spaces, commas, periods, or newlines.
3) Important: you may use the functions strlen and strcmp, but no other functions from the C string library. (Do not use strtok.)

Your program should be run as follows:
./project2E filename word1 word2 word3 ....

For example, if I run:
./project2E file.txt hank childs

And file.txt contains
hank Hank hankchilds childs hank,childs

Then the output should be:
The word “hank” occurs 2 times.
The word “childs” occurs 3 times.
Streams in Unix
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
Redirecting stderr in a shell

```bash
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_error
C02LN00GFD58:tmp hank$ cat out_error
ls: f2: No such file or directory
```
Redirecting stderr to stdout

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_error
C02LN00GFD58:tmp hank$ cat out_error
ls: f2: No such file or directory
C02LN00GFD58:tmp hank$ ls f1 f2 > out &1
C02LN00GFD58:tmp hank$ cat out
ls: f2: No such file or directory
f1

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
Very useful programs

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

C02LN00GFD58: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_sdad_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 09:42 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)
Wildcards

• ‘*’ is a wildcard with unix shells

<table>
<thead>
<tr>
<th>Abe</th>
<th>Chavarria</th>
<th>Hebb</th>
<th>Macy</th>
<th>Smith</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alajaji</td>
<td>Chen</td>
<td>Jia</td>
<td>Maguire</td>
<td>Steelhammer</td>
</tr>
<tr>
<td>Alamoudi</td>
<td>Clark</td>
<td>Kine</td>
<td>Michlanski</td>
<td>Szczepanski</td>
</tr>
<tr>
<td>Anastas</td>
<td>Collier</td>
<td>Lee</td>
<td>Moreno</td>
<td>Totten</td>
</tr>
<tr>
<td>Andrade</td>
<td>Costello</td>
<td>Legge</td>
<td>Olson</td>
<td>Vega-Fujioka</td>
</tr>
<tr>
<td>Ballarche</td>
<td>Donnelly</td>
<td>Li</td>
<td>Owen</td>
<td>Wang</td>
</tr>
<tr>
<td>Brennan</td>
<td>Etzel</td>
<td>Liu</td>
<td>Pogrebinsky</td>
<td>Whiteley</td>
</tr>
<tr>
<td>Brockway</td>
<td>Friedrich</td>
<td>Lin</td>
<td>Qin</td>
<td>Woodruff</td>
</tr>
<tr>
<td>Brogan</td>
<td>Garvin</td>
<td>Lopes</td>
<td>Rhodes</td>
<td>Xu</td>
</tr>
<tr>
<td>Brooks</td>
<td>Gonzalez</td>
<td>Luo</td>
<td>Roberts</td>
<td>Yaconelli</td>
</tr>
<tr>
<td>Bruce</td>
<td>Guo</td>
<td>Lynch</td>
<td>Rodriguez</td>
<td>Young</td>
</tr>
<tr>
<td>Carlton</td>
<td>Hampton</td>
<td>Lyon</td>
<td>Roush</td>
<td>Zhang</td>
</tr>
<tr>
<td>Chalmers</td>
<td>Harris</td>
<td>Machado</td>
<td>Rozenboim</td>
<td>de</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>fawcett:tmp child$ ls C*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carlton</td>
</tr>
<tr>
<td>Rodriguez</td>
</tr>
</tbody>
</table>
Other useful shell things

• ‘tab’: auto-complete
• esc=: show options for auto-complete
• Ctrl-A: go to beginning of line
• Ctrl-E: go to end of line
• Ctrl-R: search through history for command