Lecture 9: Pointers to Functions (and other things)
Midterm -- Tuesday Nov 5\textsuperscript{th} @ 10am

• No practice exam
• There will be coding
  – I will not deduct too much for things like missing semi-colons
  – But I will grade quite lowly if it looks like you don’t know how to code in C
• I also have emphasized memory and want you to know about memory
• There may also be:
  – True / false
  – Unix
The inclusiveness of this course:

- Is beneficial to learning: 83% (10)
- Is neutral: 17% (2)
- Needs improvement: 0% (0)

The level of support from the instructor:

- Is beneficial to learning: 83% (10)
- Is neutral: 17% (2)
- Needs improvement: 0% (0)
Midterm Review (2/4)

The quality of the course materials:

- Is beneficial to learning: 75% (9)
- Is neutral: 25% (3)
- Needs improvement: 0% (0)

The clarity of assignment instructions and grading:

- Is beneficial to learning: 58% (7)
- Is neutral: 42% (5)
- Needs improvement: 0% (0)
How many hours per week did you spend on this course (not including any face-to-face class time)?

- More than 10 hours: 8% (1)
- 8-10 hours each week: 42% (5)
- 6-8 hours each week: 33% (4)
- 4-6 hours each week: 17% (2)
- 2-4 hours each week: 0% (0)
- 1-2 hours each week: 0% (0)
- 0-1 hour each week: 0% (0)
Midterm Review (4/4)

• More comments?

• Also:
  – Email
  – Anonymous notes under my door
#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;
}
Let’s do a video on string manipulation

• Suggestions for what to implement?

• (are there other videos you would like?)
Feedback on 2C

• Also includes feedback on some 2C from last year (which was a different 2C)
Don’t Use ASCII Codes

Indent

```cpp
if (47 > str[i] || str[i] > 58 )
{IssueError();}
```

• Instead:

```cpp
if (str[i] < ‘0’ || str[i] > ‘9’)
IssueError()
```
The first part of the for loop is for initializing
Also: bitwise xor?

```c
for (i; str[i] != '.' ^ str[i] == NULL; i++){
    beforeDecimal += 1;
}
```

```c
for (start; ((str[start]) != '\0'); start++) {
```
Indent!

```c
if (error == 0 || points > 1)
    IssueError();
while (str[i] != '\0') {

double StringToDouble(char *str)
{
    int strIt;
    int TorF = 0;
    int decCnt = 0;
    for (strIt = 0; strIt != strlen(str); strIt++) {
        // Another example: indentation of one character.
        ```
For loop that slowed me down

```
for (int decimal = 0; *str; str++)
{

}
```

This was from an _exceptional_ submission.
Painful and error prone

```c
if(str[i]=='0' || str[i]=='1' || str[i]=='2' || str[i]=='3' || str[i]=='4' || str[i]=='5' || str[i]=='6' || str[i]=='7' || str[i]=='8' || str[i]=='9')
{
    // code
}
```
sizeof != strlen

```c
int num1 = 0;
int num2 = 0;

for (int i = 0; i < (sizeof(number1)-1); i++){
    if (number1[i] == ' '){
    }
```
Don’t do: int first[] = { x }; 
Instead do: int first[1] = { x }; 
(Also: didn’t need an array at all for this project)

int first[] = {x}; /**<use a new array to storage the first int IMPORTANT*/
Accidental global variable?

char zero = '0';

int convertStoI(char *num_str) {
    // Function body here
}
Don’t use “char str[]” as an argument (or at all)
Instead, use “char *str”
“while (1)” is the expected way to do this in C

```c
while (1 == 1) {
    // do something
}
```
if (op == ‘+’) is more readable than if (op == 43)

```cpp
int result;
if (op == 43){
    result = int_1 + int_2;
}
```
if (arg1[i] == '\0')
    NULL is an 8 byte pointer
    '\0' is a 1 byte character
    (the compiler happens to do a good conversion here to make this work)

    if (arg1[i] == NULL)
        {break;}
    else {do}
C programmers expect the “i++” in the third part of the definition. If you pull it out and put it in the body of the loop, it is innovating a new, unfamiliar pattern.
Talk about i++ vs ++i
else if -> else

```c
if (tmpchar != '\0')
{
    ...;
    ...
}
else if (tmpchar == '\0')
{
    break;
}
```
leftnum *= 10;

leftnum = leftnum * 10;
Indent your code!!

```java
if(number1[i] == '\0')
    break;
else
{
```
These are strings, not characters

```c
char *null = "\0";
char *plus = "+";
char *minus = "-";
char *number1 = argv[1];
```
assignment equal vs comparison equal

if (some_arg[i] = '\0')
{
    .

This is some very tricky code. (I don’t like it ... makes me have to think.)

```c
int len = 0;
do {
    len++;
}while(number1[len]);
```
New Material
Return values in shells

```sh
C02LN00GF D58:330 hank$ ./a.out copy.c copy2.c
Copying 697 bytes from copy.c to copy2.c
C02LN00GF D58:330 hank$ echo $? 0
C02LN00GF D58:330 hank$ ./a.out copy.c
Usage: ./a.out <file1> <file2>
C02LN00GF D58:330 hank$ echo $? 1
```

`$?` is the return value of the last executed command
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”);
  \rightarrow same as printf
• fprintf(stderr, “helloworld\n”);
  \rightarrow prints to “standard error”
• fprintf(f_out, “helloworld\n”);
  \rightarrow 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
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 strncmp, 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  
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

```bash
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

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
C02LN00GFD58:tmp hank$ ls f1 f2 > out 2>&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

```c
#include <stdio.h>
int main() { printf("Hello world\n"); }
```
Very useful programs

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

```bash
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_llec2.pptx
-rw-r--r-- 1 hank staff 4629257 Jan 30 10:24 CIS610_llec3.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_sdac_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_llec1.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 35099306 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 jlm_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

```
fawcett:tmp child$ ls
Abe Chavarria Hebb Macy Smith
Alajaji Chen Jia Maguire Steelhammer
Alamoudi Clark Kine Michlanski Szczepanski
Anastas Collier Lee Moreno Totten
Andrade Costello Legge Olson Vega-Fujioka
Ballarche Donnelly Li Owen Wang
Brennan Etzel Lin Pogrebinsky Whiteley
Brockway Friedrich Liu Qin Woodruff
Brogan Garvin Lopes Rhodes Xu
Brooks Gonzales Luo Roberts Yaconelli
Bruce Guo Lynch Rodriguez Young
Carlton Hampton Lyon Roush Zhang
Chalmers Harris Machado Rozenboim de
```

‘?’ is a wildcard that matches exactly one character
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
Call by value / call by reference

• Refers to how parameters are passed to a function.
  – Call by value: send the value of the variable as a function parameter
    • Side effects in that function don’t affect the variable in the calling function
  – Call by reference: send a reference (pointer) as a function parameter
    • Side effects in that function affect the variable in the calling function
C does not care that foo has an argument variable called “x” and main has an automatic variable called “x”. They have NO relation.
Call by reference

```c
#include <stdio.h>

void foo(int *x)
{
    *x = *x+1;
}

int main()
{
    int x = 2;
    foo(&x);
    printf("X is %d\n", x);
}
```

```bash
hank$ cat cbr.c
#include <stdio.h>

void foo(int *x)
{
    *x = *x+1;
}

int main()
{
    int x = 2;
    foo(&x);
    printf("X is %d\n", x);
}
```

```bash
hank$ gcc cbr.c
hank$ ./a.out
X is 3
```
Preprocessor

• Preprocessor:
  – takes an input program
  – produces another program (which is then compiled)

• C has a separate language for preprocessing
  – Different syntax than C
  – Uses macros (“#”)

macro (“macroinstruction”): rule for replacing input characters with output characters
#include

• compiler can only compile one file at a time
• takes another file and includes it in the current file
• the file is a “header” file
  – it contains function prototypes
  – a function prototype declares a function exists, but not how it is implemented
Demonstrate `#include`

```c
#include <stdio.h>
int main()
{
    printf("Hello world\n");
}

% gcc -E printf.c
```
Function Prototype

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

main.c:4:12: warning: implicit declaration of function 'doubler' is invalid in C99 [-Wimplicit-function-declaration]
  return doubler(2); ^
1 warning generated.
```
Preprocessor Phases

- Resolve #includes
- Conditional compilation (#ifdef)
- Macro replacement
- Special macros
#define compilation

```c
#define RV 2

int main()
{
    return RV;
}
```

This is an example of macro replacement.
#define via gcc command-line option

```c
int main()
{
    return RV;
}
```

```
C02LN00GFD58:330 hank$ cat defines.c
int main()
{
    return RV;
}
C02LN00GFD58:330 hank$ gcc -DRV=4 defines.c
C02LN00GFD58:330 hank$ ./a.out
C02LN00GFD58:330 hank$ echo $? 4
```
Conflicting –D and #define

C02LN00GFD58:330 hank$ cat defines.c
#define RV 2
int main()
{
    return RV;
}
C02LN00GFD58:330 hank$ gcc -DRV=4 defines.c
defines.c:1:9: warning: 'RV' macro redefined
#define RV 2
<command line>:1:9: note: previous definition is here
#define RV 4

1 warning generated.
C02LN00GFD58:330 hank$ ./a.out
C02LN00GFD58:330 hank$ echo $? 
2
Conditional compilation

```c
C02LN00GFD58:330 hank$ cat conditional.c
#define USE_OPTION 1

int main()
{
    DoMainCode();
    #ifdef USE_OPTION
        UseOption();
    #endif
    DoCleanupCode();
}
```
Conditional compilation controlled via compiler flags

```c
#include <stdio.h>

int main()
{
    #ifdef DO_PRINTF
        printf("I am doing PRINTF!!\n");
    #endif
}
```

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

int main()
{
    #ifdef DO_PRINTF
        printf("I am doing PRINTF!!\n");
    #endif
}
```

```
C02LN00GFD58:330 hank$ gcc conditional_printf.c
C02LN00GFD58:330 hank$ ./a.out
C02LN00GFD58:330 hank$ gcc -DDO_PRINTF conditional_printf.c
C02LN00GFD58:330 hank$ ./a.out
I am doing PRINTF!!
```
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

```
#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

```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 *);
    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]);
}
```

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.
Function Pointers vs Conditionals

What are the pros and cons of each approach?

```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));
}
```

```c
#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);
}
```
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);
Callback example

```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);
}
```