Lecture 10:
Potpourri:
Enum / struct / union
Advanced Unix
#include
function pointers
Questions on 2F?
Grading

• Got 2C done! 😊
• Viet will be doing more grading soon
  – And code reviews
2C feedback
sizeof != strlen

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

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

/* NOTE:  
 * - you may not use atoi
 * - you may not use *any* built-in functions -- strlen, strcpy, etc.
 * - you may assume all integers have <10 digits
 */

char zero = '0';

int convertStoI(char *num_str) {
    // Implementation

Don’t use “char str[]” as an argument (or at all)  
Instead, use “char *str”

```c
int str_to_int(char str[])
{
    return ...;
}
```
“while (1)” is the expected way to do this in C

```c
while (1 == 1)
{
    if (Number[1] == 1)
```

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)

```c
if (arg1[i] == NULL)
    {break;}
```
The value you return from a function does not need to be malloc’d.

```c
int *char_to_int(char num[]){
    int *p = malloc(sizeof(long int));
```
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.

```c
for (int i = 0; i<10;){
    // Other code here
    i++;
    // More code
}
```
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];
```
result = numero1 - numero2; 2;
if(*operation == '+')
{
    ...
}
if(*operation == '-')
{
    ...
}
else
{
    exit(EXIT_FAILURE);
}
if (operation[0] == ‘+’)  

Code below checks to see if the variable operation is pointing to the same location as the global string variable that represents “+” (It doesn’t)
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]);
```
OK, what videos am I supposed to do?

• my_strlen?
• Unix cp utility?
• one more?
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
Call by Value

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
hank$ gcc cbr.c
hank$ ./a.out
X is 3
```
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!!!
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
So important: struct data member access is different with pointers

```c
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;
}
```

```c
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;
}
```

Pointers: use “->”
Instances (i.e., not pointers): use “.”
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 union has 4 bytes
Unions

Why are unions useful?

```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;
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;
}
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 file
• 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

```
C02LN00GFD58:212 hank$ cat main.c
int doubler(int);
int main()
{
    return doubler(2);
}
int doubler(int X) { return 2*X; };
C02LN00GFD58:212 hank$ gcc main.c
C02LN00GFD58:212 hank$ vi main.c
C02LN00GFD58:212 hank$ cat main.c
/* int doubler(int); */
int main()
{
    return doubler(2);
}
int doubler(int X) { return 2*X; };
C02LN00GFD58:212 hank$ gcc main.c
main.c:4:12: warning: implicit declaration of function 'doubler' is invalid in C99 [-Wimplicit-function-declaration]
    return doubler(2);
    ^
1 warning generated.
C02LN00GFD58:212 hank$
```
Preprocessor Phases

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

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

```c
#include <stdio.h>

int doubler(int x) { return 2*x; }
int tripler(int x) { return 3*x; }
int main()
{
    int (*multiplier)(int);
    multiplier = doubler;
    printf("Multiplier of 3 = %d\n", multiplier(3));
    multiplier = tripler;
    printf("Multiplier of 3 = %d\n", multiplier(3));
}
```

128-223-223-72-wireless:cli hank$ gcc function_ptr.c
128-223-223-72-wireless:cli hank$ ./a.out
Multiplier of 3 = 6
Multiplier of 3 = 9
Function Pointer Example #2

```c
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 *);
    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
```

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

```c
#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);
}
```

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

Callback example

```c
void RegisterErrorHandler(void (*eh)(char *));
double mylogarithm(double 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);
}
esuccesful completion
128-223-223-72-wireless:callback hank$ cat program.c
128-223-223-72-wireless:callback hank$ make
128-223-223-72-wireless:callback hank$ ./program
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 !!
Return values in shells

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 copy.c
Usage: ./a.out <file1> <file2>
C02LN00GFD58:330 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)
`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.
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 allow 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?

```
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
```

ls is outputting its error messages to stderr
Redirecting stderr in a shell

```
C02L000GFD58:Documents hank$ cd ~/330
C02L000GFD58:330 hank$ mkdir tmp
C02L000GFD58:330 hank$ cd tmp
C02L000GFD58:tmp hank$ touch f1
C02L000GFD58:tmp hank$ ls f1 f2 > out
ls: f2: No such file or directory
C02L000GFD58:tmp hank$ cat out
f1
C02L000GFD58:tmp hank$ ls f1 f2 > out 2>out_error
C02L000GFD58: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.

```
 mkdir tmp
 cd tmp
 touch f1
 ls f1 f2 > out
 ls: f2: No such file or directory
 cat out f1
 ls f1 f2 > out 2>out_error
 cat out_error
 ls: f2: No such file or directory
 ls f1 f2 > out 2>&1
 cat out
 ls: f2: No such file or directory
 f1
```
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() {
    int ch = getc(stdin);
    while (ch != EOF)
    {
        printf("%c%c", ch, ch);
        ch = getc(stdin);
    }
}
```

```bash
$ cat printer.c
$ gcc -o printer printer.c
$ cat doubler.c
$ gcc -o doubler doubler.c
$ ./printer | ./doubler
```
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_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  4 07:00 childs_poster_SDAV_AHM_2014.pptx
-rw-r--r--@ 1 hank staff  592243 Feb 26 04:09 childs_sdv_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

```
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
fawcett:tmp child$ ls C*
fawcett:tmp child$ ls *z
```

‘?’ 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