Lecture 9: Potpourri:
Call by reference vs call by value
Enum / struct / union
Advanced Unix
Grading

• 1A done, 2D uploaded
• Aiming for 2C tonight
• Viet will be doing more grading soon
  – And code reviews
Great question from 1A (Liyang Wang)

• I want know why the C language has not been replaced by other new programming languages, and where do we need to use/choose C language in future? What are C's advantages?
Why Has C Stood The Test Of Time?

• A nice balance:
  – Assembly gives performance, but:
    • architecture-specific
    • makes you manage many details (registers, memory placement, etc)
  – Higher-level languages take care of details, but no performance
• C does it all! ... takes care of details + good performance
• And: C doesn’t change. This is important.
Why doesn’t my a.out work on other machines?

• Answers:
  – Machine code is different
    • Mac vs x86
  – Compilers / libraries are different
    • Windows vs Linux
Internships

• Great experience!
• Internship seminar
• MUST BE ABLE TO DEBUG
vi swap files

E325: ATTENTION

Found a swap file by the name "~/.cache/vim/swap/%home%me%Desktop%proj2e.c.swp"
  owned by: me   dated: Mon Oct 22 19:55:19 2018
  file name: ~me/Desktop/proj2e.c
  modified: YES
  user name: me   host name: archsept2018
  process ID: 11608

While opening file "proj2e.c"
  dated: Tue Oct 23 01:32:59 2018
  NEWER than swap file!

(1) Another program may be editing the same file. If this is the case,
    be careful not to end up with two different instances of the same
    file when making changes. Quit, or continue with caution.
(2) An edit session for this file crashed.
    If this is the case, use ":recover" or "vim -r proj2e.c"
    to recover the changes (see ":help recovery").
    If you did this already, delete the swap file "~/.cache/vim/swap/%home%me%Desktop%proj2e.c.swp"
    to avoid this message.

Swap file "~/.cache/vim/swap/%home%me%Desktop%proj2e.c.swp" already exists!
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);
}
```

```
$ 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
$ gcc cbr.c
$ ./a.out
X is 3
```
Example: my_strlen
Let’s look at a binary file
2E: Extend 2C For Error Checking

Assignment:
1) Write a C program parses three strings.
   a. String #1: an integer. You need to construct this integer by reading
      one character at a time.
   b. String #2: an operation. These will all be single character strings. The
      valid operations are + or -.
   c. String #3: an integer. (Same as String #1)
2) After parsing the 3 strings, you should perform the desired operation (+ or -).
3) Print the output of the operation.

Q: HOW IS THIS DIFFERENT THAN 2C?
A: a lot of inputs will be wrong. You need to deal with bad inputs. You can see the
bad inputs I throw out at you in check_2e.

Q: WHAT SHOULD I DO IF I GET BAD INPUTS?
A: you should output the string corresponding to the bad input. Note that the
correct strings can be found in check_2e.

This project will be graded by:
1) Running the Unix script “check_2e”. It is available on the course website.

If the diff program shows any difference, you will get less than half credit.

What should you upload?: Just a single file, which is your C source code.
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
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

```c
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

```c
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.
typedef

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

```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
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;
}
```
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
  – Note: enums discussed in lab, unions discussed next
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

C keyword "struct" – means struct definition is coming

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

This struct contains 6 doubles, meaning it is 48 bytes

```c
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;
}
```

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;
```

This union has 4 bytes

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

Why are unions useful?

```c
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;
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
{
    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;
}

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

  \`\`a+\`\`  Open for reading and writing. The file is created if it does not exist, otherwise it is truncated. 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

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
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...
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?
#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;
}
Return values in shells

```bash
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)

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
CIS 212: Project #2F
Assigned: October 18th, 2018
Due October 25th, 2018
(which means submitted by 6am on October 26th, 2018)
Worth 4% of your grade

Assignment: Write a program that reads the file “2F_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. Your program will be checked for good programming practices. (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:
./proj2F <input_name> <output_name>

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

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

```
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

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
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"); }
```

```c
#include <stdio.h>
int main()
{
    int ch = getc(stdin);
    while (ch != EOF)
    {
        printf("%c%c", ch, ch);
        ch = getc(stdin);
    }
}
```

```bash
c02ln00gfd58:tmp hank$ gcc -o printer printer.c
c02ln00gfd58:tmp hank$ gcc -o doubler doubler.c
c02ln00gfd58:tmp hank$ ./printer | ./doubler
HHellellllo  wwoorrllldd
c02ln00gfd58:tmp hank$ 
```
Very useful programs

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

```sh
fawcett:~ 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 Lin        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 Whiteley
```

• ‘?’ is a wildcard that matches exactly one character

```sh
fawcett:~ child$ ls C*
Carlton  Chavarria  Clark
Chalmers  Chen      Collier
```

```sh
fawcett:~ child$ ls *z
Rodriguez
```

```sh
fawcett:~ child$ ls *ee*
Lee       Steelhammer
```

```sh
fawcett:~ child$ ls *e*e*
Lee       Legge       Steelhammer  Whiteley
```
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