ADD, COMMIT, PUSH, . . .

ADD, COMMIT, PUSH, . . .

- I started working on huff on Saturday morning

```
commit 1040c90504ee0d16fe4359ce04d50c2187af19ce
Author: Kerry Veenstra <veenstra@ucsc.edu>
Date:   Sat May 27 03:46:44 2023 -0700

    snapshot
```

ADD, COMMIT, PUSH, . . .

- Additional snapshots were taken
  - Sat May 27 10:15:26 2023 -0700
  - Sat May 27 13:46:05 2023 -0700
  - Sat May 27 15:07:15 2023 -0700
  - Sat May 27 17:56:05 2023 -0700

ADD, COMMIT, PUSH, . . .

- Resulting in

```
commit b4edf159276e3ae10cebc4c2694214fe5c0cfd03
Author: Kerry Veenstra <veenstra@ucsc.edu>
Date:   Sat May 27 19:23:48 2023 -0700

    first working version
```

ADD, COMMIT, PUSH, . . .

- Sat May 27 22:20:48 2023 -0700
  - start work on dehuff, add leaf count to .huff file format
- Sun May 28 09:01:46 2023 -0700
  - added num_leaves to .huff file. git format
- Sun May 28 09:07:21 2023 -0700
  - add bitreader and bitwriter files
- Sun May 28 09:37:28 2023 -0700
  - rename parameter

ADD, COMMIT, PUSH, . . .

- Sun May 28 10:26:38 2023 -0700
  - added bwtest.c
- Sun May 28 10:49:27 2023 -0700
  - make and use io.a
- Sun May 28 13:11:51 2023 -0700
  - error for files with less then 2 unique bytes
- Sun May 28 19:02:24 2023 -0700
  - working dehuff

ADD, COMMIT, PUSH, . . .

- Sun May 28 19:39:15 2023 -0700
  - added Long's hack
- Sun May 28 19:57:06 2023 -0700
  - add tie-breaker to priority queue, reformat node weights
- Mon May 29 11:13:54 2023 -0700
  - make format
- Mon May 29 11:21:46 2023 -0700
  - .h files: edit header comments, add #ifndef

ADD, COMMIT, PUSH, . . .

- Mon May 29 12:36:44 2023 -0700
  - correct comment
- Mon May 29 15:10:39 2023 -0700
  - added tests
- Mon May 29 15:13:55 2023 -0700
  - added stack stuff for dehuff
- Mon May 29 16:11:26 2023 -0700
  - updated tests

ADD, COMMIT, PUSH, . . .

- Mon May 29 20:27:45 2023 -0700
  - added toresources script
- Mon May 29 22:13:55 2023 -0700
  - added runtests.sh to toresources
- Tue May 30 06:25:11 2023 -0700
  - use **`uname -m`** to identify current machine architecture
- Tue May 30 06:27:45 2023 -0700
  - restore .a

ADD, COMMIT, PUSH, . . .

- Tue May 30 08:37:10 2023 -0700
  - better verification of .huff file format, -v reports everything
- Tue May 30 19:15:58 2023 -0700
  - Makefile no longer builds debug by default. More .huff tests. enqueue() returns void

- Total duration: Saturday morning through Tuesday night
  - Wrote Assignment PDF
  - Wrote **`bwtest`**, **`nodetest`**, and **`pqtest`**
  - Wrote **`huff`**
  - Wrote **`dehuff`**

ADVANCED C

1. The `?:` operator
2. Preprocessor Tricks
3. `*` in `printf()`
4. `#include <time.h>`
5. Time Travel (rewinding a file)
6. `is_____()` macros
7. `setjmp()` & `longjmp()`

THE ?: OPERATOR

- The what?
- This is an if-then-else that fits into an expression

```c
const char *f(void) {
    char *message;
    bool okay;

    /*
    * Do something, and then report whether it's okay.
    * ...
    */

    if (okay) {
        message = "Everything's fine.";
    } else {
        message = "Uh, oh.";
    }

    return message;
}
```

```c
const char *f(void) {
    bool okay;

    /*
    * Do something, and then report whether it's okay.
    * ...
    */

    return okay ? "Everything's fine." : "Uh, oh.";
}
```

```c
const char *f(void) {
    bool okay;

    /*
    * Do something, and then report whether it's okay.
    * ...
    */

    return okay ? "Everything's fine." : "Uh, oh.";
    //        ^    \__________________/   \_______/
    //        |           if true          if false
    //        |          okay != 0          okay == 0
    //        |
    //        |__ some bool condition
}
```

```c
const char *f(int some_variable) {
    /*
    * Do something, and then report whether it's okay.
    * ...
    */

    if (some_variable == 88) {
        return "Everything's fine.";
    } else if (some_variable == 42) {
        return "Uh, oh.";
    } else if (some_variable == 13) {
        return "Much worse.";
    } else {
        return "Disaster";
    }
}
```

```c
const char *f(int some_variable) {
    /*
    * Do something, and then report whether it's okay.
    * ...
    */

    return some_variable == 88 ? "Everything's fine." :
           some_variable == 42 ? "Uh, oh."            :
           some_variable == 13 ? "Much worse."        :
                                 "Disaster"           ;
}
```

PREPROCESSOR TRICKS

- You've heard of

  `#define`
- You've noticed

  `#ifndef`
- Have you heard of

  `#if`

  `#`

  `##`

USE #IF TO CHOOSE SOURCE LINES

```
#define COMPILE_THIS_WAY 1

#if COMPILE_THIS_WAY
    whatever();
#else
    otherwise();
#endif
```

# #

- **#** is the preprocessor's "stringize" operator
- This definition

```
#define MACRO(P) #P
```

- used as

```
MACRO(abc)
```

- becomes

```
"abc"
```

- Can be used to make one's own assert macro

```
#define myassert(C) if (!(C)) my_assert_function(#C)
```

TOKEN PASTING

- Use to make a new identifier!
- Given this definition

```
#define   PASTE(A, B)   A ## _ ## B
```

- use as

```
PASTE(abc, def)
```

- becomes the identifier

```
abc_def
```

- Used to convert a simple table into a matrix of complex C code

* IN PRINTF()

- You can put a width parameter in a printf() format string

```
printf("-%20s-\n", "abc");
printf("-%-20s-\n", "abc");
```

- prints

```
-                   abc-
-abc                   -
 \__________________/
       20 spaces
```

* IN PRINTF()

- You can put a * instead of the width parameter
- Pass an integer width value

```
int w = 10;
printf("-%*s-\n", w, "abc");
printf("-%-*s-\n", w, "abc");
```

- prints

```
-       abc-
-abc       -
  \_______/
   w spaces
```

#INCLUDE <TIME.H>

- There are several time functions, but this one lets you measure execution time
- Start with

```
#include <time.h>
```

- then

```
clock_t t1 = clock();
/* do something that takes time */
clock_t t2 = clock();
```

- Compute the duration

```
double sec_duration = (double) (t2 - t1) / CLOCKS_PER_SEC;
printf("duration = %f seconds\n", sec_duration);
```

TIME TRAVEL! (REWINDING A FILE) FTELL() AND FSEEK()

- Open and read a file for a while. Then record where it is in **here**.

```
FILE *f = fopen()
/* Read some stuff */
long here = ftell(f);
/* Read more stuff */
fseek(f, here, SEEK_SET);
/* Reread more stuff */
```

- After calling fseek(), you can reread **more stuff**
- Used for random access to a file!

TIME TRAVEL! (REWINDING A FILE) FTELL() AND FSEEK()

- Open and read a file for a while. Then record where it is in **here**.

```
FILE *f = fopen()
/* Read some stuff */
long here = ftell(f);
/* Read more stuff */
fseek(f, here, SEEK_SET);
/* Reread more stuff */
```

- After calling fseek(), you can reread **more stuff**
- Used for random access to a file!

TIME TRAVEL! (REWINDING A FILE) FTELL() AND FSEEK()

- Open and read a file for a while. Then record where it is in **here**.

```
FILE *f = fopen()
/* Read some stuff */
long here = ftell(f);
/* Read more stuff */
fseek(f, here, SEEK_SET);
/* Reread more stuff */
```

- After calling fseek(), you can reread **more stuff**
- Used for random access to a file!

IS_____() CHARACTER MACROS

- Instead of asking a question like

```
if ('0' <= ch && ch <= '9')
```

- You can say

```
if (isdigit(ch))
```

- Others

```
isalpha(ch)
isupper(ch)
islower(ch)
```

- See `man isalpha` for the rest
- Also check out `man toupper`

SETJMP() AND LONGJMP()

- Were you told not to use goto?
- Get a load of this!
- (Goto a different function!)

```c
jmp_buf env;    // usually a global

void do_something() {
    printf("3. Here, in do_something(), we decide to longjmp()\n");
    longjmp(env, 1);
    printf("(Never) Do we get here?\n");
}

int main(void) {
    printf("1. Get ready . . .\n");
    if (setjmp(env)) {
        // longjmp() ends up here.
        printf("4. Warped here through longjmp()\n");
    } else {
        printf("2. About to do something\n");
        do_something();
        printf("(Never) Did something\n");
    }
    printf("5. Whew!\n");

    return 0;
}
```

```c
jmp_buf env;    // usually a global

void do_something() {
    printf("3. Here, in do_something(), we decide to longjmp()\n");
    longjmp(env, 1);
    printf("(Never) Do we get here?\n");
}

int main(void) {
    printf("1. Get ready . . .\n");
    if (setjmp(env)) {
        // longjmp() ends up here.
        printf("4. Warped here through longjmp()\n");
    } else {
        printf("2. About to do something\n");
        do_something();
        printf("(Never) Did something\n");
    }
    printf("5. Whew!\n");

    return 0;
}
```

```c
jmp_buf env;    // usually a global

void do_something() {
    printf("3. Here, in do_something(), we decide to longjmp()\n");
    longjmp(env, 1);
    printf("(Never) Do we get here?\n");
}

int main(void) {
    printf("1. Get ready . . .\n");
    if (setjmp(env)) {
        // longjmp() ends up here.
        printf("4. Warped here through longjmp()\n");
    } else {
        printf("2. About to do something\n");
        do_something();
        printf("(Never) Did something\n");
    }
    printf("5. Whew!\n");

    return 0;
}
```

```c
jmp_buf env;    // usually a global

void do_something() {
    printf("3. Here, in do_something(), we decide to longjmp()\n");
    longjmp(env, 1);
    printf("(Never) Do we get here?\n");
}

int main(void) {
    printf("1. Get ready . . .\n");
    if (setjmp(env)) {
        // longjmp() ends up here.
        printf("4. Warped here through longjmp()\n");
    } else {
        printf("2. About to do something\n");
        do_something();
        printf("(Never) Did something\n");
    }
    printf("5. Whew!\n");

    return 0;
}
```

```c
jmp_buf env;    // usually a global

void do_something() {
    printf("3. Here, in do_something(), we decide to longjmp()\n");
    longjmp(env, 1);
    printf("(Never) Do we get here?\n");
}

int main(void) {
    printf("1. Get ready . . .\n");
    if (setjmp(env)) {
        // longjmp() ends up here.
        printf("4. Warped here through longjmp()\n");
    } else {
        printf("2. About to do something\n");
        do_something();
        printf("(Never) Did something\n");
    }
    printf("5. Whew!\n");

    return 0;
}
```

SETJMP() AND LONGJMP()

- Why do we do this?
- Graceful handling of fatal errors