Welcome back to CS429H!

Week 1

Best Ed meme of the week:

Questions on lecture content?
Or about cats?

Quiz everyone say YAY!

Poll

```
int fd =
open("feedback.txt",O_RDONLY);
```

How was the quiz?

A. easy
B. mostly fine
C. mostly fine, but not enough time
D. too hard, but finished mostly in time
E. too hard and not enough time
F. too hard regardless of time

Stress

- 429H is not an easy class
  - Lots of new materials
  - Unfamiliar programming environments
  - Fast, often relentless pace
- Struggling in this course is normal
  - There will be times you won't know the answer of the solution
  - This is expected—we want we everyone to succeed, but the only way we can help is if you ask for it
- If you find yourself overly overwhelmed or spending more time on this class than you think you should be, please reach out to Dr. Gheith or the TAs
  - We can help out as far as the class goes
  - We can provide other resources where we are not able to help

Mental health resource available at UT

P1 Postmortem

- Grades will be released by next discussion (putting this here to hold us accountable lmao)
- Correctness
  - Good job!
  - If you want us to grade a late commit, please make a regrade request
- Test cases
  - Stress tests - ok, but you don’t need to make then 200k lines…
- Code quality
  - Very good! Keep in mind that for p2 we will start checking for memory leaks
- Reports
  - Awesome!

Slides I stole from last year

what does this code output? 1/8

```c
#include <stdio.h>

typedef struct Person {
    int age;
} Person;

Person create_person(int age) {
    Person p = {age};
    return p;
}

int main() {
   int myAge = 22;
   Person p = create_person(myAge);
   printf("Age: %d\n", p.age);
}
```

Slides I stole from last year

what does this code output? 2/8

```c
#include <stdio.h>

typedef struct Person {
    int age;
} Person;

Person *create_person(int age) {
    Person p = {age};
    return &p;
}

int main() {
   int myAge = 22;
   Person *p = create_person(myAge);
   printf("Age: %d\n", p->age);
}
```

Slides I stole from last year

what does this code output? 3/8

```
#include <stdio.h>

typedef struct Person {
    int age;
} Person;

Person create_person(int *age) {
    Person p = {*age};
    return p;
}

int main() {
   int myAge = 22;
   Person p = create_person(&myAge);
   printf("Age: %d\n", p.age);
}
```

Slides I stole from last year

what does this code output? 4/8

```
#include <stdio.h>

typedef struct Person {
    int age;
} Person;

Person create_person(int *age) {
    Person p = {*age};
    return p;
}

int main() {
   int *myAge = malloc(sizeof(int));
   *myAge = 22;
   Person p = create_person(myAge);
   printf("Age: %d\n", p.age);
}
```

Slides I stole from last year

what does this code output? 5/8

```
#include <stdio.h>

typedef struct Person {
    int age;
} Person;

Person *create_person(int age) {
    return malloc(sizeof(Person));
}

int main() {
   Person *p = create_person(22);
   printf("Age: %d\n", p->age);
}
```

Slides I stole from last year

what does this code output? 6/8

```c
#include <stdio.h>

typedef struct Person {
    int age;
} Person;

Person *create_person(int age) {
    return calloc(1, sizeof(Person));
}

int main() {
   Person *p = create_person(22);
   printf("Age: %d\n", p->age);
}
```

Slides I stole from last year

what does this code output? 7/8

```
#include <stdio.h>

typedef struct Person {
    int age;
} Person;

Person *create_person(int age) {
    Person *p = malloc(sizeof(Person));
    p->age = age;
    return p;
}

int main() {
   Person* p = create_person(22);
   free(p);
   printf("Age: %d\n", p->age);
}
```

Slides I stole from last year

what does this code output? 8/8

```
#include <stdio.h>

typedef struct Person {
    int age;
} Person;

Person create_person(int *age) {
    Person p = {*age};
    free(age);
    return p;
}

int main() {
   int myAge = 22;
   Person p = create_person(&myAge);
   printf("Age: %d\n", p.age);
}
```

✨GDB and Valgrind Demo✨

gdb cheatsheet

la / layout src // user-friendly view
r / run [args] // start program, continuing to next breakpoint / end of program
b / break <linenum/function...> <conditional> // set a breakpoint to stop at
c / continue // continue to the next breakpoint / end of program
n / next // go to next line
s / step // step into a function / over a line if not on a function call
f / finish // finish running the current function and return to the parent frame
p / print <variable> // print out value of specified thing
x <variable>/<memory address> // examine a chunk of memory
bt / backtrace // print the execution stack (like exception trace)
watch <variable> // watch a memory location (break once it changes)

Typical control flow:

1. gdb <executable>
2. b main
3. r <args>

linux terminal cheatsheet

`cd <folder>` - change working directory

`mv <src> <dest>` - move file

`cp <src> <dest>` - copy file

`man <cmd>` - manual for a command

`pwd` - tells you your current working directory

`mkdir <dirname>` - make a new directory

`ls` - shows you the files in your current directory

`make` - runs the Makefile, generally builds a binary

`touch <file>` - make a blank file

`nano <file>` - simple command-line text editor

`vim <file>` - superior command-line text editor

`ssh <username>@<ip>` - secure remote shell

`scp <username>@<ip>:~/file <dest>` - copy a file/folder over ssh

P2

Poll

How’s your status on P2?

A. What’s P2?
B. I’ve heard of it
C. I’ve cloned the starter code and/or looked through it
D. I’ve started planning/writing code
E. I’m mostly done but might still have bugs
F. P2 any% speedrun

New operators!

- <=, <, >=, >, ==, !=, &&, ||, &
  - Be careful when handling multi-character operators (look-ahead?)
- An interesting one: “,”
  - What does this do?
  - i.e.
    - `a = 1, 2, 3`

Statements vs Expressions

- Statements - complete requests (e.g. printing, assignment, return)
- Expressions - produce a value (e.g. arithmetic, logic, function call, function definition)

Statements

```
print
if
while                    <expr>
<identifier> =
else
return
```

Expressions

```
<const>
<identifier>
fun { <statements> }
<expr> <op> <expr>
<function call> (<expr>)
```

Conditional Statements

- if-else statements
- while loops
- no for loops

Fun Expression

- not “func”!
  - Provided test cases generally have precedence over the README in defining the spec
- Defines a function <u>without executing it</u>
  - Need a way to come back later to run the function
- Expression returns a uint64_t value representing function
  - No restrictions on how this value looks – as long as it is unique, you can represents functions however you want
  - This means you can treat it as a mystery expression - you can’t know anything about it other than the fact that it is a value

Function Call Expression

- **Not** a statement – must always be used as part of an expression
  - not allowed: `f(3)`
  - great: `x = f(3)`
- What if a function doesn't explicitly return a value?
  - return 0

Scope

```
it = 10
f1 = fun {
    print it
    z = f2(it*2)
    print it
}
f2 = fun {
    it = it + 1
    print it
}
print it
z = f1(15)
print it
```

What is the output? (spaces = newline)

a)	10	10	10	10	10
b)	10	15	30	31	31
c)	10	15	31	31	31
d)	10	15	31	15	10
e)	10	10	11	11	15

Tokenization

- Tokenization: take an arbitrary string and separate it into “tokens” according to some syntax rules
    - How is this useful for our interpreter?
- Pre-Tokenization: performing the tokenization step before the interpreter starts parsing a program
    - How can you use pre-tokenization to make an interpreter more efficient?
- Pre-tokenize once and run many times
    - Really useful for loops/functions/things that are run a lot
- Why should we care?
    - If you want a prize...

Enums

- Very simple in C:

```
typedef enum Keyword {
    PRINT,
    IF,
    ELSE,
    WHILE,
    FUN
} Keyword;
```

- By default, correspond to ints starting from 0 and counting up (PRINT=0, IF=1, etc)
- Why could this be useful?
- Side note: what is the typedef doing here?

Fun Pointer Magic!

- What is a function pointer, and how is it different from a function?
- In the p2 README we're told that a fun expression evaluates to an "opaque 64-bit quantity" which is used to identify the function
  - Does this remind you of anything? :3
- Running a C function using a function pointer
- Is there something like this we can do in fun?

```
void foo() { printf("hi"); }
int main() {
    void(*bar)();
    bar = foo;
    bar();
}
```

Short Circuiting

- What is the output of this fun code?

```
x = 1
f = fun {
    x = 5
}
if (1 || f()) print x
```

bool effects

- What good is it?
- Why would it be nice to have a state variable passed down during recursive descent?

Assembly Review

- What is assembly?
  - It is the lowest-level human-readable interface to encode a sequence of instructions
- Why should we care about assembly?
  - It helps us understand what the machine is doing when we run compiled code
- What are the different types of assembly?
  - There are a *lot*: x86[_64], ARM, RISC-V, PowerPC, and more!
- **Why** are there different types of assembly?
  - Each corresponds to a different underlying **architecture**, with different abstractions and operations
- In this class, we will be discussing 2 architectures: AMD64 (x86_64), and AArch64 (ARM)
  - What are some differences between these architectures?

AMD64 vs. AArch64

- They both start with an A
- CISC
- Faster or slower per instruction?
- Why do you think AMD64 is so popular for laptop/desktop/server machines?
    - Will it be in the future?

- They both end with 64
- RISC
- More energy efficient or less energy efficient?
- Why is AArch64 so popular for embedded/mobile/microcontroller platforms?
    - Will it be in the future?

Questions?

```
                          oooo$$$$$$$$$$$$oooo
                      oo$$$$$$$$$$$$$$$$$$$$$$$$o
                   oo$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$o         o$   $$ o$
   o $ oo        o$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$o       $$ $$ $$o$
oo $ $ "$      o$$$$$$$$$    $$$$$$$$$$$$$    $$$$$$$$$o       $$$o$$o$
"$$$$$$o$     o$$$$$$$$$      $$$$$$$$$$$      $$$$$$$$$$o    $$$$$$$$
  $$$$$$$    $$$$$$$$$$$      $$$$$$$$$$$      $$$$$$$$$$$$$$$$$$$$$$$
  $$$$$$$$$$$$$$$$$$$$$$$    $$$$$$$$$$$$$    $$$$$$$$$$$$$$  """$$$
   "$$$""""$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$     "$$$
    $$$   o$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$     "$$$o
   o$$"   $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$       $$$o
   $$$    $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$" "$$$$$$ooooo$$$$o
  o$$$oooo$$$$$  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$   o$$$$$$$$$$$$$$$$$
  $$$$$$$$"$$$$   $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$     $$$$""""""""
 """"       $$$$    "$$$$$$$$$$$$$$$$$$$$$$$$$$$$"      o$$$
            "$$$o     """$$$$$$$$$$$$$$$$$$"$$"         $$$
              $$$o          "$$""$$$$$$""""           o$$$
               $$$$o                                o$$$"
                "$$$$o      o$$$$$$o"$$$$o        o$$$$
                  "$$$$$oo     ""$$$$o$$$$$o   o$$$$""
                     ""$$$$$oooo  "$$$o$$$$$$$$$"""
                        ""$$$$$$$oo $$$$$$$$$$
                                """"$$$$$$$$$$$
                                    $$$$$$$$$$$$
                                     $$$$$$$$$$"
                                      "$$$""""
```