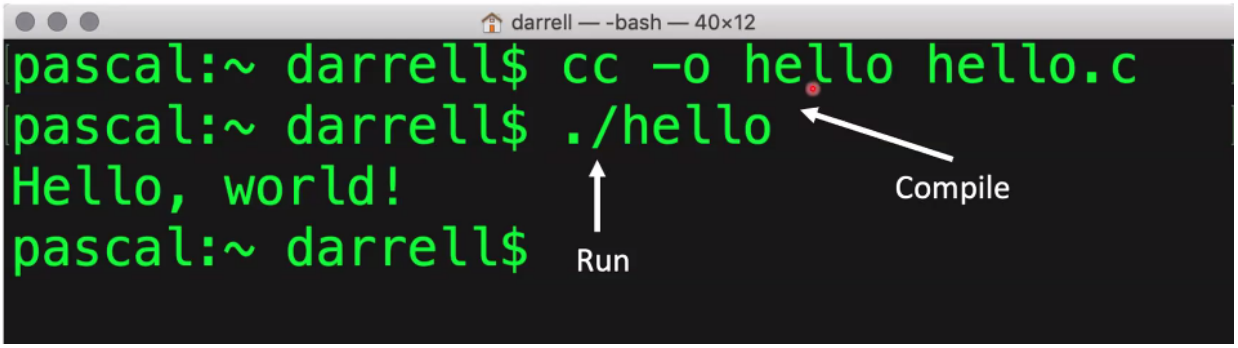
- Programs are assigned by *Saturday night*.
- Design documents are due *Thursday night*.
  - Preliminary design is 50% of the *design* grade, and
  - Final design is the other 50% of the *design* grade.
- Programs are due on *Sunday night*.
- No design on Thursday? *Zero* on the design document — no excuses!

- design document in pdf
- create an ssh key in windows
- you can write design document on overleaf
- no directions for design document; just readable
  - input output what does the program do; what do i need to do to run this program

```
» git commit -am "Dumb edit"
```

asgn 4 - use swap()

asgn1 - #include names.h



```c
#include <stdio.h>

// Print a table of °F to °C, for 0 to 300

int main(void) {
  int fahr, celsius;
  int lower = 0, upper = 300, step = 20;
  fahr = lower;
  while (fahr <= upper) {
    celsius = 5 * (fahr - 32) / 9;
    printf("%d\t%d\n", fahr, celsius);
    fahr = fahr + step;
  }
  return 0;
}
```

Should we be concerned?

```
pascal:~ darrell$ cc -o fahr fahr.c
pascal:~ darrell$ ./fahr | head -10
0       -17
20      -6
40      4
60      15
80      26
100     37
120     48
140     60
160     71
180     82
pascal:~ darrell$
```

27 September 2021

Wednesday
and distributes over or

- $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$

or distributes over and

- $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$

- In **C**, zero (0) is *false*.
- All that is not *false* is *true*.
- Logical expressions have type `int`.
- You can have `true` and `false` if you:

```
#include <stdbool.h>
```

booleans are integers

how to print variable:

```c
#include <stdio.h>

int main(void) {
  int a, b, c;
  scanf("%d %d %d", &a, &b, &c);
  if (a < b) {
    if (c < a) {
      printf("c = %d is smallest\n", c);
    } else {
      printf("a = %d is smallest\n", a);
    }
  } else {
    if (c < b) {
      printf("c = %d is smallest\n", c);
    } else {
      printf("b = %d is smallest\n", b);
    }
  }
  return 0;
}
"min.c" 20L, 365C
```

c evaluates if statements left to right and stops once it knows the statement; A && B, if a is false it won't check b

go learn switch()

you can use this for the pig code

do { } while ()

- This is a bottom-test loop.
- Used when you want to perform the statement at least once.
- Continues to execute the enclosed statement as long as the Boolean condition remains *true*.

```
i = 1;
do {
   printf("%d\n", i);
   i = i + 1;
} while (i <= 10);
```

this is like a while loop that executes at least once; you are going to roll the pig at least once

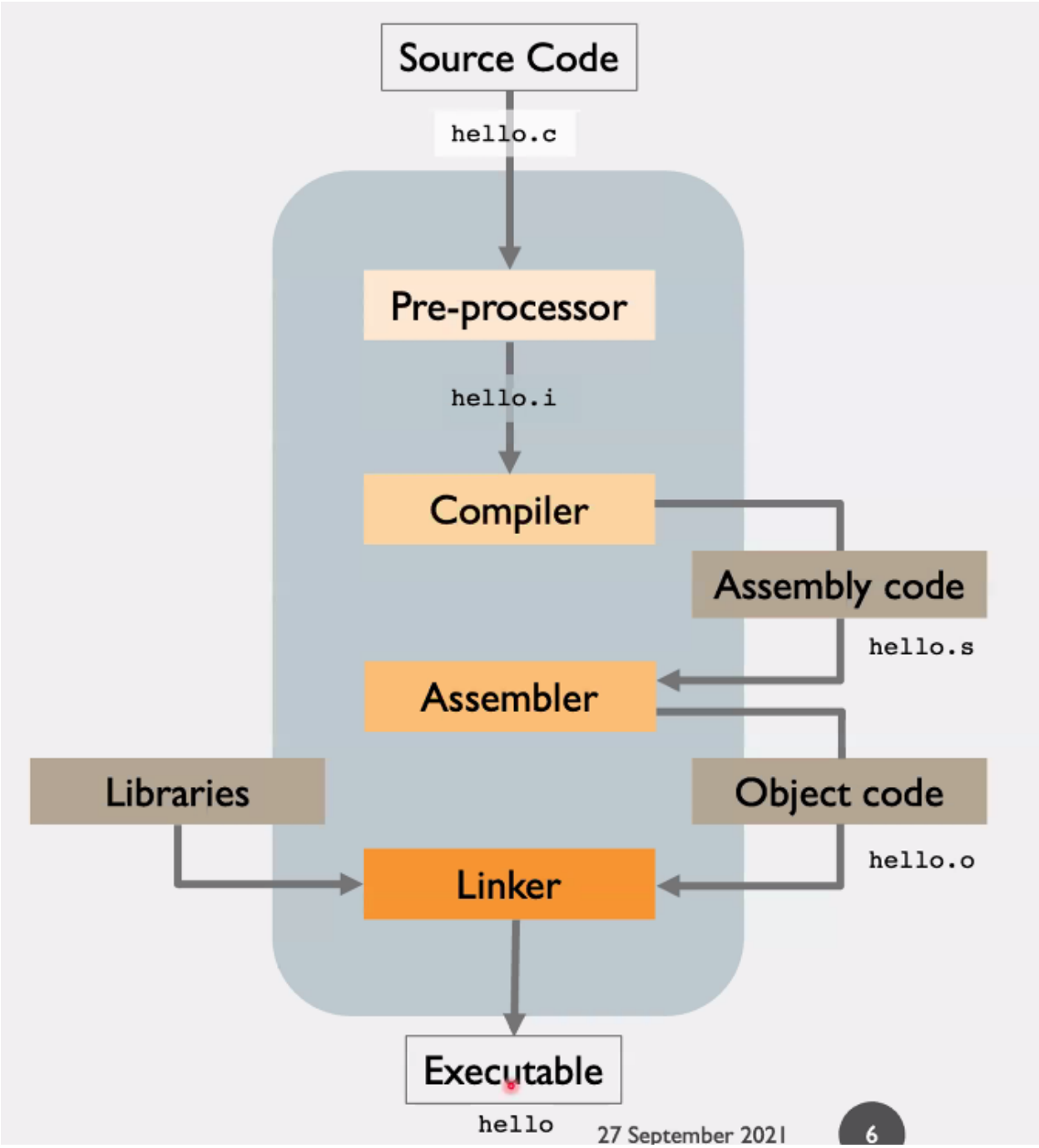
asgn2 hint; you don't need to compute factorials n(n-1); you can avoid extra work by doing algebra

goto can be used as an exception

*** switch statements are good for rolling dice

seed conditions:

1. integer
2. positive
3. smaller than the limit on unsigned integers



makefile resource in textbook

the name of the array is just the pointer; he said to write this down maybe refer back to the slide show

- sizeof function is unreliable

- arrays and pointers on the quiz; professor recommended reading the textbook
- dynamic arrays on asgn3

- understand binary search
- understand ternary operators (textbook)

Tuesday

- include 0 in natural numbers
- quiz on friday will have binary arithmetic
- Floating point numbers: *F*
- *F* is a subset of *R Q* and *Z*

- your gonna use taylor series in asgn2
    - madhava

everything lest than 10 lines of code

do 1 and then do the switch case, do another then the test case

read getopt?

```c
#include <math.h>
#include <stdio.h>

#define EPSILON 1e-14

static inline double absolute(double x) {
    return x < 0 ? -x : x;
}

double sine(double x) {
    double t = x;
    double s = t; // 1st term.

    for (double n = 3.0, p = -1; absolute(t) > EPSILON; n += 2.0, p *= -1) {
        t *= (x / n) * (x / (n - 1)); // Next odd term.
        s += p * t; // Add it in.
    }

    return s;
}

int main(void) {
    for (double x = -4.0 * M_PI; x < 4.0 * M_PI; x += 0.1) {
        printf("sine(%lf) = %16.15lf, sin(%lf) = %16.15lf, diff = %16.15lf\n",
            x, sine(x), x, sin(x), absolute(sine(x) - sin(x)));
    }
}
```

quick sort: for every pivot you can choose there exists an adversarial array that results in $O(n^2)$ time

//taken from sudo code

```c
typedef enum Fn { E, EULER, BBP, MADHAVA, VIETE, NEWTON } Fn;

int opt = 0;
while ((opt = getopt_long(argc, argv, OPTIONS, options, NULL)) != -1) {
    switch (opt) {
    case 's': { stats = true; break; }
    case 'a': { fns = set_complement(set_empty()); break; }
    case 'e': { fns = set_insert(fns, E);          break; }
    case 'r': { fns = set_insert(fns, EULER);      break; }
    case 'b': { fns = set_insert(fns, BBP);        break; }
    case 'm': { fns = set_insert(fns, MADHAVA);    break; }
    case 'v': { fns = set_insert(fns, VIETE);      break; }
    case 'n': { fns = set_insert(fns, NEWTON);     break; }
    case 'h': { usage(argv[0]); return EXIT_SUCCESS; }
    default:  { usage(argv[0]); return EXIT_FAILURE; }
    }
}
```

- be able to look at a sorting algorithm and be able to find O(n)
- how many times can you cut x in half before you hit 1?
    - $\log_2 x$

- O(n) + O(n) = O(n)
    - O(n) nested within another O(n)

for small n bubble sort is faster than quick sort (probably < 50)
bubble sort between O(n) and O(n^2)

plot.sh

```
#!/usr/bin/env bash

rm -f /tmp/{heap,insertion,quick,shell}.dat

for i in {0..2}; do
    awk -F, '{
        split($1,name," ");
        split($2,elements," ");
        split($3,moves," ");
        split$4,compares," ");
        file = sprintf("/tmp/%s.dat", tolower(name[1]));
        printf "%d %d %d\n", elements[1], moves[1], compares[1] >> file
    }' <(./sorting -a -n $(( 10**i )) -p 0)
done

gnuplot <<EOF
set terminal pdf
set bmargin 4
set key outside
set size ratio 0.75
set xlabel "Elements"

set output "moves.pdf"
set title "Moves Performed"
set ylabel "Moves"
plot "/tmp/heap.dat" using 1:2 with linespoints title "Heap Sort", \
    "/tmp/insertion.dat" using 1:2 with linespoints title "Insertion Sort", \
    "/tmp/quick.dat" using 1:2 with linespoints title "Quick Sort", \
```