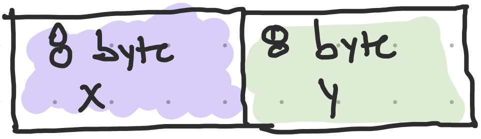
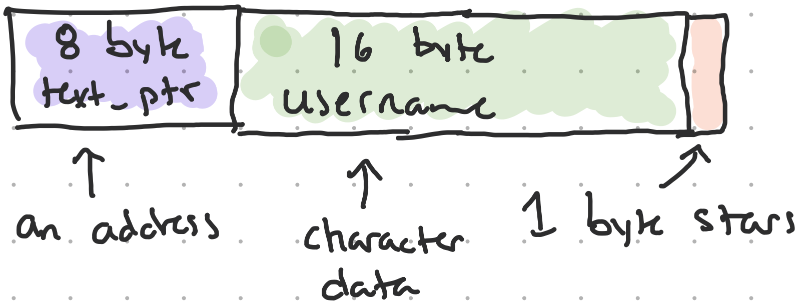
```
struct Point {
    double x;
    double y;
}
```

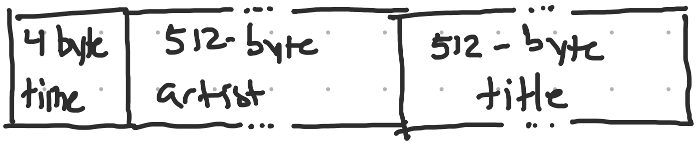


```
struct YelpReview {
    char* text_ptr;
    char username[16];
    uint8_t stars;
}
```



Q1: Fill in the struct definition for a Spotify song

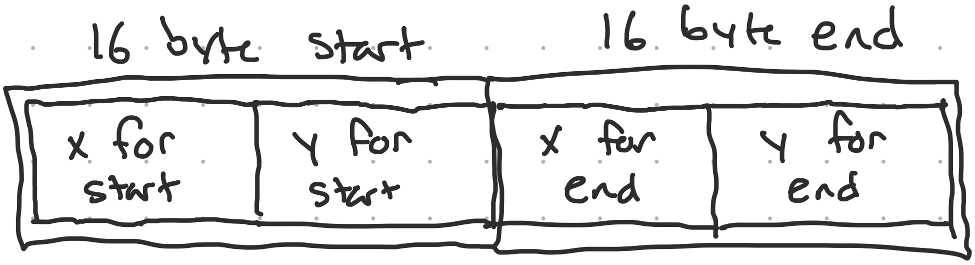
```
struct Song {
    uint32_t time;
    char artist[512];
    char title[512];
}
```



A struct describes a multi-value memory layout.

Q2: Draw the picture!

```
struct LineSegment {
    struct Point start;
    struct Point end;
}
```



Q3: What is sizeof for each struct above?

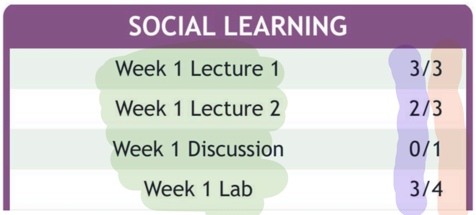
Point 16

YelpReview 25* (32)

Song 512+512+4=1028

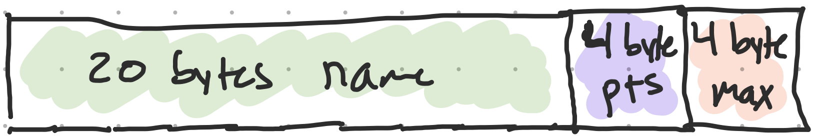
LineSegment 32

```
#include <stdio.h>

struct Grade {
  char name[20];
  int pts;
  int max_pts;
};
```



SOCIAL LEARNING

Week 1 Lecture 1	3/3
Week 1 Lecture 2	2/3
Week 1 Discussion	0/1
Week 1 Lab	3/4



```
void example1() {
  Grade g = { "Week 1 Lec 1", 1, 3 };
  printf("%ld\n", sizeof(g));
  printf("%p %p %p %p\n", &g, &g.name, &g.pts, &g.max_pts);
}
```

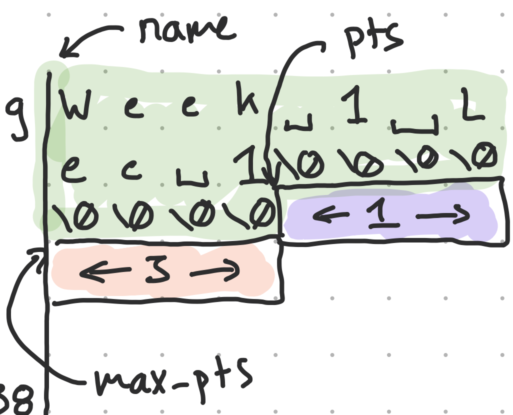
sizeof(g) = 28

&g = 0xff... 20

&g.name = 0xff... 20

&g.pts = 0xff... 34

&g.max_pts = 0xff...38



```
typedef struct Grade Grade;
```

allows us to write Grade as a type instead of "struct Grade"

```
void show_grade(Grade g, char* result) {
  sprintf(result, "%s: %d/%d", g.name, g.pts, g.max_pts);

}
```

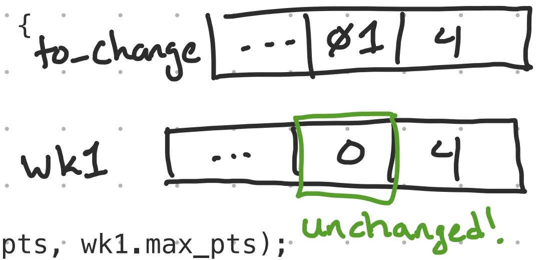
↳ like other arrays, gets address

sprintf is like printf, but stores its printed output in the 1st arg (a char*) instead of stdout.
(assumes result long enough)

```
void example2() {
  Grade g = { "Week 1 Lec 1", 1, 3 };
  char output[ 64 ];
  show_grade( g , output);
  printf( "%s\n", output )
}
```

write out a call to show_grade + print result

```
void regrade(Grade to_change, int new_pts) {
  to_change.pts = new_pts;
}

void example3() {
  Grade wk1 = { "Week 1 Lab", 0, 4 };
  regrade(wk1, 1);
  printf("Regrading 0 to 1: %d/%d\n", wk1.pts, wk1.max_pts);
}
```



This prints 0/1 even after the "regrade"!

struct instances are copied on function calls (unlike arrays!)

p->x looks up x field when p is ptr to struct

```
void really_regrade( Grade* g, int new_pts ) {
  g->pts = new_pts;
}
```

g[0].pts = new_pts;

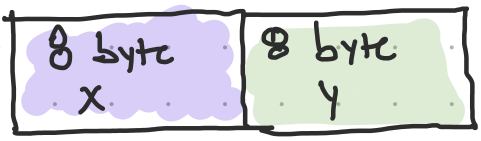
```
void example4() {
  Grade wk1 = { "Week 1 Lab", 0, 4 };

  really_regrade( &wk1, 1 );

  printf("Regrading 0 to 1: %d/%d\n", wk1.pts, wk1.max_pts);
}
```

This would print 1/4

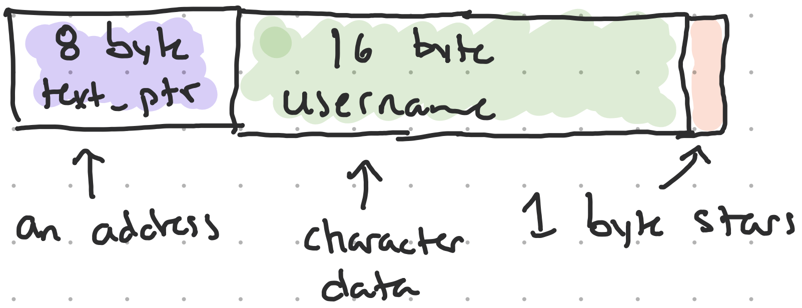
```
#define NUM_SOCIALS 40
typedef struct Summary {
  char student_name[100];
  Grade social_learning[NUM_SOCIALS];
} Summary;

Summary student1 = { "Jeremy Beremy",
  { { "Week 1 Lec 1", 3, 3 },
    { "Week 1 Lab", 4, 4 },
    { "Week 1 Discussion", 0, 1 } } };
```

```
void show_summary(Summary s) {
  printf("%s Social Learning\n", s.student_name);
  for(int i = 0; i < NUM_SOCIALS; i += 1) {
    Grade sl = s.social_learning[i];
    if(sl.name == NULL) { break; }
    printf("%s:\t %d/%d\n", sl.name, sl.pts, sl.max_pts);
  }

  int total =

  printf("Total: %d\n", total);
}

void example5() {
  show_summary(student1);
}
```

```
struct Point {
    double x;
    double y;
}
```

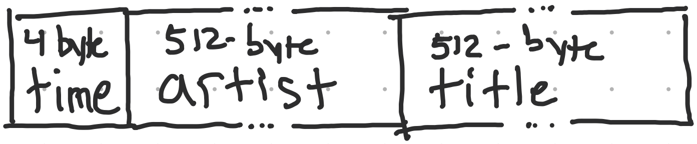


```
struct Review {
    char* text_ptr;
    char username[16];
    uint8_t stars;
}
```



Q1: Fill in the struct definition for a Spotify song

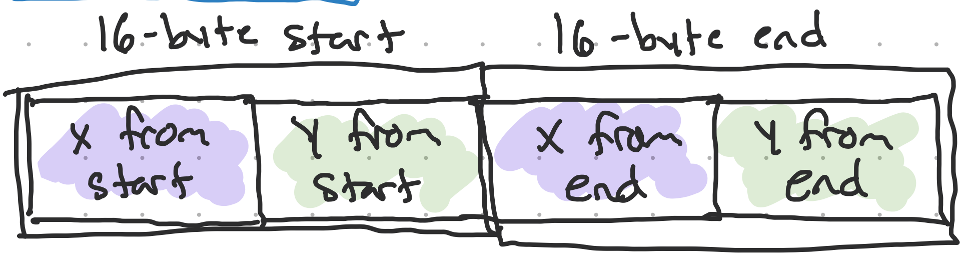
```
struct Song {  ←milliseconds
    uint32_t time;
    char artist[512];
    char title[512];
}
```



A struct describes a multi-value memory layout.

Q2: Draw the picture!

```
struct LineSegment {
    struct Point start;
    struct Point end;
}
```



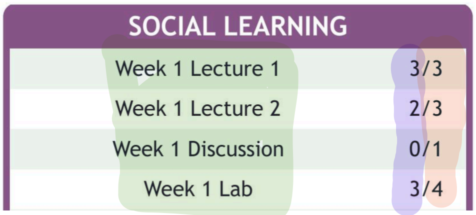
Q3: What is sizeof for each struct above?

Point 16

Song 1028

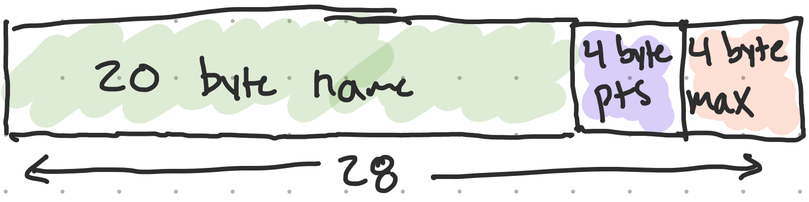
Review 25* (32)

LineSegment 32



SOCIAL LEARNING	
Week 1 Lecture 1	3/3
Week 1 Lecture 2	2/3
Week 1 Discussion	0/1
Week 1 Lab	3/4

```
#include <stdio.h>

struct Grade {
  char name[20];
  int pts;
  int max_pts;
};
```



```
void example1() {
  Grade g = { "Week 1 Lec 1", 1, 3 };
  printf("%ld\n", sizeof(g));
  printf("%p %p %p %p\n", &g, &g.name, &g.pts, &g.max_pts);
}
```

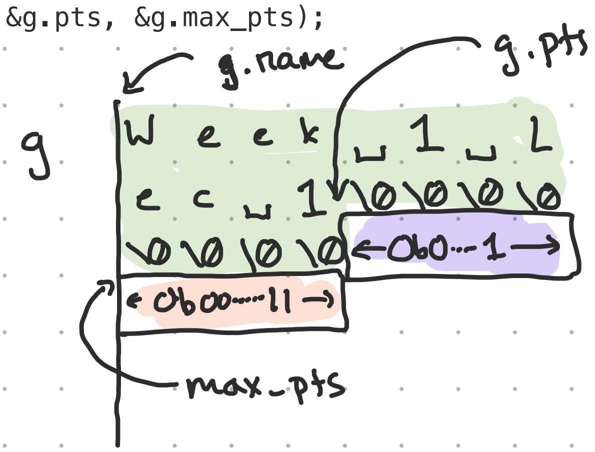
sizeof(g) = 28

&g = 0xff.... 40

&g.name = 0xff... 40

&g.pts = 0xff...54

&g.max_pts = 0xff...58



alias

```
typedef struct Grade Grade;
```

This lets us write Grade instead of "struct Grade" for the type

```
void show_grade(Grade g, char* result) {
  sprintf(result, "%s: %d/%d", g.name, g.pts, g.max_pts);

}
```

4 byte int

just like arrays "decays" into char*

sprintf does the same formatting as printf BUT stores output in the given char* (1st arg) instead of printing (assumes result is long enough)

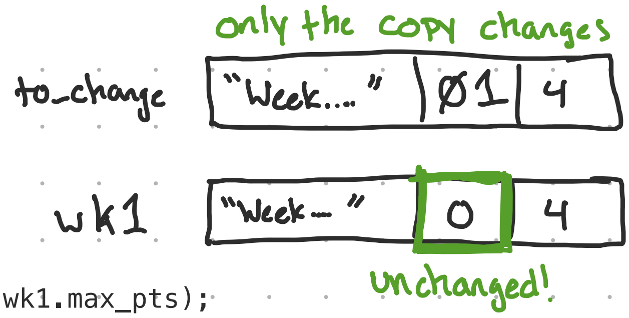
```
void example2() {
  Grade g = { "Week 1 Lec 1", 1, 3 };
  char output[ 64 ];
  show_grade(g, output);
  printf("%s\n", output);
}
```

call show_grade + print result

%d could cause more than 4 bytes of output (up to 11 characters -1 000 000 000

What if I wrote "char* output;"

```
void regrade(Grade to_change, int new_pts) {
  to_change.pts = new_pts;
}

void example3() {
  Grade wk1 = { "Week 1 Lab", 0, 4 };
  regrade(wk1, 1);
  printf("Regrading 0 to 1: %d/%d\n", wk1.pts, wk1.max_pts);
}
```



This → will print 0/4 despite the call to regrade.

struct instances are copied when passed as arguments (or returned) (unlike arrays!)

p->x = v
"dereferencing assignment"

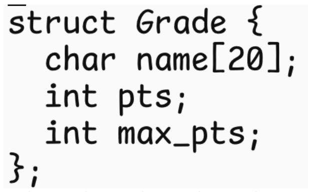
```
void really_regrade(Grade* to_change, int new_pts) {

  to_change->pts = new_pts;

}

void example4() {
  Grade wk1 = { "Week 1 Lab", 0, 4 };

  really_regrade( &wk1 , 1 );

  printf("Regrading 0 to 1: %d/%d\n", wk1.pts, wk1.max_pts);
}
```

to_change[0].pts = new_pts

prints 1/4 in result

```
#define NUM_SOCIALS 40
typedef struct Summary {
  char student_name[100];
  Grade social_learning[NUM_SOCIALS];
} Summary;

Summary student1 = { "Jeremy Beremy",
  { { "Week 1 Lec 1", 3, 3 },
    { "Week 1 Lab", 4, 4 },
    { "Week 1 Discussion", 0, 1 } } };
```

```
void show_summary(Summary s) {
  printf("%s Social Learning\n", s.student_name);
  for(int i = 0; i < NUM_SOCIALS; i += 1) {
    Grade sl = s.social_learning[i];
    if(sl.name == NULL) { break; }
    printf("%s:\t %d/%d\n", sl.name, sl.pts, sl.max_pts);
  }

  int total =

  printf("Total: %d\n", total);
}

void example5() {
  show_summary(student1);
}
```

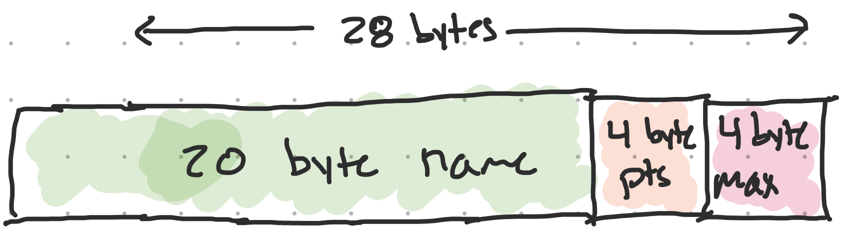


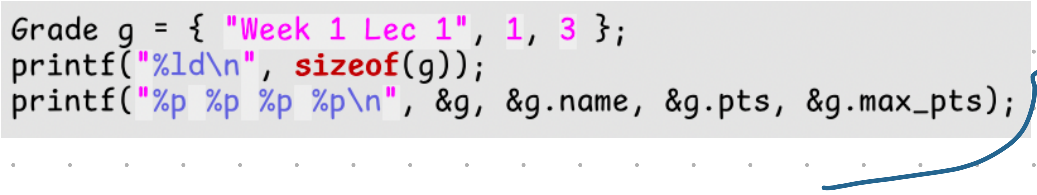
```
struct Grade {
  char name[20];
  int pts;
  int max_pts;
};
```

SOCIAL LEARNING

Week 1 Lecture 1	3/3
Week 1 Lecture 2	2/3
Week 1 Discussion	0/1
Week 1 Lab	3/4

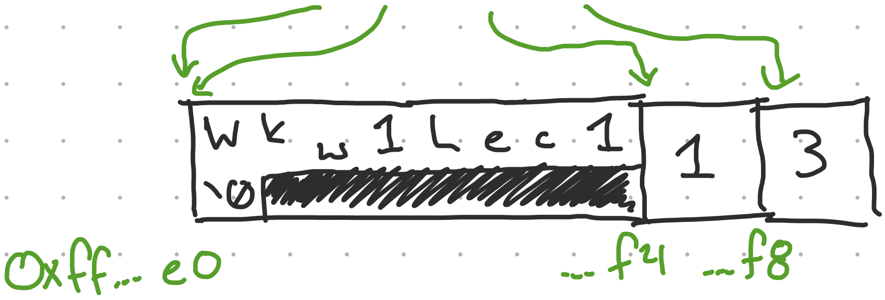
name pts max

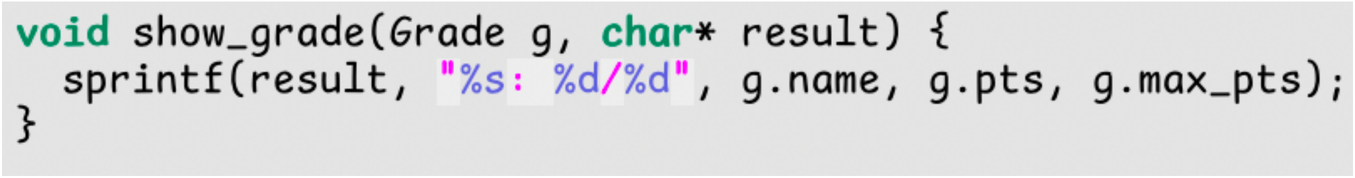




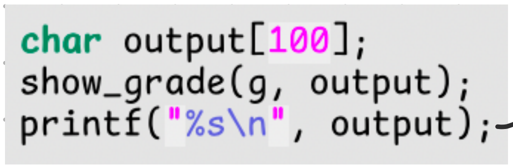
```
Grade g = { "Week 1 Lec 1", 1, 3 };
printf("%ld\n", sizeof(g));
printf("%p %p %p %p\n", &g, &g.name, &g.pts, &g.max_pts);
```

prints 28





```
void show_grade(Grade g, char* result) {
  sprintf(result, "%s: %d/%d", g.name, g.pts, g.max_pts);
}
```



```
char output[100];
show_grade(g, output);
printf("%s\n", output);
```

Week 1 Lec 1: 1/3

&x — x is a variable

&a[n] — a is an array or ptr
n is an index

&g.name — g is a struct instance
name is a field

✓ &g->pts

&(3 + x)

&(f())

✗ cannot ask for these addresses (no such thing)

---

x = v

a[n] = v

g.name = v

3 + x = v

f() = v

✗

g->pts = v

L - values
LHS - values

Expressions representing or evaluating to an assignable address

&x — x is a variable

&a[n] — a is an array or pointer
n is an index

&g.pts — g is a struct instance
pts is a field name

&g->pts — g is a pointer to a struct instance
pts is a fiel

x = v

a[n] = v

g.pts = v

g->pts = v

&(3 + y)

&(f(3))

✗ not valid assignable locations

✗ 3 + y = v

f(3) = v

L-values
LHS-values

Java

✓ o.m().x = 4

✗ o.m() = 4