CS 161
Introduction to CS I
Lecture 25

- Recursion recap
- Recursive data structures

Week 9 tips

- This week
  - Assignment 5 peer reviews – due Weds. 3/4 at midnight
  - Study session – Thursday 3/5 from 6-7 p.m. in LINC 268
  - Assignment 5 – due Sunday 3/8 at midnight
- Beyond week 9
  - Proficiency demo – week 10
  - Makeup assignment (6) – week 10
  - Final exam – Monday 3/16 from 6-7:50 p.m. in LINC 128

Oregon State University
College of Engineering

Grace Hopper Celebration Scholarship

- Conference: Sept. 29 – Oct. 2 in Orlando, FL
  - https://ghc.anitab.org/
- OSU EECS is offering scholarships for up to $1550 + conference registration
  - More info: https://oregonstate.box.com/s/vtq5ynvfdjb8lgs661lsdcvmy8es891g
  - Application deadline: March 27

Oregon State University
College of Engineering

Questions about Assignment 5?

- My Planet Treasure Chest

```
|_|_|_|
|_|D|_|
|_|T|_|
Total value of 2 items: $127
```

- You can make this nicer to look at, more color, better symbols
- Random generation of member values
  - Floats: add 0.0 – 1.7 to 2.3: `float(rand()%18)/10 + 2.3`

Review: Recursion

- What is it?
  - Function that calls itself 1 or more times (directly or indirectly)
  - Has 1 or more base cases for stopping
  - General case must eventually be reduced to a base case
- Recursive step: express relationship between problem(n) and smaller problem such as problem(n-1)
- Recursive call: calling a function inside itself.

Your turn: Palindromes with digits

- Palindrome: Same value when read forwards as backwards
  - e.g. 121, 67876, 3
- Pal(n): generate a palindromic digit string, given a starting digit

```
Input -> output
1 -> 1
2 -> 212
3 -> 32123
4 -> 4321234
```

- What is the base case?
  - 1 -> "1"
- What is the recursive step?
  - pal(n) = n + pal(n-1) + n

Your turn: Palindromes with digits

See lec25-pal-digits.cpp

- Implementation

```
1. string pal(char n) {
2.   if (n == '1')
3.     return "1";
4.   else
5.     return n + pal(n-1) + n;
6. }
```

```
Input -> output
1 -> 1
2 -> 212
3 -> 32123
4 -> 4321234
```

- What is the base case?
  - 1 -> "1"
- What is the recursive step?
  - pal(n) = n + pal(n-1) + n

Your turn: Palindromes with digits

- That could have been done easily with an iterative solution
  - Count from n down to 1 and back up to n: two for loops
- What about this version?

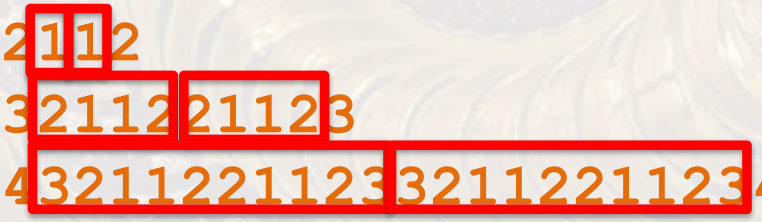
```
Input -> output
1 -> 1
2 -> 2112
3 -> 3211221123
4 -> 4321122112332112211234
```

- What is the base case?
  - 1 -> 1
- What is the recursive step?
  - pal(n) = n + pal(n-1) + pal(n-1) + n

Your turn: Palindromes with digits

- Implementation: give it a try on your own!

```
Input -> output
1 -> 1
2 -> 2112
3 -> 3211221123
4 -> 4321122112332112211234
```



- What is the base case?
  - 1 -> 1
- What is the recursive step?
  - pal(n) = n + pal(n-1) + pal(n-1) + n

Recursion with chocolate

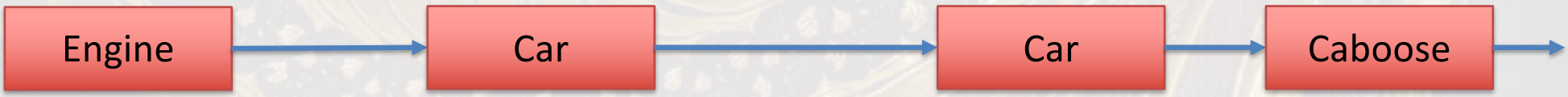
- How many chocolates are in this dish?

- Recursive definition of num_choc(dish):
  - **Base case:** num_choc(empty dish) = 0
  - **Recursive step:** num_choc(dish) = 1 + num_choc(dish – 1)

Recursive data structures

- Let's model a train
  - Train = one or more train_car items, ending with a caboose

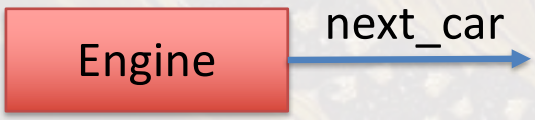
```
1. struct train_car {
2.   string kind;
3.   train_car* next_car;
4. };
```



Recursive data structures

- Let's create a train
  - First car is the engine

```
1. struct train_car {
2.   string kind;
3.   train_car* next_car;
4. };
```




```
1. train_car* my_train = new train_car;
2. my_train->kind = "Engine";
3. my_train->next_car = NULL;
```

Recursive data structures

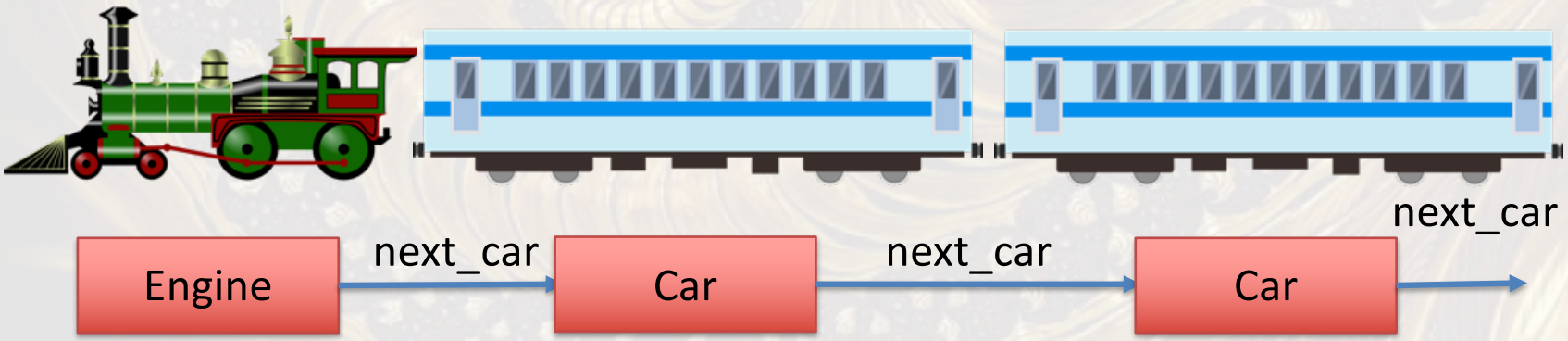
- Let's create a train
  - First car is the engine
  - Add more cars

```
1. struct train_car {
2.   string kind;
3.   train_car* next_car;
4. };
```

Recursive data structures

- Let's create a train
  - First car is the engine
  - Add more cars

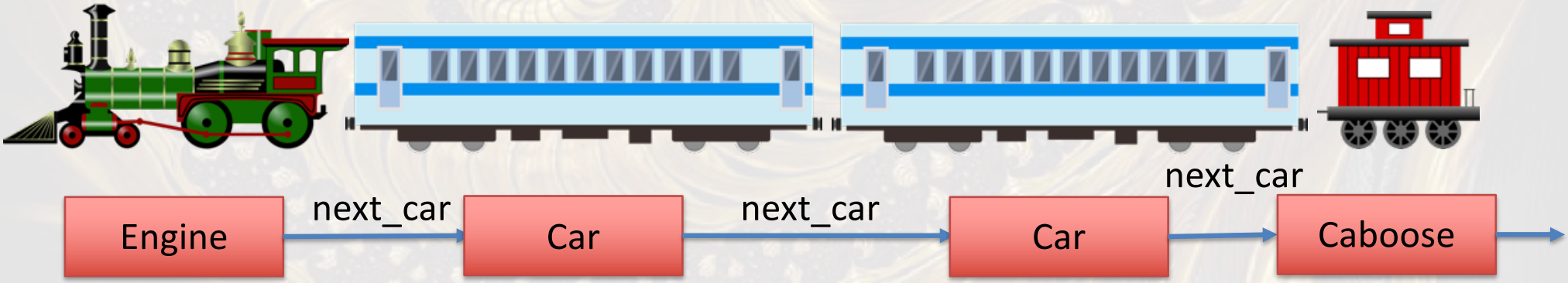
```
1. struct train_car {
2.   string kind;
3.   train_car* next_car;
4. };
```



Recursive data structures

- Let's create a train
  - First car is the engine
  - Last one is the **caboose**

```
1. struct train_car {
2.   string kind;
3.   train_car* next_car;
4. };
```



Oregon State University
College of Engineering

Recursive train creation

See lec25-recur-structs.cpp

- First car is the engine
- Last one is the **caboose**

```
1. int n_cars = rand()%10 + 1;
2. add_cars(my_train, n_cars);
```

```
1. void add_cars(train_car* t, int n_cars) {
2.   t->next_car = new train_car;  /* add a new car */
3.   t->next_car->next_car = NULL; /* be safe! */
4.   if (n_cars == 1) { /* base case: caboose */
5.     t->next_car->kind = "Caboose";
6.   } else {
7.     t->next_car->kind = "_***_";
8.     add_cars(t->next_car, n_cars-1); /* recursive call */
9.   }
10.}
```

Your turn: Recursively print the train

```
1. void print_train(train_car* t) {
2.   cout << t->kind;
3.   if (t->kind == "Caboose")
4.     cout << "\n";
5.   else
6.     print_train(t->next_car);
7. }
```

See lec25-recur-structs.cpp

```
1. struct train_car {
2.   string kind;
3.   train_car* next_car;
4. };
```

Gotchas

- Chasing your tail

```
1. train_car* t = new train_car;
2. t->kind = "Ouroboros";
3. t->next_car = t;
4. print_train(t);
```

- Walking off the end of the train

```
1. void print_train(train_car* t) {
2.   cout << t->kind;
3.   print_train(t->next_car);
4. }
```

What ideas and skills did we learn today?

- How recursion can be used to construct chains of data types (structs)
- How to traverse (e.g., print) a recursive data structure

- Challenge: implement
  `void delete_train(train_car* t);`
  to clean up the heap and avoid memory leaks

Week 9 continues

- ❑ Attend lab (laptop required)
- ❑ Read Rao lesson 7 (pp. 158-161)

  Read Miller lecture 8:
  http://www.doc.ic.ac.uk/~wjk/C++Intro/RobMillerL8.html
- ❑ **Assignment 5 peer reviews** (due **Wednesday, March 4**)
- ❑ Study session Thursday – see worksheet on calendar

See you Friday!