CS 161
Introduction to CS I
Lecture 26

• Deleting recursive data structures
• More recursion power
Final Week 9 tips

• Check Canvas for any missing grades
  • Notify cs161-020-ta@engr.orst.edu by next Wednesday (3/11)
  • Except: Missing peer grades for Assign. 2 and 3 were recently set to 0. Normally these points are given when you demo. If you missed a demo, you may incorrectly have a 0 (never graded). These are now being re-graded, so don't send an email about these unless they are still 0 next Monday.
  • Final grades are rounded (89.4 -> 89; 89.5 -> 90)
• Assignment 6 will be worth 80 points
  • Worth doing if any previous assignment earned < 80 points
  • Worth doing if you want practice with recursion 😊
Proficiency demo in week 10

• Go to your registered lab (or contact TAs)
• To prepare:
  • Review 1D arrays, 2D arrays, and C-style strings
  • Practice: Give yourself 50 minutes to try one or more of the sample prompts
  • Design on paper before you start coding
  • Take a deep breath!
• Any questions about what to expect?
Review: 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. };
Deleting recursive data structures

• Create the train:

1. `train_car* my_train = new train_car;`
2. `my_train->kind = "Engine";`
3. `my_train->next_car = NULL;`
4. `int n_cars = rand()%10 + 1;`
5. `add_cars(my_train, n_cars);`
Deleting recursive data structures

• Delete a train:

1. `train_car* my_train = new train_car;`
2. `my_train->kind = "Engine";`
3. `my_train->next_car = NULL;`
4. `int n_cars = rand()%10 + 1;`
5. `add_cars(my_train, n_cars);`
6. `delete my_train;`

This deletes the first train_car (Engine) only. The rest are lost forever.
Deleting recursive data structures

• Instead, let's delete the train with a recursive function:

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

4. int n_cars = rand()%10 + 1;
5. add_cars(my_train, n_cars);

6. delete_train(my_train);
7. my_train = NULL;
Deleting recursive data structures

- How did we create the train?

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

```cpp
1. void add_cars(train_car* t, int n_cars) {
2.     t->next_car = new train_car;
3.     t->next_car->next_car = NULL;
4.     if (n_cars == 1) {
5.         t->next_car->kind = "Caboose";
6.     } else {
7.         t->next_car->kind = "_***_");
8.         add_cars(t->next_car, n_cars-1);
9.     }
10. }
```
Deleting recursive data structures

• Delete a train:
  • Wait to delete the current train_car until the rest of the train is gone
    • Base case?
    • Recursive step?

1. struct train_car {
2.   string kind;
3.   train_car* next_car;
4.};
Deleting recursive data structures

• Delete a train:
  • Wait to delete the current train_car until the rest of the train is gone
  • Base case? Caboose

my_train

• Recursive step? Delete rest of train, then delete this car

my_train

1. struct train_car {
2.   string kind;
3.   train_car* next_car;
4.};
Your turn: Delete a train

- Delete a train:

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

1. void delete_train(train_car* t) {  
2.     if (t->kind == "Caboose") /* base case */  
3.         delete t;  
4.     else { /* recursive call */  
5.         /* Delete the rest of the train first */  
6.         delete_train(t->next_car);  
7.         /* Now delete this car */  
8.         delete t;  
9.     }  
10. }
```
How NOT to delete a train

• Delete a train:

```c
1. void delete_train(train_car* t) {
2.     if (t->kind == "Caboose") /* base case */
3.         delete t;
4.     else { /* recursive call */
5.         /* Delete this car */
6.         delete t;
7.         /* Delete the rest of the train */
8.         delete_train(t -> next_car);
9.     }
10. }
```
Our train_car is a linked list

• Add or remove cars as needed by reassigning pointers
What if each struct has two pointers? (Tree)

(From study worksheet 9)
Your turn: Define a box struct

1. struct box {
2.   int value;
3.   box* left;
4.   box* right;
5. };

Diagram: 5
          /   
        15   9
       /     
    2     28   6  14
    /     
   8     

Your turn: Set up level 1

1. `struct box {
2.   int value;
3.   box* left;
4.   box* right;
5.};`

```
1. box* my_tree = new box;
2. my_tree->value = 5;
3. my_tree->left = NULL;
4. my_tree->right = NULL;
```
Your turn: Set up level 2 (left child)

1. struct box {
2.   int value;
3.   box* left;
4.   box* right;
5.};

1. my_tree->left = new box;
2. my_tree->left->value = 15;
3. my_tree->left->left = NULL;
4. my_tree->left->right = NULL;

Same process for right child (try it on your own)
1. void delete_tree(box* b) {
2.     if (b == NULL) /* base case */
3.         return;
4.     else {
5.         /* delete sub-trees first */
6.         delete_tree(b->left);
7.         delete_tree(b->right);
8.         /* now delete this box */
9.         delete b;
10.     }
11.}

See lec26-recur-tree.cpp

Your turn: Delete the tree

my_tree
Recursion simplifies tasks: searching

- Where is the combination lock?
Recursion simplifies tasks: searching

- Where is the combination lock?
Recursion simplifies tasks: searching

- Where is the combination lock?
Recursion simplifies tasks: searching

• Where is the combination lock?

• Recursive definition of search_lock(image):
  • **Base case:** search_lock(small image) = look at image
  • **Recursive step:** search_lock(big image) = search_lock(half1) or search_lock(half2)
What ideas and skills did we learn today?

• How to delete recursive data structures
  • With a recursive function
• Data structure with single pointer: linked list
• Data structure with two pointers: tree
• How recursion can help break down bigger problems
Week 9 nearly done!

- 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 (due Sunday, March 8)

See you Monday!