Programming Abstractions

CS106B

Cynthia Bailey Lee
Julie Zelenski

Today's topics:

- Recursion ~~Week~~ Fortnight continues!
- Today:
  - Loops + recursion for *generating sequences and combinations*
- Upcoming:
  - Loops + recursion for *recursive backtracking*

Heads or Tails?

GENERATING SEQUENCES

Heads or Tails?

- You flip a coin 5 times
- What are all the possible heads/tails sequences you could observe?
  - TTTTT
  - HHHHH
  - THTHT
  - HHHHT
  - etc…
- We want to write a program to fill a Vector with strings representing each of the possible sequences.

Generating all possible coin flip sequences


```
void generateAllSequences(int length, Vector<string>& allSequences)
{
    string sequence;
    generateAllSequences(length, allSequences, sequence);
}
```

```
void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

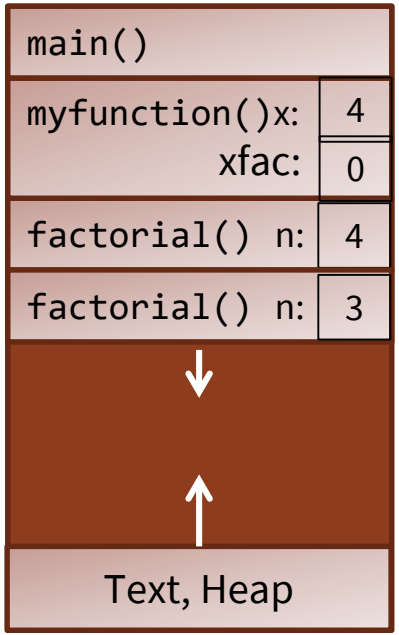
Your Turn: coin flip sequences


```
void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

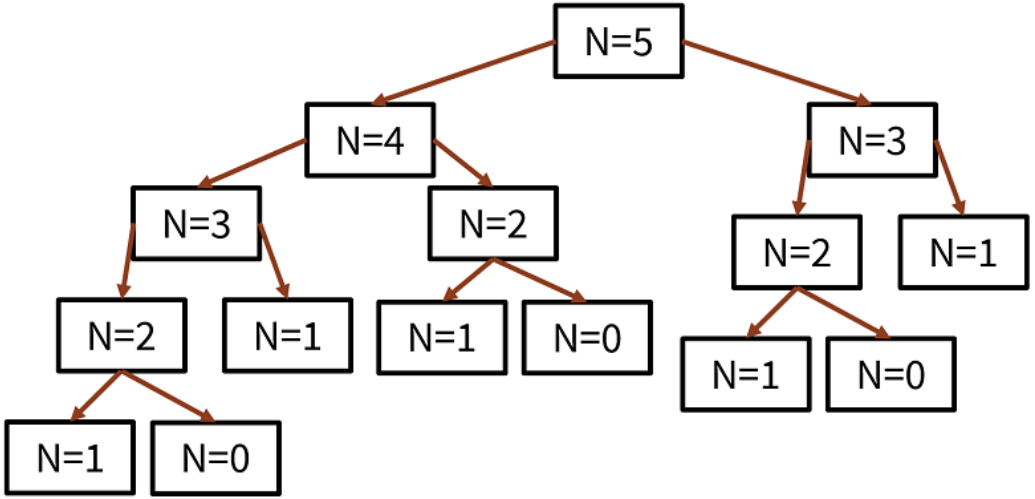
- **Q: Of these sequences (all of which should be included in `allSequences`), which sequence appears first in `allSequences`? Last?**
  - TTTTT, HHHHH, THTHT, HHHHT

Helpful mental models for recursion: the call stack, and the call tree

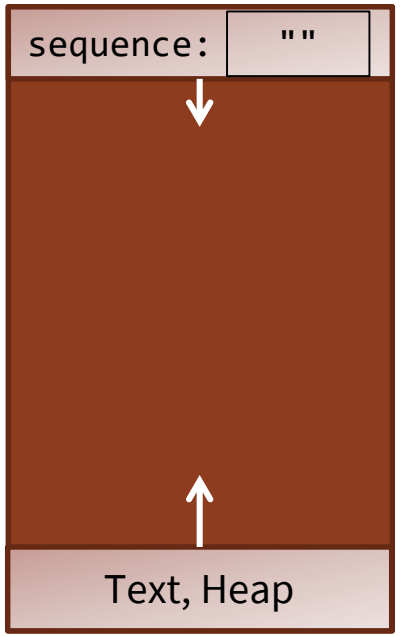
Remember we used this to help us understand **Factorial** recursion:



Remember we used this to help us understand **Fibonacci** recursion:



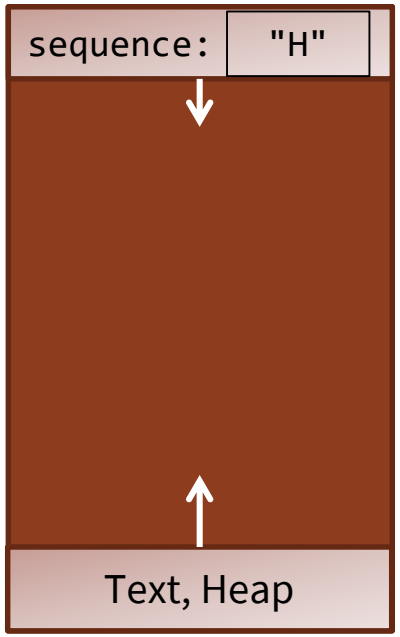
Call stack for our Heads/Tails code



Recursive code

```
void generateAllSequences(int length, Vector<string>&
allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

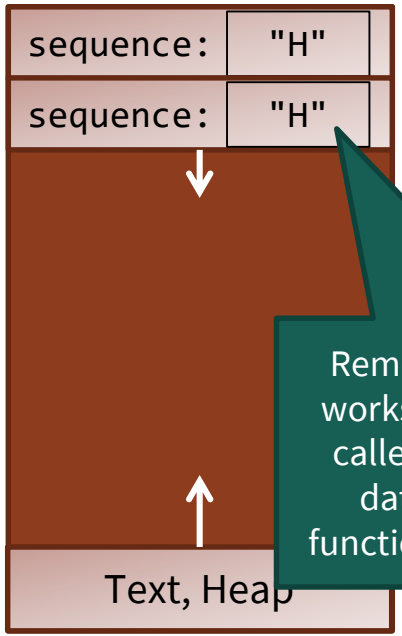
Call stack for our Heads/Tails code



Recursive code

```
void generateAllSequences(int length, Vector<string>&
allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

Call stack for our Heads/Tails code



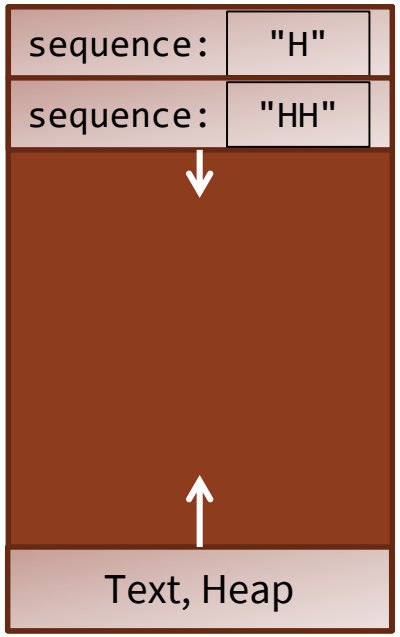
Remember how pass-by-value works is that the function being called gets its own copy of the data, so we start with both functions having the same string.

Recursive code

```
void generateAllSequences(int length, Vector<string>&
allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;

    // cases: add H or T and continue
    "H";
    equences(length, allSequences, sequence);
    se(sequence.size() - 1);
    "T";
    generateAllSequences(length, allSequences, sequence);
}
```

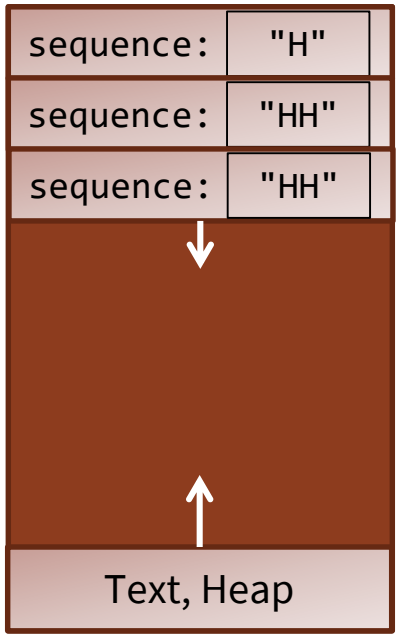
Call stack for our Heads/Tails code



Recursive code

```
void generateAllSequences(int length, Vector<string>&
allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

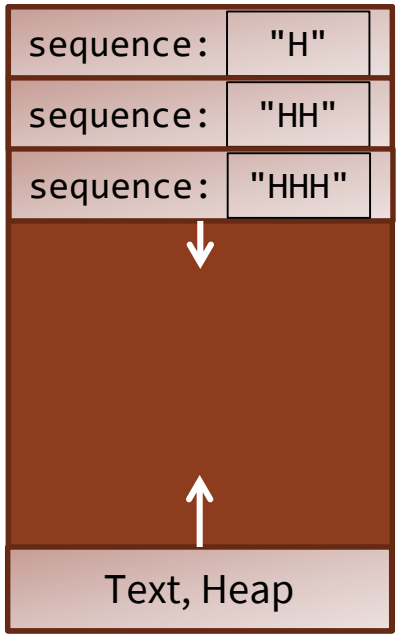
Call stack for our Heads/Tails code



Recursive code

```
void generateAllSequences(int length, Vector<string>&
allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

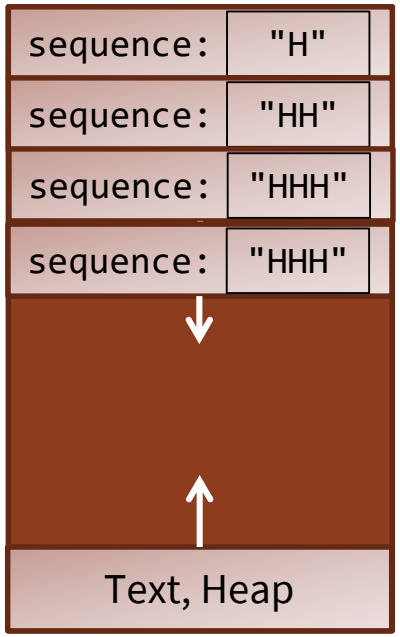
Call stack for our Heads/Tails code



Recursive code

```
void generateAllSequences(int length, Vector<string>&
allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

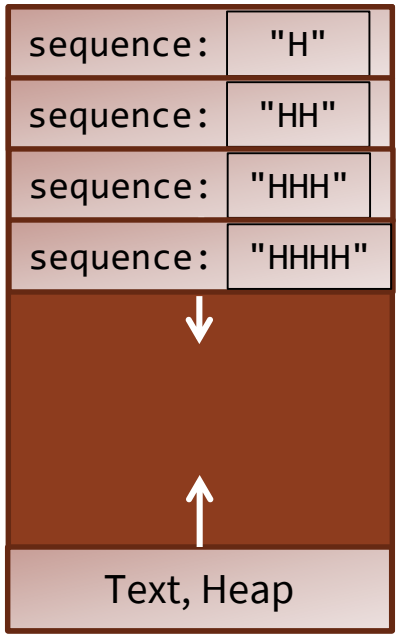
Call stack for our Heads/Tails code



Recursive code

```
void generateAllSequences(int length, Vector<string>&
allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

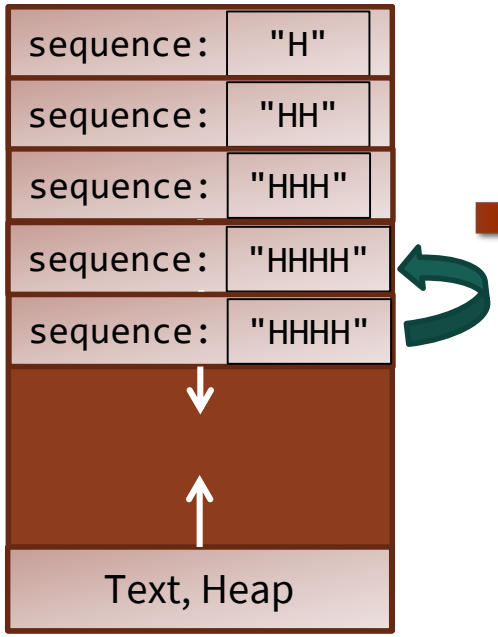
Call stack for our Heads/Tails code



Recursive code

```
void generateAllSequences(int length, Vector<string>&
allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

Call stack for our Heads/Tails code

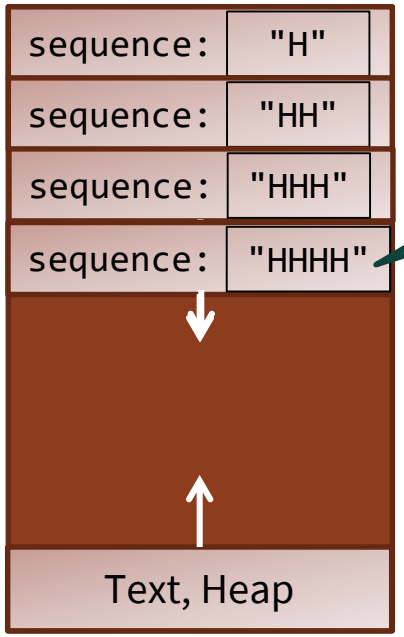


Recursive code

```
void generateAllSequences(int length,
allSequences, string sequence)
{
    // base case: this sequence is fu                dd
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

Finally hit base case! Add this sequence HHHH to our list of possible coin toss sequences of length 4, and return.

Call stack for our Heads/Tails code



Most recent stack frame "popped" off the stack when we returned.

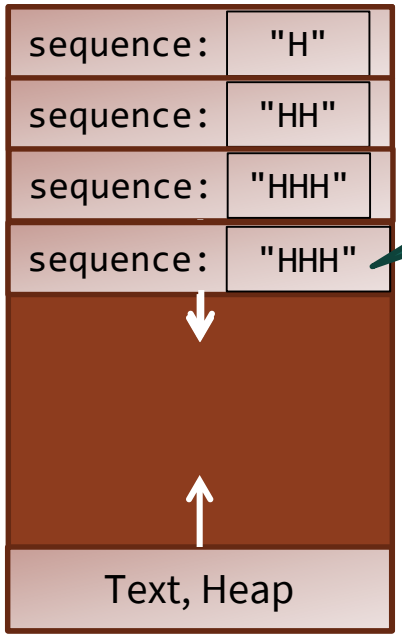
Recursive code

```
llSequences(int length, Vector<string>&
 string sequence)

se: this sequence is full-le
if (sequence.size() == length) {
    allSequences.add(sequence);
    return;
}
// recursive cases: add H or T and cor
sequence += "H";
generateAllSequences(length, allSequen  , sequence);
sequence.erase(sequence.size() - 1);
sequence += "T";
generateAllSequences(length, allSequences, sequence);
}
```

We come back to this next line that says to erase the H we added before the function call.

Call stack for our Heads/Tails code



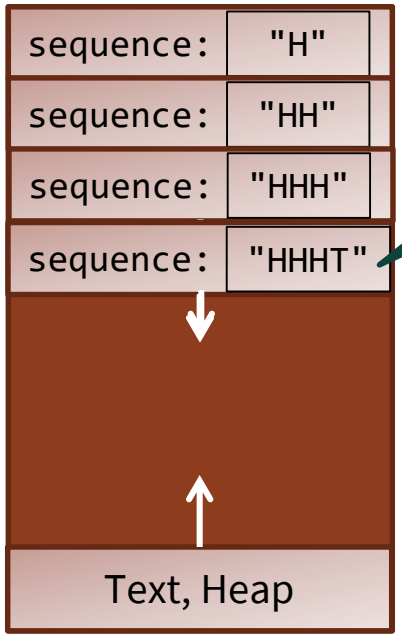
Erased the 4th H

Recursive code

```
void generateAllSequences(int length, Vector<string>&
                          , string sequence)
{
    // base case: this sequence is full-le
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and cor
    sequence += "H";
    generateAllSequences(length, allSequen     , sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

We come back to this next line that says to erase the H we added before the function call.

Call stack for our Heads/Tails code



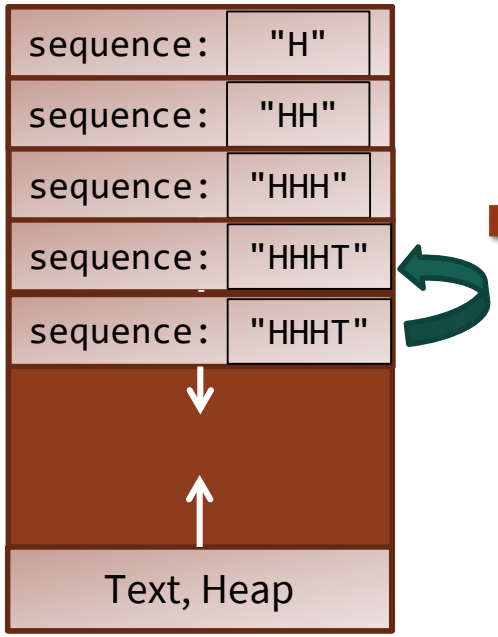
Added a T

Recursive code

```
void generateAllSequences(int length, Vector<string>&
                          quences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(len                    uence);
    sequence.erase(sequence
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

Now a second recursive call with this new sequence.

Call stack for our Heads/Tails code

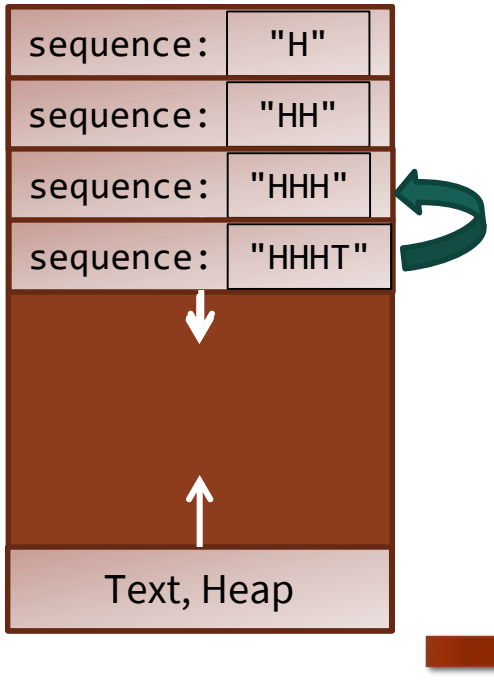


Recursive code

```
void generateAllSequences(int length, V
allSequences, string sequence)
{
    // base case: this sequence is full le                    ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

At the base case again, we add HHHT to our collection as the second completed sequence, and return.

Call stack for our Heads/Tails code

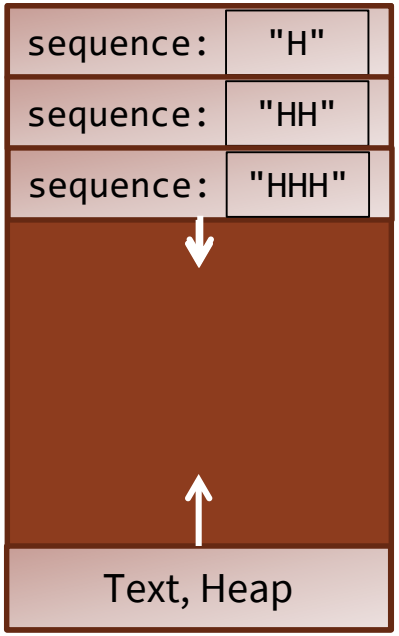


Recursive code

```
void generateAllSequences(int length, Vector<string>&
allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T
    sequence += "H";
    generateAllSequences(length, a
    sequence.erase(sequence.size()
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

This function call has reached the end (did both recursive calls), so it is done and it returns.

Call stack for our Heads/Tails code

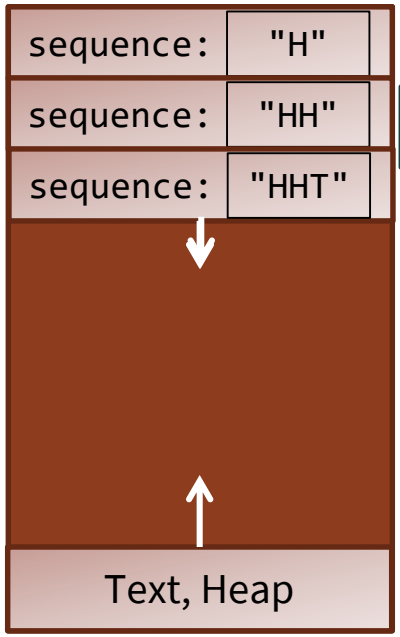


Recursive code

```
void generateAllSequences(int length, Vector<string>&
allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

This function call now needs to erase its H and add a T and try again.

Call stack for our Heads/Tails code



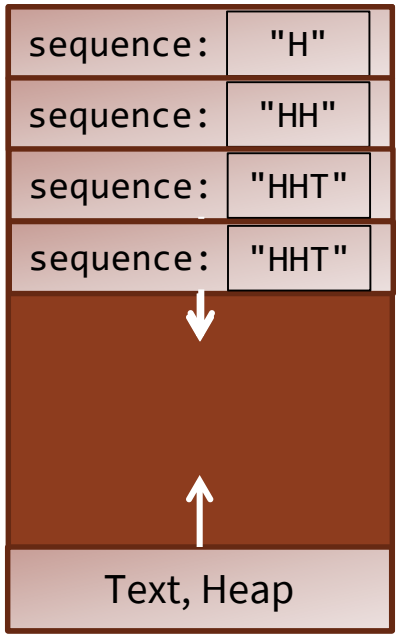
Now ends in T.

Recursive code

```
void generateAllSequences(int length, Vector<string>&
                          es, string sequence)

    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

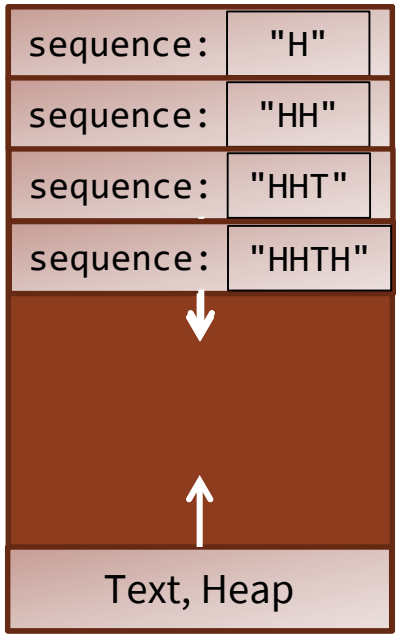
Call stack for our Heads/Tails code



Recursive code

```
void generateAllSequences(int length, Vector<string>&
allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

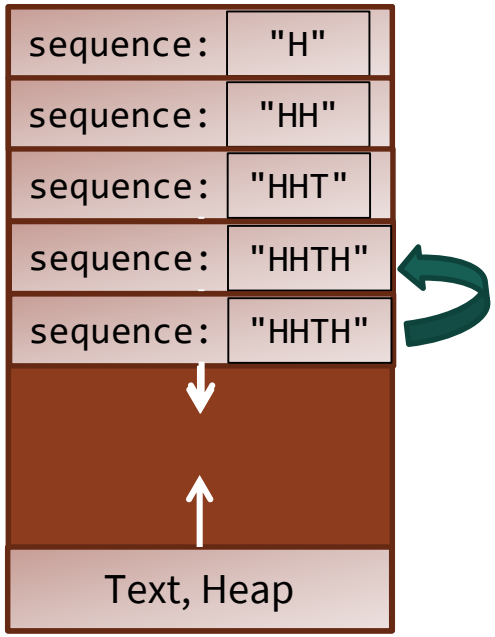
Call stack for our Heads/Tails code



Recursive code

```
void generateAllSequences(int length, Vector<string>&
allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

Call stack for our Heads/Tails code

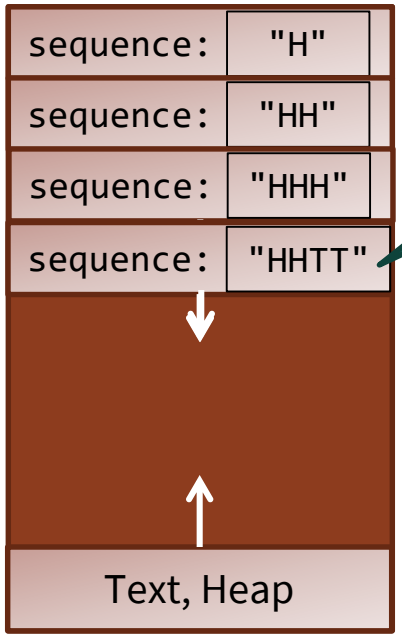


Recursive code

```
void generateAllSequences(int length, V
allSequences, string sequence)
{
    // base case: this sequence is full
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

At the base case again, we add HHTH to our collection as the third completed sequence, and return.

Call stack for our Heads/Tails code



Added a T

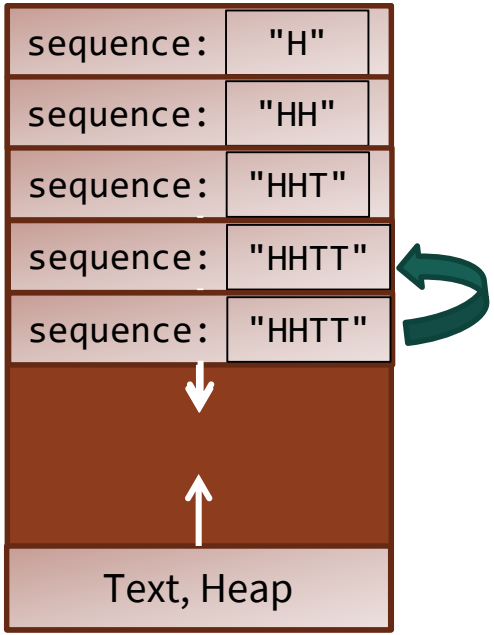
Recursive code

```
void generateAllSequences(int length, Vector<string>&
quences, string sequence)

// base case: this sequence is full-length and ready to add
if (sequence.size() == length) {
    allSequences.add(sequence);
    return;
}
// recursive cases: add
sequence += "H";
generateAllSequences(len
sequence.erase(sequenc
sequence += "T";
generateAllSequences(length, allSequences, sequence);
}
```

Came back here, erased the ending H, added a T, and ready to do second recursive call.

Call stack for our Heads/Tails code

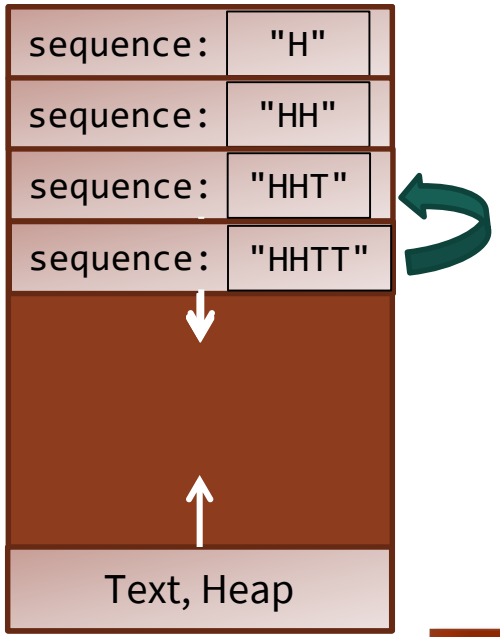


Recursive code

```
void generateAllSequences(int length, V
allSequences, string sequence)
{
    // base case: this sequence is full le          d ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

At the base case again, we add HHTT to our collection as the fourth completed sequence, and return.

Call stack for our Heads/Tails code

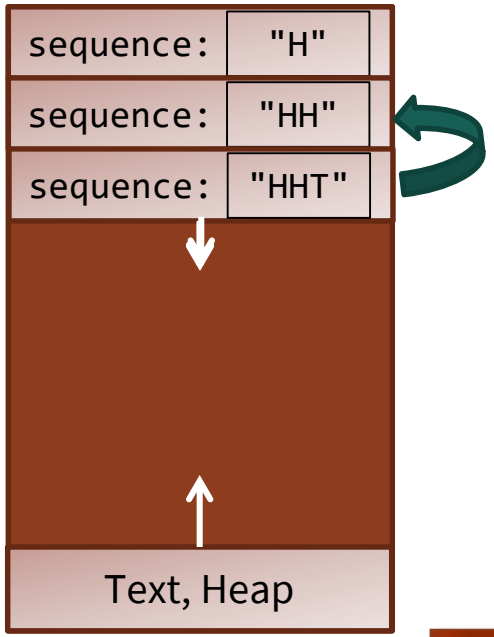


Recursive code

```
void generateAllSequences(int length, Vector<string>&
allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(s
        return;
    }
    // recursive cases: ad                    ue
    sequence += "H";
    generateAllSequences(l                    , sequence);
    sequence.erase(sequence.                  1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

We did both H and T recursive calls, so this function is done and also returns.

Call stack for our Heads/Tails code

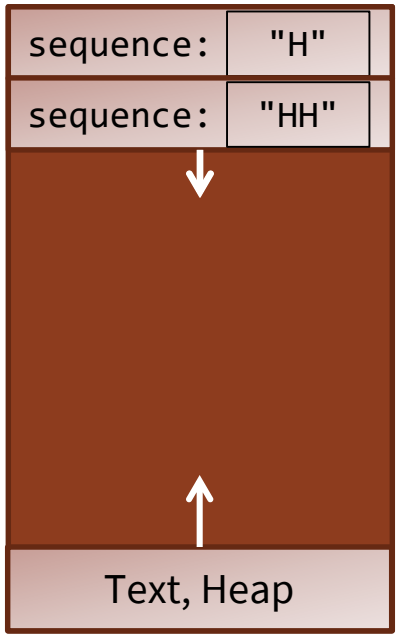


Recursive code

```
void generateAllSequences(int length, Vector<string>&
allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(s
        return;
    }
    // recursive cases: ad                    ue
    sequence += "H";
    generateAllSequences(l                    , sequence);
    sequence.erase(sequence.                  1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

We did both H and T recursive calls, so this function is done and also returns.

Call stack for our Heads/Tails code



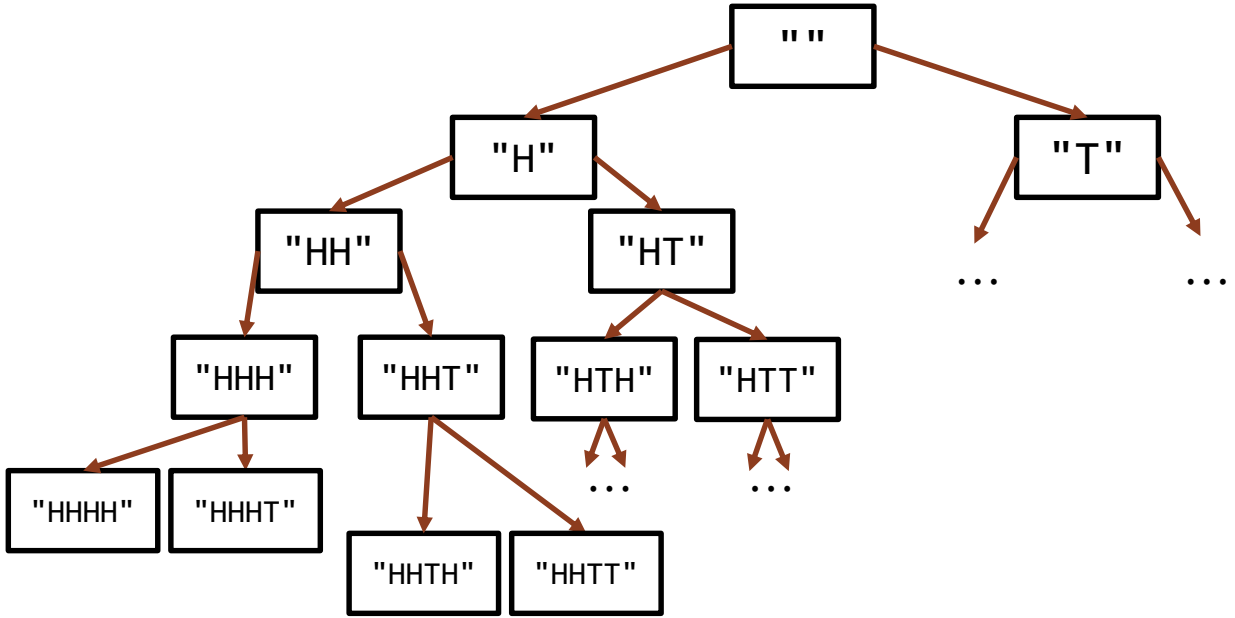
Recursive code

```
void generateAllSequences(int length, Vector<string>&
allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and co
    sequence += "H";
    generateAllSequences(length, allSequ
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

This function still needs to erase its H, and try T, but we'll end our animation here. ☺

Call tree for Heads/Tails code

Labels are based on the value of the **sequence** parameter at the time of the function call (without add/erase/add edits we make inside the function).



Your Turn: coin flip sequences


```
void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

- **Q: What would happen if we didn't do the erase (highlighted above)? Which of the following sequences would we NOT generate? Which additional sequences would we generate (that we shouldn't)?**
  - TTTTT, HHHHH, THTHT, HHHHT

Roll the Dice!

GENERATING MORE SEQUENCES

Roll the Dice!

- You roll a single die 5 times
- What are all the possible 1/2/3/4/5/6 sequences you could observe?
  - 11111
  - 66666
  - 12345
  - 21655
  - etc…
- We want to write a program to fill a Vector with strings representing each of the possible sequences.

Generating all possible ~~coin flip~~ die roll sequences

```
void generateAllSequences(int length, Vector<string>& allSequences)
{
    string sequence;
    generateAllSequences(length, allSequences, sequence);
}

void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

To adapt for die rolls, we need to change this from 2 options (H/T) to 6 options (1-6).

Generating all possible ~~coin flip~~ die roll sequences

```
    // recursive cases: add 1 or 2 or 3 or 4 or 5 or 6 and continue
    sequence += "1";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "2";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "3";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "4";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "5";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "6";
    generateAllSequences(length, allSequences, sequence);
}
```

This works, but YIKES!!
So much copy-paste!!

Generating all possible ~~coin flip~~ die roll sequences

```
    // recursive cases: add 1 or 2 or 3 or 4 or 5 or 6 and continue
    sequence += "1";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "2";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "3";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "4";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "5";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "6";
    generateAllSequences(length, allSequences, sequence);
}
```

Let's take the repeated actions and put them in a for-loop from 1 to 6.

Generating all possible ~~coin flip~~ die roll sequences

```
void generateAllSequences(int length, Vector<string>& allSequences)
{
    string sequence;
    generateAllSequences(length, allSequences, sequence);
}

void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add 1-6 and continue
    for (int i = 1; i <= 6; i++) {
        sequence += integerToString(i);
        generateAllSequences(length, allSequences, sequence);
        sequence.erase(sequence.size() - 1);
    }
}
```

Much nicer!!

Generating all possible ~~coin flip~~ die roll sequences

```
void generateAllSequences(int length, Vector<string>& allSequences)
{
    string sequence;
    generateAllSequences(length, allSequences, sequence);
}

void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add 1-6 and continue
    for (int i = 1; i <= 6; i++) {
        sequence += integerToString(i);
        generateAllSequences(length, allSequences, sequence);
        sequence.erase(sequence.size() - 1);
    }
}
```

Notice that this loop **does not replace** the recursion. It just controls how many times the recursion launches.

Your Turn: die roll sequences

```
void generateAllSequences(int length, Vector<string>& allSequences, string se
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add 1-6 and continue
    for (int i = 1; i <= 6; i++) {
        sequence += integerToString(i);
        generateAllSequences(length, allSequences, sequence);
        sequence.erase(sequence.size() - 1);
    }
}
```

- **Q: Of these sequences (all of which should be included in `allSequences`), which sequence appears first in `allSequences`? Last?**
  - 11111, 66666, 12345, 21655