Recursive Backtracking 2

Amrita Kaur

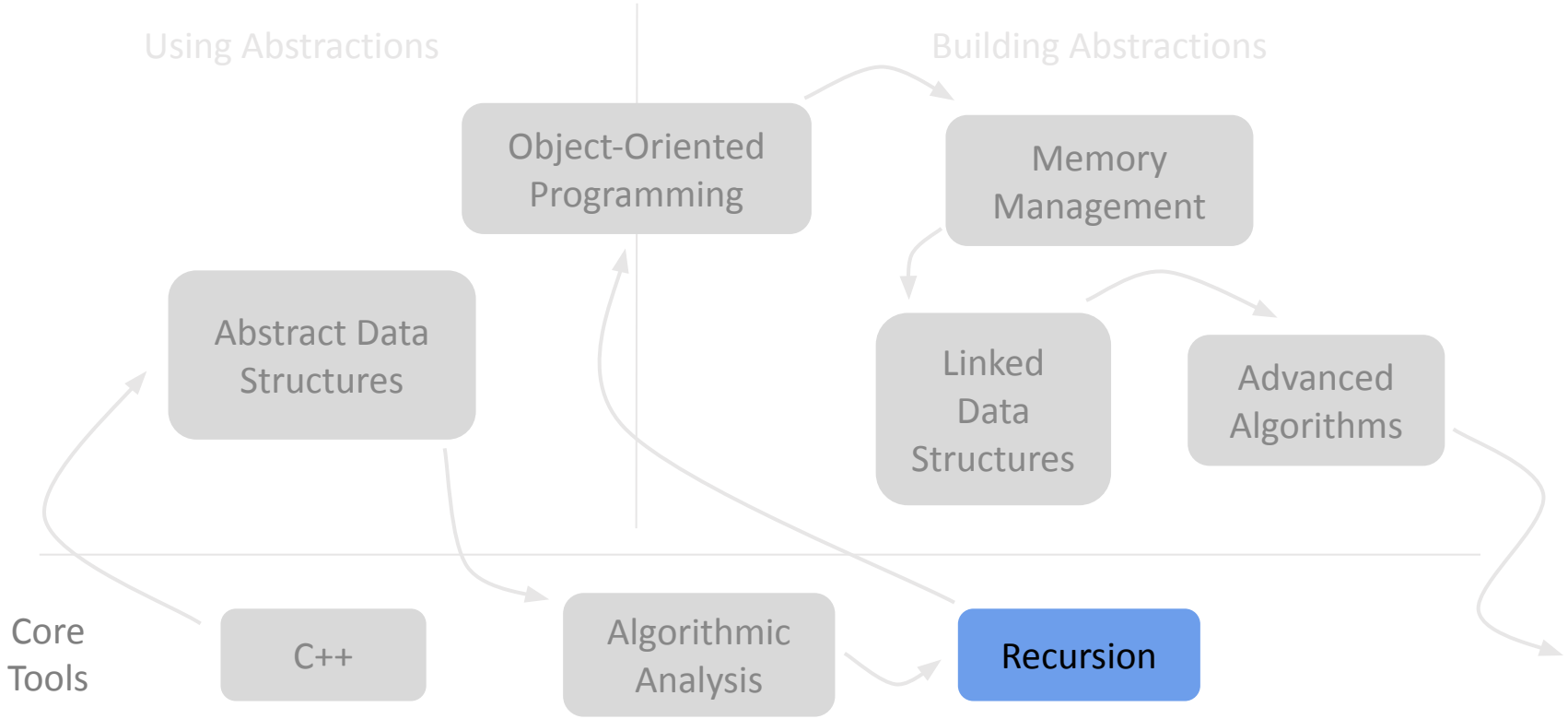
July 19, 2023

Contributions made from previous CS106B Instructors

Announcements

- Assignment 3, Part 2 will be released today after class
  - YEAH Hours are today at 3pm on Zoom

Roadmap


Using Abstractions
Building Abstractions
Object-Oriented Programming
Memory Management
Abstract Data Structures
Linked Data Structures
Advanced Algorithms
Core Tools
C++
Algorithmic Analysis
Recursion

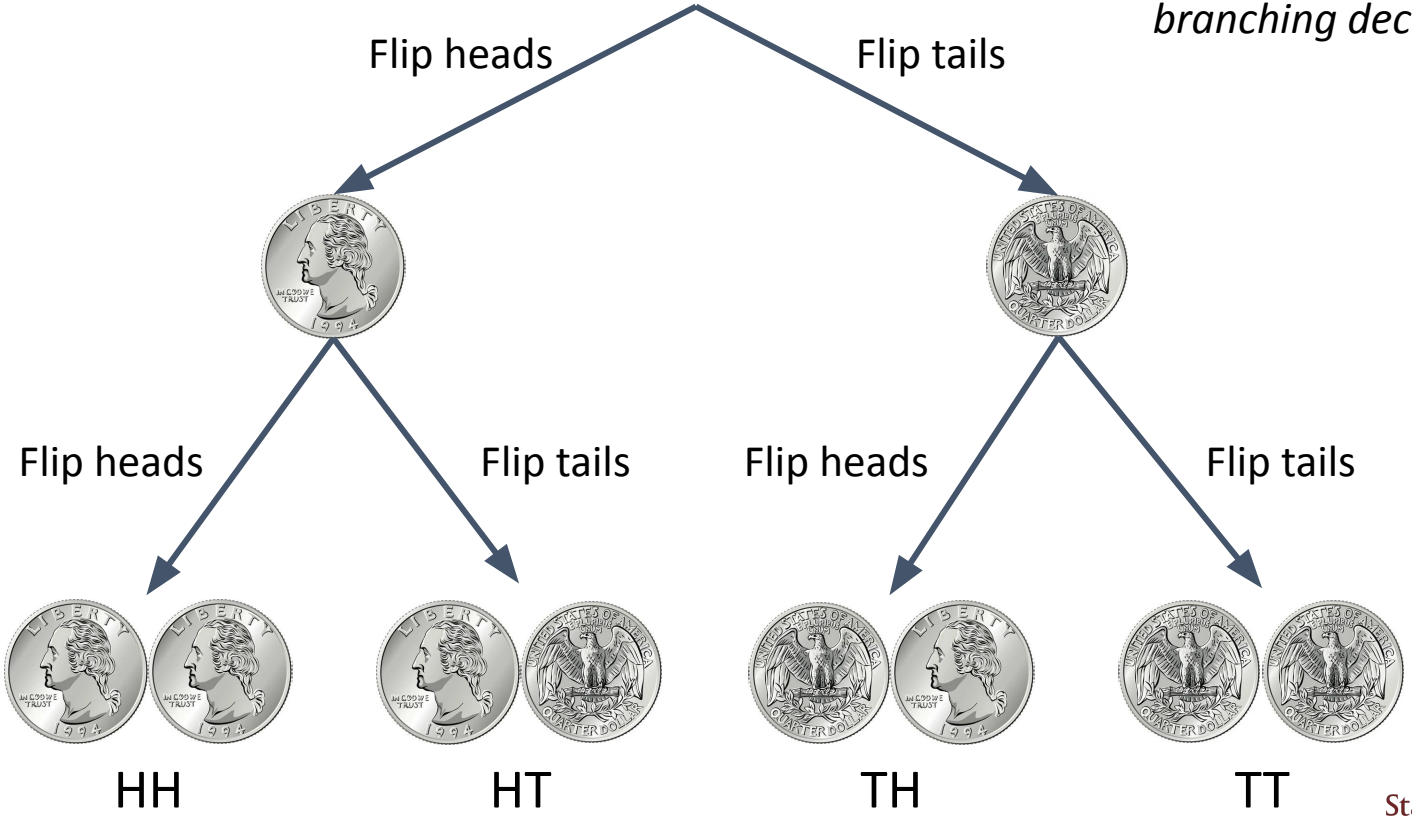
Review

Coin Sequences

- You're playing a (rather boring) game in which you flip some number of coins one by one and see whether you get heads or tails
- You'd like to know all of the possible sequences you might flip

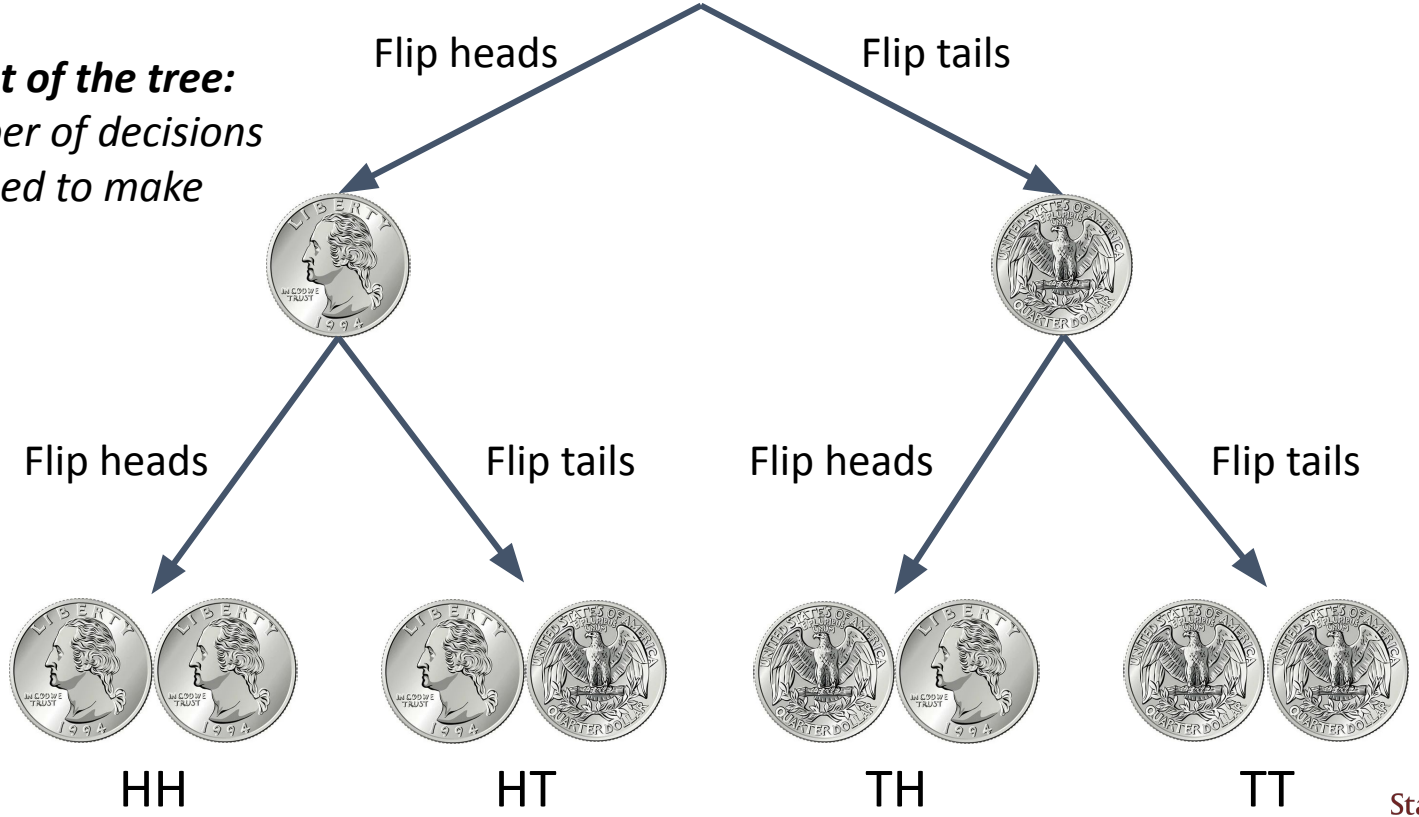
Coin Sequences

*This is called a **decision tree**: at each point we have branching decisions.*



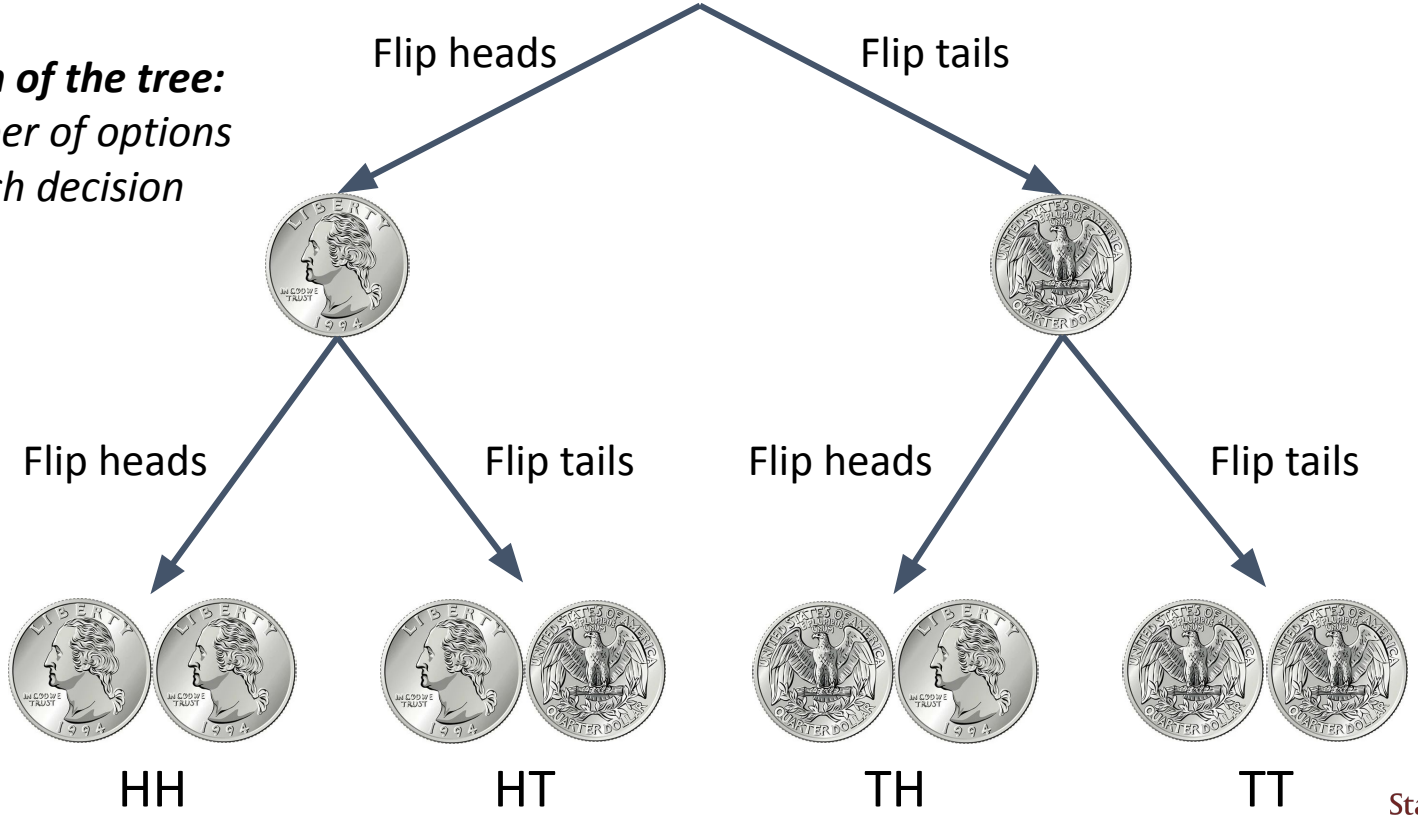
Coin Sequences

Height of the tree:
Number of decisions we need to make



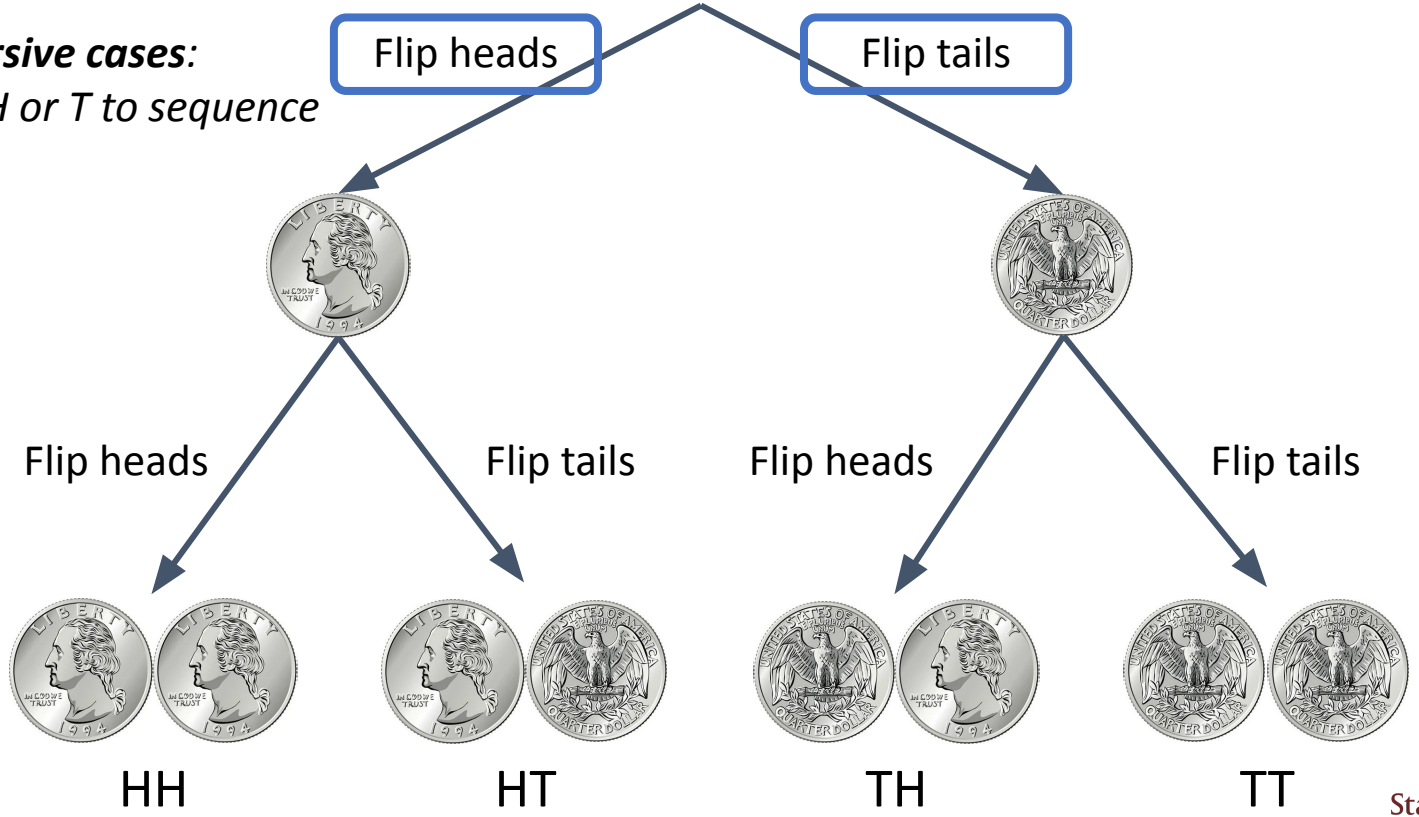
Coin Sequences

Width of the tree:
Number of options at each decision



Coin Sequences

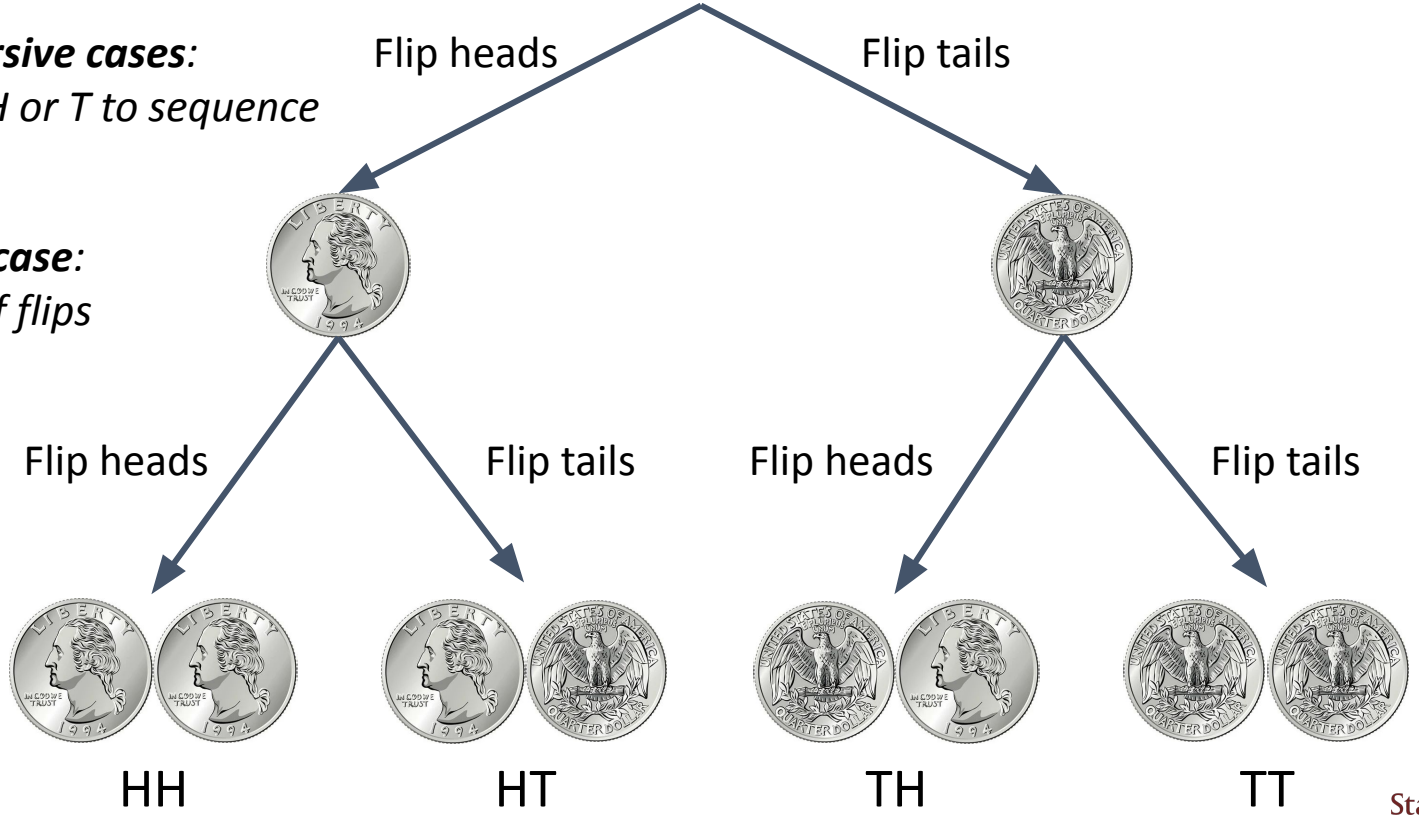
Recursive cases:
Add H or T to sequence



Coin Sequences

Recursive cases:
Add H or T to sequence

Base case:
Out of flips



Number of Flips Remaining

2

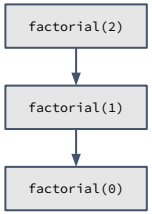
1

0

Two Types of Recursion

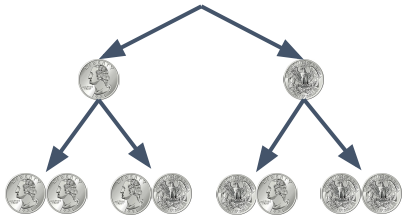
Basic recursion

- One repeated task that builds up a solution as you come back up the call stack
- The final base case defines the initial seed of the solution and each call contributes a little bit to the solution
- Initial call to the recursive function produces the final solution



Backtracking recursion

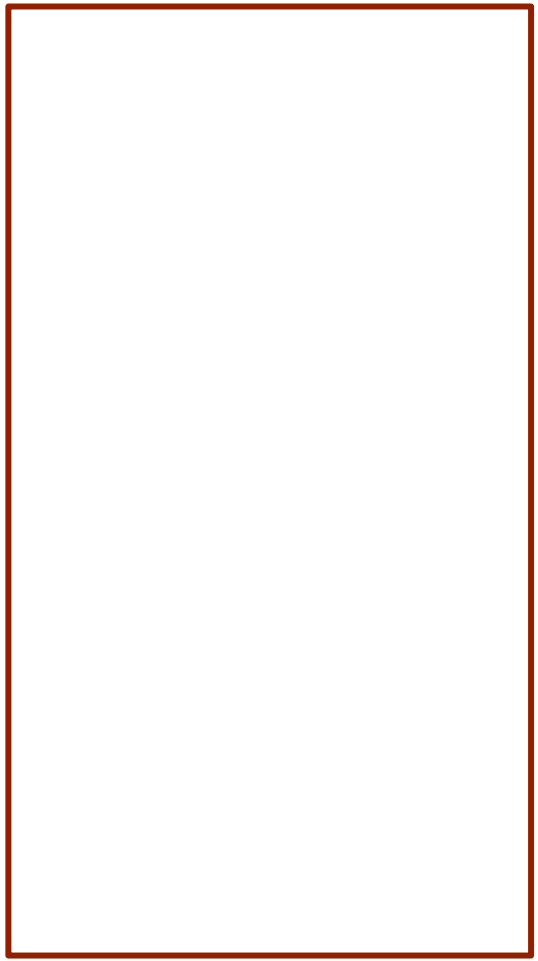
- Build up many possible solutions through multiple recursive calls at each step
- Seed the initial recursive call with an “empty” solution
- At each base case, you have a potential solution



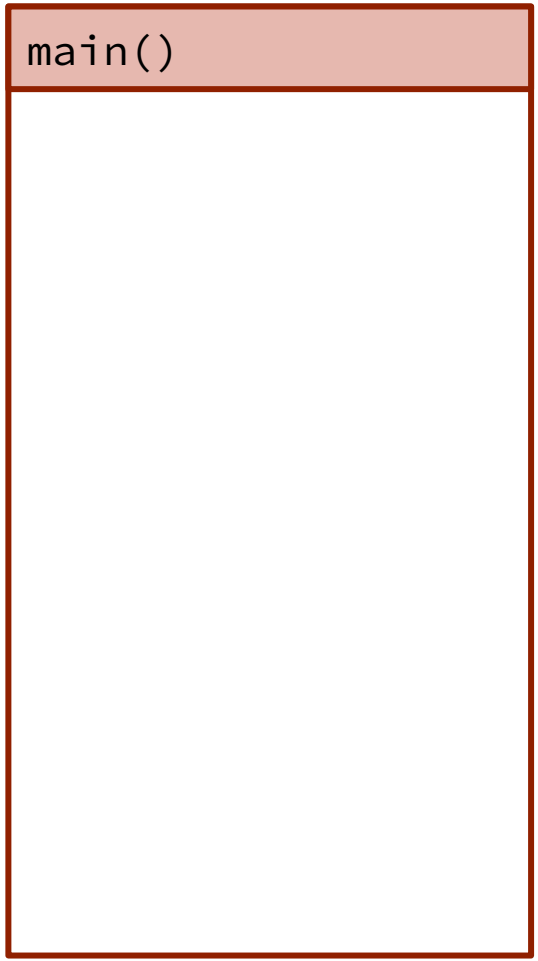
Solution Code for Coin Sequences

```
void generateSequenceHelper(int flipsRemaining, string sequence) {
    // Base case: flipsRemaining = 0, no more flips
    if (flipsRemaining == 0) {
        cout << sequence << endl;
    } else {
        // Recursive cases (when flipsRemaining > 0)
        generateSequenceHelper(flipsRemaining - 1, sequence + 'H'); // Add H to the sequence
        generateSequenceHelper(flipsRemaining - 1, sequence + 'T'); // OR add T to the sequence
    }
}

void generateSequences(int numCoins) {
    generateSequenceHelper(numCoins, "");
}
```

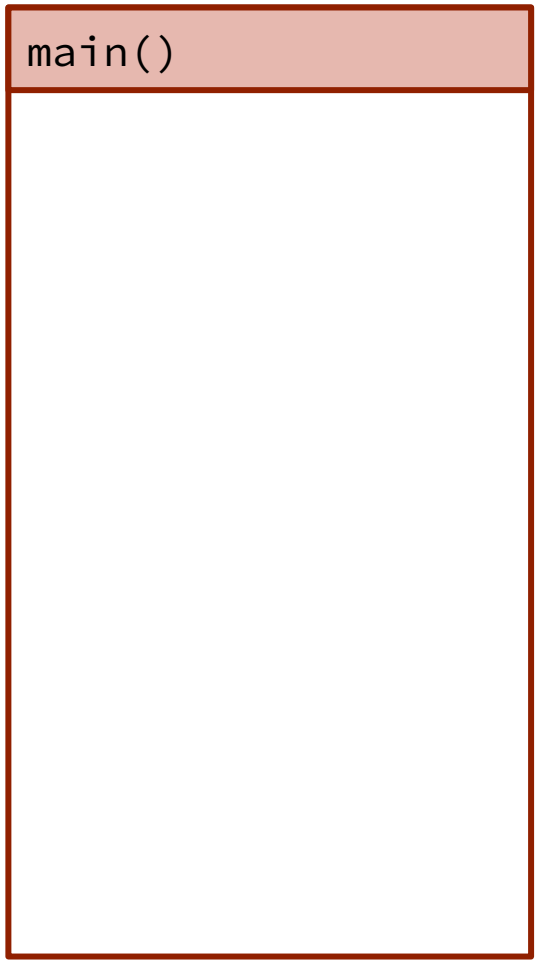
```
int main () {
    generateSequences(3);
    return 0;
}
```



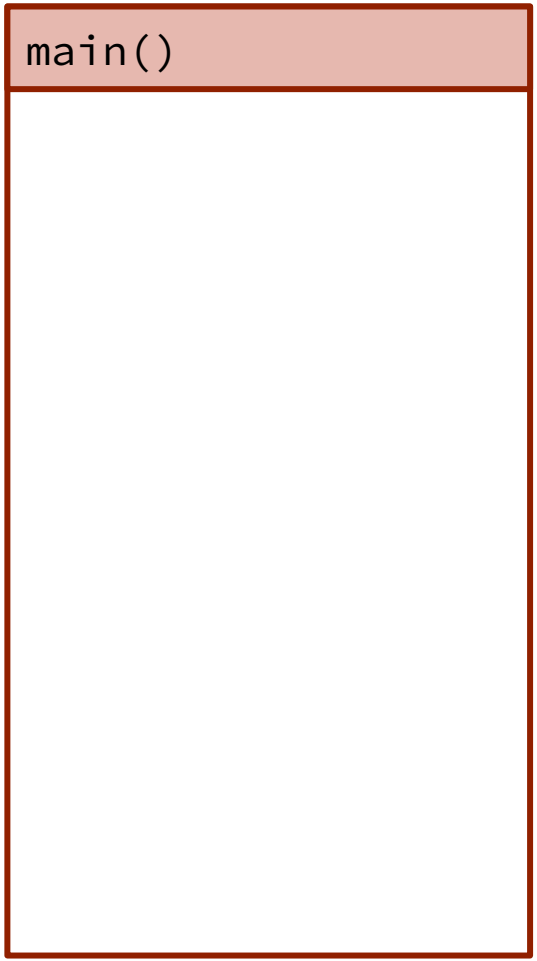
```
int main () {
    generateSequences(3);
    return 0;
}
```



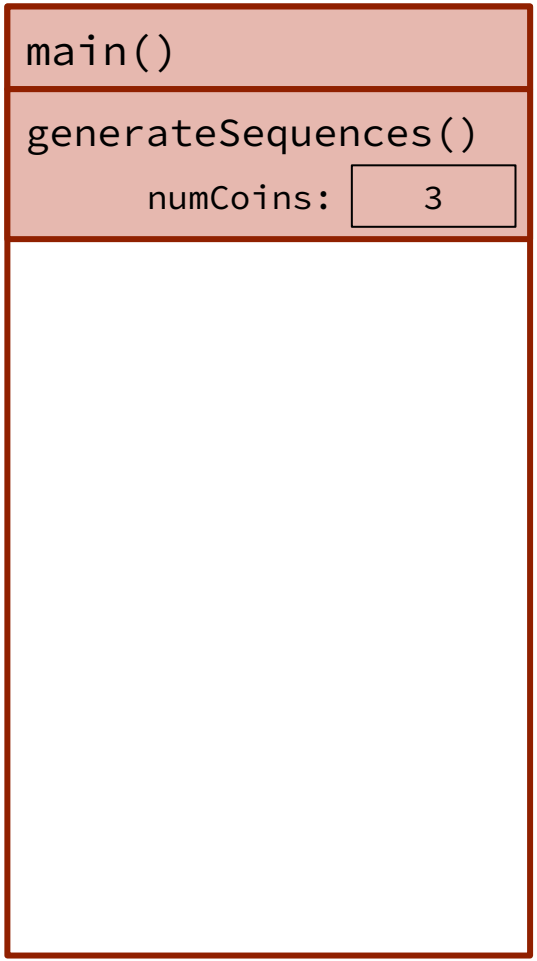
```
int main () {
    generateSequences(3);
    return 0;
}
```



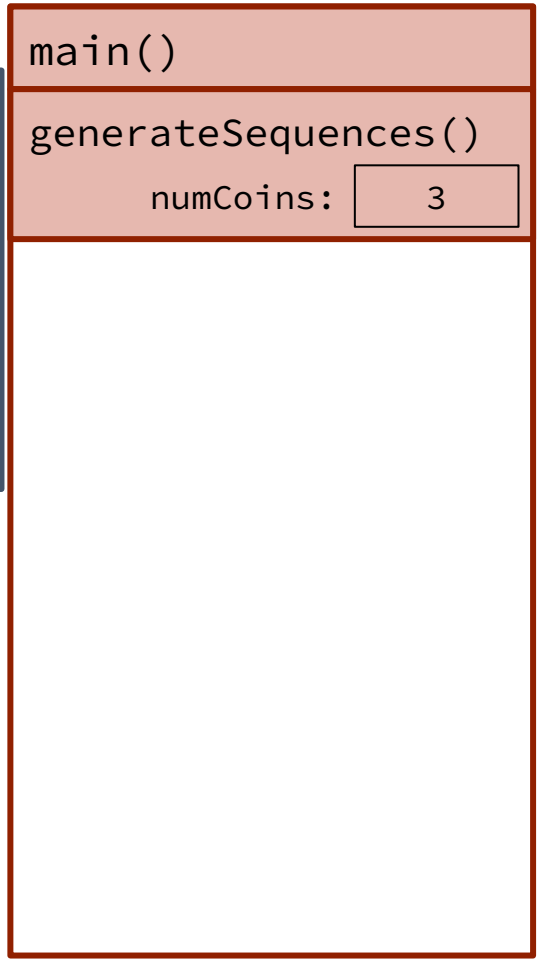
```
void generateSequences (int numCoins) {
    genSeqsHepler(numCoins, “”);
}
```



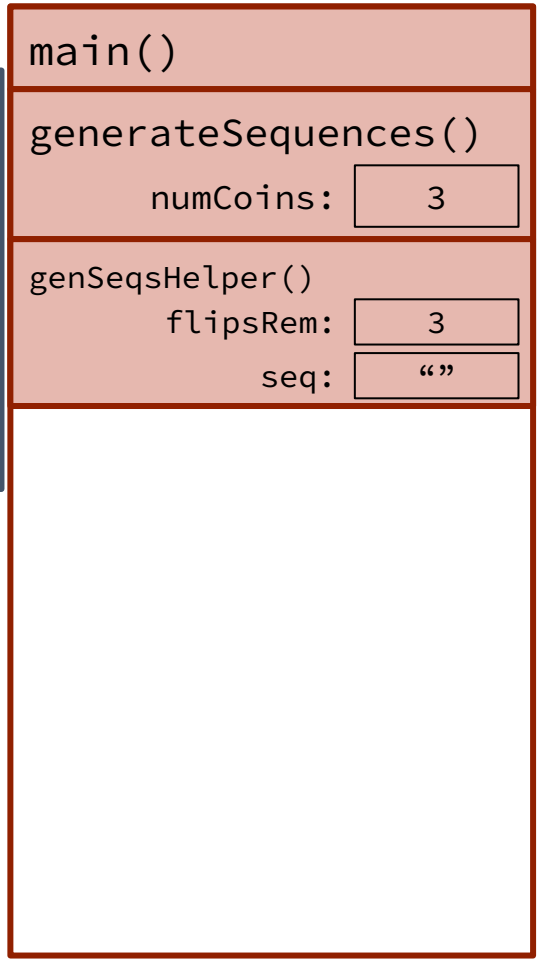
```
void generateSequences (int numCoins) {
    genSeqsHepler(numCoins, “”);
}
```



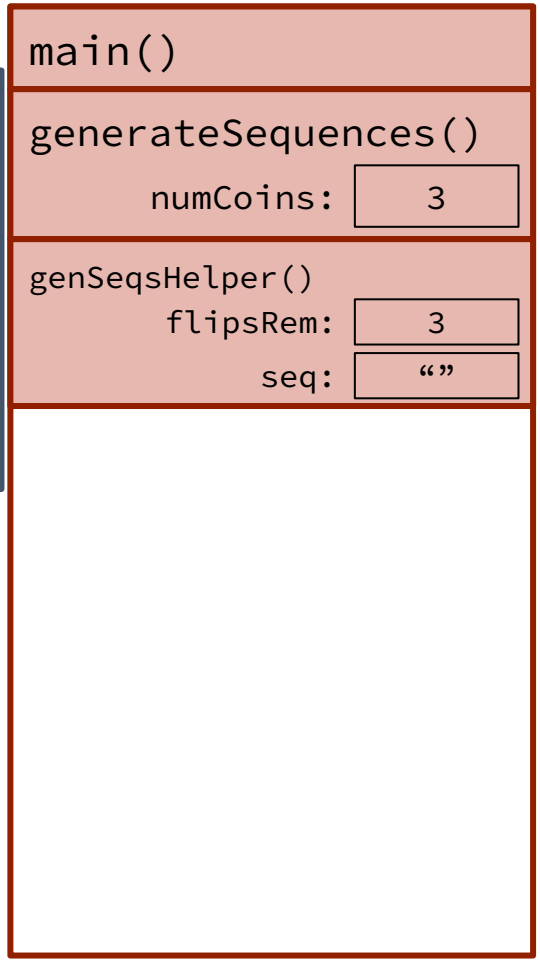
```
void genSeqsHelper (int flipsRem, string seq) {
     if (flipsRem == 0) {
          cout << seq << endl;
     } else {
          generateSeqsHelper(flipsRem-1, seq + 'H');
          generateSeqsHelper(flipsRem-1, seq + 'T');
     }
}
```



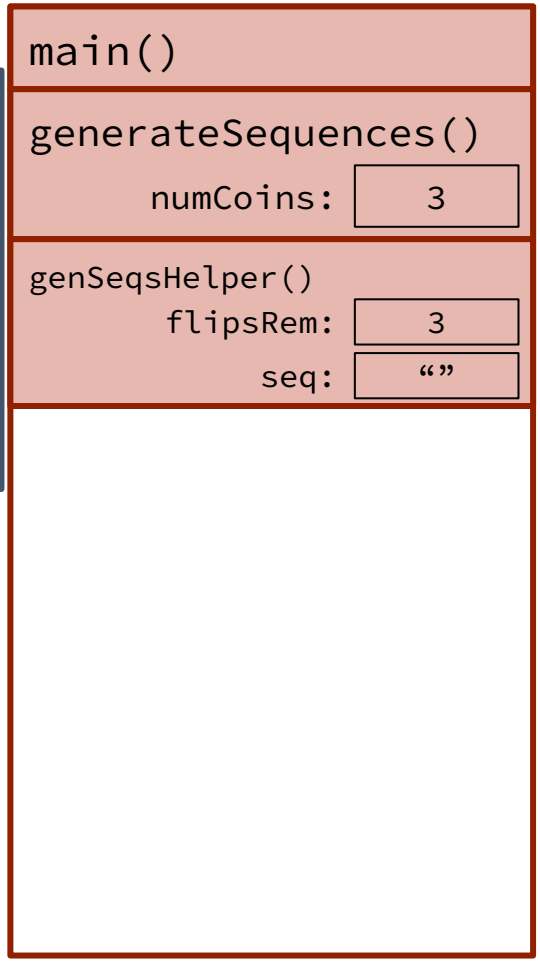
```
void genSeqsHelper (int flipsRem, string seq) {
     if (flipsRem == 0) {
          cout << seq << endl;
     } else {
          generateSeqsHelper(flipsRem-1, seq + 'H');
          generateSeqsHelper(flipsRem-1, seq + 'T');
     }
}
```



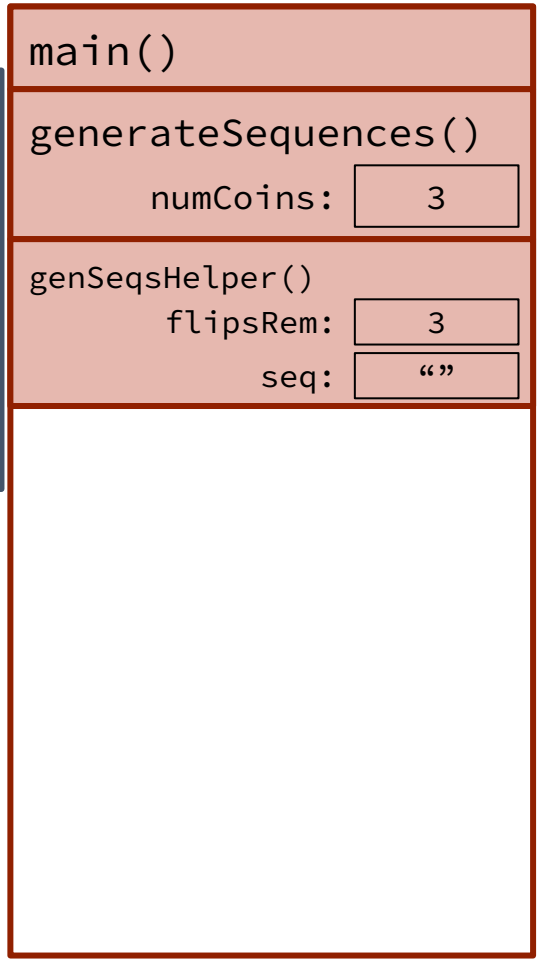
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



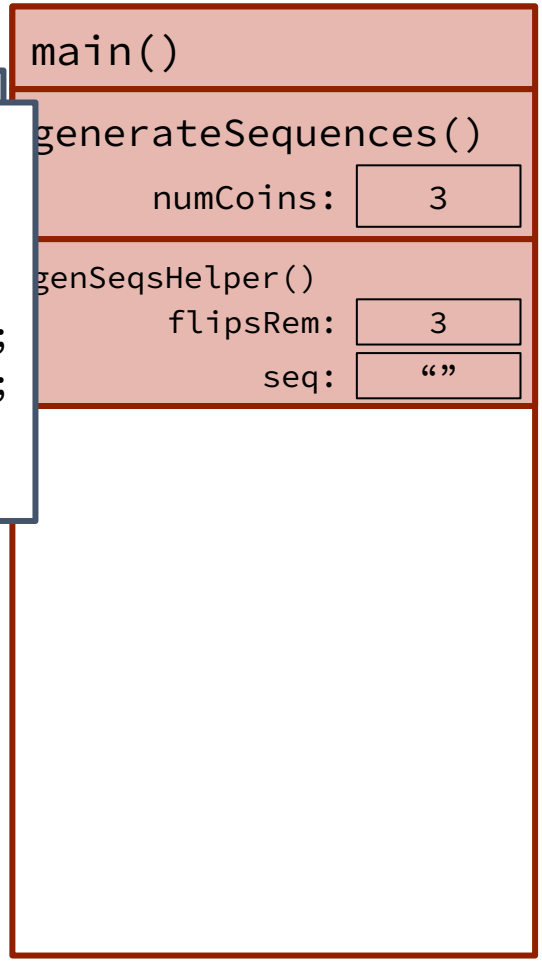
```
void genSeqsHelper (int flipsRem, string seq) {
     if (flipsRem == 0) {
          cout << seq << endl;
     } else {
          generateSeqsHelper(flipsRem-1, seq + 'H');
          generateSeqsHelper(flipsRem-1, seq + 'T');
     }
}
```



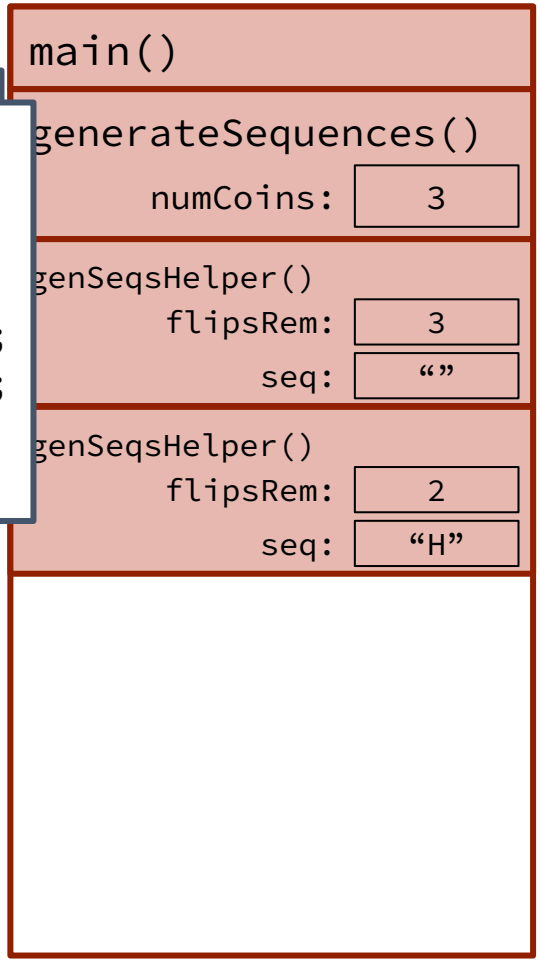
```
void genSeqsHelper (int flipsRem, string seq) {
     if (flipsRem == 0) {
          cout << seq << endl;
     } else {
          generateSeqsHelper(flipsRem-1, seq + 'H');
          generateSeqsHelper(flipsRem-1, seq + 'T');
     }
}
```



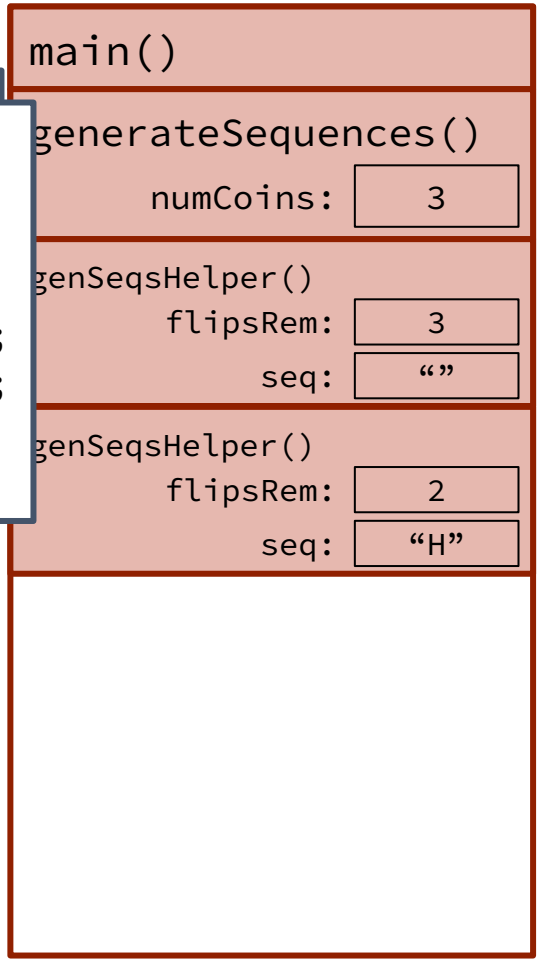
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



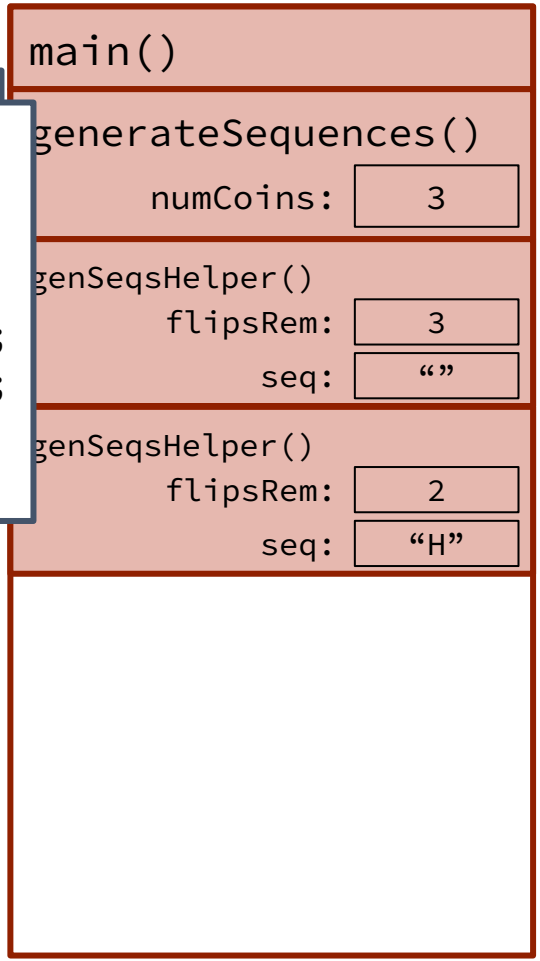
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



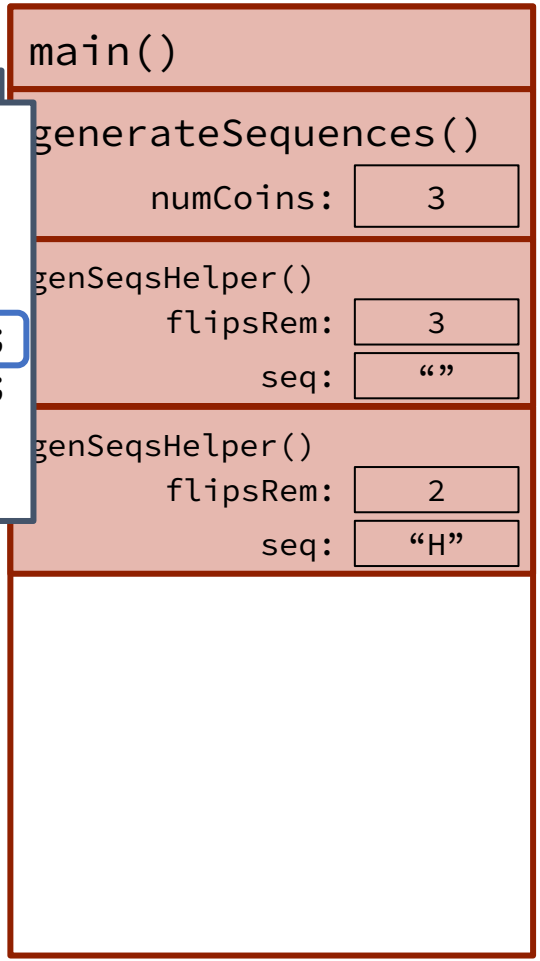
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



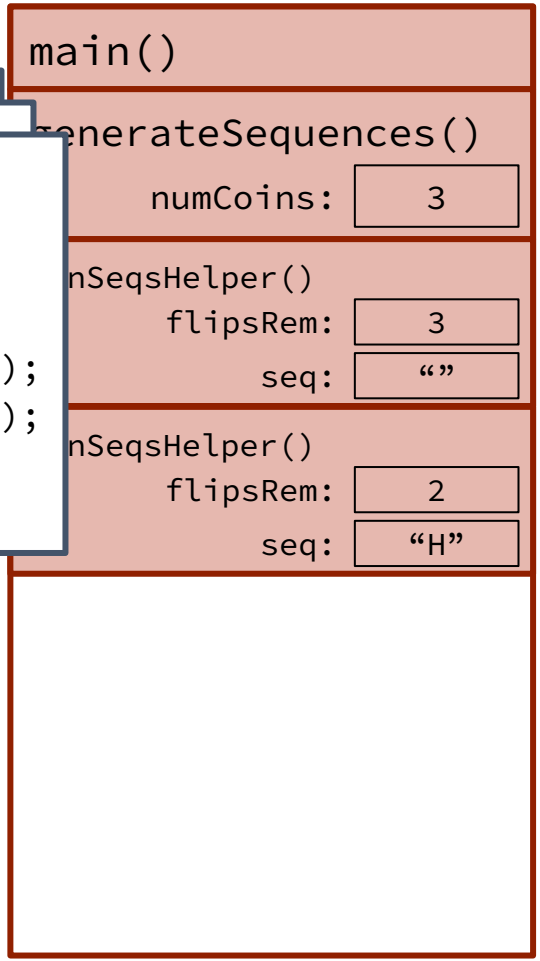
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



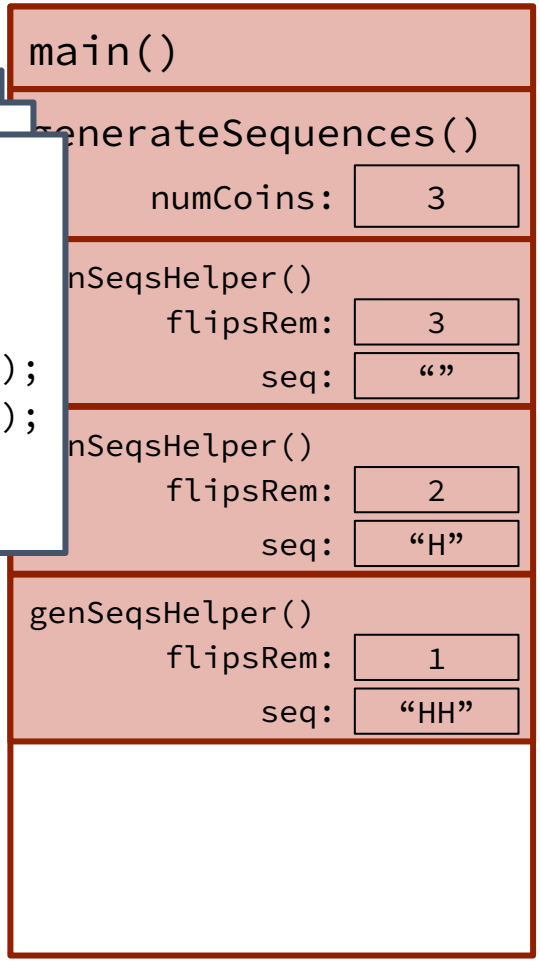
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



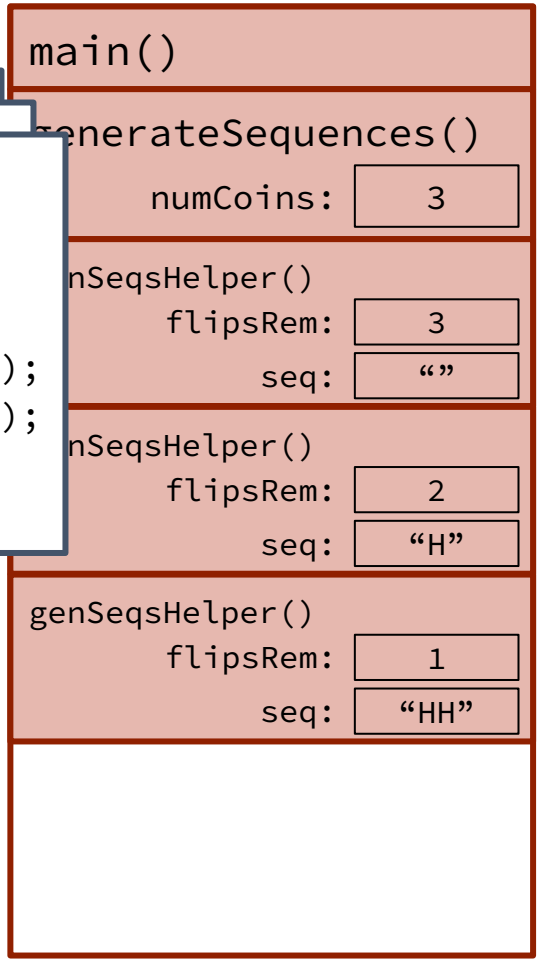
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



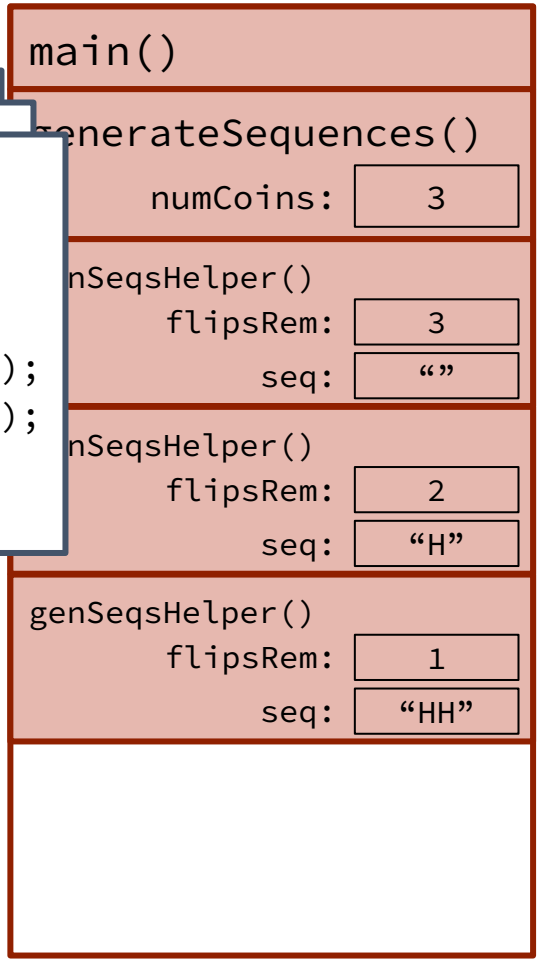
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



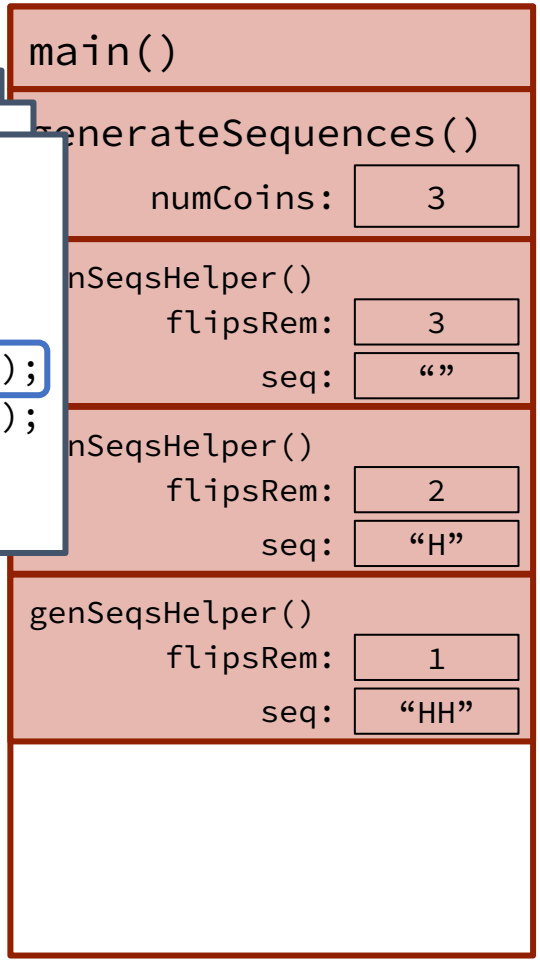
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



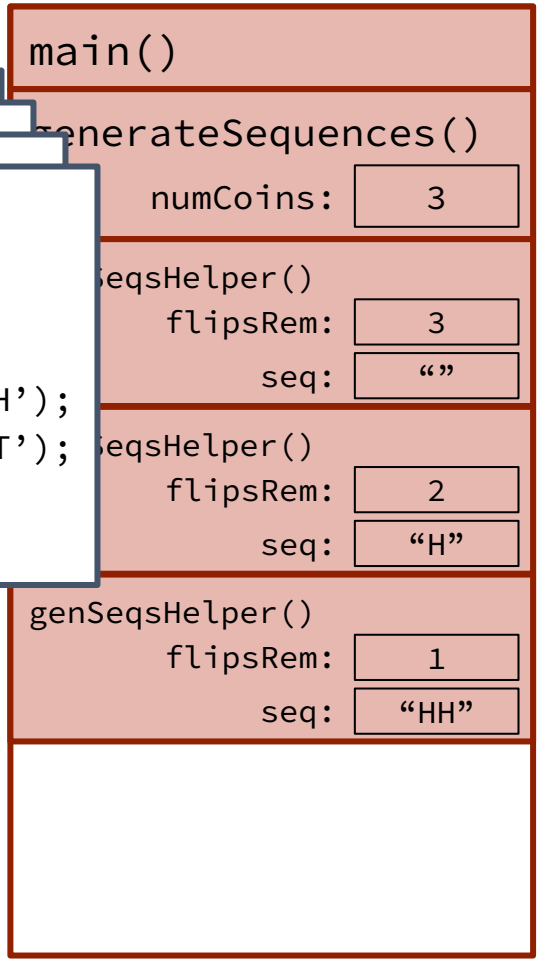
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



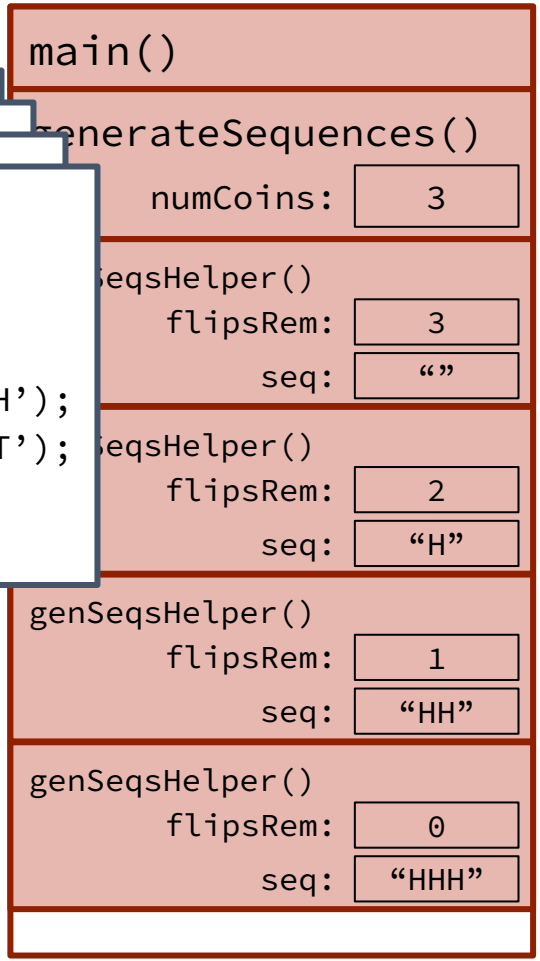
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



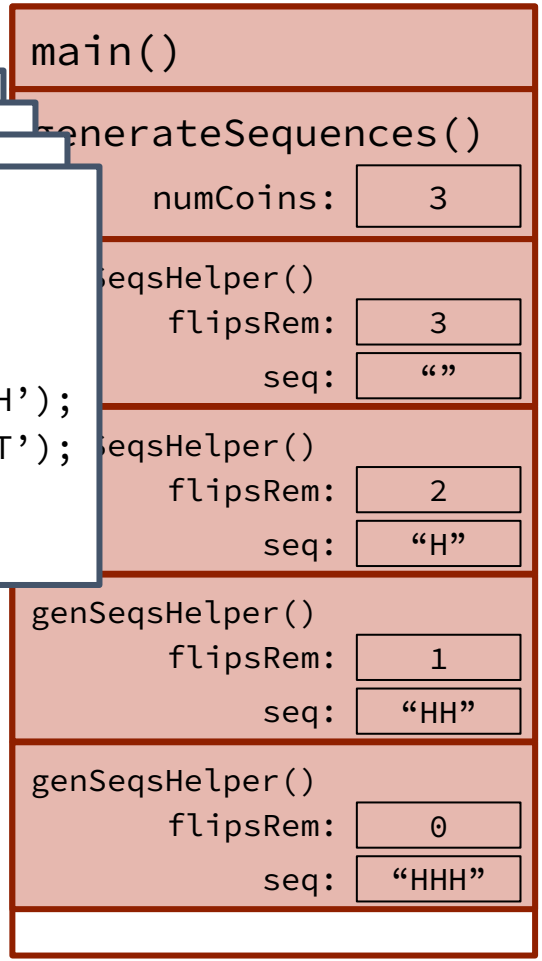
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



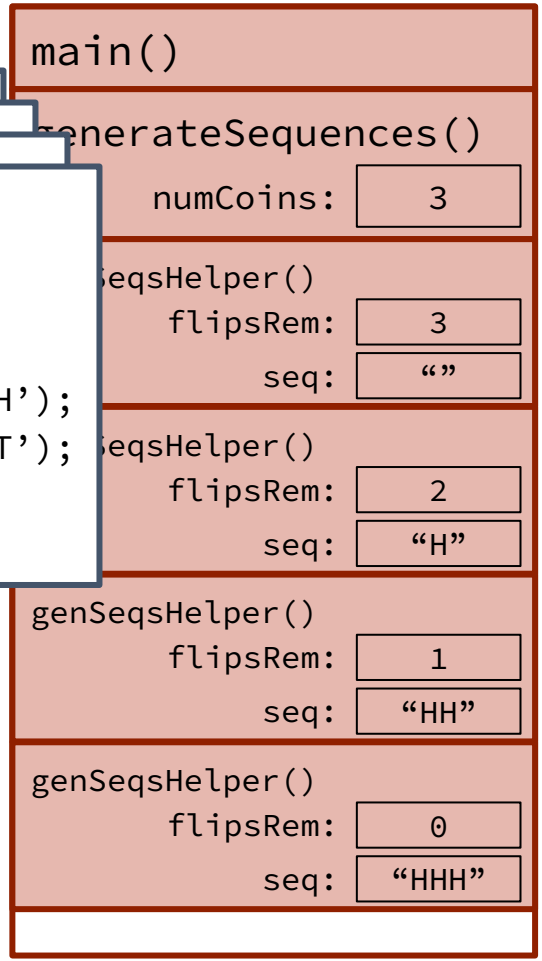
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



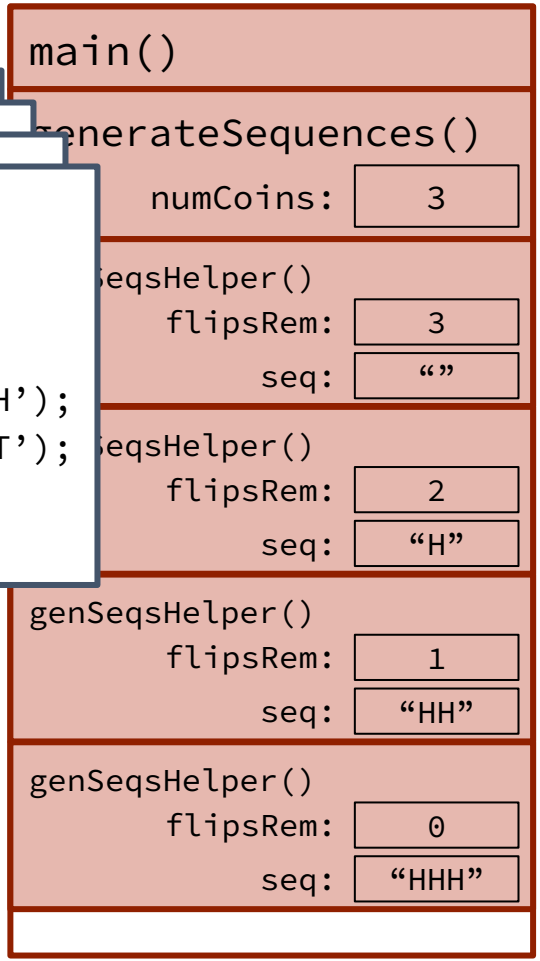
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

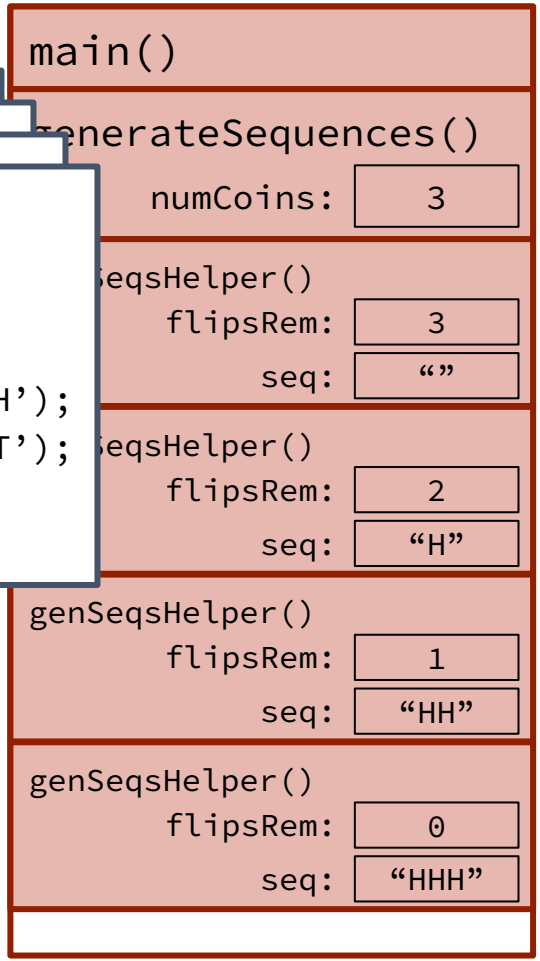
```
HHH
```



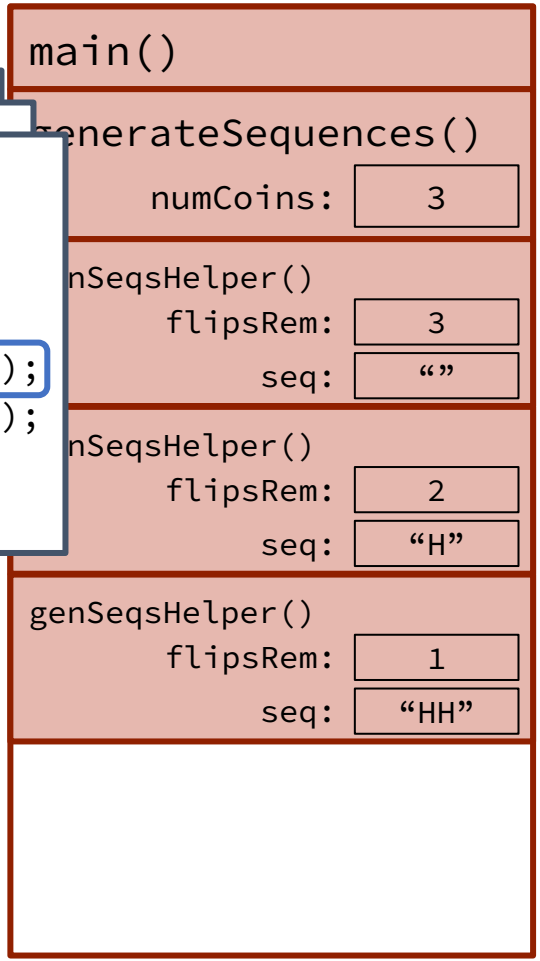
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

```
HHH
```



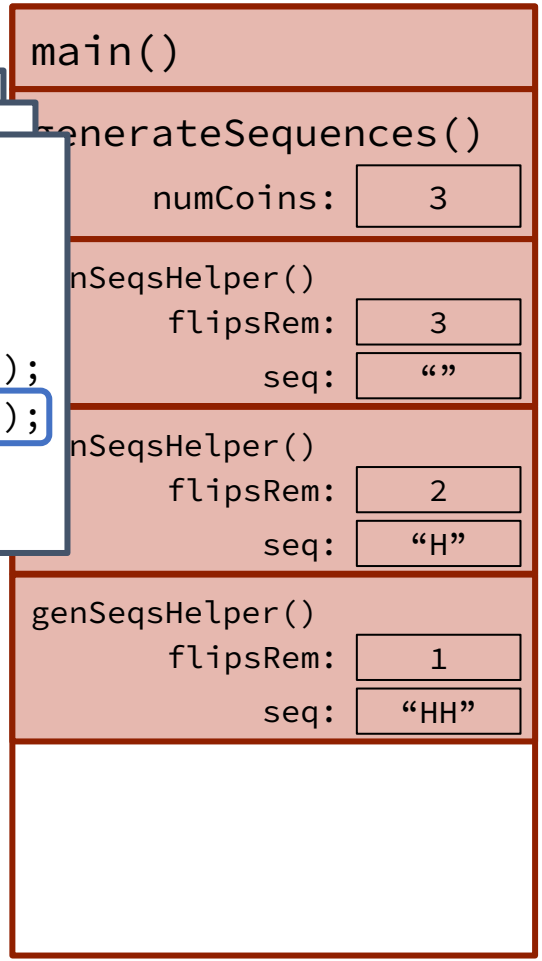
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
```

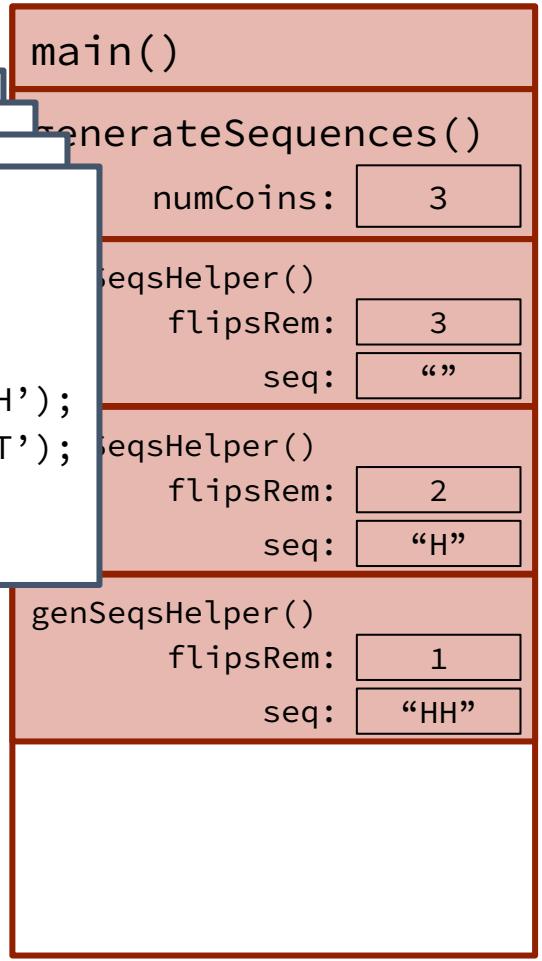
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
```

```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



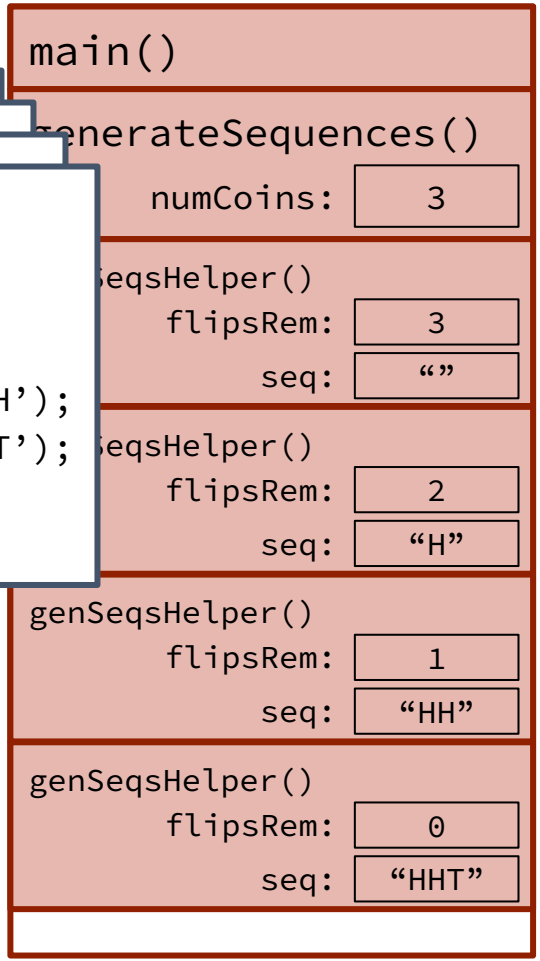
Console:

```
HHH
```

```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

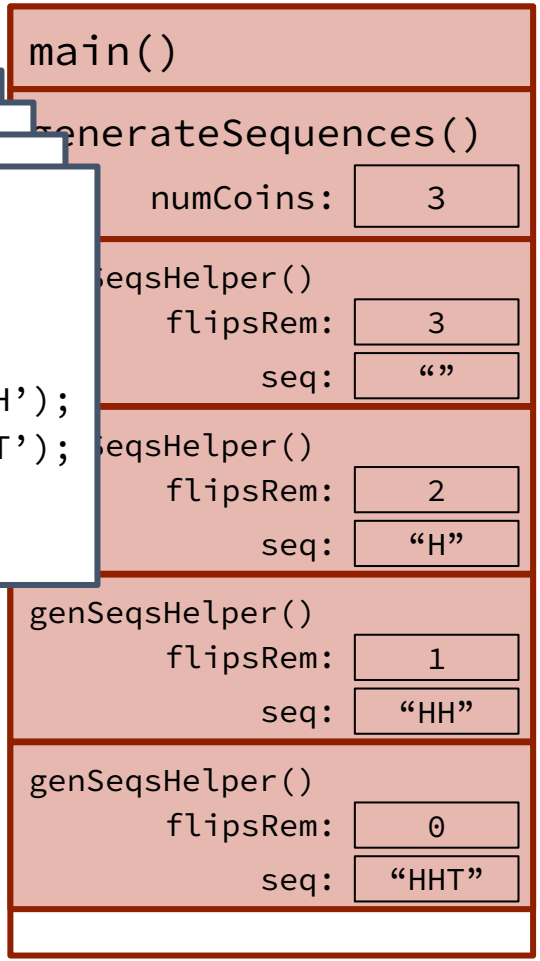
```
HHH
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

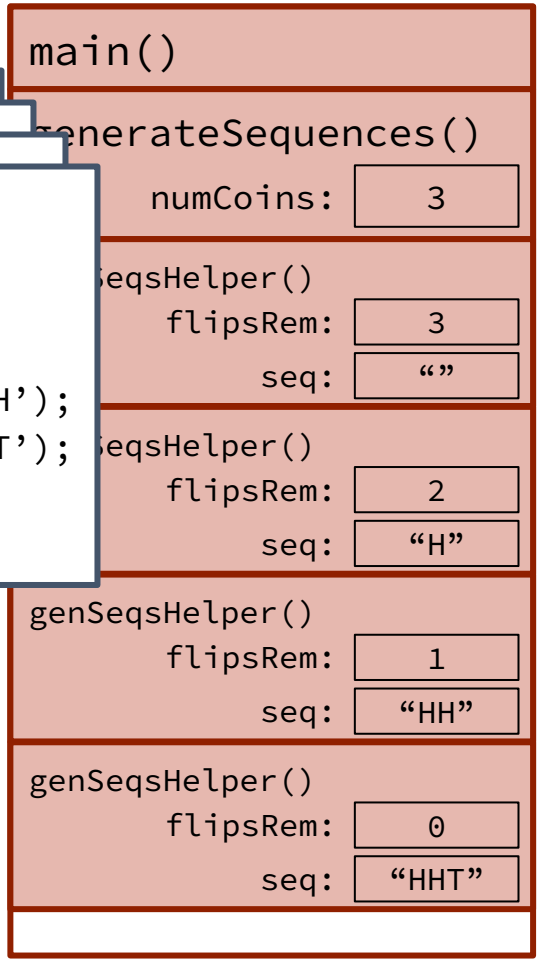
```
HHH
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

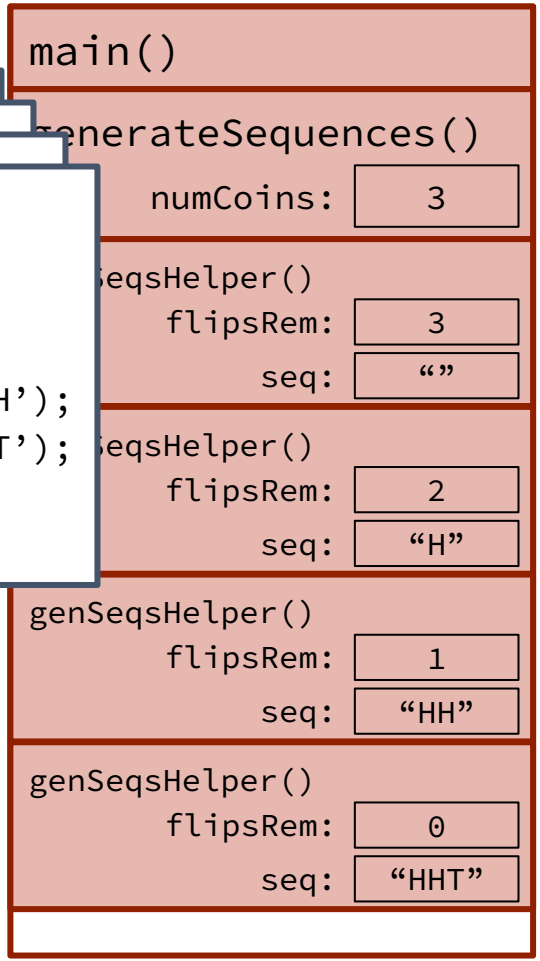
```
HHH
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

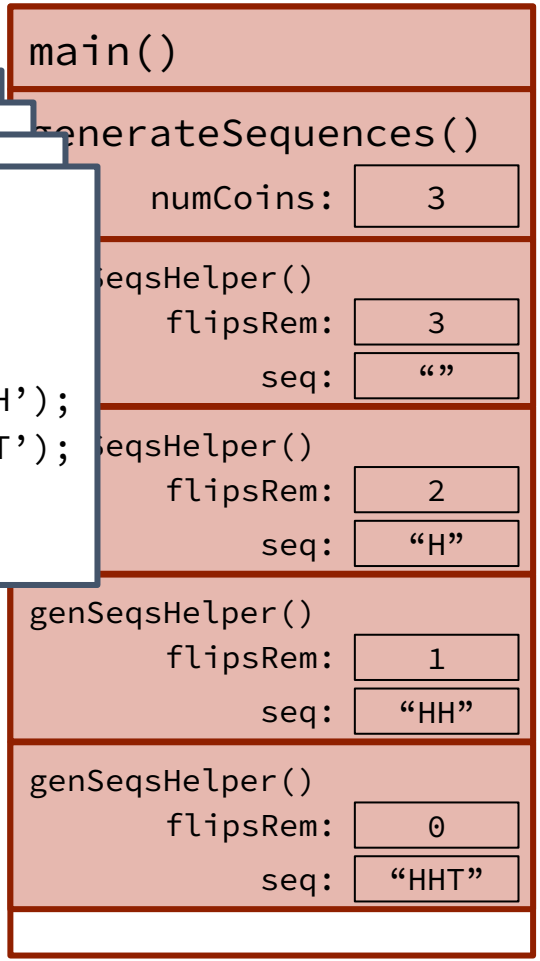
```
HHH
HHT
```



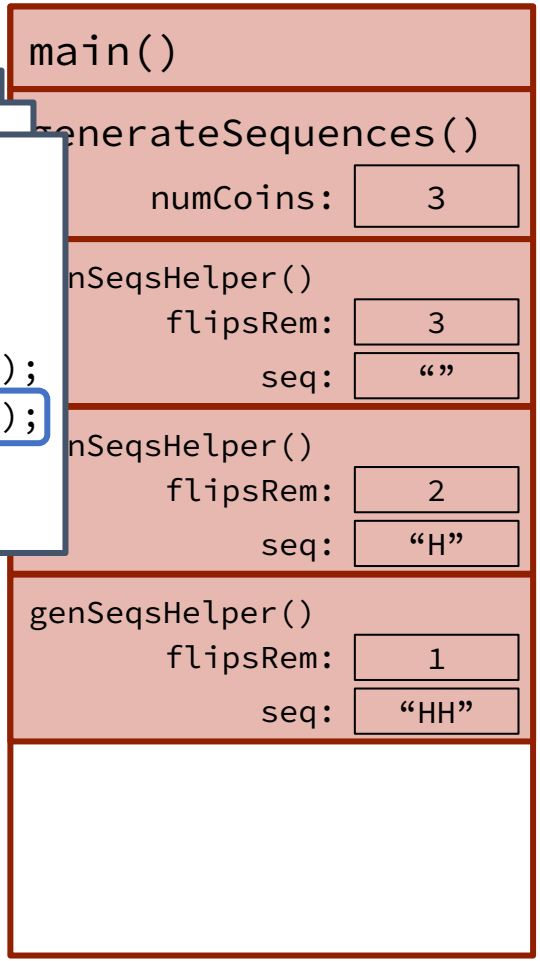
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

```
HHH
HHT
```



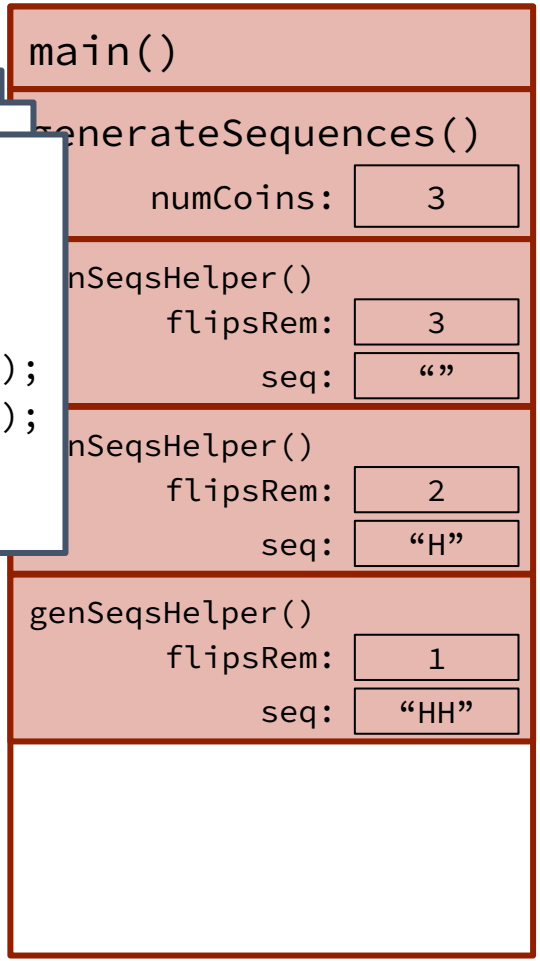
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
```

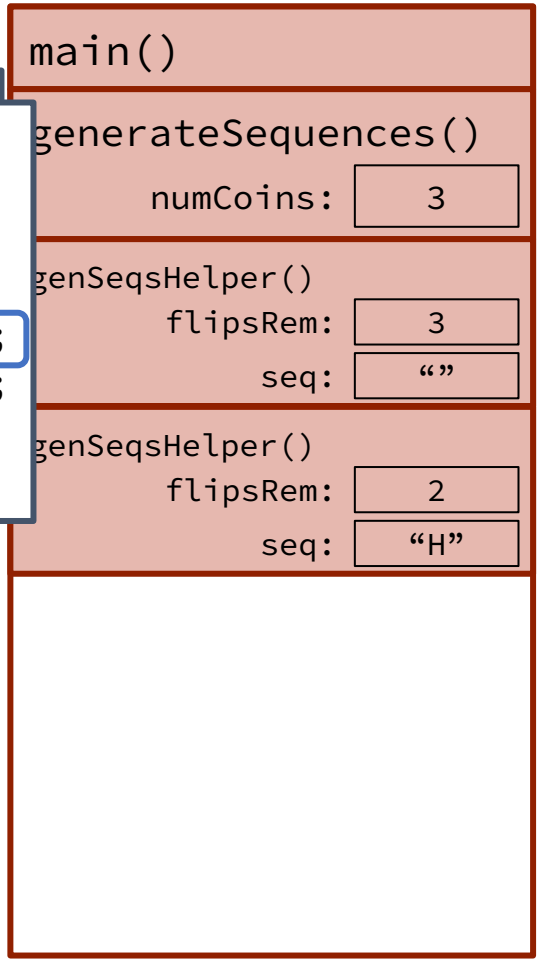
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
```

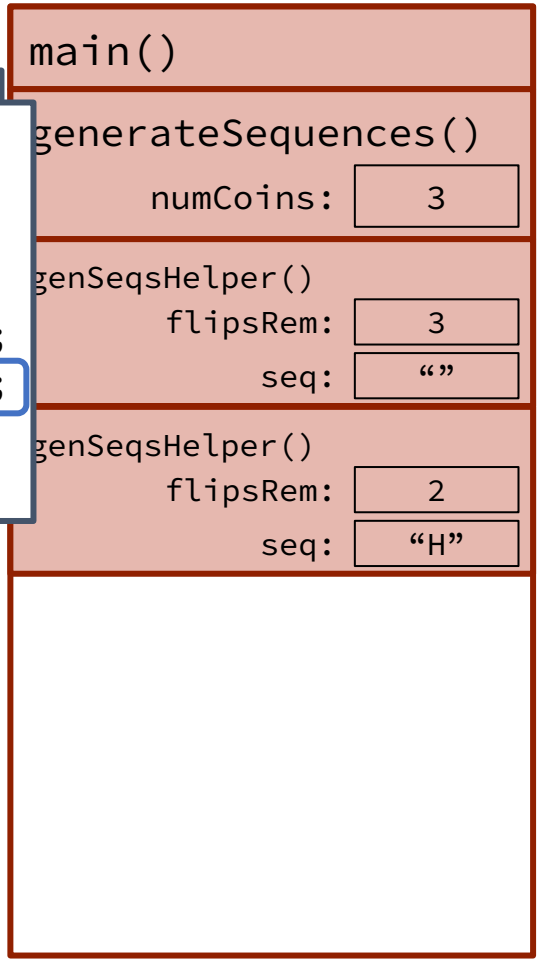
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
```

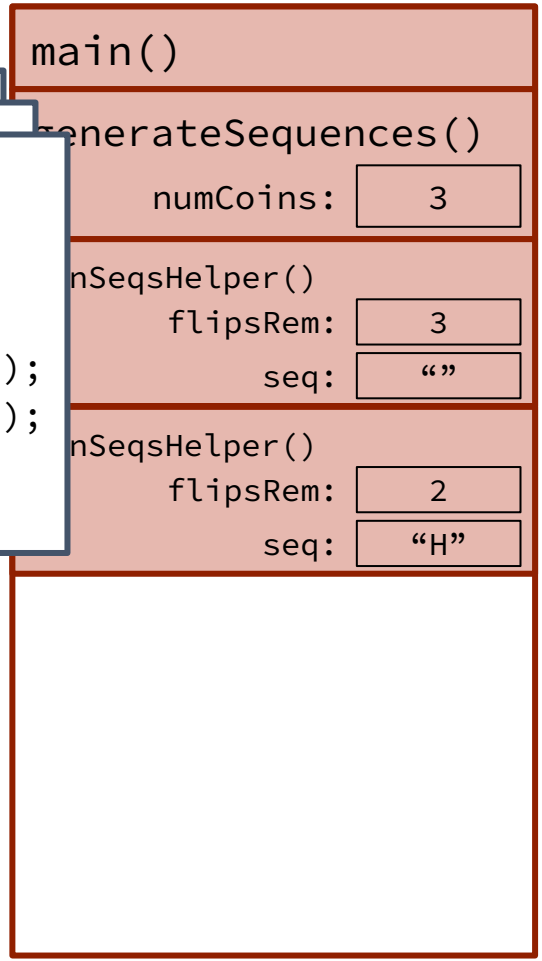
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
```

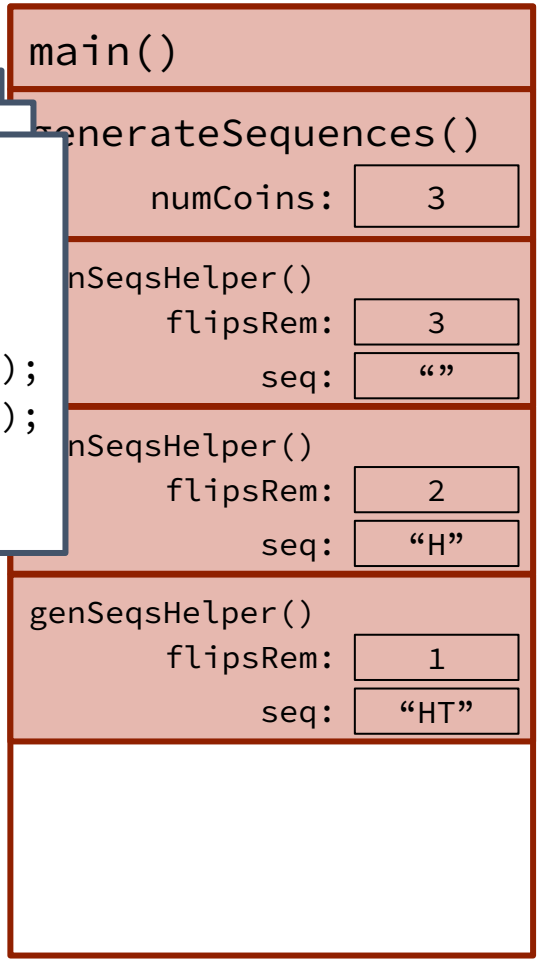
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
```

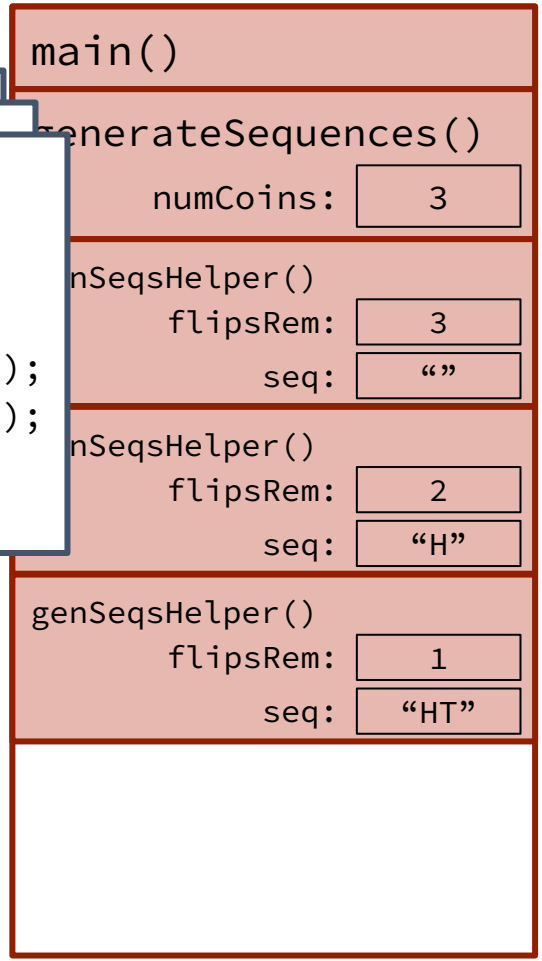
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
```

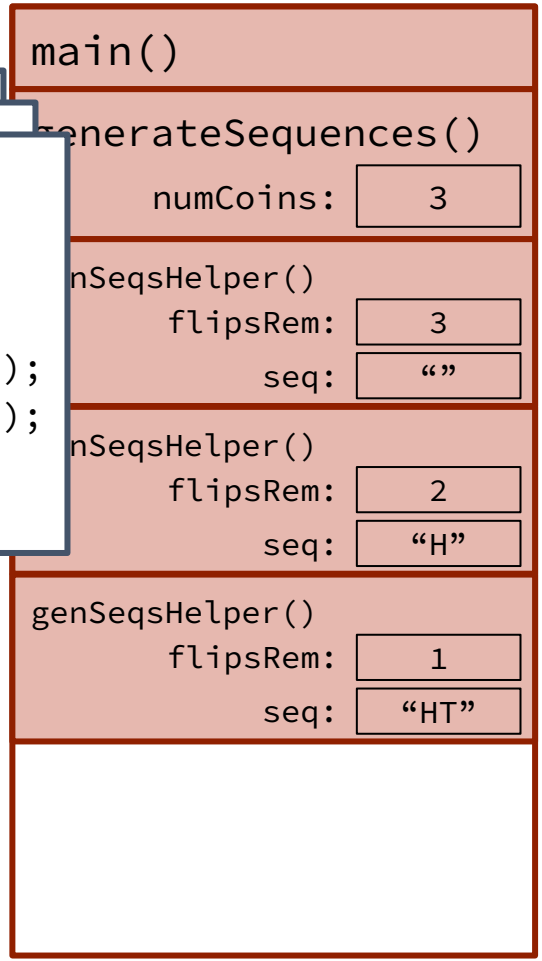
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
```

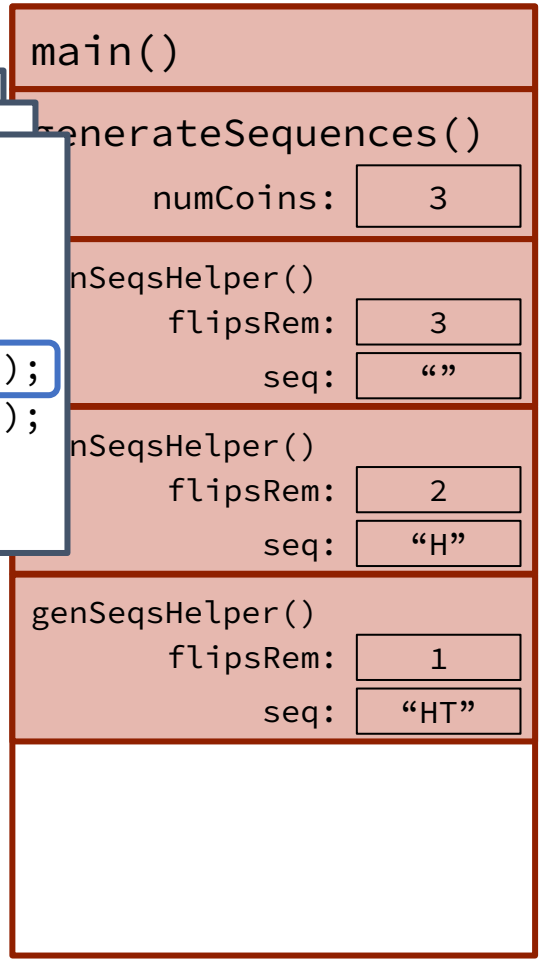
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
```

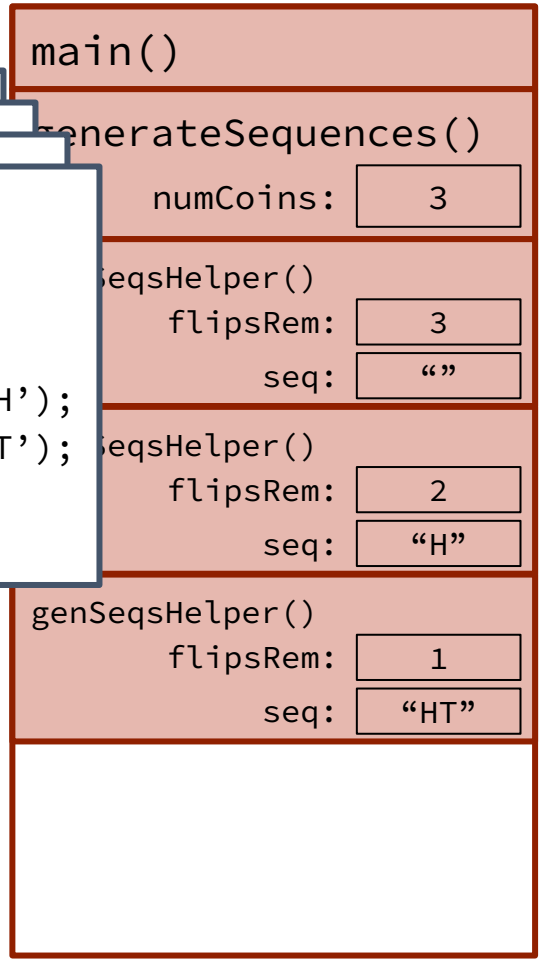
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
```

```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



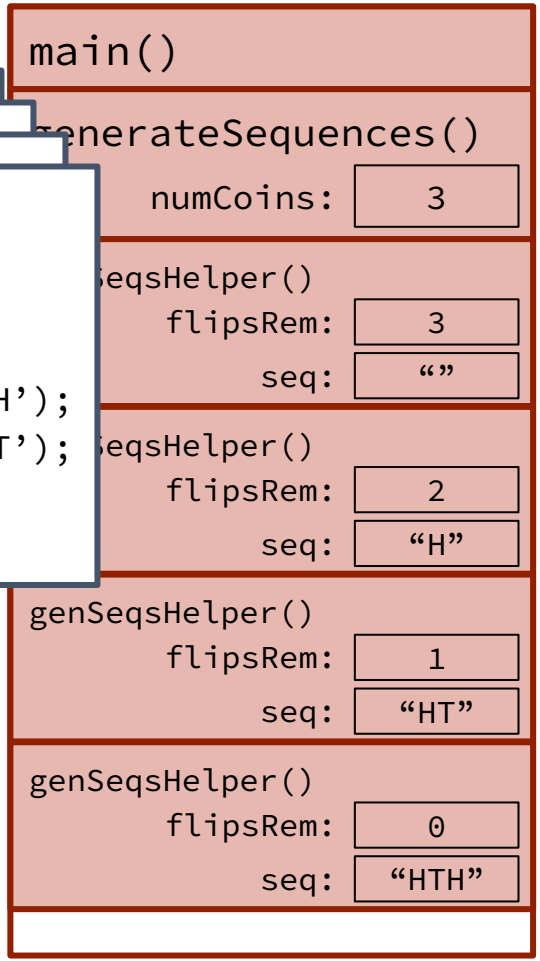
Console:

```
HHH
HHT
```

```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

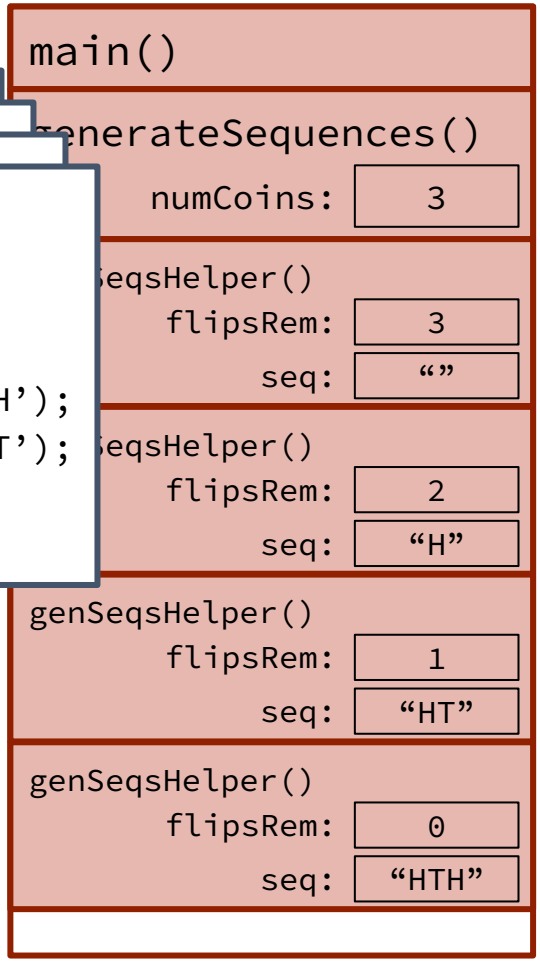
```
HHH
HHT
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

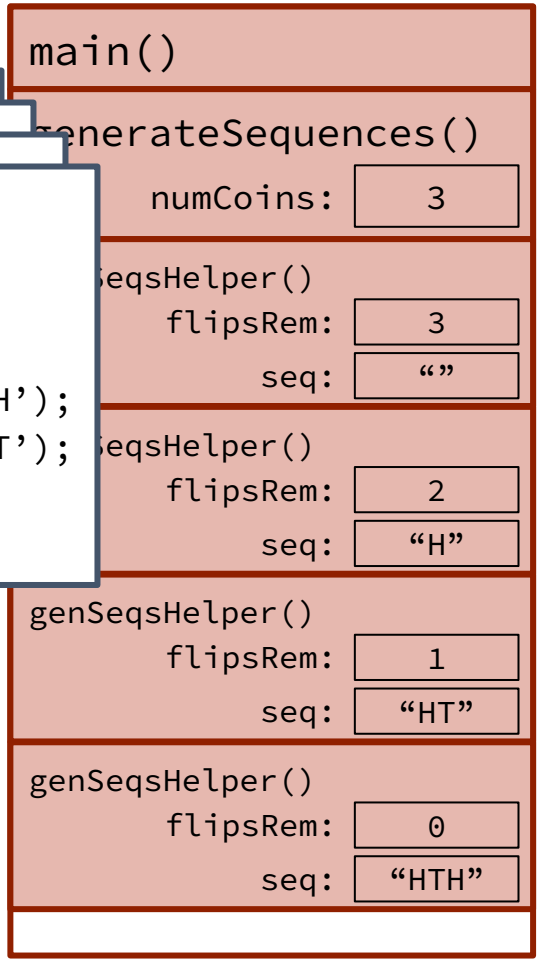
```
HHH
HHT
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

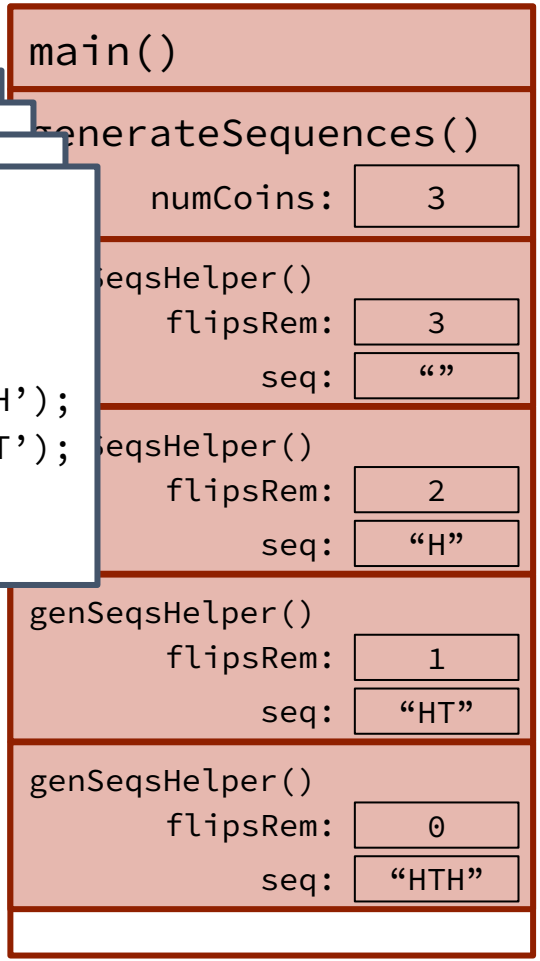
```
HHH
HHT
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

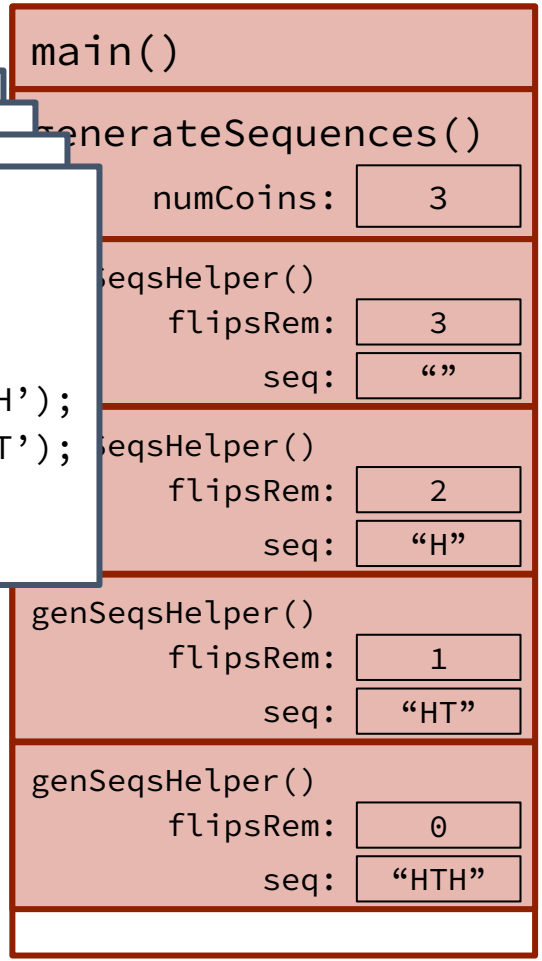
```
HHH
HHT
HTH
```



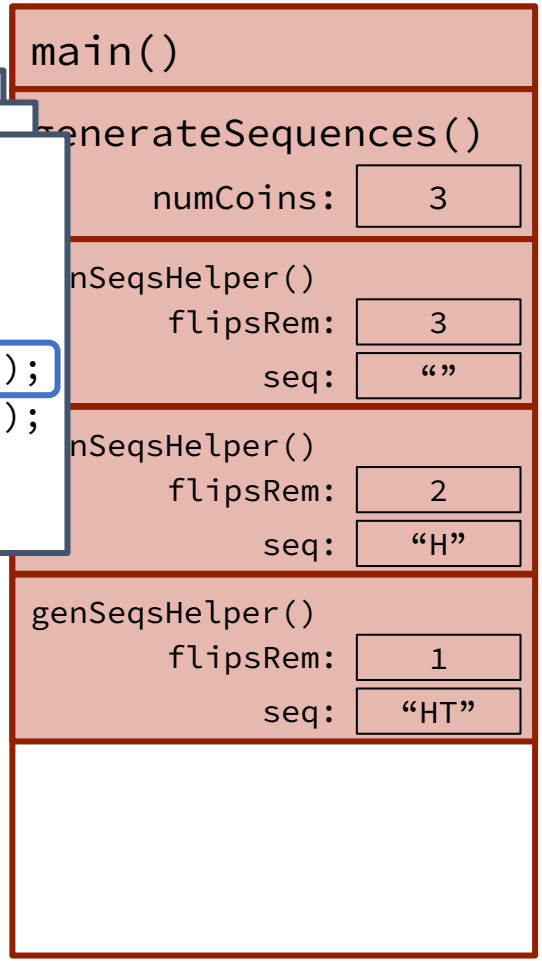
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

```
HHH
HHT
HTH
```



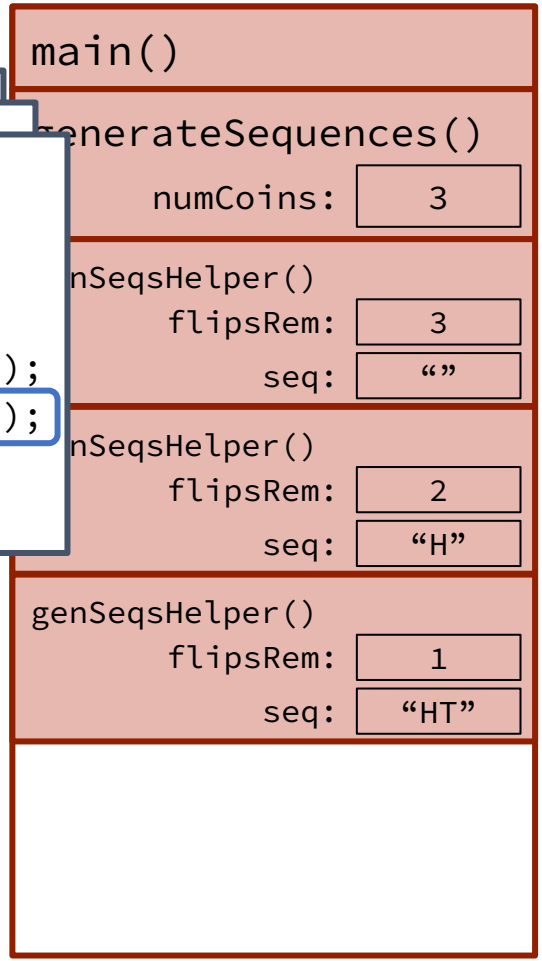
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
```

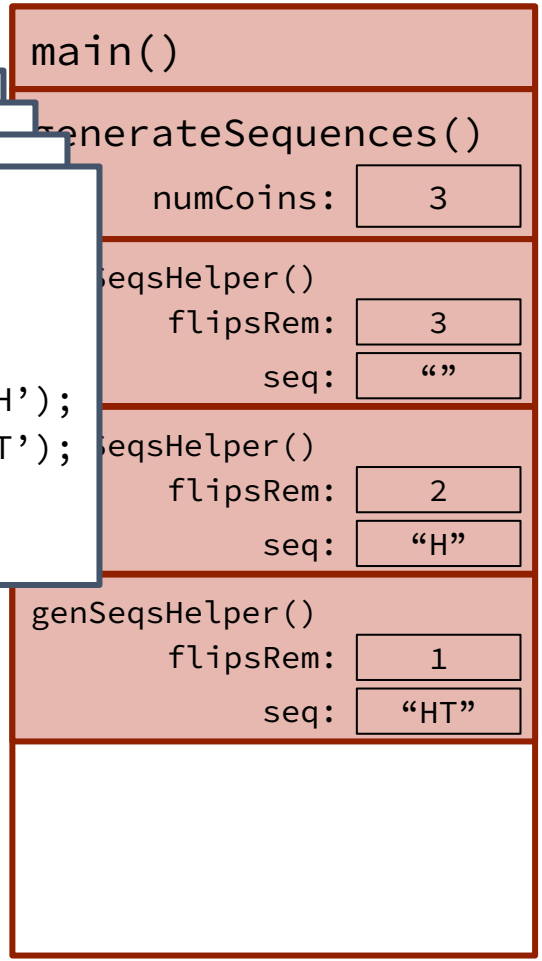
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
```

```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



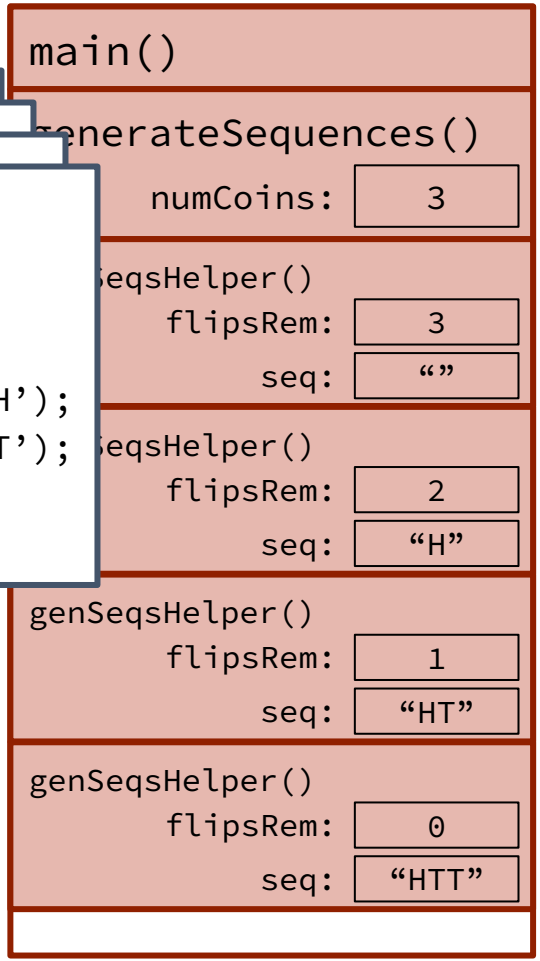
Console:

```
HHH
HHT
HTH
```

```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

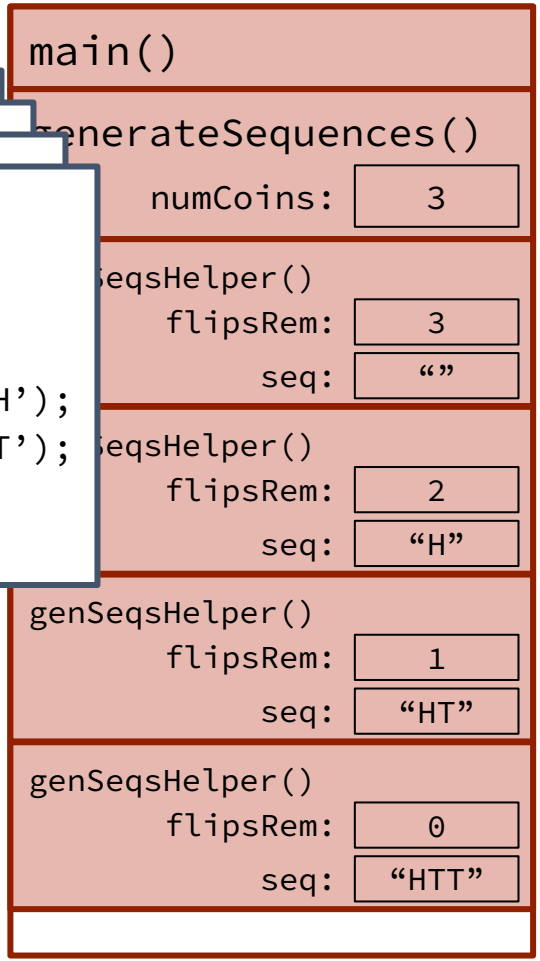
```
HHH
HHT
HTH
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

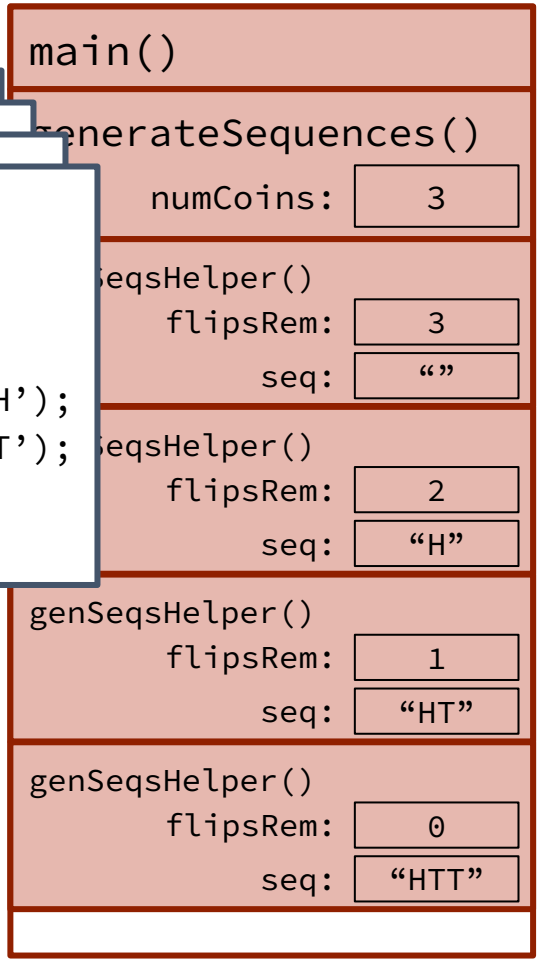
```
HHH
HHT
HTH
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

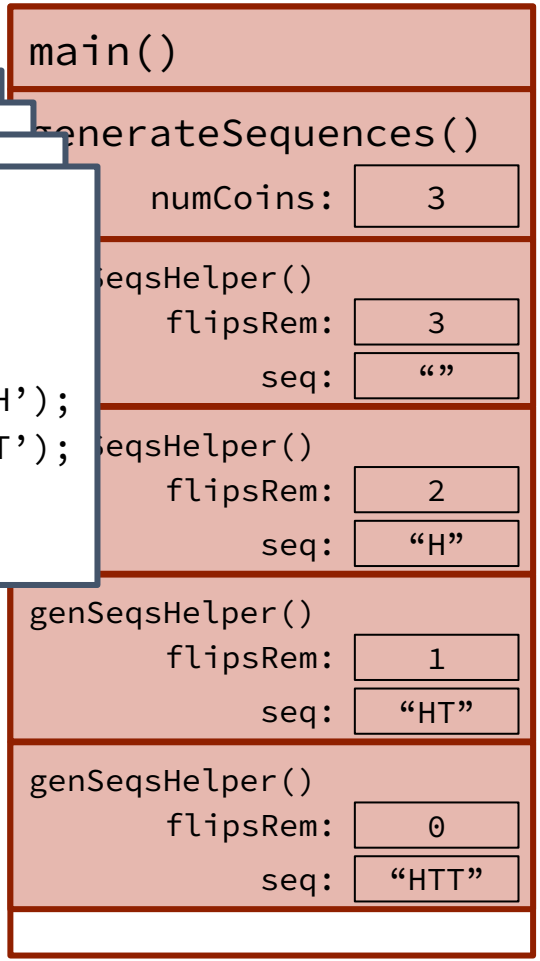
```
HHH
HHT
HTH
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

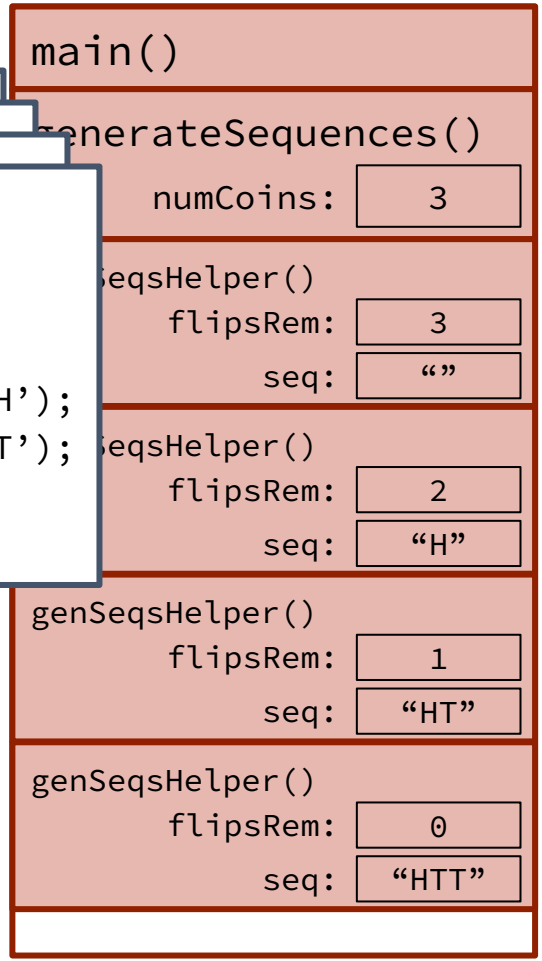
```
HHH
HHT
HTH
HTT
```



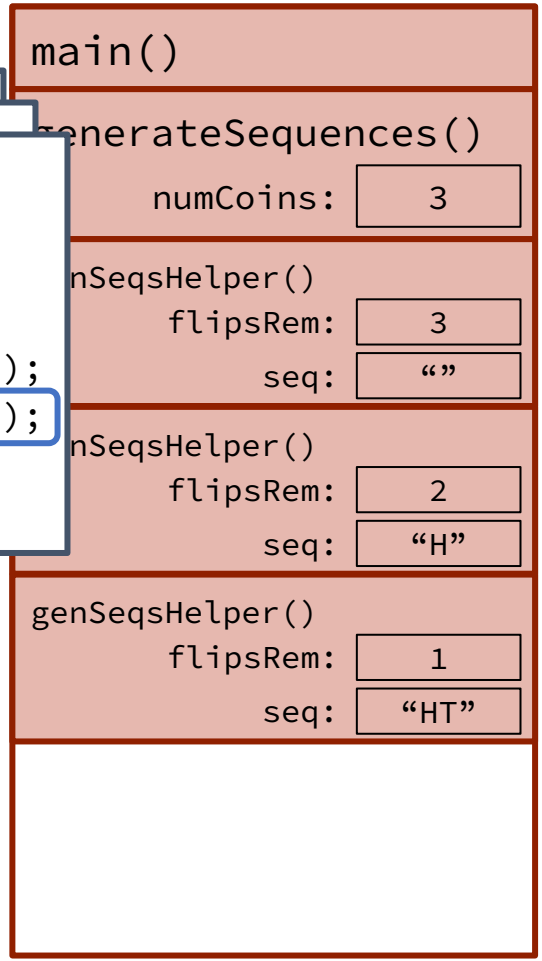
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

```
HHH
HHT
HTH
HTT
```



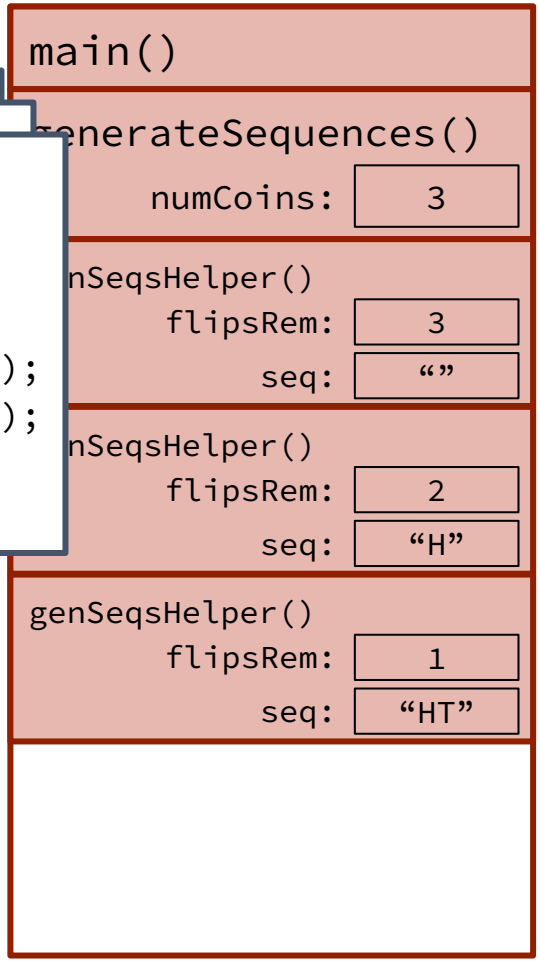
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
```

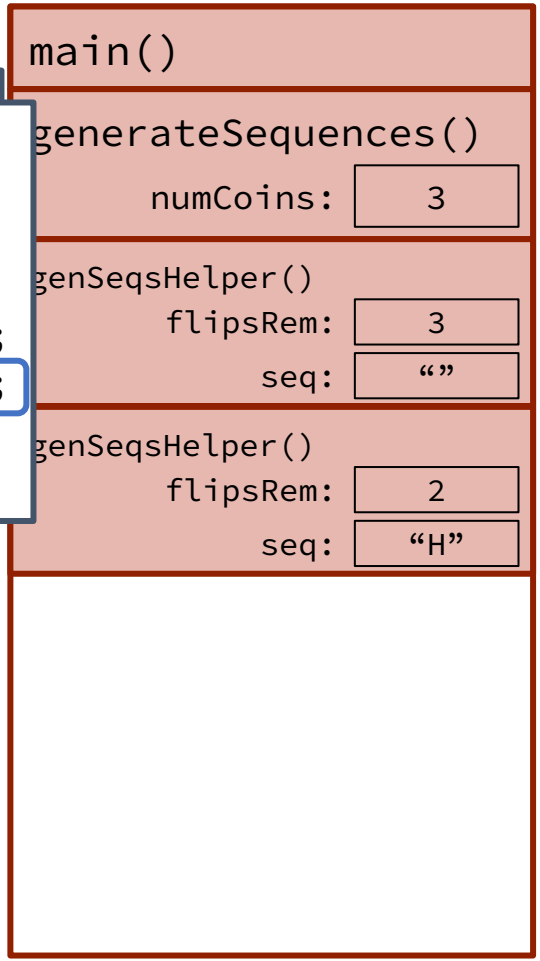
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
```

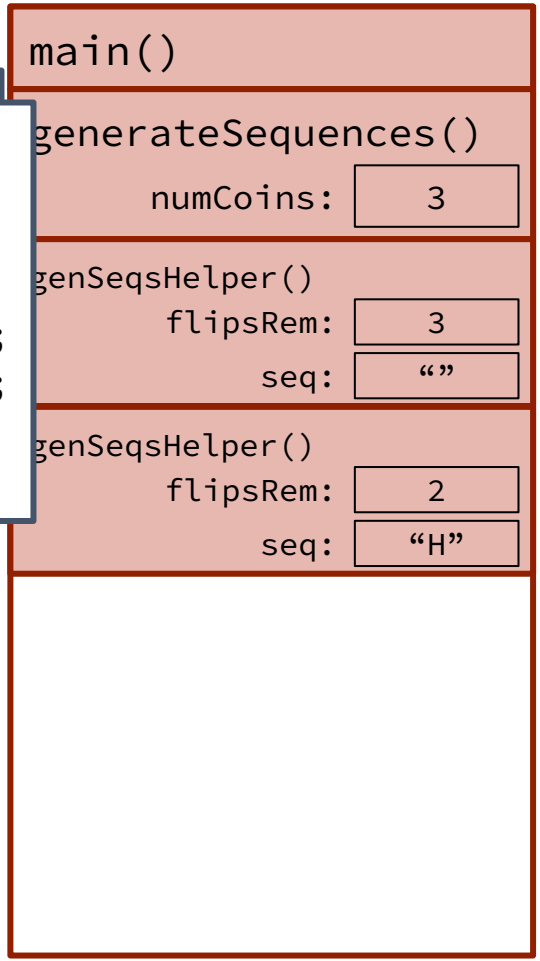
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
```

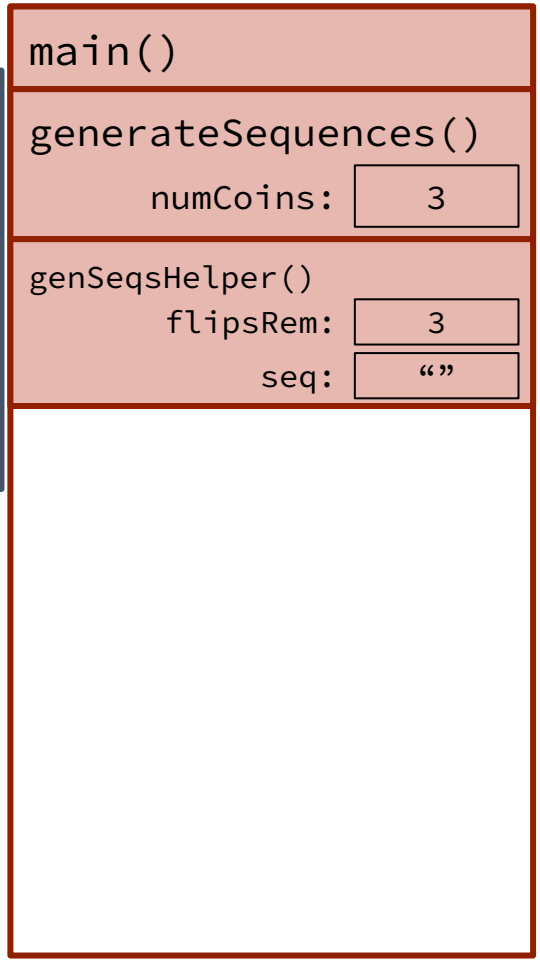
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
```

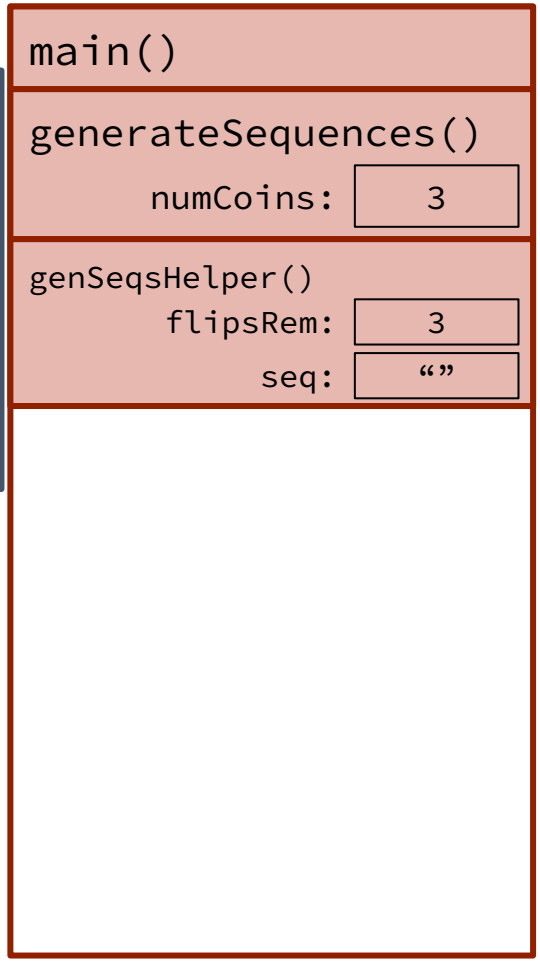
```
void genSeqsHelper (int flipsRem, string seq) {
     if (flipsRem == 0) {
          cout << seq << endl;
     } else {
          generateSeqsHelper(flipsRem-1, seq + 'H');
          generateSeqsHelper(flipsRem-1, seq + 'T');
     }
}
```



Console:

```
HHH
HHT
HTH
HTT
```

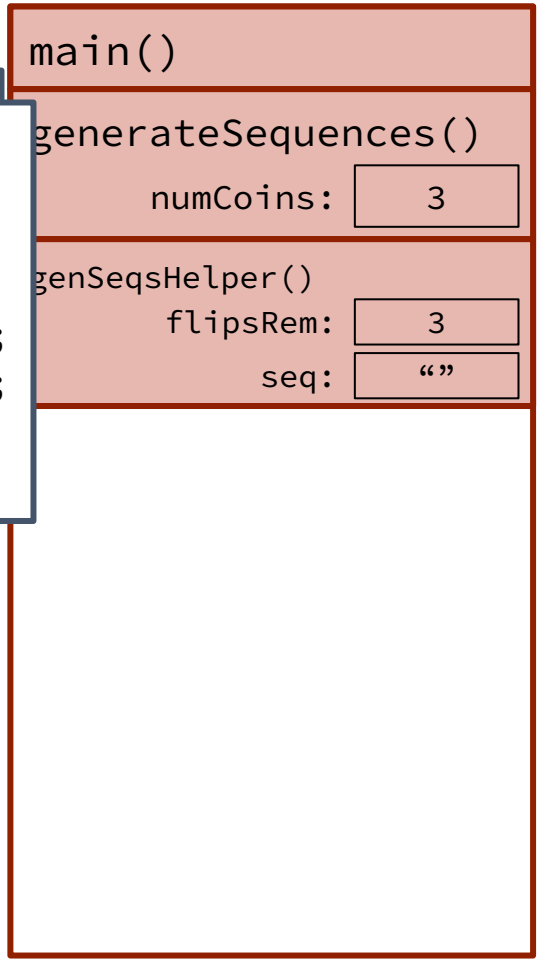
```
void genSeqsHelper (int flipsRem, string seq) {
     if (flipsRem == 0) {
          cout << seq << endl;
     } else {
          generateSeqsHelper(flipsRem-1, seq + 'H');
          generateSeqsHelper(flipsRem-1, seq + 'T');
     }
}
```



Console:

```
HHH
HHT
HTH
HTT
```

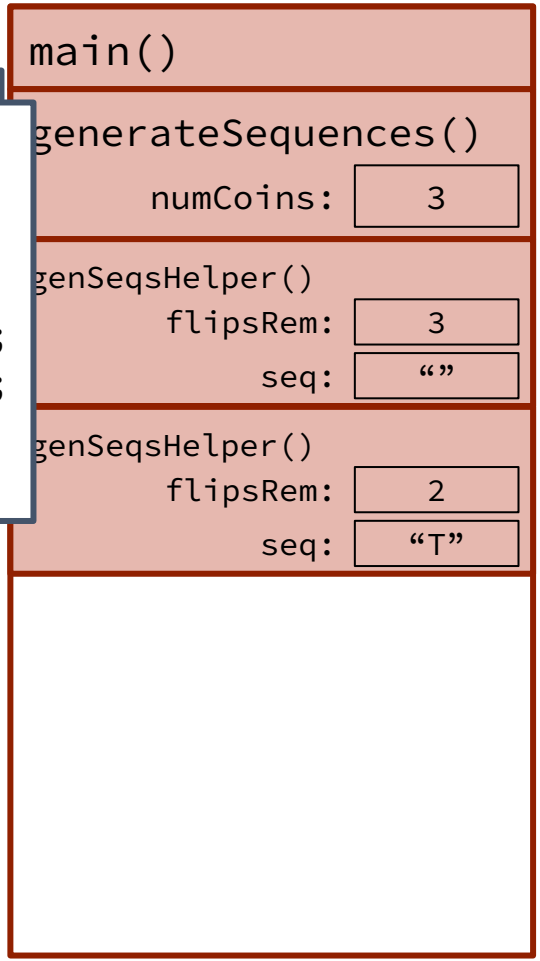
```
void genSeqsHelper (int flipsRem, string seq) {
     if (flipsRem == 0) {
          cout << seq << endl;
     } else {
          generateSeqsHelper(flipsRem-1, seq + 'H');
          generateSeqsHelper(flipsRem-1, seq + 'T');
     }
}
```



Console:

```
HHH
HHT
HTH
HTT
```

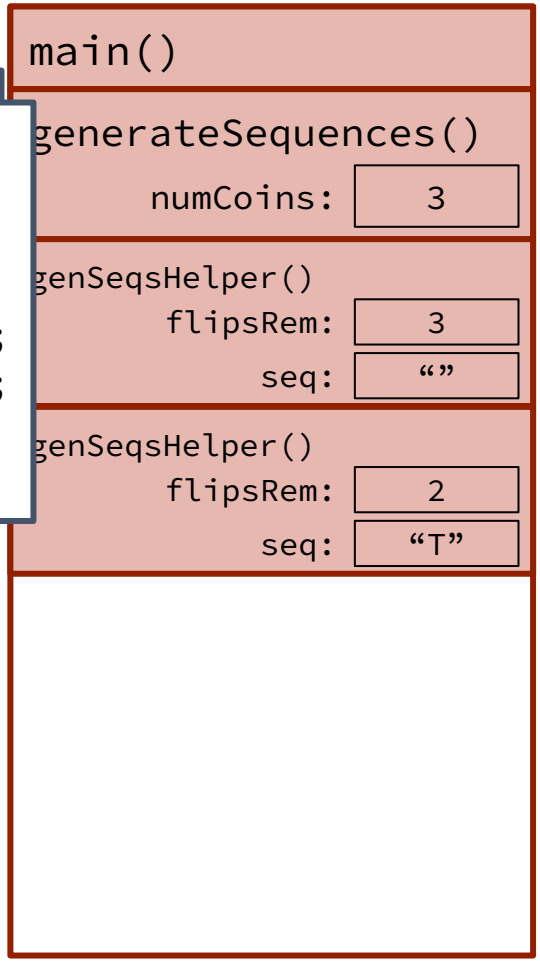
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
```

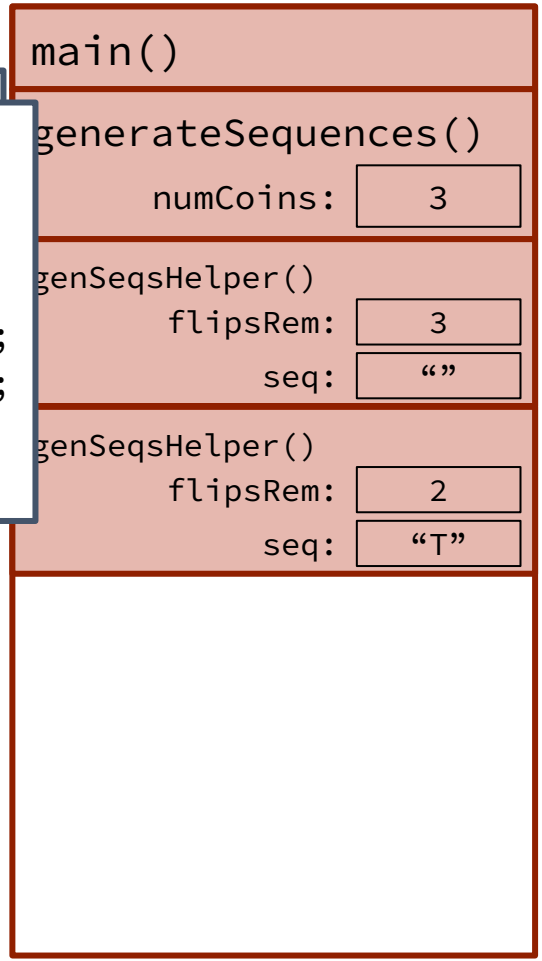
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
```

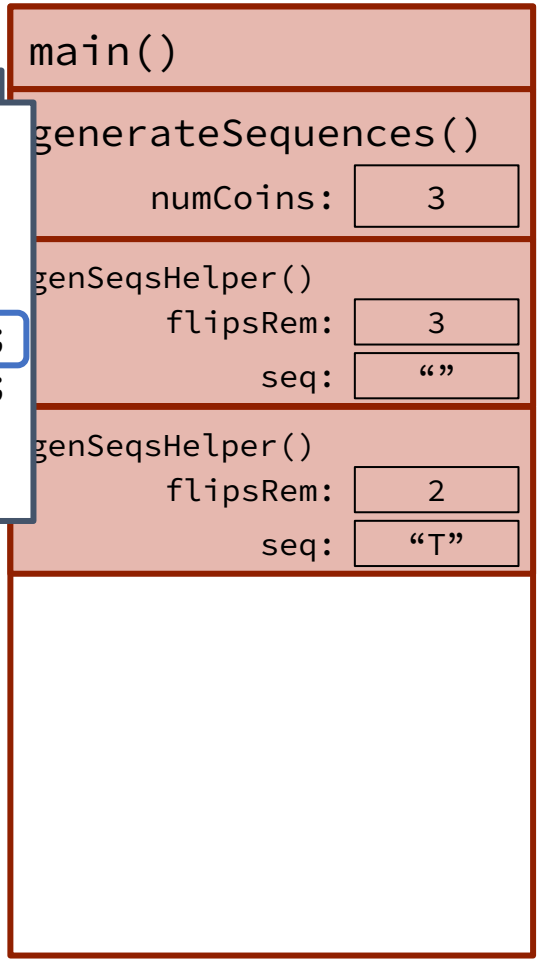
```
void genSeqsHelper (int flipsRem, string seq) {
     if (flipsRem == 0) {
          cout << seq << endl;
     } else {
          generateSeqsHelper(flipsRem-1, seq + 'H');
          generateSeqsHelper(flipsRem-1, seq + 'T');
     }
}
```



Console:

```
HHH
HHT
HTH
HTT
```

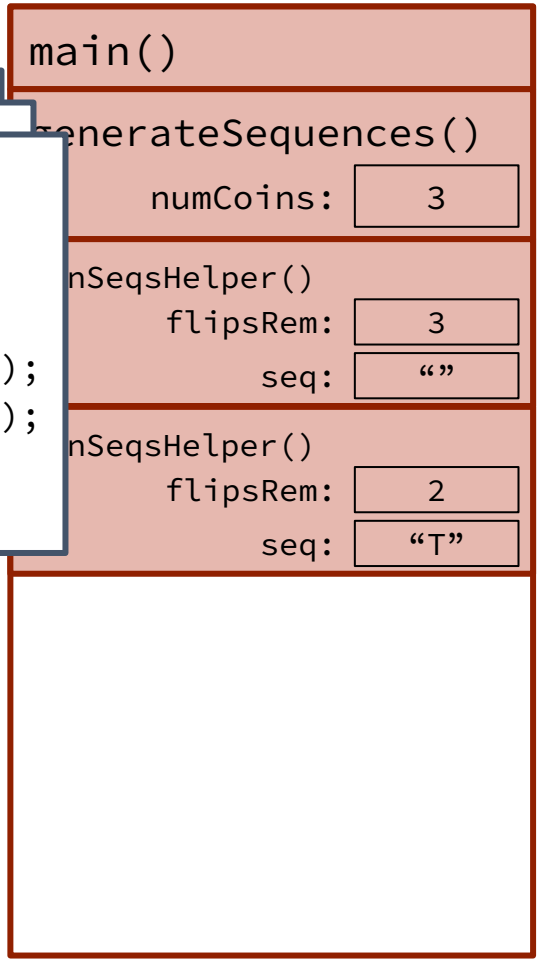
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
```

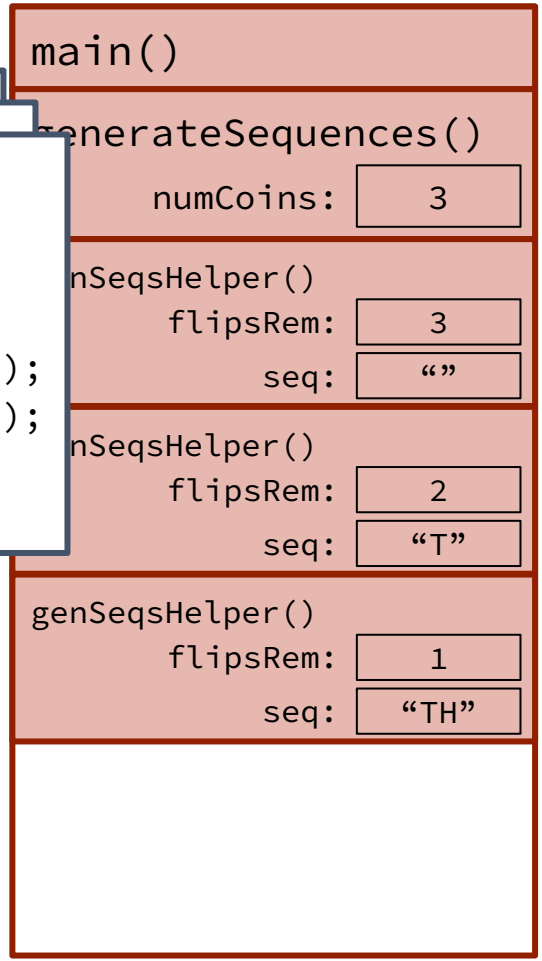
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
```

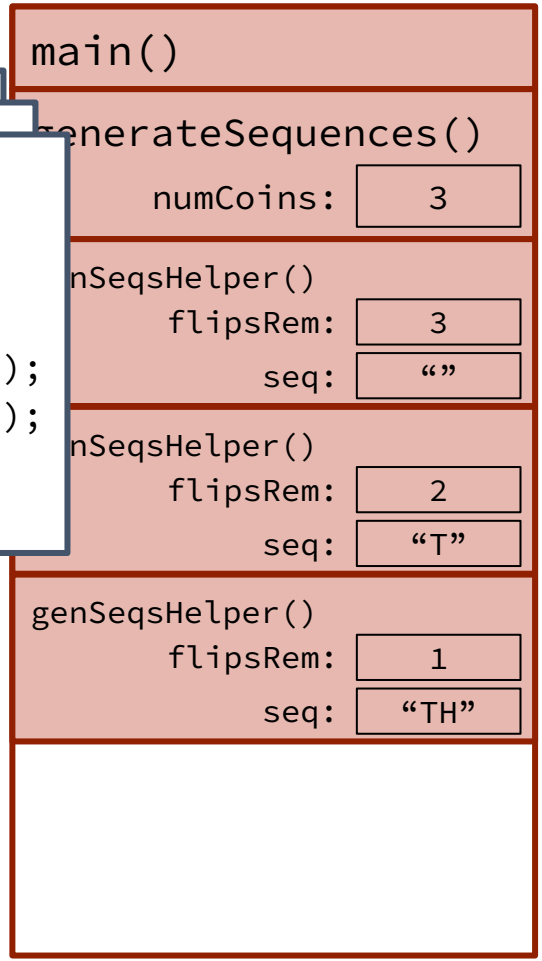
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
```

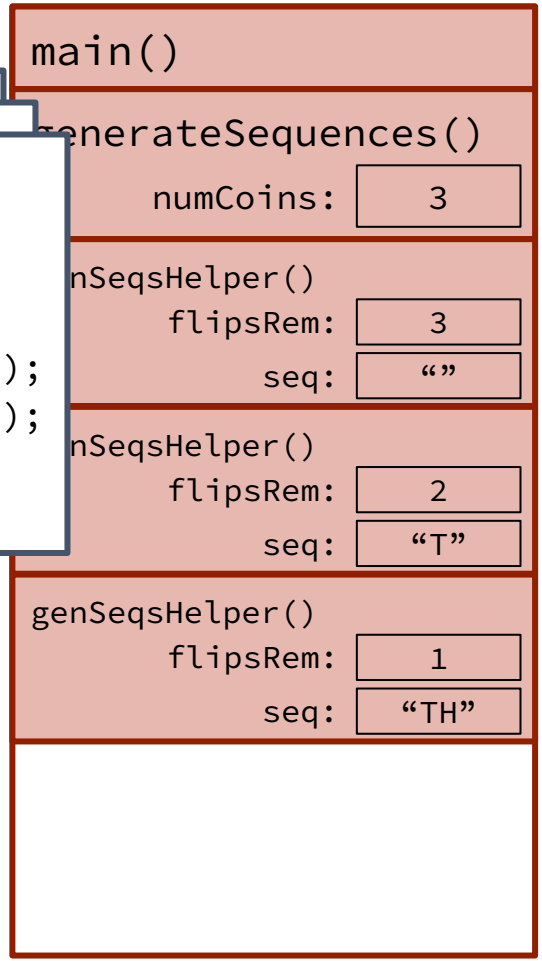
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
```

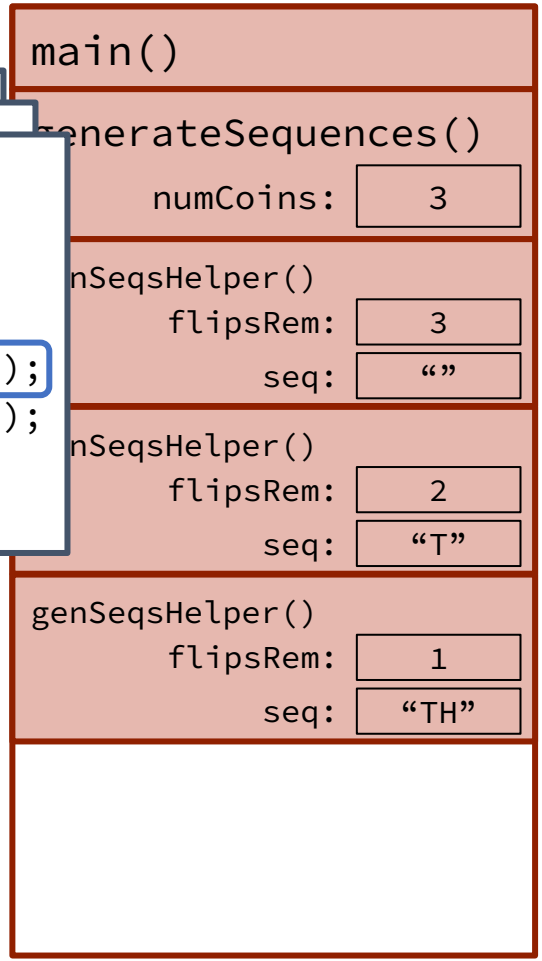
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
```

```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



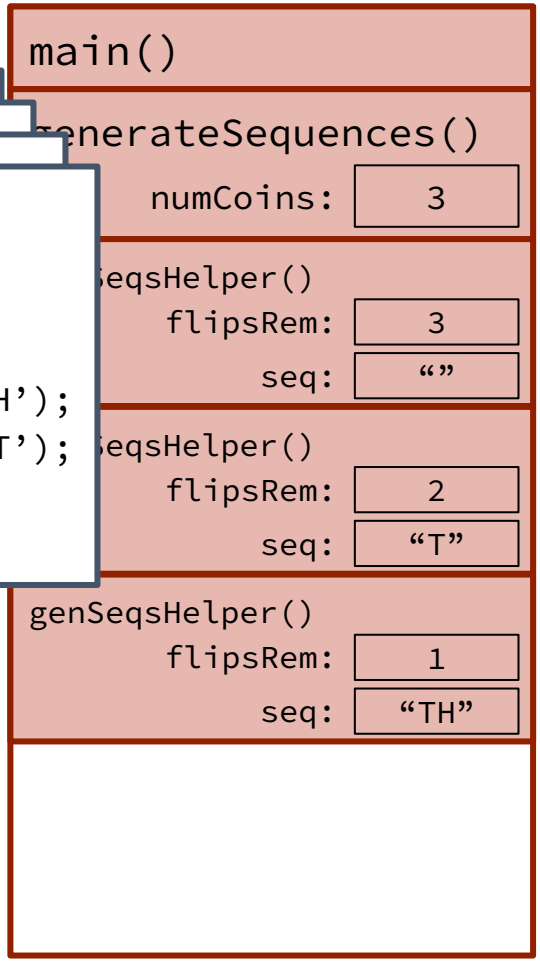
Console:

```
HHH
HHT
HTH
HTT
```

```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

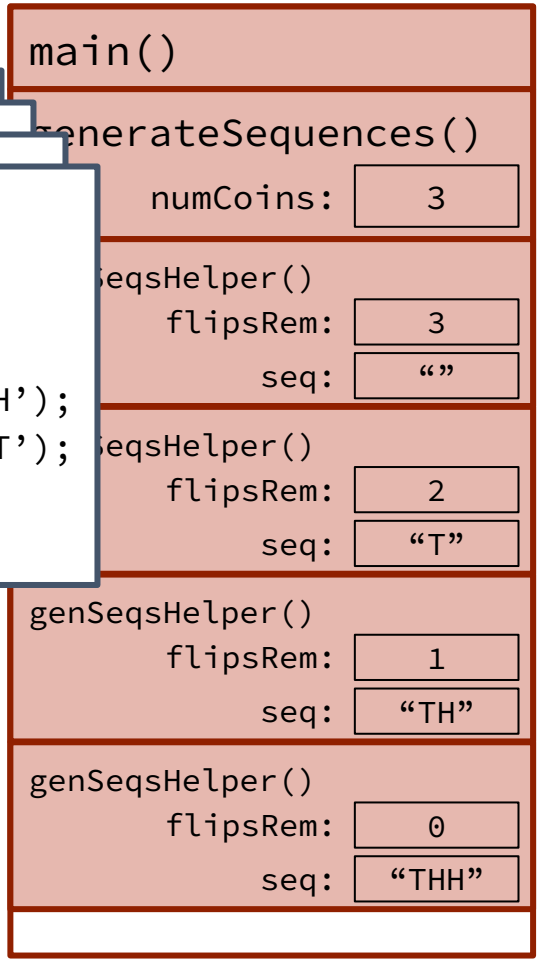
```
HHH
HHT
HTH
HTT
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

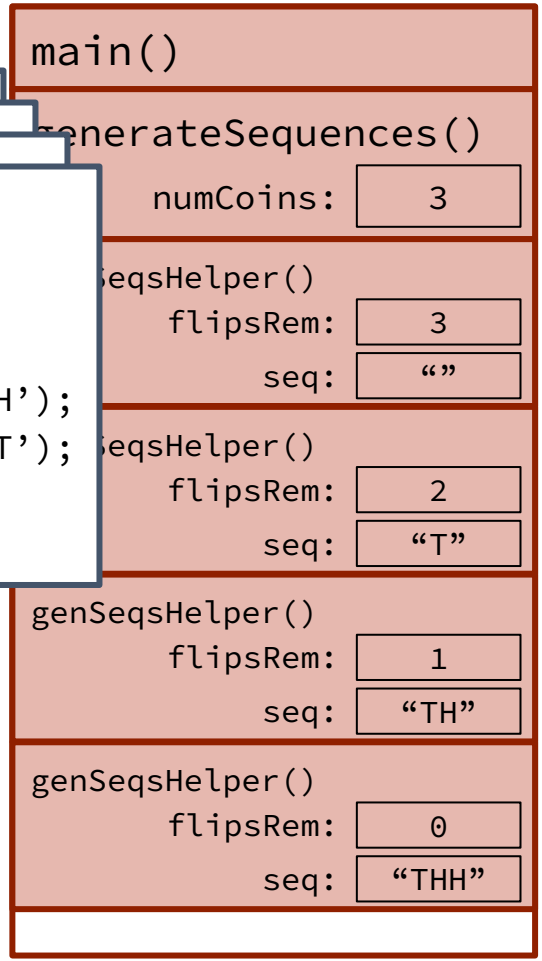
```
HHH
HHT
HTH
HTT
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

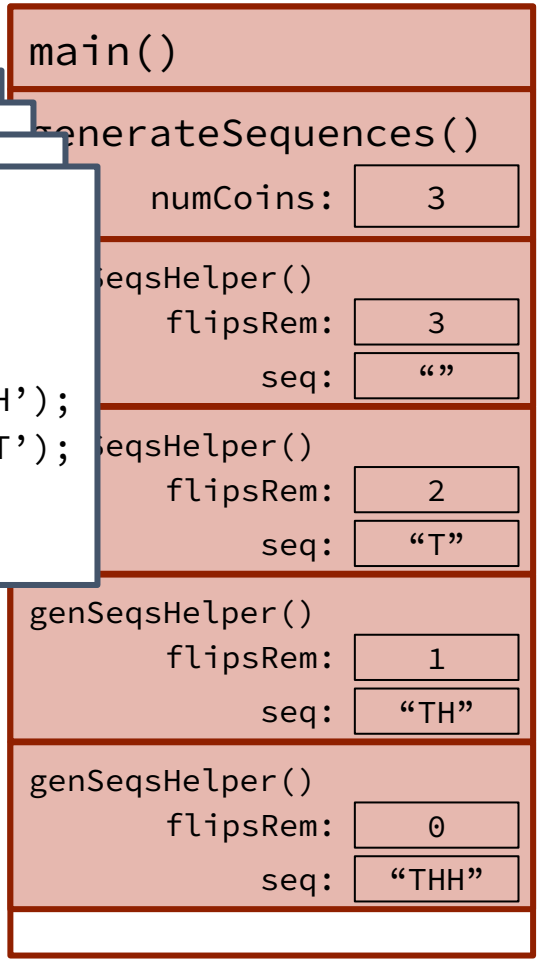
```
HHH
HHT
HTH
HTT
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

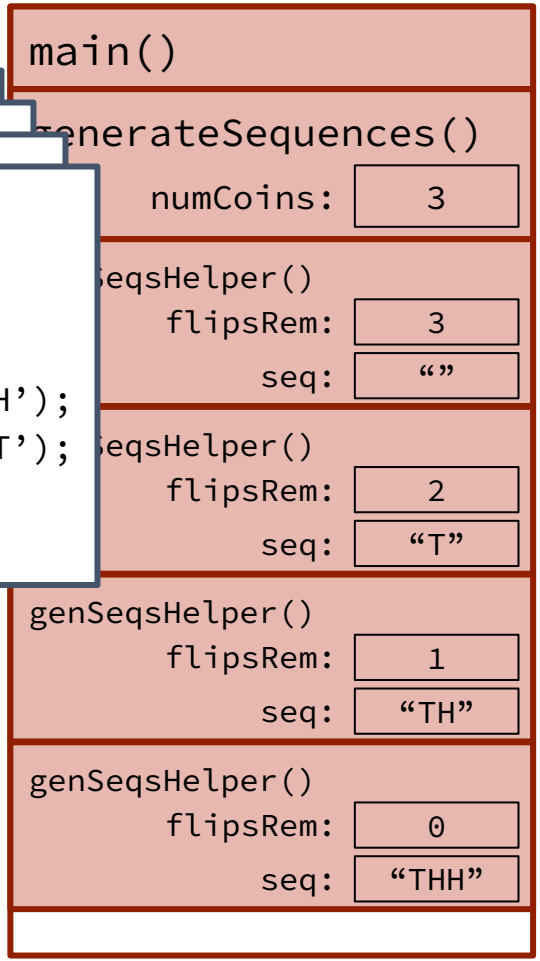
```
HHH
HHT
HTH
HTT
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

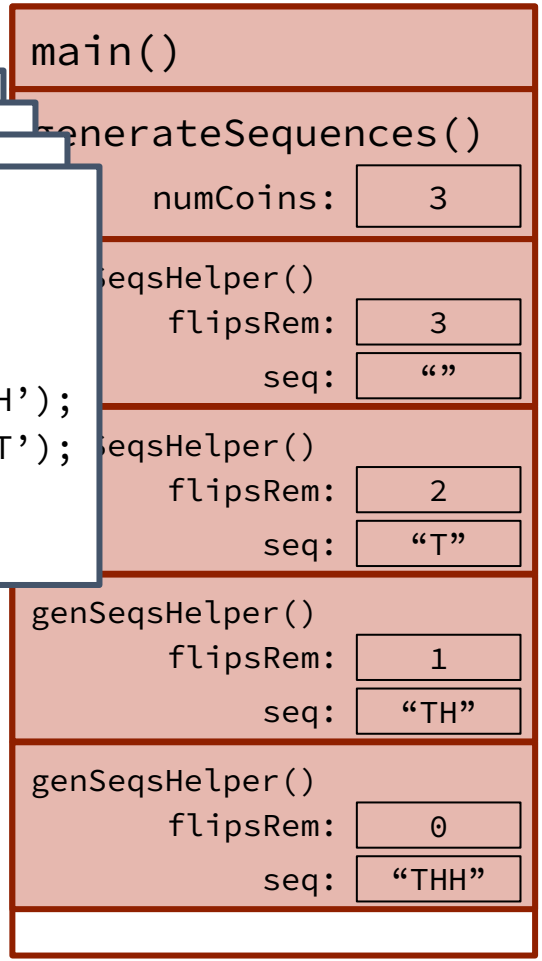
```
HHH
HHT
HTH
HTT
THH
```



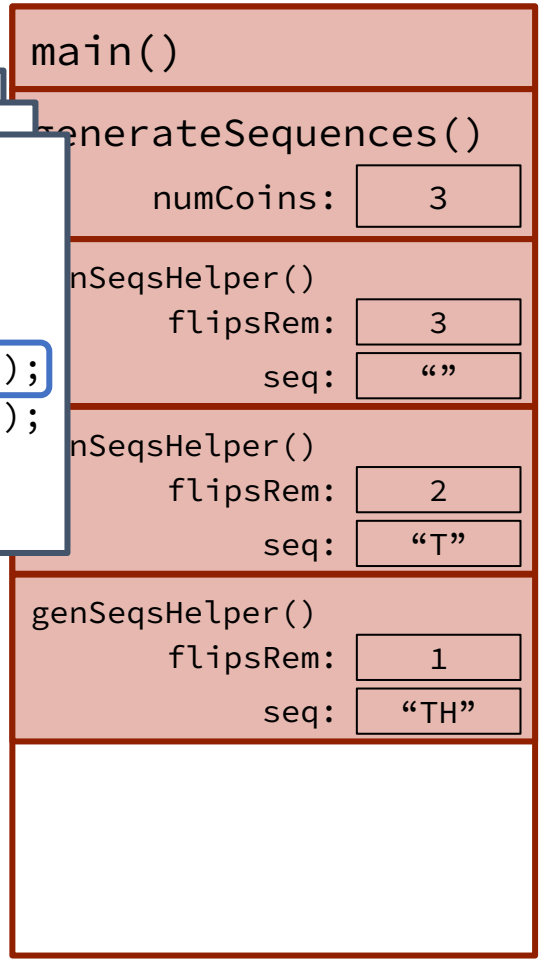
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

```
HHH
HHT
HTH
HTT
THH
```



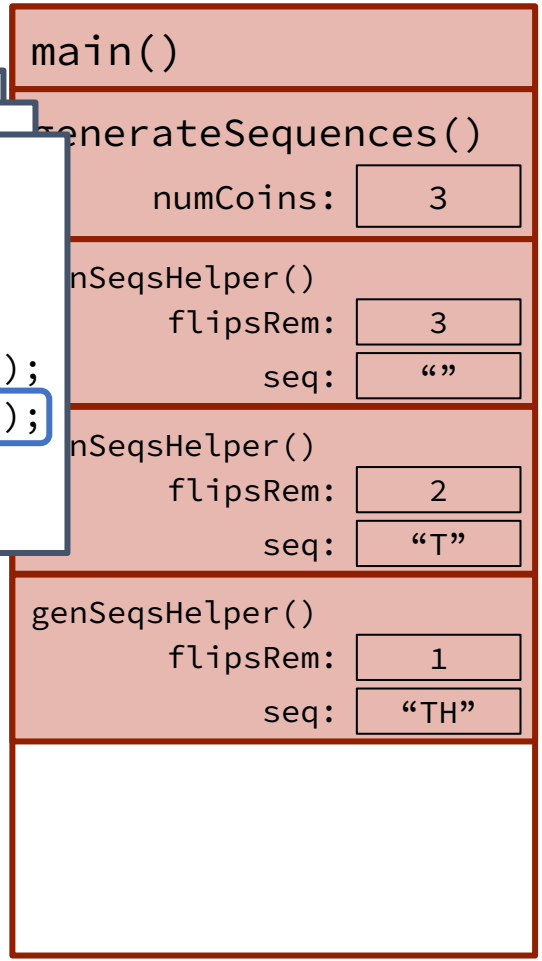
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
THH
```

```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



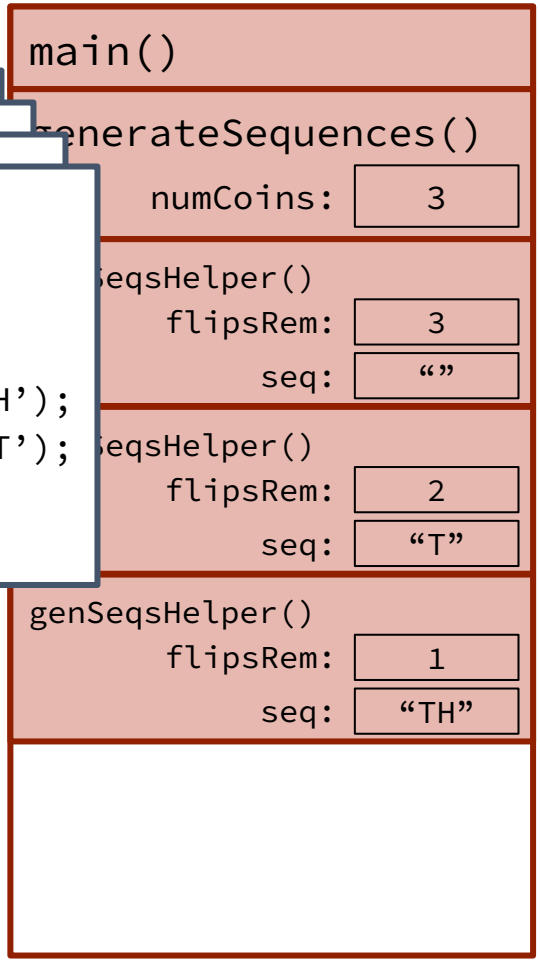
Console:

```
HHH
HHT
HTH
HTT
THH
```

```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

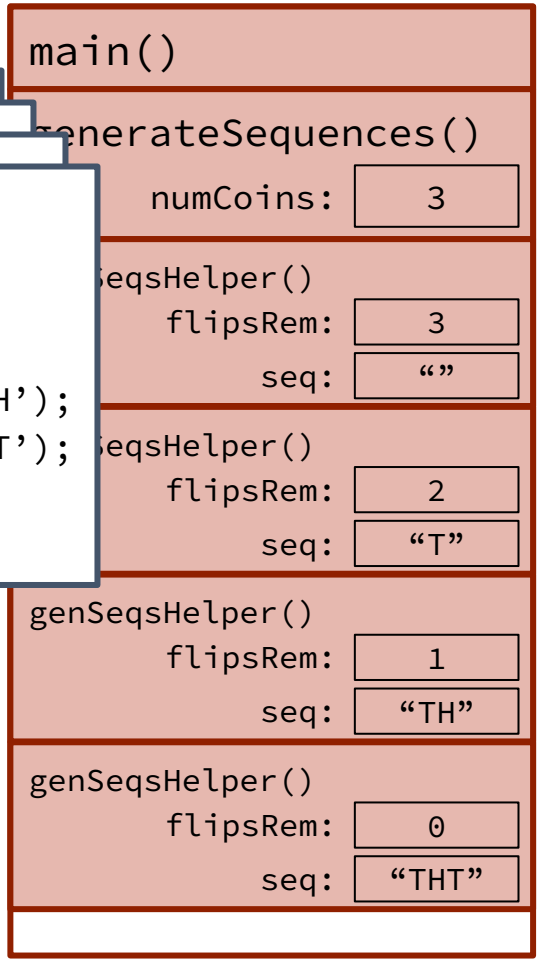
```
HHH
HHT
HTH
HTT
THH
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

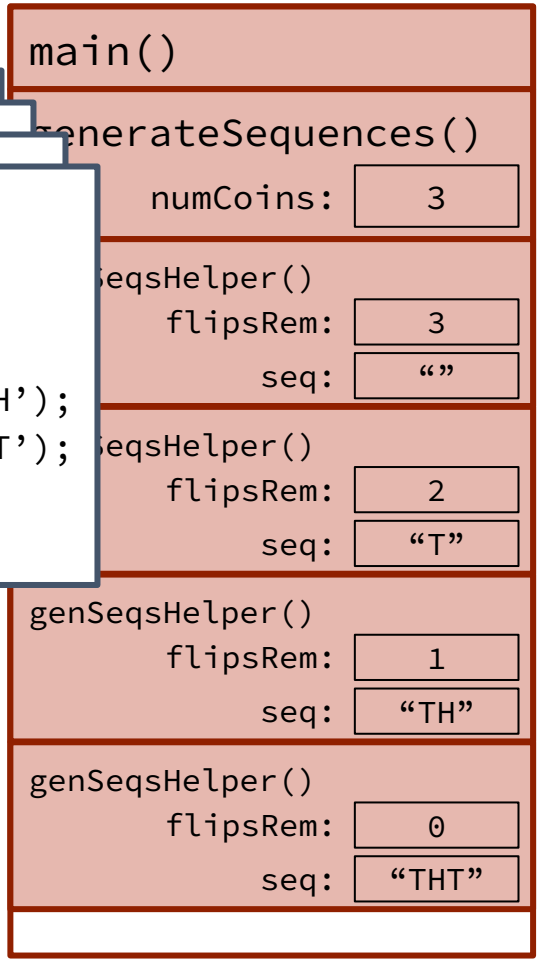
```
HHH
HHT
HTH
HTT
THH
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

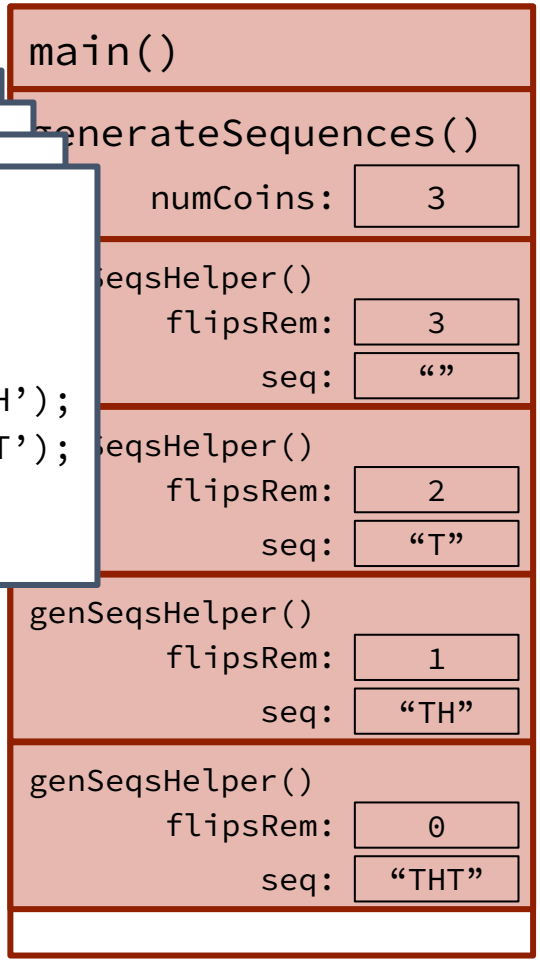
```
HHH
HHT
HTH
HTT
THH
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

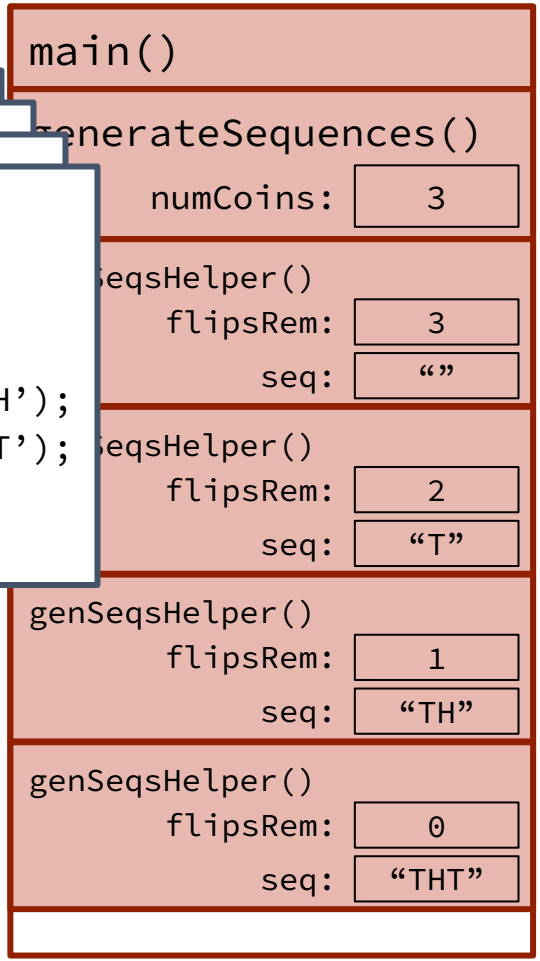
```
HHH
HHT
HTH
HTT
THH
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

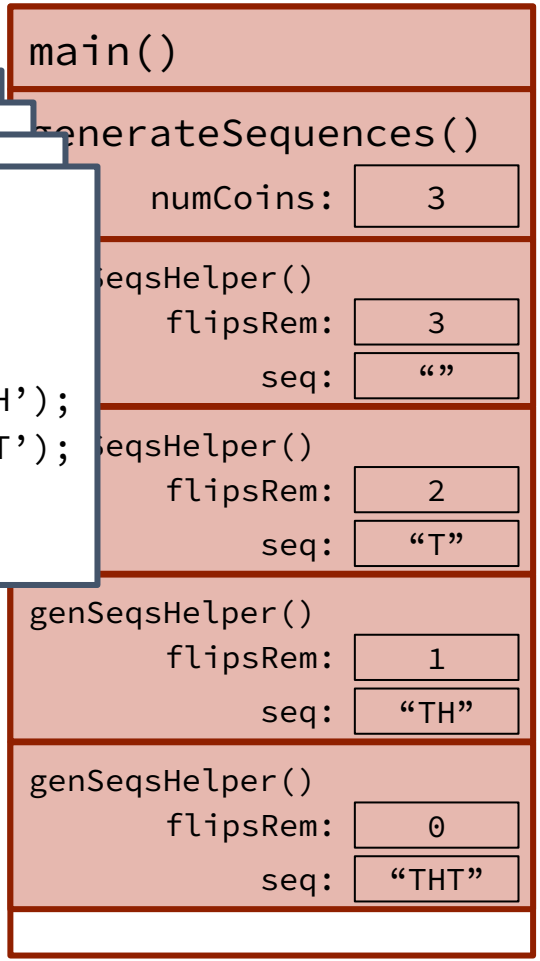
```
HHH
HHT
HTH
HTT
THH
THT
```



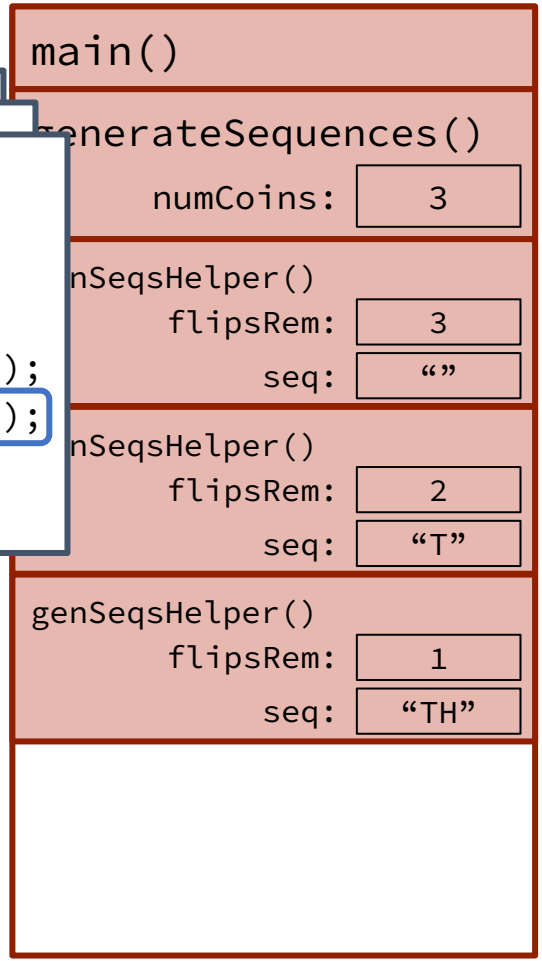
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

```
HHH
HHT
HTH
HTT
THH
THT
```



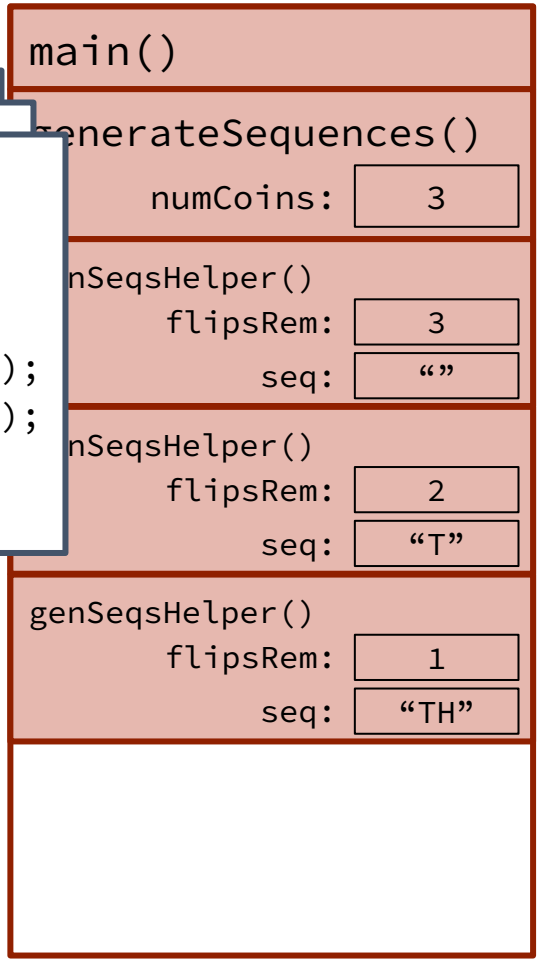
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
THH
THT
```

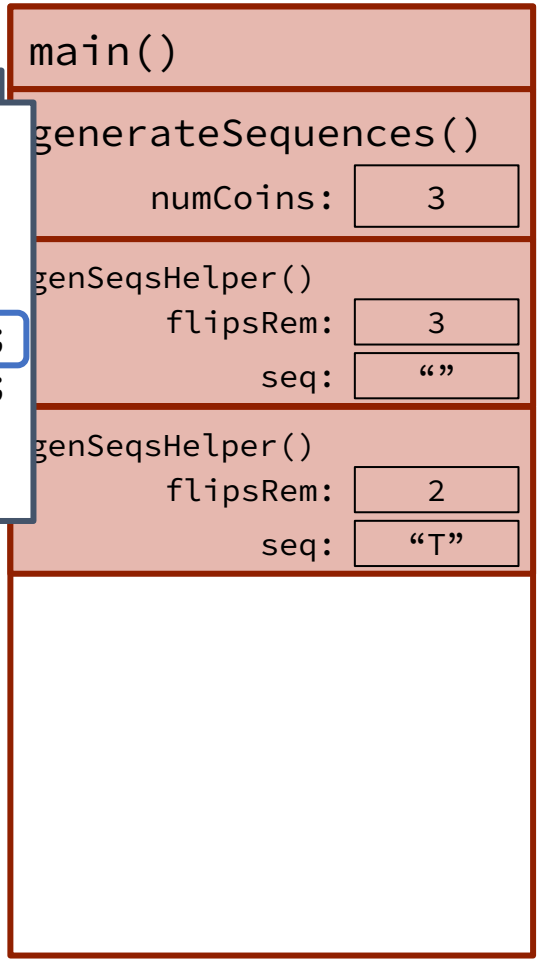
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
THH
THT
```

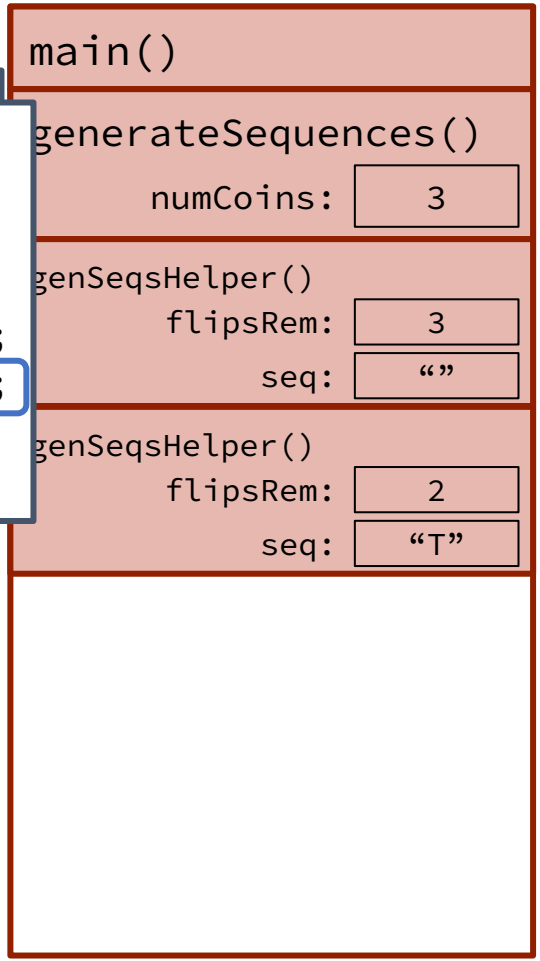
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
THH
THT
```

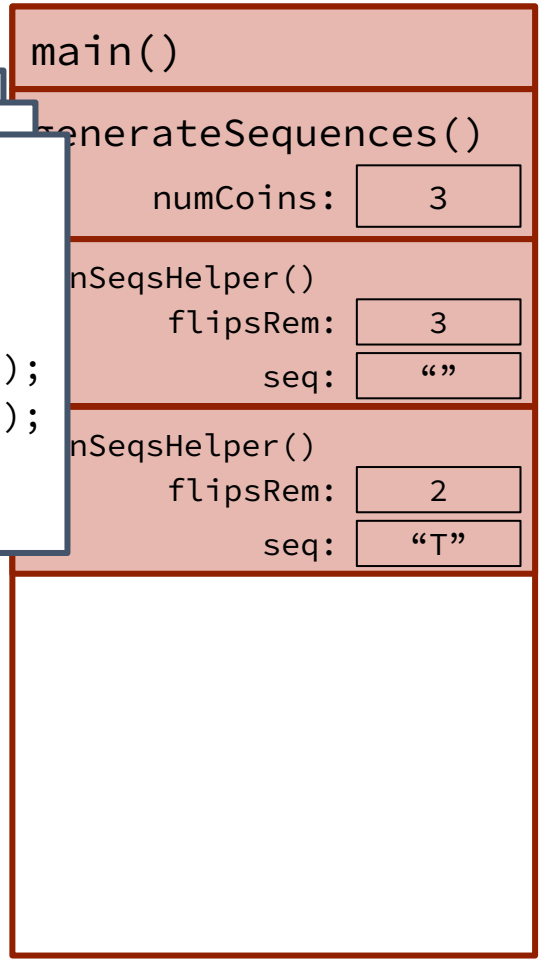
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
THH
THT
```

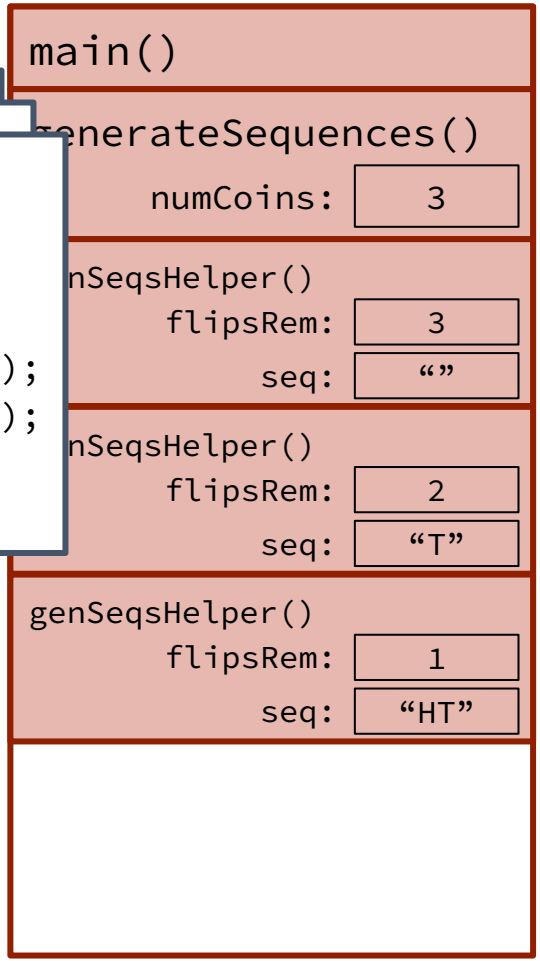
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
THH
THT
```

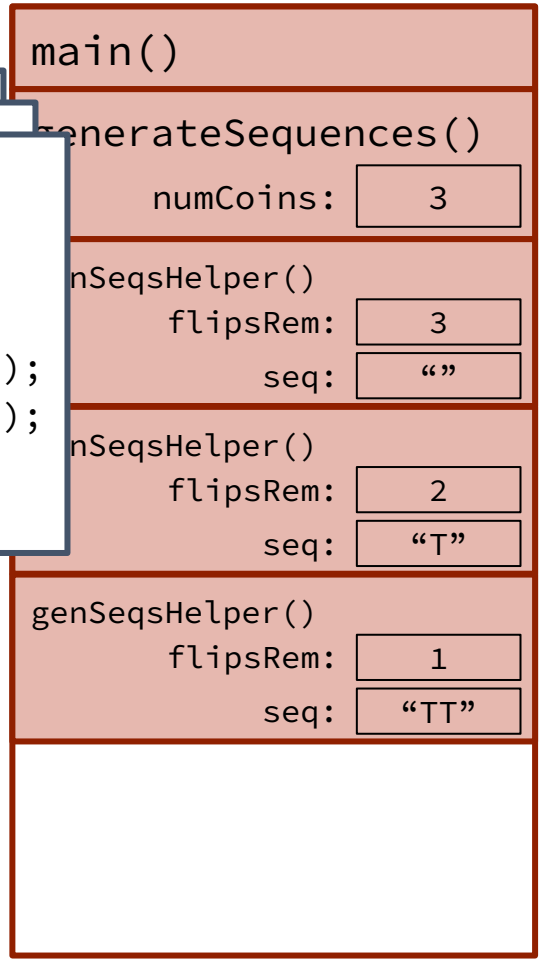
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
THH
THT
```

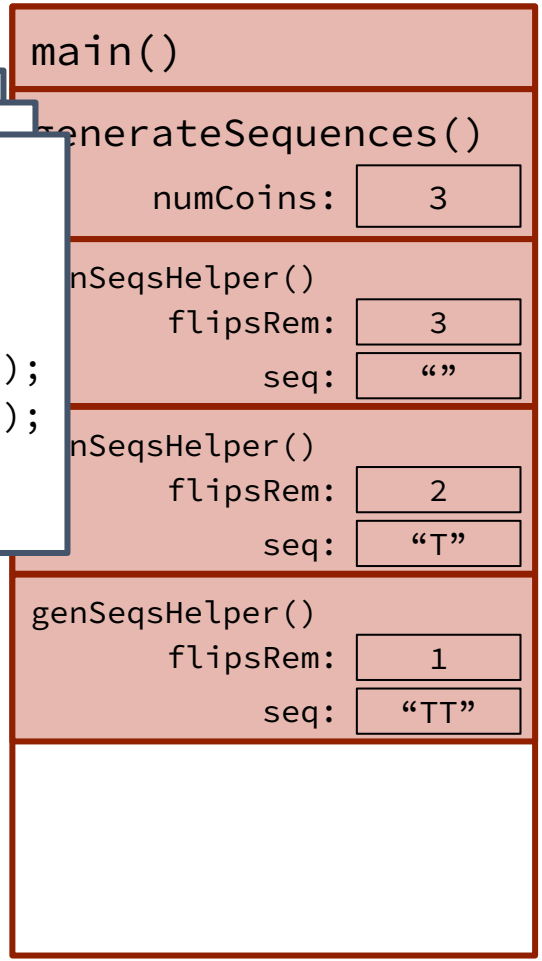
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
THH
THT
```

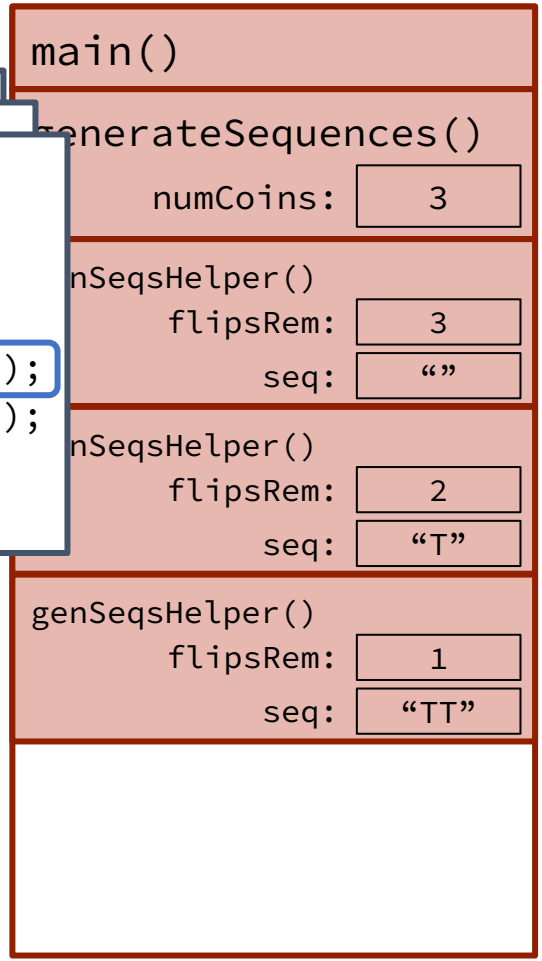
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
THH
THT
```

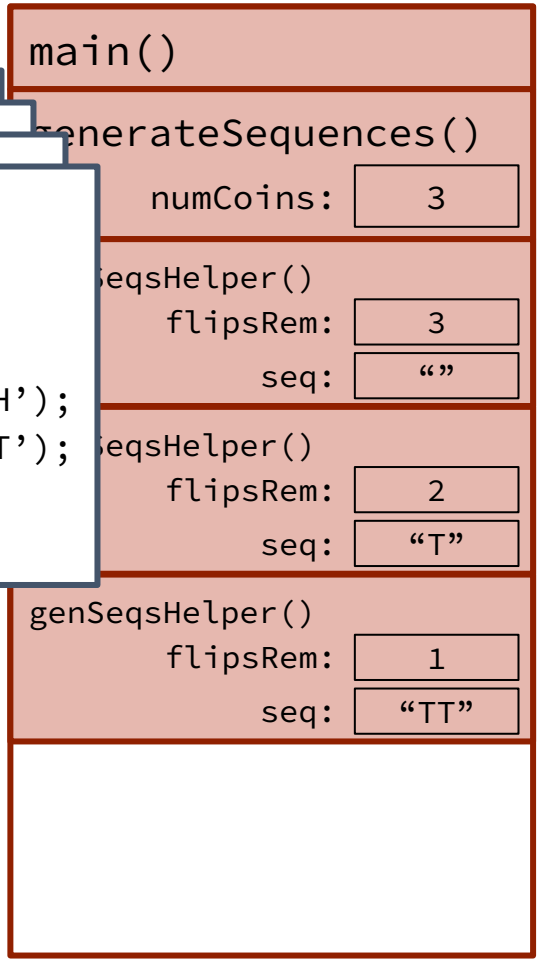
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
THH
THT
```

```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



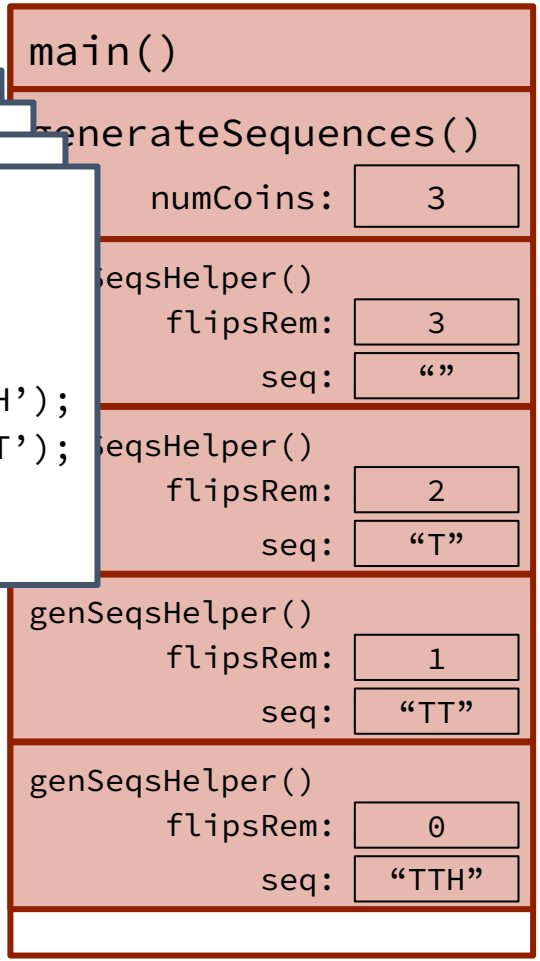
Console:

```
HHH
HHT
HTH
HTT
THH
THT
```

```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

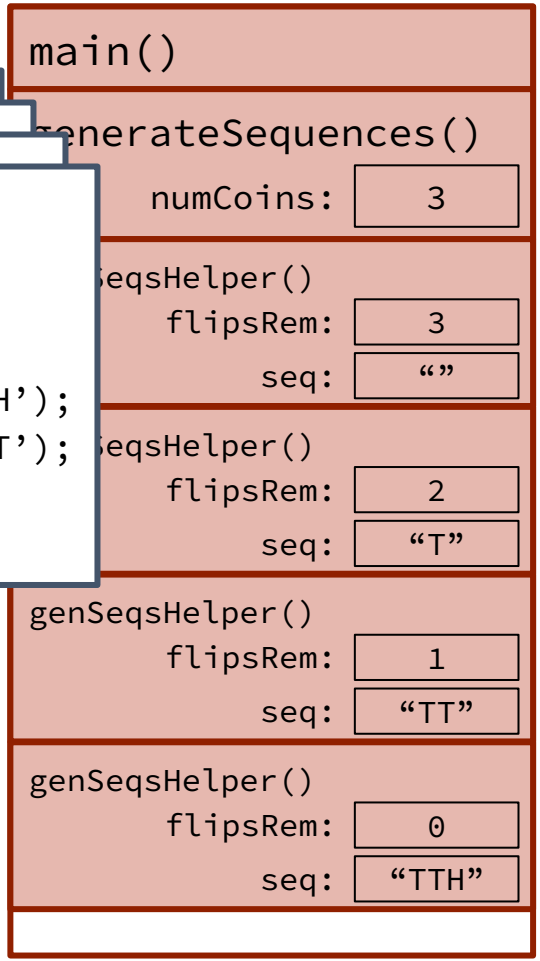
```
HHH
HHT
HTH
HTT
THH
THT
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

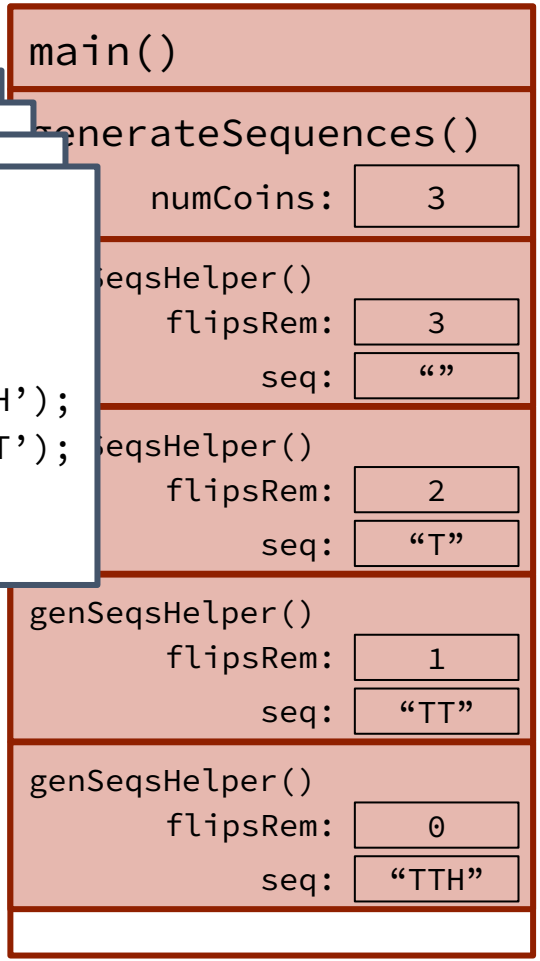
```
HHH
HHT
HTH
HTT
THH
THT
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

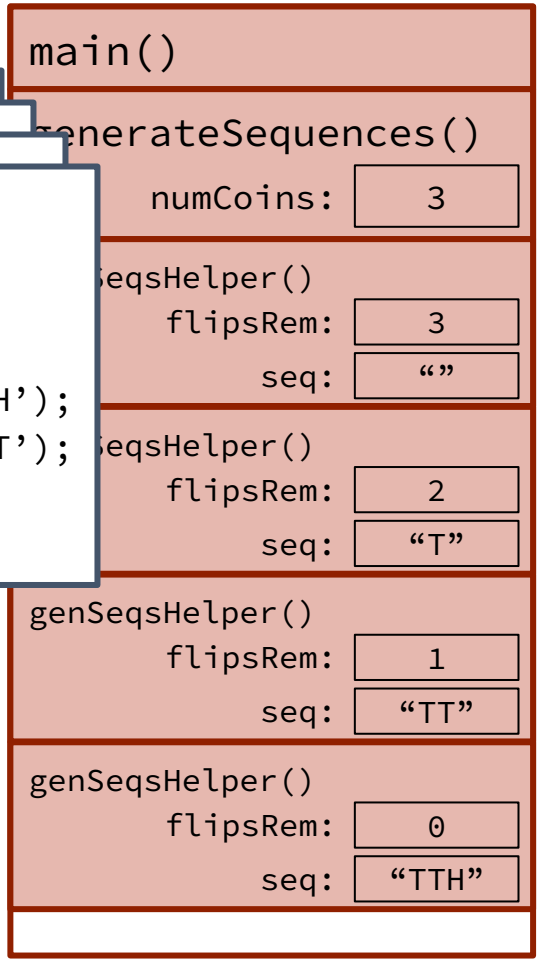
```
HHH
HHT
HTH
HTT
THH
THT
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

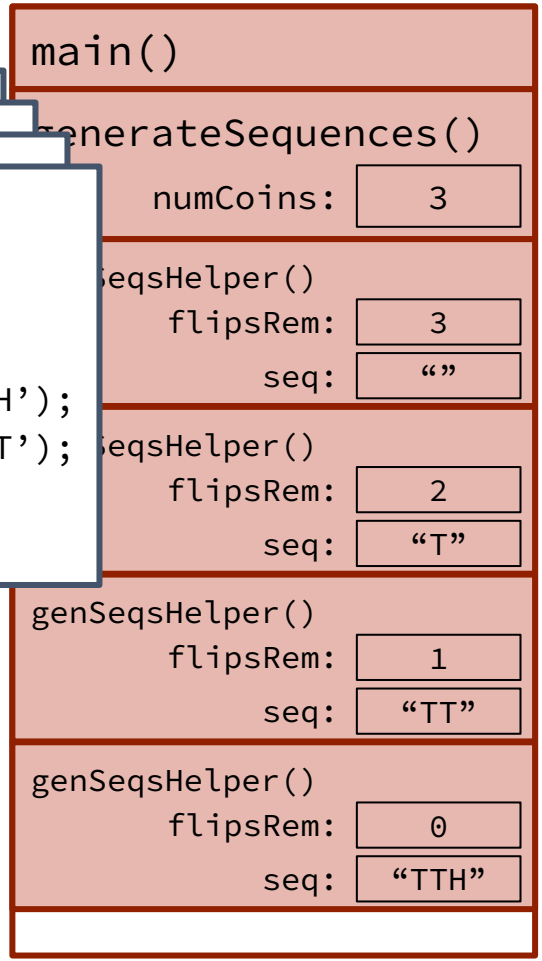
```
HHH
HHT
HTH
HTT
THH
THT
TTH
```



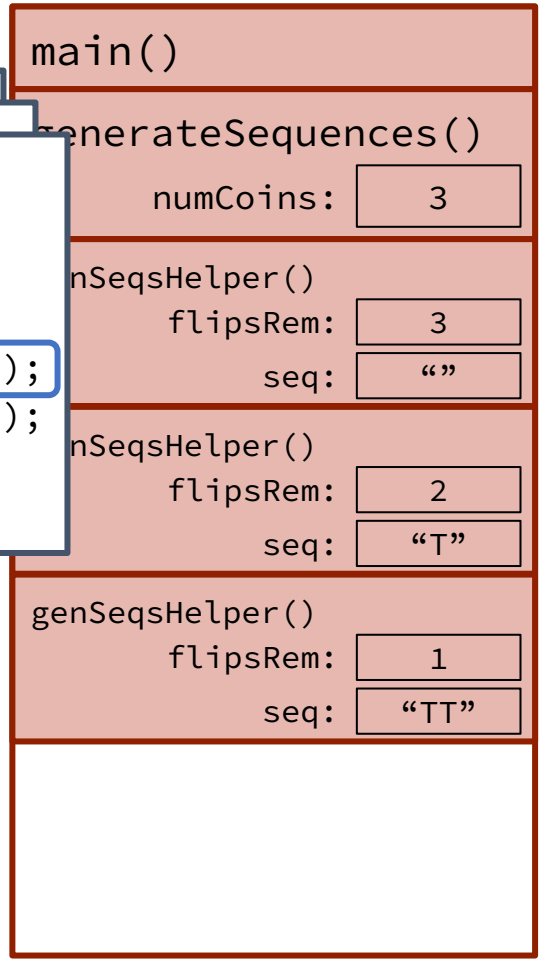
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

```
HHH
HHT
HTH
HTT
THH
THT
TTH
```



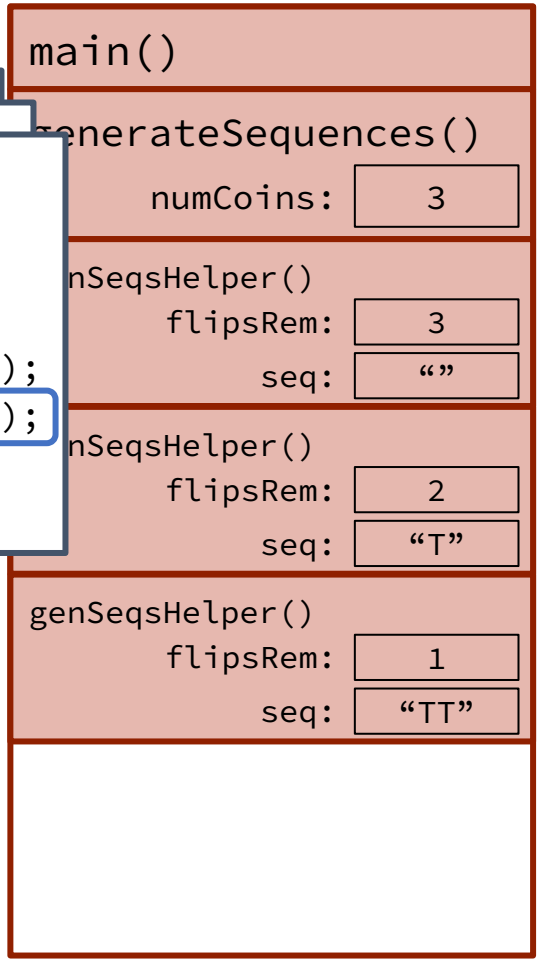
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
THH
THT
TTH
```

```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



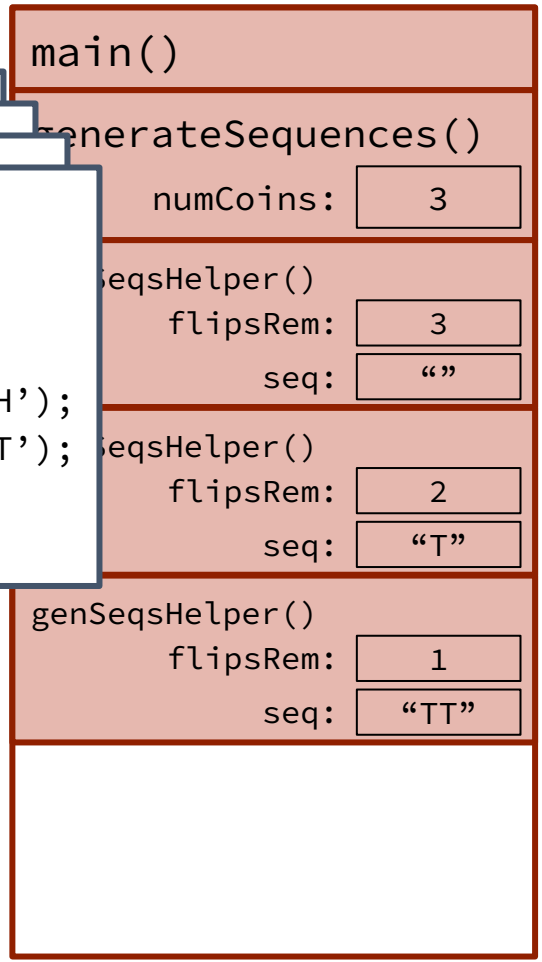
Console:

```
HHH
HHT
HTH
HTT
THH
THT
TTH
```

```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

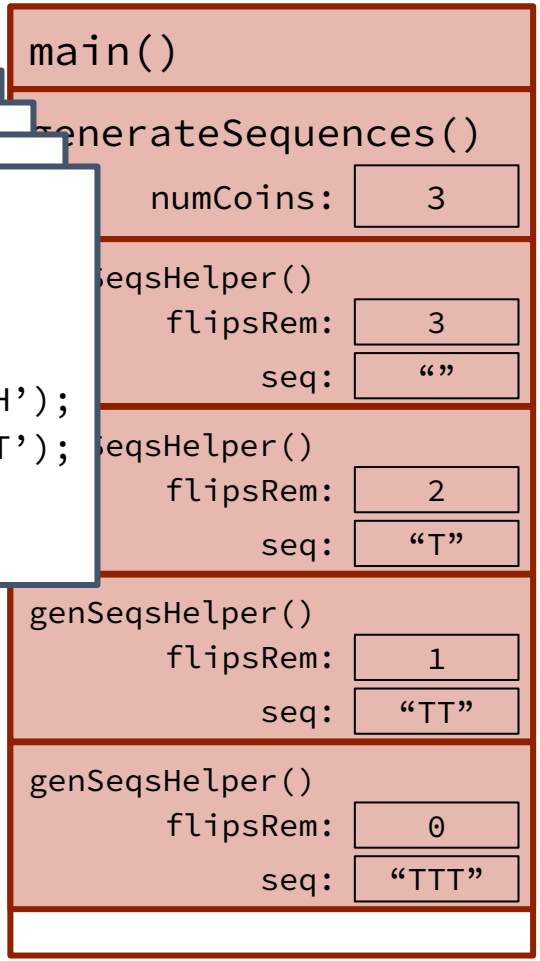
```
HHH
HHT
HTH
HTT
THH
THT
TTH
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

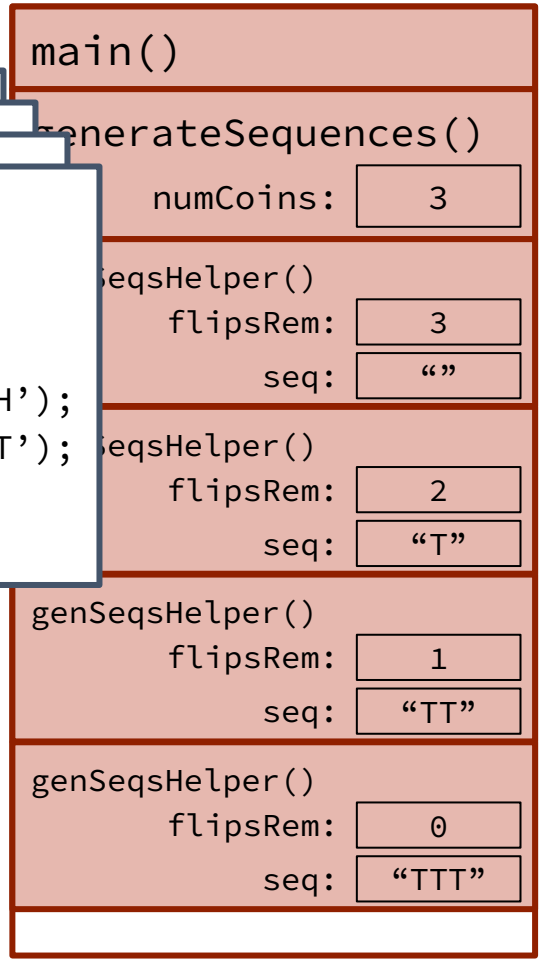
```
HHH
HHT
HTH
HTT
THH
THT
TTH
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

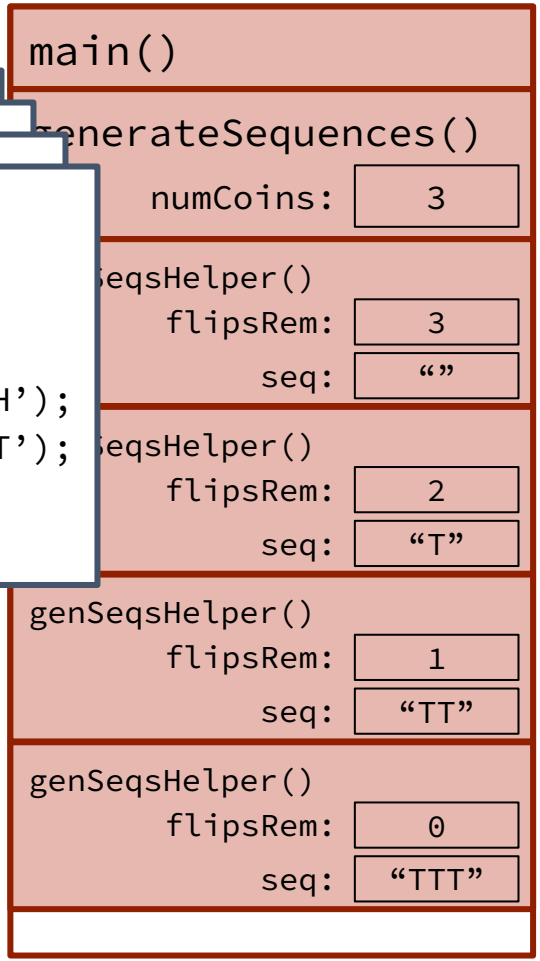
```
HHH
HHT
HTH
HTT
THH
THT
TTH
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

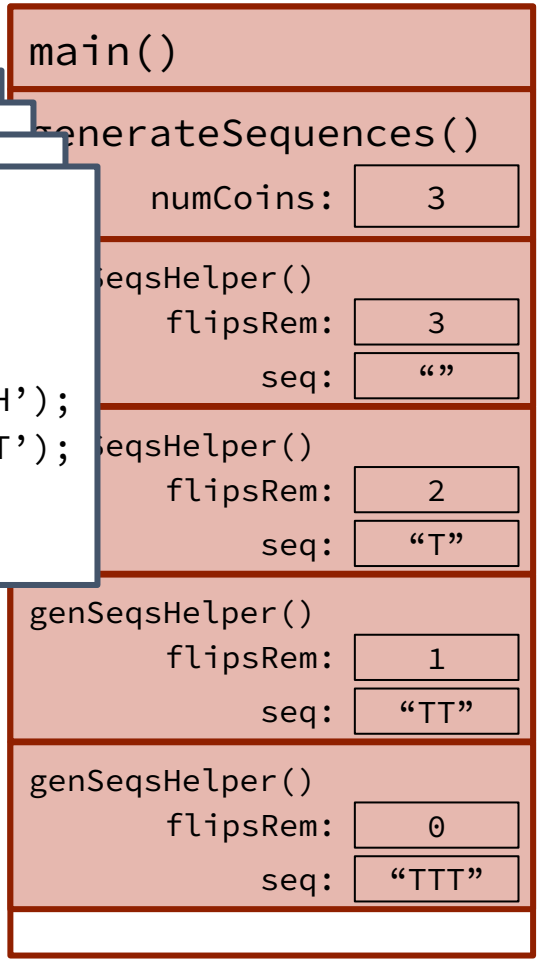
```
HHH
HHT
HTH
HTT
THH
THT
TTH
```



```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

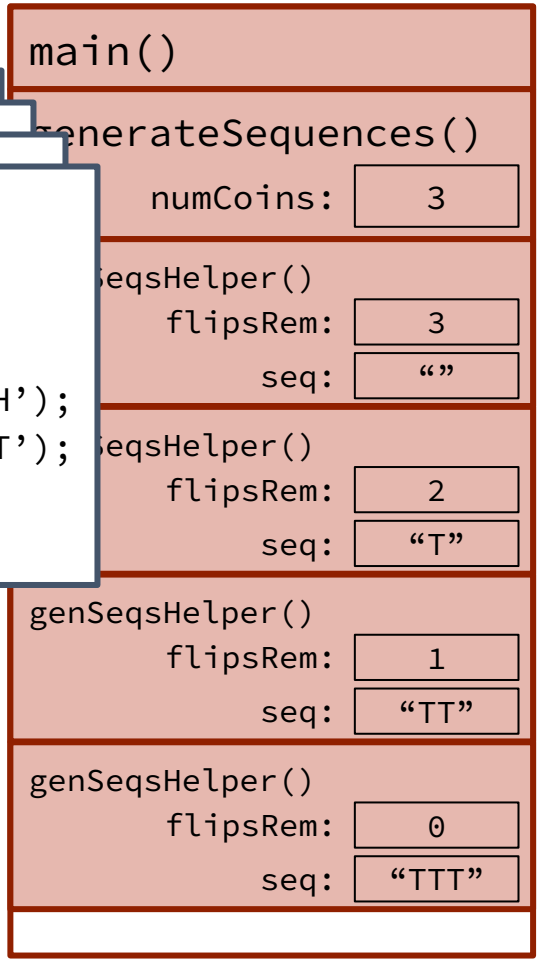
```
HHH
HHT
HTH
HTT
THH
THT
TTH
TTT
```



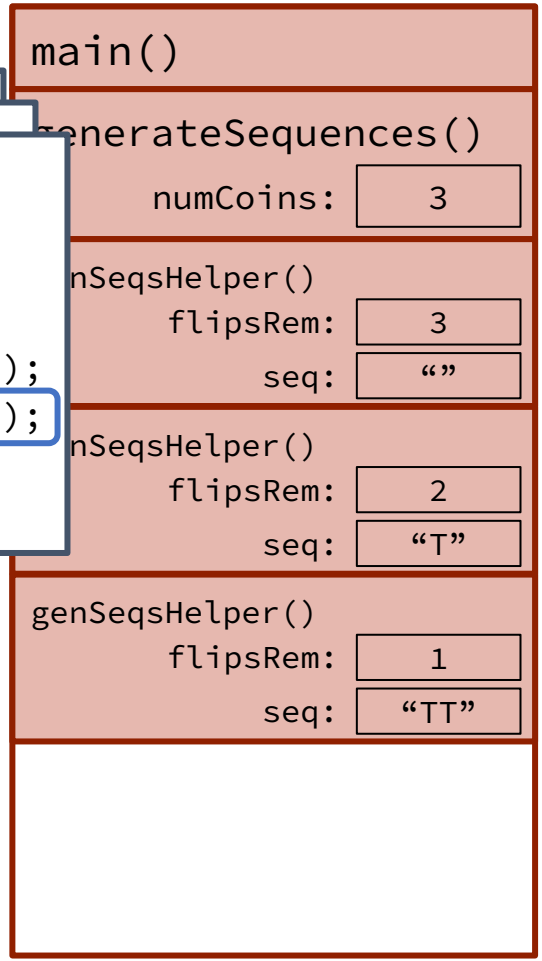
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```

Console:

```
HHH
HHT
HTH
HTT
THH
THT
TTH
TTT
```



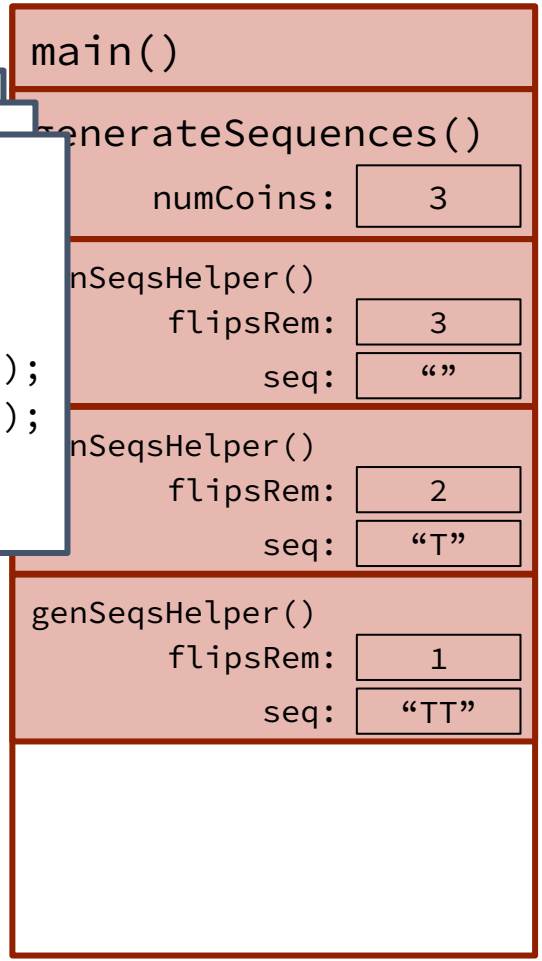
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
THH
THT
TTH
TTT
```

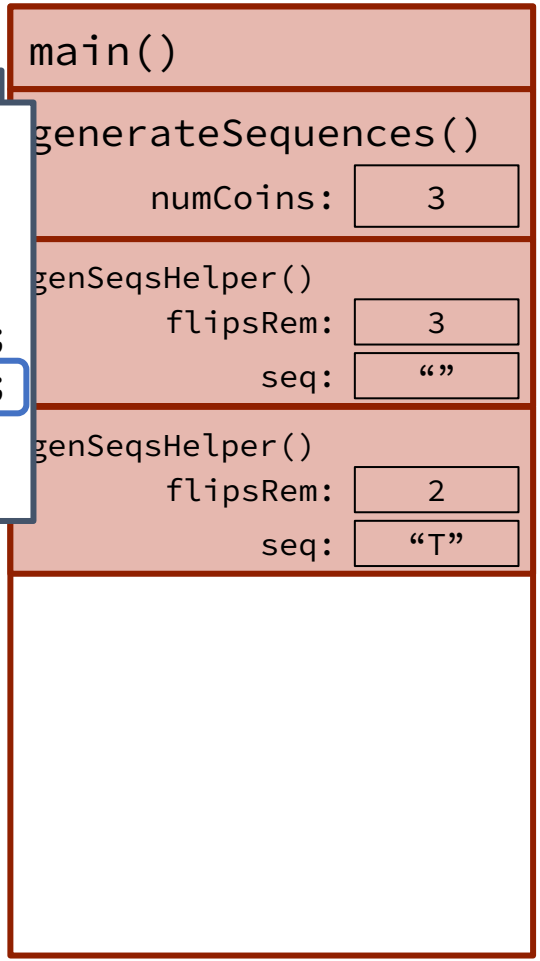
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
THH
THT
TTH
TTT
```

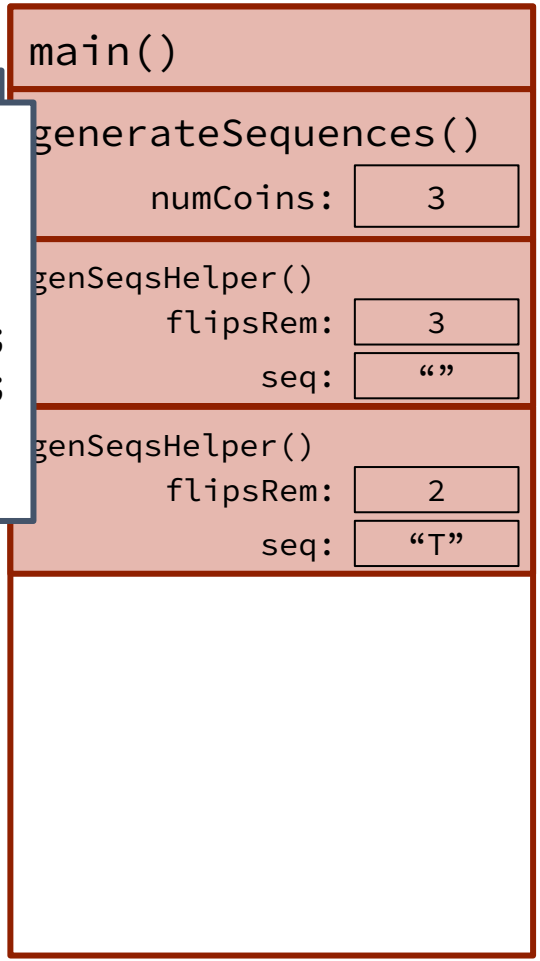
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
THH
THT
TTH
TTT
```

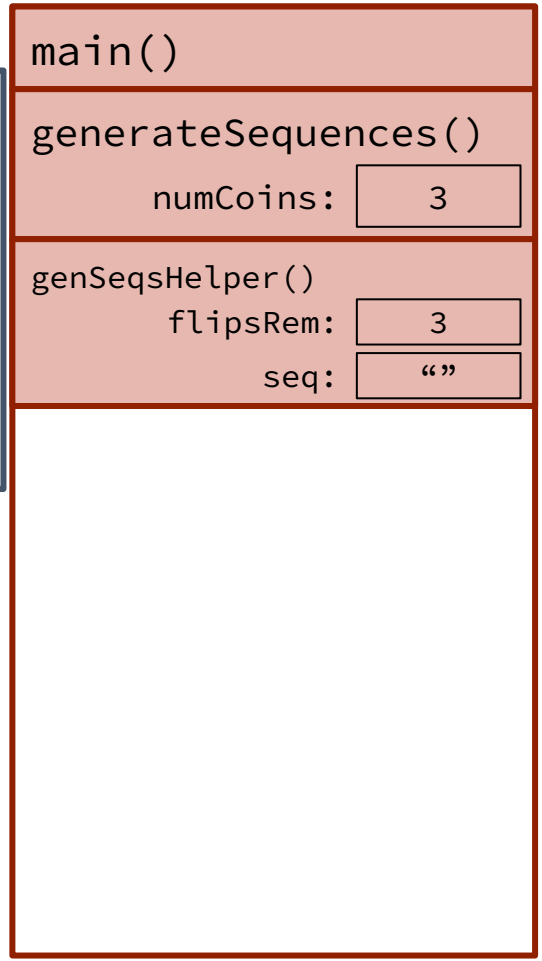
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
THH
THT
TTH
TTT
```

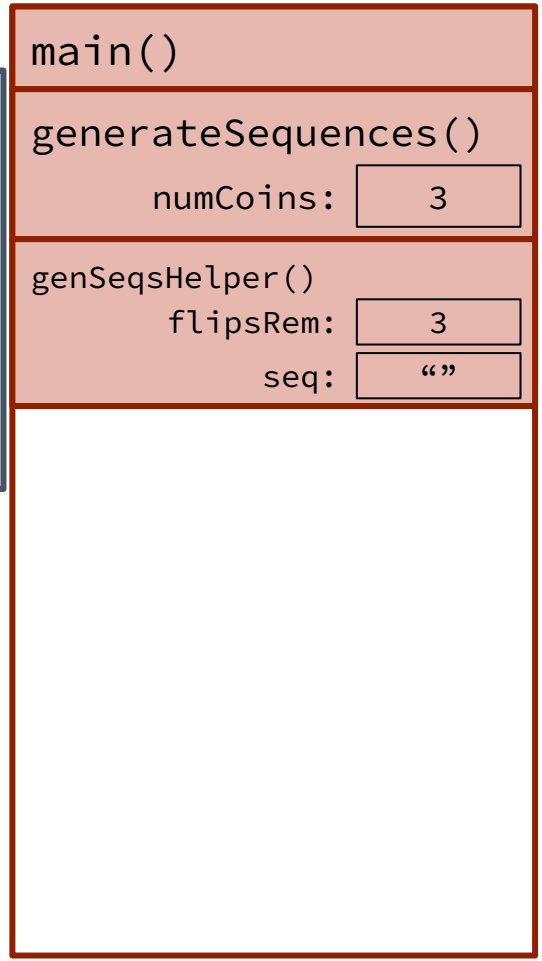
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
THH
THT
TTH
TTT
```

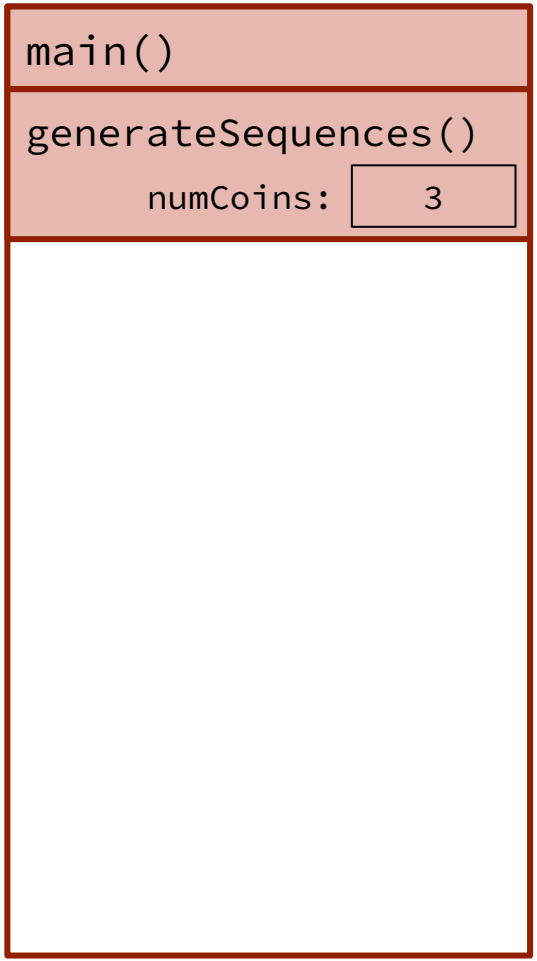
```
void genSeqsHelper (int flipsRem, string seq) {
    if (flipsRem == 0) {
        cout << seq << endl;
    } else {
        generateSeqsHelper(flipsRem-1, seq + 'H');
        generateSeqsHelper(flipsRem-1, seq + 'T');
    }
}
```



Console:

```
HHH
HHT
HTH
HTT
THH
THT
TTH
TTT
```

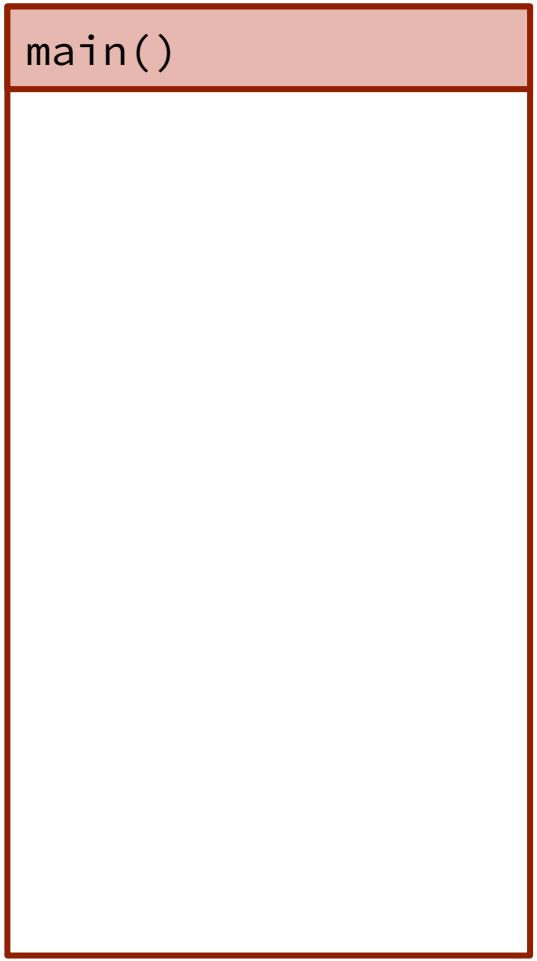
```
void generateSequences (int numCoins) {
    genSeqsHepler(numCoins, “”);
}
```



Console:

```
HHH
HHT
HTH
HTT
THH
THT
TTH
TTT
```

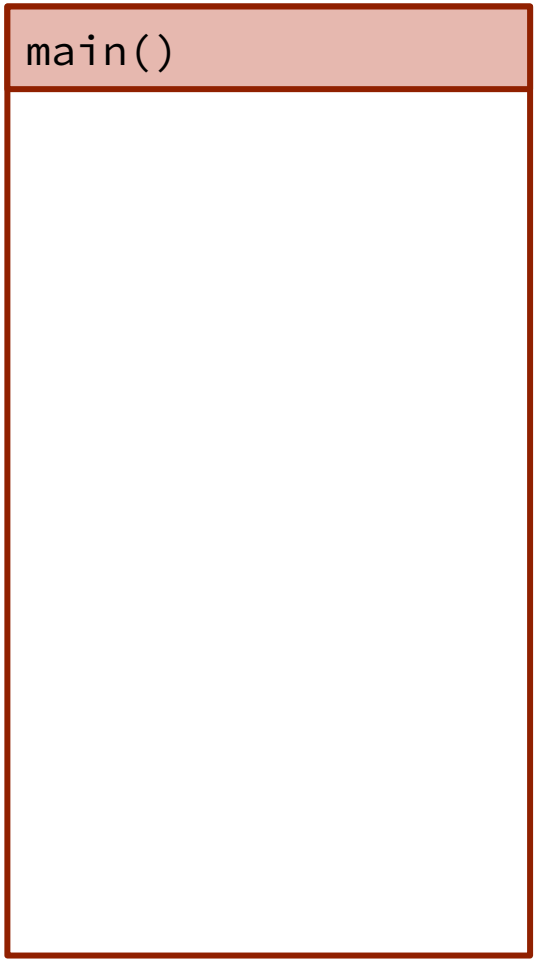
```
int main () {
    generateSequences(3);
    return 0;
}
```



Console:

```
HHH
HHT
HTH
HTT
THH
THT
TTH
TTT
```

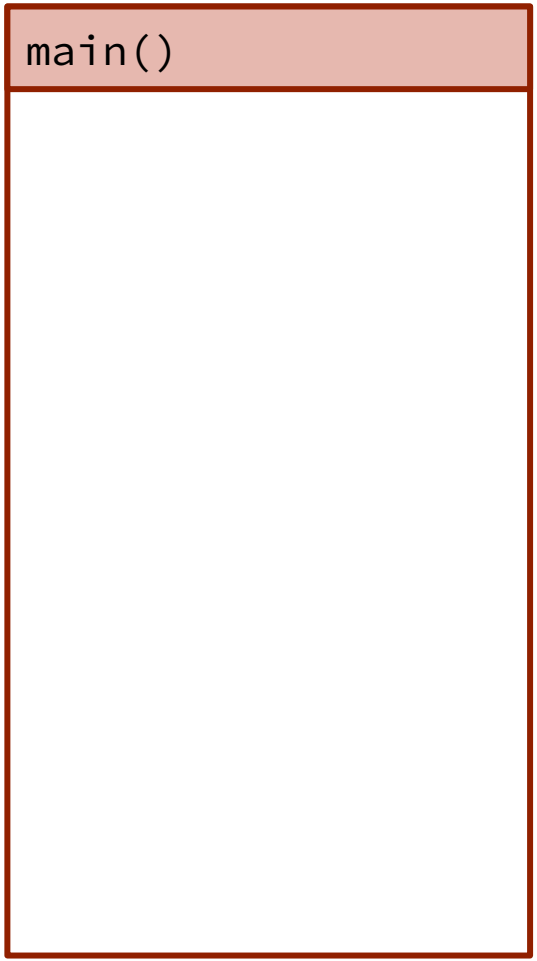
```
int main () {
    generateSequences(3);
    return 0;
}
```



Console:

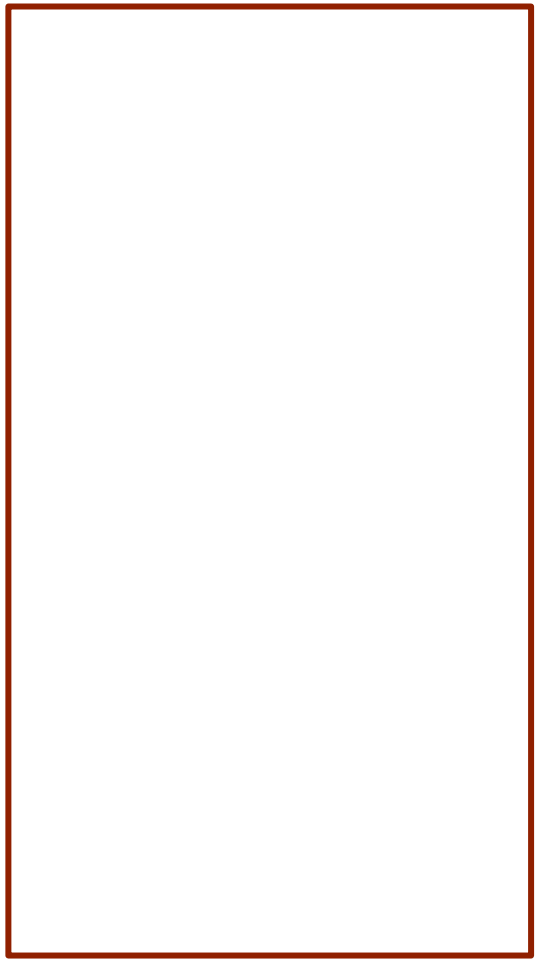
```
HHH
HHT
HTH
HTT
THH
THT
TTH
TTT
```

```
int main () {
    generateSequences(3);
    return 0;
}
```



Console:

```
HHH
HHT
HTH
HTT
THH
THT
TTH
TTT
```

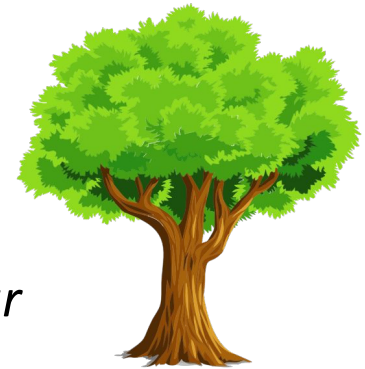


Console:

```
HHH
HHT
HTH
HTT
THH
THT
TTH
TTT
```

3 Problems to Solve with Backtracking

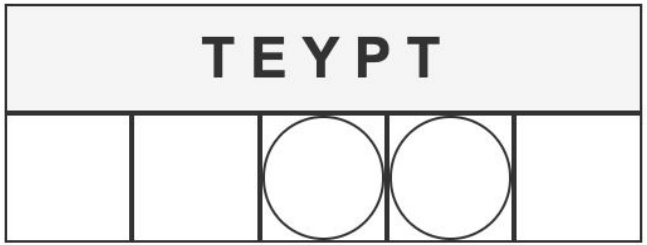
1. Generate all solutions to a problem or count number of solutions
2. Find one specific solution or prove that one exists
3. Find the best possible solution to a problem



All of these involve exploring many possible solutions, rather than proceeding down a linear path towards one solution.

Word Jumble

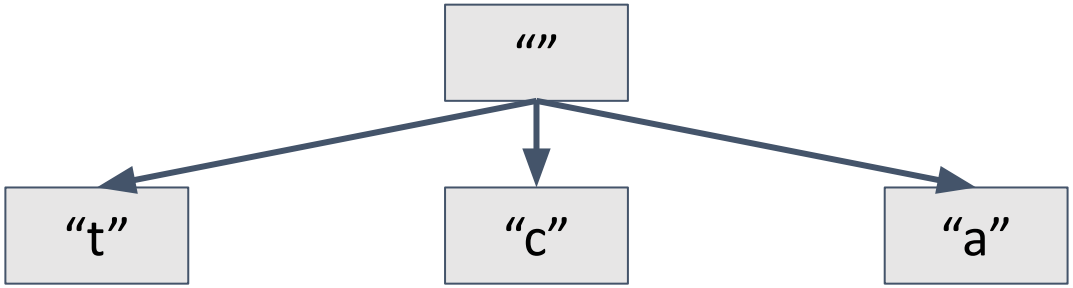
- We'd like to print every ordering of "TEYPT" to solve the puzzle
- This is much like coin sequences, but instead of choosing H or T, we are choosing a letter at each step



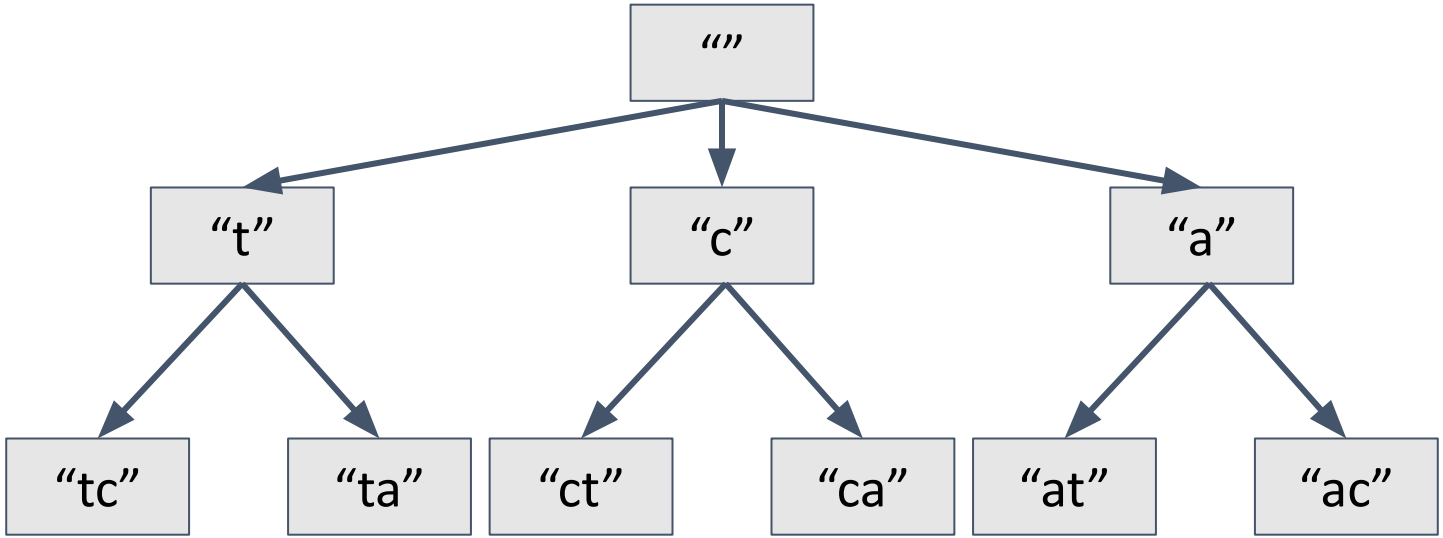
Word Jumble - ‘tca’

“”

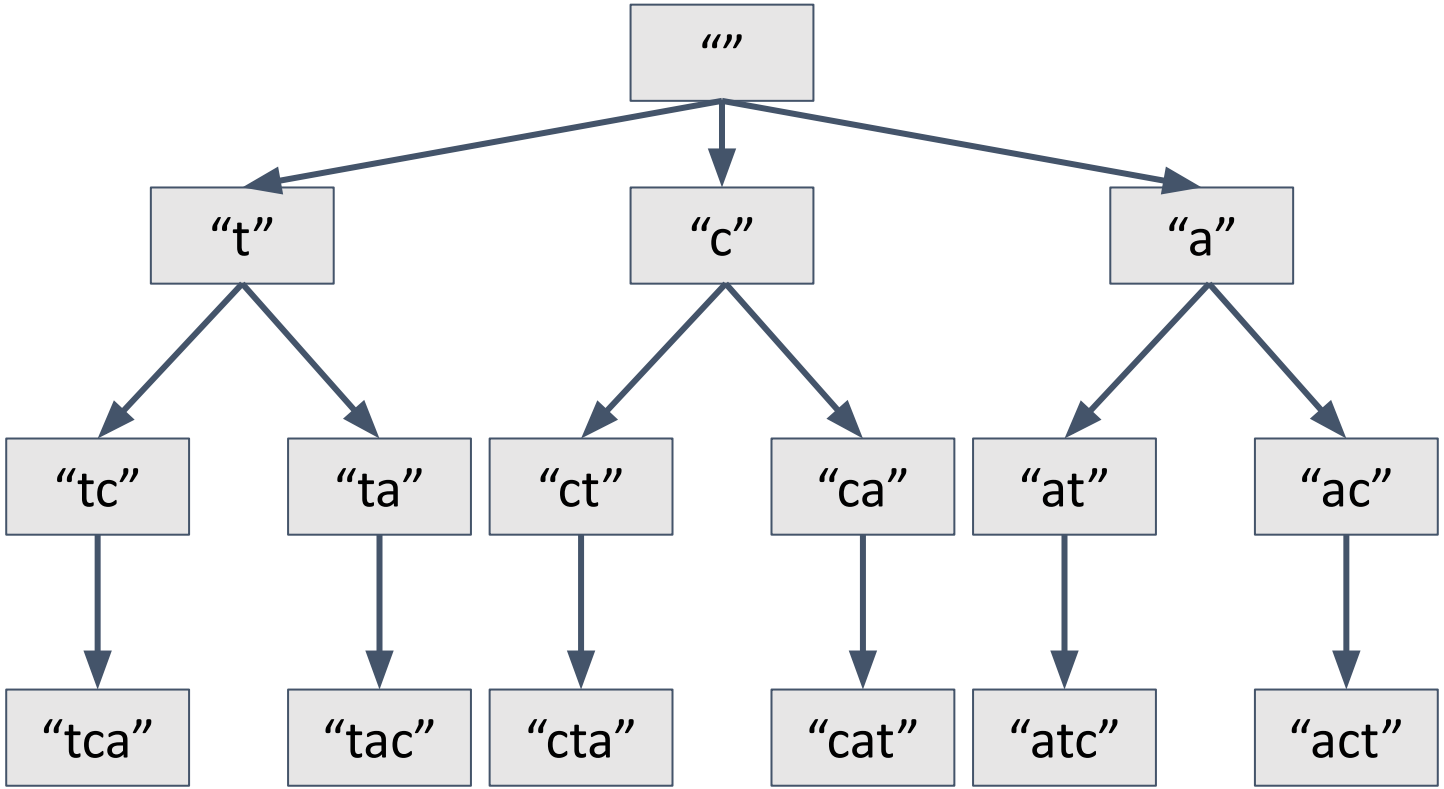
Word Jumble - ‘tca’



Word Jumble - ‘tca’

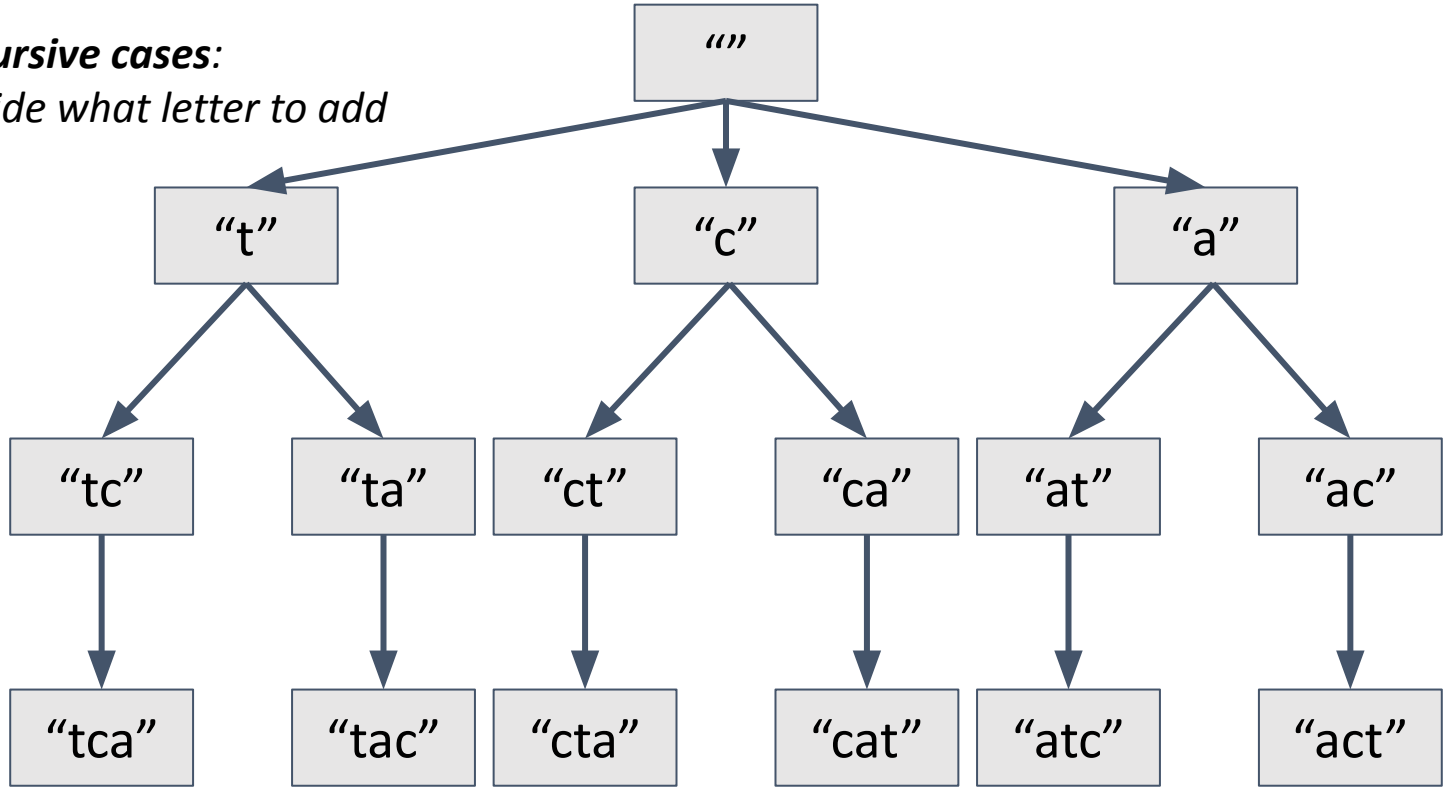


Word Jumble - ‘tca’



Word Jumble - ‘tca’

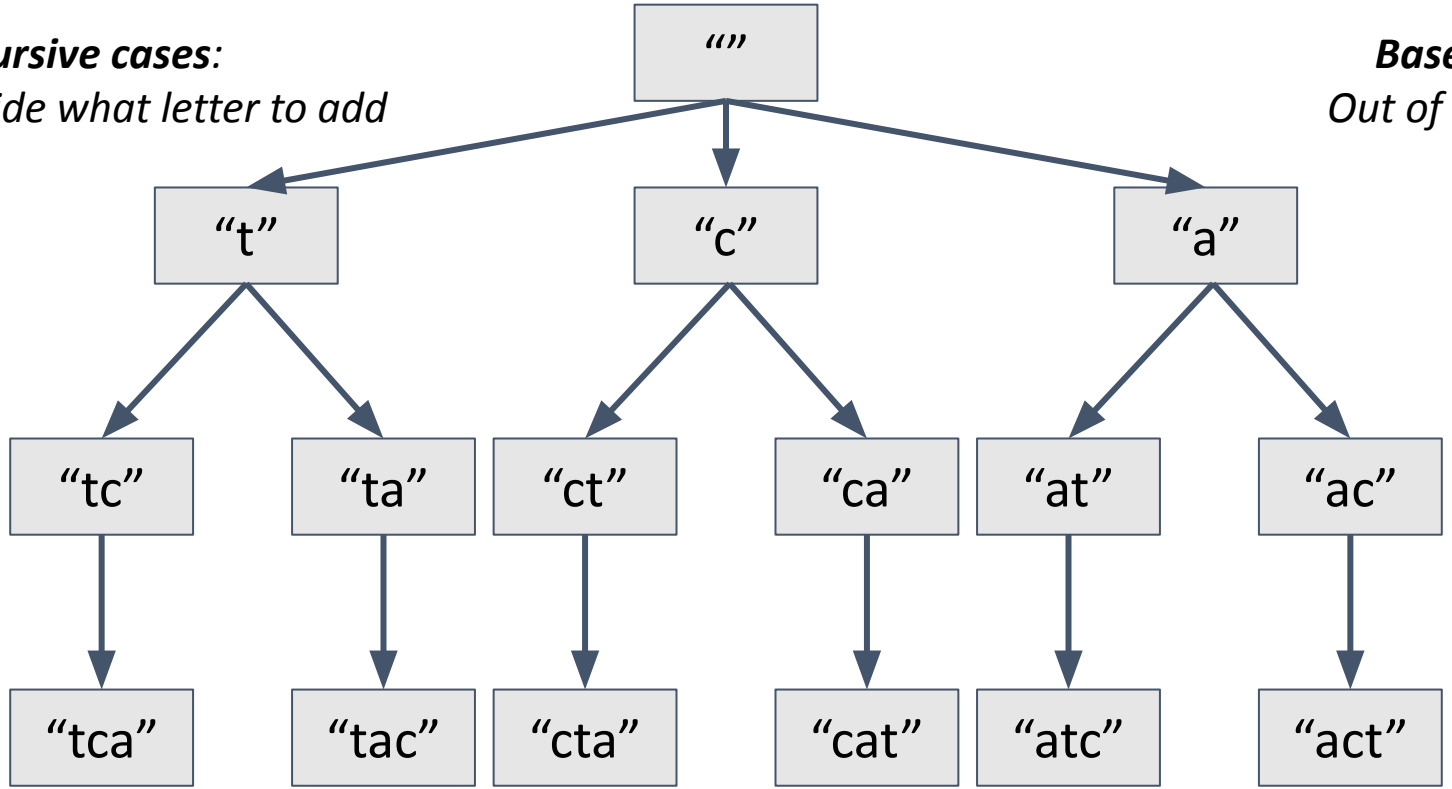
***Recursive cases**:*
Decide what letter to add



Word Jumble - ‘tca’

Recursive cases:
Decide what letter to add

Base case:
Out of letters



Permutations Solution Code

```
void generatePermutationsHelper(string lettersRemaining, string sequence) {
    // Base case: lettersRemaining = 0, no more letters to choose from
    if (lettersRemaining.length() == 0) {
        cout << sequence << endl;
    } else {
        // Many recursive cases (when lettersRemaining > 0)
        for (int i = 0; i < lettersRemaining.length(); i++) {
            char letter = lettersRemaining[i];  // choose one of our remaining letters to build on sequence
            generatePermutationsHelper(lettersRemaining.substr(0, i) + lettersRemaining.substr(i + 1),
                                       sequence + letter);
        }
    }
}

void generatePermutations(string word) {
    generatePermutationsHelper(word, "");
}
```

Takeaways

- "Choose / explore / unchoose" pattern in backtracking

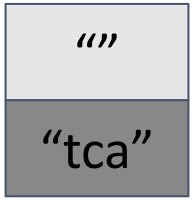
```
for (int i = 0; i < lettersRemaining.length(); i++) {
    // choose a letter
    char letter = lettersRemaining[i];

    // explore this choice by making a recursive call
    generatePermutationsHelper(lettersRemaining.substr(0, i) +
                               lettersRemaining.substr(i + 1), sequence + letter)

    // unchoose this letter by not including it in our sequence next loop
}
```

Takeaways

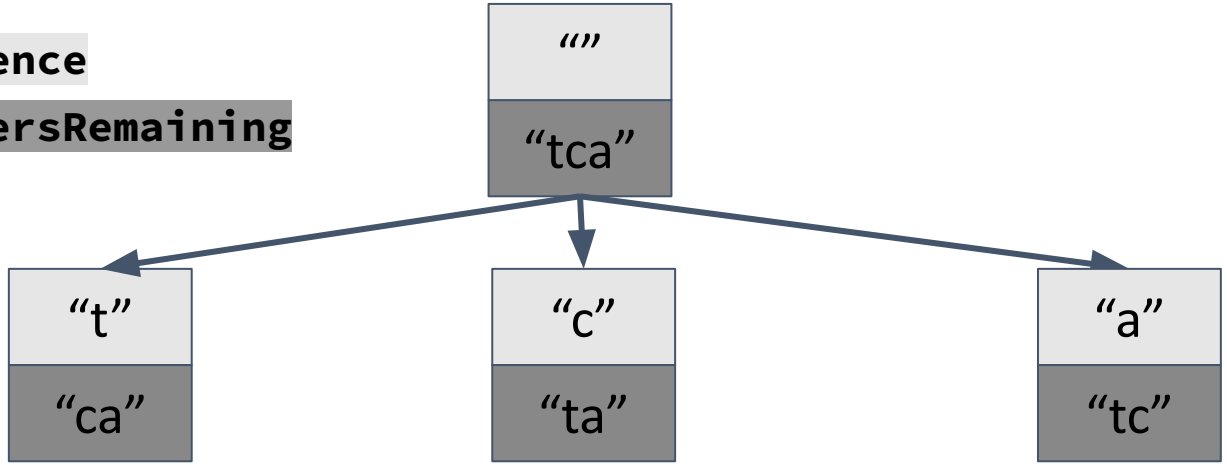
- "Choose / explore / unchoose" pattern in backtracking
- It is important to keep track of the decisions we've made so far and the decisions we have left to make

```
void generatePermutationsHelper(string lettersRemaining, string sequence) {
```

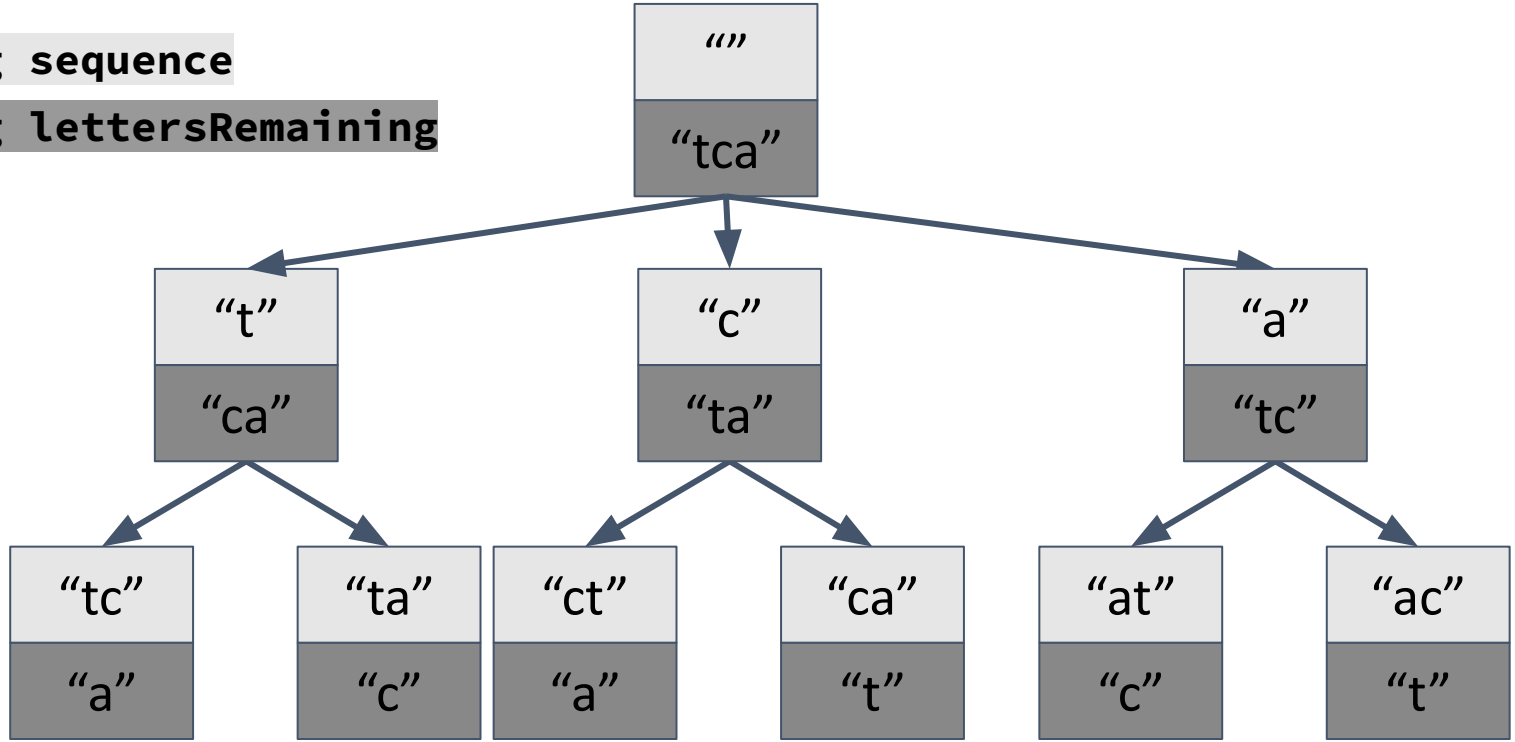
```
string sequence
string lettersRemaining
```



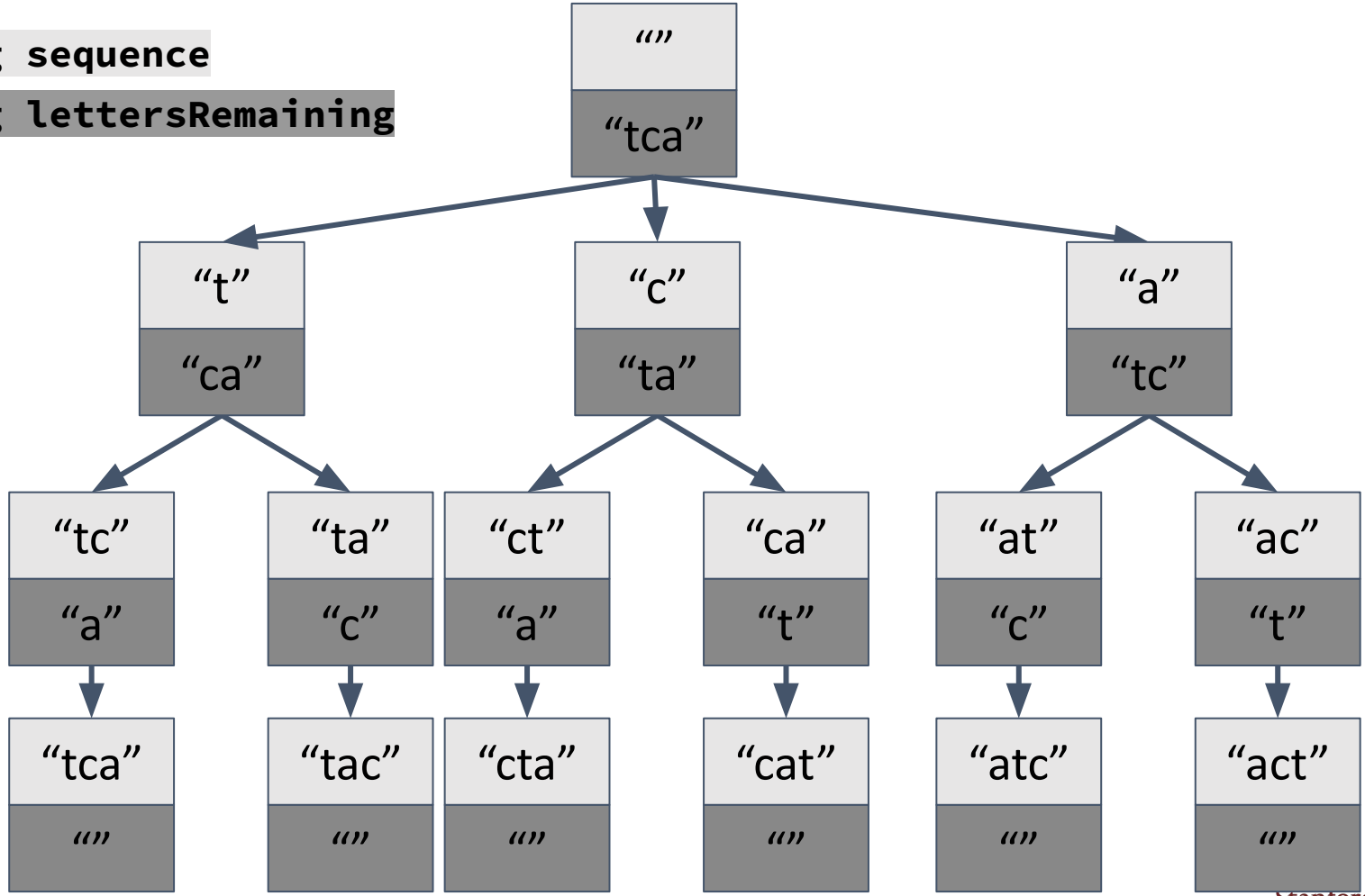
```
string sequence
string lettersRemaining
```



```
string sequence
string lettersRemaining
```



```
string sequence
string lettersRemaining
```



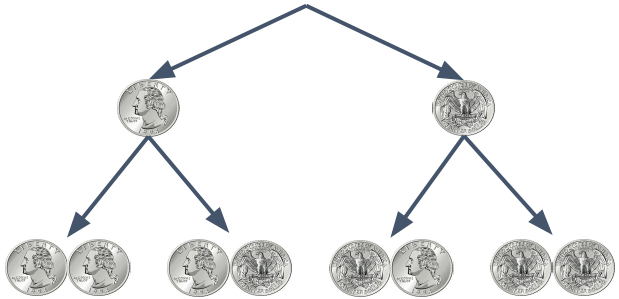
Takeaways

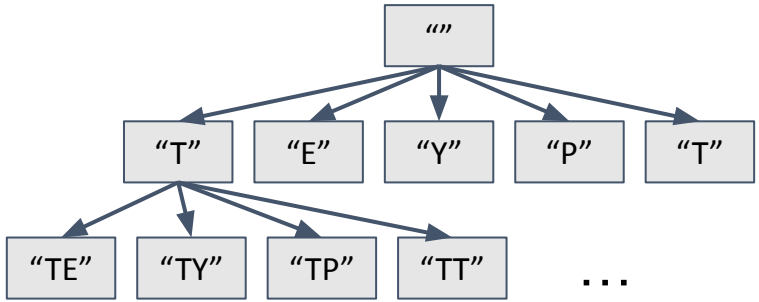
- "Choose / explore / unchoose" pattern in backtracking
- It is important to keep track of the decisions we've made so far and the decisions we have left to make

```
void generatePermutationsHelper(string lettersRemaining, string sequence) {
```

Takeaways

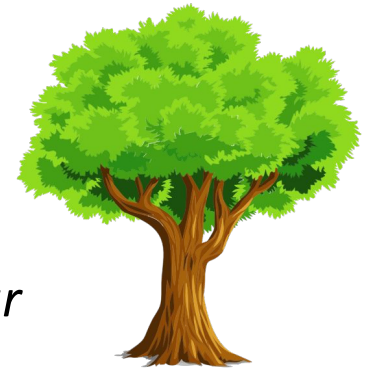
- "Choose / explore / unchoose" pattern in backtracking
- It is important to keep track of the decisions we've made so far and the decisions we have left to make
- Backtracking recursion can have variable branching factors at each level





3 Problems to Solve with Backtracking

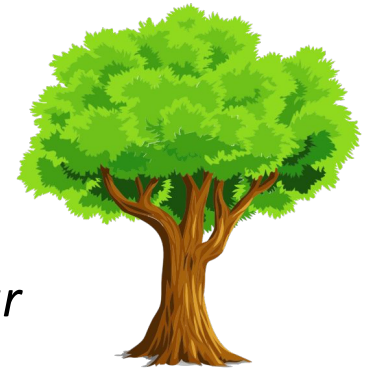
1. **Generate all solutions to a problem or count number of solutions**
2. Find one specific solution or prove that one exists
3. Find the best possible solution to a problem



All of these involve exploring many possible solutions, rather than proceeding down a linear path towards one solution.

3 Problems to Solve with Backtracking

1. Generate all solutions to a problem or count number of solutions
2. **Find one specific solution or prove that one exists**
3. Find the best possible solution to a problem

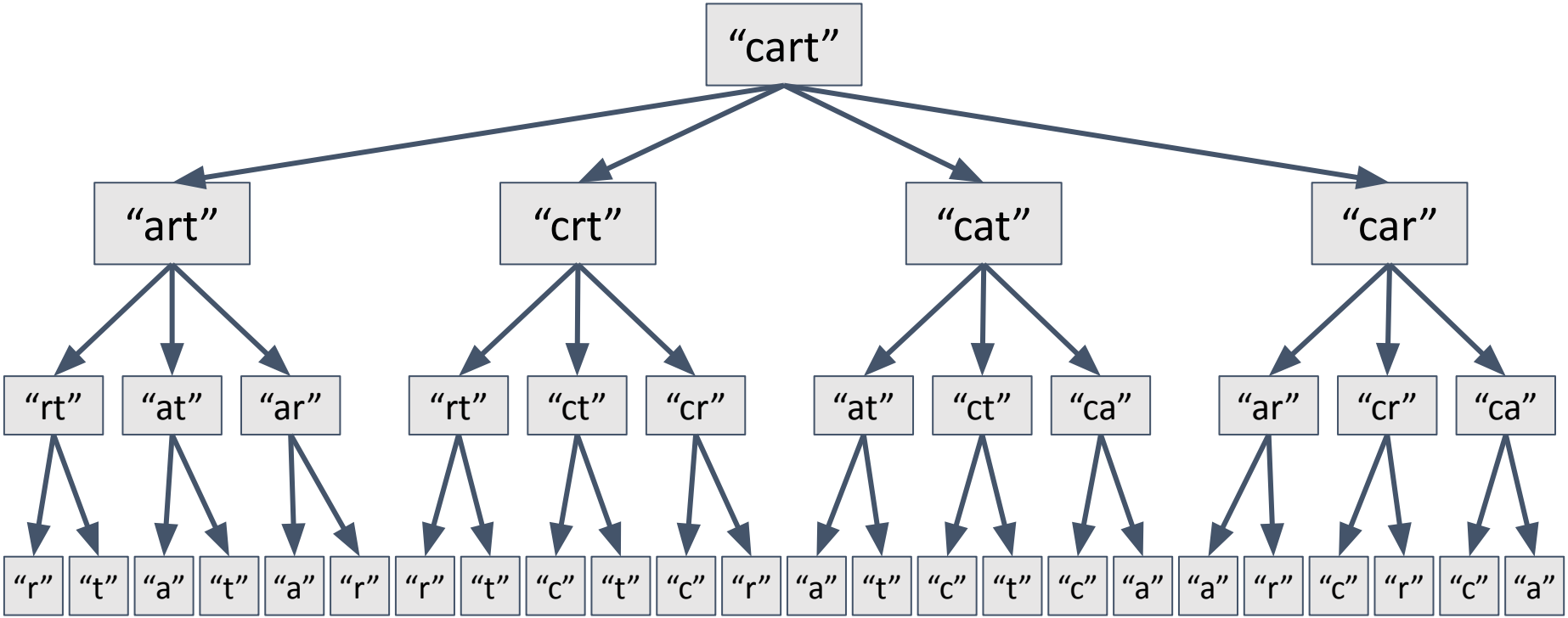


All of these involve exploring many possible solutions, rather than proceeding down a linear path towards one solution.

Shrinkable Words

- A *shrinkable word* is a word that can be reduced down to one letter by removing one letter at a time, leaving a valid word at each step
- Idea: Let's use a decision tree to remove letters and determine if a word is shrinkabe!

Shrinkable Words Decision Tree



Shrinkable Words

Base cases:

- We reach an invalid word (failure)
- We get down to a single-letter word (success)

Recursive cases:

- The word is shrinkable if you can remove any letter and get a shrinkable word
- The word is not shrinkable if no matter what letter you remove, it's not shrinkable

Lexicon

How do we check if a word is valid? We have an ADT for that:

#include “lexicon.h” (documentation here)

```
Lexicon lex("res/EnglishWords.txt"); // create from file
lex.contains("koala"); // returns true
lex.contains("zzzzz"); // returns false
// returns true if there are any words starting with "fi" in the lexicon
lex.containsPrefix("fi");
```

Solution

```
bool isShrinkable(Lexicon& lex, string word) {
    // base case 1) reach invalid word 2) reach final letter
    if (!lex.contains(word)) {
        return false;
    }
    if (word.length() == 1) {
        return true;
    }
    // recursive case: try removing every letter and if any succeeds, return true
    for (int i = 0; i < word.length(); i++) {
        string remainingWord = word.substr(0, i) + word.substr(i + 1);
        if (isShrinkable(lex, remainingWord)) {
            return true;
        }
    }
    return false;
}
```

Subsets

Subsets

Given a group of people, generate all possible teams, or subsets, of these people:

Subsets

Given a group of people, generate all possible teams, or subsets, of these people:

```
{}
{“Amrita”}
{“Elyse”}
{“Taylor”}
{“Amrita”, “Elyse”}
{“Amrita”, “Taylor”}
{“Elyse”, “Taylor”}
{“Amrita”, “Elyse”, “Taylor”}
```

Subsets

Given a group of people, generate all possible teams, or subsets, of those people:

```
{}
{“Amrita”}
{“Elyse”}
{“Taylor”}
{“Amrita”, “Elyse”}
{“Amrita”, “Taylor”}
{“Elyse”, “Taylor”}
{“Amrita”, “Elyse”, “Taylor”}
```

***Recursive Backtracking**: Generate all solutions to a problem*

Subsets

Given a group of people, generate all possible teams, or subsets, of those people:

```
{}
{“Amrita”}
{“Elyse”}
{“Taylor”}
{“Amrita”, “Elyse”}
{“Amrita”, “Taylor”}
{“Elyse”, “Taylor”}
{“Amrita”, “Elyse”, “Taylor”}
```

Subsets

Given a group of people, generate all possible teams, or subsets, of those people:

{}

{“Amrita”}

{“Elyse”}

{“Taylor”}

{“Amrita”, “Elyse”}

{“Amrita”, “Taylor”}

{“Elyse”, “Taylor”}

{“Amrita”, “Elyse”, “Taylor”}

Half the subsets contain “Amrita”

Subsets

Given a group of people, generate all possible teams, or subsets, of those people:

{}

{“Amrita”}

{“Elyse”}

{“Taylor”}

{“Amrita”, “Elyse”}

{“Amrita”, “Taylor”}

{“Elyse”, “Taylor”}

{“Amrita”, “Elyse”, “Taylor”}

Half the subsets contain “Elyse”

Subsets

Given a group of people, generate all possible teams, or subsets, of those people:

```
{}
{“Amrita”}
{“Elyse”}
{“Taylor”}
{“Amrita”, “Elyse”}
{“Amrita”, “Taylor”}
{“Elyse”, “Taylor”}
{“Amrita”, “Elyse”, “Taylor”}
```

Half the subsets contain “Taylor”

Subsets

Given a group of people, generate all possible teams, or subsets, of those people:

```
{}
{“Amrita”}
{“Elyse”}
{“Taylor”}
{“Amrita”, “Elyse”}
{“Amrita”, “Taylor”}
{“Elyse”, “Taylor”}
{“Amrita”, “Elyse”, “Taylor”}
```

Half the subsets that contain “Taylor” also contain “Amrita”

Subsets

Given a group of people, generate all possible teams, or subsets, of those people:

```
{}
{“Amrita”}
{“Elyse”}
{“Taylor”}
{“Amrita”, “Elyse”}
{“Amrita”, “Taylor”}
{“Elyse”, “Taylor”}
{“Amrita”, “Elyse”, “Taylor”}
```

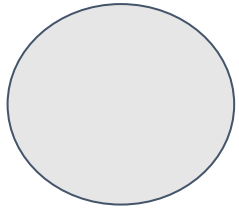
Half the subsets that contain “Taylor” and “Amrita”, also contain “Elyse”

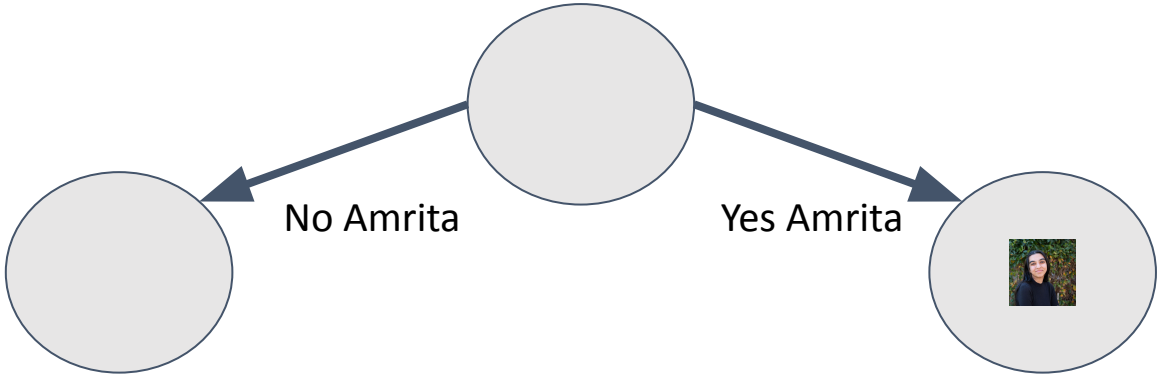
Making a Decision Tree

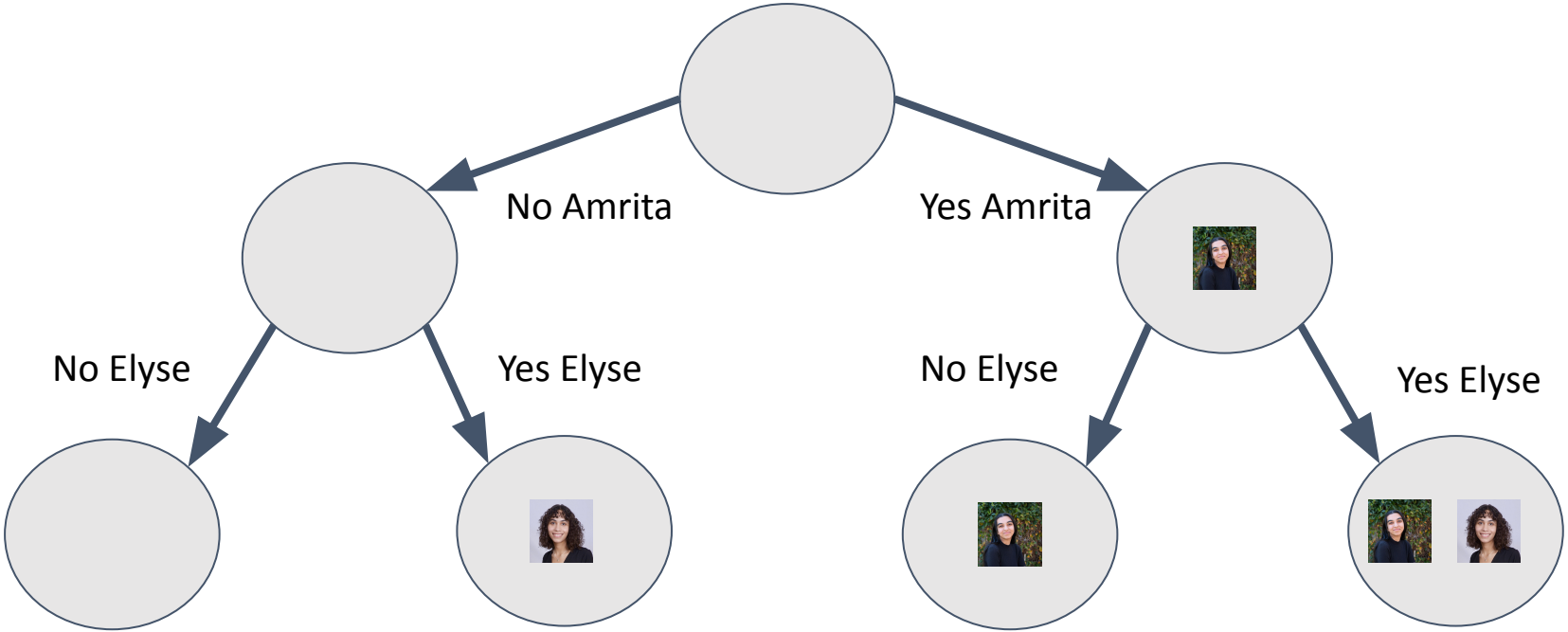
- Decision at each step (each level of the tree)
- Options at each decision (branches from each node)
- Information you need to store along the way

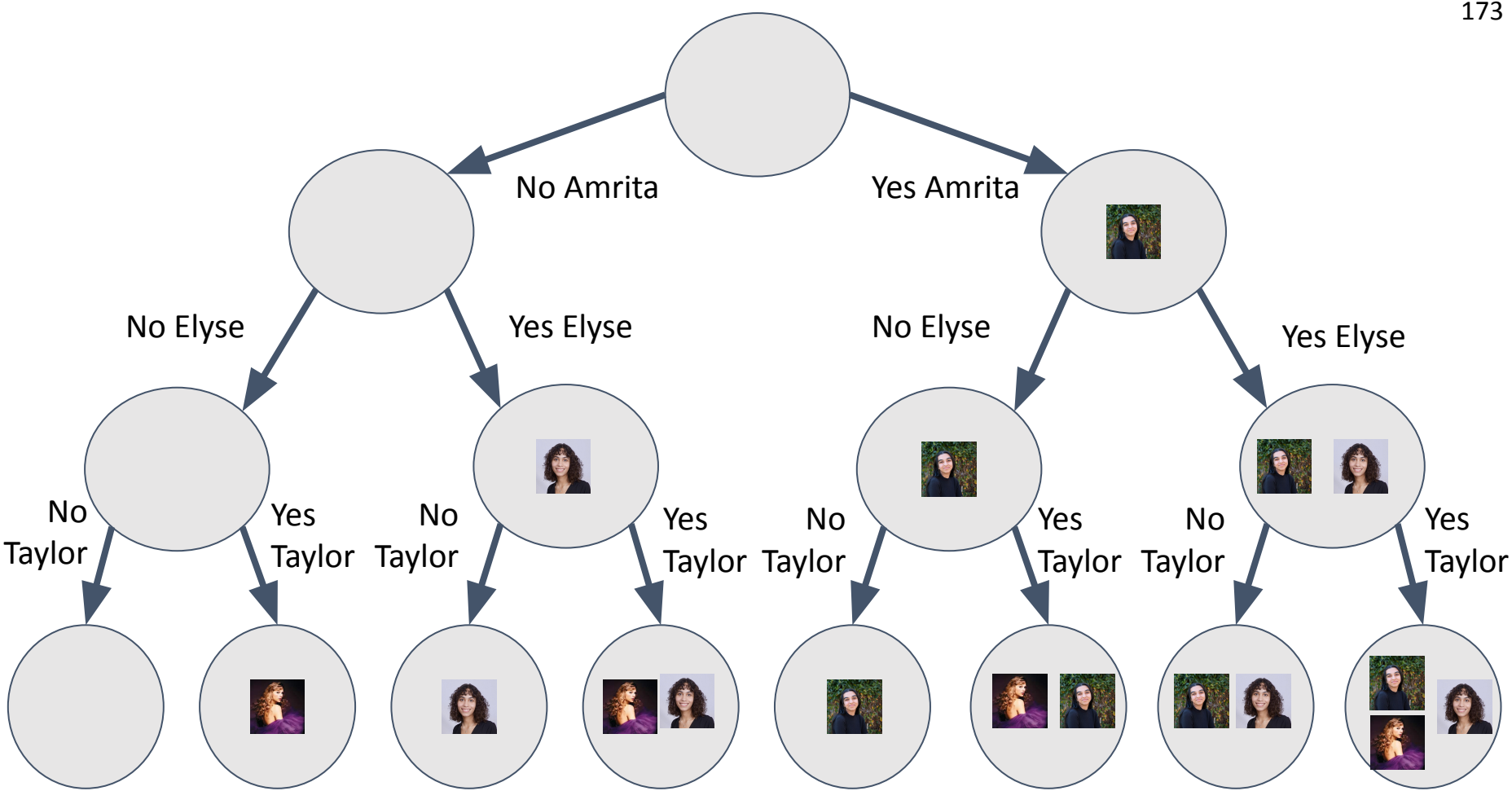
Making a Decision Tree

- Decision at each step (each level of the tree)
  - Are we going to include a given element in our subset?
- Options at each decision (branches from each node)
  - Include the element
  - Don’t Include the element
- Information you need to store along the way
  - Set you’ve built so far




No Amrita
Yes Amrita

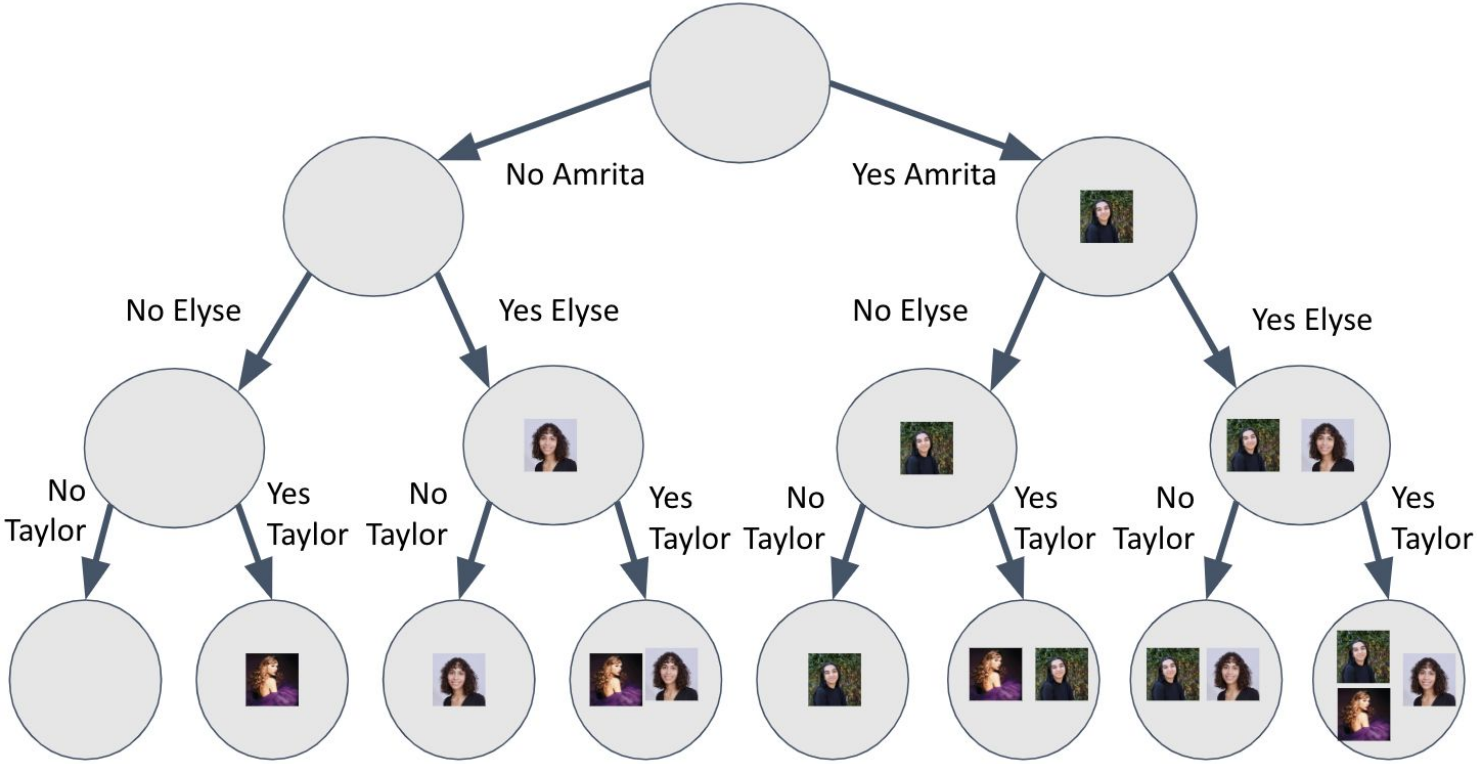

No Amrita
Yes Amrita
No Elyse
Yes Elyse
No Elyse
Yes Elyse


No Amrita
Yes Amrita
No Elyse
Yes Elyse
No Elyse
Yes Elyse
No Taylor
Yes Taylor
No Taylor
Yes Taylor
No Taylor
Yes Taylor
No Taylor
Yes Taylor

Making a Decision Tree

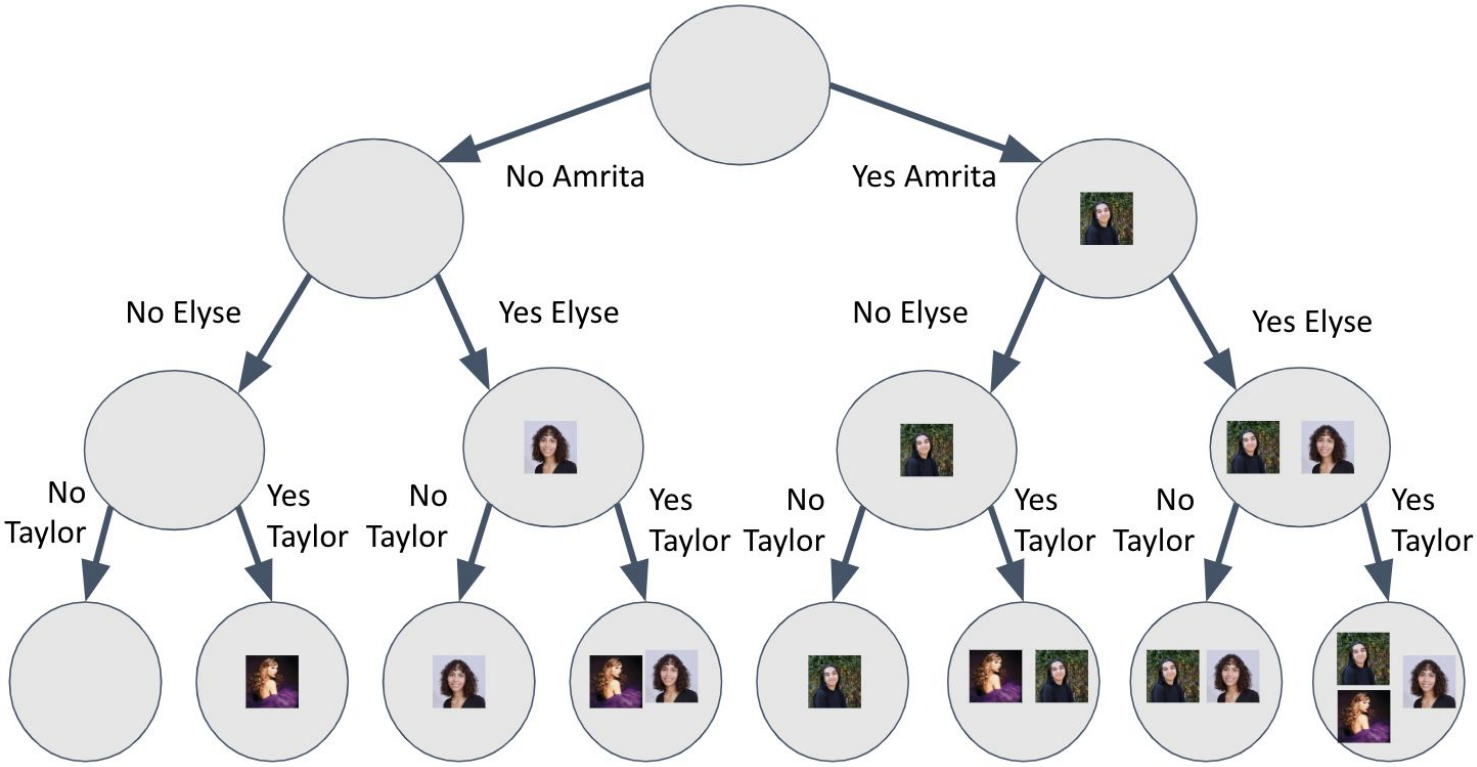
- Decision at each step (each level of the tree)
  - Are we going to include a given element in our subset?
- Options at each decision (branches from each node)
  - Include the element
  - Don’t Include the element
- Information you need to store along the way
  - Set you’ve built so far
  - **Remaining elements in original set**

Remaining Elements:



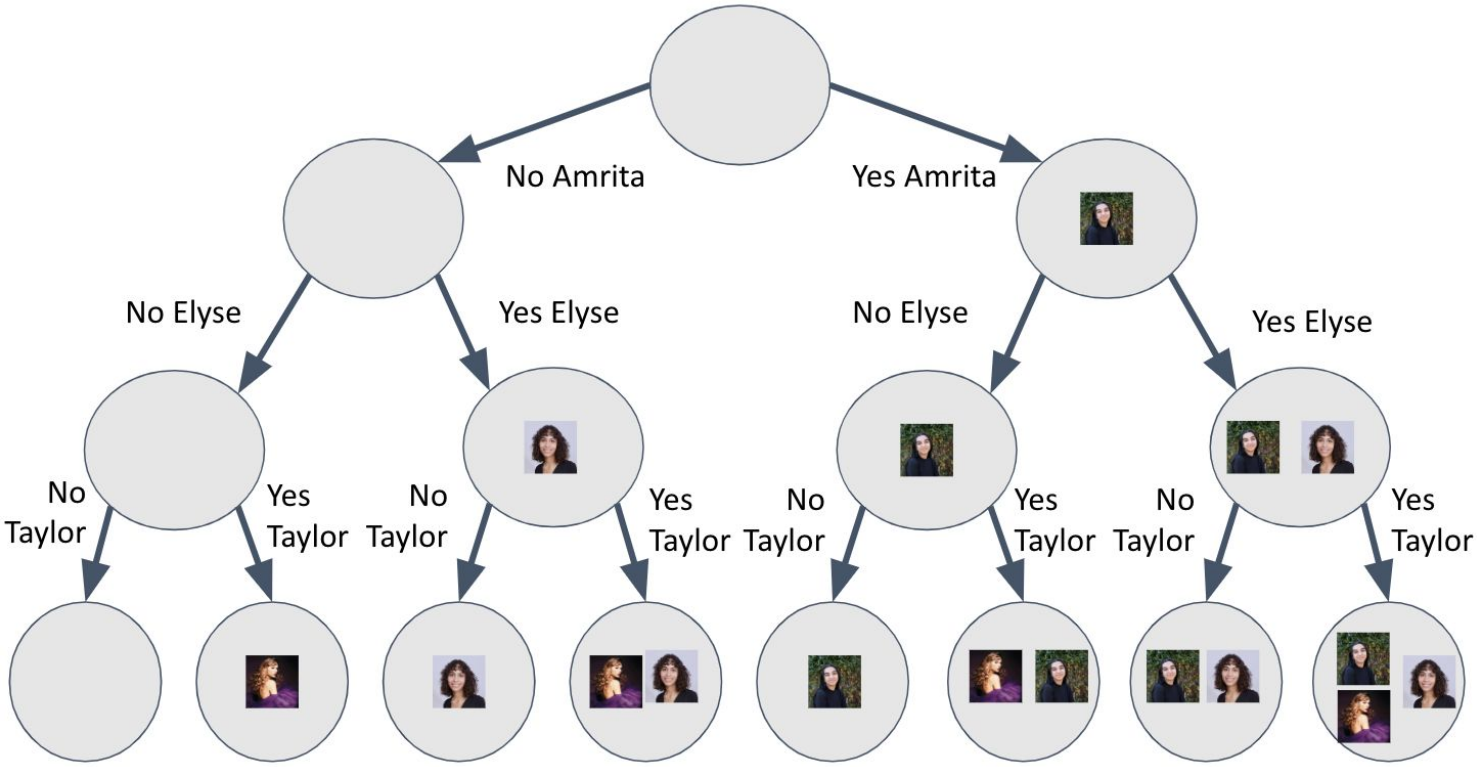
Remaining Elements:

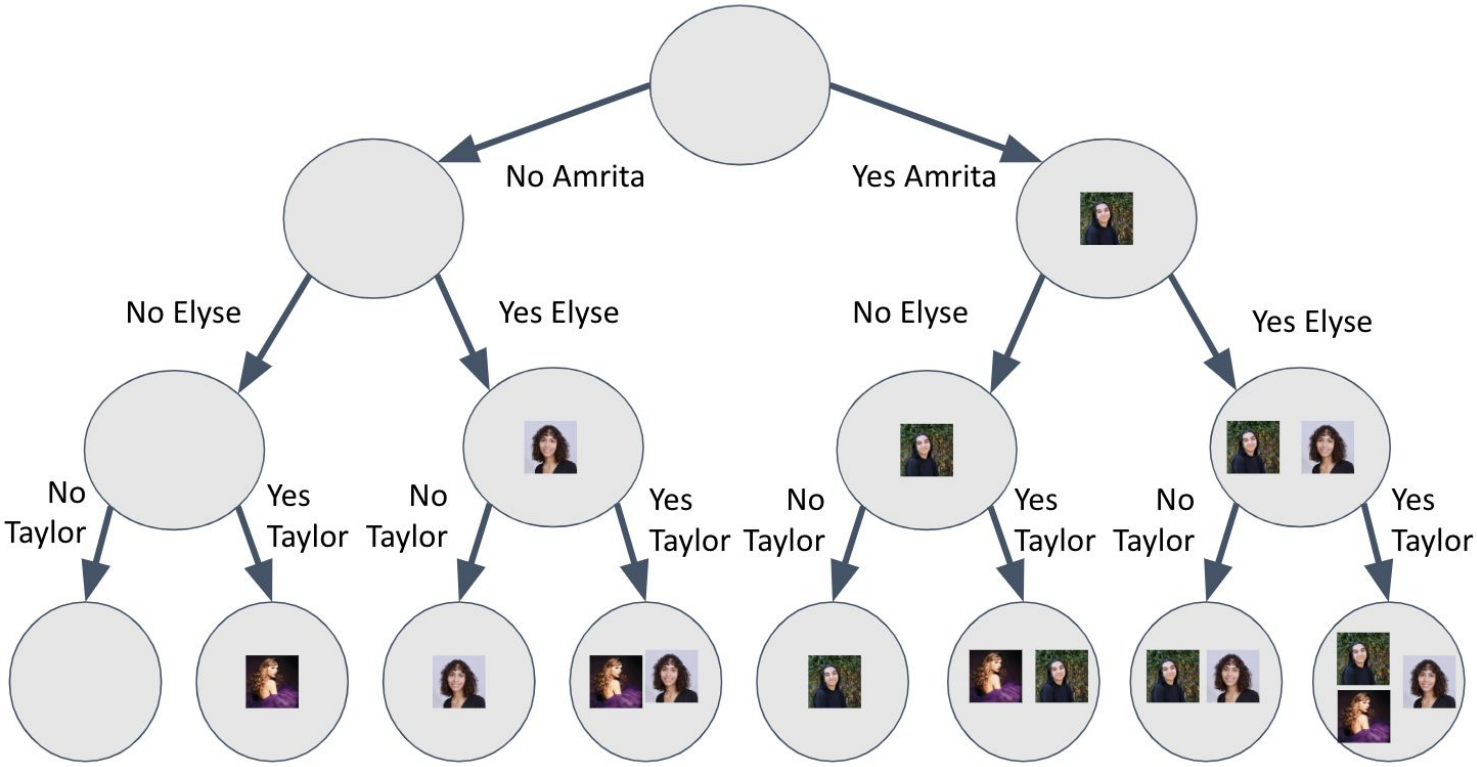
```
{“Amrita”,
 “Elyse”,
 “Taylor”}
```



Remaining Elements:

```
{“Amrita”,
 “Elyse”,
 “Taylor”}

{“Elyse”,
 “Taylor”}
```



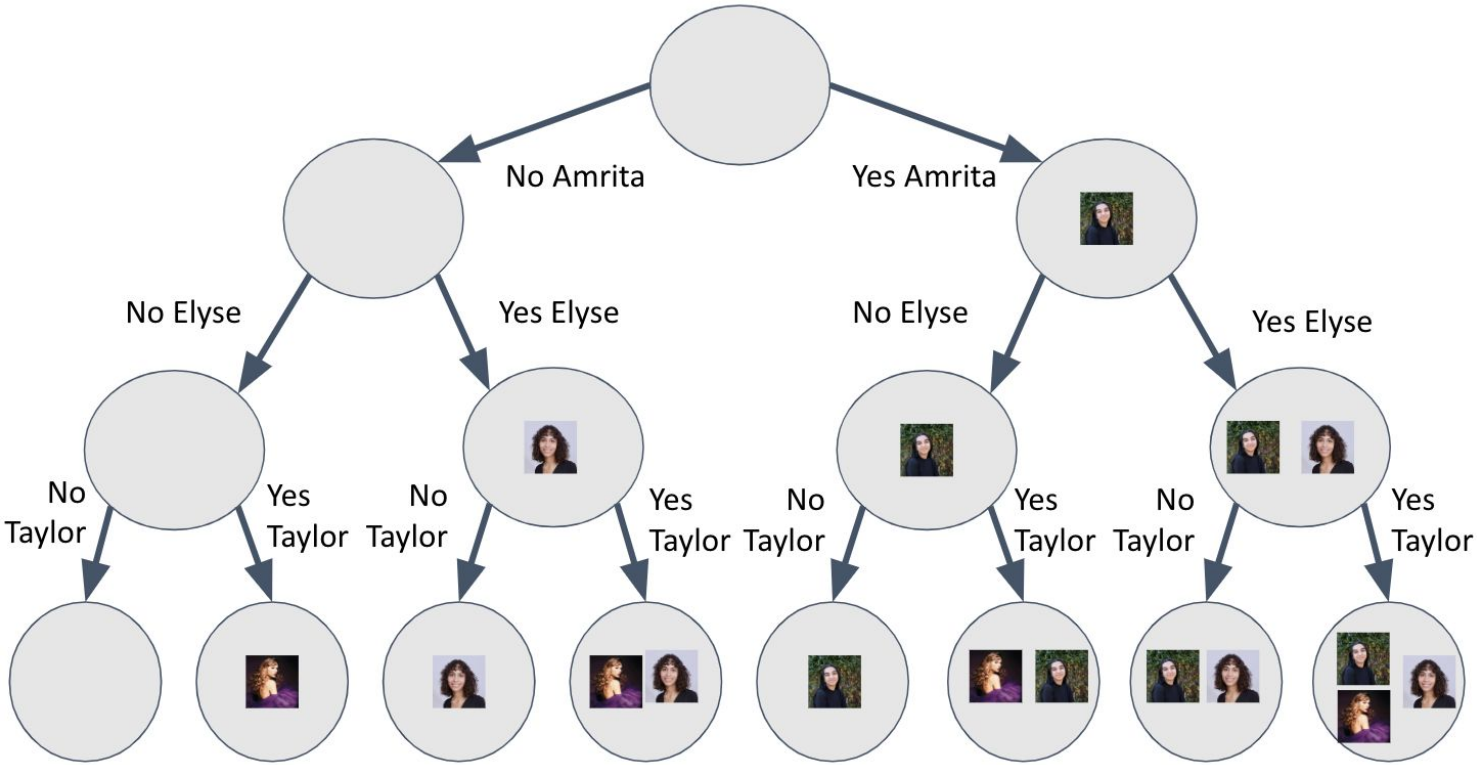


Remaining Elements:

{“Amrita”, “Elyse”, “Taylor”}

{“Elyse”, “Taylor”}

{“Taylor”}



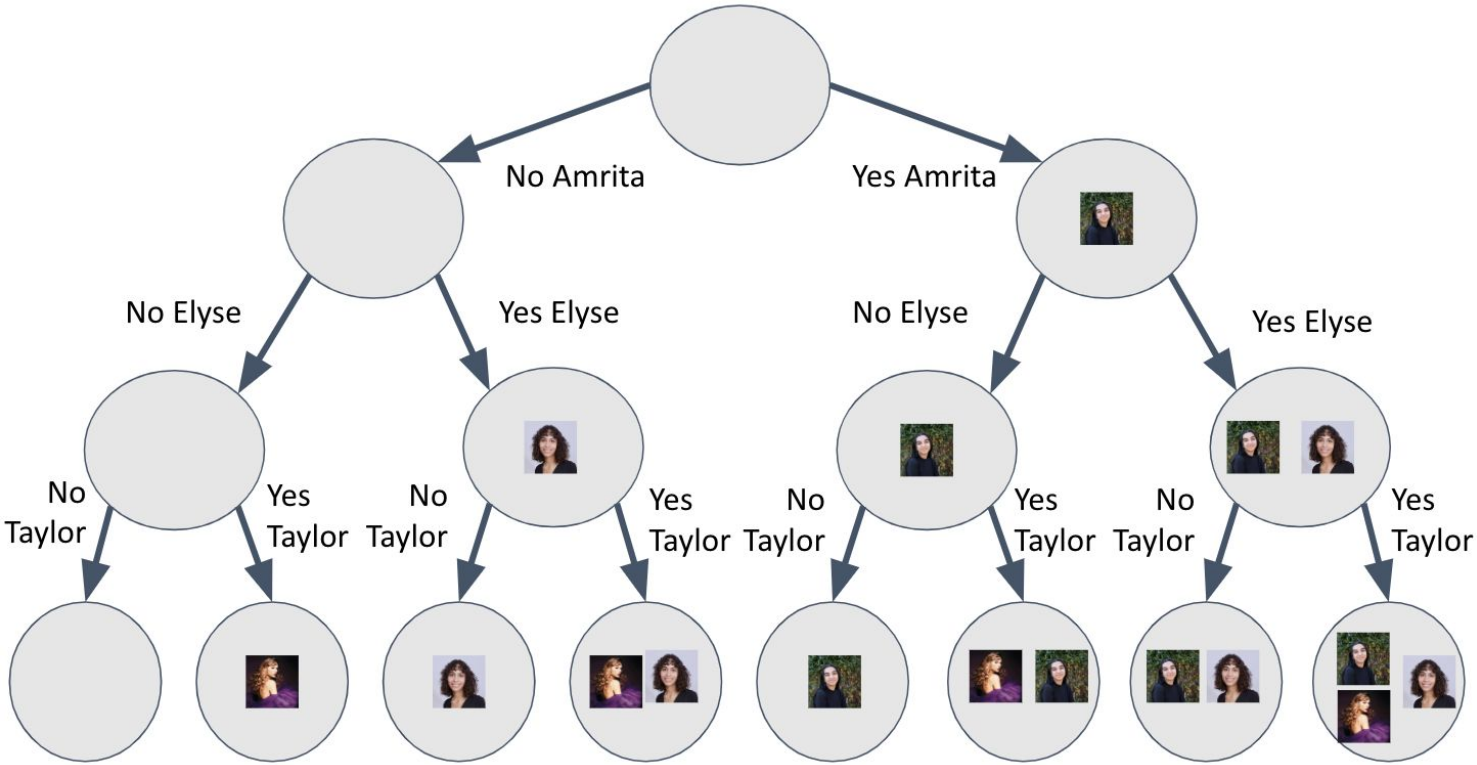
Remaining Elements:

{“Amrita”, “Elyse”, “Taylor”}

{“Elyse”, “Taylor”}

{“Taylor”}

{}



Remaining Elements:

{“Amrita”, “Elyse”, “Taylor”}

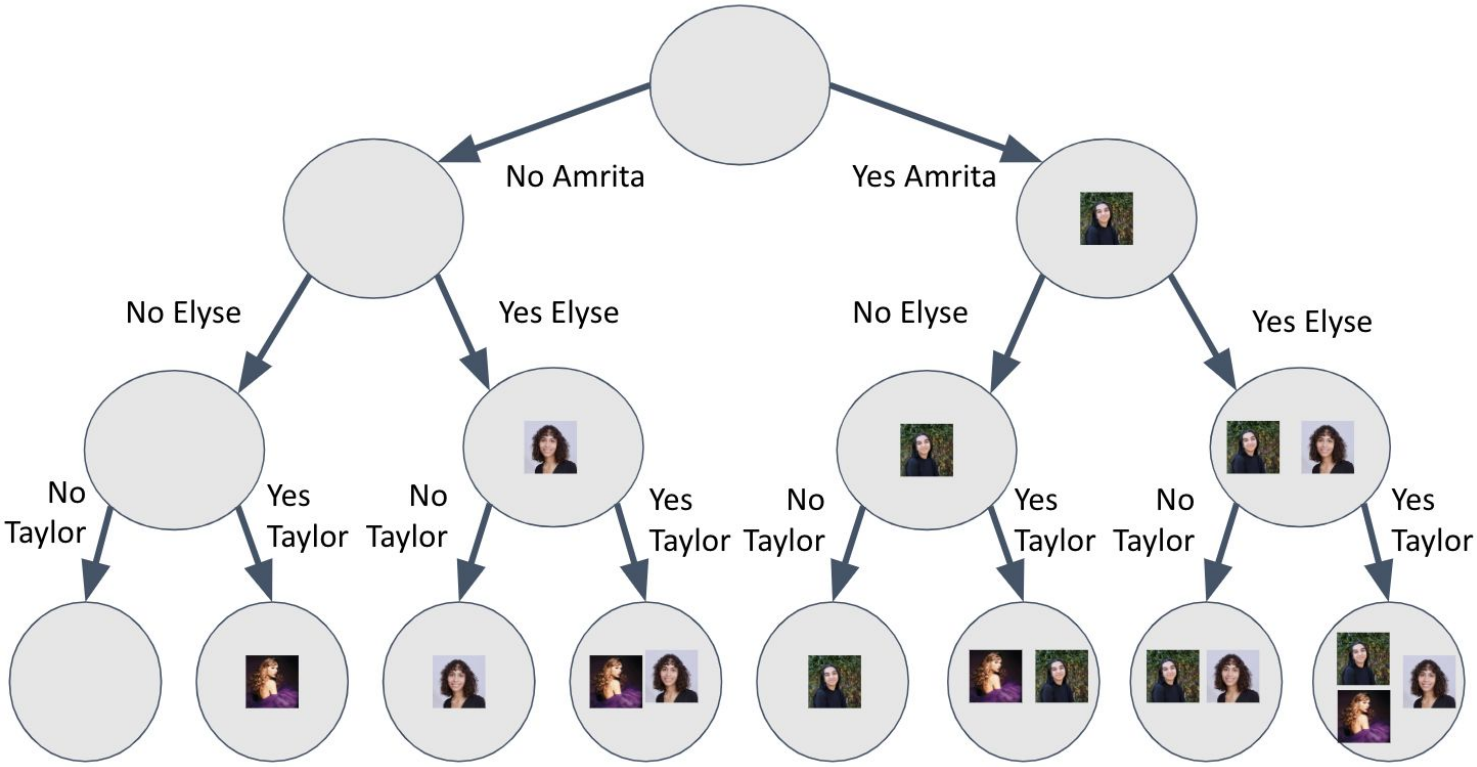
{“Elyse”, “Taylor”}

{“Taylor”}

{}

***Base Case**: No remaining people to choose from*

Remaining Elements:




```
{“Amrita”,
 “Elyse”,
 “Taylor”}

{“Elyse”,
 “Taylor”}

{“Taylor”}

{}
```

***Recursive Case**: Pick someone from set. Choose whether to include them.*

Let’s Code It Up!

Takeaways

- "Choose / explore / unchoose" pattern in backtracking
  - This is our first time seeing an explicit “unchoose” step

```
// choose
string elem = remaining.first();
remaining = remaining - elem;
// explore
listSubsetHelper(remaining, chosen);
chosen = chosen + elem
listSubsetHelper(remaining, chosen);
// unchoose this letter by adding it back to possible choices
chosen = chosen - elem;
remaining = remaining + elem;
```

Takeaways

- "Choose / explore / unchoose" pattern in backtracking
  - This is our first time seeing an explicit “unchoose” step

```
// choose
string elem = remaining.first();
remaining = remaining - elem;
// explore
listSubsetHelper(remaining, chosen); // do not add elem to chosen
chosen = chosen + elem
listSubsetHelper(remaining, chosen);
// unchoose this letter by adding it back to possible choices
chosen = chosen - elem;
remaining = remaining + elem;
```

Takeaways

- "Choose / explore / unchoose" pattern in backtracking
  - This is our first time seeing an explicit “unchoose” step

```
// choose
string elem = remaining.first();
remaining = remaining - elem;
// explore
listSubsetHelper(remaining, chosen);
chosen = chosen + elem
listSubsetHelper(remaining, chosen); // add elem to chosen
// unchoose by adding it back to possible choices
chosen = chosen - elem;
remaining = remaining + elem;
```

Takeaways

- "Choose / explore / unchoose" pattern in backtracking
  - This is our first time seeing an explicit “unchoose” step
  - Necessary because we’re passing sets by reference and editing them

```
// choose
string elem = remaining.first();
remaining = remaining - elem;
// explore
listSubsetHelper(remaining, chosen);
chosen = chosen + elem
listSubsetHelper(remaining, chosen);
// unchoose by adding it back to possible choices
chosen = chosen - elem;
remaining = remaining + elem;
```

Takeaways

- "Choose / explore / unchoose" pattern in backtracking
  - This is our first time seeing an explicit “unchoose” step
  - Necessary because we’re passing sets by reference and editing them
- Our first example of using ADTs with recursion, and we’ll see more today

Choose / explore / unchoose

- Implicit “unchoose” step
  - Pass by value; usually when memory constraints aren’t an issue
  - Works because you’re making edits to a copy
  - E.g. Building up a string over time

- Explicit “unchoose” step
  - Uses pass by reference; usually with large data structures
  - “Undoing” prior modifications to structure
  - E.g. Generating subsets (one set passed around by reference to track
  - subsets)

Choose / explore / unchoose

- Implicit “unchoose” step
  - Pass by value; usually when memory constraints aren’t an issue
  - Works because you’re making edits to a copy
  - E.g. Building up a string over time

- Explicit “unchoose” step
  - Uses pass by reference; usually with large data structures
  - “Undoing” prior modifications to structure
  - E.g. Generating subsets (one set passed around by reference to track
  - subsets)

Solving Recursive Backtracking

- Which of our three use cases does our problem fall into? (generate/count all solutions, find one solution/prove its existence, or pick one best solution)

Solving Recursive Backtracking

- Which of our three use cases does our problem fall into? (generate/count all solutions, find one solution/prove its existence, or pick one best solution)
- What's the provided function prototype and requirements? Do we need a helper function?

Solving Recursive Backtracking

- Which of our three use cases does our problem fall into? (generate/count all solutions, find one solution/prove its existence, or pick one best solution)
- What's the provided function prototype and requirements? Do we need a helper function?
  - What are we returning as our solution?
  - Do we care about returning or keeping track of the path we took to get to our solution? If yes, what parameters are we already given and what others might be useful?

Solving Recursive Backtracking

- Which of our three use cases does our problem fall into? (generate/count all solutions, find one solution/prove its existence, or pick one best solution)
- What's the provided function prototype and requirements? Do we need a helper function?
  - What are we returning as our solution?
  - Do we care about returning or keeping track of the path we took to get to our solution? If yes, what parameters are we already given and what others might be useful?
- What are our base and recursive cases?

Solving Recursive Backtracking

- Which of our three use cases does our problem fall into? (generate/count all solutions, find one solution/prove its existence, or pick one best solution)
- What's the provided function prototype and requirements? Do we need a helper function?
  - What are we returning as our solution?
  - Do we care about returning or keeping track of the path we took to get to our solution? If yes, what parameters are we already given and what others might be useful?
- What are our base and recursive cases?
  - What does the decision tree look like? (decisions, options, what to keep track of)
  - In addition to what we're building up, are there any additional constraints on our solutions?
  - Does it make sense to use an implicit or explicit unchoose step for the recursion?

Fixed-size Teams

Subsets vs Combinations

- Our goal: Pick a combination of 3 graders out of a group of 5.
  - More useful than our generating subsets solution!

Subsets vs Combinations

- Our goal: Pick a combination of 3 graders out of a group of 5.
  - More useful than our generating subsets solution!
- This sounds very similar to the problem we solved when we generated subsets
  - These 3 graders would be a subset of the overall group of 5.

Subsets vs Combinations

- Our goal: Pick a combination of 3 graders out of a group of 5.
  - More useful than our generating subsets solution!
- This sounds very similar to the problem we solved when we generated subsets
  - These 3 graders would be a subset of the overall group of 5.
- What distinguishes a combination from a subset?
  - Combinations always have a specified size, unlike subsets (which can be any size)
  - We can think of combinations as "subsets with constraints"

Subsets vs Combinations

- Our goal: Pick a combination of 3 graders out of a group of 5.
  - More useful than our generating subsets solution!
- This sounds very similar to the problem we solved when we generated subsets
  - These 3 graders would be a subset of the overall group of 5.
- What distinguishes a combination from a subset?
  - Combinations always have a specified size, unlike subsets (which can be any size)
  - We can think of combinations as "subsets with constraints"
- Could we use the code from before, generate all subsets, and then filter out all those of size 3?
  - We could, but that would be inefficient. Let's develop a better approach for combinations!

Solving Recursive Backtracking

- **Which of our three use cases does our problem fall into? (generate/count all solutions, find one solution/prove its existence, or pick one best solution)**
- What's the provided function prototype and requirements? Do we need a helper function?
  - What are we returning as our solution?
  - Do we care about returning or keeping track of the path we took to get to our solution? If yes, what parameters are we already given and what others might be useful?
- What are our base and recursive cases?
  - What does the decision tree look like? (decisions, options, what to keep track of)
  - In addition to what we're building up, are there any additional constraints on our solutions?
  - Does it make sense to use an implicit or explicit unchoose step for the recursion?

Solving Recursive Backtracking

- Which of our three use cases does our problem fall into? (generate/count all solutions, find one solution/prove its existence, or pick one best solution)
- **What's the provided function prototype and requirements? Do we need a helper function?**
  - **What are we returning as our solution?**
  - **Do we care about returning or keeping track of the path we took to get to our solution? If yes, what parameters are we already given and what others might be useful?**
- What are our base and recursive cases?
  - What does the decision tree look like? (decisions, options, what to keep track of)
  - In addition to what we're building up, are there any additional constraints on our solutions?
  - Does it make sense to use an implicit or explicit unchoose step for the recursion?

What are we returning as our solution?

- Each combination of k graders can be represented as a `Set<string>`.
- In previous string examples, we were just printing out all solutions. But what if we wanted to store all of them to be able to do something with them later?
- We want to return a container holding all possible combinations:

`Set<Set<string>>`

What are we returning as our solution?

- Each combination of k graders can be represented as a `Set<string>`.
- In previous string examples, we were just printing out all solutions. But what if we wanted to store all of them to be able to do something with them later?
- We want to return a container holding all possible combinations

```
Set<Set<string>> combinationsOf(Set<string>& graders, int k)
```

Do we need a helper function?

```
Set<Set<string>> combinationsOf(Set<string>& graders, int k)
```

Do we need a helper function?

```
Set<Set<string>> combinationsOf(Set<string>& graders, int k)
```

```
Set<Set<string>> combinationsHelper(Set<string>& remaining,
                                    int k, Set<string>& chosen)
```

Solving Recursive Backtracking

- Which of our three use cases does our problem fall into? (generate/count all solutions, find one solution/prove its existence, or pick one best solution)
- What's the provided function prototype and requirements? Do we need a helper function?
  - What are we returning as our solution?
  - Do we care about returning or keeping track of the path we took to get to our solution? If yes, what parameters are we already given and what others might be useful?
- **What are our base and recursive cases?**
  - **What does the decision tree look like? (decisions, options, what to keep track of)**
  - **In addition to what we're building up, are there any additional constraints on our solutions?**
  - **Does it make sense to use an implicit or explicit unchoose step for the recursion?**

What are the base and recursive cases?

- Base cases

What are the base and recursive cases?

- Base cases
  - Found k graders
  - There aren’t enough graders remaining

What are the base and recursive cases?

- Base cases
  - Found k graders
  - There aren’t enough graders remaining
- Recursive cases

What are the base and recursive cases?

- Base cases
  - Found k graders
  - There aren’t enough graders remaining
- Recursive cases
  - Pick someone from set and choose whether or not to include them

Implicit or explicit unchoose step?

Implicit or explicit unchoose step?

- Explicit!
  - We're passing in our set by reference, so we need to undo any choices we make

Let’s Code It Up!

Next Class

- Recursive backtracking to solve mazes!
- Misc.
  - Const reference
  - Structs
- Classes and Object-Oriented Programming