Thinking Recursively

Part II

Outline for Today

- ***The Recursive Leap of Faith***
  - On trusting the contract.
- ***Enumerating Subsets***
  - A classic combinatorial problem.
- ***Decision Trees***
  - Generating all solutions to a problem.

Some Quick Refreshers

Set Refresher

- What's printed at Line *A* and Line *B*?

```
Set<int> mySet = {1, 2, 3};
cout << (mySet + 4) << endl; // Line A
cout << (mySet - 3) << endl; // Line B
```

Answer at
https://cs106b.stanford.edu/pollev

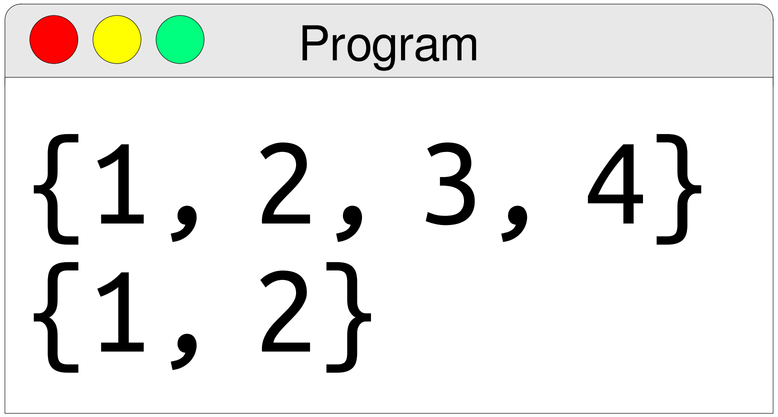
Set Refresher

- What's printed at Line *A* and Line *B*?

```
Set<int> mySet = {1, 2, 3};
cout << (mySet + 4) << endl; // Line A
cout << (mySet - 3) << endl; // Line B
```

```
{1, 2, 3}
```

```
Set<int> mySet
```



```
{1, 2, 3, 4}
{1, 2}
```

Recursion Refresher

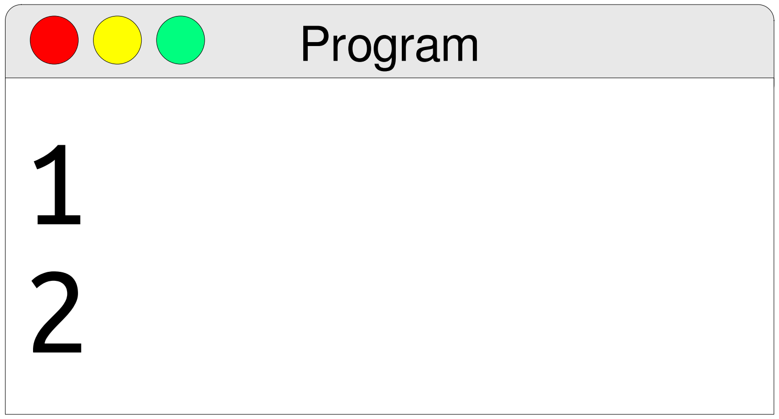
- What does this code print?

```
void squigglebah(int n) {
   if (n != 0) {
      squigglebah(n - 1);
      cout << n << endl;
   }
}

squigglebah(2);
```

Answer at
https://cs106b.stanford.edu/pollev

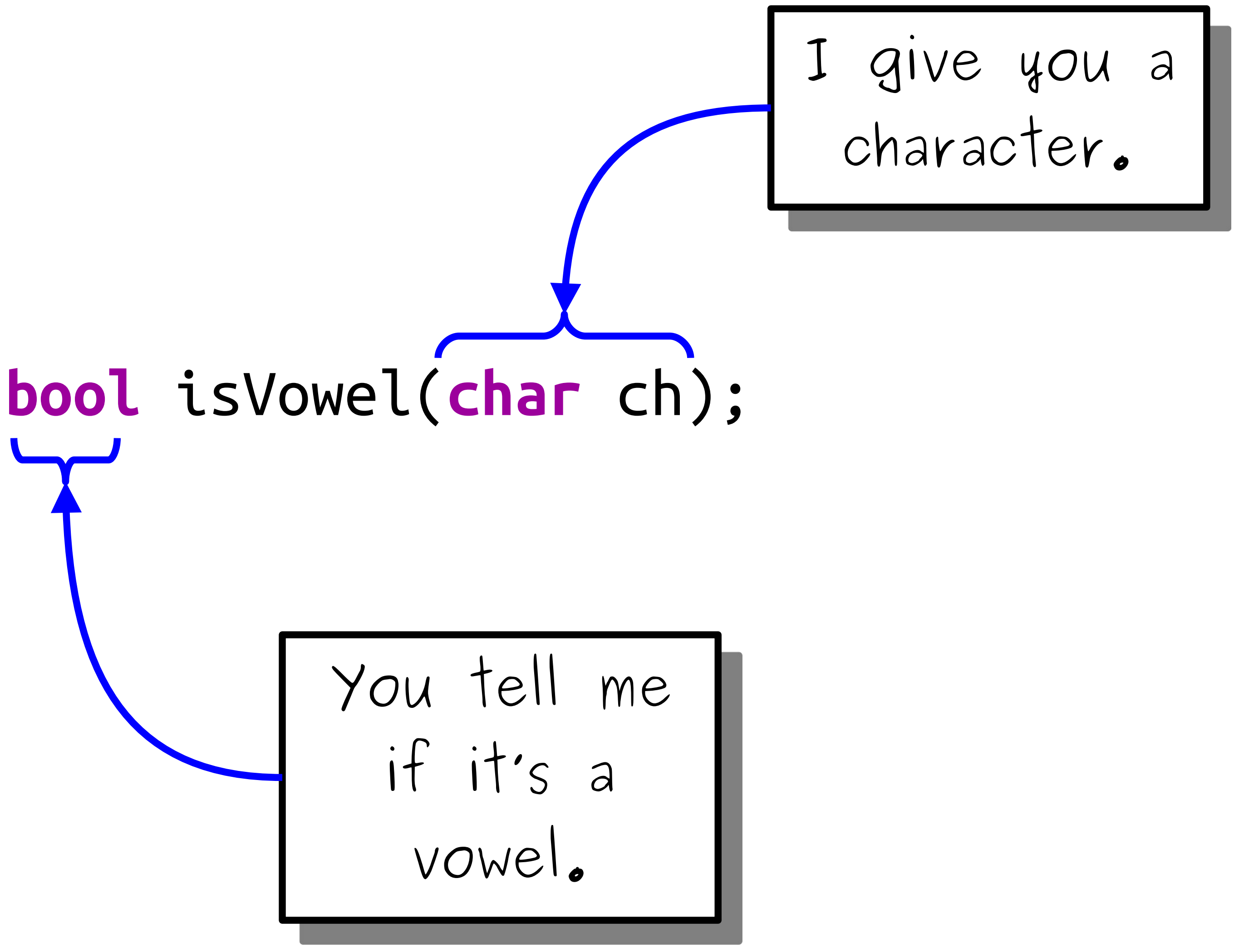
```
squigglebah(2);
```



```
1
2
```

The Recursive Leap of Faith

The Contract



```
bool isVowel(char ch);
```

The Contract

```
bool isVowel(char ch) {
    ch = toLowerCase(ch);
    return ch == 'a' ||
           ch == 'e' ||
           ch == 'i' ||
           ch == 'o' ||
           ch == 'u';
}
```

The Contract

```
bool isVowel(char ch) {
    switch(ch) {
        case 'A': case 'a':
        case 'E': case 'e':
        case 'I': case 'i':
        case 'O': case 'o':
        case 'U': case 'u':
            return true;
        default:
            return false;
    }
}
```

The Contract

```
bool isVowel(char ch) {
    ch = tolower(ch);
    return string("aeiou").find(ch) != string::npos;
}
```

The Contract

I give you a string.

```
bool hasConsecutiveVowels(const string& str);
```

You tell me if it has two or more consecutive letters that are vowels.

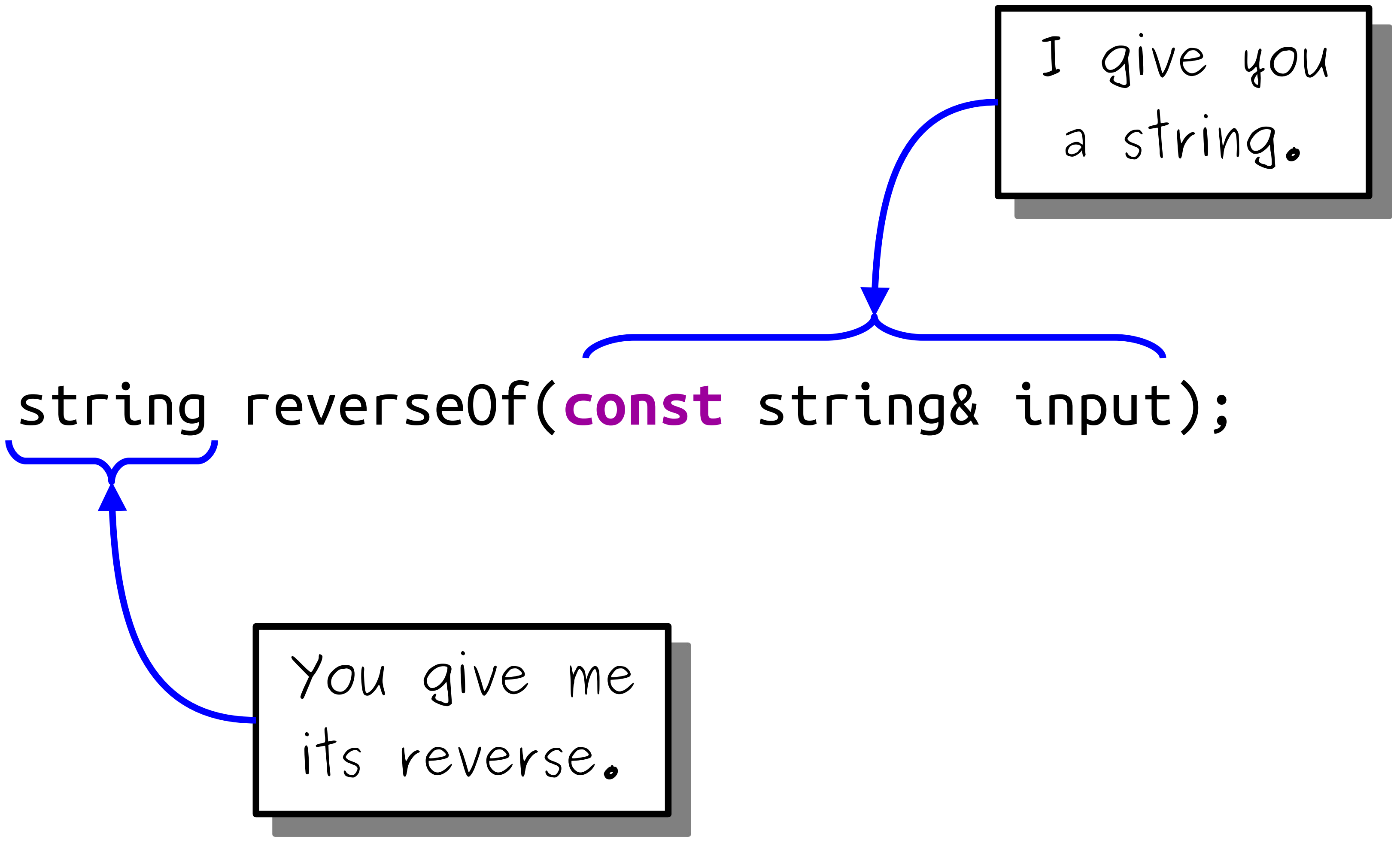
Trusting the Contract

```
bool isVowel(char ch);

bool hasConsecutiveVowels(const string& str) {
  for (int i = 1; i < str.length(); i++) {
    if (isVowel(str[i - 1]) && isVowel(str[i])) {
      return true;
    }
  }
  retur
}
```

It doesn't matter how isVowel is implemented. We just trust that it works.

The Contract



```
string reverseOf(const string& input);
```

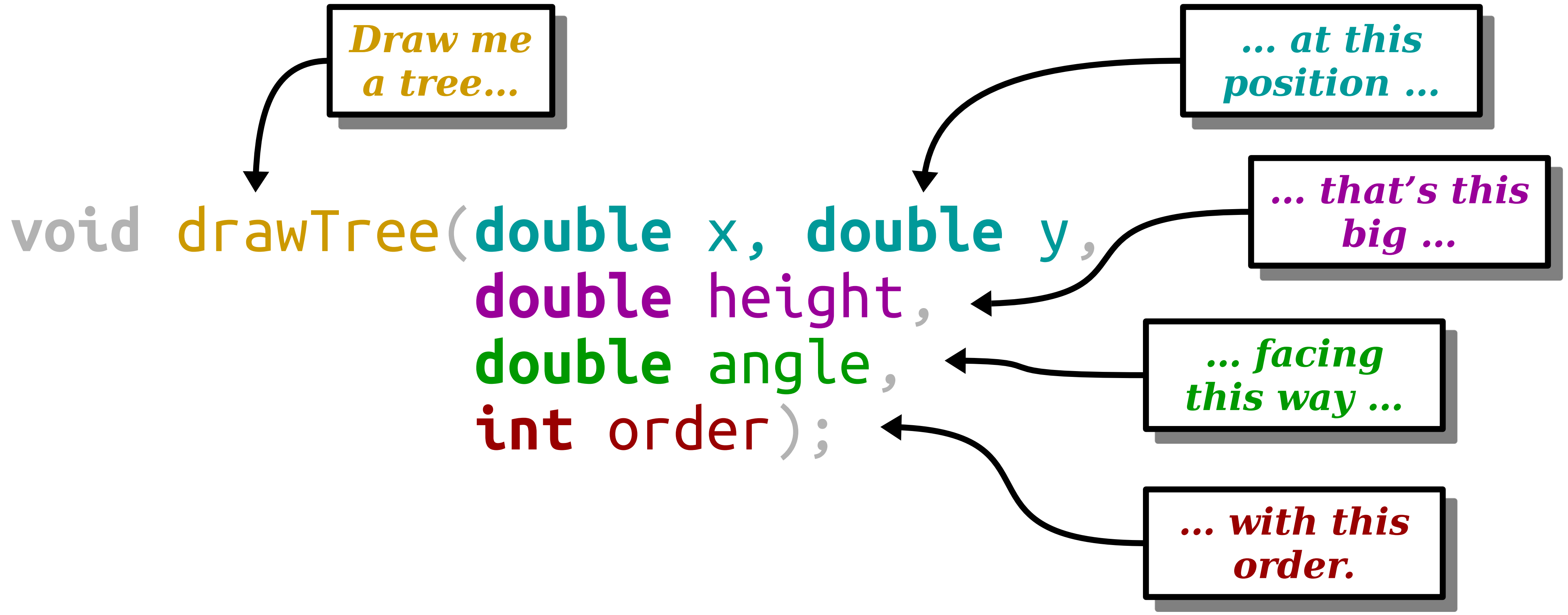
Trusting the Contract

```
string reverseOf(const string& input);

string reverseOf(const string& input) {
    if (input == "") {
        return "";
    } else {
        return reverseOf(input.substr(1)) + input[0];
    }
}
```

It doesn't matter how **reverseOf** reverses the string. It just matters that it does.

The Contract



```
void drawTree(double x, double y,
              double height,
              double angle,
              int order);
```

Draw me a tree…

… at this position …

… that's this big …

… facing this way …

… with this order.

Trusting the Contract

```
void drawTree(double x, double y,
              double height, double angle,
              int order);

void drawTree(double x, double y,
              double height, doubl
              int order) {
    if (order == 0) return;

    GPoint endpoint = drawPolarLine(/* … */);

    drawTree(/* … */);
    drawTree(/* … */);
}
```

It doesn't matter how **drawTree** draws a tree. It just matters that it does.

The Recursive Leap of Faith

- When writing a recursive function, it helps to take a ***recursive leap of faith***.
- Before writing the function, answer these questions:
  - What does the function take in?
  - What does it return?
- Then, as you’re writing the function, trust that your recursive calls to the function just “work” without asking how.
- This can take some adjustment to get used to, but is a necessary skill for writing more complex recursive functions.

Time-Out for Announcements!

Recursive Drawing Contest

- We are holding a (purely optional, just for fun) Recursive Drawing contest!
- Visit ***http://recursivedrawing.com/***, draw whatever you'd like, and post it to the EdStem thread for the contest.
- We'll award recursion-themed prizes to a small number of entries.
- Deadline to submit is Monday at 1:00PM Pacific.

Assignment 2

- Assignment 2 is due this Friday at 1:00PM.
  - If you’re following our timetable, you’ll have finished Rosetta Stone at this point and be midway through Rising Tides.
- Have questions?
  - Stop by the LaIR!
  - Ask on EdStem!
  - Visit our office hours!

Back to CS106B!

Recursive $\underbrace{\text{Enumeration}}$

e·nu·mer·a·tion
noun

The act of mentioning a number of things one by one.

(Source: Oxford Languages, via Google)

Listing Subsets

- A set S is a ***subset*** of a set T when every element of S is an element of T.
- There are two subsets of {2}:

  { } {2}

- There are four subsets of {2, 3}:

  { } {2} {3} {2, 3}

- How many subsets are there of {2, 3, 4}?

Answer at
https://cs106b.stanford.edu/pollev

Listing Subsets

- A set *S* is a ***subset*** of a set *T* when every element of *S* is an element of *T*.
- There are two subsets of {2}:

  { }   {2}

- There are four subsets of {2, 3}:

  { }   {2}   {3}   {2, 3}

- How many subsets are there of {2, 3, 4}?

  { }
  {2} {3} {4}
  {2, 3} {2, 4} {3, 4}
  {2, 3, 4}

- The only subset of { } is { }.

You need to send an emergency team of doctors to an area.

You know which doctors you have available to send.

List all the possible teams you can make from your list of all the doctors.

$$\{ 1, 2, 3, 4 \}$$

```
{                 }          {  1                }
{              4  }          {  1 ,           4  }
{          3      }          {  1 ,       3      }
{          3 , 4  }          {  1 ,       3 , 4  }
{      2          }          {  1 , 2            }
{      2 ,     4  }          {  1 , 2 ,       4  }
{      2 , 3      }          {  1 , 2 , 3        }
{      2 , 3 , 4  }          {  1 , 2 , 3 , 4    }
```

{ 1, 2, 3, 4 }

{ }	{ 1 }
{ 4 }	{ 1 , 4 }
{ 3 }	{ 1 , 3 }
{ 3 , 4 }	3 , 4 }
{ 2 }	}
{ 2 , 4 }	{ 1 , 2 , 4 }
{ 2 , 3 }	{ 1 , 2 , 3 }
{ 2 , 3 , 4 }	{ 1 , 2 , 3 , 4 }

These are all the subsets of { 2, 3, 4 }.

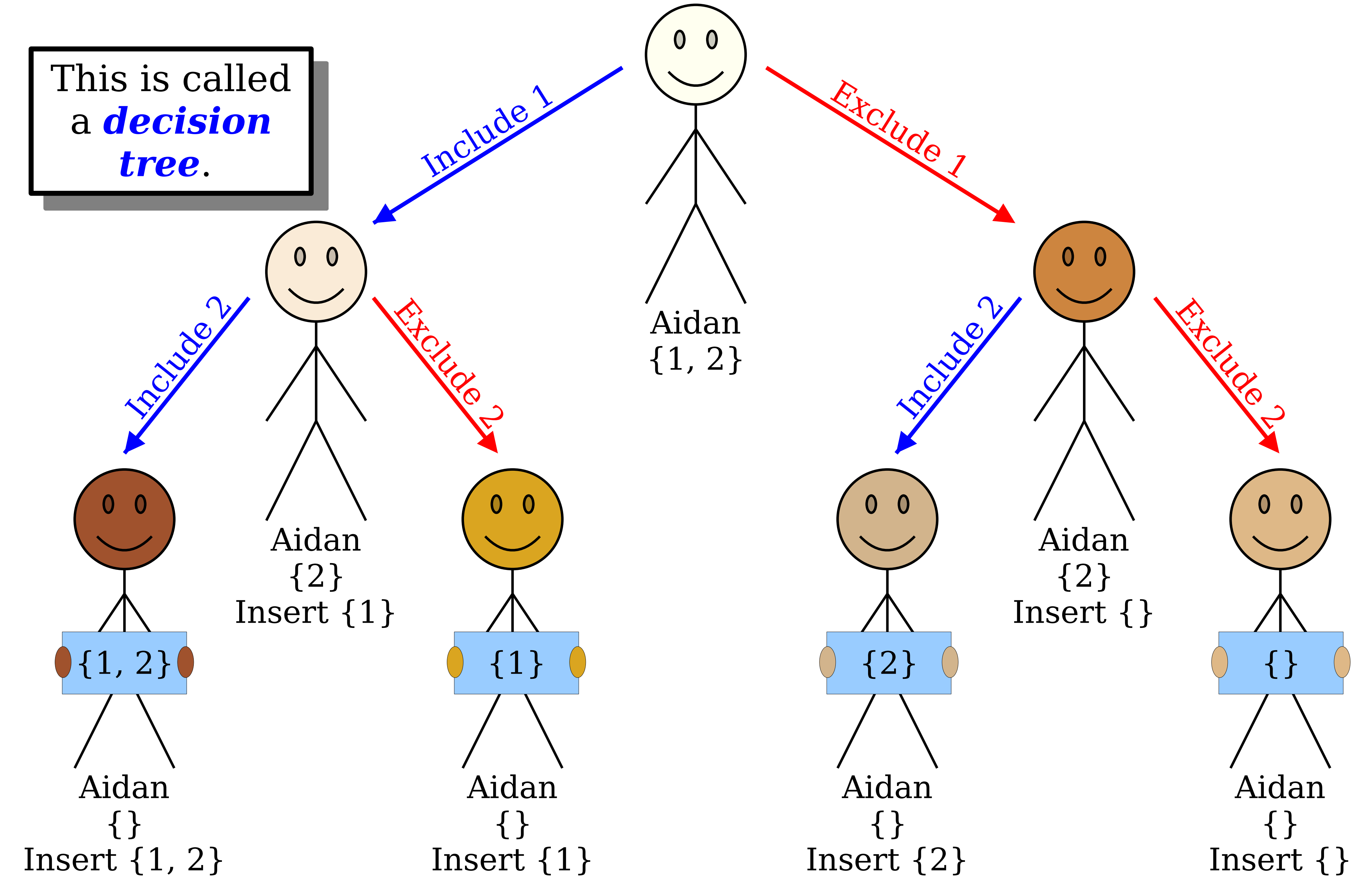
{ 1, 2, 3, 4 }

```
{           }      { 1           }
{         4 }      { 1 ,       4 }
{                  { 1 ,   3     }
{                  { 1 ,   3 , 4 }
{ 2                { 1 , 2       }
{ 2 ,     4 }      { 1 , 2 ,   4 }
{ 2 , 3     }      { 1 , 2 , 3   }
{ 2 , 3 , 4 }      { 1 , 2 , 3 , 4 }
```

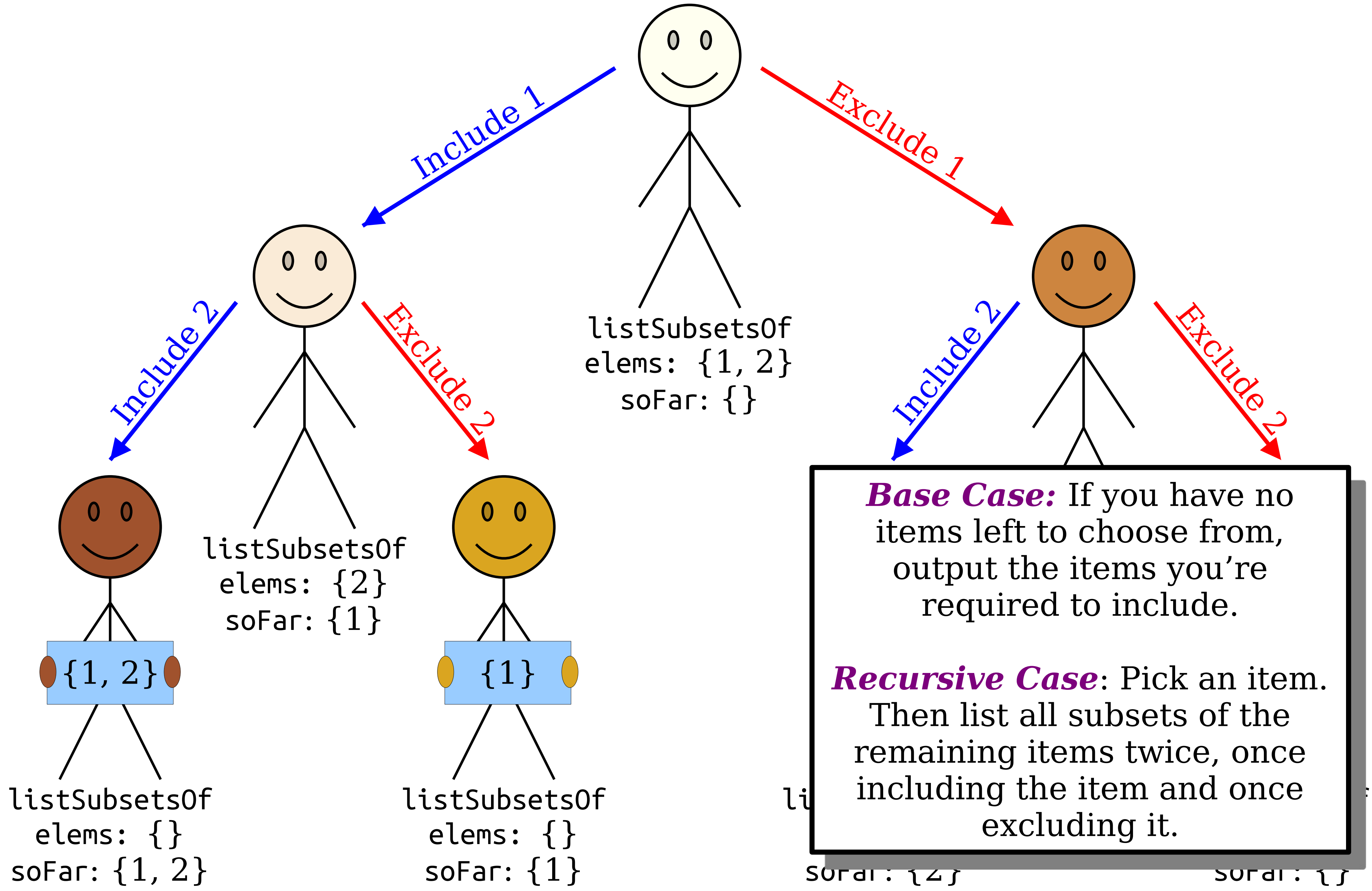
These are all the subsets of { 2, 3, 4 } with 1 inserted into them.

Aidans List Subsets

This is called a ***decision tree***.



Aidans List Subsets



Base Case: If you have no items left to choose from, output the items you're required to include.

Recursive Case: Pick an item. Then list all subsets of the remaining items twice, once including the item and once excluding it.

Summary For Today

- Making the ***recursive leap of faith*** and trusting that your recursive calls will perform as expected helps simplify writing recursive code.
- A ***decision tree*** models all the ways you can make choices to arrive at a set of results.

Your Action Items

- ***Read Chapter 8.***
  - There’s a lot of great information there about recursive problem-solving, and it’s a great resource.
- ***Read the Slide Appendix***
  - There’s a trace through how this function works; review this before next lecture.
- ***Finish Assignment 2***
  - If you’re following our suggested timetable, at this point you’ll have finished Rosetta Stone and will have started working on Rising Tides.
  - Come to LaIR or ask on EdStem if you have any questions!

Next Time

- ***Iteration + Recursion***
  - Combining two techniques together.
- ***Enumerating Permutations***
  - What order should we perform tasks in?

Appendix: Tracing the Recursion