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.
- ***Wrapper Functions***
  - Hiding parameters and keeping things clean.

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
```

Formulate a hypothesis, but ***don’t post anything in chat just yet***.

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
```

Now, ***private chat me your best guess***. Not sure? Just answer "??"

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
```

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
```

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
```

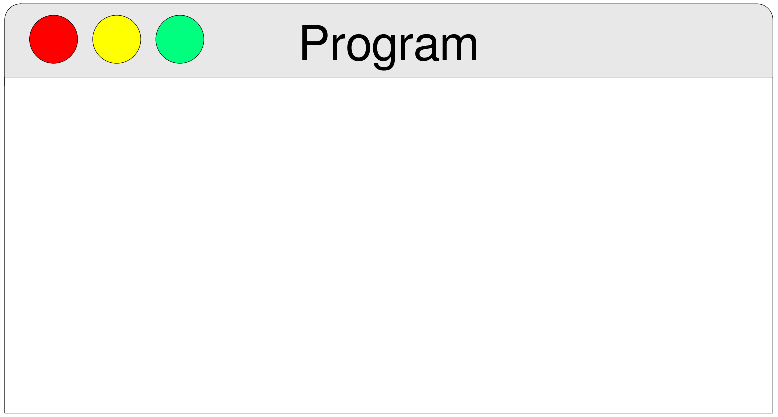
Set Refresher

- 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
```



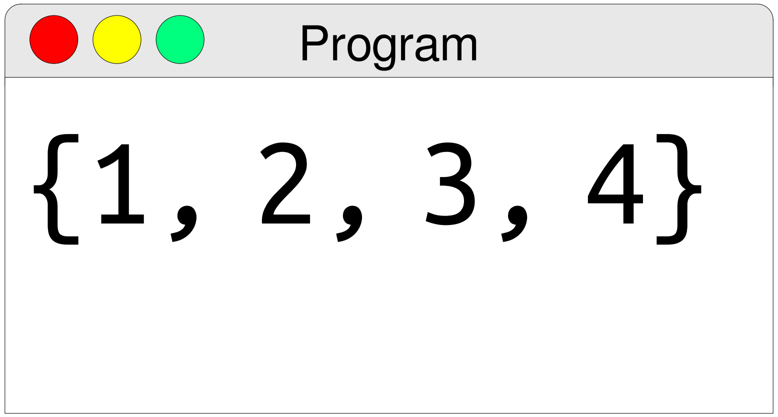
Set Refresher

- 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}
```

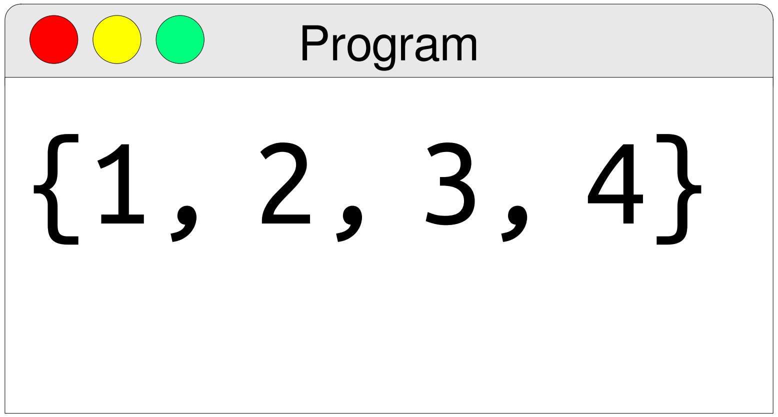
Set Refresher

- 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}
```

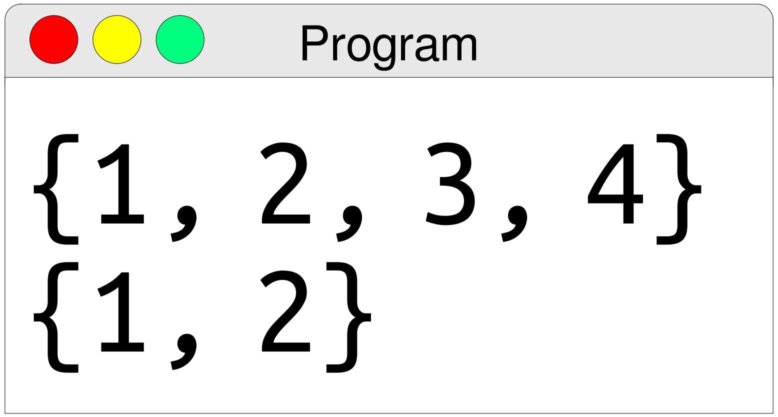
Set Refresher

- 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}
```

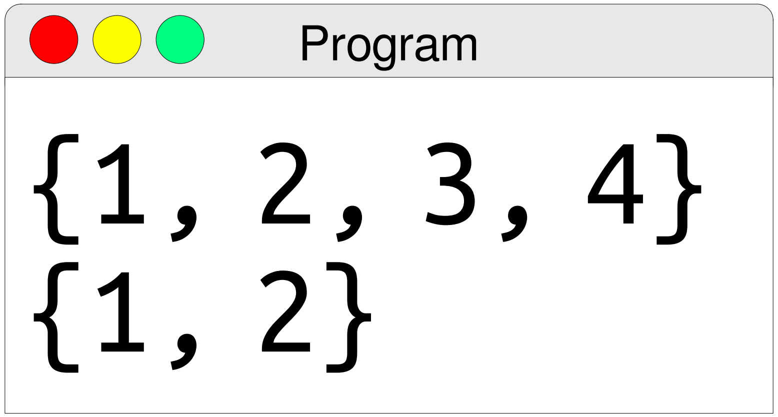
Set Refresher

- 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);
```

Formulate a hypothesis, but ***don't post anything in chat just yet***.

Recursion Refresher

- What does this code print?

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

squigglebah(2);
```

Now, ***private chat me your best guess***. Not sure? Just answer “??”

```
squigglebah(2);
```

```
s
```

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

2

int n

s

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

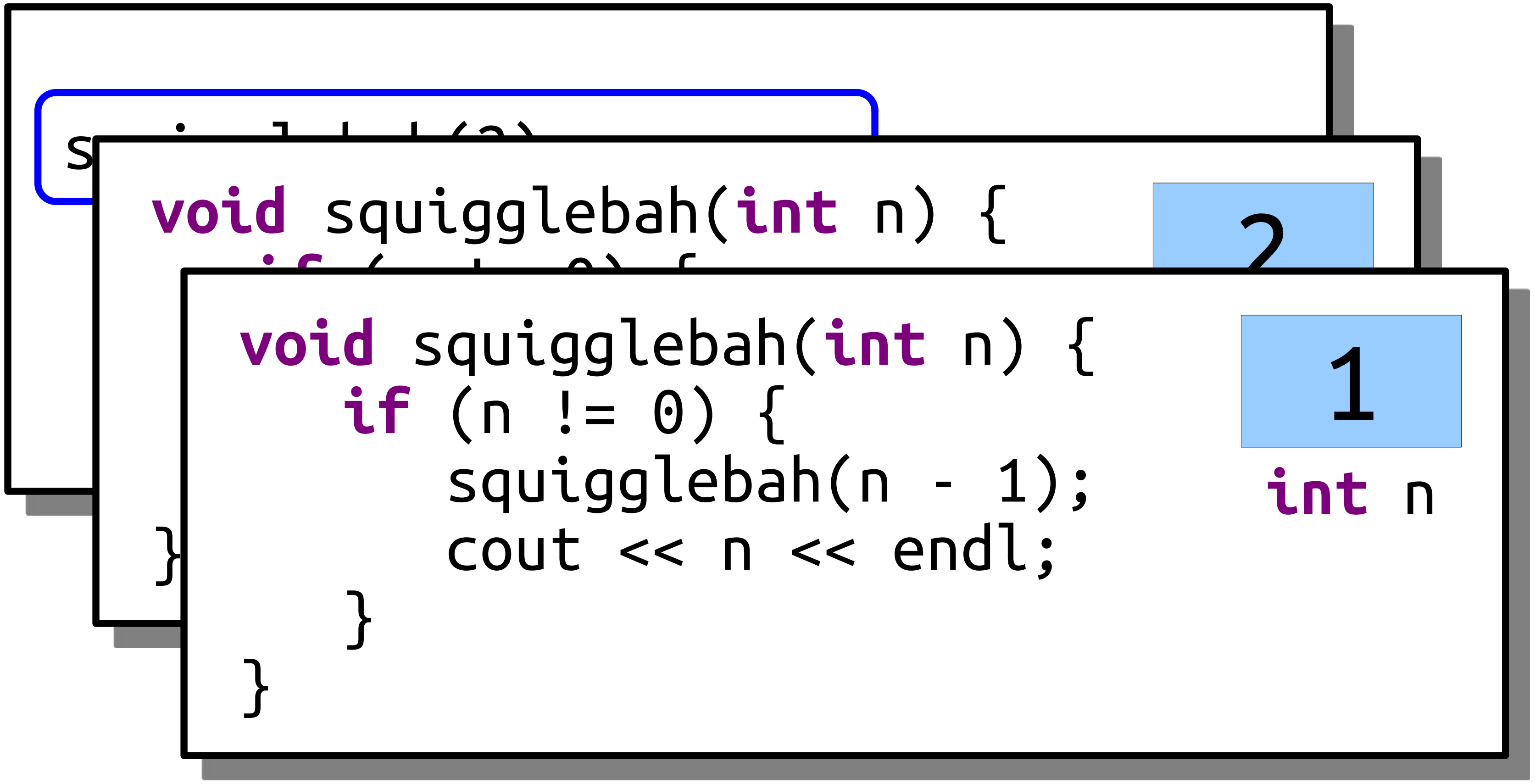
2

int n

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

2

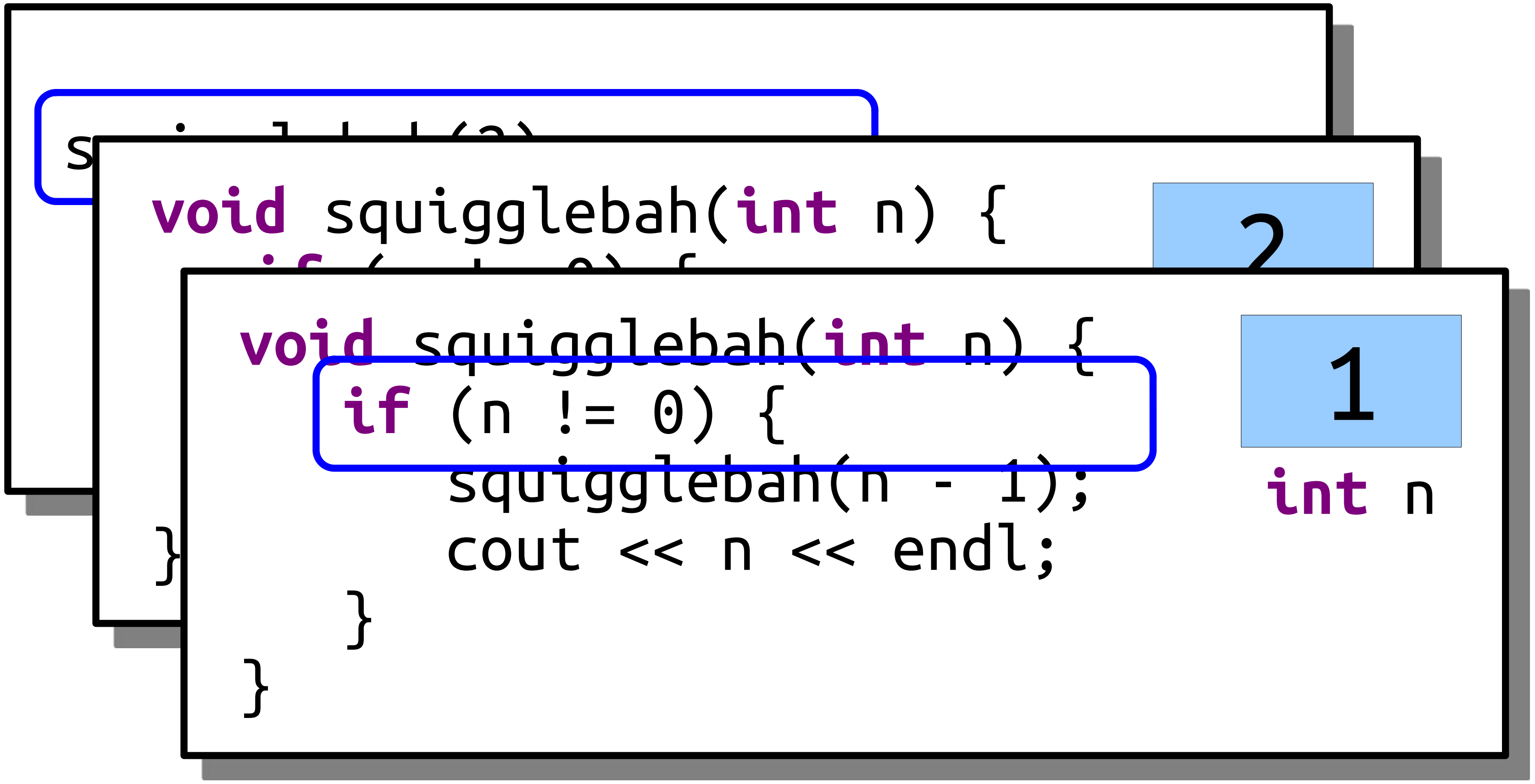
int n



```
void squigglebah(int n) {
```

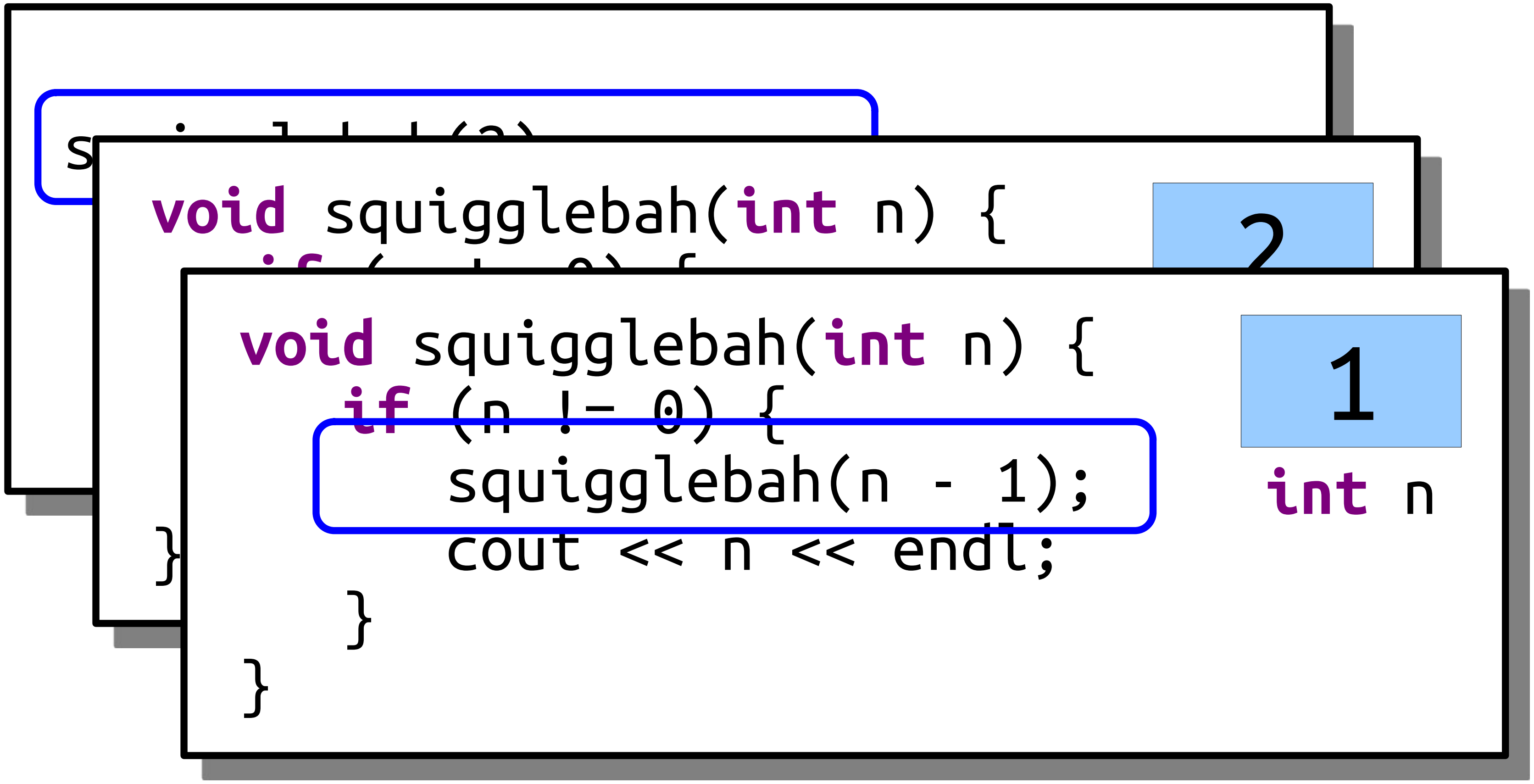
```
}
```

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



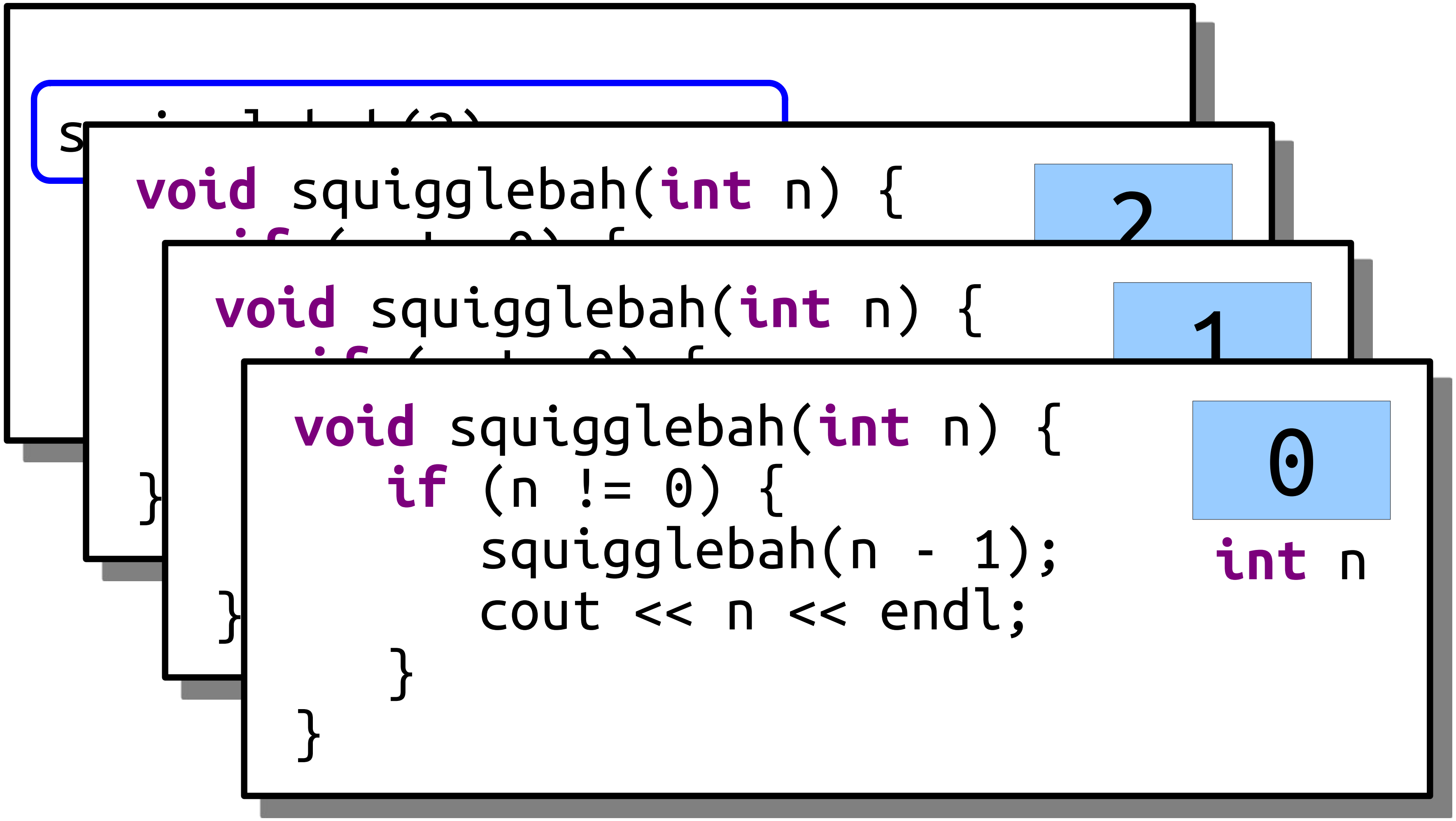
```
void squigglebah(int n) {
```

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

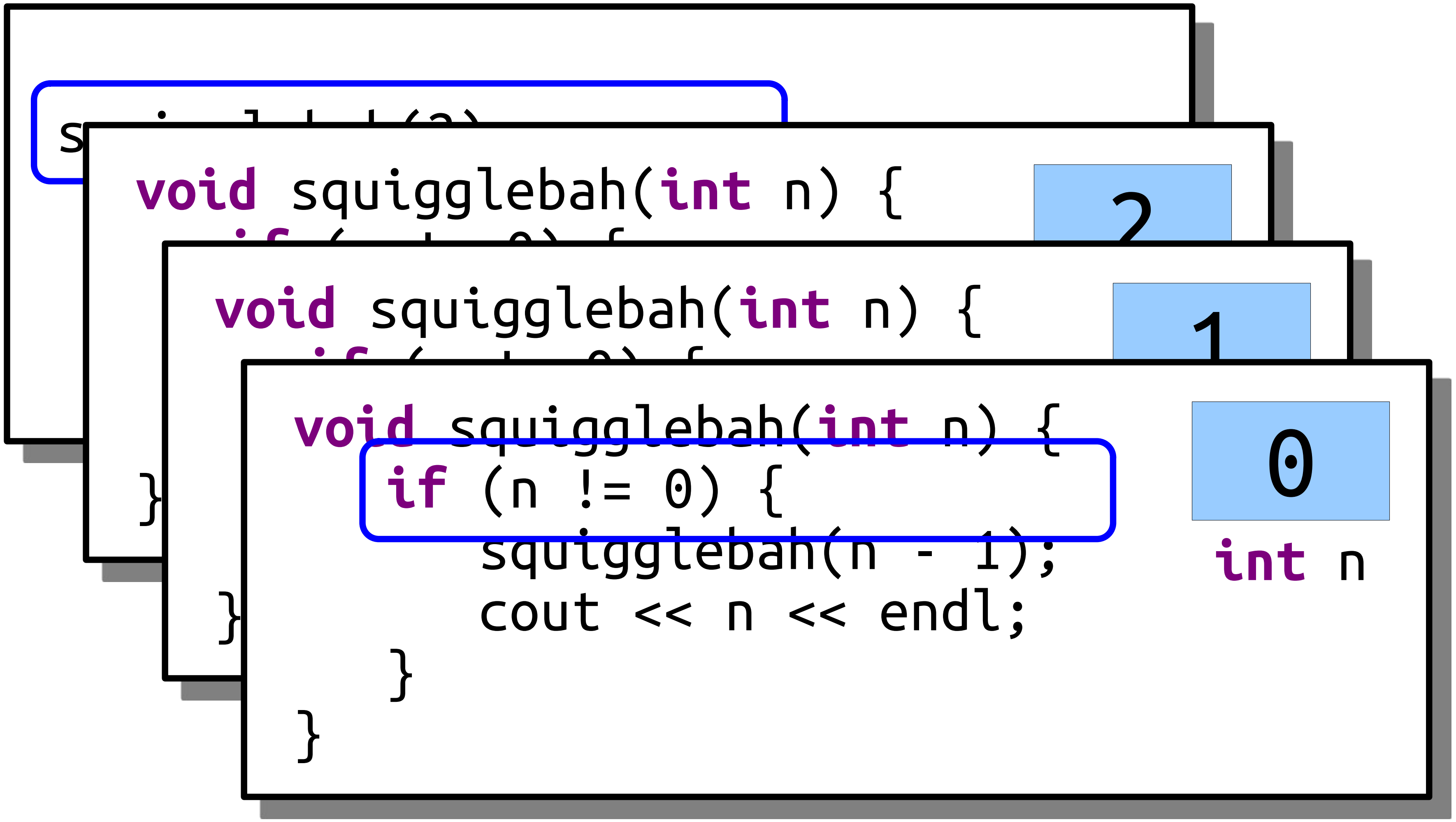


```
void squigglebah(int n) {
```

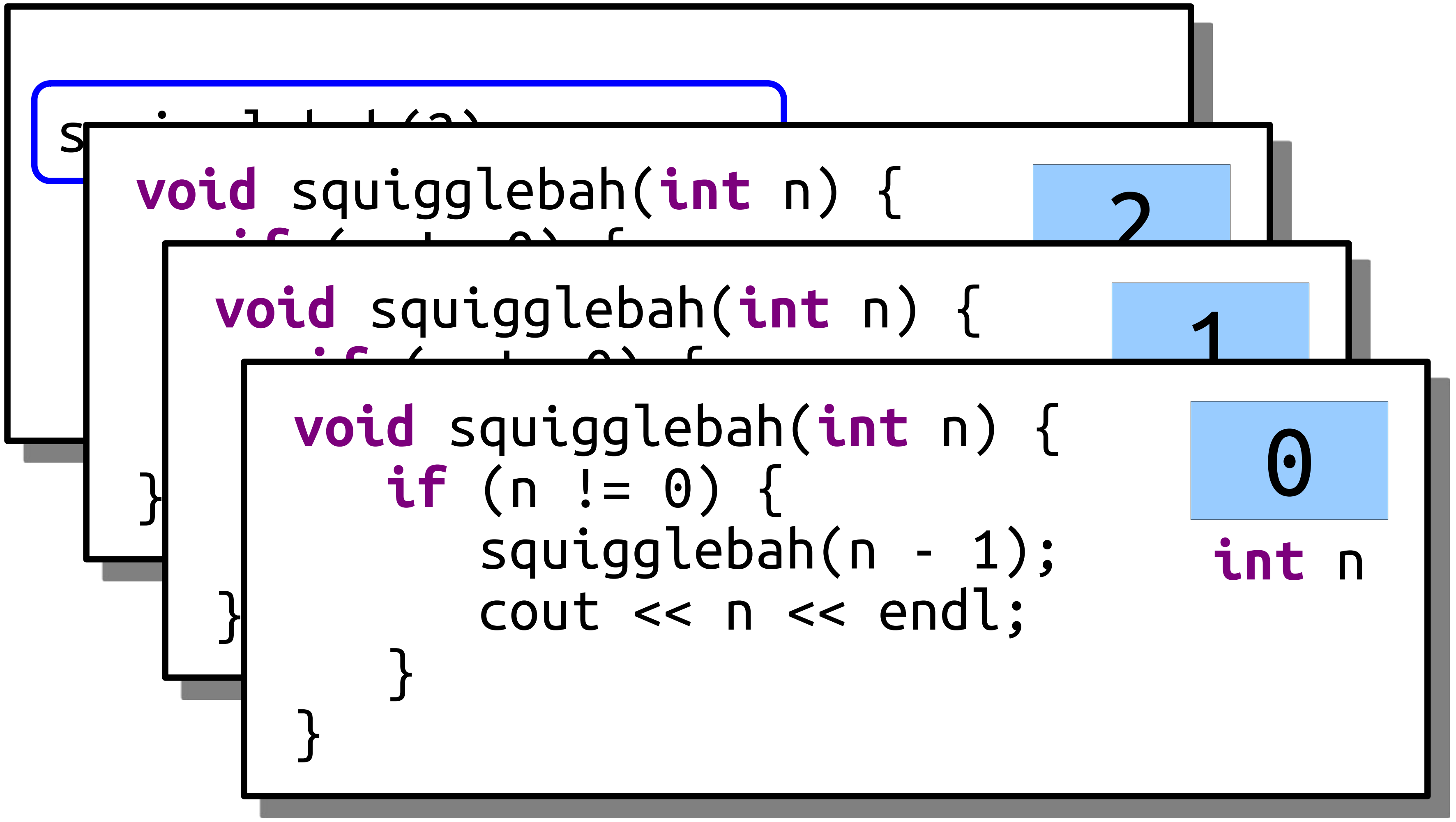
```
void squigglebah(int n) {
    if (n != 0) {
        squigglebah(n - 1);
        cout << n << endl;
    }
}
```



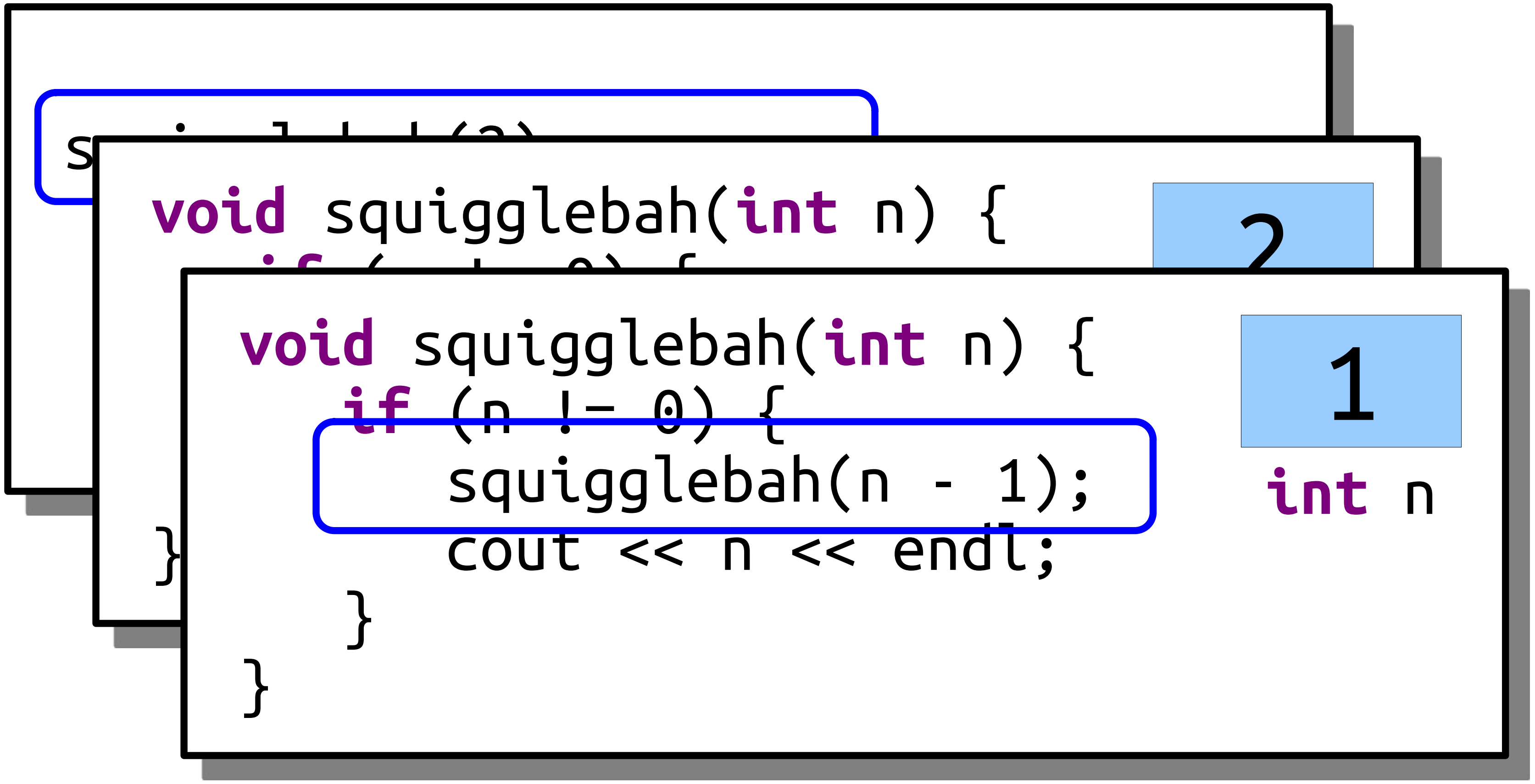
```
void squigglebah(int n) {
    if (n != 0) {
        squigglebah(n - 1);
        cout << n << endl;
    }
}
```



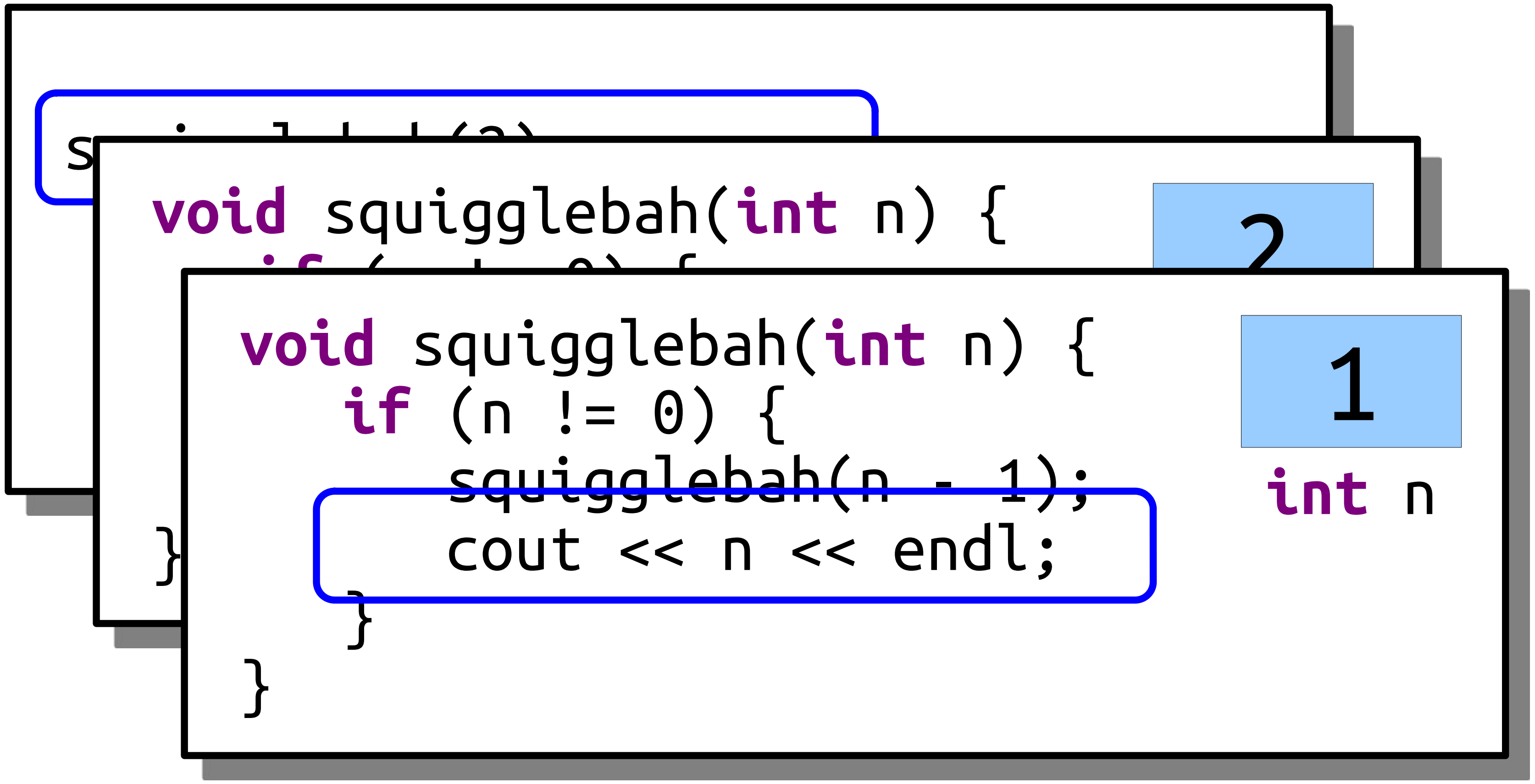
```
void squigglebah(int n) {
    if (n != 0) {
        squigglebah(n - 1);
        cout << n << endl;
    }
}
```



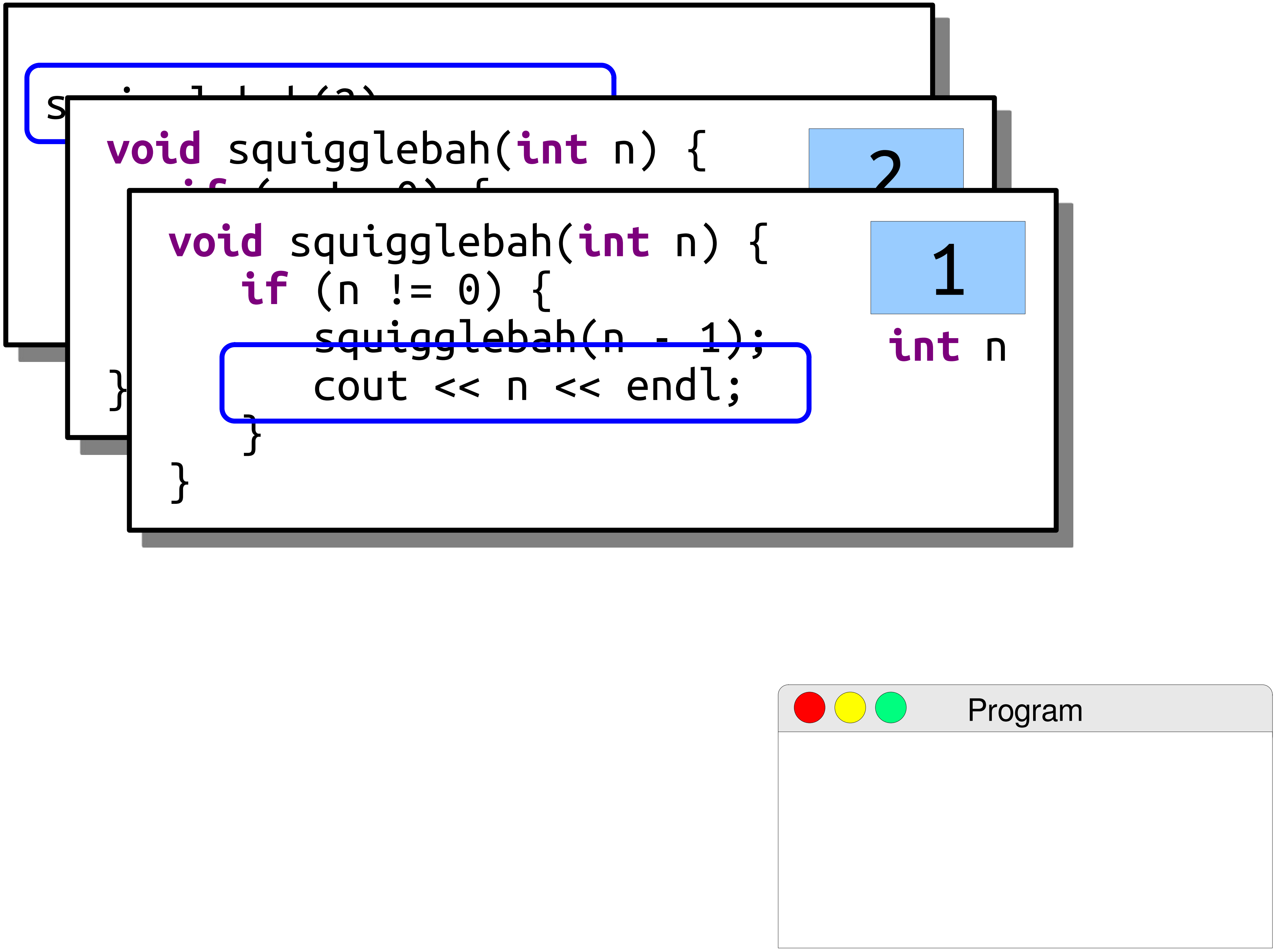
```
void squigglebah(int n) {
    if (n != 0) {
        squigglebah(n - 1);
        cout << n << endl;
    }
}
```



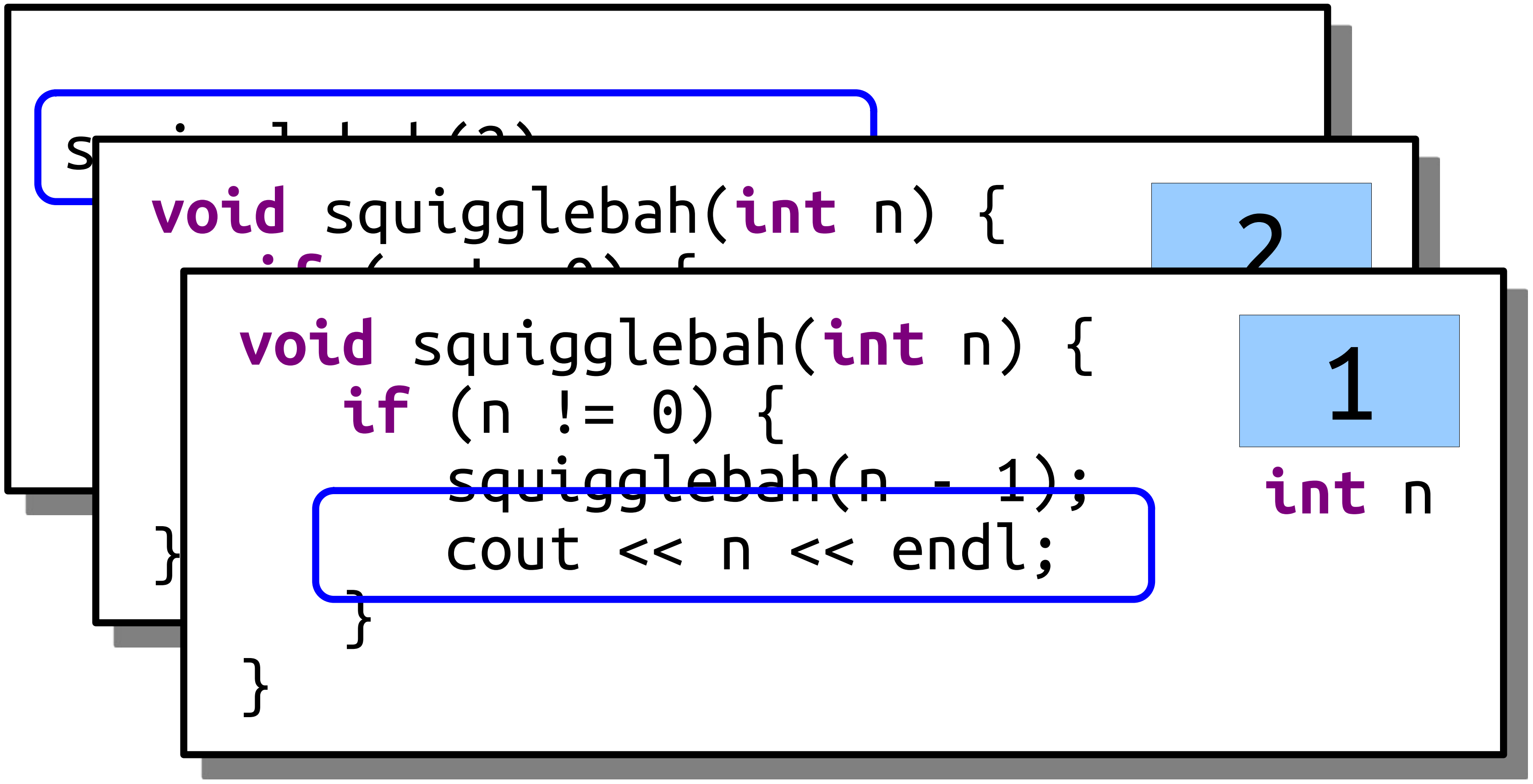
```
void squigglebah(int n) {
    if (n != 0) {
        squigglebah(n - 1);
        cout << n << endl;
    }
}
```



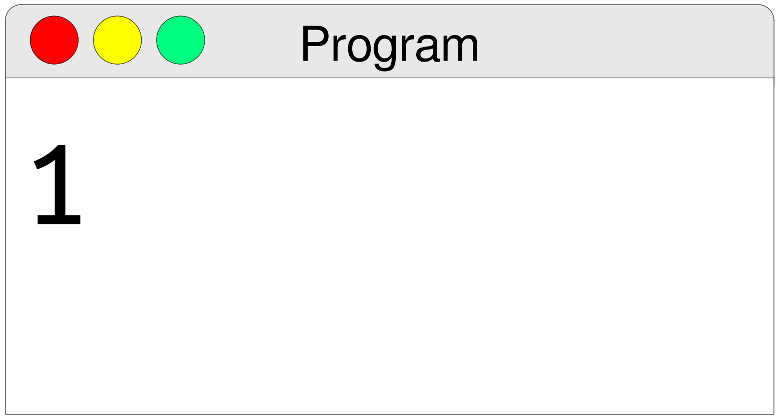
```
void squigglebah(int n) {
    if (n != 0) {
        squigglebah(n - 1);
        cout << n << endl;
    }
}
```



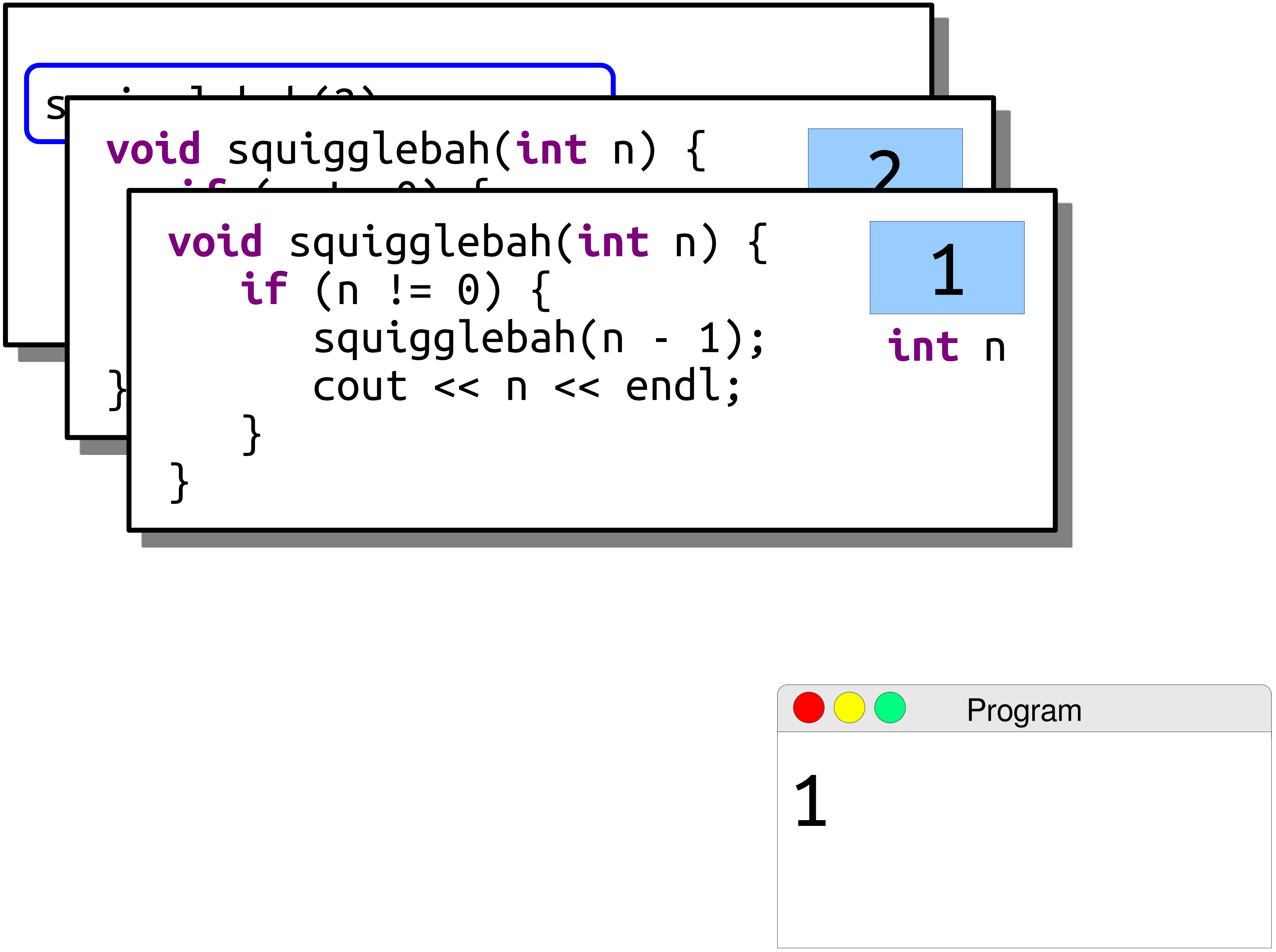
```
void squigglebah(int n) {
    if (n != 0) {
        squigglebah(n - 1);
        cout << n << endl;
    }
}
```



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



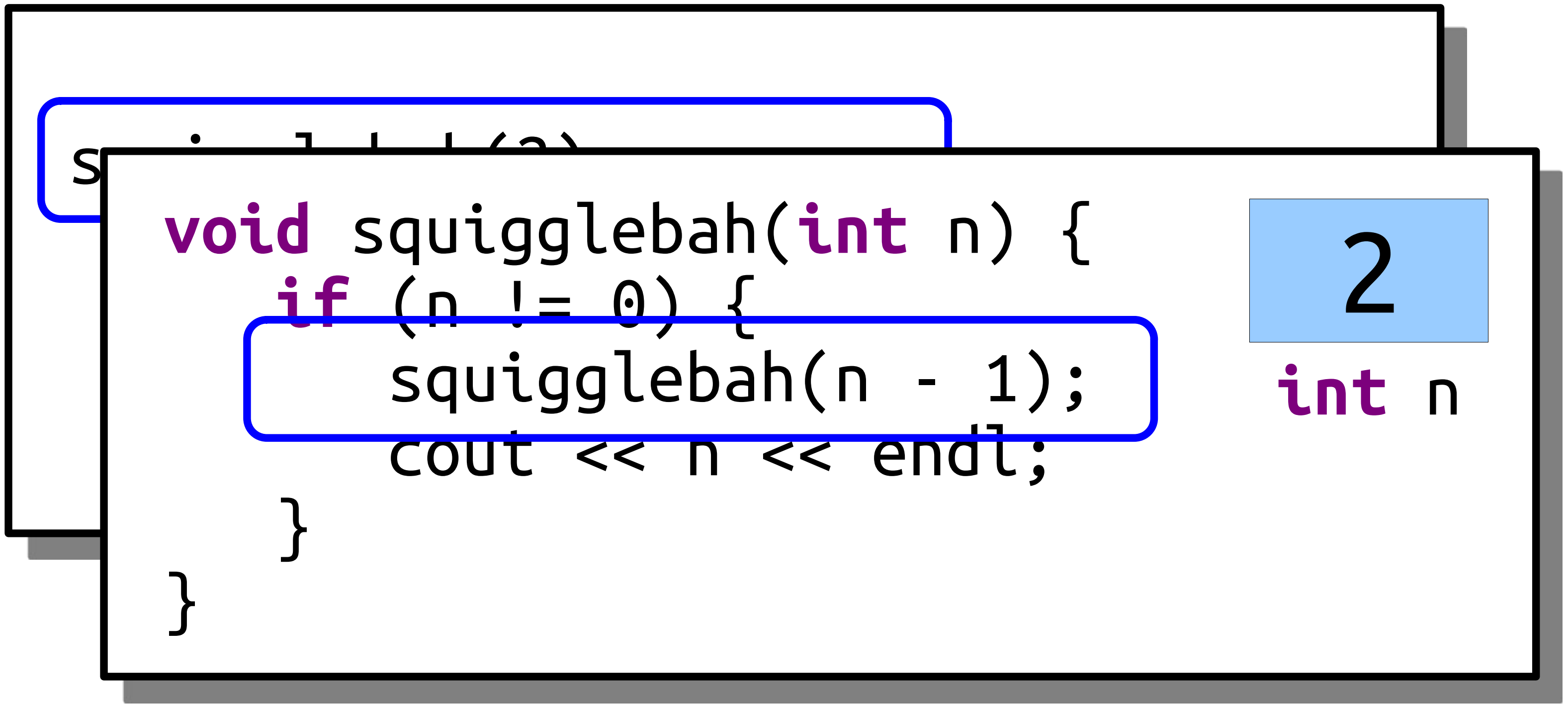
```
1
```



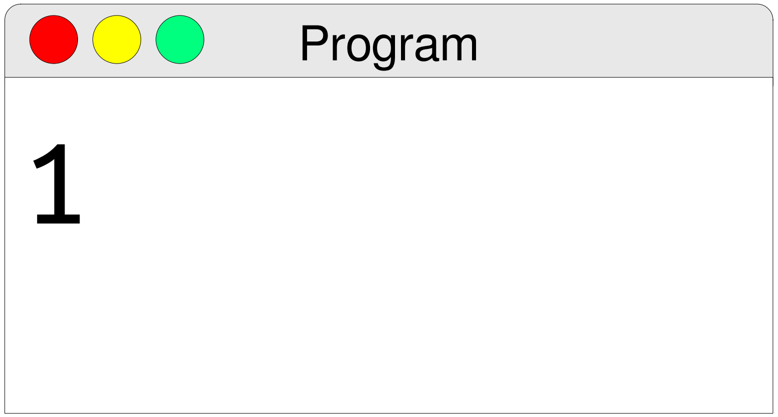
```
void squigglebah(int n) {
```

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

```
1
```



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



```
1
```

```
s
```

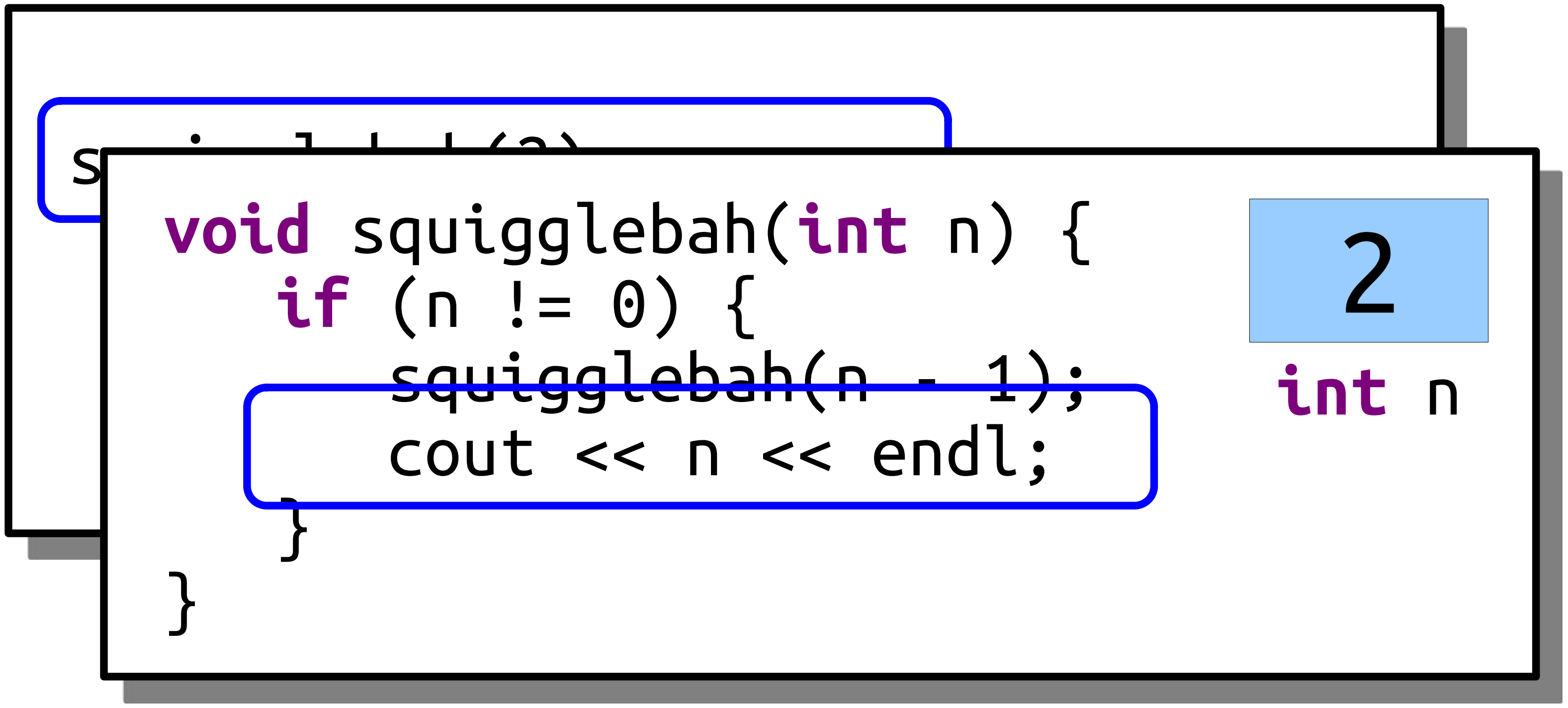
```
void squigglebah(int n) {
    if (n != 0) {
        squigglebah(n - 1);
        cout << n << endl;
    }
}
```

2

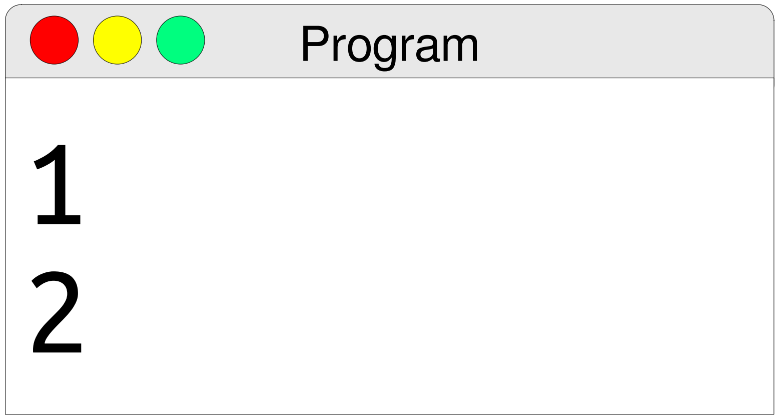
int n

Program

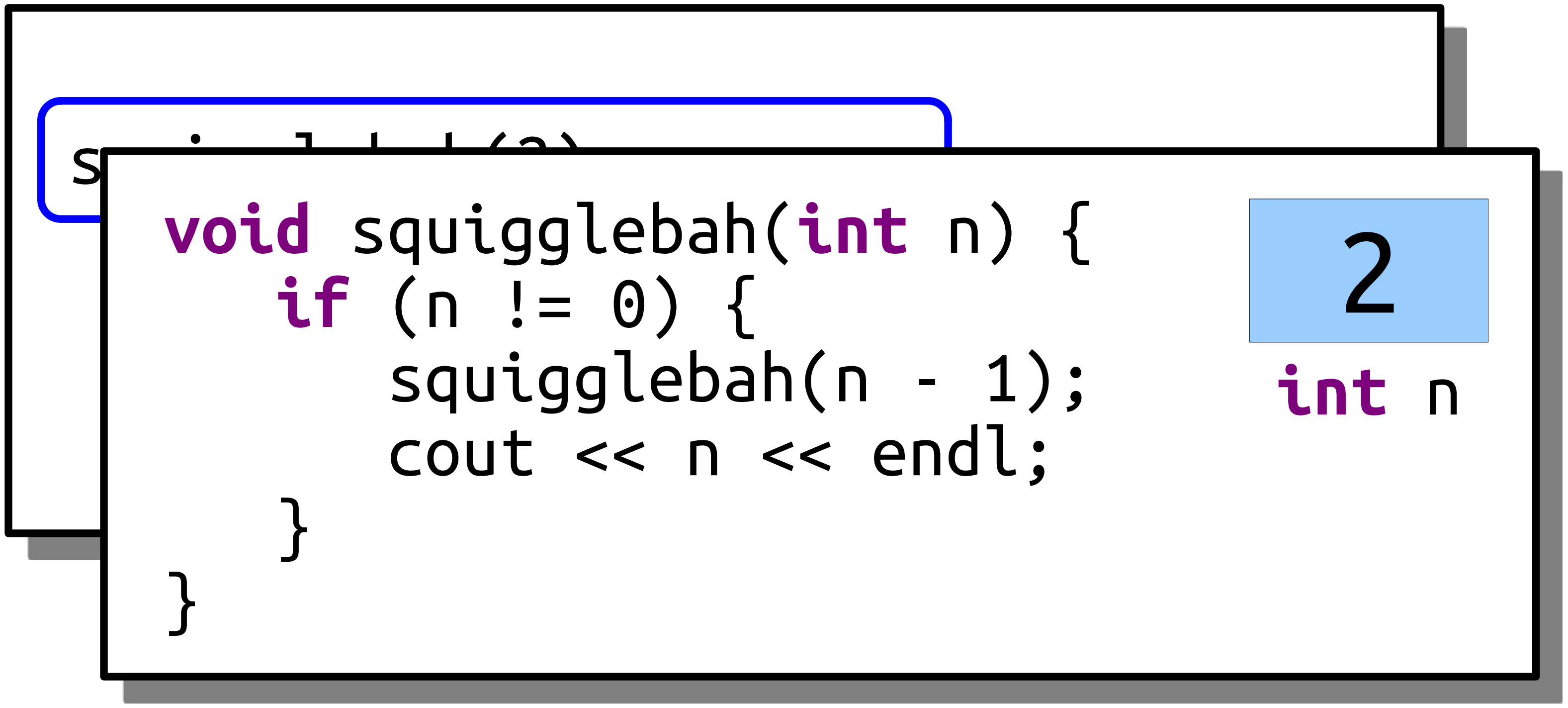
```
1
```



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

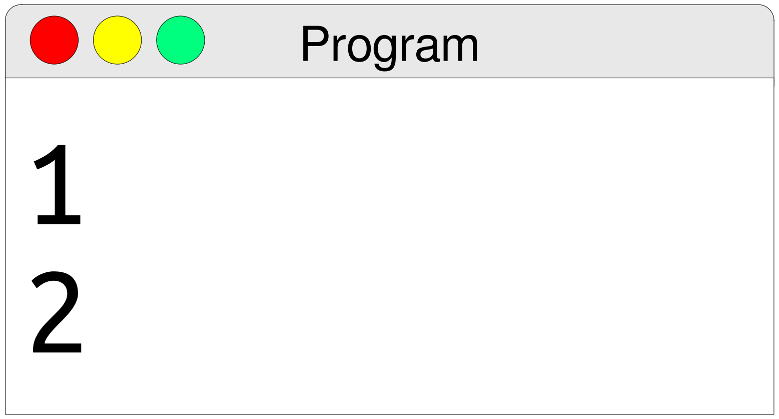


```
1
2
```



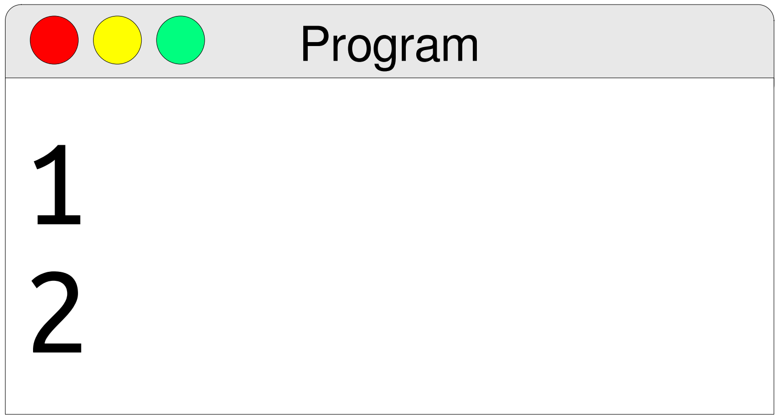
```
s
```

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



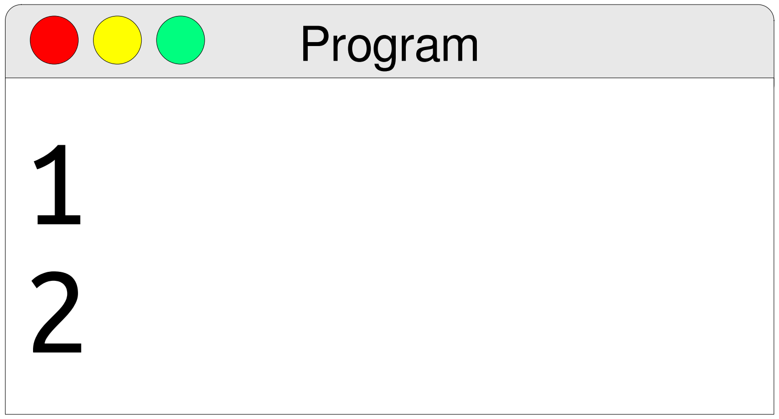
```
1
2
```

```
squigglebah(2);
```



```
1
2
```

```
squigglebah(2);
```



```
1
2
```

The Recursive Leap of Faith

The Contract

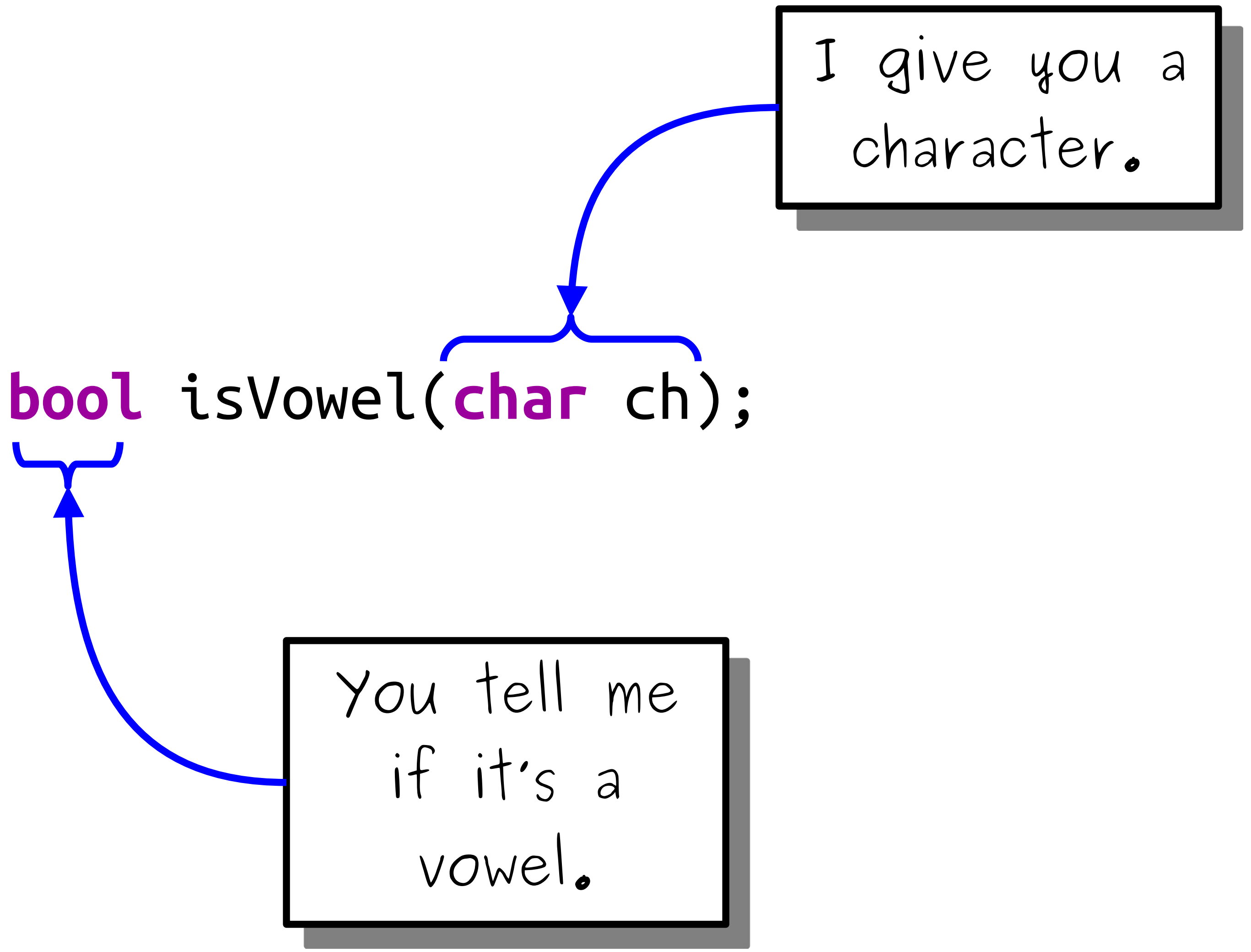
```
bool isVowel(char ch);
```

The Contract

I give you a character.

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

The Contract



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

I give you a character.

You tell me if it's a vowel.

The Contract

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

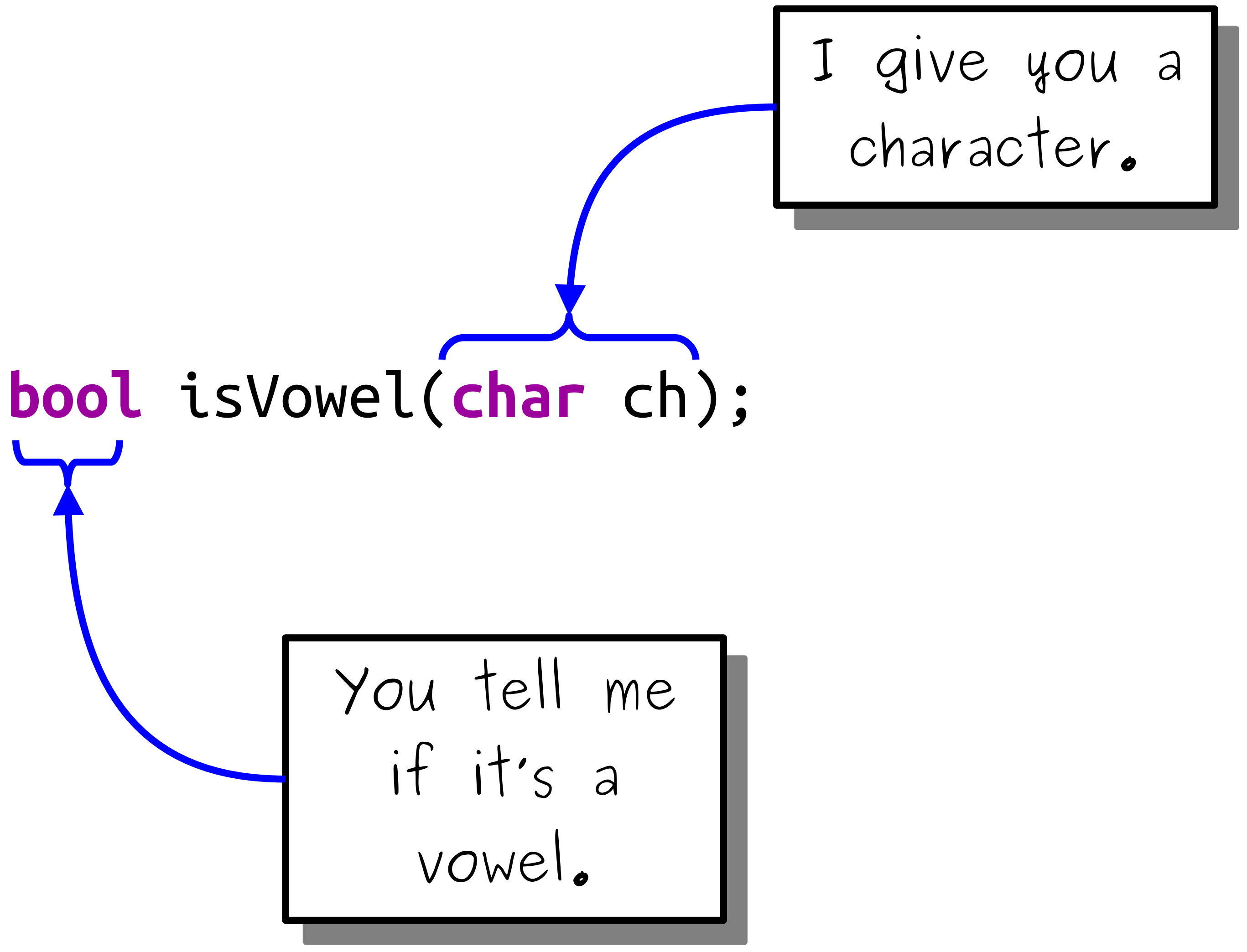
The Contract

```
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



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

The Contract

The Contract

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

The Contract

I give you a string.

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

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) {




}
```

Trusting the Contract

```
bool isVowel(char ch);

bool hasConsecutiveVowels(const string& str) {
  for (int i = 1; i < str.length(); i++) {




  }

}
```

Trusting the Contract

```
bool isVowel(char ch);

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

}
```

Trusting the Contract

```
bool isVowel(char ch);

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

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;
    }
  }
  return false;
}
```

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

The Contract

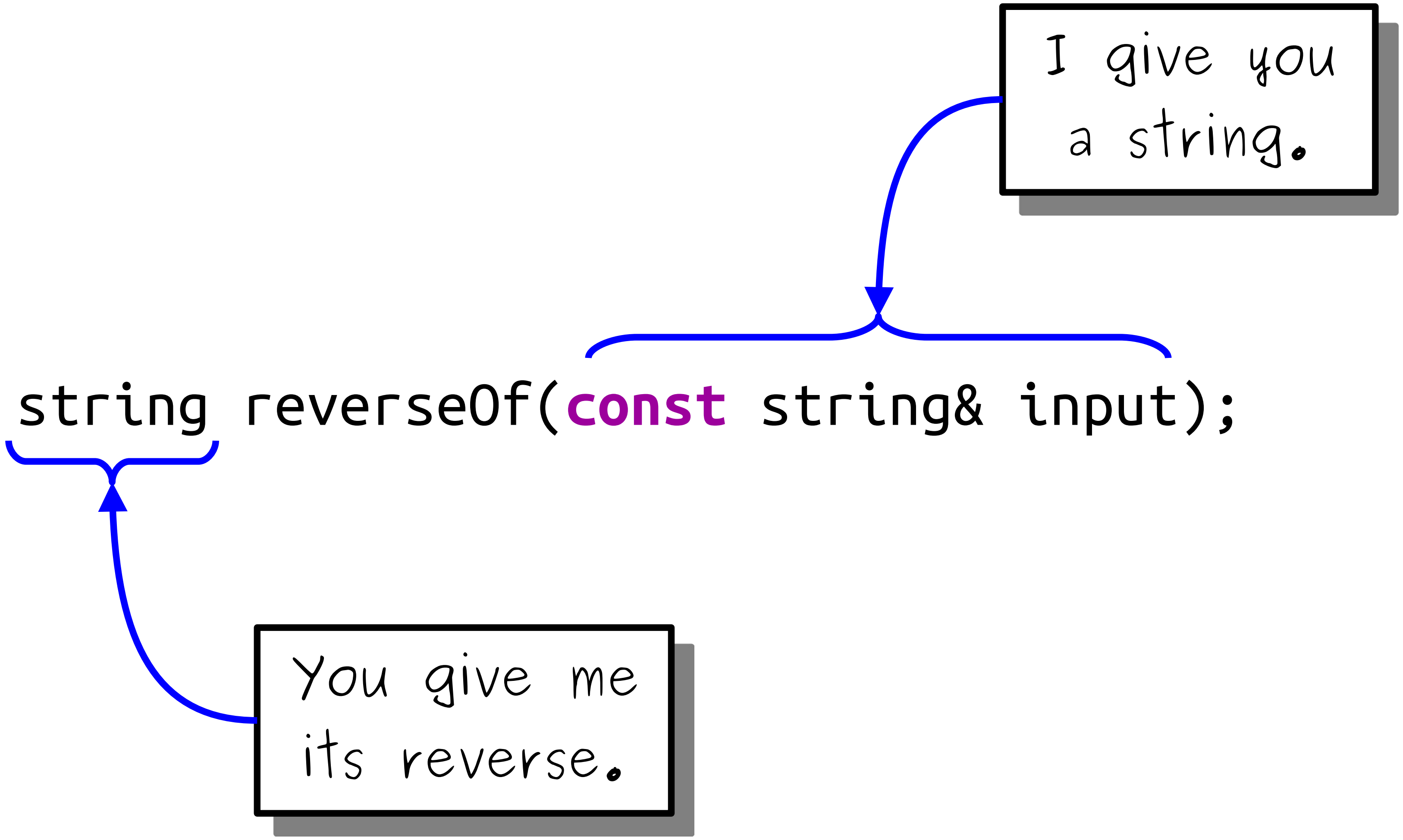
```
string reverseOf(const string& input);
```

The Contract

I give you a string.

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

The Contract



I give you a string.

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

You give me its reverse.

Trusting the Contract

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

string reverseOf(const string& input) {




}
```

Trusting the Contract

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

string reverseOf(const string& input) {
    if (input == "") {

    } else {

    }
}
```

Trusting the Contract

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

string reverseOf(const string& input) {
    if (input == "") {
        return "";
    } else {

    }
}
```

Trusting the Contract

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

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

Trusting the Contract

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

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

Trusting the Contract

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

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

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

The Contract

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

The Contract

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

The Contract

Draw me a tree...

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

The Contract

Draw me a tree…

… at this position …

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

The Contract

Draw me a tree…

… at this position …

… that's this big …

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

The Contract

Draw me a tree…

… at this position …

… that's this big …

… facing this way …

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

The Contract

Draw me a tree…

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

… 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, double angle,
              int order) {





}
```

Trusting the Contract

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

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






}
```

Trusting the Contract

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

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

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



}
```

Trusting the Contract

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

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

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

    draw a tree angling to the left
    draw a tree angling to the right
}
```

Trusting the Contract

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

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

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

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

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!

Assignment 3

- Assignment 3 (***Recursion!***) goes out today. It’s due next Friday at 10:30AM.
  - Play around with recursion and recursive problem-solving!
- ***This assignment may be completed in pairs***. Some reminders:
  - You are not required to work in a pair. It’s totally fine to work independently.
  - If you do work in a pair, you must work with someone else in your discussion section.
  - ***Work together, not separately***. Doing only half the assignment teaches you less than half the concepts. Working collaboratively and interactively with your partner will improve your learning outcomes.

LaIR Updates

- Starting Sunday, there will be two ways to get help at the LaIR.
- ***In-Person:***
  - Visit ***Room 353 of the Durand building***, next to the Engineering Quad.
    - Take the elevator to the third floor and turn right. Your ID card will let you in at the building's front entrance.
    - Please only visit Durand 353 unless directed otherwise – other people may be working in the building and we haven't reserved other rooms there.
    - Exercise common courtesy with the building: please clean up any messes you make, etc.
- ***Remotely:***
  - You can sign up as you did before using the link below rather than the previous OhYay link.
- In either case, use the following link to sign up for help:

☞ ***https://cs198.stanford.edu/lair*** ☜

- If you're in person, say which room you're in when signing up. If you're remote, paste a Zoom link for your location.

Section Updates

- Starting next week, sections will return to in-person instruction.
- Section locations are available online at https://cs198.stanford.edu. Log in to see your section location.

Back to CS106B!

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

e·nu·mer·a·tion

noun

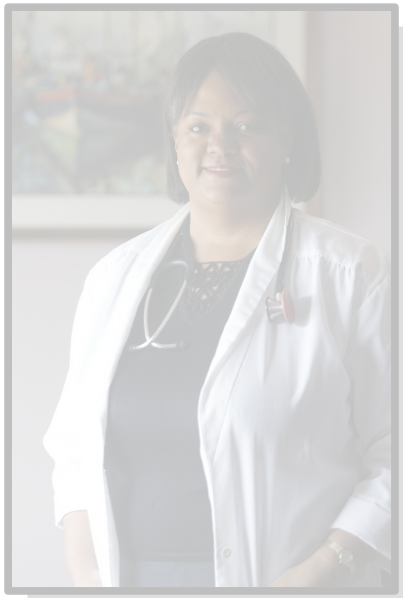
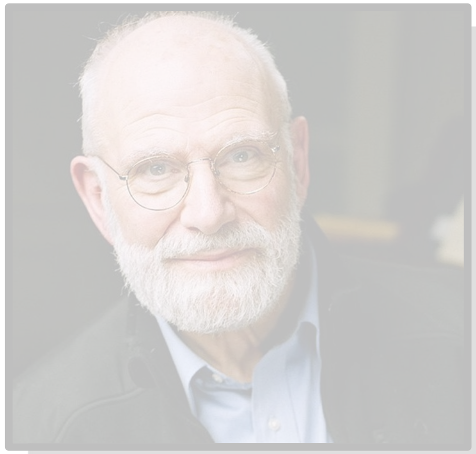
The act of mentioning a number of things one by one.

(Source: Google)

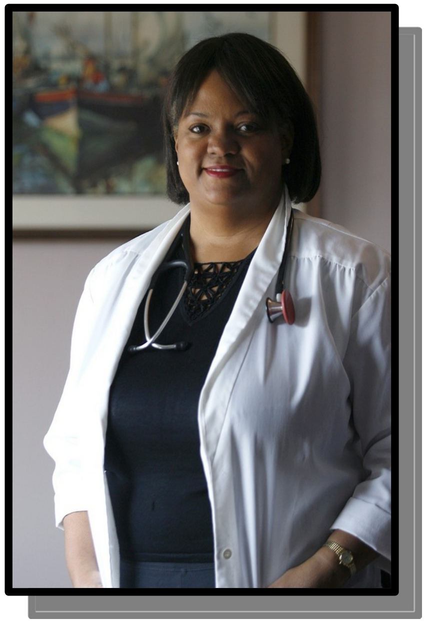
You need to send an emergency team of doctors to an area.

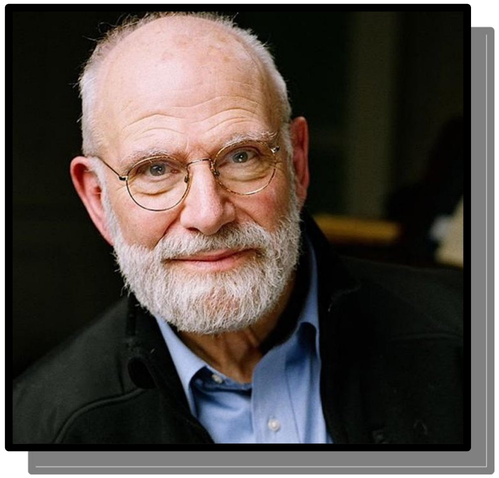
You know which doctors you have available to send.

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

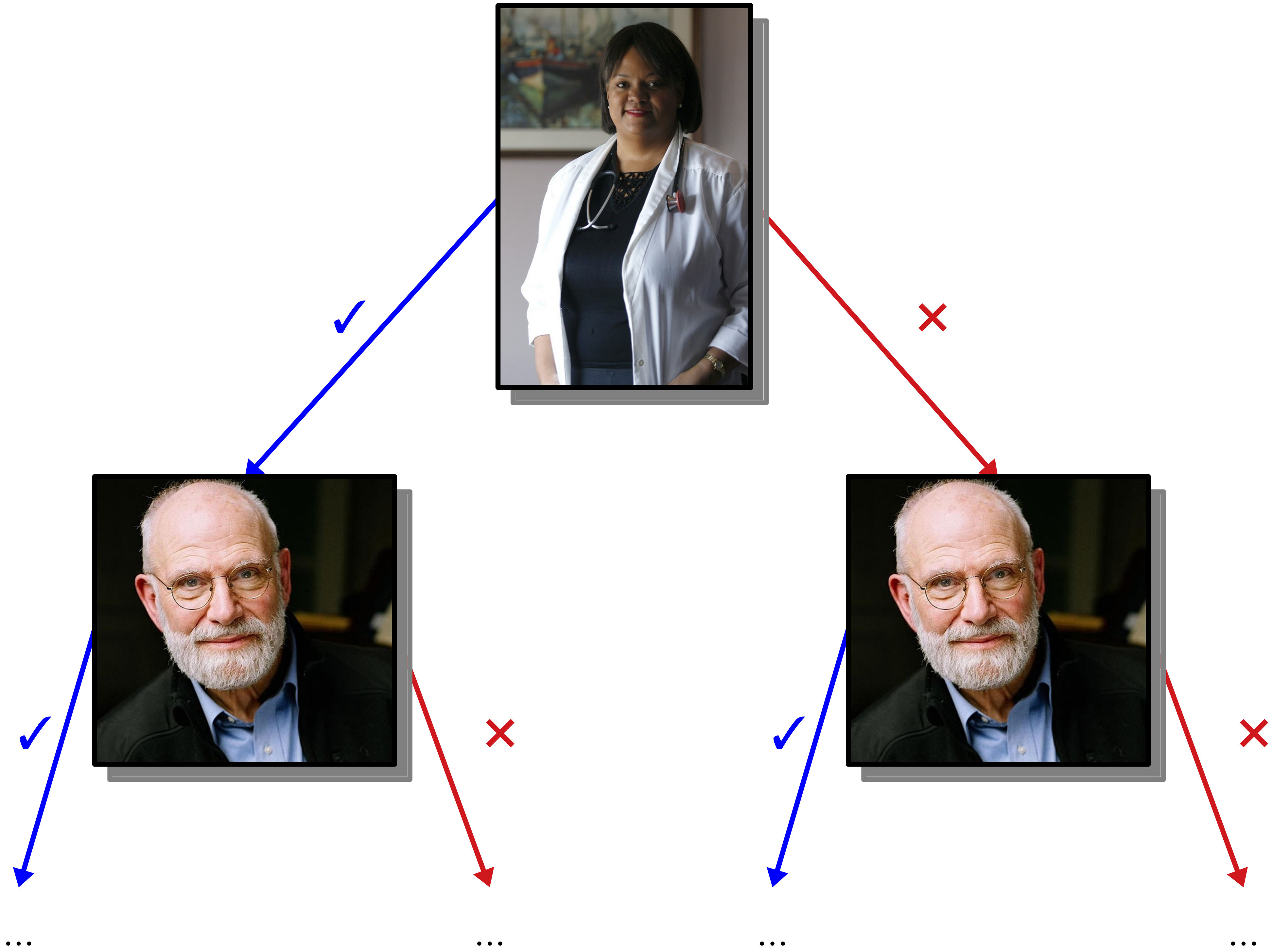



NYU
Medical
Center





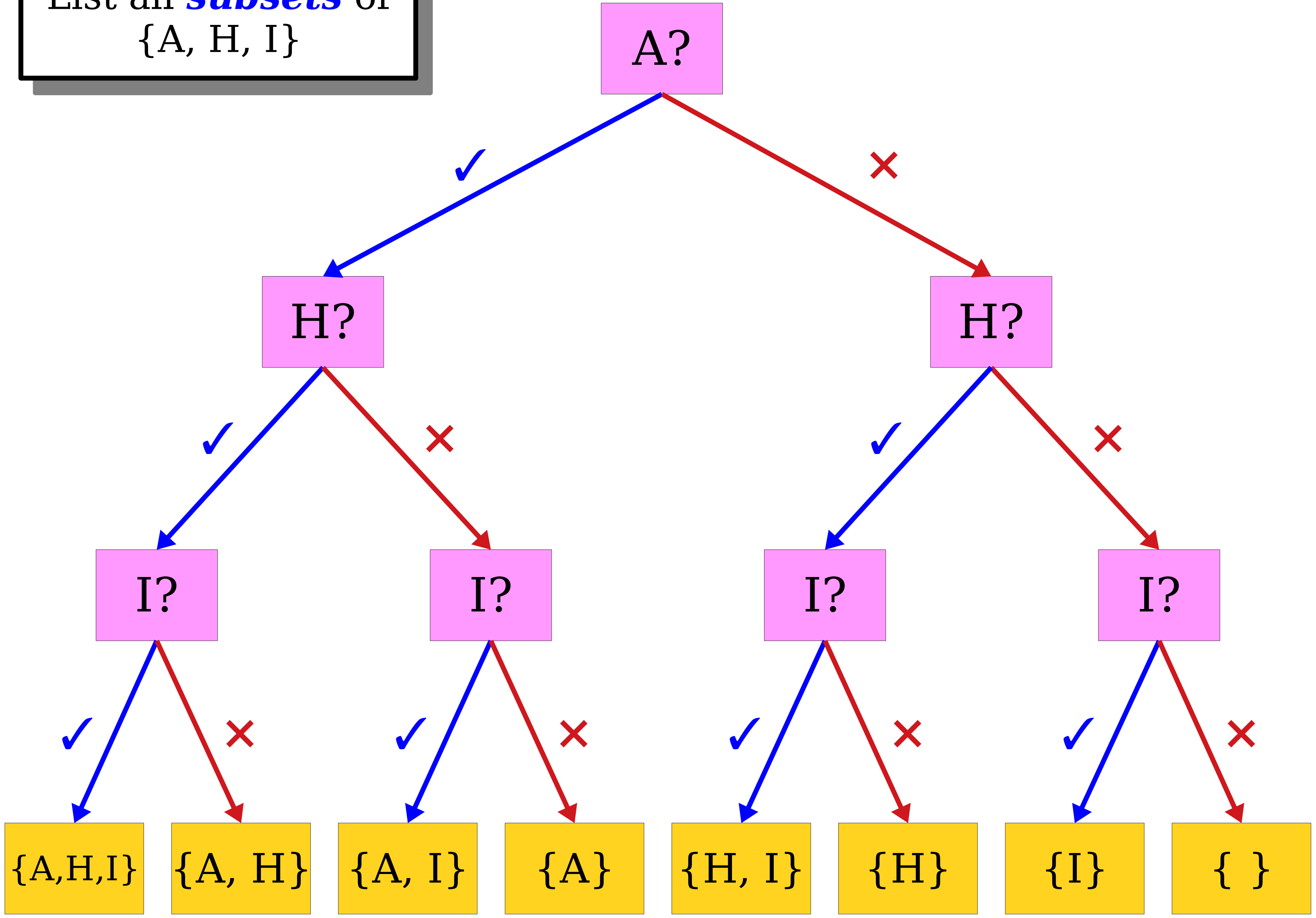
NYU
Medical
Center
Langone Medical Center



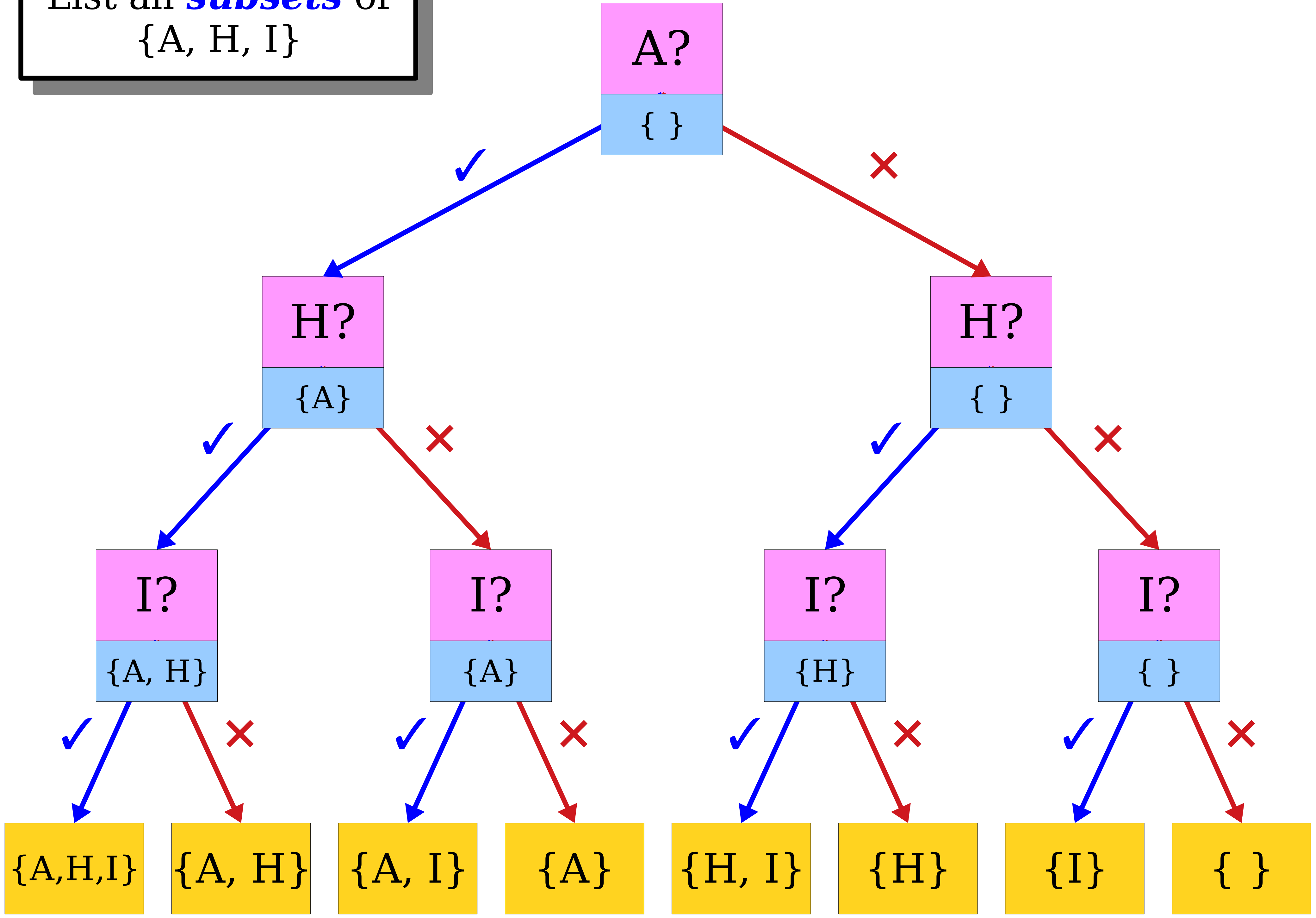
This structure is called a ***decision tree***.

List all ***subsets*** of {A, H, I}

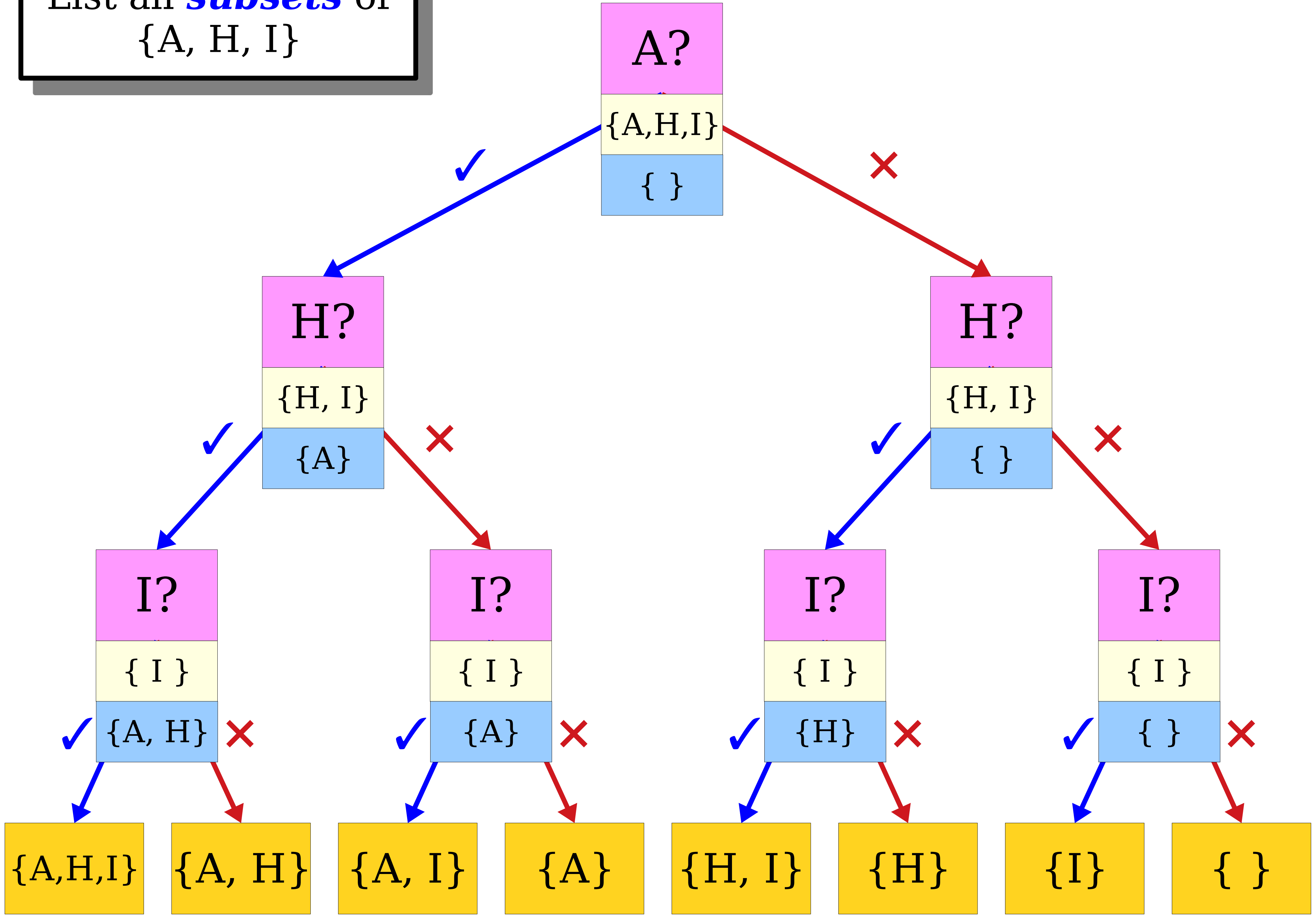
List all ***subsets*** of {A, H, I}



List all ***subsets*** of {A, H, I}



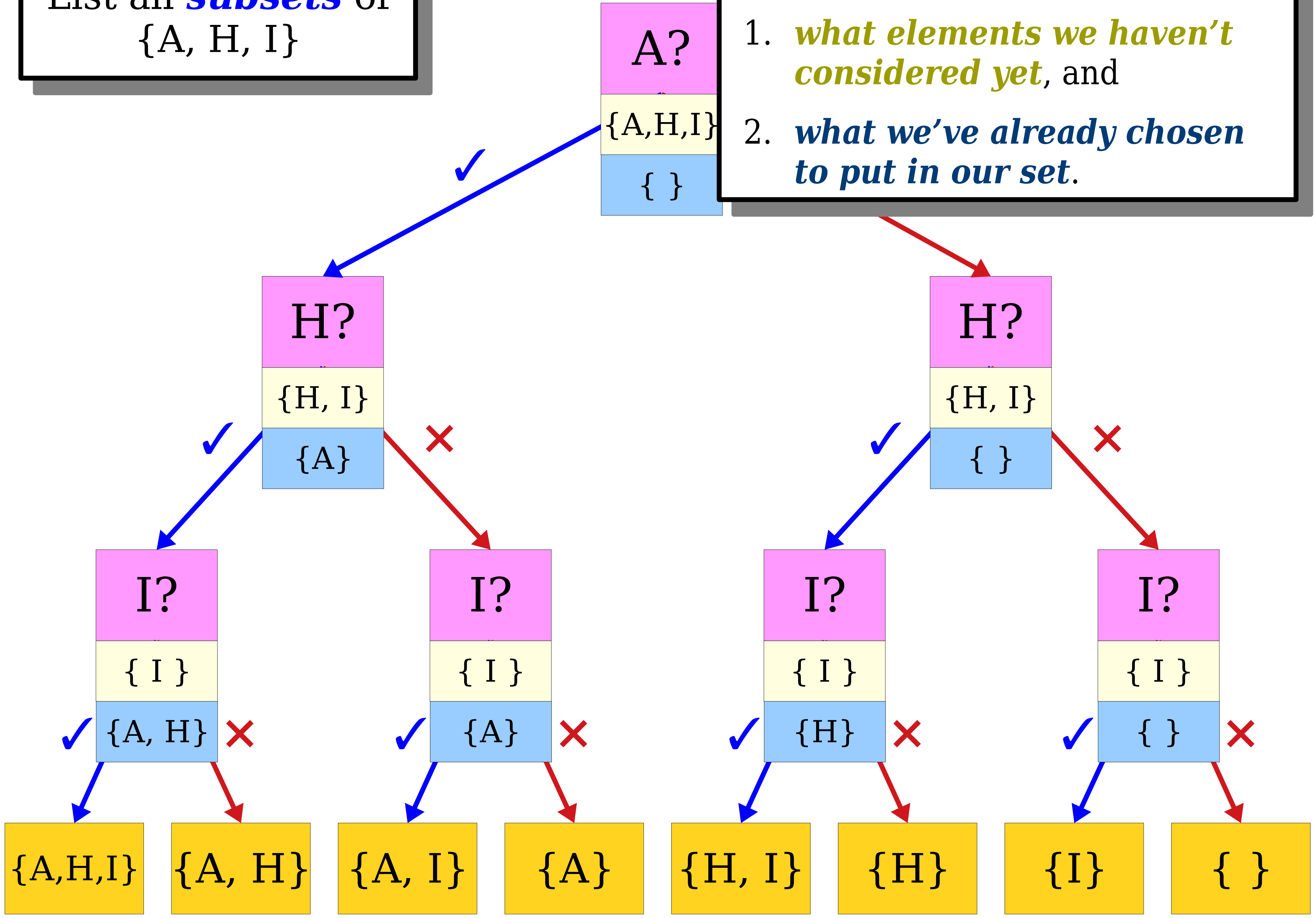
List all ***subsets*** of {A, H, I}



List all ***subsets*** of {A, H, I}

At each step, we need to know

1. ***what elements we haven't considered yet***, and
2. ***what we've already chosen to put in our set***.



The Contract

```
void listSubsetsOf(const Set<int>& elems,
                   const Set<int>& soFar);
```

The Contract

List all the subsets of `elems`…

```
void listSubsetsOf(const Set<int>& elems,
                   const Set<int>& soFar);
```

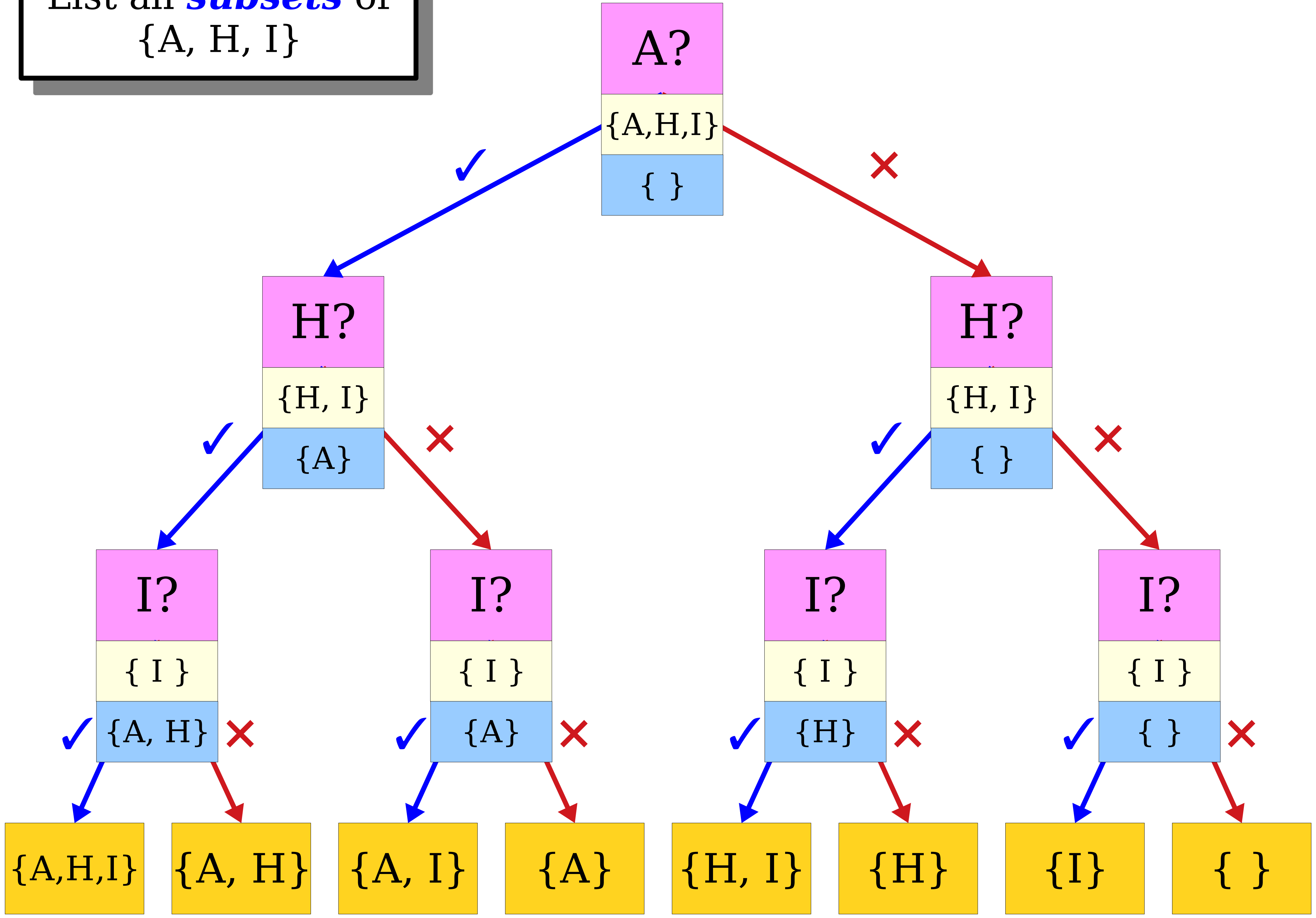
The Contract

List all the subsets of `elems`…

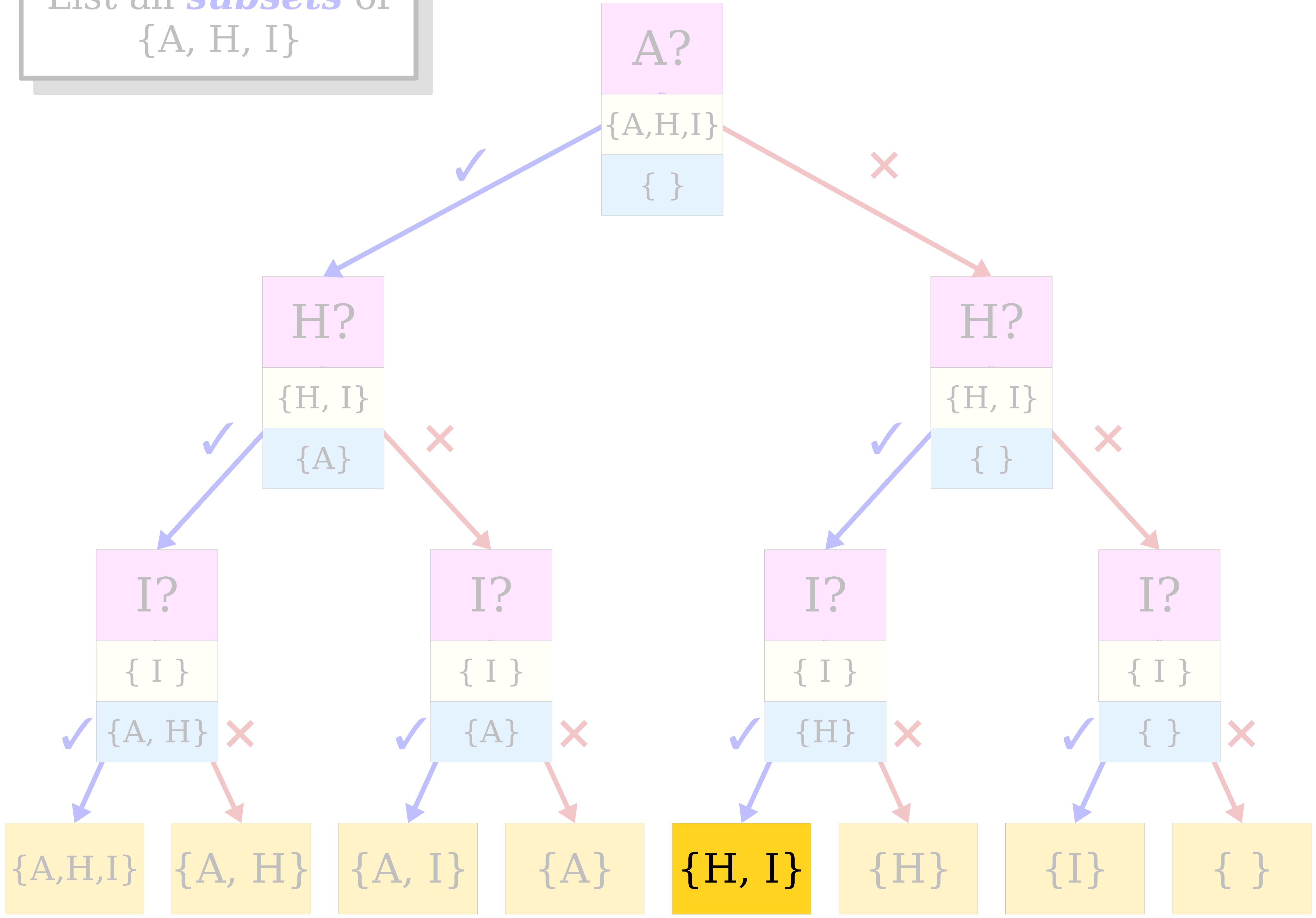
```
void listSubsetsOf(const Set<int>& elems,
                   const Set<int>& soFar);
```

… given that we've already committed to choosing the integers in `soFar`.

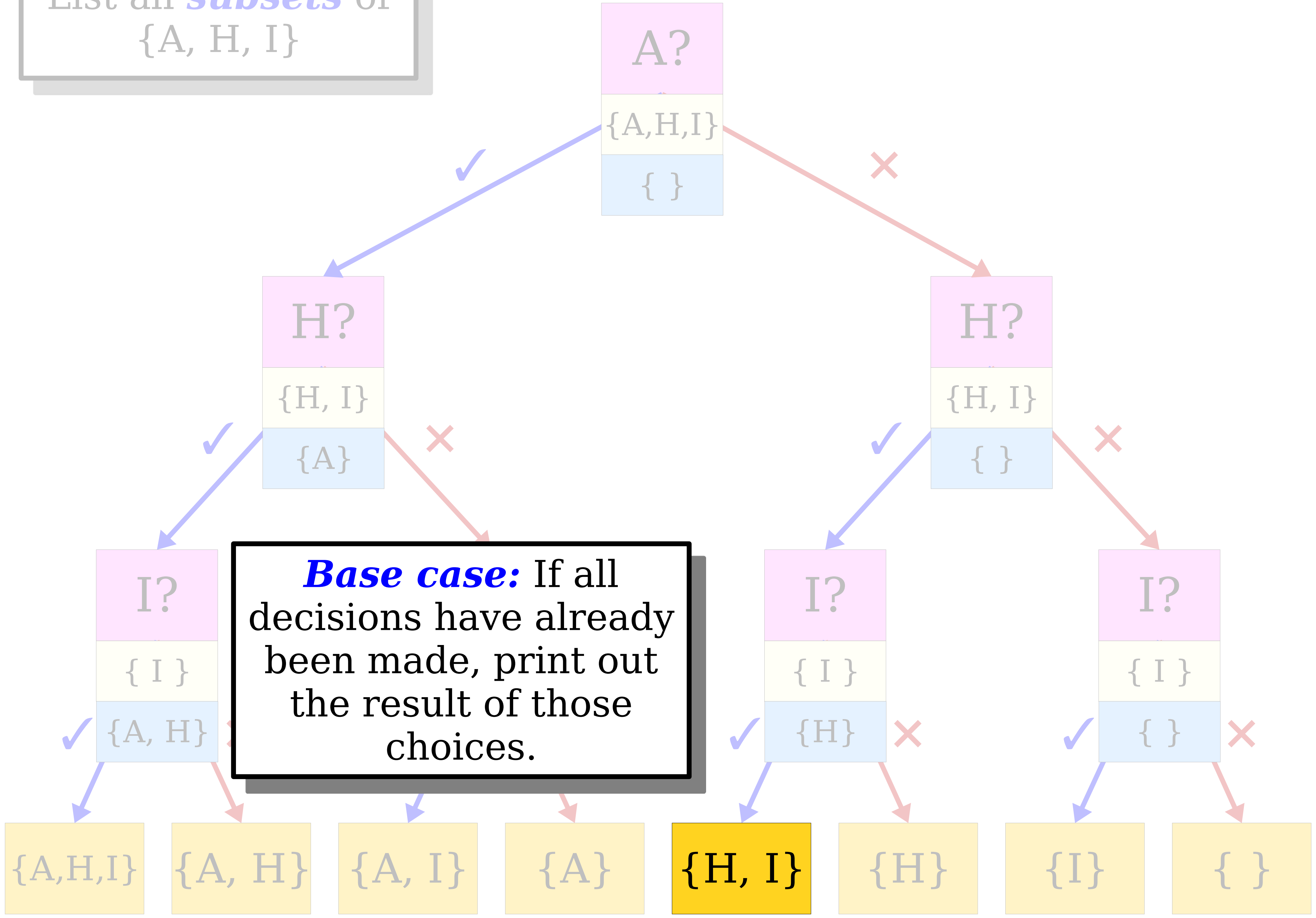
List all ***subsets*** of {A, H, I}



List all ***subsets*** of {A, H, I}

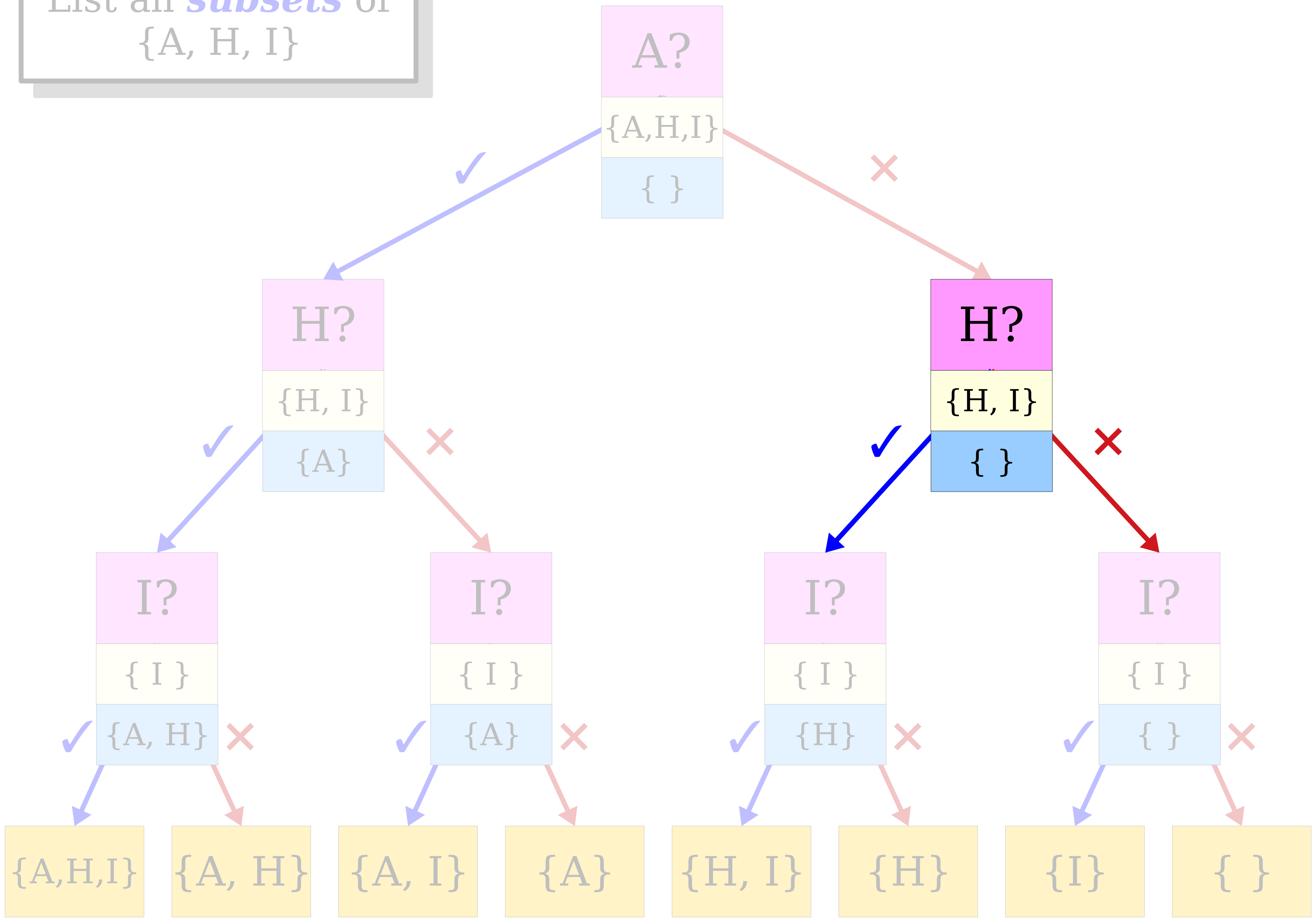


List all ***subsets*** of {A, H, I}

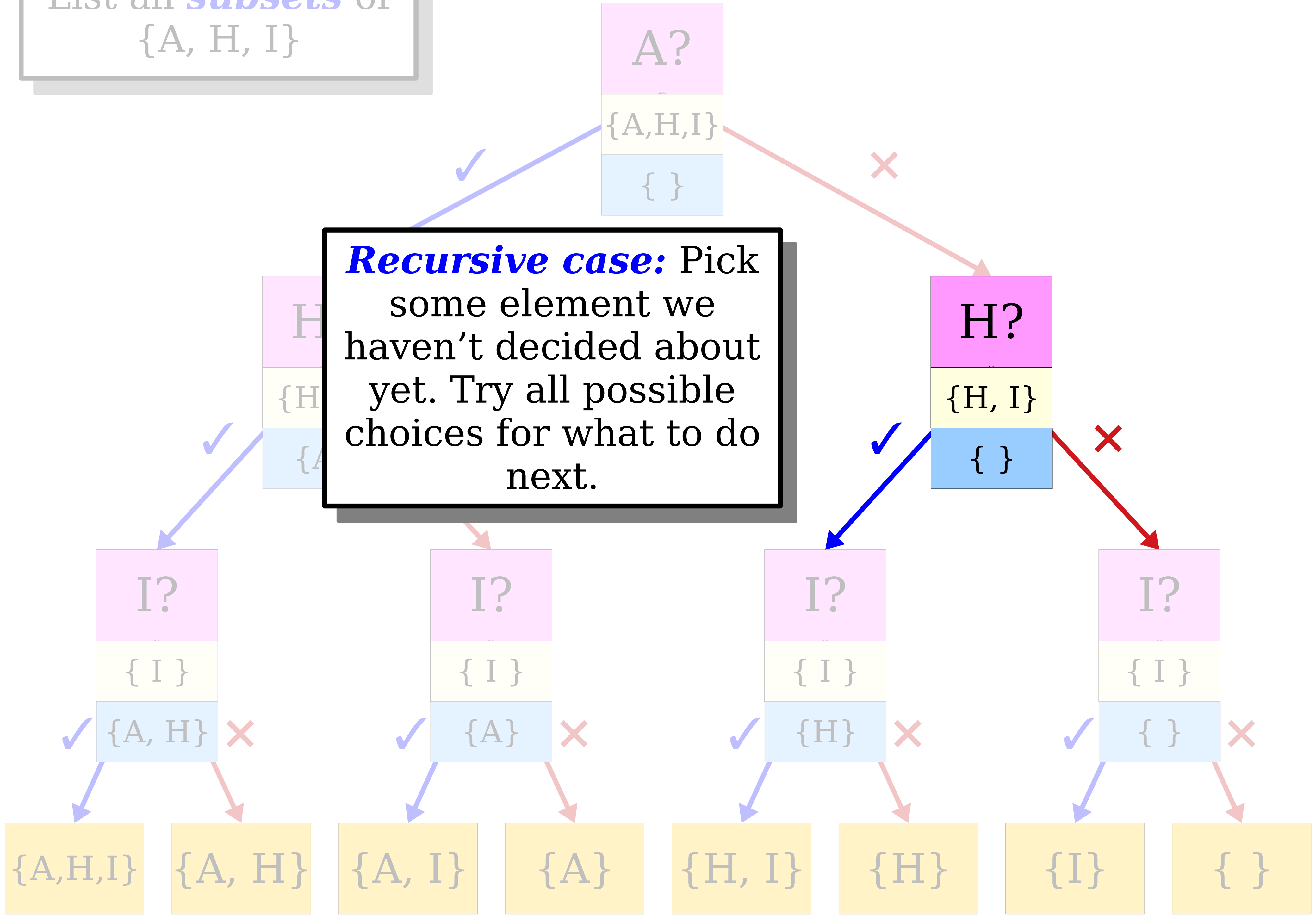


Base case: If all decisions have already been made, print out the result of those choices.

List all ***subsets*** of {A, H, I}

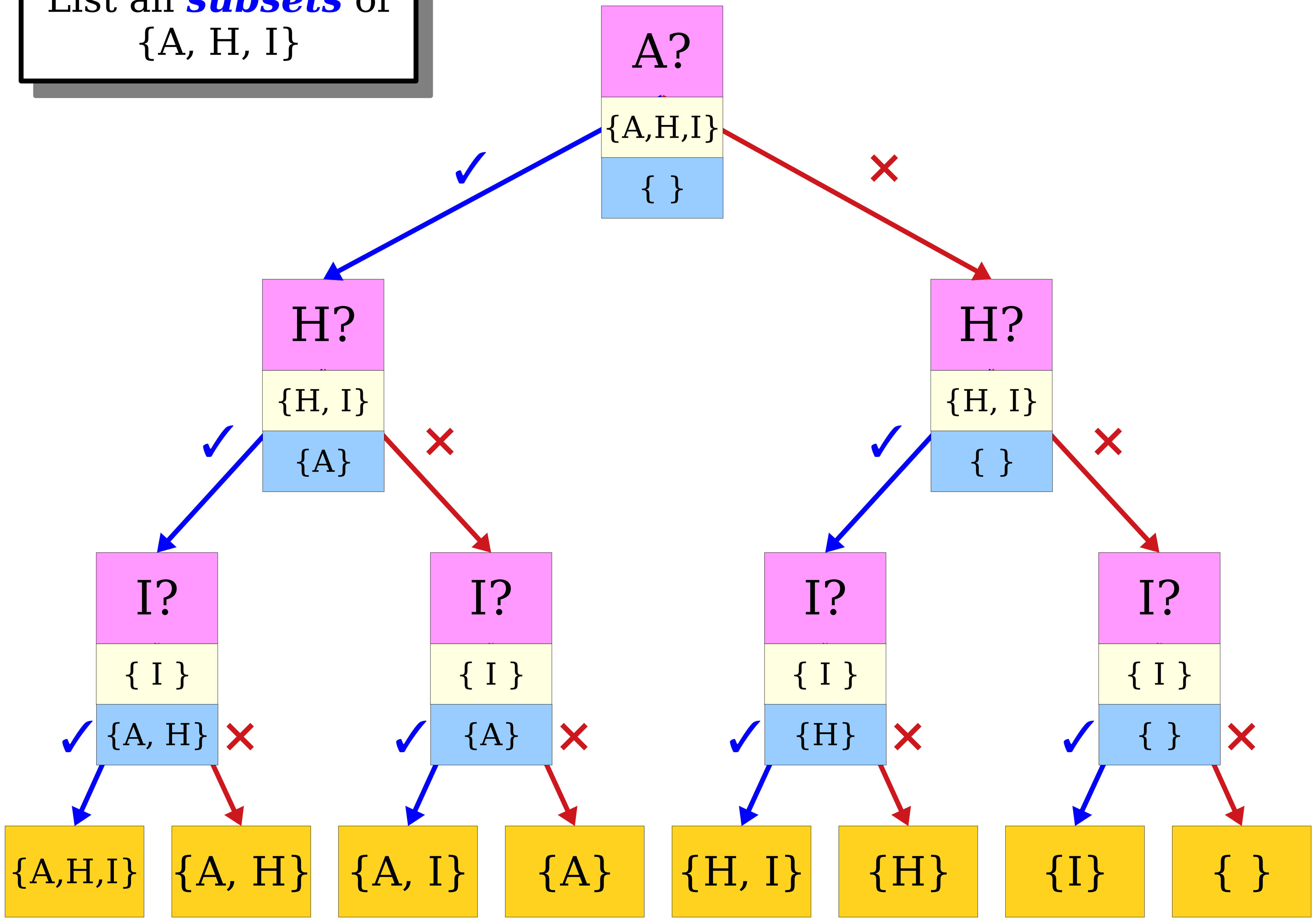


List all ***subsets*** of
{A, H, I}



Recursive case: Pick some element we haven’t decided about yet. Try all possible choices for what to do next.

List all ***subsets*** of {A, H, I}



Base Case:
No decisions remain.

Decisions yet to be made

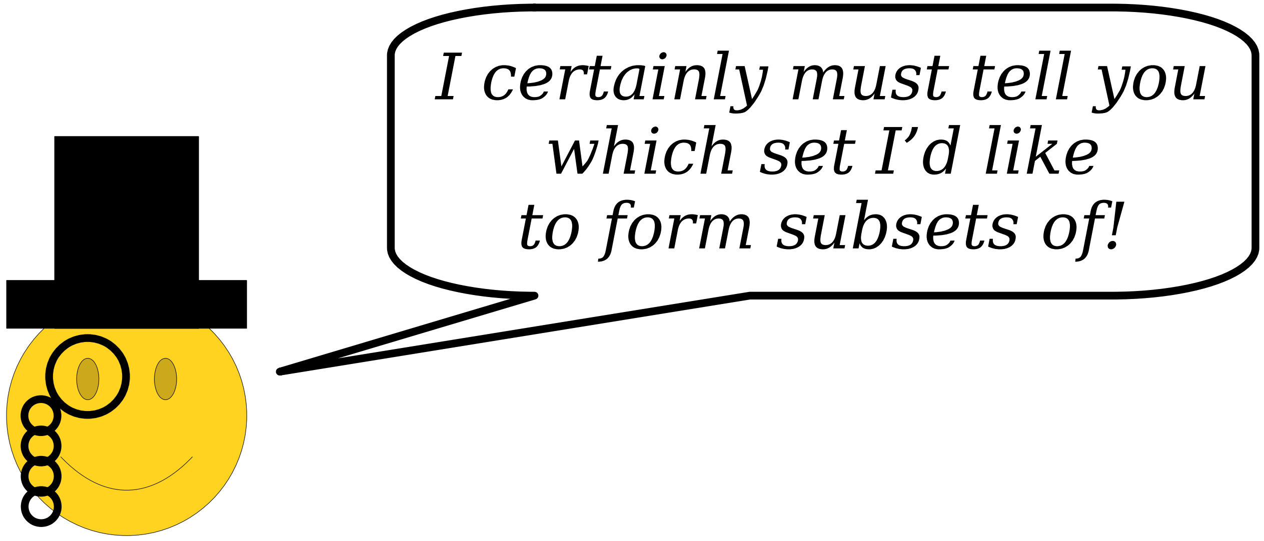
```
void listSubsetsOf(const Set<int>& elems,
                   const Set<int>& soFar) {
  if (elems.isEmpty()) {
    cout << soFar << endl;
  } else {
    int elem = elems.first();
    Set<int> remaining = elems - elem;

    /* Option 1: Include this element. */
    listSubsetsOf(remaining, soFar + elem);

    /* Option 2: Exclude this element. */
    listSubsetsOf(remaining, soFar);
  }
}
```

Decisions already made

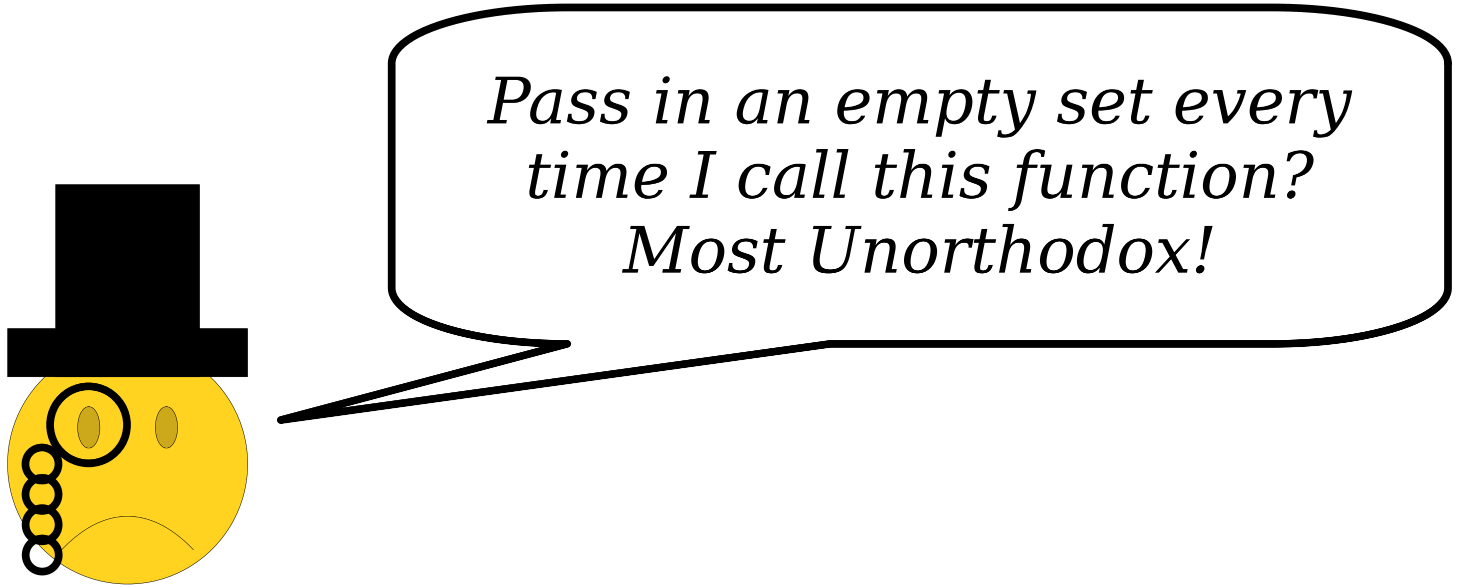
Recursive Case:
Try all options for the next decision.

A Question of Parameters

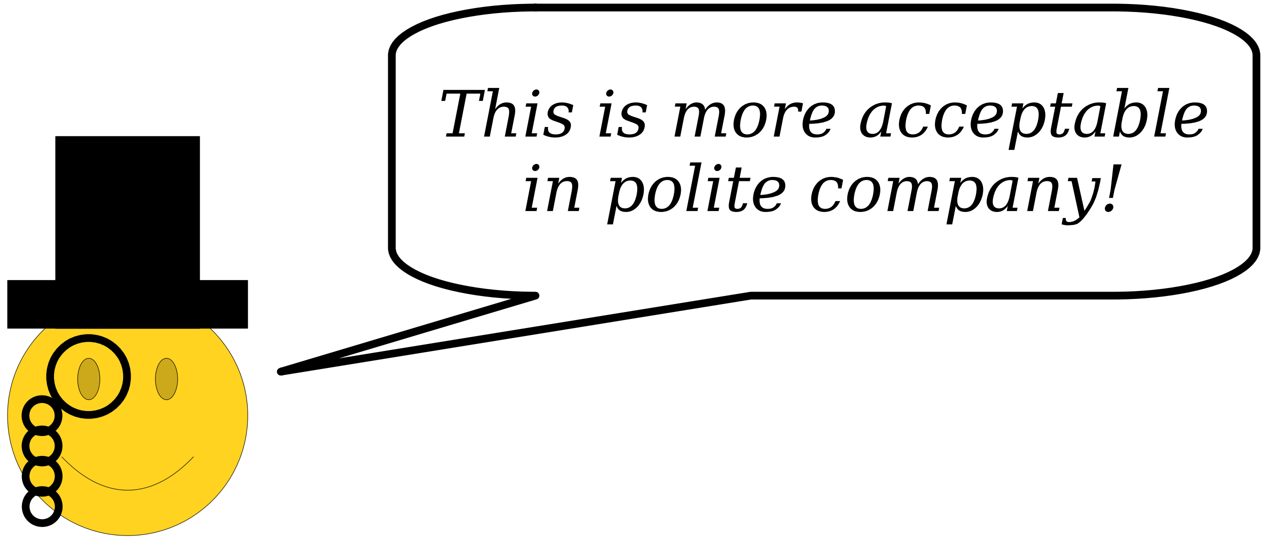
```
listSubsetsOf({1, 2, 3}, {});
listSubsetsOf({1, 2, 3}, {});
listSubsetsOf({1, 2, 3}, {});
```


I certainly must tell you which set I'd like to form subsets of!

```
listSubsetsOf({1, 2, 3}, {});
listSubsetsOf({1, 2, 3}, {});
```

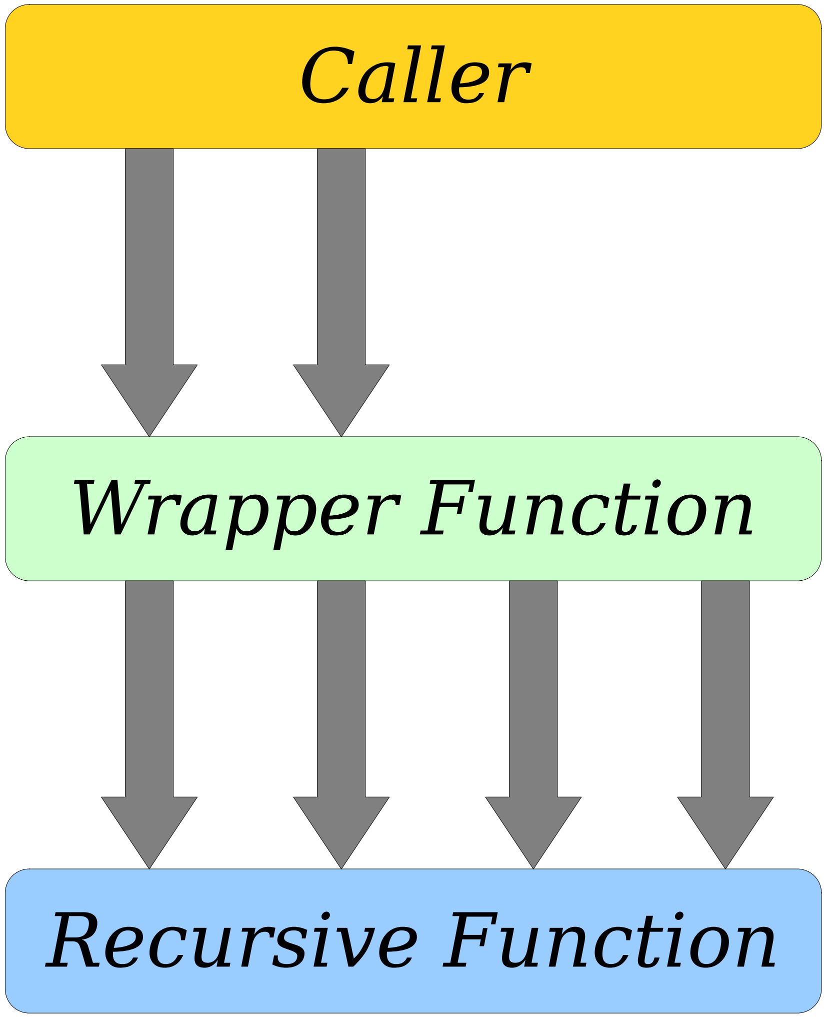

Pass in an empty set every time I call this function? Most Unorthodox!

```
listSubsetsOf({1, 2, 3});
listSubsetsOf({1, 2, 3});
```



Wrapper Functions

- Some recursive functions need extra arguments as part of an implementation detail.
  - In our case, the set of things chosen so far is not something we want to expose.
- A ***wrapper function*** is a function that does some initial prep work, then fires off a recursive call with the right arguments.



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.
- A ***wrapper function*** makes the interface of recursive calls cleaner and harder to misuse.

Your Action Items

- ***Read Chapter 8.***
  - There’s a lot of great information there about recursive problem-solving, and it’s a great resource.
- ***Start Assignment 3***
  - Aim to complete the Sierpinski Triangle and Human Pyramids by our Monday lecture.
  - If you have time, start tinkering around with “What Are YOU Doing?”

Next Time

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