COMP1406

Puzzles and Practical Problem Solving

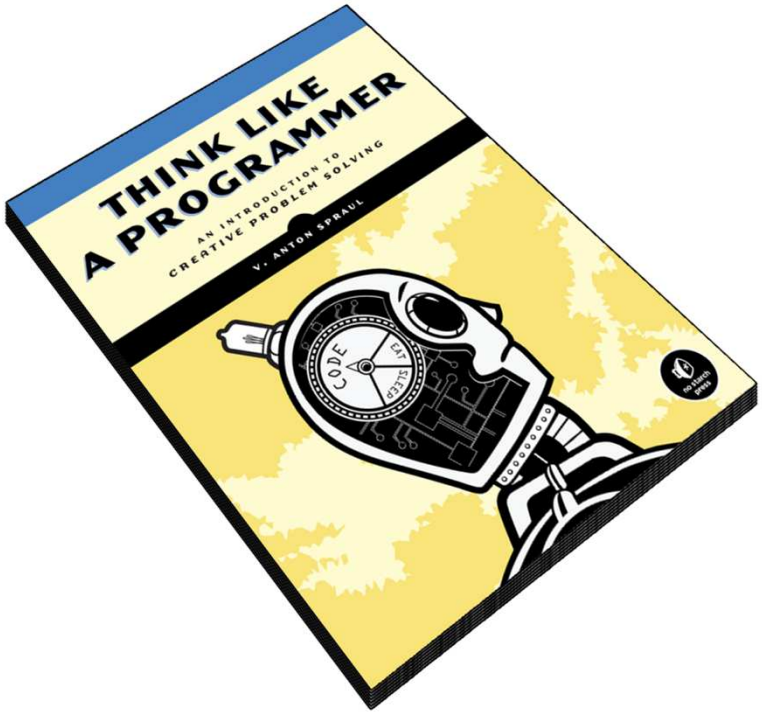
Acknowledgements

- Gail Charmichael, COMP1406 – W15
- Robert Collier, COMP1406 – W16
- V. Anton Spraul, Think Like a Programmer

Reading Assignment

"Chapter 1"

"Chapter 2"



How to Solve It

1. **Understand the Problem**

   Identify the data/unknown/condition

2. **Devise a Plan**

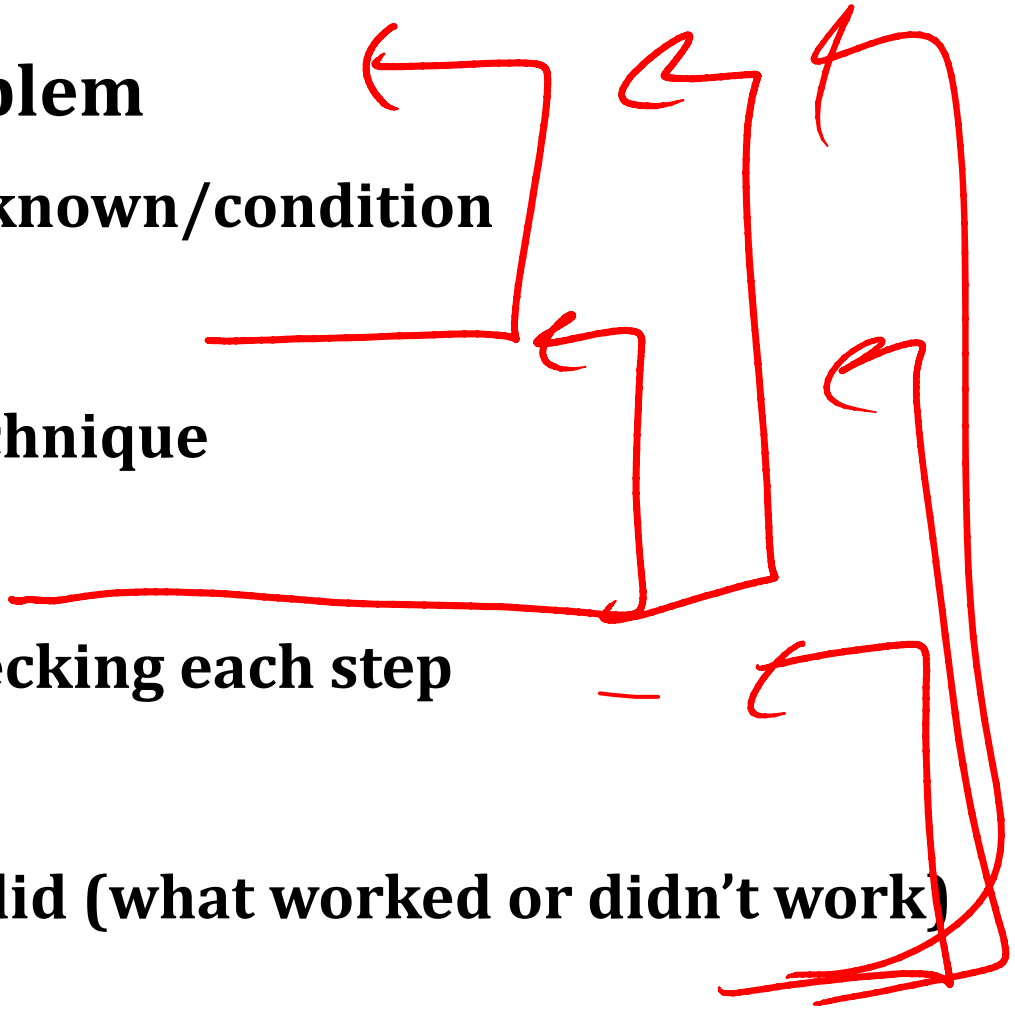
   Select a heuristic/technique

3. **Carry out the Plan**

   Follow your plan, checking each step

4. **Look Back**

   Reflect on what you did (what worked or didn't work)



Problem Solving (TLAP)

- **Always have a plan**
- **Restate the problem**
- **Break the problem down**
- **Start with what you know**
- **Reduce the problem**
- **Look for analogies**
- **Experiment**
- **Don't get frustrated!**

Problem Solving

- **Always have a plan**

Guess solution

Aimless wandering wastes time.

Without a plan, you are hoping for a lucky break.

Plans give you intermediate goals.

Plans can change.

Problem Solving

- **Restate the problem**

Check out the problem from every angle before starting.

We may find the goal is not what we thought.

Use restatement to confirm understanding.

Problem Solving

- **Break the problem down**

Divide the problem into steps or phases.

Difficulty for each phase can be an order of magnitude lower.

Sometimes the sub-problems are hidden.

Problem Solving

- **Start with what you know**

Fully investigate a problem with the skills you have first.

Build confidence and momentum towards your goal.

You may learn more about the problem this way.

- **Reduce the problem**

Reduce scope by adding or removing constraints.

Work on a simpler problem that isn't easily divided.

Pinpoint where remaining difficulties lie.

Problem Solving

- **Look for analogies**

Look for similarities to problems you've already solved.

Recognizing analogies improves speed and skill.

You need to build up a store of prior problems before you can find analogies.

Problem Solving

- **Experiment**

Try things and observe the results (this is not guessing!).

One form: make small test programs.

Other forms are similar to debugging.

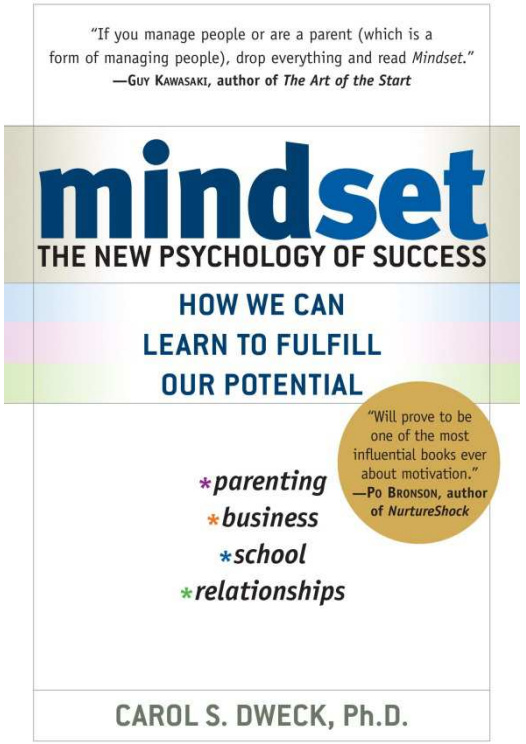
Problem Solving

- **Don't get frustrated!**

Everything will seem to take longer and be harder!

Avoiding frustration is a decision *you* make.

Go back to the plan, work on a different problem, or take a break.

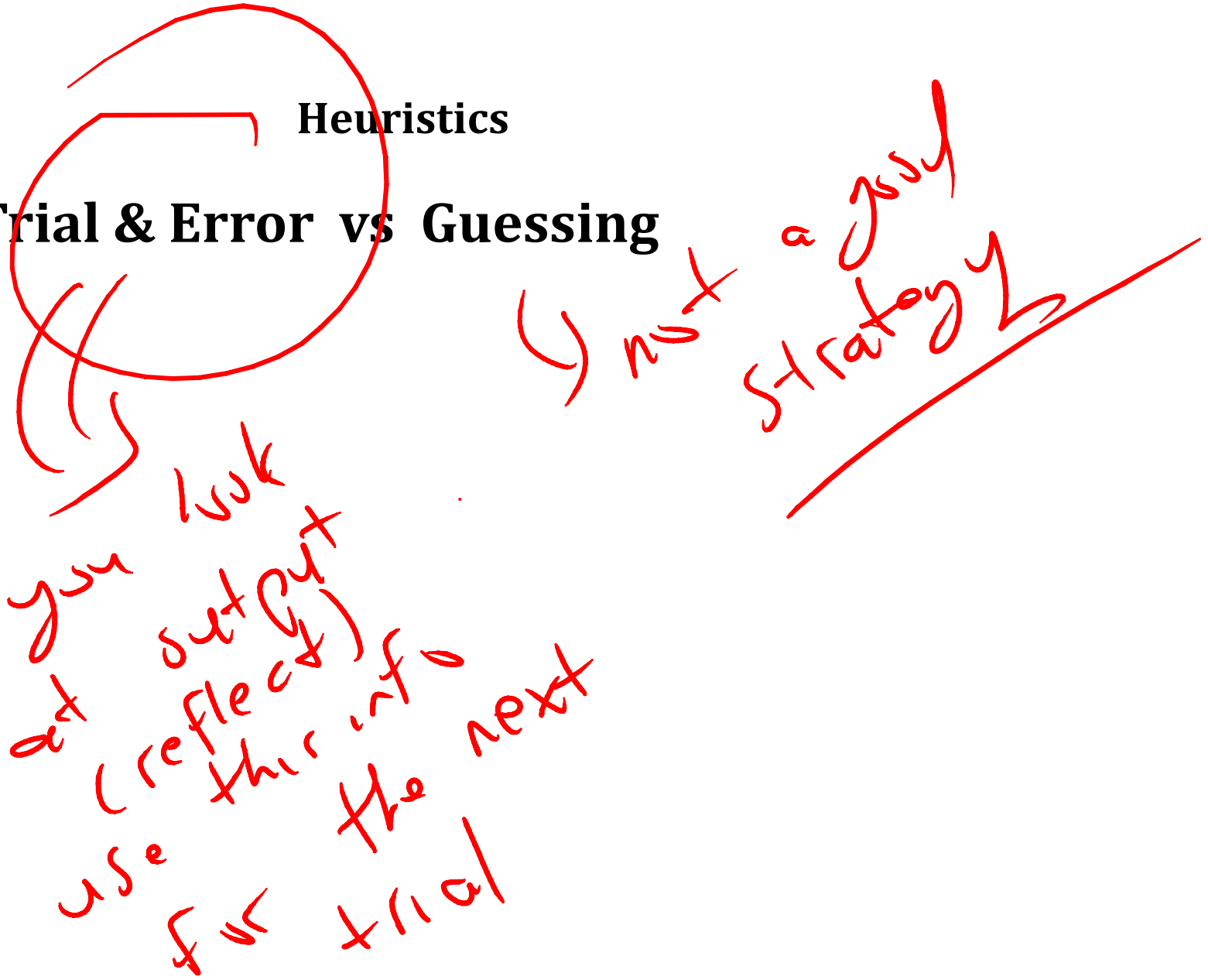


Heuristics (1405 – Collier)

- **Related Problems**
  - Transform your problem into one with a Known Solution
- **Abstraction**
  - Remove Details that are Not Relevant to the problem
- **Divide and Conquer**
  - Disassemble the problem into Easier Subproblems
- **Backward Chaining**
  - Start from the solution and Work Backwards

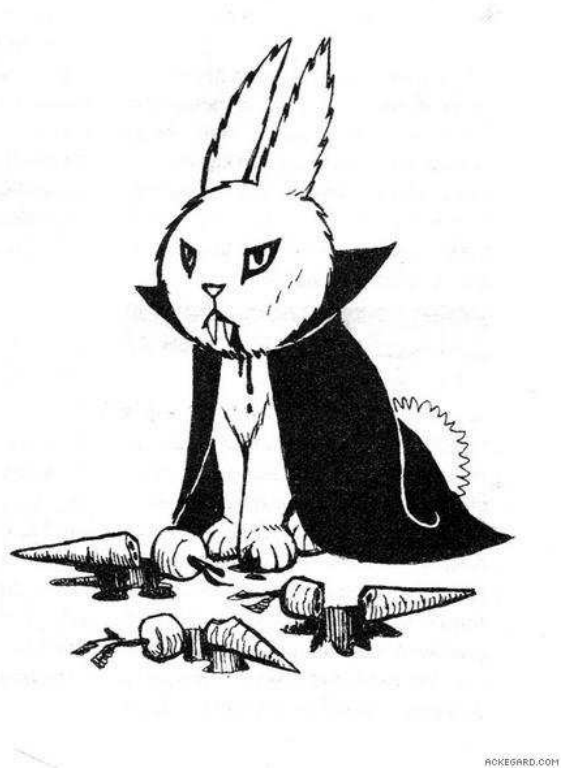
Heuristics

- **Trial & Error vs Guessing**



Vampire Numbers

What is a "Vampire Number"?



***Side Note*: *Mathematicians* are a *Wild Bunch*...**

Vampire Numbers

Vampire Numbers are Non-Prime (i.e., Composite), Positive Integers with an Unusual Property

a Vampire Number that is n Digits in Length has Two Factors (the "Fangs"), each n/2 Digits in Length

the Digits of the Vampire Number Must Be the Digits of Both the Fang Numbers

Vampire Numbers

1260 is the Smallest Vampire Number and it has the Numbers 21 and 60 as its Fangs, because 21 and 60 are both Two-Digit Numbers

Vampire Numbers

How would you Write a Program to Find Each of the Four-Digit Vampire Numbers?

Branching Controls Structures are Typically a Relatively Easy Control Structure to Grasp

Repeating (and Iterating) Control Structures (i.e., While / Do-While / For Loops) are More Difficult

Multiple Levels of Nesting, Each Potentially with Its Own Operations, are Complex to Visualize

Tracing (or Hand-Tracing) is the Process of Manually Simulating the Execution of a Computer Program

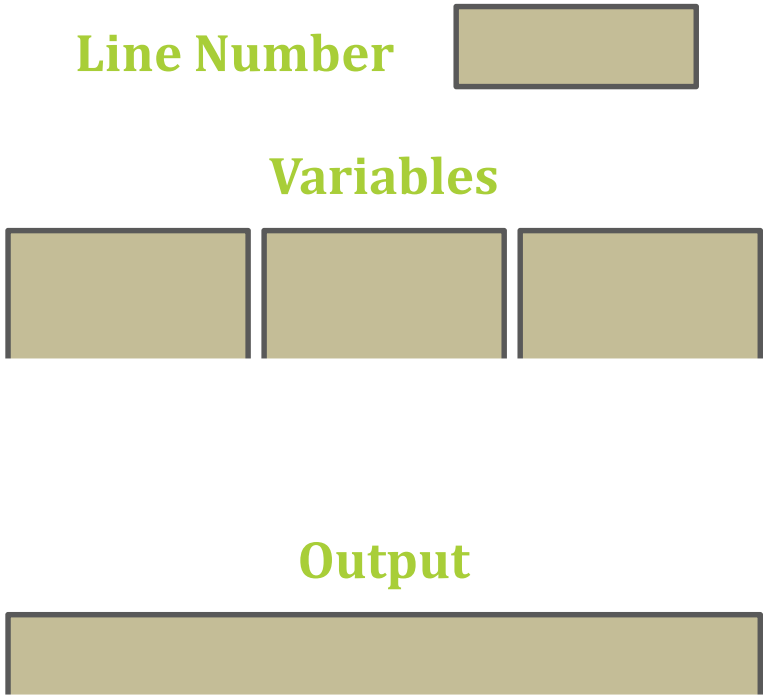
it is a Very Important Debugging Strategy but it Requires Patience and Attention to Detail

Tracing Source Code Manually

Program Being Traced

```
01      #include <iostream>
02      using namespace std;
03      int main()
04      {
05         for (int i = 1; i < 20; i++)
06         {
07            for (int j = 2; j < i; j++)
08            {
09               z = i % j;
10               if (z == 0)
11               {
12                  cout << i << endl;
13               }
14            }
15         }
16         return 0;
17      }
```

Tracing Area



Tracing Source Code Manually

Program Being Traced

```
01      #include <iostream>
02      using namespace std;
03      int main()
04      {
05         for (int i = 1; i < 20; i++)
06         {
07            for (int j = 2; j < i; j++)
08            {
09               z = i % j;
10               if (z == 0)
11               {
12                  cout << i << endl;
13               }
14            }
15         }
16         return 0;
17      }
```

Tracing Area

Line Number: 05

Variables

Output

How do we Approach the Vampire Number Problem?

Vampire Numbers

Adopt a Trial and Error Approach
with a Divide-and-Conquer Heuristic

Vampire Numbers

1260 is the Smallest Vampire Number

(the Fangs for 1260 are 21 and 60)

"Length Constraint?"

1260 is Four Digits Long and 21 and 60 are Two Digits Long

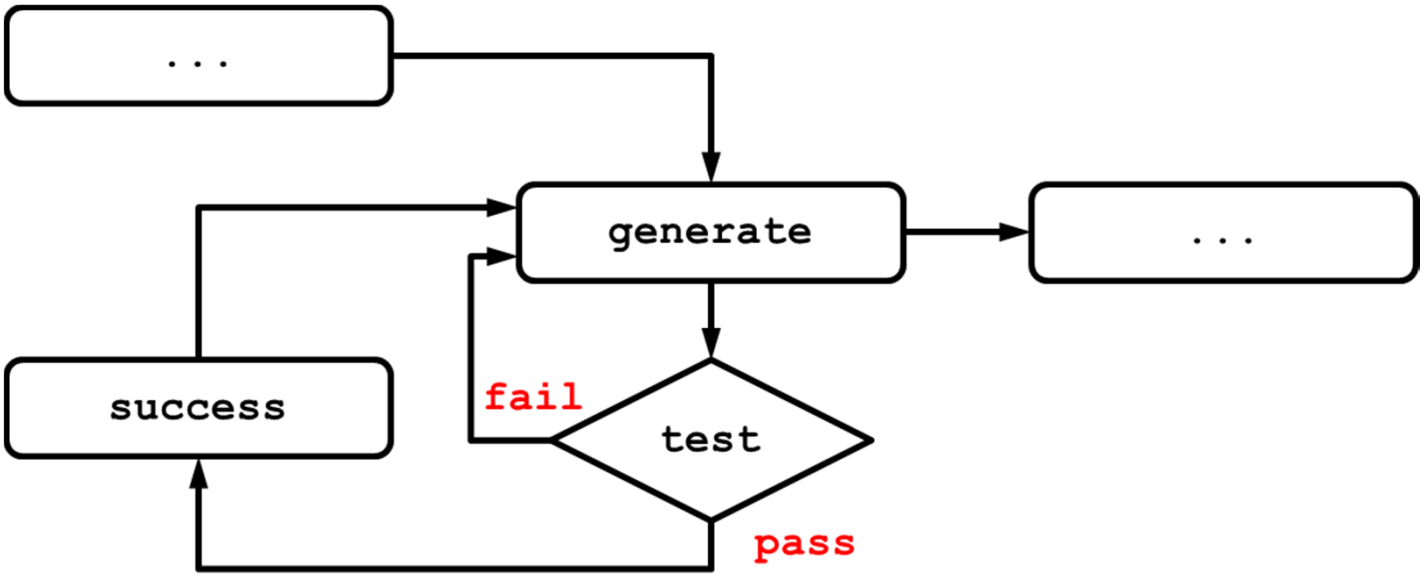
"Factor Constraint?"

$1260 = 21 \times 60$

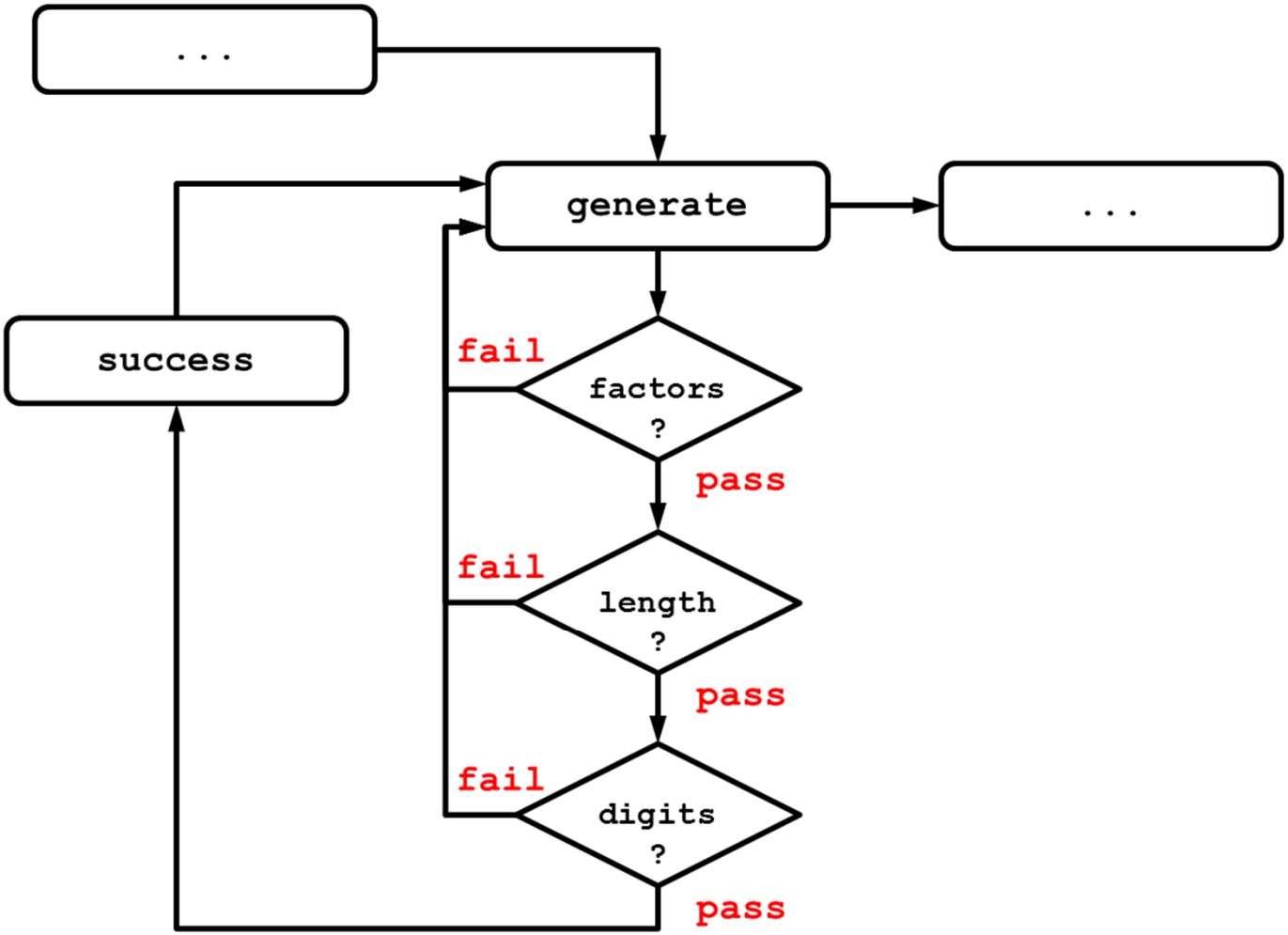
"Digit Constraint?"

the Digits of 1260 - {"1", "2", "6", "0"} are Also the Fang Digits

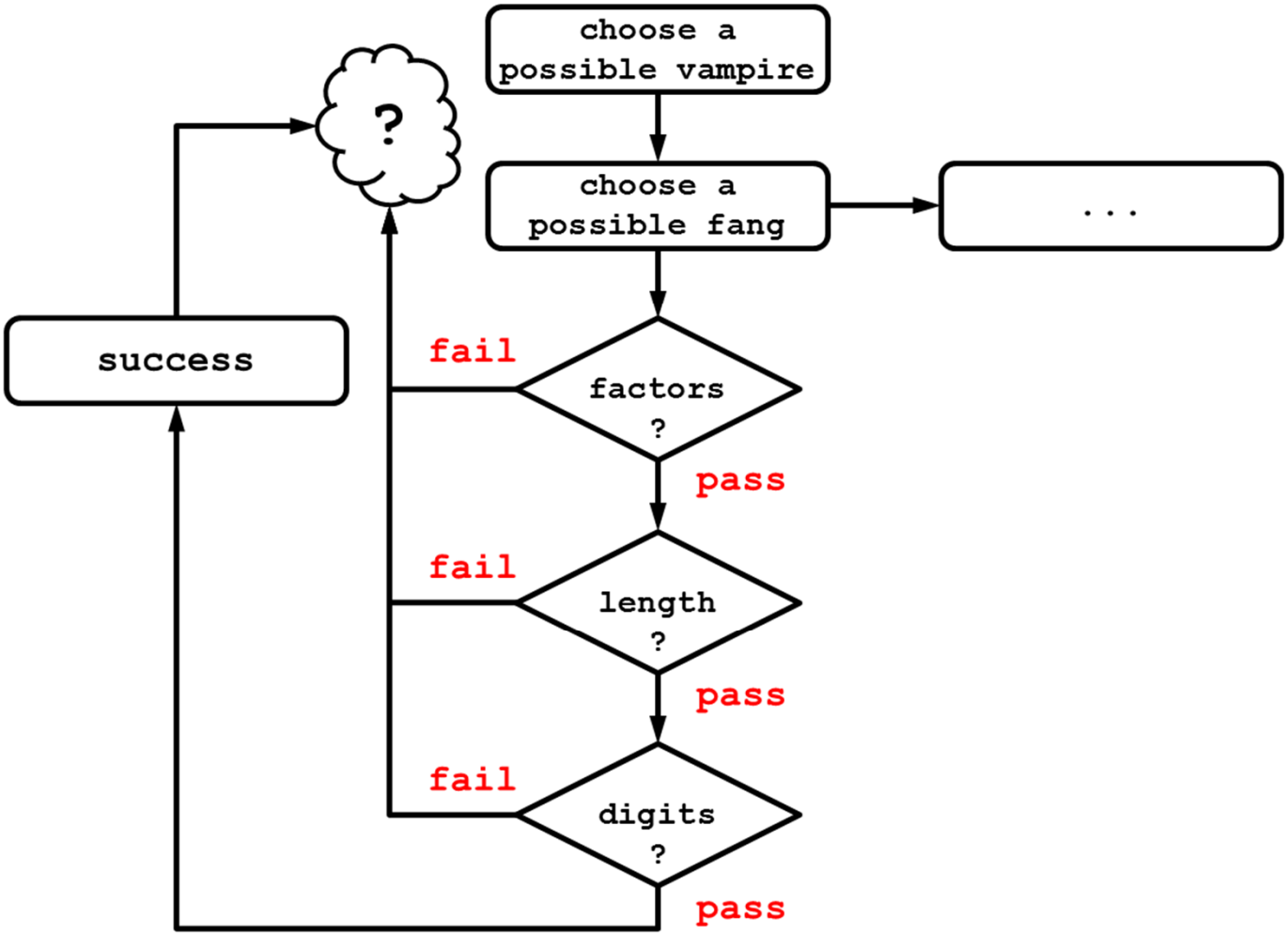
Trial and Error (Brute Force) Design Pattern



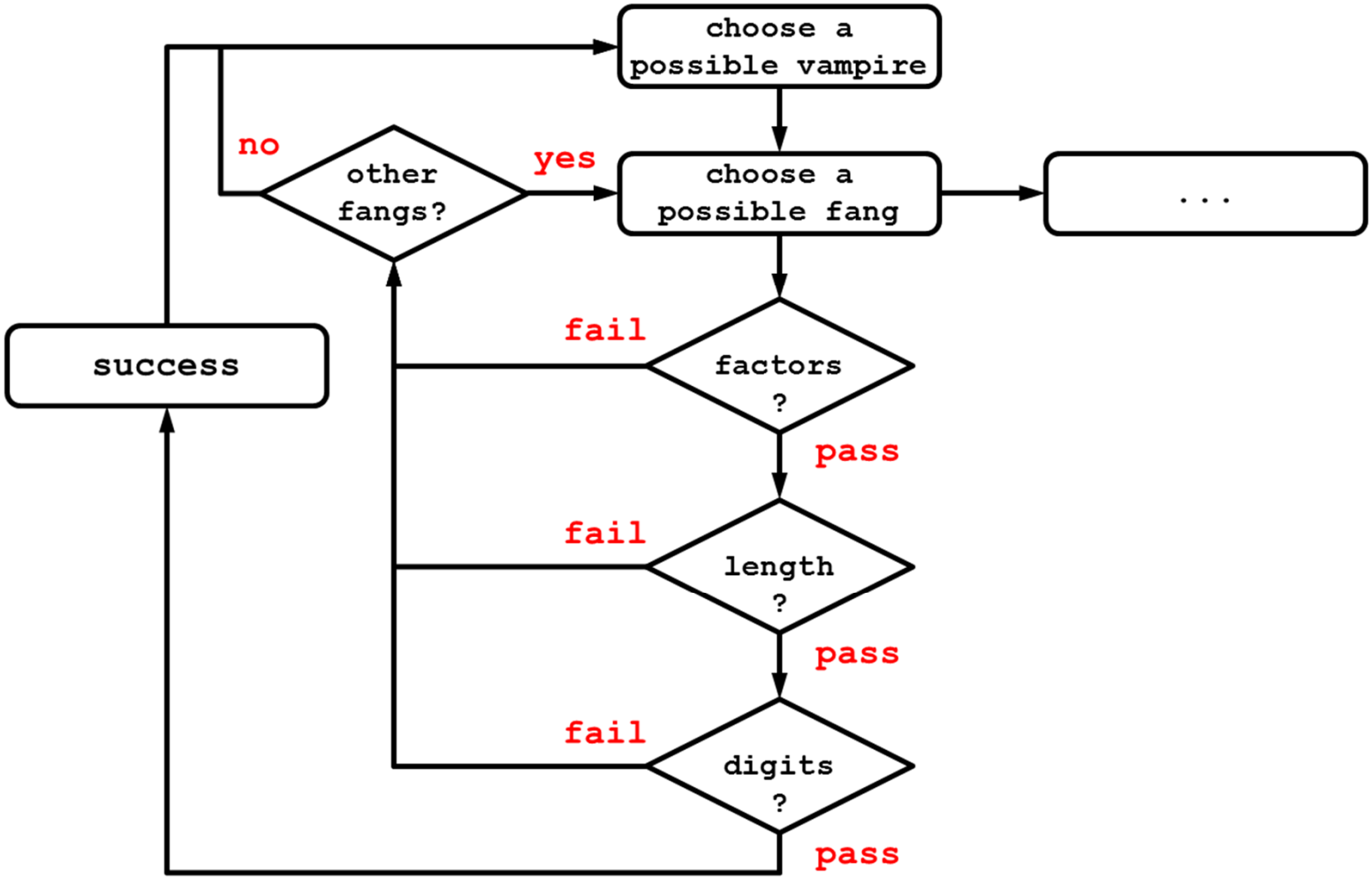
Trial and Error (Brute Force) Design Pattern



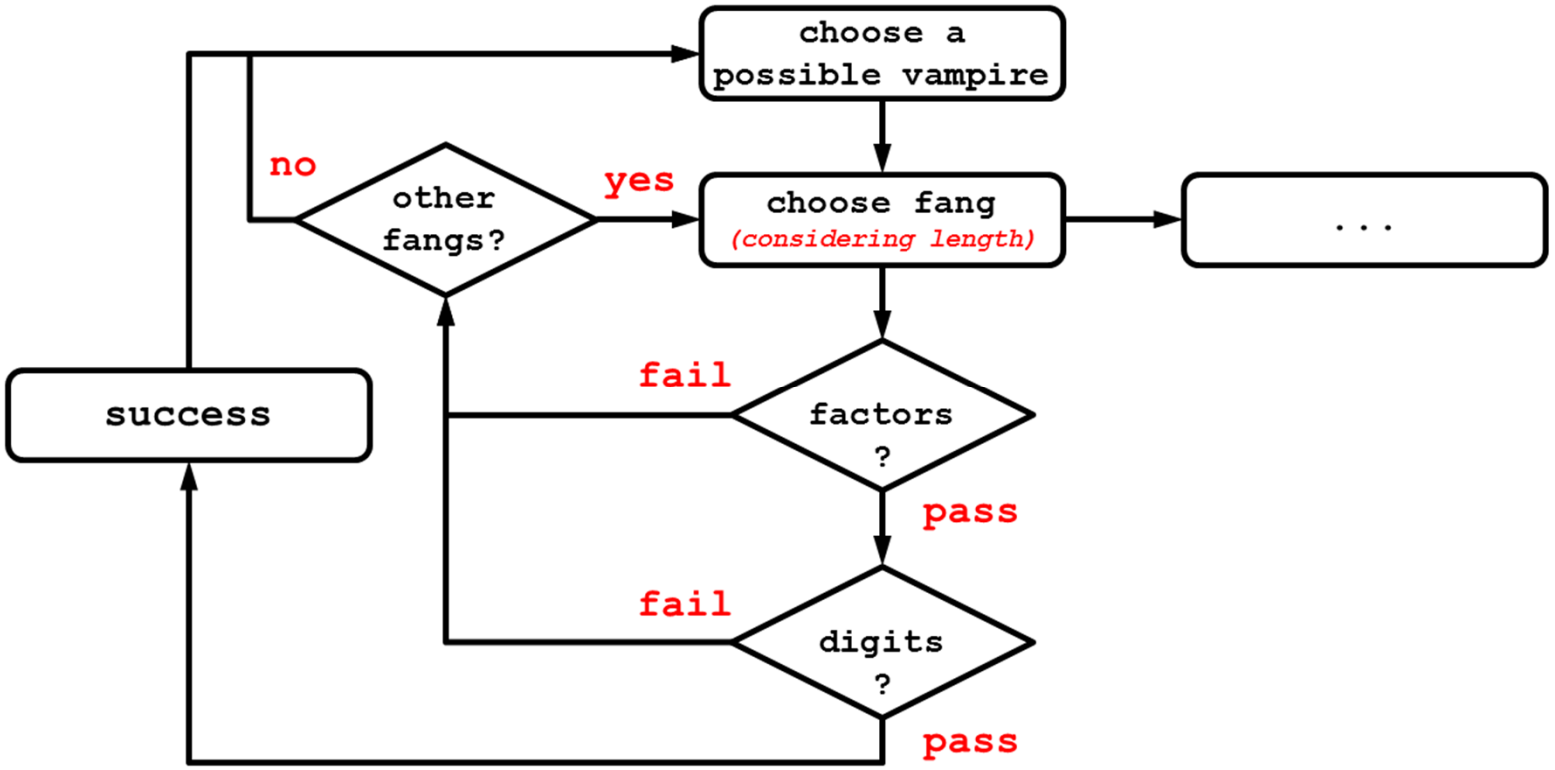
Trial and Error (Brute Force) Design Pattern



Trial and Error (Brute Force) Design Pattern



Trial and Error (Brute Force) Design Pattern



C++ Streams

a "Stream" is a Serial Data Sequence that can be Used as an Interface to (for example) a Terminal or File

the Programmer Does Not Require much Detail about "How" the Writing Process actually Occurs

If the Object to Print has a Stream Representation (which you can define yourself if necessary), it can be Inserted (i.e., Written) to a Stream with ">>" and Extracted (i.e., Read) from a Stream with "<<"

Error Detection

How can the Validity of an Identification Number (or a Credit Card Number, etc.) be Confirmed?

Checksum Validation

a Checksum is the Result of a Specified Calculation that can be Used for Error Detection

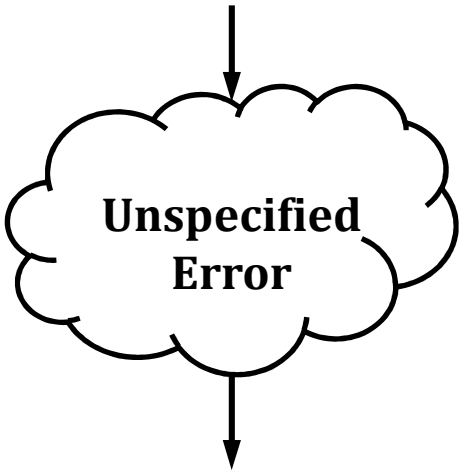
when Transmitting Information, Some Extra Data (a Checksum) is Also Transmitted

the Receiver can then Calculate the Checksum and Compare the Result with the Checksum Transmitted;

If the Computed Checksum Does Not Match the Transmitted Checksum, an Error has Occurred

Simplified Checksum Example

Transmitter:



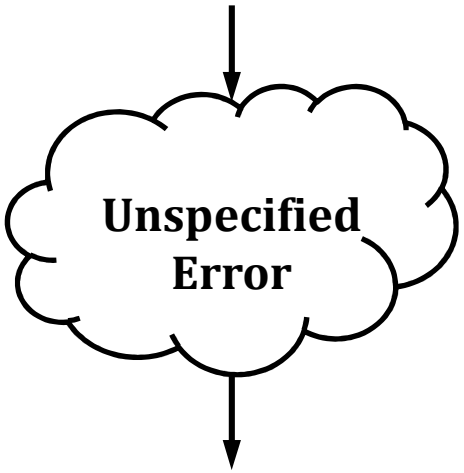
"The Number is 1223"

Receiver: (Receives the Number 1233)

the Receiver has No Way to Know that an Error Occurred

Simplified Checksum Example

Transmitter:



"The Number is 1223"

"The Sum of the Digits Should be 8"

Receiver: (Receives the Number 1233)

"The Sum of the Digits I Received was 9, Not 8."

Transmitter:

"Okay, I will Retransmit the Number."

Luhn Algorithm

the Luhn Algorithm uses a Check Digit (i.e., One Digit)
Instead of a Checksum, but the Principle is the Same

Given a N-Digit Number to Be Transmitted,
Transmit the N-Digits, Followed by 1 Check Digit

How do you Calculate the Luhn Check Digit?

Luhn Algorithm

Transmitting N+1 Digits (n.b., First Digit has Index 1)

"Start from the Rightmost Digit"

"Double the Value of Every Second Digit"

(i.e., double digits at index N, N-2, N-4, etc.)

"Add the Digits of Every Value Doubled"

"Add the Numbers 1→N, Multiply by 9, Modulo 10"

the Result should be the $N+1^{th}$ Digit

Luhn Check Digit Calculation Example

	Identification Number						Check Digit
Compute Check Digit	6	5	7	6	1	3	X

Luhn Check Digit Calculation Example

	Identification Number						Check Digit
Compute Check Digit	6	5	7	6	1	3	X
Double Every 2nd Digit (from Right)		10		12		6	

Luhn Check Digit Calculation Example

	Identification Number						Check Digit
Compute Check Digit	**6**	**5**	**7**	**6**	**1**	**3**	**X**
Double Every 2nd Digit (from Right)		**10**		**12**		**6**	
Add the Digits of those Doubled		**1** (1 + 0)		**3** (1 + 2)		**6**	

Luhn Check Digit Calculation Example

	Identification Number						Check Digit
Compute Check Digit	**6**	**5**	**7**	**6**	**1**	**3**	**X**
Double Every 2nd Digit (from Right)		**10**		**12**		**6**	
Add the Digits of those Doubled		**1** (1 + 0)		**3** (1 + 2)		**6**	
	6	**1**	**7**	**3**	**1**	**6**	

Luhn Check Digit Calculation Example

	Identification Number						Check Digit
Compute Check Digit	**6**	**5**	**7**	**6**	**1**	**3**	**X**
Double Every 2nd Digit (from Right)		**10**		**12**		**6**	
Add the Digits of those Doubled		**1** (1 + 0)		**3** (1 + 2)		**6**	
	6	**1**	**7**	**3**	**1**	**6**	
Sum All Digits	**24** (6 + 1 + 7 + 3 + 1 + 6)						

Luhn Check Digit Calculation Example

	Identification Number						Check Digit
Compute Check Digit	**6**	**5**	**7**	**6**	**1**	**3**	**X**
Double Every 2nd Digit (from Right)		**10**		**12**		**6**	
Add the Digits of those Doubled		**1** (1 + 0)		**3** (1 + 2)		**6**	
	6	**1**	**7**	**3**	**1**	**6**	
Sum All Digits	**24** (6 + 1 + 7 + 3 + 1 + 6)						
Multiply by 9, Modulo by 10	$24 \times 9 \bmod 10 = 6$						**6**

Luhn Check Digit Calculation Example

	Identification Number						Check Digit
Compute Check Digit	6	5	7	6	1	3	X
Double Every 2nd Digit (from Right)		10		12		6	
Add the Digits of those Doubled		1 (1 + 0)		3 (1 + 2)		6	
	6	1	7	3	1	6	
Sum All Digits	24 (6 + 1 + 7 + 3 + 1 + 6)						
Multiply by 9, Modulo by 10	$24 \times 9 \bmod 10 = 6$						6
Transmit	6	5	7	6	1	3	6

Luhn Check Digit Verification

Assuming that a Luhn Check Digit was Transmitted with an Identification Number, How Can the Validity of the Entire Sequence of Digits be Confirmed?