Computational Fairy Tales

Jeremy Kubica

Copyright

Cover design and art by Meagan O’Brien
Interior design by Marjorie Carlson

A Note to Readers

This book focuses on computational thinking. The stories are written to introduce and illustrate computational concepts. As such, they focus on high-level concepts, the motivation behind them, and their application in a non-computer domain. These stories are not meant as a substitute for a solid technical description of computer science. Instead, these stories are meant to be used like illustrations, supplementing the full concept.

The book covers a range of material, from introductory programming through more advanced algorithmic concepts. The stories are organized into sections by concept. Each section covers progressively more advanced concepts.

Finally, each story is meant to (approximately) stand alone. While most of the stories follow Ann's quest to save the kingdom, there are multiple side stories that are disjoint from the main thread. All of these stories take place in the same kingdom.

Major Characters

Ann is the teenage daughter of King Fredrick and heir to the throne. She has been tasked by the prophets with rescuing the kingdom from the coming darkness. Unfortunately, the prophecy was very vague about the specific threat facing the kingdom.

King Fredrick is the king and Ann's father. He is a wise, fair, and occasionally lazy ruler.

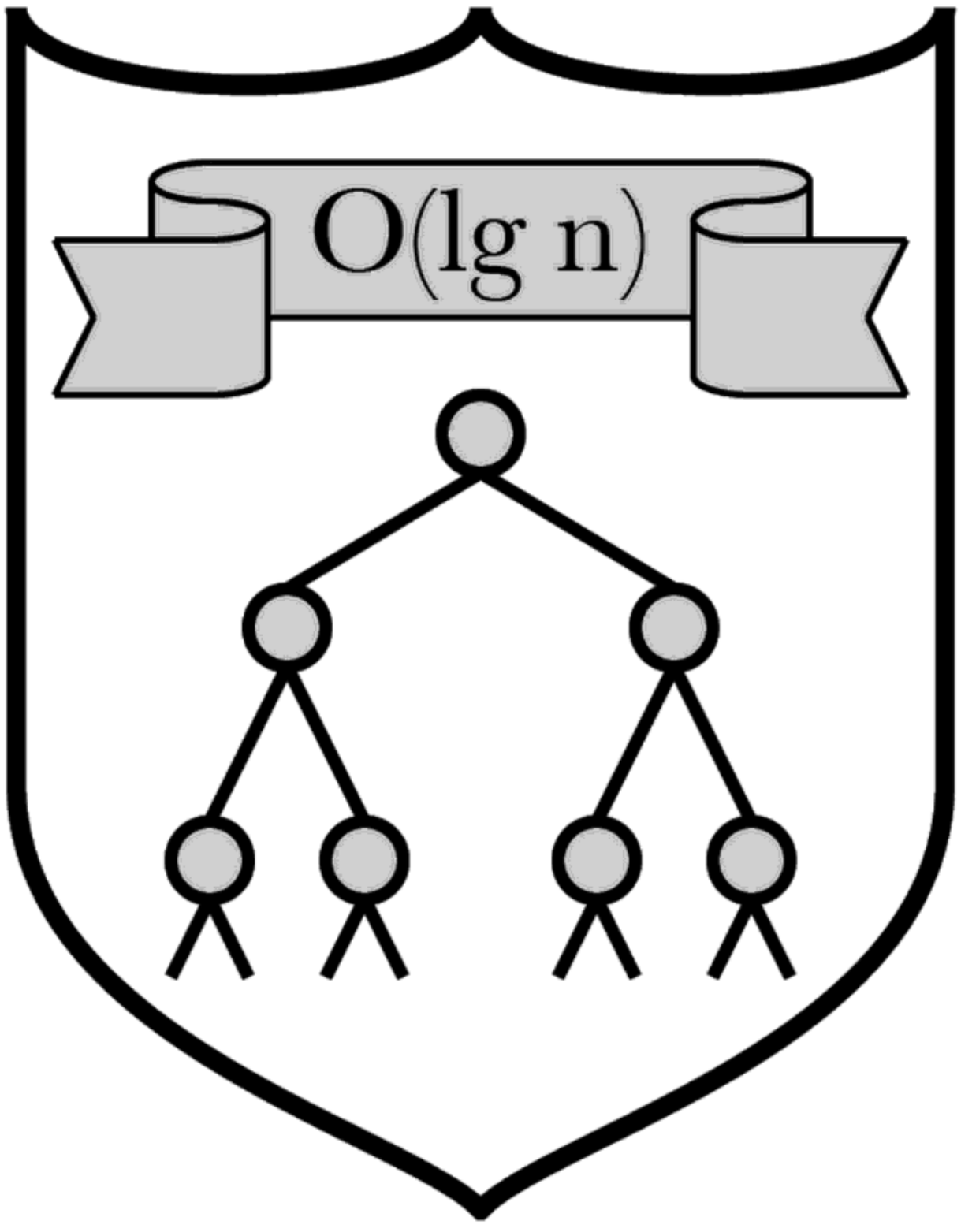
Marcus is a powerful wizard and staunch friend of the king's. He is a firm believer in the importance of the practical aspects of magic, such as testing and commenting.

Clare O'Connell is the kingdom's brightest computational theorist. She spends her days working at the Bureau of Farm Animal Accounting: Large Mammal Division, where she has assembled a talented team of theorists.

Peter is an apprentice at the Library of Alexandria and an eager, though occasionally arrogant, student of computational ideas.

Sir Galwin is the king's trusted head knight. He has years of experience with difficult quests.

THE START OF A QUEST



The Darkness is Coming

“The darkness is coming,” stated the seer.

He was the fifteenth seer to stand in front of King Fredrick this week. Fifteen seers—all with the same prophecy. Darkness. Chaos. Doom.

Granted, each seer brought his or her own twist to the prophecy. One spoke of “ill winds” and added a small shiver for effect. Another shouted loudly about the “end of times” until he was forcibly removed from the room. A third gave the prophecy as a malformed limerick, causing everyone in the room to wince at each attempted rhyme.

In contrast, this seer was much less dramatic; he was calm and to the point. Princess Ann liked that. She had never been a fan of unnecessary theatrics, especially in prophecies.

Of course, Ann didn’t care for the message itself. Why did the kingdom have to be doomed? Things were going really well for her. Now some horrid “darkness” was going to consume the lands.

As her father began to cross-examine the seer, Ann idly wondered how many seers were left today. She tired of the constant proclamations of doom. Why not intersperse a minstrel or two? Anything to lighten the mood would be welcome.

“Ann?” asked King Fredrick in a loud voice. He stared at her. Ann realized that he was waiting for an answer, but she had no idea what the question had been. She had stopped paying attention after the initial prophecy.

“I’m sorry. What was the question?” she asked.

Ann heard a few surprised gasps from the back of the room. Her father sighed and looked at her crossly.

“Will you take on this quest? Will you go forth and save the kingdom?” he asked.

Ann froze. What quest? Why her? And why was Sir Galwin, the land’s most famous knight, glaring at her? He should be beaming at the mere mention of a quest. He loved quests.

“What quest?” Ann asked.

“To travel forth and find a way to stop the darkness,” responded her father. He spoke slowly, as though to ensure Ann was paying attention. It reminded Ann of how he ordered his breakfast in the morning—her father left no room for misunderstanding in some aspects of life.

Ann rather liked the suggestion of a quest. She longed to travel the kingdom, but had never been allowed. With the summer holidays starting next week, this quest sounded like a wonderful opportunity. She might get to see the famed upside-down pyramids of South Patagonia or the great Library of Alexandria. Her father even supported it.

“Sure. I can do that,” she answered quickly.

"It will be a long, lonely, and dangerous journey," her father added. "But you must find a way to save the kingdom and hold back the darkness. The prophecies have said you must travel forth alone to find the answer."

"Wait, what? Alone?" sputtered Ann. They never assigned solitary quests to teenagers. Usually a first quest involved a whole platoon of veteran knights. She didn't know the first thing about questing.

Her father sighed again. "Did you listen to any of the prophecy at all?"

"No," Ann admitted. "I stopped listening when he reached the you-are-all-doomed part. I've heard that a hundred times already. It gets boring."

Everyone in the room stared incredulously at her. She started to feel uncomfortable. She briefly wondered if she could somehow escape.

"I see," her father began. "In short, the seer said that you can save the lands. You need to go find a way to stop the darkness, or we are all doomed. Then he said something about Fortran being the one true language. Honestly, from that point on it was incoherent."

"But … alone?" asked Ann.

Her father gave her a serious look. "Alone."

Ann nodded numbly, and again wished that she could run out of the room.

After that, there might have been more said. There might have been cheers or mumbled messages of luck. Her father might have given words of encouragement. The seer might have provided more information. However, Ann didn't hear anything else. The entire room faded from her mind as her new responsibility dawned on her.

* * *

Computer science is inherently a way of thinking about problems. How can you route pieces of information across a distributed network that spans the globe? How can you render pictures that look more realistic? How can I get this stupid program to stop crashing?

The answers to these questions build on a set of core concepts—approaches to solving fundamental problems in computer science. This book focuses on these core concepts, the problems they address, and how the concepts can be combined to solve even larger and more complex problems.

An Algorithm for Quests

An algorithm is a set of specific steps or instructions for solving a problem. For example, there are algorithms to sort numbers, compute mathematical results, and render images.

* * *

Ann started to panic as she packed for her quest. How was she going to find the answer and save the kingdom? She rarely traveled out of the capital city, and even then she had never gone beyond Millington. Now she had to search all the known lands for a way to save the kingdom. It quickly dawned on her that she had no idea what she was doing.

Her thoughts were interrupted by a sharp knock on the door. Sir Galwin stood rigidly at the entryway, looking mildly uncomfortable.

"Sir Galwin," Ann greeted him cautiously. He had been sulking since Ann had received her quest, and she was afraid of setting him off again.

"I came to wish you luck," Sir Galwin offered. "I'm sure you'll be successful in your quest."

"Thank you," replied Ann.

The knight nodded a stiff acknowledgement and turned to leave.

"Sir Galwin, do you have any advice for me?" asked Ann before he could go.

The knight turned back toward Ann. From the wide smile on his face, Ann knew that she had asked the right question. Sir Galwin loved to share his stories about quests almost as much as he loved questing itself.

"Follow the established algorithm for quests, and you'll be fine," Sir Galwin assured her.

"An algorithm?" asked Ann. She had never heard of an algorithm for quests. Hope flowed through her. She could handle algorithms.

"It's simple," started Sir Galwin. "If you have one or more leads, you follow the best one. Otherwise, if you don't have any leads, you travel to where you can find more information. Break any ties by flipping a coin."

This advice surprised Ann. She didn't know what she had been expecting, but this certainly wasn't it. It took her a few moments to figure out how to voice her confusions.

"This approach seems to involve a lot of guessing," ventured Ann.

"I prefer to think of it as a heuristic," said Sir Galwin.

"A heuristic is basically an educated guess—a rule of thumb, if you prefer," said Ann. "Is there anything more exact? Something without any guessing, perhaps? Something that guarantees that I find a solution quickly?"

Sir Galwin let out a deep throaty laugh. “I said the same thing when my mentor described this approach to me. I resolved to develop a better algorithm for solving all quests.”

Ann waited for him to continue, but Sir Galwin appeared to be watching a pigeon outside her window. As far as Ann could tell, the pigeon was not doing anything particularly interesting. It paced along the window ledge, bobbing its head.

“Did you?” she finally asked.

“What? Oh. The algorithm. No. I never invented anything better. I eventually realized that the established algorithm was pretty good. It turns out that quests always involve some guessing.”

“So my entire plan is to keep following the best lead and collecting new information?” Ann confirmed.

“Yes. I call it the Information Maximization for Issue Resolution algorithm,” said the knight. “I think it sounds much better than what my mentor used to say. He would call it ‘figuring stuff out.’

“Think of it as a search for an answer. At each step you try to either move closer to the answer or learn more about the problem itself. Hopefully, learning about the problem will help you find an answer.”

“How do I figure out the best lead? How do I figure out where to get more information?” asked Ann.

“You have to find a strategy that works for you,” said Sir Galwin. “I rank things according to a gut feeling. I use 0 to indicate ‘feels utterly normal’ and 10 to indicate ‘feels wrong.’ For me, a 10 feels similar to eating three pounds of refried beans. I also use a special data structure to track everything. That system saved my life hundreds of times. One time, I was hunting a particularly nasty bog dragon through some marshlands __”

“Is there anything else I should know?” interrupted Ann. She was desperate for any more information.

Sir Galwin thought for a moment. Finally, he said, “Avoid chasing bog dragons through marshlands.”

For the twentieth time this hour, Ann wondered what she had gotten herself into.

Variables and Magic Gifts

A variable is a place in memory where you can store a single piece of data. Each variable is associated with a name. Programmers can reference, modify, or set the value of a variable using its name. Variables can also have associated types, such as integer, Boolean, or float. These types indicate what kind of information can be stored in the corresponding variable.

* * *

Ann made it less than two miles from the castle before the crushing weight of her task once again descended on her. The fate of the kingdom depended on her finding a way to stop the darkness, yet she didn't know what it was or even how to find out. She felt utterly alone.

Ahead of her, Ann saw a man walking up the road wearing a bright blue wizard's cloak with silver threading. She instantly recognized Marcus; no other wizard dressed so fashionably. He was also one of the kingdom's most powerful wizards and a staunch friend of her father's. If anyone could help her in the quest, he could.

"Sir Wizard!" she called out to him, embarrassed that she had never learned the proper etiquette for addressing a wizard.

Marcus looked up with a smile. "Princess Ann. How are you this lovely morning? Out for a ride, I see."

"Unfortunately, I'm not," responded Ann. "I'm embarking on an important quest. The seers have predicted a coming darkness, and I must stop it."

"Alone?" asked Marcus. His smile vanished.

"Yes. The prophecy said that I 'must journey forth alone to stop the coming darkness.' But … perhaps you could still join me. Technically, we met after I had already journeyed forth alone. In fact, I've been journeying alone for about two miles. And, I could really use your help," Ann pleaded.

Marcus shook his head. "That wouldn't be a good idea. Prophecies are fiddly things, and they don't like it when you try to find technicalities. One time I thought I found a loophole in a prophecy; as a result, it rained Haborian Slugs for three days. It was terribly messy. You must go alone."

Ann's heart sank. Tears started to well up in the corners of her eyes, but she fought them back and nodded bravely to Marcus. She knew he was right.

"Maybe I can still help you, though," continued Marcus. "Let me see what I have with me." As he spoke he rummaged through a small pack. After a moment, he extracted a couple of curious looking objects.

“I have with me some of my latest magical works,” he explained. “They’re based on variable magic.”

“Variable magic?” asked Ann. “What’s that? And what happened to your other work?”

“I’m taking a break from all plant-related magic for a while. A terrible accident with roses,” Marcus said without further explanation. He trailed off, and Ann thought she detected a hint of anger in his expression.

He shook his head as though clearing a horrible image. Then he continued, “Variable magic is a useful, but often overlooked, form of magic. It’s based on the simple idea of storing values. Take this rock for instance. It uses what’s known as a ‘location’ variable.”

“What does it do?” asked Ann, her eyes wide with interest. After algorithmic design, magic was her second favorite conversation topic.

“Stores a value, of course,” answered Marcus. “Weren’t you listening? It stores a location.”

“So, you could use it to …” Ann paused as she thought. She couldn’t think of a single use case.

“You can use it to find your way back to a given place,” Marcus finished. “For example, you could hide treasure and use the rock to store its secret location. Or you could store your current location before heading into a dangerous bog, so you can find your way out. Or you could use it to find your horse in a particularly large parking lot. It has many practical, everyday uses.”

“I see,” acknowledged Ann. She silently wondered what Marcus’s “everyday” life must be like.

“You simply tap the rock with your index finger five times,” explained Marcus. “Then the rock will store your current location in a magic variable. No matter where you are, the rock will continue to point toward the saved location until you set a new one.”

“Like a compass?” asked Ann.

“Exactly!” exclaimed Marcus. “Except you set the location instead of it always pointing north.”

“It can only store a single location?” asked Ann.

“That’s how a variable works; it only stores one piece of information,” answered Marcus. “Think about a small pocket—you can fit one thing in it. You can change what you have in it, but you can never have two things in it at the same time.”

“I guess it would depend on the size of the pocket,” commented Ann.

“Tiny, tiny, tiny,” responded Marcus. “A tiny pocket that can only fit one thing.”

“Oh. Well, thank you.” said Ann. She was still uncertain about the actual usefulness of a compass rock.

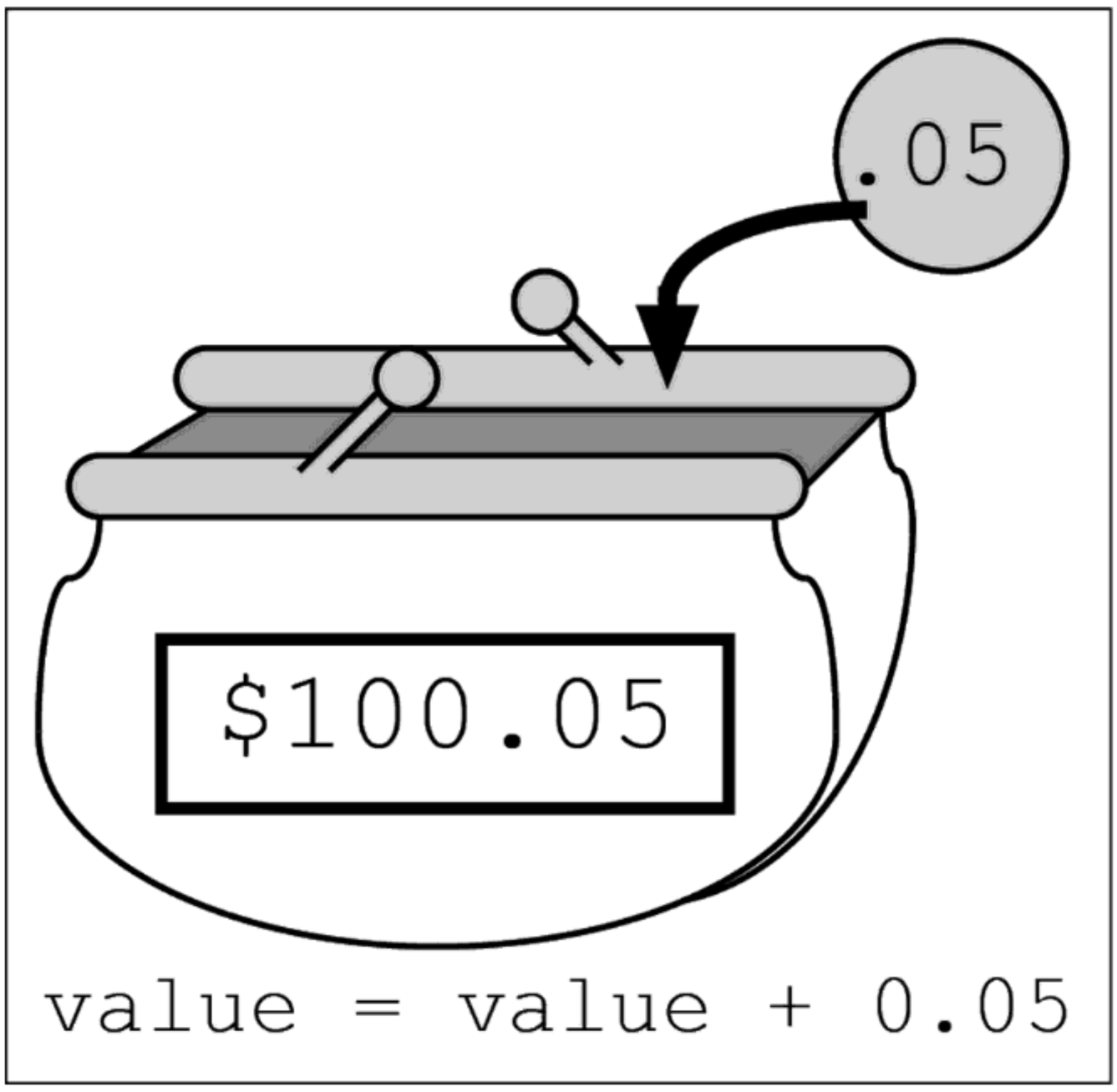
“I also have this for you,” said Marcus, handing her a small coin purse with a counter on the front.

“A purse?” asked Ann.

"A *magic* purse," corrected Marcus. "It works like a calculator. It uses a variable called 'value' to track how much money the purse contains. This purse displays that amount on the front.

"When you put a coin in, that amount is added to the 'value' variable, and when you take a coin out, the appropriate amount is subtracted from 'value.' It always tells you how much money you have in the purse."

Marcus demonstrated the concept by inserting a nickel into the purse. The counter on the front increased by five cents. Ann imagined a magical variable within the purse changing as the result of the addition.



"Why would I need that?" asked Ann. "I can always count the money."

"Ah," said Marcus with a smile. "Counting takes time. What if you're in a hurry?"

"I see. Thank you again for these wonderful gifts," Ann responded with false enthusiasm. She already knew that the gifts wouldn't help her.

"I hope they help you on your quest," replied Marcus. Then he quietly added, "If I had known you were departing on a quest, I would have brought better magical items for you."

He shrugged, closed up his pack, and prepared to leave.

"Sir?" ventured Ann. "May I ask you for one more favor? Do you have any advice to give me on my quest? Any helpful pointers on where to start?"

Marcus paused for a long moment and looked off into the distance. "I don't know what the darkness is, or where you should go. I'm sorry. Instead, I'll leave you with the following advice: don't get eaten by a dragon. I hear it's terribly unpleasant."

With those words, Marcus continued his journey toward the castle.

He hummed to himself as he went.

Once again, Ann felt a pit of despair in her stomach.

The IF-ELSE Life of the King's Turtle

IF-ELSE statements allow programs to branch off and execute one of two different blocks of code. The IF statement starts by evaluating a Boolean (true/false) clause. If this clause evaluates to true, then the block of code conditioned on the IF statement is executed. Otherwise, it is skipped. An ELSE clause can be included to provide an alternate block of code in the cases where the original Boolean clause evaluates to false.

* * *

Fido, King Fredrick's prized pet turtle, lived a charmed life. He spent his days in the garden fountain, swimming and sleeping. He didn't have any magic powers, aside from the ability to amuse himself for an hour by staring at a pebble, but King Fredrick was quite fond of him. Due to his quiet nature and lack of razor-sharp teeth, Fido had always been Ann's favorite pet as well. The castle's servants took good care of him. They made sure that his fountain was always mostly clean—Fido did enjoy the occasional patch of slime.

Fido lived by a series of simple rules. In fact, since his brain was roughly the size of a pebble, they were incredibly simple IF-ELSE–style rules. These rules made up Fido's entire daily routine. For example, he had simple logic to determine when he ate:

```
IF he was hungry then he ate
```

This logic worked well for Fido, because he ate when he was hungry. And, as a natural consequence, he didn't eat when he wasn't hungry. It was quite a good system.

For some aspects of life, the IF statement could have two different actions depending on the condition. For example, when he was swimming:

```
IF the fountain is on then play in the fountain
ELSE swim around the large rock
```

Obviously, Fido enjoyed the fountain more than the rock.

Sometimes the decisions would be complex enough to require a series of chained IF-ELSE statements:

```
IF it is sunny then sit in the grass
ELSE IF it is warm then go swimming
ELSE sleep
```

On sunny days, Fido would happily sit in the grass. When it was warm but not sunny, Fido would swim in the fountain. And on those rare days when it was neither warm nor sunny, Fido would sleep. He hated those days.

The gardener responsible for taking care of Fido often joked that "All that turtle does is eat, sleep, and swim," which wasn't far from the truth. The logic that ruled Fido's life consisted of about fifty different actions contained within chained and nested IF-ELSE statements.

When Ann was a child, a visiting scholar had once spent a week studying Fido. With Ann's eager assistance, he recorded the entire logic for Fido's routine on a single scroll of parchment. If Fido had been intelligent enough to understand what that meant, he might have been offended. Instead, he sat in the grass—it was sunny.

Then, five days after the start of Ann's quest, the unthinkable happened. The gardener, worried that Fido would be bored without Ann's visits, added a second large rock to the garden. This addition threw off Fido's IF-ELSE–based routine completely. It took almost a full week for Fido to determine a new routine. In the end, he added another IF-ELSE:

```
IF he is closer to the right rock then swim around the right
rock
ELSE swim around the left rock
```

Thus order was restored to his life.

Loops and Making Horseshoes

Loops, such as the FOR loop or WHILE loop, are programming constructs for repeating a set of instructions until some termination criterion is met. Two primary things define a loop: 1) What you do inside the loop, and 2) the conditions to stop looping.

* * *

Hundreds of miles north of the capital, in the small outpost of Garroow, the blacksmith Drex was losing his patience. His new apprentice, Simon, wasn't working out. In fact, Drex had never had a worse apprentice in his thirty-five years as Garroow's master blacksmith. Simon could barely lift the hammer, let alone swing it with sufficient force to shape metal. However, worse than that, Simon also lacked the necessary intelligence to carry out even simple tasks. Had it not been for his diminutive size, Drex might have thought that Simon was actually an ogre.

Drex found himself constantly repeating instructions:

"Now, hit the metal again."

"And again."

"And again . …"

Drex's patience was wearing thin. He hated repeating himself.

Finally, Drex decided to try an experiment. "Simon, hit the metal twice," he commanded.

Clank. Clank. Simon complied.

"Now, turn it over and flatten it," Drex commanded.

Simon flipped the deformed-looking horseshoe and hit it once. Then he paused and looked back at Drex, confused.

Drex sighed loudly. Was that really too much for Simon to handle? The boy was hopeless.

"It's a loop!" shouted Drex. He knew that Simon wouldn't understand, but at least shouting made Drex feel better. "A simple, simple loop."

"A loop?" asked Simon.

"Haven't you ever heard of a loop?"

Simon shook his head sadly.

Drex realized that they had hit the core of the problem. How could Simon function as a reasonable blacksmith without understanding how loops worked? Then again, Drex had no idea how Simon could function as a *human* without understanding loops.

"A loop is defined by two things: something to do and a way to know when to stop doing it. You keep doing that one thing over and over until you stop," Drex explained calmly, reciting the favorite description by Garroow's famous scholar Dr. Whileton. Of course, Dr. Whileton

tended to repeat himself, so he would have explained it at least a few times.

Simon stared back blankly.

"Think about a one-mile race," Drex suggested. "You run around the track until you have gone a mile. That's four laps, right? So, running is the thing you do and having run a full mile is how you know when to stop. The track even looks like a loop."

"I run until someone tells me to stop," said Simon.

"Of course you do," muttered Drex.

"In this case," continued Drex, "I want you to keep hammering that horseshoe until it's flat. As soon as it's flat, you can stop. WHILE the horseshoe is not flat, hit it with the hammer."

"Okay," agreed Simon happily. He promptly set about hitting the horseshoe over and over again. By the end, Simon breathed heavily from the effort, but he had succeeded in flattening the horseshoe.

Drex was stunned. How had Simon understood that?

"Good. Now go get the coals hot," Drex commanded.

Simon looked confused again.

Drex sighed. "It's another loop. Pump the bellows 10 times. FOR each number that you count from 1 to 10, give the bellows a pump."

"Okay." Simon again got to work, pumping the bellows exactly ten times. He counted loudly each time:

"*One … two … three … four … five … six … seven … eight … nine … ten.*"

Over the course of a week, Drex determined that Simon would repeat tasks if they were well specified in a loop. Drex would tell Simon exactly what task to repeat and exactly how long to repeat it. Sometimes he told Simon to count up to a certain number. Other times he phrased the command like a WHILE loop, telling Simon to continue doing something until he had met a goal.

Simon responded well to these structured commands. The blacksmith's shop was filled with the noise of Simon cheerfully counting and hammering. "One … *bang* … two … *bang* …"

Eventually, Drex introduced nested loops, issuing instructions such as "WHILE the sword is not thin enough, turn it over and FOR each number from 1 to 5, hit it with the hammer." Simon would happily go about banging the sword into shape while turning it over after every five hits.

Unfortunately, this formalized approach only worked to a point. Disaster finally struck when Drex tried to teach Simon more complicated computational concepts. As Drex and Simon stood outside the burning blacksmith's shop, Drex admitted defeat. Before leaving town for an open blacksmith's position in New Athens, he found Simon a better job —counting laps for runners on the local track team.

The Town of Bool

Boolean logic is based on two values: TRUE and FALSE (or alternatively ON and OFF for physical transistors). Complex logical expressions can be formed by using a few simple operations, such as AND, OR, and NOT. These expressions allow computers to perform logic such as adding binary digits, determining whether an IF statement executes, or controlling when a loop terminates.

* * *

The town of Bool was home to one of the kingdom's most respected logicians, Ellis Conjunctione. Ann had decided to visit the kingdom's scholars in the hope that they would provide insights into her quest. Unfortunately, upon arrival, Ann was informed that Dr. Conjunctione wouldn't be seeing anyone.

"I am sorry, but Dr. Conjunctione is busy at the current time," stated one of Dr. Conjunctione's graduate students. His voice projected a rare combination of formality and boredom. It reminded Ann of a over-practiced lecture, during which the teacher struggles to maintain his own interest in the material.

The graduate student stood in front of the university's doors, physically blocking Ann's path. He crossed his arms. Ann wondered if the student thought that pose would make him look menacing. In reality, he just looked uncomfortable.

"I'm on an important quest," insisted Ann.

The student appeared unswayed. "Dr. Conjunctione is already working on the single most important problem facing the kingdom: a logic problem called 3-SAT. Your quest will have to wait. He gave me explicit instructions not to be interrupted by anyone."

Despite arguing her case for two hours, Ann couldn't convince the student. He refused to compromise at all. Finally, she admitted defeat and resolved to move on to the next name on her list. She decided to stay the night in Bool before continuing.

Ann found her short stay in the town of Bool most annoying. She had always heard that the Booleans were strict believers in binary logic—everything was either true or false. She had naturally assumed that this simply meant that they were opinionated. For example, she wouldn't expect anyone in Bool to state "Jazz is *okay*." Opinions would be definite. However, she hadn't expected this philosophy to apply to absolutely every single aspect of life.

The first surprise came at a local restaurant.

"May I get more water, please?" Ann asked a waiter.

"No," he replied. "I only refill a glass if it is empty AND you're still eating."

"I *am* still eating," Ann assured him.

"But your glass is NOT empty," he responded as he moved off to the next table.

Ann looked down at her glass. It contained at most three drops of water. Ann sighed and finished those drops in preparation for the waiter's return. She decided that in this case she was going to embrace the Boolean philosophy and NOT give him a tip.

Luckily, Ann was well equipped for her stay. She had studied Boolean logic as an elective in kindergarten. It all came down to a few simple rules:

- There are only two options: TRUE and FALSE,
- A **AND** B evaluates to TRUE if and only if both A and B are TRUE,
- A **OR** B evaluates to TRUE if either A or B (or both) is TRUE,
- **NOT** A evaluates to TRUE if and only if A is FALSE.

The logic matched how people used the terms in everyday life. Unfortunately, though, the laws of Boolean logic weren't designed for living everyday life.

Over the course of her 16-hour stay, Ann continued to experience the frustration of dealing with the Booleans' world. She found that when the park "closed at sunset," the patrons would stay until the second the sun dropped below the horizon and then run out of the park. Similarly, getting directions turned out to be extremely aggravating.

"Is the hotel in that direction?" she asked, pointing approximately southeast.

"It is NOT in that direction," proclaimed a Boolean on the street. "It is in *that* direction." The Boolean pointed in almost, but not exactly, the same direction. Ann sighed and walked in approximately the correct direction.

"You are NOT going in the correct direction," the Boolean shouted after her. Ann ignored him.

She also resolved to program Marcus's compass to guide her back to the hotel. That way, if she went out, she could avoid having to ask for directions again.

Even the signage in Bool was overly logical. The crosswalk light actually said "Cross when the WALK light is on AND there are no cars speeding toward you." Did they really need to clarify that? Ann wondered what would happen if someone misprinted the sign to use an OR. Would it be chaos?

Ann only fully understood the Booleans' true adherence to this logical formulation when she reached the hotel. There, on the back of her hotel door, was a fire escape plan like you would find at any hotel—except, in this case, all of the conditions were specified as long Boolean

logic statements. “Use the south stairs IF (they are NOT on fire AND the north stairs are on fire) OR (there is an obstruction in the hall toward the north stairs) OR …”

After reading the sign four times, Ann decided that in the event of a fire she would be too confused to escape. She promptly resolved to leave Bool as soon as she could.

Unhappy Magic Flowers and Binary

Binary is a number system in which each digit can take one of only two values: 0 or 1. Binary allows the computer to encode information in a series of switches that are either on (1) or off (0).

Each binary digit represents a power of two. The first (right-most) digit represents the 1's place, the second digit represents the 2's place, the third represents the 4's place, and so forth. For example, the binary number 10110 = $(1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) = 22$ in decimal.

* * *

The deliveryman paused outside of the wizard Marcus's New Athens townhouse. Marcus smiled. He had recently returned from visiting King Fredrick in the capital, and he was waiting for the backlog of missed deliveries. There should be at least five important potion ingredients and a new hat arriving today.

Yet the deliveryman didn't continue toward the door. He stood transfixed, staring at the flowers. After two minutes, Marcus went outside to see if there was a problem.

"Your flowers have changed since yesterday," observed the deliveryman. "I'm sure the one on the right was red yesterday. Today it's blue."

"Those are the same flowers," responded Marcus. "Some of them are sulking today. Stupid flowers."

"Sulking? Do flowers sulk?" the deliveryman asked.

"Actually, it's more of a protest," clarified Marcus. "They are, of course, magic. They protest whenever it doesn't rain. It's quite aggravating, really. I water them every day, yet they still insist on sulking."

"Huh?" The deliveryman looked back and forth between Marcus and the flowers, trying to make sense of Marcus's statement.

"They protest by changing color to blue. Roses are supposed to be red. You have probably heard all of the poems to that effect, 'Roses are red' and such. But these roses insist on telling me how long they have had to 'suffer' without rain." Marcus gestured angrily at the roses as he spoke.

"They talk to you?" The deliveryman took a step away from Marcus.

"Of course not. They simply change color."

"What does color have to do with when it rained?"

"Well," started Marcus. "they used to all change color together

after three days without rain—a sort of mass protest. Then they started to organize. They want to let me know exactly how unhappy they are. So now they count the days."

"Only two are blue," observed the deliveryman. "It hasn't rained all week."

"Nine days, to be exact," corrected Marcus after looking at the flowers. "They use binary."

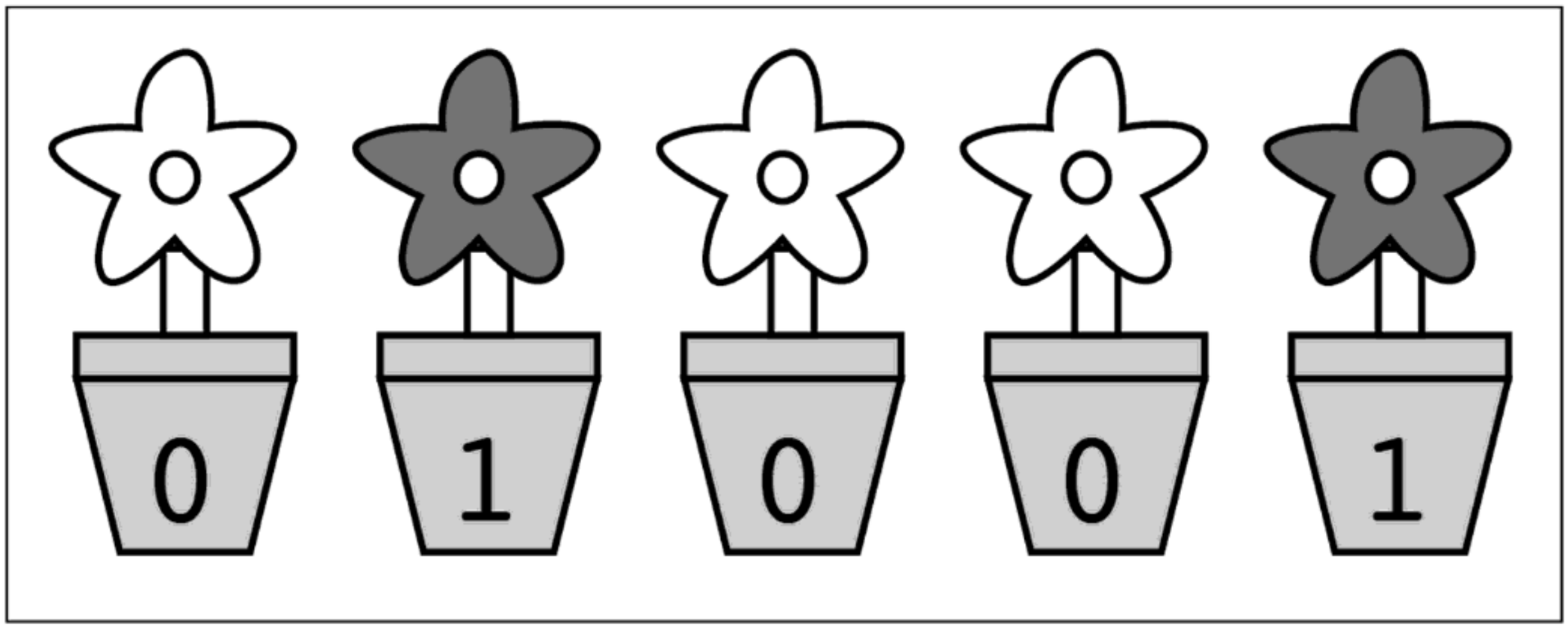
"Huh?"

"Red flowers are zero and blue flowers are one," Marcus added.

That explanation did not help. In fact, it seemed to further confuse the deliveryman. However, on the positive side, he no longer looked as though he wanted to run away.

"Binary?" prompted Marcus. "Each flower represents a different digit, and thus a different power of two. The rightmost flower means one (2^0), the one next to it means two (2^1), the one next to that means four (2^2), and so forth. Add up the numbers represented by the blue flowers and you get the total number of days. Right now only the first ($2^0 = 1$) and fourth ($2^3 = 8$) flowers are blue, so it's been $2^0 + 2^3 = 1 + 8 = 9$ days."

The deliveryman looked. Sure enough, the five flowers across Marcus's porch were: Red Blue Red Red Blue (or 01001).



"Why do they use binary?" asked the deliveryman.

"They tried to spell out the numbers on their petals, and they got too confused. So they had to settle for each flower being either all red or all blue. It turns out that flowers aren't that smart. Binary is a simple enough system for them. If they were smart enough for anything else, do you think they would be complaining to me about the rain? There's nothing I can do about it!" Marcus shouted the last part directly at the flowers.

"But how do they work together?" The more absurd the story got, the more interested the deliveryman became. He leaned in toward the flowers.

"It's really quite simple for them to count in binary," started Marcus. "When it rains, they're all happy and turn red. They effectively reset the counter to 00000. I like those days a lot.

"Then, each morning all of the flowers wake up and decide what

color they'll be for the whole day. If it hasn't rained, they increase the count.

After 1 day: Red Red Red Red Blue (00001 = 1)
After 2 days: Red Red Red Blue Red (00010 = 2)
After 3 days: Red Red Red Blue Blue (00011 = 2 + 1 = 3)
After 4 days: Red Red Blue Red Red (00100 = 4)
After 5 days: Red Red Blue Red Blue (00101 = 4 + 1 = 5)

and so on.

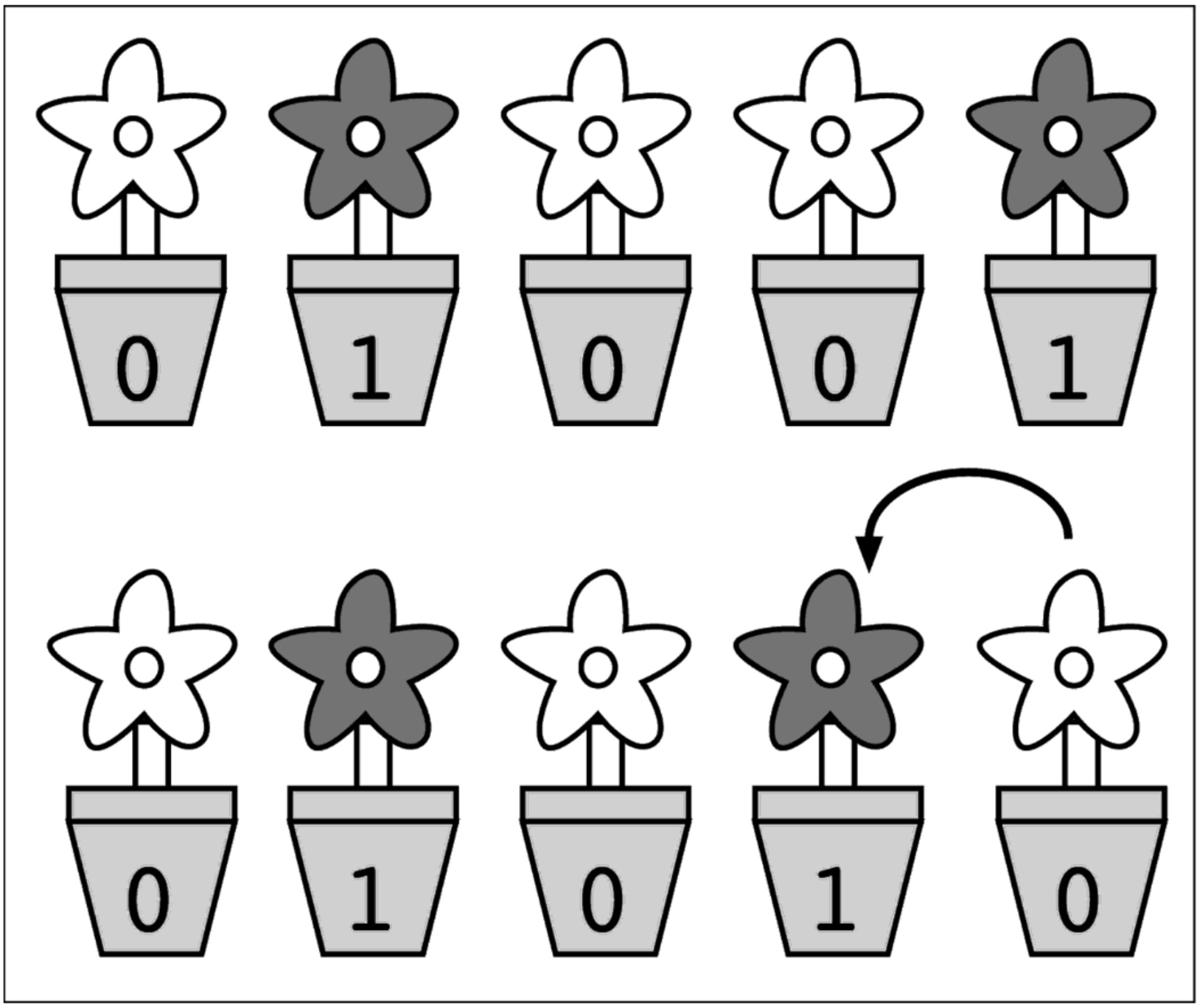
"You see, each flower looks to its right in order to decide what to do. If its right-hand neighbor changes from blue to red (1 to 0), then the flower flips its own color. A blue flower changes to red, and a red flower changes to blue. This keeps happening until one of the flowers doesn't change from blue to red."

"What about the right-most flower?" asked the deliveryman. "How does it know what to do?"

"Ah. *That one is the instigator!* I'm sure he's the one that started it. Every morning there's no rain, he flips. He's the one that starts the process off. Red to blue to red to blue."

The deliveryman thought about it. "Why does a flower only change when its neighbor flips from blue to red?"

"Think about it the way that you would count with numbers 0-9. When you hit 9, you can't go any further with that digit. So you increase the next digit by one and roll the current digit back to 0. It's like going from 19 to 20 or from 29 to 30. Only here there are exactly two options for each digit, 0 and 1, so things roll over more frequently."



“That system always works?” interrupted the deliveryman.

Marcus tore his attention away from the right-most flower. He suddenly wondered how the discussion had gone from a rant about magic flowers to counting in binary. Did the deliveryman not have any other deliveries? For that matter, where was the delivery for Marcus?

“Yes. They’ve already counted out nine days, haven’t they?” answered Marcus flippantly.

Then, seeing the look of interest on the deliveryman’s face, Marcus returned to his teaching tone. “Consider what happens if doesn’t rain tomorrow. The first flower will switch from blue to red, so the second flower will switch from red to blue. The count will go from 01001 = 9 to 01010 = 10.”

The deliveryman looked impressed. Marcus couldn’t understand why. The flowers were really annoying.

The Importance of (Variable) Names

Writing ‘readable’ code is vital to the long-term usability of the code. Code that is clearly written is easy to understand (both for the original programmer and for future users), easier to maintain, and easier to check for mistakes. One important aspect of readable code is the use of clear, meaningful variable names. Using meaningful variable names can greatly improve the understandability of code.

* * *

By the time Princess Ann reached the northernmost outpost within the kingdom, she was losing hope. Her father, King Fredrick, had sent her on a quest to save the kingdom from impending darkness weeks ago. So far, Ann had found nothing.

The outpost of Garroow had been hit particularly hard by the chaos. The frequency of goblin attacks had increased in recent weeks. The commander, Sir Aat, had sent word to Ann’s father that the outpost desperately needed reinforcements. At a loss for better stops on her quest, Princess Ann headed north to Garroow. While there she also hoped to consult with the world’s second-most expert in loops, Dr. Whileton.

Ann found the situation in Garroow worse than she had expected. First off, Dr. Whileton had left for Guelph to start an “important collaboration” with another loop scholar. Nobody could provide details on the project or the timing of his return. Second, and perhaps more importantly, the outpost itself teetered on the verge of collapse.

During her first night at the outpost, a small goblin attack almost overwhelmed it. The fifty-person garrison barely held off three relatively tired goblins. She heard the captain shouting orders at his soldiers:

“Ut, guard the south wall. No, I meant Ot. Ut, stay where you are.

“Drex—no, I mean Dex—swap places with Plex. We need an archer on the wall, not a blacksmith.

“Et, secure that door.”

Eventually, the soldiers repelled the attack and put out the fires. However, the lingering feeling of chaos and confusion continued to bother Ann. It worried her that the garrison’s response had been so disorganized. It was like watching her father’s turtle Fido try to chase its own tail. The problem wasn’t the number of soldiers in Garroow, but rather how they were being commanded.

Ann resolved to fix the situation before leaving the garrison. She spent the entire night pondering the different algorithmic strategies, certain that one of them would help the garrison run more efficiently. As

she had been taught from an early age, almost every problem has an algorithmic solution. Ultimately, the true problem dawned on her at 3 a.m., and she fell asleep confident that she knew how to fix the situation.

"Sir Aat," she addressed the commander at breakfast the next morning. "We need to discuss the attack last night."

"Yes," agreed the commander. "The goblin threat is real. Now you see why we need the reinforcements?"

"No," responded Ann.

The commander looked shocked. The rest of the dining hall fell silent. Everyone waited to see what Ann would say next.

"You need better names," Ann continued.

The commander laughed deeply. "You don't understand. We've already improved our names. When a soldier joins the outpost, I assign him a new name. Every name is short so that commanders can call out orders quickly in battle."

"No," disagreed Ann. "It's inefficient."

"No offense, Princess Ann, but what do you know about commanding in battle?" he asked.

"Only what I observed last night. But from that limited introduction, I can assure you that the names are hurting your efforts."

"I think you're mistaken," declared the commander. "They allow us to issue commands at incredible speeds."

"Yes, they do," agreed Ann. "But they're prone to mistakes. Last night, you corrected yourself 89 different times. The names are too similar and thus too easy to confuse. Plex and Dex. Ut, Ot, Et, and Aat. The short names don't help!"

"Ha! What would you suggest?" scoffed the commander.

"Use descriptive names. For example, Plex should be called 'South Tower Archer' or at least 'Archer Plex.' That more accurately reflects his role."

"That's crazy!" bellowed the commander as he slammed his mug of coffee on the table. His anger at being lectured overrode his manners toward the future ruler of the kingdom. "Do you know how long it takes to say 'South Tower Archer' in the heat of battle? We would waste valuable time."

"Do you know how long it takes to say 'Dex, swap places with Plex, we need an archer on the wall, not a blacksmith'? Any measure of efficiency needs to take into account the time spent on corrections," Ann countered.

"Well—you see—our old blacksmith Drex recently relocated, so —" started the commander.

"What about you?" Ann interrupted. "Why not have them call you 'Commander' or 'Captain'?"

"Our names already reflect rank," replied the commander. "The names proceed down the ranks in alphabetical order. It allows any soldier to instantly know who outranks them! It makes life simple!"

“No it doesn’t. In order for the soldiers to refer to each other, they have to learn new, made-up names. Why not have them learn the ranks instead? Either way they have to learn something new. Only, in this case, the ranks mean something.”

“We have a good system!” argued the commander.

Ann sighed. “It’s like programming a complex algorithm,” she explained. “Using short variable names can make it feel more efficient to program, because you can type out the code faster. But, in the long run, it can do more harm than good. It becomes easy to make mistakes and difficult to sort out what’s happening. Oftentimes, slightly longer names can make a significant difference.”

The commander opened his mouth to argue but couldn’t think of a rebuttal. Instead, he sat at his table, mouth open, with a confused look on his face. After a while, he spoke.

“Princess Ann, I think you might have a point.” Secretly, the commander also felt a small pang of relief. He had never been fond of his own assigned name. He often found himself daydreaming of his soldiers saluting and shouting “Yes, Commander!” in unison.

That afternoon, the commander changed every soldier’s name to be longer but more meaningful. Over the next few days, the troops stumbled through drills, getting used to their longer names. But soon Ann began to see efficiency improve.

A week later, on Ann’s final night in Garroow, goblins attacked again. This time the invading force consisted of ten highly trained goblin special-forces troops. The Garroow soldiers turned away the attack with ease.

As Ann left the garrison, she took a small bit of pride in the dramatic improvements in the forces there. After indulging in the brief moment of happiness, she turned her horse south and continued on her quest to save the kingdom.

Pseudocode for the Quest Algorithm

Pseudocode is an informal way of writing algorithms in order to make them easily understandable. While it represents actual computer code, it does not adhere to the syntax of any programming language. Instead, it is often written as a mixture of programming syntax and natural language.

* * *

As she rode, Ann pulled a small scrap of paper out of her bag:

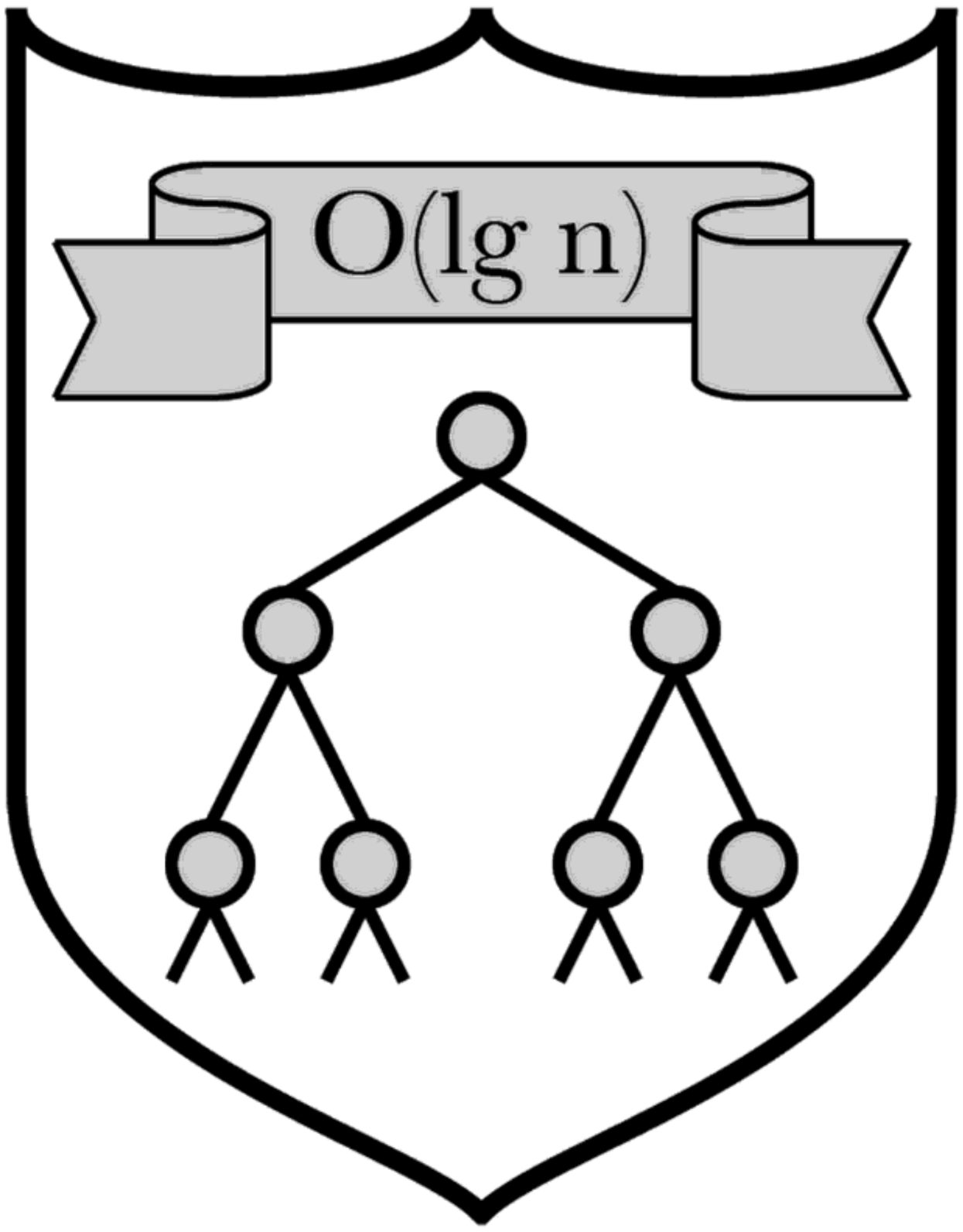
```
WHILE I have NOT stopped darkness:
    best_lead = "Find new information"
    FOR each lead in my list of leads:
        IF lead is better than best_lead
            best_lead = lead
    Follow the best lead
    Check if I have stopped the darkness
```

She already knew it by heart. It contained the algorithm that Sir Galwin had taught her. Despite its simplicity, looking at the algorithm always gave her hope.

So, without any leads, Ann picked a new city in which to gather information. She would travel to Guelph and consult the foremost expert in loops, Dr. Iterator. With any luck, she would also find an opportunity to speak with Dr. Whileton.

She carefully folded the paper and returned it to her bag. At least she had a plan.

DATA STRUCTURES



Arrays, Linked Lists, and Zed's Coffee

Arrays and linked lists are both simple data structures that store multiple values in memory. These data structures differ in how they store and allow access to the data.

Arrays are like a set of bins with a fixed number of slots. Their structure makes it easy to read from or write to an arbitrary element in the array.

In contrast, linked lists are easily expandable chains of data. However, you must scan to the correct location in the chain to read or modify a piece of data in that node.

* * *

One year after Zed opened his coffee shop in the capital, business was great. Zed had a devoted set of regulars who bought coffee every morning on their way to the castle. They were mostly bureaucrats, specializing in such jobs as counting the kingdom's cattle or copying maps. King Fredrick's steward had become a particularly devoted patron, drinking an alarming amount of coffee each day. Even Princess Ann used to frequent the shop before she departed on her quest.

Then, one day, a competitor opened shop across the street. Zed started losing business to MegaCup's low prices and flashy signs. Zed knew he had to expand.

Looking over the books, Zed noticed that he sold a lot of coffee in the morning but almost none at night. None of his customers wanted to be jittery as they headed home and went to sleep. Zed needed a new product—something he could sell at night.

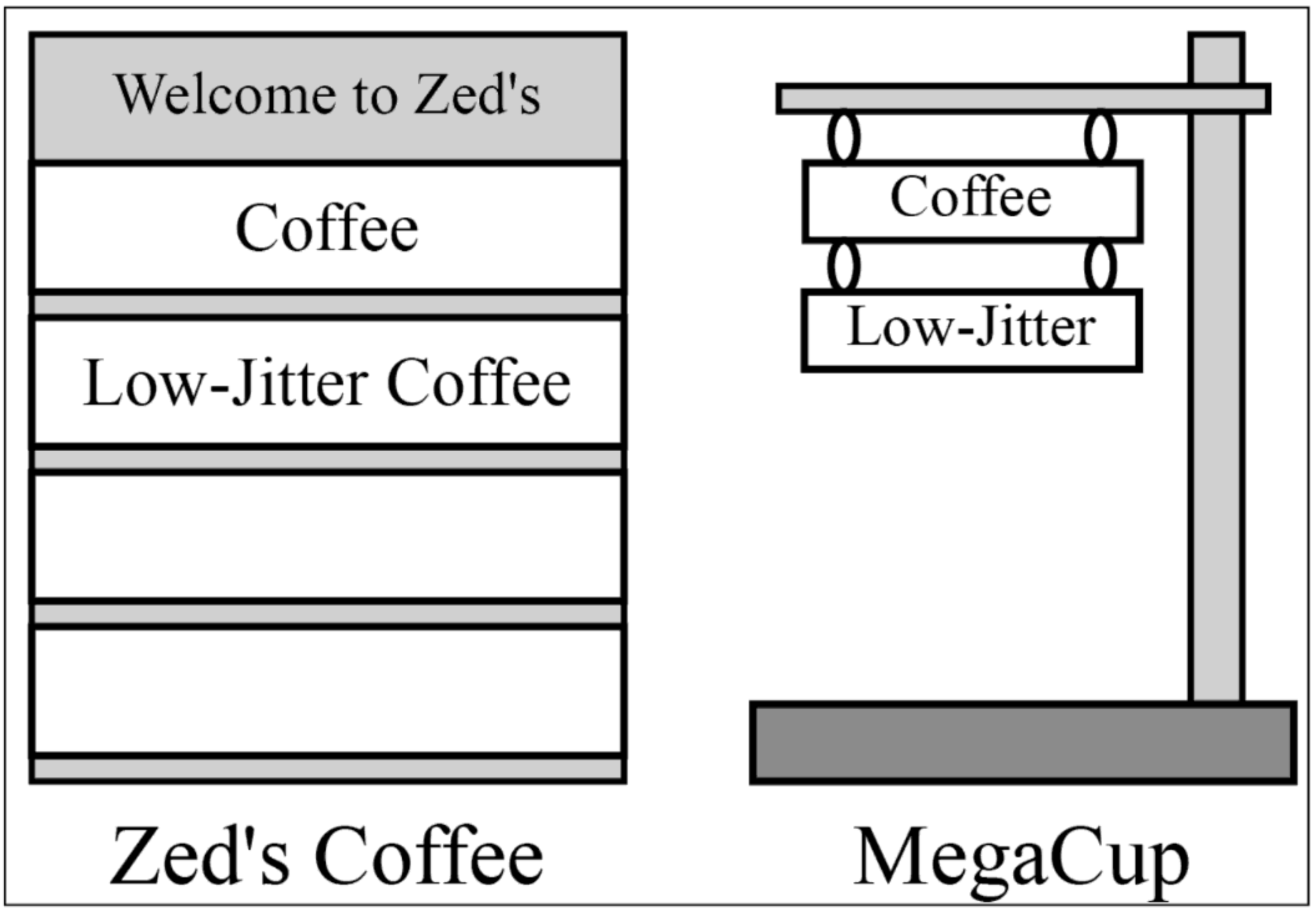
His supplier told him about a new type of coffee coming from the southern region of the kingdom, "Low-Jitter Coffee." Immediately, Zed knew this coffee would solve his evening sales slump. He ordered eight cases.

Zed needed a way to market his new coffee. The sign outside his store read "Coffee" and didn't have room for anything else. After a week of intense thought, Zed ordered a new ArrayDesignBoard menu board for outside his shop. The board had four slots into which you could slide the menu items you wanted to display. He slid in "Coffee" and "Low-Jitter Coffee" tags.

The new coffee became a huge success. Zed's business doubled in a week. He added four baristas to the evening shift. He even attracted a few new morning customers, such as the king's tailor, who had long ago learned not to mix sewing and strong coffee.

However, Zed's competition soon caught on. A week later, Zed

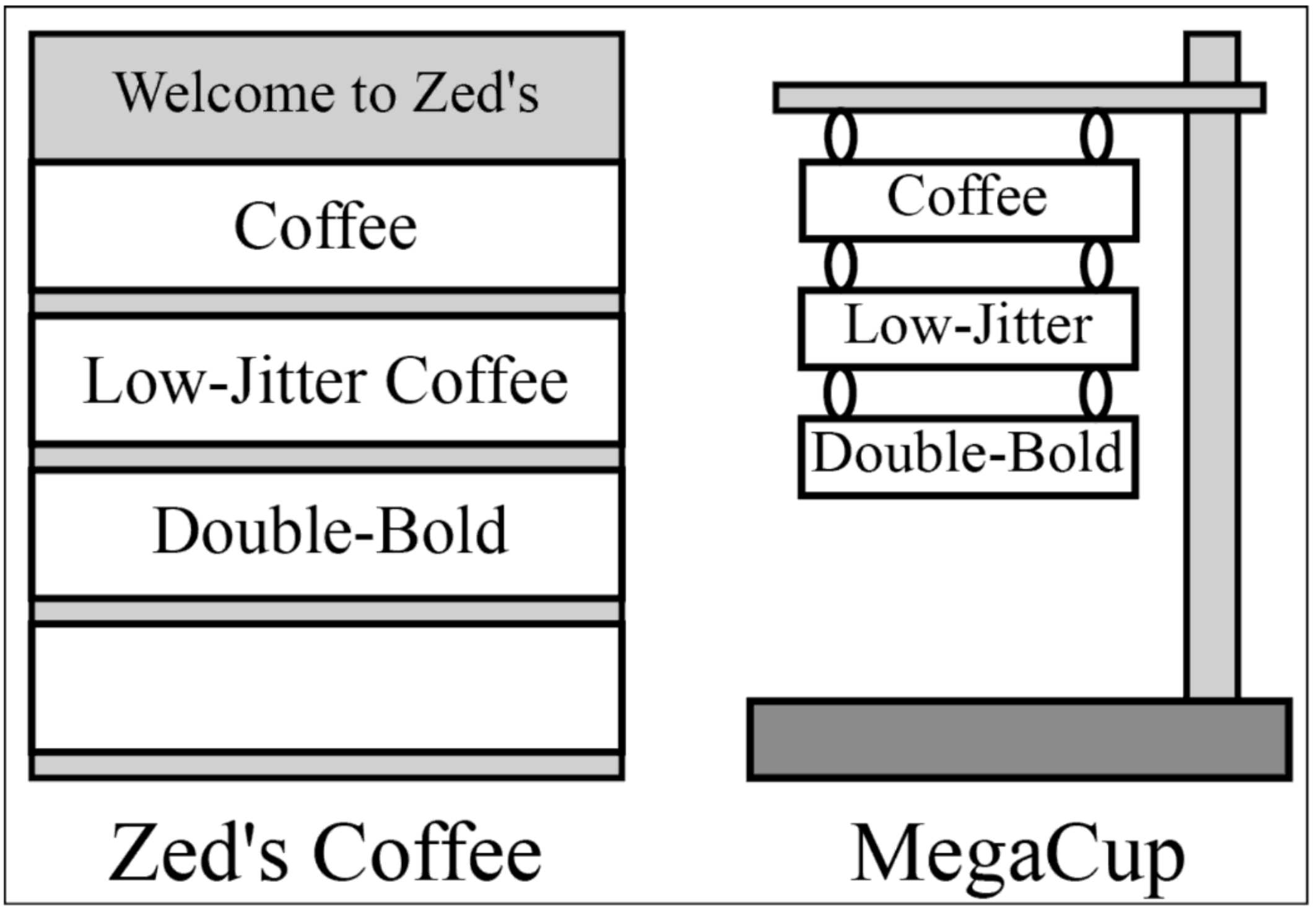
noticed a new shingle on MegaCup's sign: "Low-Jitter Coffee." The war was on.



Then his supplier told him about another type of coffee. Called "Double-Bold Coffee," it was significantly stronger than the normal brew. A single cup could keep you awake all night. Zed ordered eight cases and a new menu tag for the ArrayDesignBoard menu.

Again, the new coffee became a rapid success. His morning crowd loved it. The steward alone ordered three extra-large cups each morning. Zed also started attracting new customers from the castle's night guards. They needed something strong to keep them awake during their watch.

Alas, MegaCup soon added a new shingle to the end of its sign.



The next time his supplier visited, Zed grilled him on the other types of coffee available. After obsessing over the supply lists, Zed decided to try a novel approach. He ordered one case each of ten different flavors. He put these flavors into a rotation, constantly offering new variety.

This rotation approach worked particularly well with Zed's sign. Every time he switched a flavor, he would remove one tag and slide a new one in. Sometimes he changed the menu a few times in one day, such as replacing "Double-Bold" with "Low-Jitter" after noon.

MegaCup took a different approach. The manager quickly found that, while adding new shingles to the end of the list was easy, removing them was frustrating. In order to remove a shingle, he had to: unlink it from both the shingle above and the shingle below, then reattach the shingle below to the one above. It was a time-consuming process. He decided to take advantage of the sign's ability to easily expand offerings. He instead offered six different coffees on a semi-permanent basis. On rare occasions, he would grudgingly spend fifteen minutes unlinking a shingle on his sign and adding a new one.

The two coffee shops operated in that mode for years. Zed's coffee shop rotated through different options, and MegaCup offered a more constant, but larger, selection.

Both businesses thrived as the market for coffee grew. Eventually, Zed's Coffee House became one of the largest businesses in the kingdom, with over a hundred different locations. Zed continued to expand aggressively until the great sugar famine hit. With business dropping due to the lack of sugar, Zed decided to leave the world of

coffee and speculate in coconut sales.

Strings and Pigeon Messages

Strings are sequences of characters. In many programming languages, strings are implemented as an array of characters, and you can access each character as you would any value in an array.

* * *

“Only twenty letters?” asked Ann. The limit seemed ridiculous. Who had ever heard of a pigeon-carrier message with a length limit?

“Yes,” confirmed Guelph’s pigeon master. “We don’t have the kingdom’s strongest pigeons here. We need to be careful about the weight of the messages.”

“But twenty letters is so short,” objected Ann.

“It’s actually a twenty character limit,” clarified the pigeon master. “Spaces count toward your total. So does punctuation.” He laid out a tiny rectangle of parchment on the counter. It had twenty tiny gray squares. Each square was large enough to hold a single character.

“I suppose we could send for a stronger pigeon,” offered the pigeon master. “It might take a while, but we’ve done it before. I hear the castle has pigeons that can carry multiple pages of information. Can you imagine that?” He smiled wistfully and looked out the window. Ann had never seen a case of pigeon envy this bad.

Ann shrugged. Truthfully, twenty characters would be more than enough. She had yet to find any useful information about the darkness. This message was a courtesy to her father; she had promised regular updates.

Without a good reason to argue for a longer message, Ann set about filling in the tiny squares: “No progress. -Ann.” Seventeen letters.

N	o		P	r	o	g	r	e	s	s	.		-	A	n	n			

Ann took a moment to consider where she could add more information. She could strip out the punctuation, but that would save her only two characters. Anyway, she had nothing more to say.

She paid the pigeon master and watched him attach the message to the pigeon’s leg. The bird lethargically flapped away, barely clearing the windowsill. Ann briefly wished that she could follow the pathetic bird back to the castle. Instead, she left the communications office and continued on her quest.

As prescribed by Sir Galwin’s algorithm, Ann needed to find more information. She had no leads and her recent attempts to consult Dr. Conjunctione, Dr. Whileton, and now Dr. Iterator had failed