Computational Thinking—An Overview

Paul S. Wang, Kent State University

February 23, 2020

Why Computational Thinking?

Digital computers brought us the information revolution. People in the information age have such wonderful tools as computers, smartphones, and the Internet. *Information Technology* (IT) brought us tremendous benefits as well as brand new challenges. For example, we can *ask Google* to answer any questions we may have on just about any subject and usually get answers instantly. The Internet spans the globe and brings all parts of the world within instant reach. Yet, it also allows hackers to steal our information or worse, holding our computer or sensitive data for ransom.

As modern citizens, we need to adopt IT techniques and products effectively, and to wisely mitigate the risks brought by the information highway. That's where *computational thinking* (CT) can come into play.

What Is Computational Thinking?

Computational thinking is the mental skill to apply fundamental concepts and reasoning, derived from modern computers and IT, in all areas, including

day-to-day activities. CT is thinking inspired by an understanding of IT, its advantages, limitations, and potential problems. CT also encourages us to keep asking questions such as "*What if we automate this?*" "*What instructions and precautions would we need if we were asking young children to do this?*" "*How efficient is this?*" and "*What can go wrong with this?*"

CT can expand your mind, help you solve problems, increase efficiency, avoid mistakes, and anticipate pitfalls, as well as interact and communicate better with others, people or machines. CT can make you more successful and even save lives!

Who Promotes Computational Thinking?

Back in March 2006, Dr. Jeannette M. Wing published an article on computational thinking in *The Communications of ACM* and boldly advocated it as a skill for everyone:

> "Computational thinking builds on the power and limits of computing processes, whether they are executed by a human or by a machine. Computational methods and models give us the courage to solve problems and design systems that no one of us would be capable of tackling alone. ... Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking."

Within the academic research community, there have been significant discussions on computational thinking, what it encompasses, and its role inside the education system.

In educational circles, there is an increasing realization of the potential importance of learning to think computationally. According to a recent report on computational thinking by the National Research Council of The National Academies (NRC):

> "... Computational thinking is a fundamental analytical skill that everyone, not just computer scientists, can use to help solve problems, design systems, and understand human behavior. ... Computational thinking is likely to benefit not only other scientists but also everyone else. ..."

A Powerful Way of Thinking

OK great, CT is important. But what exactly are the concepts and methodologies it provides? Here is a list of some main aspects of CT:

- Simplification through abstraction—Abstraction is a technique to reduce complexity by ignoring unimportant details and focusing on what matters. For example, a driver views a car in terms of how to drive it and ignores how it works or is built. A user cares only about which mouse button to click and keys to press and generally overlooks how computers work internally.

- Power of automation—Arranging matters so they become routine and easy to automate. Working out a systematic procedure, an algorithm, for carrying out recurring tasks can significantly increase efficiency and productivity.

- Iteration and recursion—Ingeniously reapplying the same successful techniques and repeatedly executing the same set of steps to solve problems.

- An eye and a mind for details—Small things such as characters in uppercase versus lowercase or with an extra space can make all the difference. Any piece of data may be subject to interpretation depending on the context. You need eyes of an eagle, mind of a detective, and a careful and meticulous approach. Overlooking anything can and will lead to failure.

- Precision in communication—Try telling the computer to do what you mean and not what you say `;-)`. You need to spell it out precisely and completely. Don't spare any details. Vagueness is not tolerated. And contexts must be made explicit.

- Logical deductions—"Cold logic" rules. Causes will result in consequences, whether you like it or not. There is no room for wishful or emotional thinking. Don't we all wish some of this seeps into such things as our politics?

- Breaking out of the box—A computer program executes code to achieve any task. Unlike humans, especially experts, it does not bring experience or expertise to bear. Coding a solution forces us to think at a dumb computer's level (as if talking to a one-year-old) and get down to basics. This way, we will naturally need to think outside any "boxes."

- Anticipating problems—Automation relies on preset conditions. All possible exceptions must be met with prearranged contingencies. Ever said "I'll take care of that later"? Because there is a chance you might forget, according to CT, you should have a contingency plan ready in case you do forget. Otherwise, you have set a trap for yourself.

These are just some of the main ideas. CT offers you many more concepts and ways to think that can be just as, if not more, important.

In the author's new textbook *From Computing to Computational Thinking* (CRC Press 2015), a new word is defined.

Definition: **computize**, verb. To apply computational thinking. To view, consider, analyze, design, plan, work, and solve problems from a computational perspective.

When considering, analyzing, designing, formulating, or devising a solution/answer to some specific problem, computizing becomes an important additional dimension of deliberation.

Where to Apply CT?

People say "hindsight is 20/20." But, since automation must deal with all possible applications in the future, we must ask "what if" questions and take into account all conceivable scenarios and eventualities. Let's look at a specific example. Hurricane Sandy was one of the deadliest and most destructive hurricanes in US history. Thank goodness it didn't hit Kent or Ohio for that matter.

With CT at multiple levels, dare we say that many of the disasters from Sandy might have been substantially reduced?

- The New York City subway entrances and air vents are at street level. What if streets are flooded? What if flood water enters the subway?
- What if we need to fight fires in a flooded area? Do we have fire boats in addition to fire trucks? Do we have firefighters trained for boats?
- Most portable emergency power generators run on gasoline. What happens if gas runs out and gas stations are flooded?
- What if the drinking water supply stops? Can we provide emergency water from fire hydrants? In that case, can we use a mobile contraption that connects to a hydrant, purifies the water, and provides multiple faucets?
- What if emergency power generators are flooded? Should we waterproof generators in designated at-risk buildings?
- What if cell towers lose power? How hard is it to deploy airborne (drone?) cell relays in an emergency?
- What if we simulate storm damage with computer modeling and find out ahead of time what to prepare for?

So let's computize at multiple levels and do our best to get 20/20 hindsight beforehand.

CT Success Story

A *loop* in a program is a construct that applies the same set of steps repeatedly until a certain goal is achieved. This technique is known as *iteration* in CT.

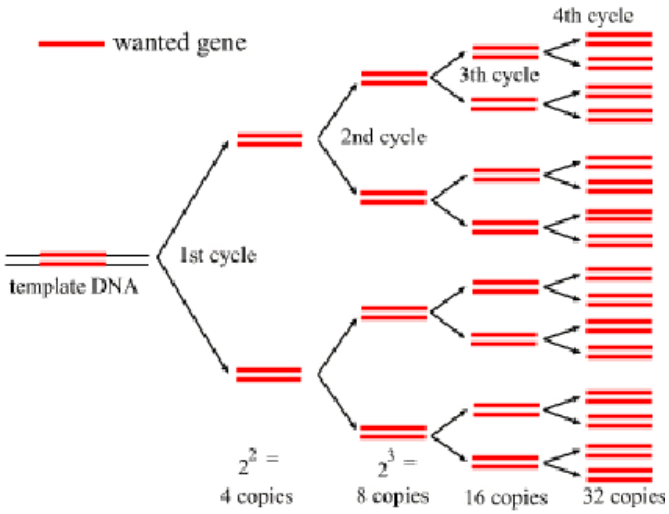
Iteration of a process has led to the invention of the *polymerase chain reaction* (PCR), a technique in molecular biology to generate thousands to millions of

copies of a particular DNA sequence. Developed by Dr. Kary Mullis in 1983, PCR is now indispensable in medical and biological research and applications, including DNA testing and genetic fingerprinting. The impact of automated PCR is huge and far-reaching. Mullis was awarded the 1993 Nobel Prize in Chemistry for his part in the invention of PCR.

In recounting his invention, Dr. Mullis wrote in his book *Dancing Naked in the Mind Field*:

> *I knew computer programming, and from that I understood the power of a reiterative mathematical procedure. That's where you apply some process to a starting number to obtain a new number, and then you apply the same process to the new number, and so on. If the process is multiplication by two, then the result of many cycles is an exponential increase in the value of the original number: 2 becomes 4 becomes 8 becomes 16 becomes 32 and so on.*
>
> *If I could arrange for a short synthetic piece of DNA to find a particular sequence and then start a process whereby that sequence would reproduce itself over and over, then I would be close to solving my problem.*

At the time of the invention, the "polymerase" and other related DNA duplication techniques were already known. It was the "chain reaction" part that was missing. Well, we have Dr. Mullis and his computational thinking to thank for the invention. And what a significant invention! *The New York Times* described it as "highly original and significant, virtually dividing biology into the two epochs of before P.C.R. and after P.C.R."



Still need more convincing? Just ask the Cleveland Clinic, the Innocence

Project, any guiltless person freed from jail, or people finding their genealogical roots, through DNA testing.

Conclusion

In this brief overview, it is impossible to cover the many aspects of Computational Thinking and how it can help individuals, organizations, and society as a whole. We may continue with a series of articles in the future. For now, it suffices to say that, as history demonstrates time and again, a society, that is better educated in the next dominating technology and that can merge a new way of thinking into other disciplines, will have a significant competitive edge over others.

Everyday Computational Thinking
It Can Save Lives

Paul S. Wang, Kent State University

February 23, 2020

Introduction

The great digital revolution is here and now. The Internet, the Web, and the computer in its many different forms, are providing instant communication across vast distances and changing nearly every aspect of our day-to-day living.

In the March 2017 edition of *AroundKent* (Vol 13, online at `aroundkent.net`), we gave an overview of *Computational Thinking* (CT) and stated "CT is thinking inspired by an understanding of IT, its advantages, limitations, and potential problems" and understood that CT was a powerful way of thinking.

We also defined a new verb **computize**:

> *To apply computational thinking. To view, consider, analyze, design, plan, work, and solve problems from a computational perspective. When considering, analyzing, designing, formulating, or devising a solution/answer to some specific problem, computizing becomes an important additional dimension of deliberation.*

In this article, we will explain how CT can be applied by everyone in everyday situations and how it can make important differences and even save lives.

What Is An Algorithm?

Remember the hugely successful movie *The Social Network* (2010)? It told the story of Mark Zuckerberg and how he created Facebook. The motion picture also introduced the term *algorithm* to much of the world for the first time.

From the author's book *From Computing to Computational Thinking* we understand an algorithm is a step-by-step procedure.

The origin of the word "algorithm" traces back to the surname Al-Khwārizmī of a Persian mathematician (780–850 CE), who was well-known for his work on algebra and arithmetic with Indian numbers (now known as Arabic numbers). The modern-day meaning of algorithm in mathematics and computer science relates to an effective step-by-step procedure to solve a given class of problems or to perform certain tasks or computations.

Specifically, a procedure becomes an algorithm if it satisfies all of the following criteria:

- **Finiteness**: The procedure consists of a finite number of steps and will always terminate in finite time.
- **Definiteness**: Each step is precisely, rigorously, and unambiguously specified.
- **Input**: The procedure receives certain data (or none) as input before it starts. Possible values for the data may vary within limitations.
- **Output**: The procedure produces results as its output.
- **Effectiveness**: Each operation in the procedure is basic and clearly doable.

For a given problem, there usually are multiple algorithms for its solution. The design and analysis of algorithms are central to computer science and programming.

What have algorithms to do with everyday computational thinking? Good question. Well, it has to do with setting goals, devising concrete steps to achieve them, anticipating problems, and arranging solutions in advance. That's pretty important for everything everyday, right?

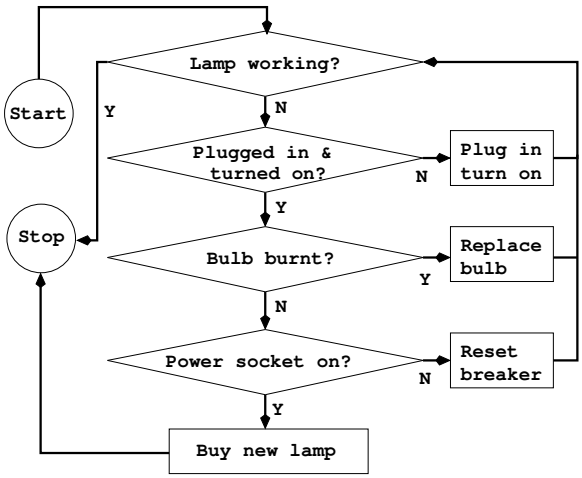
Flowcharts

An algorithm is basically a procedure to achieve a certain goal.

A *flowchart* presents a procedure visually with words and diagrams. For any procedure, we can use a flowchart to plan the sequences of steps, to refine the solution logic, and to indicate how to handle different possibilities. Here is a simple flowchart for the task of "getting up in the morning."

We begin at the `Start` and follow the arrows to each next step. A diamond shape is used to indicate a fork in the path. Which way to turn depends on the conditions indicated. Obviously, we use diamond shapes to anticipate possibilities. The snooze option leads to a branch that repeats some steps. In programming, such a group of repeating steps is called a *loop*. The procedure ends when the person finally climbs out of bed.

As another example, let's look at a flowchart for troubleshooting a lamp.



The very first step after `Start` is significant. Although the purpose of the procedure is to troubleshoot a lamp, we nonetheless, make no implicit assumption that the lamp is not working. Without this step at the beginning, the procedure would potentially troubleshoot a perfectly good lamp, and, worse yet, would decide to replace it with a new lamp!

Each of the next three steps tests for a particular problem and makes a fix. Then the same procedure is reiterated by going back to step one to determine if the lamp is now working. This flowchart is a bit more complicated. Yet, it is worth careful examination. That is also a good way to get into the head of a programmer.

All you need is pencil and paper to start drawing your own flowcharts. Try it and you may find it not so difficult. To make nice looking flowcharts you can find many tools on your computer as well as online.

Correctly setting goals and anticipating potential problems are important aspects of devising a procedure.

We all were, and still are, horrified and outraged by the United Airlines passenger dragging incident (April 2017). But if the airline had set "passenger service" truly as its goal then it would have used a procedure that increased

the incentives until getting enough volunteers to give up their seats. One can't help but wonder. Could this be the tip of an iceberg of problems caused by a wrong-headed culture? The incident unnecessarily inflicted much harm on the victim, the company, employees, shareholders and the police.

Everyday Applications

For us, we want to avoid fussy, vague, confused, wishful, emotional, impulsive, optimistic, or pessimistic thinking. We want to practice CT anytime and anywhere we can. We want to set clear goals, have a sequence of steps to achieve them, anticipate problems and prepare solution plans in advance.

Take driving a car for example. What is the goal? It is to get to a destination safely. It is not enjoying the sound system, watching the scenery, or engaging in conversation, although we have nothing against any of that as long as it does not get in the way of safe driving. Texting and driving is never safe.

Stopped at a traffic light, we wait for the light to turn green. But, we may need to run the red light if an 18-wheeler is about to crash into us from behind. That means we need to be checking our rear-view mirror while waiting for the green light. When the light turns green, do we blindly rush into the intersection? What if a car is running the tail end of the yellow light or the red light?

Thus, the goal is not to obey traffic signals, but to make sure it is safe. In the United States, a car crash kills a person every 12 minutes on average. If you are thinking straight, is a car a fun machine or a dangerous one? CT can keep you focused on the goals, make you pay attention to details, plan for contingencies, and shield you from distractions. CT can save the day, and perhaps even your life!

Now let's apply CT to the task of "getting ready to drive a car" and write down an algorithm-inspired pre-drive checklist.

1. Am I ready to leave? Forgot to bring anything?
2. Walk around the car, check windows, tires, lights, back seat, and any objects and activities near the car.
3. Get in the car, foot on brake, close and lock all doors.
4. Adjust seat and steering column positions as needed. Check positions of all rear-view mirrors, buckle up.
5. Check the instrumentation panel, pay attention to the fuel level.
6. If necessary, familiarize yourself with the controls for lights, turn signals, wipers, heat/AC, and emergency signal. Make sure they are working properly.
7. Release the hand break, start the engine, shift gear.
8. Make sure the gear is in D or R as intended, then start driving.

Airlines have developed rigorous preflight checklists for safety. Incidents, sometimes fatal, happen when pilots and crew, failing computational thinking, do not follow the exact procedure. The same goes for doctors and nurses in hospitals, especially in operating rooms.

You Can Do It!

We have given a sampling of computational thinking in everyday situations. However, we have by far, not exhausted the possibilities and will perhaps continue this series of articles on CT.

We hope you liked this article and please feel free to give your feedback directly to the author (`pwang@cs.kent.edu`).

For most people, CT does not come naturally. Research has shown that snap judgment based on intuition and experience is the norm. But, by keeping CT in mind and consciously making it an added dimension to our deliberations, we can make it work for us.

By being creative, everyone can derive benefits from CT every single day. As a result, our community, even the entire society, will be better off by becoming more efficient and effective.

Acknowledgments

Many thanks to Matt Keffer for the nice pictures and to Lonnie Hawks as well as Jennifer Wang for proofreading and helpful suggestions.