# Abstraction

Principles of Computing (15-110)

Instructor: Giselle Reis

Lecture on Sunday 19th January, 2020

## 1 Simplifying sequences of steps

You might have noticed that writing sequence of steps in English (or in any natural language) quickly becomes quite cumbersome. The lines are too long and explaining that we need to compare a card with "the next card which is on the right side of the card before" requires a lot of thinking and not loosing track of the sentence before it ends. That is why we need to use a more constrained and less ambiguous language when we want to be really clear (such as when writing sequences of steps).

In order to do that, we are going to use *names* and *indices* in a way that is closer to math (and consequently to computing). Let's transform our program for finding the highest card in a deck of cards. The first thing we do is define the names we are going to use, what they mean, and what are their values (if any). When we have a sequence of things, it is useful to denote it using indices, such as $a_0, a_1, a_2, ...a_n$ (this is a sequence with $n + 1$ elements).

```
1. Let $c_0, c_1, ..., c_n$ be the set of cards.
2. Let $\mathsf{max} = c_0$ represent the highest card we have seen.
3. Let $i = 0$ be the index of the last card we checked.
4. Check if $i < n$:
   (a) If yes: Check if $c_{i+1} > \mathsf{max}$
       i. If yes: $\mathsf{max} = c_{i+1}$; $i = i + 1$, go back to step 4.
       ii. If no: $i = i + 1$, go back to step 4
   (b) If no: highest card is $\mathsf{max}$
```

Try rewriting the age computation algorithm using names for the values and see how concise it becomes!

Another technique we can use is identifying when a sequence of steps is repeating, and for what values, and simply say: "repeat this for all values...". In the algorithm above, steps 4.a to 4.b are supposed to be executed for every $i$ from 0 to $n - 1$. This means we can rewrite it as:

```
1. Input: Let $c_0, c_1, ..., c_n$ be the set of cards.
2. Let $\mathsf{max} = c_0$ represent the highest card we have seen.
```

```
3. Let $i = 0$ be the index of the last card we checked.
4. Repeat for $i = 0, 1, 2, ...n - 1$:
   (a) Check if $c_{i+1} >$ max
       i. If yes: max $= c_{i+1}$
5. Output: Highest card is max
```

Note that incrementing $i$ each time and going back to step 4 are now implicit in the repeat command. That means that the "If no" part of the check is empty, and we can completely get rid of it. In this version, we have also indicated what is the *input* and *output* of the algorithm. The input is the information that you are given, and the output is the final answer we were looking for.

Using this kind of technique will typically make your steps more precise and less confusing. As a bonus, they become much closer to a program. That means that it will be easier to type this in Python when the time comes!

## 2 Abstraction

In a previous lecture we have analysed the problem of choosing snacks given a certain budget and calorie intake. Let's rephrase it: You are at the cafeteria and you need to get a snack costing at most 15 Ryials. Because of your diet, you know that you need to eat a high-calorie snack at that time of day. Your options are:

| | | |
|---|---|---|
| Muffin: | 5 QAR | 500 |
| Croissant: | 7 QAR | 450 |
| Chips: | 10 QAR | 700 |
| Hamburger: | 8 QAR | 800 |
| Chocolate: | 2 QAR | 300 |
| Fruit salad: | 6 QAR | 200 |

How do we choose the best combination? We have previously established that we need to test each possible choice.

Now take a different problem. Your character in a role playing game has a bag that can only carry 15 kg of items. You have a set of items of different weights and values, and you would like to have the most valuable bag as possible. In other words, how can you choose 15 kg of items such that they add up to the highest possible value? Your items are:

| | | |
|---|---|---|
| Sceptre: | 4kg | 10$ |
| Shoes: | 1kg | 1$ |
| Helmet | 1kg | 2$ |
| Armour: | 12kg | 4$ |
| Dagger: | 2kg | 2$ |

How do you choose the items to carry?

Do you think the two problems above have something in common?

They are, actually, the *same* problem! If we find a step-by-step solution for the first one, we can definitely transform it (with minimal effort) into a solution of the second one. The *core* of the two problems are:

- There are a set of items to choose from with two associated values.
- The answer consists of a subset of items such that one value is minimized/maximized and the other value adds up to a certain amount $k$.
- The items cannot be split.

*Abstraction* is the ability to overlook the unimportant details of a problem and focus only on the important core parts of it. By doing this, we can transform the problem into something else for which we have a solution.

**Example: Balancing chemical equation**

When learning about chemical reactions, we often need to balance chemical equations such as this one:

$$\mathrm{Al} + \mathrm{O}_2 \longrightarrow \mathrm{Al}_2\mathrm{O}_3$$

The goal is to assign coefficients for each molecule such that the number of atoms is the same on each side. In school, we usually learn how to do this by trial and error, but if we stop to think about this problem a little bit more, there is a *systematic* way of finding the solutions (i.e., not by trial and error).

The goal is to find the coefficients, so let's write them as *variables* in the equation:

$$x\mathrm{Al} + y\mathrm{O}_2 \longrightarrow z\mathrm{Al}_2\mathrm{O}_3$$

The number of Al atoms on the left is $x$ and on the right is $2z$. We want that $x = 2z$, or $x - 2z = 0$. The same holds for O, we want that $2y = 3z$, or $2y - 3z = 0$. Therefore we need to find $x, y, z$ such that the following equations are satisfied:

$$\begin{cases} x - 2z = 0 \\ 2y - 3z = 0 \end{cases}$$

We have transformed the problem of balancing chemical equations into a problem of solving a system of linear equations. A computer does not know about chemical equations, but it sure knows a lot about linear equations. There are many algorithms for solving this type of problem, so it is just a matter of choosing your favorite one.

## 3 Exercise: binary numbers

Numbers are represented in a binary system by using their representation in base-2. To understand this representation, we will look at the one we usually use, in base-10, and generalize from there. You might not realize every time you read a number, but here's whats actually going on:

$$\begin{aligned} 42 &= 4 \times 10 + 2 \times 1 \\ &= 4 \times 10^1 + 2 \times 10^0 \end{aligned}$$

Our numbers are a combination of powers of 10 multiplied by a coefficient, and their representation is simply these coefficients put together in a certain order: starting from 0, concatenate them from right to left until the last non-zero coefficient. These coefficients range from 0 to 9, can you guess why?

This same idea can be applied to any natural number, not only 10. The next most popular bases are 2 and 16[1]. Let's apply the same idea to a number in binary (this time the number on the left of $\equiv$ is in base 2 and the number on the right is in base 10):

$$\begin{aligned} 1010 &\equiv 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\ &\equiv 8 + 2 \\ &\equiv 10 \end{aligned}$$

Hence all numbers can be represented in a binary system. Write the step-by-step instructions for transforming a binary number into its decimal representation (use variables to make your life easier).

```
1. Input: $d_0 d_1 ... d_n$ a binary number
2. Let $sum = 0$
3. Let $p = 1$
4. Repeat for $i = n, n-1, n-2, ..., 0$:
    (a) $sum = sum + d_i * p$
    (b) $p = p * 2$
5. Output: $n$
```

[1]Fun fact: for base 16 some coefficients might have two digits. In order to make the representation unambiguous the letters A to F are used for coefficients 10 to 15.