6. 알고리즘과 절차적 사고

2010 데이터로 표현하는 세상 요약본
고려대학교 김현철 교수
hkim64@gmail.com

6.1 알고리즘(Algorithm)이란 무엇인가?

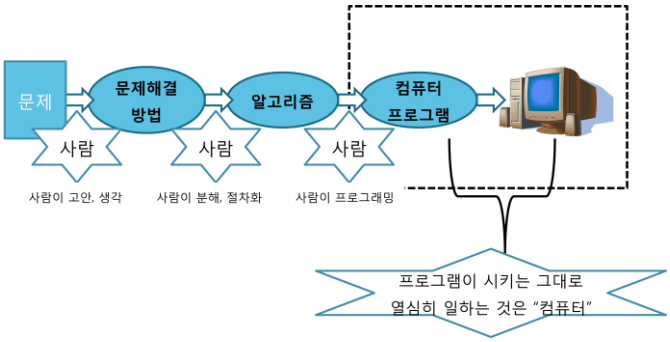
위키피디아에 나온 알고리즘에 대한 정의를 나름대로 의역하여 번역하면 다음과 같다.

"수학, 컴퓨팅, 언어학과 같은 분야에서 계산이나 데이터 처리에 주로 사용되는, 유한(有限) 개수의 명령어들의 순차적인 나열이다. 어떠한 일을 수행하기 위하여, 잘 정의된 명령어들을 적절한 순서대로 나열을 하고, 그 나열된 순서대로 명령어들을 차례대로 수행하면 결국 최종적으로 그 일이 완성되게 하는 효과적인 방법이다."

알고리즘을 잘 설명하는 예는 흔히 컴퓨팅의 아버지라고 불리는 알란 튜링 (Alan Turing: 1912-1954)에 대한 일화에서 볼 수 있다. 튜링의 집에는 교육을 받지 못한 가정부가 있었다고 한다. 튜링은 그 당시 어려운 수학문제를 어떻게 하면 자동으로 해결할 수 있는 가상의 기계를 만들 수 있을까를 고민하고 있을 때였는데, 어느 날 그 가정부에게 그 수학 문제 풀이를 할 수 있도록 하는 방법을 생각을 해 봤다고 한다. 교육을 전혀 받지 못한, 그래서 기본적인 단순 논리 사고만 할 수 있는 그 가정부가 어려운 수학문제를 풀 수 있도록 하게 한다면, 기계에게도 그 수학문제를 풀 수 있도록 만들 수 있을 것이라고 생각한 것이다.
튜링이 사용한 방법은 그 어려운 수학 문제 풀이 과정을 아주, 아주 잘게 쪼개어, 아주 단순한 기계적인 사고만으로도 해결할 수 있는 단위까지 잘게 쪼개어, 그것을 순서대로 나열해 놓고, 그 가정부에게 그 순서대로 따라 하도록 하는 것이었다. 그 가정부는 그 순서대로만 일을 수행을 하였고 결국 어렵지 않게 수학 문제 풀이를 완성하더라는 것이다. 따라서 튜링은 인간이 해결하는 어떠한 문제라도 그 해결 과정을 기계가 처리할 수 있는 단위까지 아주 잘게 분해하여 순서대로 나열해 놓고, 그 기계에게 차례대로 처리하도록 할 수만 있다면 인간의 문제를 자동으로 처리하는 기계를 설계할 수 있다고 생각한 것이다.
이 가정은 다시 생각해 보면, 정보를 아주 작은 최소의 단위로 분해하면 0과 1로 분해할 수 있고, 이 0과 1의 순차적인 나열을 읽고 처리할 수 있는 기계를 만들 수가 있는데, 튜링은 이러한 모델을 만들었으며 이것을 "튜링머신(Turing Machine)"이라고 부른다. 이 튜링머신이라는 가상의 모델은 오늘날 우리가 사용하고 있는 모든 컴퓨터의 이론적 모델로 사용되었다.
기계가 이해할 수 있을 정도로 문제 해결 방법을 잘 쪼개어 순서대로 나열하기만 하면 컴퓨터라는 기계는 그것을 자동으로 처리할 수 있다. 예를 들어, 직원이 10만 명이 있는 회사에서 모든 직원의 연봉을 계산해 내는 방법을 알고리즘으로 만들어 컴퓨터로 자동으로 처리하고자 한다고 가정해보자. 알고리즘은 매우 일반적으로 만들어야 하는데, 그래야만 어떤 직원의 조건과 정보가 "입력"으로 들어오더라도 그 알고리즘에 의하여 계산된 연봉이 "출력"될 수가 있기 때문이다. 그 알고리즘을 프로그램으로 변환을 하여 컴퓨터에 집어 넣으면, 컴퓨터는 같은 처리 순서대로 연봉 계산을 10만 명이 아니라 100만 명에 대해서도 빠르게 자동으로 처리할 수 있게 되는 것이다. 한번 만이 아니라 매달, 매년, 계속 자동으로 처리할 수 있게 되는 것이다.

여기서 한가지 의문점이 있다.
문제해결방법을 알고리즘으로 만들어 기계에 넣어주면 그 다음은 기계가 그 모든 복잡하고 지루한 반복적인 작업을 빠르게 실수없이 처리할 수 있게 된다. 그러면, 문제해결 방법은 누가 생각해 낼 것인가? 그 생각해 낸 문제해결 방법은 누가 알고리즘 형태로 바꾸어 줄 것인가? 그것도 컴퓨터라는 기계가 자동으로 처리 해 줄 수 있는 것인가?
대답은 "아니다" 이다. 그냥도 아니고 "전혀 아니다"이다. 문제가 있을 때 그 문제를 해결하는 효율적인 방법은 100% 사람이 생각해 내야 하는 것이고, 그것의 반복적인 지루한 작업만 컴퓨터에서 자동으로 처리할 수 있게 할 수 있는 것이다.

아래의 그림을 보자.



어떠한 문제가 있으면, 그 문제의 해결 방법은 "사람"이 생각해 나고, 그것을 알고리즘으로 "사람"이 변환해 내고, 그 알고리즘을 "사람(프로그래머)"이 컴퓨터 프로그램으로 코딩하고, 그 컴퓨터 프로그램이 컴퓨터에 들어가야만 비로서, 그 컴퓨터는 "사람"이 지시한 문제 해결 방법대로 자동으로 쉬지 않고 계산처리 하는 것이다. 앞서 예를 들었던 10만 명의 종업원을 둔 회사의 경우를 생각해 보자. 직원들의 연봉과 월급계산이라는 "문제"에 대한 "문제해결방법(solution)"은 그 회사의 사장이나 임원이나 인사전문가가 제시하여야 하는 것이고, 그 제시된 것에 의해서 알고리즘으로 변환하는 것은 인사전문가나 정보담당전문가, 혹은 프로그래머가 할 수도 있다. 만들어진 알고리즘을 프로그램으로 코딩하는 것은 컴퓨터 프로그래머가 하는 일이고, 그것을 컴퓨터를 돌려서 운영하는 것은 또 다른 사람이 하는 일이다. 만약, 프로그램된 컴퓨터가 없다면 경영진에서 만든 그 "방법"에 의해 계산 담당 직원(사람)이 직접 10만 명의 월급 계산을 일일이 해야 했어야 할 것이다. 예를 들어, 1명의 월급 계산하는데 1분의 시간이 걸린다고 하면, 10만분이 걸리게 되는데 10만분은 69일이 걸리는 시간이다. 10만 명에 대하여 같은 계산 "방법"이 적용되므로, 당연히 계산담당자는 자동화 할 수 있는 방법을 찾게 될 것이며, 그 자동처리를 담당하는 것이 바로 "컴퓨터"이다. 컴퓨터는 단 몇 분 만에 10만 명의 연봉을 계산 해 낼 수 있을 것이다.

따라서 그 회사에서 연봉과 월급 처리하는 과정에서 무슨 에러가 발생하고 문제가 발생했다면, 그 잘못은 "컴퓨터 때문이 아니라", 그 컴퓨터에게 계산처리 명령을 준 "사람"때문이라는 것을 알 수 있을 것이다. 그 "사람"은 경영/인사전문가일 수도 있고, 프로그래머일 수도 있다. 하지만, 시킨 대로 정확하게 일만 열심히 한 컴퓨터는 잘못이 없다. 따라서 위의 그림에서 가장 중요하게 강조되어야 하는 부분은 바로 문제해결방법을 생각해 내는 단계와 그것을 알고리즘으로 표현해 내는 단계라고 할 수 있다. 일단 알고리즘만 만들어지면 그 다음 단계는 그리 어려운 일이 아니기 때문이다.

문제해결방법을 알고리즘으로, 즉 작은 단위로 분해하여 순차적 나열로 표현한 것은, 꼭 컴퓨터에 넣어서 자동화 기계로 만들지 않아도, 우리의 일상에서 아주 많은 도움을 줄 수 있다. 어떠한 업무조직의 복잡한 일 처리 과정이 이러한 알고리즘을 통하여 명확하게 표현되고, 다시 효율적으로 조정될 수 있게 된다. 또한 컴퓨터 프로그램을 전혀 모르더라도, 주변에서 업무능력이 뛰어나고 분석능력이 뛰어난 사람들을 보면 대부분 이러한 "알고리즘적 사고" 혹은 "절차적 사고" 능력이 뛰어난 사람들이라고 할 수 있다. 알고리즘적 사고는 정확하고 효율적인 문제해결 방법을 이끌어 낸다.

이후의 내용은, 알고리즘의 표현 방법에 대하여 먼저 살펴 보고, 그 다음으로 어떻게 인간이 효율적인 문제해결 방법을 생각해 낼 수 있을까에 대하여 살펴보도록 하자.

6.2 알고리즘의 절차적 표현방법

알고리즘은 문제해결 방법을 작은 작업 단위로 분해하여 step-by-step으로 순차적으로 나열한 것이다라고 하였다. 좀 더 엄격하게 정의하자면, 이 알고리즘은 무한대로 계속 진행되는 것이 아니라 유한한 시간 안에 끝나야 하는 것이며, 그 분해된 작업 단위도 분야에 따라서 엄격하게 미리 정해 놓기도 한다.

앞서 설명한 10만 명 직원 회사의 예를 다시 생각해보자. 직원의 연봉과 월급 계산하는 알고리즘을 만들었다고 한다면, 어떠한 직원의 정보 (예를 들면, 나이, 업무, 근무 연수, 자격증 소지 여부 등)가 "입력(input)"으로 들어가면, 그 알고리즘은 입력된 데이터를 가지고 정의된 절차에 따라서 계산을 하여, 그 결과값을 "출력(output)"하게 된다. 즉, 알고리즘은 단독으로 존재한다기보다는 "입력되는 데이터"와 "출력되는 결과값"에 의하여 정의가 된다. 즉, Input이 들어 왔을 때, 그것을 가지고 원하는 output을 내보낼 수 있도록 하는 과정을 step-by-step으로 순차적으로 표현한 것이라고 말할 수 있다. Input data를 output으로 변환시키기 위한 "절차"를 정확하게 기술한 것이라고도 할 수 있다.

그래서 문제해결을 위한 알고리즘(문제 해결 방법)을 생각해 내기 위하여 흔히 사용되는 방법 중의 하나는, 먼저 input이 무엇

인가를 정의하고, 그 다음에 이 input이 들어 갔을 때 어떤 output이 나와야 하는가를 정의하고, 그 다음에 이 input이 들어갔을 때 이 output이 나오게 하려면 어떤 절차대로 하여야 하는지를 고민해 보는 것이다. 이 방법은 후반에 다시 자세하게 살펴보도록 하고, 먼저 알고리즘을 표현하는 일반적인 방법에 대하여 살펴보자.

알고리즘을 표현하는 가장 기본적인 방법은 다음 두 가지가 많이 사용된다.

1. 순서도(Flowchart) 는 알고리즘을 도형을 이용하고 시각적으로 표현한 것이다. (Flowchart is a pictorial representation of an algorithm.)
2. 의사코드(수도코드; Pseudo-code) 는 알고리즘을 쉬운 언어로 표현한 것이다. 의사코드는 각 언어로 표현된 절차 단계들을 명시적으로 구조화시킨 형태이다. 컴퓨터 프로그램과 유사한 형태로 표현한 것이라고도 할 수 있다. (Pseudo-code is an English-like representation of an algorithm. Pseudo-code uses a mixture of English and formatting to make the steps in the solution explicit)

우리는 책에서는 pseudo-code를 사용하여 알고리즘을 표현하는 방법을 사용하도록 하겠다.

먼저 아주 간단한 예를 들어, "컴퓨터의 전원을 켜는 방법"을 생각해보자.

① 컴퓨터의 전원을 켠다.
② 컴퓨터 모니터를 켠다.
③ login화면이 나타나면 id와 password를 type한다.

이렇게 순서대로 나열해 주면 된다. 간단하지 않은가?

하지만, 자세히 살펴보면, 우리는 금방 이러한 표현이 모든 상황을 다 정확하게 표현할 수 있지 않음을 알게 된다. 즉, ②에서 만약에 모니터가 이미 켜져 있으면 어떻게 할 것인가. ③에서 password가 잘 생각이 나질 않아서 몇 번 계속 맞을 때까지 시도해봐야 한다는 것은 어떻게 표현할 것인가. 정확하게, 모든 경우와 가능성을 모두 생각해 두고 작성을 해야 한다.

순서의 흐름제어 (Sequence Flow Control)

분해 된 각 단위를 명령어(instruction)이라고 부르기로 하자. 그리고 알고리즘은 이러한 instruction들을 어떠한 순서대로 나열해서 표현하는 것이라고 하자. 그런데 instructions를 모두 한 줄로 차례대로 나열하는 것은 어쩌면 너무나 길어질 수 있고 step-by-step을 효과적으로 나열하기 힘든 경우가 생길 수가 있다. 따라서 이러한 step-by-step을 표현하는데, 그 순서를 제어(control; 효율적으로 조정) 하는 것이 필요할 것이다. 아래의 그림을 보면 세가지 흐름제어(flow control) 구조를 보여주고 있다. 첫 번째는 시작서부터 끝까지 차례대로 모두 다른 instruction이 흘러가는 것이고, 두 번째는 그 중간에 어느 instruction들은 몇 차례 계속 반복해서 수행되어야 하는 경우를 표시한 것이다. 물론 이 경우도 첫 번째 구조처럼 한 줄로 늘일 수 있지만 같은 그렇게 하는 것은 보기도 힘들고 비효율적이므로 반복되는 부분만 표시하면 될 것이다. 세 번째 그림은 어떤 조건에 따라서 이쪽 instruction이 수행될 수도 있고 또는 다른 instruction이 수행될 수도 있는 경우를 보여준다. 어떤 input이 들어와 어떤 조건이 될지 모르므로 알고리즘에서는 모두 다 표시를 해두어야 하며, 이 경우에는 세 번째 와 같은 구조를 사용할 수 있다.

이 세가지 흐름 구조는 instruction들의 순차적인 나열을 좀 더 효율적으로 표현하기 휘한 것이며, 이것을 pseudo-code 형태로 다시 표현하면 다음 그림과 같다.

✓ Sequence (순차)

```
do action 1
do action 2
...
...
do action n
```

✓ Decision 혹은 Branching (분기; 길이 갈라지는 것)

```
if a condition is true,
then
    do a series of actions
else
    do another series of actions
```

✓ Repetition (반복)

```
While a condition is true,
    do action 1
    do action 2
    ...
    ...
    do action n
```

이 세가지 형식을 이용하는 것에 대하여 충분히 이해하고 익숙해져 있어야 한다.

sequence, decision, repetition을 사용하는 방법에 대한 쉬운 예를 하나 더 들어 보도록 하자.

[이를 닦자.]
"이를 닦는 것"이라는 단어를 들으면 우리의 머리 속에는 이빨을 닦는 과정이 순차적으로 생각이 떠 오를 것이다. 누구나 다 실수 없이 할 수 있는 것. 그것을 바보(컴퓨터)에게 그대로 하게 하려면 아주 세밀하게, 정확하게 그 과정을 순차적으로 나열할 수 있어야 한다.
먼저,
① 칫솔을 든다.
② 치약을 든다.
③ 치약 뚜껑을 연다.
④ 치약을 짠다.
아, 잠깐. ③에서 치약의 뚜껑을 여는 것인데, 만약에 치약 뚜껑이 이미 열려 있다면 ③은 수행할 수 없게 되고, 그러면 바보(컴퓨터)는 어찌할 바를 모르고 멈춰 버릴 수가 있다. 이 부분을 좀 더 정확히 해 두어야겠다. ③은 치약 뚜껑이 닫혀 있을 때만 수행하도록 해야 한다. 만약 열려 있다면, ③을 건너 뛰고 그 다음 ④을 수행하게 되면 된다. 어떻게 해야 할까? Decision을 사용하면 된다. 즉

① 칫솔을 든다.
② 치약을 든다.
③ If (치약 뚜껑이 닫혀 있다면)
④ Then 치약 뚜껑을 연다.
⑤ 치약을 짠다.
로 하면 된다. 여기 ③에서 만약 치약 뚜껑이 열려 있다면, 그대로 ⑤로 진행하면 된다.

계속 나가보자.
⑥ 치약이 묻은 칫솔을 입 속에 넣는다.
⑦ 칫솔을 이빨에 대고 위아래로 한번 brush한다.

문제가 있지 않은가?
⑦에서 칫솔질을 한번만 brush하고 끝낼 것인가? 계속 반복해야 하지 않는가. 언제까지 반복하는가. 당신 머리 속에 있는 이미 들어 있는 알고리즘을 한번 생각 해보자. 당신은 언제 칫솔질을 멈추는가? 스스로 만족되었을 때 칫솔질을 멈춘다고 가정해보자. 그러면 ⑦을 어떻게 바꿔야 하는가.

⑥ 치약이 묻은 칫솔을 입 속에 넣는다.
⑦ 칫솔을 이빨에 대고 위아래로 한번 brush한다.
⑧ If(불만족)
⑨ Then 위아래로 한번 brush한다.
⑩ Else exit (* exit 은 이 알고리즘을 멈추고 빠져 나간다는 의미임.)
⑪ If(불만족)
⑫ Then 위아래로 한번 brush한다.
⑬ Else exit
⑭ If(불만족)
⑮ …..계속 반복……

if(불만족) 부분을 계속 반복한다면 너무나 비효율적으로 표현하는 것일 수가 있다. 이렇게 같은 부분이 계속 반복될 때에는 repetition을 사용하자. 즉,

⑥ 치약이 묻은 칫솔을 입 속에 넣는다
⑦ while (not 만족) {
⑧ 칫솔을 이빨에 대고 위아래로 한번 brush한다.
⑨ }
⑩ exit

여기서 "while (not 만족) {"은 만족하지 않는 동안에는 계속 반복해서 { 와 } 사이에 있는 instruction들을 수행하라는 의미이다.

내가 쓰는 칫솔은 전동칫솔인데, 2분이 지나면 "삐익" 소리가 나서 시간이 그 정도 지났음을 알려준다. 이런 경우에는

⑥ while(not beeping) {
⑦ brush teeth
⑧ }
⑨ exit

로 하면 된다.

자, 이제 if (즉, decision 혹은 branching)와 while (즉 repetition) 을 어떻게 사용하는지, 어떤 의미를 가지는지 조금 익숙해 졌는가?

자 이제, 우리는 pseudo-code를 이용하여 문제 해결을 위한 step-by-step instructions를 표현하는 방법을 간략히 배웠다. Sequence, decision, repetition 을 이용하여 instruction들을 순차적으로 표현할 수 있는가?

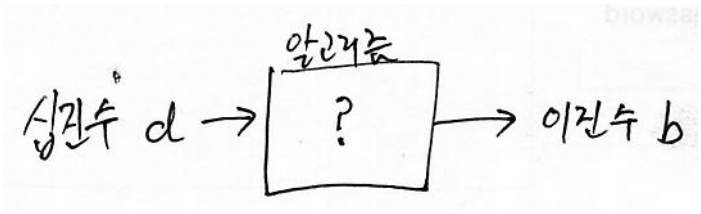
```
My_life {
    While (not dead) {
        get up();
        promotion = company(work);
        go to bed();
    }
}
```

지금까지는 일상적인 예를 통하여 순차적인 작업처리 방법을 표현하는 것에 대해서 알아보았다. 이제 조금 어려운 예를 보도록 해보자.

[이진수로 바꾸는 방법]

우리가 앞선 chapter에서 보았던 문제 중에서 10진수를 2진수로 바꾸는 방법을 "일반화"된 알고리즘으로 만들어 보도록 하자.
먼저, input과 output을 정의 해 놓고, 그 input과 output사이에 알고리즘을 놓아 보도록 하자.

- Input: d 〈〈십진수 숫자를 d라고 해보자〉〉
- Output: b 〈〈알고리즘에 의해 변환된 이진수 값을 b라고 해보자〉〉
- 알고리즘: d2b 〈〈d를 b로 변환시키는 방법을 순차적으로 적은 것이고, 이 알고리즘의 이름을 d2b라고 하자〉〉



먼저, 그냥 한번 가볍게 써본다면

1. d를 2로 나누어서 나머지가 0이면 0을, 1이면 1을 바닥에 쓴다.
2. 바로 위에서 나눈 몫을 다시 2로 나누어서 나머지를 바닥에 써 있는 숫자의 바로 왼쪽에 쓴다.
3. 2를 반복. (언제까지? 더 이상 나누지 못할 때까지)
4. 2의 반복이 멈추면, 바닥에 써있는 숫자를 b라고 하고 output시킨다.

맞는가? 헷갈리지 않는가? 어떻게 하면 보다 명확하게 쓸 수 있을까?

위에서 2번이 반복하는 것을 보다 명확하게 표현하기 위하여, while을 사용하면

1. while (몫이 0이 아니면) {
2. d를 2로 나눈다.
3. 나머지 값을 바닥에 써있는 숫자의 바로 왼쪽에 쓴다.
4. d를 몫 값으로 바꾼다. }

아, 그런데 여기서 바닥에 써있는 숫자가 처음에 지정을 해주지 않아서 혼란을 초래하고 있다! 그러면 어떻게 할까.
조금 더 명확한 표현 방법을 사용하여 보자.

- ✓ 변수(variable)을 써라.
- ✓ Assignment를 써라.

"A=3" 이라는 표현은 A라는 변수에 3을 대입시키라는 것이다. 이 instruction을 하게 되면, 변수 A는 3값을 의미하는 것이다. A=5를 하게 되면, 이제 변수 A는 5를 의미하게 된다. (그래서 이러한 것은 변수라고 부른다!) 임시로 값을 저장하기 위하여 사용하면 된다. 다시 한번 알고리즘을 써보자.

변수를 먼저 정해 놓고 시작하자. 나머지를 변수 r이라고 해보자.

1. d2b(d) { 〈〈decimal to binary〉〉
2. while (d != 0) {
3. r = d/2 의 나머지

```
4.          r을 b에 들어 있는 숫자 바로 왼쪽에 붙인다
5.          d = d/2 의 몫
6.        }
7.     return b;
8.     }
```

- 이 알고리즘의 이름을 d2b라고 했고, 이 알고리즘의 input인 d는 d2b(d)의 괄호 안에 d를 넣어 표시했다. 즉 이 d2b에 d가 input으로 들어간다는 말이다.
- Output인 b는 알고리즘 마지막에 return b; 에서 표시하였다. 즉 b를 return하라는 말이다.
- 기호 "{"과 "}"는 구역을 표시하기 위하여 사용했다. 즉 d2b(d) 이라는 알고리즘은 그 바로 다음의 {부터 시작해서 8번째 줄의 }까지라는 말이며, 2번째 줄의 while에서 반복되는 구역은 2번째 줄의 {부터 시작해서 6번째 줄의 }까지라는 말이다.
- 2번째 줄의 "!="표시는 "not equal", 즉 "not ="을 표시한 것이다.

좀 더 명확해 졌는가? (아직도 멀었다)

알고리즘에서의 절차표현은 아주 정확하게 기술이 되어야 한다. 정확하지 않으면 그 과정에 의하여 input이 output으로 변환된다는 것을 보장하지 못한다. 이진수 변환의 예를 다시 들면, 우리는 10진수 d를 2진수 b로 바꾸는 방법을 이미 알고 있다. 그 공식을 안다. 그러면 d에서 b로 변환하는 과정을 step-by-step으로 정확하게 기술하여 보아라. 그리고 확인(test)하여 보아라.

그대로 따라 하기만 하면 d가 b로 되는가. 당신이 만든 그 step-by-step instructions를 이진법 변환이 무엇인지도 전혀 모르는, 10진이 무엇인지 2진이 무엇인지도 모르는 사람에게 주고, d를 주었을 때, 그 사람이 정확한 b를 당신에게 줄 것인가? 만약에 성공했다면, 그 알고리즘은 "명확한" 표현을 사용하여 "정확하게" 기술된 것이다. 절차를 "명확하게" 표현하는 언어와 그 언어를 사용하여 나의 생각을 정확하게 표현하는 것은 매우 중요한 기술이다. 컴퓨터 프로그래밍을 생각하여 보자. 컴퓨터는 아무것도 모르는, 아무 생각도 없고 지능도 없는 바보와 같은 기계이다. 나는 input 데이터와 알고리즘을 그 바보에게 주어서 내가 원하는 output이 나오도록 하고자 한다. 바보가 이해할 수 있는 용어를 사용하여야 하며 (프로그래밍 언어), 나는 step-by-step instruction을 정확하게 서술하여야 한다. 그 용어/언어는 애매모호성을 완전히 배제한 명확한 것이어야 한다.

6.3 알고리즘(Algorithm)의 명확한 표현, 프로그램

문제해결 방법을 step-by-step instructions로 표현을 하는 것이 알고리즘이라고 했다. 알고리즘을 만들어 봄으로써 자신의 아이디어를 정확하고 효율적으로 표현할 수 있고, 그것을 통하여 자신의 아이디어를 좀 더 정제할 수가 있을 것이다. 결국은 나의 아이디어를, 나의 생각을 바보(컴퓨터)에게 자세하게 가르치는 과정이기 때문이다. (초등학생에게 과외 가르쳐 본 적 있는가?)

우리가 일상에서 사용하는 언어는 매우 문화 맥락적 표현 방법이다. 따라서 같은 언어를 사용하여 상대방에게 이야기 하면 그 상대방은 나의 표현에 담긴 문화 맥락적 지식을 은영 중에 사용하여 그 말을 이해하게 된다. 이것은 말을 하는 사람과 말을 듣는 사람이 같은 문화 맥락적 지식을 공유하고 있을 때에만 가능하다. 우리가 어른에게 이야기 하는 식으로 어떤 방법을 설명하면 초등학생은 전혀 다르게 이해할 수도 있다. 공유하는 문화 맥락적 지식이 많지 않기 때문일 것이다.
이전에 알고리즘을 설명하면서 예를 들었던 요리방법을 다시 생각해보자. 요리 잘하는 우리 어머니가 자취하고 있는 나에게 전화하여 된장찌개 끓여 먹는 방법을 이야기 해주고 있다고 해보자.

" 파를 어슷어슷하게 썰어서, 찌개가 한소끔 끓으면 한 움큼 넣어야 한다."

요리를 자주 하는 사람들이야 이 말을 정확하게 이해하겠지만, 요리를 처음 해보는 내가 이 말을 정확하게 이해 할 수 있을까? "어슷어슷하게"는 도대체 어떻게 하라는 말인지, "한소끔 끓으면"은 무슨 말인지, "한 움큼"은 얼마큼인지… 인간도 이해하기 힘든데, 하물며, 아무런 의식과 사고능력이 없는 컴퓨터라는 기계는 오죽할까. 문화 맥락적인 범주를 벗어나서 누구든지 이해할 수 있게 정확하게 표현하려면 "어슷어슷하게"라는 표현은 아마도 "정확하게 31°의 각도로 0.3cm의 간격으로"라고 표현하여야 할 것이다. 아마도 대형식품공장에서 사용하는 파 써는 기계는 그렇게 프로그램 되어 있을 것이다.
다음 그림은 조금 더 재미있는 예를 보여주고 있다. 자기의 아이디어, 머릿속의 생각을 다른 사람에게 알고리즘으로 표현하여 주었는데, "명확성의 부족" 때문에 제대로 된 output이 나오지 않았다. 그래서 다시 알고리즘을 나름대로 더 명확하게 표현해 주었는데, 여전히 명확하게 전달되지 않았다. 어떻게 하면 명확한, 그리고 정확한 알고리즘을 기술 할 수 있겠는가?

우리가 만든 알고리즘이라는 문제해결 방법을 컴퓨터가 자동으로 처리하게 하려면, 그 알고리즘을 컴퓨터가 이해할 수 있는 (즉, 문화 맥락적 요소가 완전 배제된) 언어로 표현해야 한다. 그것을 컴퓨터 프로그램이라고 한다.

알고리즘과 프로그램과의 관계

앞서 언급하였듯이, 알고리즘은 문제 해결에 대한 인간의 기본적인 생각이고 방법이며 그것이 절차적인 형태로 표현된 것이다. 이것을 실제 기계(컴퓨터)에서 자동 처리되게 하기 위하여서는 그 기계가 그 알고리즘을 인식하여 그 기계 내부에서 처리할 수 있는 방법이 필요하다.

컴퓨터 프로그래밍 언어라는 것은 그러한 목적으로 필요한 것이다. 기계(컴퓨터)는 특정 컴퓨터 프로그래밍 언어로 쓰여진 프로그램을 이해하여 처리할 수 있게 되어 있고, 그럼 인간이 만든 알고리즘을 그 특정 프로그래밍 언어로 바꾸어 주기만 하면 되는데, 그 과정을 "컴퓨터 프로그래밍"이라고 하며 그 일을 하는 사람을 "컴퓨터 프로그래머"라고 부른다.

컴퓨터 프로그래밍 언어는 지금까지 수백 가지가 넘게 소개되었지만, 현재 사용되고 있는 것은 그리 많지는 않다. 컴퓨터에 문외한인 여러분이라도 자바(Java)라든가 C, C++과 같은 이름은 들어 봤으리라고 생각된다.

이러한 프로그래밍 언어는 인간이 쉽게 사용할 수 있도록 만든 것이라서, 사실은 기계가 직접 이해하지는 못한다. 그 내부로 들어가면, 기계는 기계 회로에 박혀 있는 몇 가지 전기신호 처리만 할 수 있는데 이것은 "기계어(machine language)"로 표현된다. 따라서, 인간이 작성한 컴퓨터 프로그래밍 언어는 그 내부에서 컴파일러에 의해서 다시 기계어로 변환되는 과정을 거치게 된다. 아, 걱정하지 말라. 우리는 기계어까지는 알 필요가 없다. 그리고 컴퓨터 프로그래밍도 이 책에서 다루는 부분은 아니다. 다만, 프로그래밍이라는 말을 이해하고, 엄격하게 약속된, 미리 정해진 명령어들만 가지고 알고리즘을 서술 할 수 있어야 한다는 예를 설명하기 위하여 컴퓨터 프로그래밍을 여기서 이야기 하고 있는 것이다.

"컴퓨터 프로그램밍" 이라는 말을 사용했는데, 이것은 프로그래밍이라는 단어는 일반적인 단어라는 말을 내포하고 있다. 즉, 프로그래밍을 컴퓨터에서 한다, 컴퓨터에 집어 넣는다라는 말이므로, 프로그래밍이라는 말은 컴퓨터가 존재하기 이전에도 인간이 일상에서 사용하던 단어이다.

컴퓨터는 미리 정해진 명령어들만 수행하도록 되어 있다. 따라서, 우리 인간은 알고리즘을 그 컴퓨터가 이해할 수 있는 프로그램 언어로 변환하여 주기만 하면 된다. 정확하게 기술하여 주지 않으면 "에러"가 나게 된다. 다음의 간단한 예를 보도록 하자.
아래의 그림은 저자가 2007년에 한국교육학술정보원의 지원을 받아 연구개발한 초등학생용 ICT소양 능력 평가 문제중의 한 문제이다. 초등학교 5-6학년 문제인데 전국 초등학교학생의 31%가 답을 맞추었다. 다시, 본론으로 돌아가서, "문제"가 주어졌고, 단지 "3가지의 명령어"만 주어졌다. 문제해결은 이들 명령어들을 어떠한 순서대로 "순차적으로 나열"하는 것이 된다. 이것이 컴퓨터 프로그래밍의 기본적인 구조이다. 어떠한 프로그래밍 언어를 사용하던지 간에 그 프로그래밍 언어는 한정된 수의 명령어들을 가지고 있고 (아마도 수백 개? 수천 개?), 프로그래머는 그것들을 어떠한 순서로 나열하여 주어진 문제를 해결하는 프로그램을 만드는 것이다.

6.4 문제해결을 위한 창의적 아이디어

이전 단원에서는 내가 문제해결 방법을 가지고 있을 때, 그 방법을 알고리즘으로 표현하는 방법에 대하여 살펴보았다. 즉, 알고리즘은 문제해결에 대한 나의 아이디어이며 생각이다. 그것이 프로그램으로 바뀌어 컴퓨터에서 돌아가게 되는 것이므로, 프로그램 된 컴퓨터는 결국 어떤 사람의 생각과 아이디어가 기계로 구현된 것이라고 할 수 있다. 컴퓨터 소프트웨어는 누군가의 아이디어이며 생각의 산출물이다. 내가 어떠한 컴퓨터 소프트웨어를 사용한 다는 것은 어느 누군가의 생각과 아이디어를 사용하고 있다는 것을 말한다. 그럼 이제 남은 것은, 문제해결에 대한 창의적인 아이디어와 생각은 도대체 어떻게 이끌어 낼 수 있을까이다. 그것은 전적으로 사람의 몫이다. 하지만, 알고리즘을 만들어 내는 과정은 그 아이디어와 생각을 이끌어 낼 수 있게 도와준다. 명확하고 정확하고 효율적인 해결 방법을 만들 수 있도록 도와준다.

문제해결 아이디어를 정리하는 방법 (알고리즘으로 만들기 전에)

우리가 어떤 문제를 바라보고, 그 문제를 해결할 수 있다는 생각이 들면 (혹은 과거에 그 문제를 별 어려움 없이 해결한 경험이 있다면), 우리의 머리 속에는 그 문제 해결 알고리즘이 이미 존재한다는 것이다. 이렇게 우리가 이미 알고 있는 문제해결 방

법이 있, 문제해결에 필요한 input이 무엇인지 분명할 때에는 (우리가 지금까지 본 예와 같이) 큰 어려움이 없었으나, 그렇지 않은 경우들이 우리가 해결해야 하는 것들의 대부분이다.

[대형할인매장에서 계산대 줄 고르는 문제]

대형할인매장에서 어느 라인에 서야지 가장 빨리 계산하고 나갈 수 있을까. 어느 줄에 서는 것이 시간이 가장 짧게 기다릴 수 있을까? 당신은 이미 당신의 방법을 가지고 있다. 아무 생각 없이 무작위(random)로 아무 계산대에 설까? 혹은 어떠한 자신만의 규칙에 의하여? 그 규칙은 효율적인가? 우리의 머리 속에 존재 하는 그 알고리즘에 대하여 생각해보자. 자신의 경험을 기억해 보면, 자신은 분명히 어떠한 규칙, 혹은 알고리즘에 의하여 행동했음을 알 수 있다. 자신의 행동을 결정지은 그 알고리즘을 pseudo-code로 작성해 보자.

먼저, 어떤 것을 input으로 사용할 것인가를 생각해봐야 한다. 그 input을 사용하여 나의 알고리즘이 기술될 것이기 때문이다. Output은 어느 특정한 계산대가 될 것이다. 대략 생각할 수 있는 input으로는 줄의 길이, 카트에 담긴 물건의 양, 줄에 있는 사람의 나이, 성별, 계산대의 계산원 의 일하는 모습, 표정 등이 내가 지금까지 경험적으로 나의 의사결정에서 사용했던 던 요소들이다. 문제해결을 위한 알고리즘에서 input도 내가 정해줄 수가 있구나! 문제해결의 창의성은 input을 정확하게 선택하는 것에서부터 시작된다. Input에 줄의 길이만 사용한 것과, 그 보다 더 많은 것을 사용한 것과는 알고리즘의 모양이나 질에 많은 차이가 생긴다.

[사람의 얼굴을 인식할 수 있는 문제]

우리는 사람의 얼굴을 구별할 수 있다. 그 사람이 안경을 쓰거나 모자를 써도 우리는 구별할 수 있다. 우리의 뇌는 사람의 얼굴을 어떻게 구별을 해 낼 수 있을까? 얼굴의 정보에서 어떠한 요소의 정보를 이용하여 그가 옥주현이 아니라 이효리 임을 인식하여 낼 수 있을까?

당신의 뇌 속에는 이미 얼굴을 구별하는 알고리즘이 들어 있다. 그것을 사용하면 된다. 그런데 문제는 내가 나의 뇌를 모른다는 것이다 (긴- 한숨-).

이런 난감한 경우에는 몇 가지 실험을 해서 나의 뇌를 테스트해 볼 수가 있다. 먼저, 얼굴 정보에서 어떠한 요소의 정보가 우리의 뇌에서는 인식에 사용되는 지가 궁금하다. 먼저, 안경을 쓴 경우에 우리는 그가 누구인지 구별하는가? 그렇다. 만약에 커다란 검은 썬글라스를 쓴 경우에는? 좀 인식을 할 수도 있고 좀 가물가물할 수도 있다. 이것으로 우리는 눈의 정보가 중요 요소임을 알 수 있다. 조금의 noise도 상관 없다. 복면을 하면? 어떠한 정보도 받을 수 없으므로 인식 못한다. 감기 마스크를 쓴 경우에는? 이런 식의 실험을 반복해서 하면 우리는 중요 정보 요소를 정할 수가 있다. 그러면 그 정보 요소를 가지고 어떠한 알고리즘을 만들어 인식에 사용할 것인가?

이 문제 해결에서는, 얼굴의 각 중요 부분을 서로 연결한 그래프 형태로 얼굴정보를 표현하고, 이 그래프에서의 각 부분들간의 비율로서 같은 사람인지 내가 알고 있는 어떤 사람인지를 인식한다.

창의적인 문제해결 아이디어

다행스럽게도, 우리는 우리가 실생활에서 해결하고 있는 많은 문제들에 대한 우리 자신만의 방법을 가지고 있다. 십진수를 이진수로 바꾸는 것, 아침에 이빨을 닦는 것, 고스톱을 잘 쳐서 돈을 따는 것, 한달 용돈을 효율적으로 사용 하는 것 등 이러한 문제들을 우리는 "잘" 해결하고 있다. 어떻게? 잠재적으로 우리의 머리 속에 저장되어 있는 어떠한 "절차적인 지식"에 의하여. 더 똑똑한 사람은 더 효율적으로, 그리고 더 창의적으로 문제를 해결하는 방법을 알고 있을 것이다. 하지만, 무의식적으로, 잠재적으로 사용하고 있는 당신의 그 "절차적 지식"을 종이 위에 구체적으로 적을 수 있는가? 그리고 종이에 적은 그 절차를 다른 사람, 혹은 바보(컴퓨터)에게 주고 그대로 따라 하라고 했을 때, 내가 그 문제를 해결하는 것과 똑같이 그가 문제를 해결하게 할 수 있을 것인가? 나의 방법이 아무리 창의적이라 할 지라도 그것을 "정확하게" 표현하지 못하면 사용할 수 없을지도 모른다.

우리가 일상생활에서 쉽게 접하지 못하는 문제를 만났을 때, 우리는 사람마다 다른 해결방법을 생각하게 될 것이다. 어느 것은 다른 것보다 더 효율적이고 창의적일 것이다. 그것을 구체적으로 절차로서 표현하여 봄으로써 우리는 보다 더 창의적이고 효율적인 방법을 발견해 낼 수가 있다.

[Warming Up!: 가벼운 당구공 고르는 문제]

다음의 문제를 당신에게 주었을 때, 당신은 어떠한 해결방법을 생각해 낼 수 있는가?

"똑같이 생긴 당구공 7개가 있다. 6개는 무게가 같은데, 하나는 무게가 적게 나간다. 당구공을 얼마든지 올릴 수 있는 커다란 접시를 양쪽에 달고 있는 양팔 저울이 있다. 이 저울을 이용하여 무게가 적게 나가는 당구공을 찾아내야 한다. 단 저울을 가장 적게 사용해야 한다. 저울을 몇 번 이용해야 하는가?"

1분 안에 답을 내 보시오.
Input은 무엇인가?
Output은 무엇인가?
Algorithm은?

답은 "두 번"이다.
이 문제를 일반화 시켜 보자. 즉, n개의 당구공이 있고 그 중의 하나만 다른 공에 비하여 가벼운 것이라고 하자. 그러면 저울을 몇 번 사용하면 될 것인가? log_2(n)
log_2 는 2로 나눌 수 있는 횟수를 말하는 것이다. 즉, log_2(4)=2는 4를 2로 두 번 나눌 수 있다는 말임. log(7)=2.*** 이므로 2번이다. log(8)=3이므로 3번이다.
하지만, 여기서는 우리는 전체 당구공을 이등분을 계속 해 나가면서 나머지 반을 없애 나가는 전략을 사용하고 있기 때문에 log_2를 사용하고 있음을 기억하자. 그런데 당구공이 8개일 때를 가만히 생각해 보면, 양팔 저울 각각에 4개씩 놓아서 이등분하는 것 보다, 3개는 왼쪽 팔에, 3개는 오른쪽 팔에 놓고 2개는 바닥에 놓으면, 세 그룹으로 분리하는 것이고, 그 중에 하나의 그룹을 골라서 마찬가지로 한번만 무게를 재면 가장 적은 무게의 당구공을 찾을 수 있음을 알 수 있다. 그러면 당구공이 9개 일 때에도 마찬가지로 두 번 만에 "그" 공을 찾을 수 있을 것이다. 하지만, 10개 일 때에는 삼등분을 하면 3-3-4가 되므로 4는 최소 2번의 비교를 하여야 하므로 3번 만에 "그" 공을 찾을 수 있다. 이렇게 3등분을 하여 그 중에 한 그룹만 골라가는 과정을 반복하여 "그" 공을 찾을 수 있으므로 우리는 log_3을 사용하면 된다. 즉 ceiling(log_3(n))이 "그 횟수"가 된다.

자 여기까지 본 문제들을 다시 정리 하여 보자.
어떠한 문제(세상의 문제)가 주어졌을 때, 그 문제를 추상화 하여 그 문제를 효율적으로 해결할 수 있는 창의적인 아이디어를 만들어 낸다.
- ✓ 그 창의적인 아이디어는 수학적 논리에 의거한 규칙일 수도 있고 (당구공 문제)
- ✓ 단순한 작업을 반복하게 함으로써 문제를 해결할 수도 있다. (Traveling Salesman Problem)

어떤 문제 해결 방법을 수학적 논리를 가진 규칙으로 표현하는 것은 매우 창의적인 것이다. 이미 있는 수학적 규칙을 적용하는 것 이외에도, 문제 해결을 위하여 내가 나름대로 수학적 규칙을 만들어 낼 수 있다. (훌륭!!) 세상의 문제를 관찰하여 내 나름대로 그 세상이 흘러가는 현상을 하나의 규칙으로 만들어 보고, 그 규칙에 의하여 세상이 흘러가는 것을 확인하는 것은 무척이나 흥분되고 또한 재미있는 일이다.

알고리즘에는 그 문제를 해결하는 당신의 창의적인 아이디어가 녹아 들어가 있어야 한다. 어떠한 문제를 해결하는 당신의 창의력이 명쾌하고 정확하게 표현되어 있어야 한다. 따라서, 당신은 다음 두 가지의 능력을 가져야 한다. 문제 해결을 위한 창의적인 그리고 효율적인 방법을 가지는 것, 그리고 그것을 명쾌하고 정확하게 표현하는 방법을 아는 것. 창의성만 있고 표현을 못하는 것은 내공만 있고 외공이 없는 것이며, 창의성은 없고 표현력만 있는 것은 내공은 없고 외공만 있는 것이다.

.끝.