

# Homework #2

## (Introduction to Data Structures)

Due date: Mar 29, 2018

학번: 2014310407

이름: 이 준 혁(JunHyuk Lee)

### 1. Palindrome Checker

#### 1.1. Solution

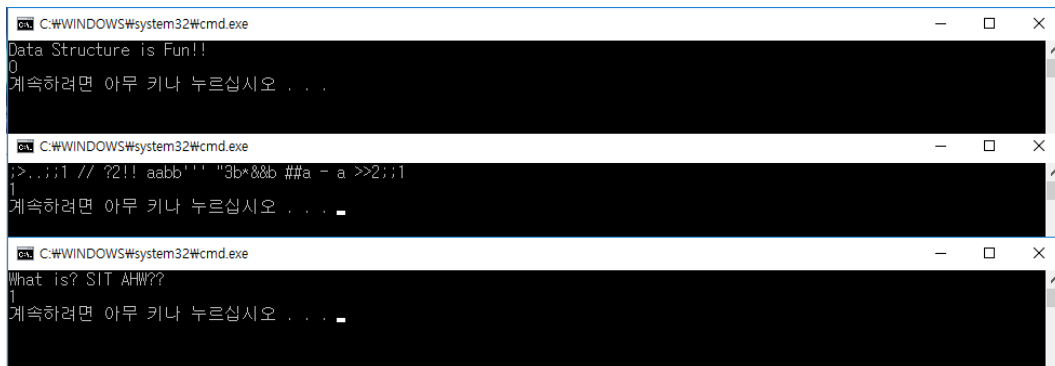
우선 주어진 문자열을 NULL 문자가 나올 때까지 읽어 들이며, 알파벳인 경우 소문자로 바꿔서, 숫자인 경우는 그대로 문자형 배열에 저장했다. 나머지 문자인 경우는 별도의 저장 없이 무시해 버렸다.

그리고 문자 배열의 길이의 절반을 나타내는 변수를 만들어서(half), 배열의 길이가 짝수인 경우는 절반 까지만 비교하면 되므로, 2로 나눠서 저장 했고, 홀수인 경우에는 절반 위치의 수는 무시해도 되므로(즉, 12321에서 3을 무시해도 되는 것처럼), 1을 뺀 값을 2로 나눠서 저장했다.

그리고 half까지 모든 문자 배열의 값을 stack에 넣고, 전체 길이 - half부터 배열의 끝까지의 값과 stack에서 뽑아낸 값들을 비교하며 Palindrome인지 확인했다. 만약 일치하지 않는 값이 있다면 Palindrome이 아니라는 것이므로 0을 반환했고, 0으로 반환되지 않았다면 Palindrome이라는 것이므로 1을 반환했다.

이 때 half + 1이 아닌 전체 길이 - half부터 시작한 이유는, 길이가 홀수인 경우, 즉, 12321같은 경우 1,2가 스택에 저장되는데, Palindrome인지 검사하기 위해서는 3부터가 아닌, 두 번째로 나오는 2부터 확인을 해야 했기 때문이다. 즉, 가장 가운데 값을 무시하기 위해서 였다.

#### 1.2. Result (snapshot)



```
C:\WINDOWS\system32\cmd.exe
Data Structure is Fun!!
0
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
;>...;1 // ?2!! aabb''' "3b*&&b ##a - a >>2;1
1
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
What is? SIT AHW??
1
계속하려면 아무 키나 누르십시오 . . .
```

## 2. Calculating Parentheses

### 2.1. Solution

우선 입력이 보관하는 스택 `sp` 와 이 입력이 괄호인지, 혹은 정수인지를 알려주는 스택 `num` 을 선언했다. `num` 은 스택에 정수를 넣을 때 마다 `TRUE` 값을, 괄호를 넣을 경우 `FALSE` 를 Push 했다. 이 `num` 함수가 필요한 이유는 괄호는 아스키코드 값으로 어떤 정수인데, 이렇게 되면 계산해서 나온 정수 값과의 구분이 불가능 하기 때문이다. 따라서 각 값을 넣을 때 마다 정수이면 `TRUE` 를 넣고, 괄호인 경우는 `FALSE` 를 넣는 방식으로 정수인지 괄호인지를 구분했다.

이후 주어진 문자열을 `NULL` 문자가 나올 때까지 읽어 들이며, 만약 읽은 값이 숫자이면(즉 아스키코드 값으로 '0' ~ '9' 사이인 경우), 그 값을 정수형 스택에 저장했다. 그리고 `num` 에 `TRUE` 를 저장했다.

반면 읽은 값이 왼쪽 괄호라면, 즉 (, [, <라면 괄호를 스택에 쌓았다. 그리고 이는 숫자가 아니기 때문에 `num` 에 `FALSE` 를 저장했다.

읽은 값이 오른쪽 괄호라면, )인 경우는 `CalAdd` 함수를, ]인 경우는 `CalMin` 함수를, >인 경우는 `CalGcd` 함수를 이용해 각각 모든 정수 값은 더한 값, 최소값, 최대공약수를 반환 받아 다시 스택에 Push 했다.

각 함수는 괄호가 제대로 매칭되지 않을 경우 -1 을 반환 시켰고, 이 함수 들에게서 -1 을 반환 받는다면, 잘못된 식이므로 -1 을 최종적으로 반환했다.

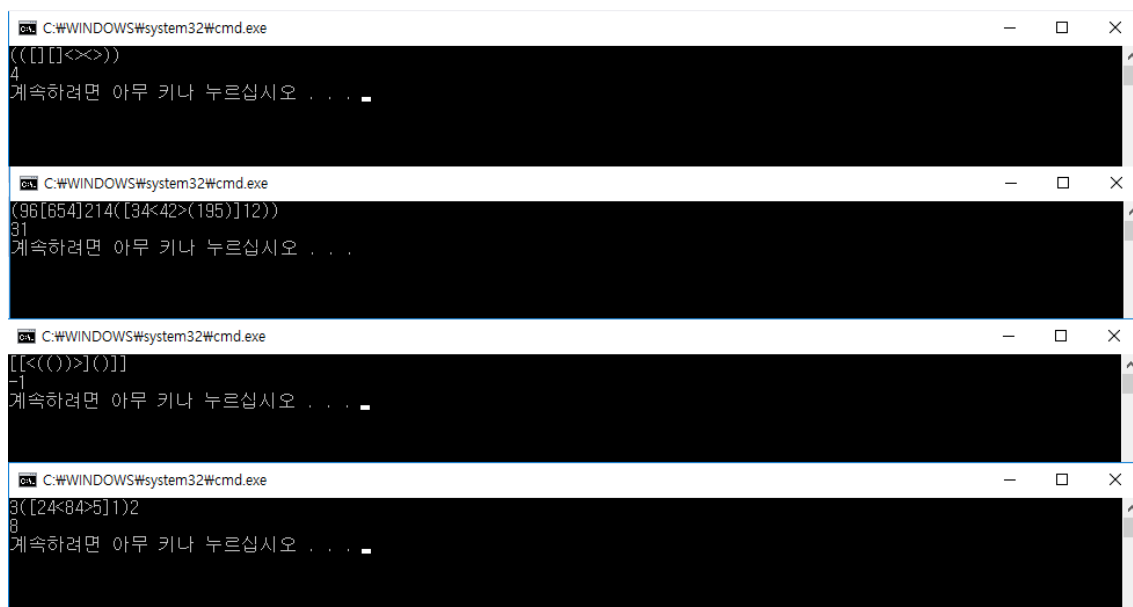
CalAdd 함수는 스택에서 문자인 '('가 나올 때까지 모든 정수 값 들을 더한 후, 그 값을 반환했다. 만약 문자인 '('를 찾지 못한다면, -1 을 반환 시켰다.

CalMin 함수는 스택에서 문자인 '['가 나올 때까지 모든 정수 값 들의 최솟값을 계산 후, 그 값을 반환했다. 만약 문자인 '['를 찾지 못한다면, -1 을 반환 시켰다.

CalGcd 함수는 스택에서 문자인 '<'가 나올 때까지 모든 정수 값 들의 최대 공약수 값을 계산 후, 그 값을 반환했다. 만약 문자인 '<'를 찾지 못한다면, -1 을 반환 시켰다. 세 함수들은 괄호 안에 숫자가 없을 경우, 1 을 반환했다.

이 과정이 다 끝나면 스택에는 괄호가 아닌 정수들만 남아 있을 것이고, 그 수들을 모두 다 더해서 값을 반환했다.

## 2.2. Result (snapshot)



```
C:\WINDOWS\system32\cmd.exe
(([] []<>>))
4
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
(96[654]214([34<42>(195)]12))
31
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
[[<(<())>]()]
-1
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
3([24<84>5]1)2
8
계속하려면 아무 키나 누르십시오 . . .
```

## 3. Finding a Path in Maze

### 3.1. Solution

우선 위, 아래, 좌, 우를 나타내기 위해 Move 라는 구조체를 만들고, 멤버 변수로 가로를 나타내는 hor 변수와 세로를 나타내는 ver 변수를 멤버로 선언했다. 또한 오른쪽으로 가는 행동을 {0,1}과 같은 식으로 나타내서, 위, 아래, 좌, 우를 나타내는 Move 형 배열을 {{ 0,-1},{ 1, 0},{ 0, 1},{ -1, 0 }}으로 선언했다.

그리고 찾았는지 여부를 나타내는 정수형 변수를 FALSE 로 초기화했다. 또한, 길을 방문했는지 여부를 알려 주는 2 차원 배열을 선언하고, 모두 0 으로 초기화했다. (0,0)위치부터 시작하므로 이 값을 1 로 초기화 하고, 스택에 (0,0)을 넣었다. 이 과정에서, 정수 두 개를 넣을 수 없으므로, 스택에 행을 넣고 열을 넣는 방식으로 진행했다.

그리고 found 가 TRUE, 즉 길을 찾거나 혹은 isEmpty 가 TRUE, 즉 스택이 모두 비어 길을 더 이상 찾을 수 없을 때까지 계속 길을 찾았다.

우선 스택에 저장되어 있는 값에 해당하는 행 / 열의 위치로 이동한 후, 가장 먼저 해당 값이 음수인지 확인했다. 만약 그 값이 음수이면 FindTarget 함수로 같은 음수 값을 가지는 위치를 찾아서, 해당 위치를 방문한 적이 없으면, 그 값을 스택에 넣었다.

FindTarget 함수는 이차원 배열과 위치, 다음 위치를 입력으로 받아서, 위치에 값에 해당되는 블랙홀의 다른 값을 찾아서, 그 값으로 다음 위치를 바꿔주는 함수이다.

그 후에는 모든 Move 의 방향에 대해, index 가 음수가 되거나 최대 행 / 열을 초과하지 않는 선에서, 네 방향으로 움직여서, 움직였을 때 미로의 출구라면, 길 발견 여부를 TRUE 로 바꿨다. 아닌 경우에는 해당 위치가 0 이하이고(갈 수 있는 길), 방문한 적이 없다면 방문했다고 표시 후 해당 위치를 스택에 넣었다.

이 과정을 반복했을 때, 길 발견 여부를 나타내는 변수가 TRUE 라면 길이 있는 것이므로 1 을 반환, 아닌 경우 길이 없는 것이므로 0 을 반환했다.

### 3.2. Result (snapshot)

```
C:\WINDOWS\system32\cmd.exe
7 7
0 -1 1 1 0 0 0
1 1 1 1 0 0 -1
1 0 0 0 0 0 1
1 1 1 0 0 1 1
1 0 0 0 1 1 1
1 1 0 -2 1 1 1
1 1 1 1 -2 0
1
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
6 6
0 0 0 0 0 -1
1 1 1 1 1 1
0 0 0 0 0 0
-2 1 1 1 1 0
0 1 -2 0 1 1
0 1 1 -1 1 0
0
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
5 5
0 0 1 1 1
1 0 0 0 0
0 0 1 1 1
0 1 1 1 0
0 0 0 0 0
1
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
10 10
0 -1 0 0 0 0 1 1 1 0
1 1 1 1 1 1 0 0 0 1
1 0 1 1 1 0 0 0 0 1
1 0 1 -2 1 0 1 1 0 1
0 0 1 1 1 0 0 1 1 1
0 0 0 0 -1 0 1 0 0 1
1 1 0 1 1 1 0 1 -3 1
0 0 0 0 1 1 1 1 1 1
0 1 1 1 0 0 -2 1 0 0
0 -3 1 1 1 1 0 0 0 0
0
계속하려면 아무 키나 누르십시오 . . .
```

## 4. Funny Calculator

### 4.1. Solution

우선 문자열을 postfix 로 바꾸어 저장할 문자 배열과, 숫자인지 연산자인지의 구별을 위해 필요한 정수 배열을 선언했다. 그리고 while 문과 getchar 함수를 통해, NULL 문자일 때 까지 입력을 받았다. 그리고 두 자릿수 이상의 입력을 받기 위해, 숫자를 입력으로 받은 경우에는 변수를 하나 설정해서 이전에 입력 받은 값 \* 10 + 현재 입력 받은 값을 더해서 저장했다.

그리고 문자를 입력으로 받으면, 숫자가 끊긴 것이므로 이 변수를 0 으로 초기화 시킨 후에, 해당되는 숫자를 문자형 배열에 저장했다. 이 때 아스키코드 값이 아닌, 그 자체의 정수 값으로 저장 했다. 즉, '2'를 아스키코드 값이 아닌 2 자체로 저장한 것인데, 이렇게 되면 연산자와 구분이

불가능해 지므로, 숫자를 하나 저장할 때 마다 숫자 여부를 판단하는 배열의 같은 index 에 TRUE 값을 저장했다.

그리고 연산자 우선순위를 위해 `op_priority` 라는 함수를 구현했는데, 이 함수는 괄호인 경우 가장 작은 값인 1 을, 그 외의 연산자인 경우 우선순위에 따라 높은 값을 반환하고, 주어진 연산자가 아닌 경우에만 0 을 반환했다. 따라서, 이 값이 0 인 경우에는, 아무 행위도 하지 않고 다음 위치를 탐색했다. 괄호를 가장 낮은 우선 순위를 준 이유는, 괄호는 별도로 취급 되기도 할 뿐더러, 다른 연산자에 의해 스택에서 나가면 안 되므로 스택에 머물러 있을 수 있도록 가장 낮은 우선 순위를 가지게 했다.

만약 입력이 왼쪽 괄호인 경우, 스택에 괄호를 넣었고, 입력이 오른쪽 괄호인 경우에는 왼쪽 괄호가 나올 때까지 모든 연산자들을 다 문자형 배열에 넣었다. 이 때, 이 값들은 다 연산자이므로, 숫자 여부 판단 배열의 값은 매 번 FALSE 로 저장했다. 이렇게 왼쪽 괄호까지의 모든 연산자를 다 제거했다.

그리고 괄호가 아닌 기타 연산자의 경우에는 만약 스택에 있는 연산자가 입력으로 받은 연산자보다 우선 순위가 높다면, 모두 다 뽑아내서 postfix 문자 배열에 넣었다. 만약 이 과정에서 스택이 모두 비어 버린다면 그만 하도록 했다. 이렇게 입력보다 우선 순위가 높은 연산자들을 다 뽑아낸 후에는, 입력 받은 연산자를 스택에 넣었다.

이 과정을 모두 다 거친 후, 만약 숫자가 남아 있다면 그 값을 문자 배열에 넣었고, 또한 남은 연산자가 있다면, 그 연산자들을 모두 문자 배열에 넣었다. 이 과정을 거치면 문자 배열은 기존 Infix 표기법이 Postfix 로 바뀔 것이고, 이 값을 다시 입력 받은 문자열에 넣었다.

그리고 postfix 로 바뀐 문자 배열을 스택을 이용해 계산했다. 문자열의 앞에서부터 읽으면서, 이 값이 숫자이면, (숫자 여부 판별 배열 이용) 스택에 넣고, 연산자인 경우에는 수를 뽑아낸 후 값을 연산해서 다시 넣는 방식으로 계산했다.

이때 |연산자는  $a|b$  를 계산할 때,  $b$  의 자릿수를 구한 후, 그 수만큼  $a$  에 10 을 곱한 후,  $b$  를 더해서 그 값을 반환하는 함수를 이용해서 계산했다.

~연산자는, 역시 함수를 만들어,  $\sim a$  를 계산할 때 0 혹은 1 인 경우는 소수가 아니라고 반환하고, 2 인 경우에는 소수라고 반환했다. 나머지 경우에는 2 이상  $\text{ceil}(\text{sqrt}(a))$  이하의 모든 수 들에 대해, 이 수들 중 단 하나라도 나뉘진다면 소수가 아니라고 반환하고, 나뉘지지 않는다면 소수라고 반환했다.

이 과정을 모두 끝내면 스택에는 최종 계산 결과값이 남아 있을 것이고, 그 값을 반환했다.

## 4.2. Result (snapshot)



```
C:\WINDOWS\system32\cmd.exe
99-2*(2|23%6)
49
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
(52%7+-9)*7-2|3*(7%4)
6
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
(5*(24-3*(16+3)%7+8)*2)|~7|(21/4+6%2)
31015
계속하려면 아무 키나 누르십시오 . . .
```

## 5. Counting SOS Signal

### 5.1. Solution

우선  $N$  개의 건물이 있을 때, 첫 번째 건물부터  $N-1$  번째 건물까지, 자신의 오른쪽에서 자신보다 큰 건물을 찾을 때까지 반복하며, 만약 큰 건물을 찾는다면, 그 건물에는 신호를 보낼 수 있으므로 `count` 를 하나 올리고 해당 번째 건물의 오른쪽을 더 이상 탐색하지 않는다. (더 큰 건물을 찾았다면 그 뒤의 건물들에게 신호를 보낼 수 없으므로)

그 전까지는, 매 건물마다 스택에 지금까지 관찰된 건물의 높이 중 가장 큰 높이를 매 번 저장한다.(단, 이 값은 기준으로 잡은 건물의 높이보다 작거나 같다.) 그리고 높이가 저장될 때 마다, 지금까지의 살펴본 건물보다 높이가 더 크거나 같다는 의미이므로, 이 건물에 신호를 보낼 수 있음을 의미하고, 이 경우에도 역시 `count` 를 1 증가시킨다. 또한 매 번째 건물에 대한 탐색이 끝나면, 스택을 초기화 시켰다.

이 과정을  $N-1$  번째 건물까지 반복하고 나면 중복 없이 모든 가능한 건물 사이를 확인한 것이므로, `count` 는 총 보낼 수 있는 신호의 수가 된다. 이 값을 반환한다.

## 5.2. Result (snapshot)

