이진 트리 라인 계산기

라경준, 이덕우
계명대학교 컴퓨터공학부(컴퓨터공학전공)
gj3447@gmail.com, dwoolee@kmu.ac.kr

Binary Tree Line Calculator

Gyungjun Ra, Deokwoo Lee
Department of Computer Engineering, Keimyung University

요약

최근 Kotlin, Swift 와 같은 목표 기능에대해 최적화된 다양한 고급 컴퓨터 언어들이 출범함에 따라, 이에대한 중요도와 이해력이 필요하다. 그에따라 고급 컴퓨터 언어 제작의 프로토타입 제작에 의의를 두고, 전세계 사람들에게 통용되는 사칙연산에 기반한 수학기호 코드 언어를 해석하는 프로그램을 구현한다. 단순한 계산기라기보다는. 수학기호 문자를 컴파일하여 프로세스로 변환하는 컴파일러로 접근하고. 특성파악, 구상, 구현 과정을거쳐 본 프로그램을 제작하고 고급 컴퓨터 언어에 대한 저자의 이해를 서술한다.

1. 서론

본 논문은 최근 기능에 따라 세분화된 Kotline, Swift 등등의 고급 컴퓨터언어(high-level programming language)들이 많이 출범하고 있음에 따라, 해당언어의 규칙을 공부함에 앞서, 고급 컴퓨터 언어 와, 고급언어의 코드가 기계어로 프로세스되는 과정을 연구를 진행한다.

고급 컴퓨터언어(high-level programming language)란 쉽게말해, 인간이 쓰기쉬운 문자를 일정한 규칙에맞춰 적은뒤, 컴퓨터가 동작하도록 변환 알고리즘을 적용해 기계어로 변환후 프로세스를 진행하는 프로그램이다.

현재 활발히 상용되는 고급 컴퓨터언어들은, 각자 다른 개성을가진 문법과 성능, 기능의 세분화들로 우열을 따질수없이 각자의 언어마다의 특성을 가지고, 각자의 분야에서 다양하게 활용되는 중 이다.

여기서 저자는 실험실에서 연구원의 심심풀이로 탄생한 Python 이라는 고급 컴퓨터 언어가, 특유의 간편함으로 4차산업시대를 문턱에 서있는 현재에 머신러닝에 관한 부분에서 아주 중요한 주축을 담당하는것을 보고, 주어진 프로그래밍 툴을 잘 활용하는것도 좋지만, 그 기반이되는 기술들을 잘 이해하여, 앞으로 펼쳐질 미래에 대응하는것이 좋을것이라 생각하고,

고급 컴퓨터 언어 프로그램 제작에 대한 이해를 위해, 간단한 프로토 타입으로써, 전세계 사람들에게 통용되는 사칙연산에 기반한 수학기호 코드를, 그 주어진 규칙에따라 수행 되어 컴퓨터가 프로세스 할수 있도록 변환되고, 변환된 값이, 특정 알고리즘을 통해 프로세스 후 값을 도출하는 일종의 컴파일러를 개발한다.

2. 본론

일단 본격적인 구상에앞서, 우리들이 사칙연산 수학코드를 해석하는 특성들을 먼저 살펴보니 위와같은 4개의 특성이 있었다.

특성파악

특성 #1 괄호와 마이너스를 제외하고, 코드를 분석 해보니, 숫자와 연산자가 번갈아가며 코드를 구성하고 양쪽의 맨끝 에는 숫자가 위치하게 된다.

특성 #2 연산자는 앞과 뒤, 총 두개의 숫자, 또는 연산자로 연산된 값을 가지고, 자신의 연산기호로 연산하게된다.

특성 #3 연산자의 실행순서는 ^ (power operation) 연산자가 가장 먼저수행되고, 그다음으로 * (multiplycation operation), / (division operation) 연산자 들이 수행된 뒤 마지막으로 + (plus operation) , - (minus operation) 연산자가 실행된다.

특성 #4 괄호가 있을경우, 가장 안쪽 괄호에있는 연산코드부터 실행된다.

위의 특성들을 가진 수학 코드를 해석 하기위해서 다음 과 같은 알고리즘이 필요하다.

구상

특성 #1 숫자와 연산자는 무조건 번갈아가면서 나온다는것을 착안하여, 굳이 두개를 다른 객체로 생각해 연산하지않고 숫자+연산자 라는 단위로 묶어 프로세스한다, 그렇게 구상하니 프로세스되는 라인이 하나로 압축되었다.

특성 #2 연산자는 무조건 두개의 값을 가진다는것에 이진트리를 착안하여, 그림 #1과 같은 연산기호를 값으로 가지고, 두개의 자식노드를 가진 노드를 구상하고, 말단노드는 전부 숫자로 이루어져있고, 연산순서에 맞춰 노드연결 구조화 된 이진트리를 제작하고, 루트노드를 반환하여 깊이 우선 탐색(DFS)으로 루트노드의 결과값을 연산하여 최종 계산값을

구한다.

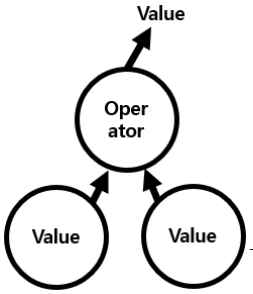

그림 #1

특성#3 먼저 연산되는 연산자들의 우열순위를 연산자 우선순위라는 값으로 표현하여

^ 연산자의 연산자 우선순위 = 2

* , / 연산자의 연산자 우선순위 = 1

+ , - 연산자의 연산자 우선순위 = 0

으로 정의한 후, 앞의 연산자가 뒤의 연산자보다 같거나 클 경우, 앞의 연산자가 먼저 실행 되고, (뒤의 연산자 노드가 앞의 연산자 노드를 자식노드로 가지는 구조) 앞의연산자가 뒤의 연산자보다 클 경우, 앞의 연산자의 연산을 보류하고, 탐색한후 연산자 우선순위가 높연산을 실행하도록, (앞의 연산자 노드가 뒤의 연산자 노드를 자식노드로 가지는 구조) 구상한다.

특성#4 를 구상하기위해서, 3번의 구상변경이 있었는데, 각 구상방법의 장점과 단점을 서술하고 최종적으로 구현할 방법을 채택한다.

1. 노드 스택 리스트

그림 #2와 같은 구조로 코드를 괄호단위로 분해해 괄호안의 코드들을 각각 다른 객체의 작은 코드로 보고, 코드를 값으로 치환하고 계산한다고 생각하여 연산자가 보류되고 처리되는 노드 스택을 배열로만들어, 괄호가 열릴때 배열을 추가하고, 괄호가 닫힐때 해당 배열 스택의 코드 계산이 전부 끝났다고 생각하고 계산하여 노드구조를 짠 뒤 루트노드를 리스트의 그 전 스택의 값 노드로 반환한다.

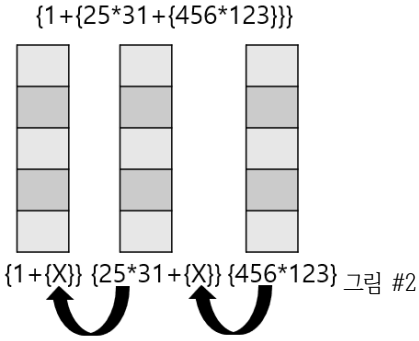

그림 #2

단점으로, 여러개의 리스트로 구성되어있기 때문에 괄호로 인한 오류처리를 하기가 매우 복잡하다는 점이 있다.

2. 특수 괄호 노드

노드 스택 리스트 방법을 보완하여 생각한것으로, 하나의 스택에서 노드 프로세스를 처리를 하면서 괄호가 열릴때 (본래 스택 리스트에 스택이 추가될때) 괄호 노드 라는 특수 노드를 스택에 추가하게 되고, 스택의 프로세스에서 이 특수 노드는 스택의 맨 마지막 층인것처럼 프로세싱한다. 괄호가 닫힐때, 특수 노드가 나올때까지 노드스택을 코드가 끝난것처럼 프로세스하고, 특수 노드가 나오면 해당 특수노드를 제거한후 계속 코드를 프로세스한다.

장점은 노드 스택리스트의 여러개의 스택이 있어서 오류처리하기가 까다롭다는 단점을 제거하였다는 점이다.

단점은 굳이 추가했어야하는 특수노드를 오로지 괄호연산을위해서만 사용하여 노드스택의 메모리와 연산량을 증가 한다는 점이다.

3. 유동 연산자 우선순위

앞서 연산자 우선순위를 0,1,2 으로 매겼다고했 는데, 각 노드에 bracket 이라는 변수를 추가하고, 코드 프로세싱이 되는동안, 전역변수 bracket 을 괄호가 열릴때 1을 추가하고 괄호가 닫힐때 1을 뺀다. 그후 노드를 생성할때 bracket에 전역변수 brcket을 대입해 연산자 우선순위를 생각할때 해당 노드의 연산자 우선순위+bracket*3 으로 생각하여 이진트리 구성한다.

장점은, 구현이 용이하고, 오류가 거의 나지않는다. 심지어 사칙연산 문법을 틀리더라도 따로 문법오류 구문을 넣지않는 이상 자기 나름대로의 프로세스를 진행한다. 단점은 기존의 연산자 우선순위 값을 저장하고있는 Dictionary 구조를 활용하는것과 다르게 각 노드가 유동적인 연산자 우선순위를 알고있어야 하고 그에 따라 노드의 변수가 하나 더 추가된 만큼 메모리의 사용량이 증가된다.

위의 3가지 방법중에 유동 연산자 우선순위 방법이 구현이 용이하다는 장점때문에 채택하고 구현을 진행한다.

구현

파이썬을 이용하여 구현하였다.

console창에 사칙연산 수학코드가 입력값으로 들어오면, 한글자씩 읽어가며, 기존에 입력해둔 0~9 와 연산자 기호들로 해당글자가 문자인지 글자인지 판별 후, 글자의 속성이 달라지면 해당 글자까지 입력된 값을 노드로 변환하여 내보낸다.

제일 먼저 나온 숫자노드는 따로처리하여 루트노드로 가진후, 다음의 프로세스를 진행한다.

숫자와 연산자 두개의 노드가 내보내지면 한개의 단위로 생각하여 노드 스택을 이용하여 이진트리를 구성하게되는데, 그림 #3과 같이 연산자와 숫자 단위가 들어올때,

스택이 비어있을 경우, 또는 들어온(INPUT) 연산자와 연산자 스택에 가장 위의(TOP) 연산자와 비교하여 들어온 (INPUT) 연산자가 우선순위가 더 높을경우 그대로 스택에 쌓인다.

작거나 같을 경우 에는 그림 #4과 같이 스택의 가장 위의(TOP) 연산자가 루트노드를 자식노드로 가지고 빠져나가게(POP) 된다.

(연산자 우선순위의 원판크기를 가진 역 하노이탑이라고 생각하면 이해가 쉽다.)

그 후 비교연산을 실시하며 들어온 (INPUT) 연산자가 쌓일때까지 반복한다. 그리고 코드가 종료되면, 스택에 있는 노드들을 들어온순서대로 내보내면서(POP) 노드스택이 빌때까지 반복한다.

그 후의 루트노드를 깊이 우선 탐색으로 (DFS) 노드를 탐색한후, 양 자식노드를 자신이 가지고있는 연산자로 연산한후 반환하는 형태로 연산하여, 최종 루트노드의 결과값을 반환한다.

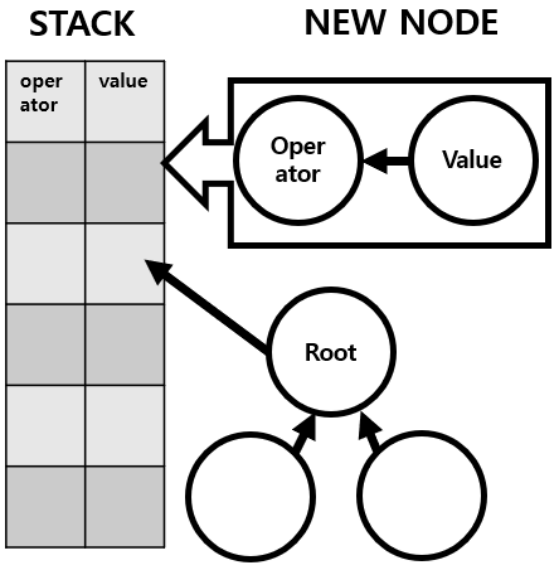

그림 #3

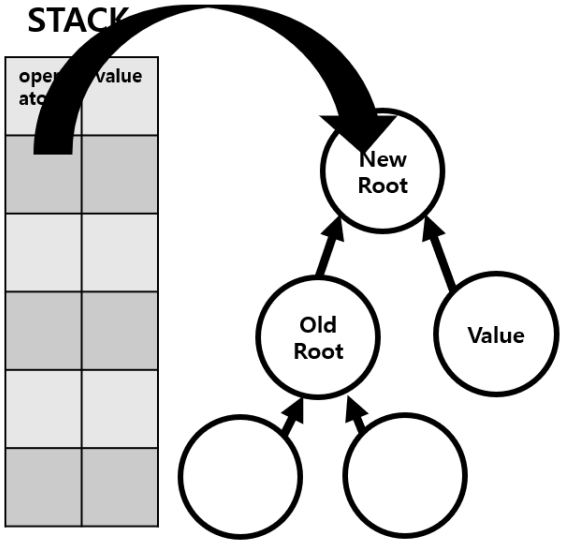

그림 #4

4.결론

사칙연산이라는 단순한 규칙을가진 수학코드를 컴파일하는 프로그램을 제작한 결과,

고급 프로그래밍 언어를 구현하기 위해서 스택과 자료구조를 잘 활용해야한다는 사실을 확인하고,

이미 존재하는 규칙임에도, 생각보다 규칙의 특징을 찾아내어 프로세스 를 구상하는과정이 복잡하다는걸 이해하고, 직접 고급 프로그래밍 언어를 제작하기위해선 이 과정이 잘 수행될수있도록, 추가적인 연구가 필요해 보인다.

5.참고문헌

[1] https://www.youtube.com/watch?v=rt_428Ub6M4

[2] https://github.com/gj3447/line_input_calcurator.git

감사의 글

본 논문은 교육부와 한국연구재단의 계명대학교 대학 혁신지원사업비를 지원받아 수행된 것입니다.