고급프로그래밍

2019203102
유지성

2020.05.17.

소프트웨어학부

요약

fix every syntactic & logical errors in calculator_buggy.cpp		O
Extend code to handle unary plus as well		O
Pre-declare the “pi” and “e”		O
Add an exponentiation operator	Use a binary ^ Operator to represent “exponentiation”	O
	Make ^ operator bind tighter than * and /	O
	Make ^ operator right-associative	O
Add mathematical functions	sqrt(), sin(), cos(), and tan()	O
	Catch attempts to give invalid arguments, such as negative number for sqrt()	O
Add any other useful features	sec(), csc(), cot()	O

요구조건

Basic

1. fix every syntactic & logical errors in calculator_buggy.cpp

---class Token_stream의 public 영역 함수

void putback (Token t) { buffer = t; full = false; }

->

void putback (Token t) { buffer = t; full = true; }

putback 함수를 호출하면 버퍼에 token이 할당되므로 버퍼가 차있다는 것을 표현하기 위해 full값을 true로 지정

----class Token_steam의 public에 Token get(): 추가

Token Token_stream::get()함수를 사용하기 위해 추가

---Token get() 함수 -> Token Token_stream::get()

Token stream 내부에 있는 get 함수를 사용할 것이기에 수정

---Token Token_stream::get() 함수
switch문의 case에 'Q'추가

```
switch (ch) {
case '(':
case ')':
case '+':
case '-':
case '*':
case '/':
case '%':
case ';':
case '=':
    return Token(ch);
```

->

```
switch (ch) {
case '(':
case ')':
case '+':
case '-':
case '*':
case '/':
case '%':
case ';':
case '=':
case '^':
case 'Q':
    return Token(ch);
```

토큰이 Q일 때 프로그램을 종료할 것이므로 Q값을 추가

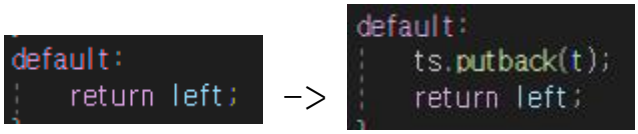
---double primary() 함수의 switch문 case '(' 일 때 d값을 return하도록 수정

d는 괄호 안에 있는 첫 번째 토큰. 이를 사용하기 위해 d값을 return

---double primary() 함수의 switch문 case name의 return값
return get_value(t.value); -> return get_value(t.name);

case name은 값을 얻어오기 위한 조건이 아니라 name을 얻어오기 위한 조건이므로 수정

---double term()함수의 switch문 default에 ts.putback(t) 추가



default일 경우 토큰이 사용되지 않으므로 반환하도록 수정

---double expression()함수의 switch문 default에 ts.putback(t) 추가 (위와 같음)

---void calculator()함수의 t.kind값이 quit일 때 return되는 것을 break되는 것으로 수정

종료를 위해선 무한루프인 while문을 빠져나와야 하므로 break로 수정

2. Extend code to handle unary plus as well

double primary()함수에 switch문에서 case'+'일때를 추가하였다. case '+'일 때 primary()함수를 호출하여 '+'다음 값을 받아오고 받아온 primary값 앞에 +기호를 붙여 반환하도록 하였다.

3. Pre-declare the “pi” and “e”

```
if (s == "e") return Token(number, E);
if (s == "pi")return Token(number, PI);
```

우선 e값을 #define을 이용해 E로 정의하였고, pi값도 마찬가지로 PI로 정의하였다.

Token Token_stream::get() 함수에서 문자열을 받아들이는 부분인 switch문의 default 부분에 입력받은 문자가 “e”일 때와 “pi”일 때의 리턴값을 설정해줬다. 리턴값에 number를 넣어 e와 pi의 kind가 숫자(number)라는 것을 선언하고 value값에 각각 해당하는 수를 넣어주도록 하였다.

또한 let으로 e와 pi를 선언하지 못하도록 declaration 함수에 error를 설정해놓았다.

Advanced

exponentiation operator

*와 /보다 먼저 수행되어야 하기에 term과 primary 사이에 새로운 문법을 하나 만들었다. term은 *와 /연산자가 있는 문법이므로 새로운 문법은 term보다는 먼저 계산하고, primary는 음수, 양수를 표시하는 기호와 숫자, 이름이 포함되어 새로운 문법보다 먼저 적용되어야 하므로 새로운 문법은 term과 primary 사이에 위치하도록 하였다.

```
double zegob()
{
```

새 문법의 이름은 zegob으로 하였다.

거듭제곱 연산을 하기 위해선 최소 하나의 연산자와 두 개의 피연산자가 필요하다. 첫 번째 피연산자를 알아내기 위해 primary()함수를 호출한다. primary()함수로 알아낸 피연산자 값은 double 타입의 변수 left에 저장해놓는다. primary()로 left에 값을 저장한 이후부터는 while문 무한 루프를 이용해 뒤에 나올 계산식들을 감싼다. while로 감싸는 이유는 후에 설명하겠다.
그 다음, 연산자를 알아내기 위해 ts.get()함수를 이용한다. ts.get()으로 토큰은 Token 변수인 t에 저장된다. 이때, ts.get으로 받아온 토큰이 어떤 것일지 예측할 수 없다.

그래서, t에 저장된 kind에 따라 수행하는 작업이 달라져야 한다. 지금 만들고 있는 문법(함수)은 거듭제곱 연산을 위한 것이므로

switch 문을 이용해 거듭제곱 연산자(^)인 경우와 그렇지 않은 경우로 나눈다.

t.kind가 거듭제곱 연산자가 아닐 경우, ts.get으로 받아온 토큰을 ts.putback(t)로 반환한다. 그리고 left값을 return해 term에서 사용할 수 있도록 한다.
t.kind가 거듭제곱 연산자일 경우, primary()함수를 이용해 두 번째 피연산자를 불러온다. 불러온 값은 double타입의 변수 d에 저장해놓는다. 여기서 거듭제곱 연산의 형태를 다시 생각해봐야 한다. 3^4와 같이 단순히 연산자 하나와 피연산자 두 개로 끝나는 형태도 있지만 2^3^4^5와 같이 여러 연산자와 피연산자로 이루어진 형태도 있다. 여러 연산자와 피연산자로 이루어진 형태의 경우 계산은 뒤쪽부터 해야한다. 그래서 피연산자들을 저장해놓을 벡터 c를 선언했다. 이제 여기서 다시 두가지 케이스가 생긴다. 두 번째 피연산자 다음 토큰이 거듭제곱 연산자인 경우와 아닌 경우. 우선 두 번째 피연산자 다음에 나오는 토큰이 어떤 것인지 알기위해 ts.get()으로 토큰을 받아오고 t값에 저장한다. 그 후 if문으로 t에 저장된 kind가 거듭제곱 연산자인 경우와 아닌 경우로 나누어 작업을 수행한다.

거듭제곱 연산자가 아닐 경우, 두 번째 연산자를 확인하기 위해 받아온 토큰 t를 ts.putback(t)를 이용해 반환한다. 그리고 첫 번째 피연산자인 left와 두 번째 피연산자인 d를 pow()를 이용해 계산한다. pow()를 이용해 계산한 값을 left에 저장하고 break로 switch문을 탈출한다. 여기서 left에 pow값을 저장하고 탈출하는 이유는 이렇다. switch 조건문을 탈출한 후, 두 번째 피연산자 다음 토큰은 거듭제곱 연산자가 아니기 때문에 while문으로 인해서 다시

switch 문으로 들어와 default 로 가게 된다. default로 가면 left를 반환하기에 pow값을 left에 저장하고 탈출한 것이다.
거듭제곱 연산자일 경우, 우선 벡터c에 두 피연산자 left와 d를 순서대로 저장한다.

```
c.push_back(left);
c.push_back(d);
```

c에 두 피연산자 값 저장

그 후 while문으로 t.kind가 거듭제곱 연산자 '^' 일 동안 다음을 반복한다. ts.get으로 다음 피연산자를 받아와 t에 저장한다. 그 다음 t.value를 c값에 넣는다. 넣은 후에 다시 ts.get으로 다음 토큰을 받아온다. 이 작업을 반복한다. 만약 c값에 t.value를 넣은 후 받아오는 토큰이 거듭제곱 연산자 '^'이 아닐 경우에는 while문이 종료되고 ts.putback(t)를 이용해 토큰을 다시 반환한다.

```
while (t.kind == '^')
{
    t = ts.get();
    c.push_back(t.value);
    t = ts.get();
}
ts.putback(t);
```

반복 후 토큰값을 반환

벡터 c에 피연산자들이 다 저장되면 저장된 값들의 뒤부터 계산을 하면 된다. 우선 double형 변수 temp에 벡터 c의 마지막 값을 저장한다. 그 다음 for문을 이용해 벡터 c의 처음 피연산자까지 계산한다.

```
double temp = c[c.size() - 1];
for (int i = (c.size() - 1); i > 0; i--)
{
    temp = pow(c[i - 1], temp);
}
```

c에 저장된 피연산자들을 계산

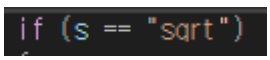
temp에 저장되어있는 최종 계산 값을 left에 저장하고 left값을 return해서 거듭제곱 연산을 끝낸다.

이렇게 계산이 끝나면 함수 zegob에 계산값이 저장되므로 함수 term의 primary함수들을 전부 zegob함수로 바꿔준다.

Add mathematical functions

Token Token_stream::get() 함수에 문자열을 이용하는 구간이 있어서 이를 사용해 구현하기로 하였다.

switch문의 default에 문자열이 sqrt, sin, cos, tan인 경우를 추가하였다.



sqrt인 경우

문자열이 sqrt인 경우, 함수 sqrt_f()를 호출에 double타입 x에 저장한 후 kind는 number고 value는 x인 Token 타입을 반환한다.
sqrt_f 함수는 primary 함수를 호출에 double타입 변수 x에 저장한다. 그 후 sqrt()를 이용해 sqrt(x)값을 x에 저장한 후 x값을 반환한다.
여기서 primary 함수를 호출한 이유는 문자열 sqrt다음에 있는 문자들을 읽기위해 호출한 것이다. 문자열 sqrt 바로 다음에 있는 문자는 '('이므로 primary로 호출할 경우 ')'가 나올 때 까지 안에 있는 식들을 계산해준다. 만약 primary가 아닌 term이나 expression으로 호출할 경우 ')'까지의 식을 계산하는 것이 아닌 그 뒤에 식까지 계산해버린다. 예를 들어 sqrt(4+5)+7;을 입력하였다고 하면 primary는 (4+5)만 계산하는지만 다른 함수들은 그 뒤에까지 계산하게 된다.

sqrt는 루트 계산으로 0이나 음수가 올 수 없다. 그래서 sqrt_f 함수에서 x값이 양수가 아닐 경우 에러 메시지를 출력하도록 하였다.

```
if (s == "sqrt")
{
    double x = sqrt_f();
    return Token(number, x);
}
```

```
double sqrt_f()
{
    double x = primary();
    if (x <= 0) error("please insert positive number in sqrt()");
    x = sqrt(x);
    return x;
}
```

sqrt_f()함수와 토큰값을 반환

sin과 cos, tan도 sqrt와 비슷한 논리로 계산한다. 다만 x값이 양수가 아닐 때 에러 메시지가 출력하지 않는다.
예시로 sin을 들자면, sin의 계산 방법은 다음과 같다.

```
if (s == "sin")
{
    double x = sin_f();
    return Token(number, x);
}
```

```
double sin_f()
{
    double x = primary();
    x = sin(x);
    return x;
}
```

tan는 sin과 cos과는 다르게 pi/2, 3*pi/2… 일 때 값이 정의되지 않는다. 이 프로그램에서 정의한 pi값이 정확하지 않아 tan값이 정의되지않는 모든 곳에 표시를 할 수가 없었다. 그래서 tan값이 정해지지 않는 값들중 몇 개만 에러 메시지가 뜨게 설정해놓았다.

Optional

sec, csc, cot

삼각함수들을 구현하는 김에 sec, csc, cot도 구현했다.
sec는 1/cos, csc는 1/sin, cot는 1/tan 이므로 구현해 놓았던 cos, sin, tan에서 약간만 손을 봐주면 되었다.
예로 sec를 들자면, cos_f 함수에서 x값은 cos(x)로 구했다. sec_f 함수에서 x값은 1/cos(x)로 구하면 된다.
sec, csc, cot 도 tan처럼 값이 정의되지 않는 곳들이 있다.
프로그램의 pi 값이 정확하지 않아 모든 곳을 표시할 수 없어 몇몇 값들만 에러 메시지가 뜨게 하였다.

Comment

프로젝트를 진행하면서 가장 어려웠던 것은 sqrt, sin, cos, tan를 구현하는 것이었다. 맨 처음 구상했던 것은 지금의 코드와 비슷하지만 primary로 받아오는 것이 아닌 expression으로 값을 받아오는 것이었다. 이렇게 expression으로 받아오면 위에 상술한 바와 같이 expression이 괄호 다음 토큰들을 전부 먹어서 계산을 해버린다. 그래서 문자열을 받아온 뒤 '(' 토큰을 잡아먹고 안에 있는 식을 계산하는 것을 생각해내었다. 이렇게 하면 여는 괄호와 닫는 괄호의 수가 맞지 않아 마지막 닫는 괄호에서 "')(' expected" 에러가 뜬다. 이를 이용해 코딩을 하려 했으나 이렇게 하려면 코드의 많은 부분을 손봐야 해서 다른 방법을 생각해내기로 하였다. 해답은 생각보다 가까운 곳에 있었다. expression 대신 primary로 불러오면 되는 것이었다. primary로 값을 받아오면 sqrt식 뒤에 어떤 식이 나오던 상관없이 계산이 되었다. 어렵고 복잡하고 고급 기술을 생각할게 아니라 넓게 생각해야 한다는 것을 깨닫게 되었다.

이 프로그램이 돌아가는 방식을 이해하기까지 꽤 많은 시간이 소요되었다. 아직 코딩 경력과 경험이 부족해 내가 생각해낸 알고리즘대로 코딩을 하지 못하는 경우들이 있었다. 그래서 좀 더 쉬운 알고리즘을 생각해내느라 시간이 더 소요되었다. 추가로 구현하고 싶었던 것들도 있었으나 지금 나의 기준에서 복잡하고 어려워 하지 못한것들이 있다. 더 많은 경험을 쌓아 내가 생각한 알고리즘대로 코딩을 할 수 있도록 할 것이다.