KOI - Lab 6

Anton Söderlund, DT501G
2020-01-08

Part A
Trädet är färdigbyggt då *tree* får tokenet *exec*, dvs då vi ger *exec* som input till terminalen, så det är då vi kommer kalla *execute(TreeNode *p)* i *parser.ypp*.

I *treeExec.cpp* finns funktionen som kommer användas i den här uppgiften. *treeExec(TreeNode *p)* är den rekursiva funktionen som kommer behandla noder beroende på dess typ, och så *calc(int type, int op1, int op2)* beräknar resultaten av operationerna.

Funktionen är byggd på en switch sats som hanterar olika tokens, och för beräkningar är det näst intill samma funktion som användes till stacken i lab 3. Här är en lista över hur de olika typerna behandlas.

LIST: Det är ett nytt påstående avslutat med semikolon, och kommer då alltid rekursera ner till dess första argument, och vi behöver göra en kontroll ifall det andra argumentet används (till IF,ELSE,WHILE).

NUM: Det är en värde, och kan inte vara en nod så vi returnerar det värdet direkt.

ID: Det är en symbol i symboltabellen, och kan inte vara en nod så vi returnerar dess värde som också finns i symboltabellen.

Operator: Alla operatorer som tar två operatorer beräknas med *calc*-funktionen. Om det förekommer så rekurserar vi ner mot operanderna och “hämtar dem” eftersom operanderna själva kan vara av typ ID, ett annat uttryck etc.

‘=’: Värdet “hämtas” igen genom rekursion och när rekursionen för det här case:et är klart tilldelas det till symtable[ID].value.

IF: Först måste vi göra en beräkning på villkoret för if-satsen, om det är sant. Om det är intressant rekurserar vi ner i kodblocket som består av nya lines, expressions, osv. Om det inte är sant tittar vi först om else-blocket existerar, och eftersom det är en nod till if-satsens nod så kan vi helt enkelt räkna argumenten och titta på det finns fler än 1. Är argumenten fler än 1 så rekurserar vi ner i det kodblocket efter.

WHILE: Här alternerar de olika rekursionerna beroende på villkoret, om det är sant rekuserar vi ner i kodblocket tills det är klart, och om igen tills vilkoret inte är sant. T.ex.

```
anton@anton:~/Documents/compiler-interpretors/anton_soderlund_dt501a/lab6$ ./infix2postfix
Type your code in c++ and execute your program with the 'exec' command
a = 0;
while (a < 5) { a = a+1;}
exec
```

```
Setting a = 0
Calculating 0 < 5
Calculating 0 + 1
Setting a = 1
Calculating 1 < 5
Calculating 1 + 1
Setting a = 2
Calculating 2 < 5
Calculating 2 + 1
Setting a = 3
Calculating 3 < 5
Calculating 3 + 1
Setting a = 4
Calculating 4 < 5
Calculating 4 + 1
Setting a = 5
Calculating 5 < 5
```

Figur 1, en while loop

Här ser vi tydligt hur vi går mellan varje träd i loopen.

PRINT: Den är enkel, printa det värde som returneras från rekursionen.

READ: Här frågar vi efter ett värde till en variabel som bestäms i koden, och sätter dess värde till det inmatade värdet i symboltabellen.

Här är ett exempel på det hela programmet från lab5,

```
anton@anton:~/Documents/compiler-interpretors/anton_soderlund_dt501a/lab6$ ./infix2postfix
Type your code in c++ and execute your program with the 'exec' command
sum=0;
product = 1;
i=1;
read(n);
while (i < n) {
    if (i % 100 > 0) { } else { print(i); }
    sum = sum + i;
    product = product * i;
    i = i + 1;
}
print(sum);
print(product);
exec
-----------------------------------------------------------------------------------------------------
Show syntax tree? Y/N
n
-----------------------------------------------------------------------------------------------------
        Setting sum = 0
        Setting product = 1
        Setting i = 1
Enter value for variable n
7
1
        Setting sum = 1
        Setting product = 1
        Setting i = 2
2
        Setting sum = 3
        Setting product = 2
        Setting i = 3
3
        Setting sum = 6
        Setting product = 6
        Setting i = 4
4
        Setting sum = 10
        Setting product = 24
        Setting i = 5
5
        Setting sum = 15
        Setting product = 120
        Setting i = 6
6
        Setting sum = 21
        Setting product = 720
        Setting i = 7
21
720
```

Figur 2, ett helt program