SEK_UE06

1.Beispiel

Ansatz

Um dieses Problem zu lösen, muss folgende Grammatik in C++ umgesetzt werden.

```
Expression  = Term { AddOp Term } .
Term        = Factor { MultOp Factor } .
Factor      = [ AddOp ] ( Unsigned | PExpression ) .
PExpression = „(“ Expression „)“ .
AddOp       = „+“ | „–“ .
MultOp      = „*“ | „/“ .
```

Dabei wird der pfc_scanner verwendet. Zunächst müssen Funktionen implementiert werden, welche für jeden Teil der Grammatik erkennen können ob es sich um den Anfang dieses Teils handelt. Um eine Expression zu parsen, wird zunächst ein Term eingelesen. Solange weitere Terme mit einem AddOp verknüpft eingelesen werden, wird die Sequenz weiter geparsed. Ein Term setzt sich aus einem Faktor und einem optionalen, mit einem MultOp verknüpften zweiten Faktor zusammen. Faktoren sind entweder positiv oder negative Zahlen oder positiv oder negative Expressions in Klammern.

Umsetzung

Zur Umsetzung ist in der Solution “UE06_Fallmann” im Projekt “T01” zu finden.

Testcases

```
Test 1:
Testing non existent file
Error:
Error parsing 'Expression'

Test 2:
Testing existing file
Expected Result : -24
Actual Result : -24
Test successfull

Test 3:
Testing a simple expression
Expression: 1+2*3
Expected Result : 7
Actual Result : 7
Test successfull

Test 4:
Testing negative numbers
Expression: -2-3
Expected Result : -5
Actual Result : -5
Test successfull

Test 5:
Testing an expression with parentheses
Expression: (1+2)*3
Expected Result : 9
Actual Result : 9

Test 6:
Testing an multiplication and division
Expression: 1*3/2
Expected Result : 1.5
Actual Result : 1.5
Test successfull

Test 7:
Testing additon and subtraction
Expression: 1+2-3
Expected Result : 0
Actual Result : 0
Test successfull

Test 8:
Testing a invalid Expression
Expression: invalid+3
Error:
Error parsing 'Expression'
```

```
Test 9:
Testing a invalid Term
Expression: 3+invalid
Error:
Error parsing 'Term'
```

```
Test 10:
Testing a invalid Factor
Expression: 3*(-invalid)
Error:
Error parsing 'factor'
```

```
Test 11:
Testing division by zero
Expression: 1/0
Error:
Divison by zero exception
```

2.Beispiel

Ansatz

Um dieses Problem zu lösen, muss folgende Grammatik in C++ umgesetzt werden.

```
Term = Operation Term Term | Number
Operations = "+" | "-" | "/" | "*"
Number = Digit{Digit}
```

Hierbei setzt sich ein Term aus entweder einer Number oder einer Operation und zwei darauf folgenden Termen. Es werden keine Klammern benötigt, da die Reihenfolge durch die Struktur des Ausdruckes angegeben wird. Auch muss darauf geachtet werden, dass die Eingaben durch Abstände getrennt werden müssen, da es sonst zu falschen Interpretationen des Parsers kommen kann.

Umsetzung

Zur Umsetzung ist in der Solution “UE06_Fallmann” im Projekt “T02” zu finden.

Testcases

```
Test 1
Testing non existent file
Error:
Error parsing Term

Test 2
Testing existing file
Expected Result : 14
Actual Result : 14
Test successfull

Test 3
Testing simple expression
Expression: + 1 * 4 5
Expected Result : 21
Actual Result : 21
Test successfull

Test 4
Testing negative number
Expression: - 0 1
Expected Result : -1
Actual Result : -1
Test successfull
```

```
Test 5
Testing muliplication and division
Expression: / 1 * 2 2
Expected Result : 0.25
Actual Result : 0.25
Test successfull

Test 6
Testing addition and subtraction
Expression: + 1 - 3 4
Expected Result : 0
Actual Result : 0
Test successfull

Test 7
Testing invalid expression
Expression: + invalid 3
Error:
Error parsing Term


Test 8
Testing invalid expression
Expression: invalid 3
Error:
Error parsing Term
```

```
Test 9
Testing division by zero exception
Expression: / 1 0
Error:
Division By Zero
```

3.Beispiel

Ansatz

Um dieses Problem zu lösen, muss Beispiel 1 so erweitert werden, sodass auch Variablen anstelle von Nummern verwendet werden können. Dazu wird die Arithmetic Parser Klasse um eine Map erweitert, welche zu den jeweiligen Variablennamen die Werte selbiger speichert. Zusätzlich werden Funktionen benötigt, welche Variablen anlegen, updaten und löschen können.

Umsetzung

Zur Umsetzung ist in der Solution “UE06_Fallmann” im Projekt “T02” zu finden.

Testcases

```
Test 1:
Testing existing variable
Expression: pi*10
Expected Result : 31.4
Actual Result : 31.4
Test successfull

Test 2:
Testing non-existant variable
Error
Error Variable not found

Test 3:
Testing variable update
Expression: pi*10
Before variable update(pi = 3.14):
Result: 31.4
After variable update(pi = 3):
Result: 30

Test 4:
Testing variable update
Expression: pi*10
Before variable update(pi = 3.14):
Result: 31.4
After variable update(pi is no longer defined):
Error
Error Variable not found
```