Metodologies de la programació

Pràctica 3

Cristòfol Daudén
Aleix Mariné
Cristina Izquierdo

Càlcul d'expressions

Solució general

Passos per obtenir una solució final

1. Eliminem els parèntesis redundants
2. Ens assegurem que la expressió no contingui cap error i sigui del tipus **(A op B)**
3. Si és del tipus **A op B op C** la convertim a **A op (B op C)** o **(A op B) op C** depenen de la prioritat de op
4. Si l'expressió es tracta:
    a. d'un valor numèric → el retornem
    b. d'una expressió (A op B) → retornem el càlcul de (operació A) op (operació de B)
5. Es crida a aquesta operació recursivament

Exemple d'execució

opertate(**20/(4-(2*2))** , **12**)
elimina_parentesis
20/(4-(2*2)) 1 = 12
Prioritza_operacions
Op = 20/(4-(2*2)) 1 = 12
opA = 20 1 = 2
operador = /
opB = (4-(2*2)) 1 = 9
operate(**20**, **2**) /
cas directe

operate(**(4-(2*2))** , **9**)
elimina_parentesis
4-(2*2) 1 = 7
opA = 4 1 = 1
operador = -
opB = (2*2) 1 = 5
operate(**4**,**1**) -
cas directe

operate(**(2*2)**, **5**)
elimina_parentesis
2*2 1 = 3
opA = 2 1 = 1
operador = *
opB = 2 1 = 1
operate(**2**,**1**) *
cas directe

operate(**2**, **1**)
cas directe

20 / 0

excepció, divisió per 0

4 - 4

2 * 2

Mètodes per la detecció d’errors

sintax_validation

Valida que el veïnatge de les operacions sigui adequat

α={0, 1, 2,...,length-1} i β ={0, 1, 2,...,number.length-1} llavors, si (op[α] == number[β]) retorna (op[α+1] == (number[β] | (op[α+1] es un nombre) & op[α+1] != '(')))

Fa servir el mètode esSigne

recorre l'expressió buscant un nombre:

- si troba que acaba de llegir un nombre i hi ha un parèntesis: veïnatge incorrecte
- Si troba que acaba de llegir un signe i hi ha un signe: veinatge incorrecte

op_validation

Valida que els símbols siguin correctes

Per a qualsevol α tal que α pren valors de 0 fins length - 1

llavors op[α +1] = següent xifra del nombre | operador & !parèntesis

Busca que l'expressió es composi dels caràcters següents:

char caracters_permesos[18] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '+', '-', '*', '/', '%', '(', ')','^'};

Sinó, retorna fals.

parenthesis_validation

Valida que els parèntesis siguin correctes

Recorre l'expressió buscant els parèntesis, si hi ha un que obre ("(") suma al comptador, si hi ha un que tanca (")") resta.

El comptador ha de quedar a 0 per a que sigui correcte.

Càlcul d’expressions

operate

* PRE = rep una expressio sense errors
* POST = calcula l'expressió rebuda i retorna el seu valor

```
char opA[256], opB[256], operador;  // conte els operands
int length_opA, length_opB, num, divisor;     // longitud de opA, opB, i el nombre si hem rebut una EXPRESSIO que es un NOMBRE

eliminaParentesi(op, length); // eliminem parentesis redundants sobre l'expressio
prioritzaOperacions(op, length);
if ((num = esNumero(op, *length)) != -1) return num; // cas directe, si l'EXPRESSIO rebuda es tracta d'un NOMBRE el retornem
else trobaOperands(op, *length, opA, opB, &length_opA, &length_opB, &operador); // cas no directe, descomposem l'expressio
```

Tenint en compte l'esquema presentat anteriorment, una vegada hem trobat els operands i l'operador, fem un `switch` per fer l'operació segons el signe de l'operador → **recursivitat de pila**

Millores realitzades

A op B op C op D...

La nostra calculadora és capaç de calcular operacions amb més de dos operands. → prioritzaOperacions

Permet aplicar prioritat d'operacions afegint parentesi.

L'avaluador d'expressions en la fase 1 llegeix sempre **de dreta a esquerra**, si no hi ha parèntesis de prioritat. Pel que totes les expressions **A op B op C** s'avaluen com **A op (B op C)**. Sabent això, els casos que poden existir:

1. **A op B op C | A OP B OP C** → Indiferent, de dreta a esquerra la prioritat es la mateixa.
2. **A op B OP C** → segueix la prioritat d'operació implícita en la funció opera.
3. **A OP B op C** → L'únic cas que cal tractar, ja que si no afegim parèntesis explícits l'EXPRESSIÓ s'avaluarà com **A OP (B op C)**, de manera incorrecta. Manera correcta: **(A OP B) op C**

Altres millores

Casos extra

La nostra calculadora no només opera amb sumes, restes, multiplicacions i divisions, sinó que també té en compta la operació del mòdul i les potències.

```
case '%': return operate(opA, &length_opA) % operate(opB, &length_opB); // cas extra, operacio modul
case '^': return pow (operate(opA, &length_opA), operate(opB, &length_opB)); // cas extra, operacio potencia
```