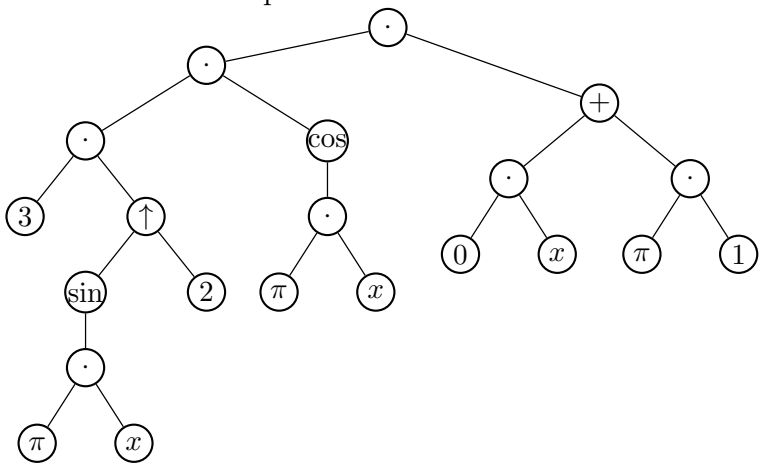
Datastructuren 2022

Programmeeropdracht 2: Expressies

Deadlines. Dinsdag 11 oktober 23:59, resp. dinsdag 1 november 23:59.

Inleiding. Deze opdracht is losjes gebaseerd op *programming assignment* 5.13.2 in het boek van Drozdek. U wordt gevraagd expressies (of eigenlijk expressiebomen) te implementeren, met methoden om expressies in te lezen, af te drukken, te vereenvoudigen, te evalueren en te differentiëren. U levert in twee stappen in. Allereerst alleen het gedeelte voor inlezen en weergeven van de expressie, in tweede instantie de volledige functionaliteit. Lees eerst de hele opdracht.



Deel Eén

1 Expressies

Expressies bestaan hier uit constanten (gehele en decimale getallen, het getal π), eenvoudige variabelen en operaties. We gebruiken de bekende rekenkundige operaties optellen, aftrekken, vermenigvuldigen en delen, evenals sinus en cosinus. Verder is machtsverheffen toegestaan, maar de exponent mag alleen een getal zijn (en niet zelf weer een expressie). Variabelen zijn enkele 'kleine' letters.

Een expressie is dus bijvoorbeeld $\dfrac{x^3(y - \sin(2x + 6))}{-3.7 + 15x} + 5$.

Tips. Omdat een expressieboom maar één soort gegevens opslaat, expressies, zou het *geen* template-klasse moeten zijn.
Een voorbeeld van zo'n knooptype is het volgende:

```
struct Token {
```

```
        enum {
            PLUS, MINUS, NUMBER, VARIABLE // , ...
        } type;
        union {
            char variable;
            double number;
        };
    };
```

Definieer bepaalde helper-functies; op een eigen type zou je bijvoorbeeld een volgende methode kunnen implementeren:

```
bool isBinaryOperator() const;
```

Het is handig als je eigen type geprint kan worden met `cout`, dit kan door het implementeren een methode als:

```
std::ostream &operator<<(std::ostream &s, const Token &tok) {
    s << "TOKEN";
    return s;
}
```

Een onmisbare functie is er een voor het kopiëren van subbomen, het makkelijkst is om dit recursief te doen.
En altijd: na gebruik pointers opruimen!

2 Inlezen

Bij het inlezen van expressies (vanaf het toetsenbord) zullen we gebruik maken van de *prefix notatie* van expressies, soms Poolse notatie genoemd. Dat heeft als voordeel dat de expressie vrij eenvoudig te parsen is. We hoeven immers geen voorrangsregels en haakjes te schrijven. Verder worden spaties toegevoegd om de tekens te scheiden: we willen immers het getal 35 onderscheiden van 3 gevolgd door 5.

Het is gebruikelijk om vermenigvuldigen weg te laten bij standaard infix rekenkundige expressies. Bij prefix notatie moeten alle operatoren expliciet aanwezig zijn. We schrijven `*` en `^` voor vermenigvuldigen en machtsverheffen, en `sin`, `cos` voor de goniometrische functies. De constante π wordt gerepresenteerd door `pi`.

Bij prefix notatie moet de 'ariteit' van de operatoren bekend zijn. Dat betekent dat we niet zowel een unaire als een binaire `-` kunnen hebben. We zullen daarom geen unaire `-` gebruiken. Wél kan de `-` onderdeel van een getal zijn. Dat betekent dat `-35` een legale uitdrukking is, terwijl $-x^3$ moet worden gelezen als $(-1)(x^3)$ en dus geschreven als expressie `* -1 ^ x 3`.

Het genoemde voorbeeld: `+ / * ^ x 3 - y sin + * 2 x 6 + -3.7 * 15 x 5`.

Tips. Lezen van een input stream, bijvoorbeeld `std::stringstream` of `std::istream`, naar een `std::string` leest een enkel woord in.
Voor het omzetten van als string ingelezen getallen kunnen `std::atoi` en `std::atof` gebruikt worden.

3 Weergeven

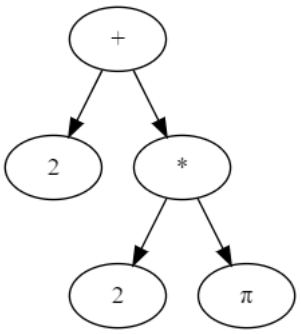
We vragen twee methoden om de boom 'af te drukken'. Allereerst de infix expressie van de boom. Dat wil zeggen, bij binaire operatoren wordt infix gebruikt, en bij unaire operatoren prefix. Het is onvermijdelijk dat er haakjes gebruikt worden. De expressie uit Paragraaf 1 is bijna in juiste vorm: alleen de deelstreep moet nog naast elkaar gezet worden (in plaats van onder elkaar).

De tweede manier is het maken van een grafische reresentatie van de boom, weggeschreven naar een file, in de zogenaamde DOT-notatie[1]. Dit is een taal om grafen te beschrijven. De boom is dan bijvoorbeeld op Linux te bekijken met `dotty` en op Windows met `Graphviz`. Er zijn ook online viewers beschikbaar.

In een expressieboom kunnen twee knopen met dezelfde inhoud voorkomen. Dit heeft consequenties voor de representatie in DOT, waar we de *identiteit* en het *label* van de knopen moeten onderscheiden. De knopen kunnen in willekeurige volgorde staan, maar vóór pijlen waaraan ze gekoppeld zijn. De pijlen kunnen ook als 'paden' gegeven worden, maar dat is niet nodig.

Hieronder een voorbeeld, gemaakt via dreampuf.github.io/GraphvizOnline

```
digraph G {
  1 [label="+"]
  2 [label="2"]
  1 -> 2
  3 [label="*"]
  4 [label="2"]
  3 -> 4
  5 [label="&pi;"]
  1 -> 3 -> 5
}
```



Dit is het einde van het eerste deel, maar het is verstandig om het vervolg door te nemen, en lees in ieder geval de instructies op de laatste bladzijden.

[1] `http://en.wikipedia.org/wiki/DOT_(graph_description_language)`

Deel Twee

4 Vereenvoudigen

Zoals hieronder duidelijk zal worden kan de manipulatie van expressies veel termen opleveren die vereenvoudigd, verwijderd of samengevoegd kunnen worden. Dit kan vrij gecompliceerd zijn omdat de subbomen die tegen elkaar weggestreept kunnen worden niet direct naast elkaar hoeven te liggen.

Hier verwachten we dat lokaal de volgende vereenvoudigingen worden uitgevoerd.

- Als de operanden van een operatie (functie) bekende waarden zijn, wordt de operatie uitgevoerd: bijvoorbeeld $3+11 = 14$, $\cos(14) = 0.1367372182$. Dit kan herhaald worden, zodat bomen waarvan alle bladeren getallen zijn worden uitgerekend. Het resultaat is dan een getal (dat in de wortel staat).
- De speciale eigenschappen van 0 en 1 worden benut. Als E een willekeurige expressie is, dan $E + 0 = E$, $E \cdot 0 = 0$, $E \cdot 1 = E$, maar ook $E^0 = 1$, en $E^1 = E$.
- Variabelen kunnen bij aftrekken en delen tegen elkaar weggestreept worden: $x - x = 0$, $x/x = 1$.
- Denk zelf na over hoe $E/0$ afgehandeld gaat worden.

Tips. Merk op dat als je `double`s gebruikt, niet iedere waarde nauwkeurig opgeslagen of berekend kan worden, zo is $\sin(\pi)$ niet 0. Indien je `double`s gaat vergelijken met elkaar is het dus handig dit te doen door te kijken of het verschil kleiner is dan een van tevoren vastgestelde kleine waarde.

5 Evalueren

Bij de evaluatie vullen we een gegeven waarde voor de *vaste variabele* x in en rekenen dan de expressie uit. Als de expressie geen andere variabelen bevat is de uitkomst een getal. In het algemeen kan de expressie niet tot een getal teruggebracht worden wanneer er nog andere variabelen in staan. Wel is het de bedoeling dat bij evaluatie de uitkomst naar vermogen vereenvoudigd wordt, zie hiervoor.

6 Differentiëren

Onderdeel van de opdracht is een beperkte vorm van formule manipulatie, namelijk het bepalen van de afgeleide van de functie, en wel naar de *vaste variabele* x.

De afgeleide naar x kan in principe *recursief* worden bepaald. Bij die rcursie worden soms kopieën van subbomen gemaakt. We gebruiken ∂ om differentiëren naar de vaste variable x voor te stellen. De basis regels zijn dan gegeven in Tabel 1.

De kettingregel uit de tabel kent u waarschijnlijk als $(f(g))' = f'(g) \cdot g'$. Deze formule wordt gebruikt als we de afgeleide willen bepalen van een (standaard-)functie $f(\cdot)$ waarvan het argument niet x is maar een functie $g(x)$ van x.

$\partial(C)$	$=$	0	constante
$\partial(x)$	$=$	1	
$\partial(y)$	$=$	0	variabele $y \neq x$
$\partial(x^C)$	$=$	$C \cdot x^{C-1}$	macht met constante
$\partial(\cos(x))$	$=$	$-\sin(x)$	cosinus
$\partial(\sin(x))$	$=$	$\cos(x)$	sinus
$\partial(f+g)$	$=$	$\partial(f)+\partial(g)$	som
$\partial(f \cdot g)$	$=$	$\partial(f) \cdot g + f \cdot \partial(g)$	product
$\partial\left(\frac{f}{g}\right)$	$=$	$\frac{\partial(f) \cdot g - f \cdot \partial(g)}{g^2}$	quotient
$\partial(f \circ g)$	$=$	$(\partial(f) \circ g) \cdot \partial(g)$	kettingregel

Tabel 1: Regels voor differentiëren.

We rekenen bijvoorbeeld 'symbolisch' uit:

$\partial(\,\sin^3(\pi x)\,)$	$=$	macht + ketting
$3\sin^2(\pi x) \cdot \partial(\,\sin(\pi x)\,)$	$=$	sinus + ketting
$3\sin^2(\pi x) \cdot \cos(\pi x) \cdot \partial(\,\pi x\,)$	$=$	product
$3\sin^2(\pi x) \cdot \cos(\pi x) \cdot (\partial(\,\pi\,)x + \pi\partial(\,x\,))$	$=$	constante, variabele
$3\sin^2(\pi x) \cdot \cos(\pi x) \cdot (0 \cdot x + \pi \cdot 1)$		

hetgeen nog vereenvoudigd kan worden.

Het plaatje op de eerste bladzijde geeft de resulterende boom weer.

Je ziet dat er geen aparte regel $\partial(C \cdot f) = C \cdot \partial(f)$ is vermeld. Met de gewone productregel wordt dat achteraf opgevangen door te vereenvoudigen: $\partial(C \cdot f) = \partial(C) \cdot f + C \cdot \partial(f) = 0 + C \cdot \partial(f) = C \cdot \partial(f)$.

Tips. Teken de regels voor afgeleiden in boomvorm. Dan wordt zichtbaar hoe kopieën van subbomen en recursieve aanroepen gecombineerd worden.
Het kan handig zijn als je functie voor het bepalen van een afgeleide een nieuwe subboom returnt, in plaats van de boom aan te passen.
In C++ kan een klasse methodes hebben met dezelfde naam, als het returntype en/of de types van de argumenten (inclusief hoeveelheid) anders is. Dit kan bijvoorbeeld gebruikt worden om je private en public versies van recursieve functies dezelfde naam te geven.

En wat doet het programma dan precies?

Uw programma houdt een expressieboom bij, die met instructies kan worden gemanipuleerd. We verwachten in `main` een "interactief" programma dat de volgende instructies accepteert.

- `exp <expressie>` – slaat de gegeven prefix-expressie op als boom.
- `print` – print de opgeslagen boom, met infix notatie. Let op de haakjes.
- `dot <bestandsnaam>` – slaat de huidige boom op naar de gegeven file, in DOT notatie.
- `end` – om het programma af te sluiten.

Voor de tweede deadline komt daar dan nog bij:

- `eval <waarde>` – evalueert de opgeslagen boom, met als x de opgegeven waarde.
- `diff` – differentieert de opgeslagen expressie (naar x).
- `simp` – past simplify, de vereenvoudigingsoperatie, toe op de boom.

De operaties `exp`, `eval`, `simp` en `diff` vervangen de boom dus door een nieuwe. Met die boom wordt vervolgens doorgegaan. In het bijzonder kun je dus nieuwe expressies inlezen met `exp`.

Een voorbeeld van een simpele invoer voor het eerste deel is:

```
exp + / * ^ x 3 - y sin + * 2 x 6 + -3.7 * 15 x 5
dot test1.dot
exp cos * * pi x 2
print
end
```

Wanneer de print functie wordt aangeroepen is het de bedoeling dat er alleen uitvoer wordt weergegeven, het is dus niet de bedoeling om zoiets als "De uitvoer is: ..." af te drukken. Ook moet het menu op een eenvoudige manier uit te zetten zijn door middel van flags/argumenten.

```
#include<iostream>

int main(int argc, char** argv){
   bool debugMode;
   if(argc < 2){debugMode = false;}
   else{
      if (std::string(argv[1]) == "d"){debugMode = true;}
   }
   std::cout << debugMode << std::endl;
}
```

Tips. Met `./main d < file.txt` kan de inhoud van een bestand met vooraf bepaalde instructies naar standaard `in` worden gestuurd, in "debugmode", dat maakt het testen makkelijker. *Zorg dat dit werkt!* (want wij gaan dat zo ook toepassen).

Bijvoorbeeld met de volgende instructies in `file.txt`

```
exp + / * ^ x 3 - y sin + * 2 x 6 + -3.7 * 15 x 5
dot test1.dot
exp * x cos * * pi x 2
print
diff
print
simp
print
end
```

verwachten we iets als

```
x*cos(pi*x*2)
1*cos(pi*x*2)+x*-1*sin(pi*x*2)*((0*x+pi*1)*2+pi*x*0)
cos(pi*x*2)+x*-1*sin(pi*x*2)*6.28319
```

Natuurlijk kan de gebruiker een fout maken en een incorrecte expressie ingeven; probeer zoveel mogelijk te zorgen dat dit opgevangen wordt zonder dat het programma vast loopt.

Instructies. Werk in tweetallen. Volg de aanwijzingen voor nette code en documentatie. Voeg een makefile toe. Lever je werk als één file in, gezipt. Succes.

(v2) 3.10'22 (27.9'22)