Een Interpreter in Functionele Programmeerstijl

Contactinformatie:
Adres *ELIS (verdieping -3), Technicum (blok I), Sint-Pietersnieuwstraat 41*
E-mail *compiler@elis.ugent.be*

BELANGRIJK: Het indienen van de oplossingen dient steeds te gebeuren via het Dropbox-menu op Minerva, dit zowel voor een tijdelijke versie op het einde van elk practicum als de finale versie voor onderstaande dealine. Gebruik het `make_tarball.sh` script om een archief te genereren, en zorg dat het naar de begeleiders verzonden wordt.

DEADLINE: 24 februari, 23:59.

1 Doelstelling

In dit practicum is het de bedoeling dat je een interpreter schrijft voor heel eenvoudige programma's zonder lussen of sprongen (zgn. straight-line programma's). Teneinde ons niet te moeten bezighouden met het parsen van de programmatekst zullen de programma's gespecificeerd worden in een boomstructuur, de abstracte syntaxisboom. Zoals later in de cursus zal blijken, is dit een zeer belangrijke voorstelling van een programma. Het is de bedoeling dat je in dit practicum, dat tegelijk dient als een opfrissing van je kennis van (of eerste kennismaking met) de taal C, gebruik maakt van een functionele programmeerstijl. Dit wil zeggen dat je programma's schrijft zonder neveneffecten. Concreet zorg je ervoor dat je variabelen enkel een waarde toewijst tijdens de initialisatie en deze waarde nooit meer verandert (op voorhand declareren mag natuurlijk). Hetzelfde doe je met velden van struct's. Om de initialisatie van een struct eenvoudiger te laten verlopen maak je best een constructorfunctie aan (voorbeelden hiervan kan je terugvinden in de initiële programmabestanden).

2 De taal

De taal wordt beschreven door de grammatica in onderstaande tabel:

Een programma in deze taal is eigenlijk gewoon een statement. De abstracte syntaxisboom zal dan ook als wortel een statementknoop hebben. De semantiek van deze taal spreekt grotendeels voor zich, maar er zijn toch een aantal constructies die extra aandacht behoeven:

- een PrintStm zal de expressies in de expList van links naar rechts evalueren en de resultaten uitschrijven gescheiden door een spatie. De uitvoer wordt beëindigd met een newline. Bij de evaluatie van een deelexpressie moet rekening gehouden worden met de neveneffecten die het gevolg zijn van alle voorgaande evaluaties.
- bij de EseqExp wordt eerst *stm* uitgevoerd, vervolgens wordt *exp* geëvalueerd, rekening houdend met de neveneffecten van de uitvoering van *stm*.
- bij de OpExp wordt steeds eerst het linkerargument geëvalueerd, en wordt het rechterargument geëvalueerd rekening houdend met de neveneffecten van de evaluatie van het linkerargument.

lhs	rhs	betekenis
stm	*stm*;*stm*	CompoundStm
stm	id := *exp*	AssignStm
stm	print (*expList*)	PrintStm
exp	id	IdExp
exp	num	NumExp
exp	*exp binop exp*	OpExp
exp	(*stm*,*exp*)	EseqExp
expList	*exp*,*expList*	PairExpList
expList	*exp*	LastExpList
binop	+	Plus
binop	-	Minus
binop	*	Times
binop	/	Div

3 Opgave

Download de initiële programmabestanden van Minerva, pak het bestand uit en bekijk de gegeven code. In `slp.h` staan de definities van de datastructuren van de abstracte syntaxisboom. In `main.c` vind je de functies `interpStm` en `interpExp` terug, die jij moet implementeren. Zoals je ziet, aanvaarden deze functies twee argumenten: een statement of expressie en een `A_table`. Dit laatste argument belichaamt de omgeving waarin het statement of de expressie moet worden geëvalueerd. Deze omgeving is niets meer dan een gelinkte lijst met de gedefinieerde variabelen en hun huidige waarde.

Teneinde de functionele programmeerstijl niet te hoeven verbreken, worden de oude waarden van variabelen niet uit de omgeving verwijderd: telkens een assignatie gebeurt, wordt de nieuwe waarde gewoon vooraan de gelinkte lijst toegevoegd. De huidige waarde van de variabele kan dan ook eenvoudig gevonden worden door de lijst van voor af aan te doorzoeken en de eerste verschijning van de variabele weer te geven. Hiervoor zijn de functies `lookup` en `update` voorzien in `main.c`.

De interpreterfuncties geven beide een waarde terug van het type `struct IntAndTable`. Deze structuur bevat de waarde van de geëvalueerde expressie (in het geval van een statement mag je hier een willekeurige waarde invullen), en de nieuwe omgeving na de evaluatie van de expressie of het statement. Eens je de code hebt aangevuld, kan je het geheel compileren met het commando `make`. Er wordt dan een uitvoerbaar bestand met de naam *main* aangemaakt, dat de programma's gedefinieerd in `prog1.c` en `prog2.c` uitvoert.

Als alles goed gaat, is de uitvoer van deze programma's:

```
Output program 1:
-----------------
8 7
80

Output program 2:
-----------------
```

```
8 8 8 8 8 8
88 88 88 88
888 888 888
88888 88888
888 888 888
88 88 88 88
8 8 8 8 8 8
```

Probeer ook nog wat andere programmaatjes uit door `prog1.c` aan te passen, tot je ervan overtuigd bent dat de interpreter correct werkt.

Let op: bij het verbeteren wordt enkel gekeken naar `main.c`, andere bestanden mogen niet gewijzigd worden en moeten niet opgenomen worden in het archief dat jullie indienen.