Practicum Programmeerprincipes 2006-2007

cderoove@vub.ac.be

OPLOSSINGEN REEKS 1
KENNISMAKING MET PICO

Evaluatie van expressies

Oefening 1. "Oplossing" van deze kennismakingsoefening gegeven in de les.

Oefening 2. Voorspel, controleer in de programmeeromgeving en verklaar de resultaten van de volgende Pico-sessie:

a. `2 * 2 + 5`

Pico gebruikt de normale voorrangsregels en geeft dus het verwachte resultaat.

b. `2 * (2 + 5)`

Idem.

c. `*(2,2+5)`

De rekenkundige bewerkingen zijn in Pico ingebouwd als functies. Als je de expressie `"+"` evalueert, krijg je namelijk een functie terug die als volgt door Pico neergeschreven wordt: `"<native function +>"` (probeer dit zelf!). Functies worden in Pico opgeroepen met de normale wiskundige notatie: `"naam(argument_1, ..., argument_n)".` Je kan de expressie `2 * (2+5)` dus net zo goed neerschrijven als `*(2,2+5).` Het omgekeerde is echter niet waar: het is niet altijd mogelijk om een willekeurige functienaam in rekenkundige infix notatie te gebruiken. De klasse functienamen waarvoor dit wel mogelijk is, worden operatoren genoemd. Zij mogen ook steeds slechts met een of twee argumenten opgeroepen worden. Geldige operatoren bestaan uit minstens een van (in de notatie aangeduid met een +) de volgende tekens:

`{+,-,*,/,\,^,=,<,>,§,#,%,&,.,|}+`

d. `*(2,+(2,5))`

Idem. We hebben hier de naam "*" die naar een functie evalueert

die toegepast wordt op twee argumenten. Het eerste argument evalueert naar een getal terwijl het tweede argument opnieuw een functietoepassing is waarvan het resultaat uiteindelijk ook een getal zal opleveren. Argumenten van functies worden steeds van links naar rechts geëvalueerd.

e. `5(10+2)`

```
> 5(10,2)
excess input
> |
```

Een getal is geen geldige naam voor een functie. Pico produceert deze foutmelding bij het lezen van ingevoerde expressies (er werd een ander cijfer of een operator verwacht op de plaats die rood gekleurd is). Geldige namen zijn van de volgende vorm (een letter gevolgd door geen of meerdere (de * in de notatie) alfanumerieke tekens):

{a,A,b,B,...}{a,A,b,B,...,0,1,2,...}*

Bij het evalueren van functieoproep "`naam(argument_1, ..., argument_n)`" wordt ook verondersteld dat `naam` de naam van een geldige functie is (deze naam moet met andere woorden naar een functiewaarde evalueren zoals het geval was bij de naam **+** uit oefening 1):

```
> z : 5
5
> z(10)
***Traceback for error #17
***evaluating:
z(10)
function required
>
```

In de eerste expressie uit het bovenstaande voorbeeld wordt de naam **z** gebonden aan het getal **5**. De volgende expressie is van de vorm `naam(argument_1, ..., argument_n)`: een toepassing van de functie gebonden aan de naam **z** op het getal **10**. Pico verwacht dus dat **z** aan een functie gebonden is, maar z blijkt aan het getal **5** gebonden te zijn wanneer Pico **z** in het woordenboek opzoekt. Dit wordt gesignaleerd met de foutmelding "function required".

f. `x:2`

Het toevoegen van een naam aan het Pico woordenboek (het definiëren van een variabele) geeft net als elke andere expressie in Pico een waarde terug. Deze waarde kan dan verder gebruikt

worden als deelexpressie van een grotere expressie:

```
> z : 2 + 2
4
> z
4
> (z:2) + 2
4
> z
2
>
```

g. x:=5*x

In het woordenboek was reeds eerder de waarde 2 verbonden met de naam "x". Een expressie van de vorm "naam := expressie" zal de huidige waarde verbonden met de naam "x" overschrijven met een nieuwe waarde. Deze waarde is gelijk aan het resultaat van de evaluatie van de expressie aan de rechterkant. Zelfde opmerking als voorheen: het resultaat van de evaluatie van deze toewijzing is opnieuw een waarde die in expressies verder gebruikt kan worden.

h. x

Pico zoekt de waarde verbonden met de naam "x" op in het woordenboek en geeft deze terug. De oude waarde van x werd overschreven met de nieuwe waarde: het getal 10.

i. z

```
> z
***Traceback for error #52
***evaluating:
z
undefined reference: z
>
```

Pico probeert net zoals in de vorige oefening de waarde verbonden met de naam "z" op te zoeken in het woordenbooek. Deze naam kan echter niet gevonden worden in het woordenboek omdat er nog geen waarde aan toegekend werd in deze oefeningenreeks.

j. z:=10

```
> z := 10
***Traceback for error #52
***evaluating:
10
***evaluating:
z:= 10
undefined reference: z
>
```

Idem. Om de waarde van "z" te kunnen overschrijven, moet deze eerst opgezocht worden in het woordenboek, waar er nog geen binding bestaat voor z.

k. `f(x,y,z):x+y+z`

Aan de naam **"f"** wordt als waarde een functie met 3 parameters gebonden. Het functievoorschrift bestaat uit de optelling van deze parameters. Als resultaat van deze definitie krijgen we de gedefinieerde functie terug.

l. `f(1,2,3)`

Bij de evaluatie van een functie-oproep wordt de waarde verbonden aan de functienaam opgezocht en het gevonden voorschrift wordt toegepast op de geëvalueerde argumenten: `1 + 2 + 3`. Merk op dat het resultaat van deze functieoproep NIET `10 + 2 + 3` is. De waarde van een argument wordt namelijk tijdelijk in het woordenboek (dit wil zeggen: tijdens het toepassen van de functie) gebonden aan de naam van dat argument. Tijdens de uitvoering van het functievoorschrift **"x+y+z"** heeft x dus de waarde 1 in plaats van 10. Na de uitvoering heeft x terug de waarde 10. Op dit mechanisme zal later nog verder ingegaan worden.

m. `f("hal","l",char(111))`

De optelling werkt in Pico niet alleen op getallen, maar ook op tekenreeksen. Dit zijn aaneenschakelingen van tekens omgeven door twee "". Pico geeft deze bij evaluatie weer als gewone tekst. De optelling is op tekenreeksen gedefinieerd als de concatenatiefunctie. Je kan een teken (een tekenreeks van grootte 1) ook omvormen naar een getal met de `ord(teken)` functie of een getal naar een teken met de `char(getal)` functie.

Je kan deze tekenreeksen gebruiken om met de gebruiker van een programma te communiceren. Tekenreeksen toon je bijvoorbeeld aan de gebruiker (= printen op het scherm) door deze aan de functie `display(tekenreeks)` te geven. Je kan ook invoer van de gebruiker terugkrijgen door de functie `accept()` op te roepen. Het resultaat van deze oproep is de door de gebruiker ingegeven tekst.

```
> "o"+"p"+"sakee"
opsakee
> tekenreeks : char(111) + char(ord("p")) + "sakee"
opsakee
> display("en nog eens " + tekenreeks)
en nog eens opsakee
> weekdag : accept()
woensdag
woensdag
>
```

n. f(2,3,4,5)

```
> f(x,y,z) : x+y+z
<function f>
> f(2,3,4,5)
***Traceback for error #18
***evaluating:
f(2, 3, 4, 5)
invalid argument count
>
```

De functie f is een functie van 3 argumenten en kan dus niet toegepast worden op 4 getallen.

o. f

Aan de naam "f" is een functie gebonden in het woordenboek.

p. {z:15; "hallo" }

Op deze manier kunnen meerdere expressies na elkaar geëvalueerd worden: {subexpressie_1; subexpressie_2; ....; subexpressie_n} Merk op dat de punt-komma een teken is dat de verschillende deelexpressies van elkaar scheidt. Na de laatste expressie hoort dus geen scheidingsteken meer te staan. De waarde van zulk een accolade-expressie is de waarde van de laatste deelexpressie. De deelexpressies worden van links naar rechts geëvalueerd.

q. begin("hallo", z)

Intern worden accolade-expressies vertaald naar een oproep van de begin-functie (de accolades zijn als het ware syntactische suiker die de bittere pil van een lange functieoproep zoeter moeten maken, maar functioneel zijn de twee methodes equivalent).

r. true

Booleans zijn functies (van twee argumenten) ! (Church)

s. true(2,5)

Bij oproep geeft de true-functie steeds de waarde van het eerste argument terug.

t. false(2,5)

Bij oproep geeft de false-functie steeds de waarde van het tweede argument terug.

u. abc : xyz

xyz is nog niet gedefinieerd in het woordenboek.

v. y : +x

Merk op dat de waarde van x hersteld is na de functie-oproep van

f.

w. `g(x) : void`

`void` is een speciale plaatshouder-waarde (die eigenlijk het ontbreken van een andere waarde aangeeft)

x. `g(display("ok"))`

Het argument van `g` (= de expressie `display('ok')`) wordt altijd geëvalueerd bij de oproep van `g` waardoor `'ok'` op het scherm geprint wordt.

Let ook op het verschil tussen de manier waarop Pico het resultaat van de evaluatie van een expressie op het scherm zet en de manier waarop de display-functie een tekenreeks op het scherm toont. De display-functie toont tekst in een kader in het paars terwijl de waarde waarnaar een expressie geëvalueerd wordt in het blauw getoond wordt.

In dit voorbeeld evalueert elke oproep van de functie `g(x)` naar de waarde `<void>` die in het blauw op het scherm gezet wordt. De paarse tekst op het scherm is afkomstig van de evaluatie van het argument van de oproep `g(display("ok"))`.

```
> g(x):void
<function g>
> g(display("ok"))
ok
<void>
> g(3)
<void>
> display("ok")
ok
>
```

y. `true(display("ok"),display("ko"))`

```
> true(display("ok"), display("ko"))
ok
>
```

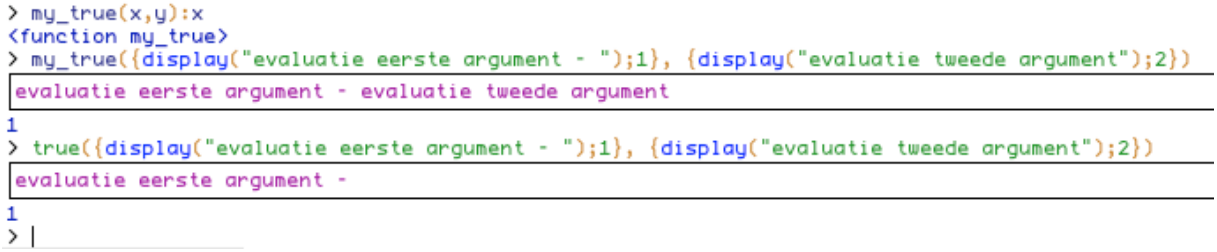
Functies evalueren normaal gezien alle argumenten van links naar rechts. De booleaanse functies vormen hierop (uiterlijk, maar hierover later meer) echter een uitzondering. `true` evalueert alleen het eerste argument terwijl `false` alleen het tweede argument evalueert.

`my_true(x,y) : x`

De bovenstaande `my_true(x,y)` functie zal zoals elke gewone functie in Pico AL haar argumenten van links naar rechts evalueren maar enkel het resultaat van de evaluatie van het eerste argument teruggeven. De ingebouwde `true(x,y)` (waarvan we later nog de definitie zullen bestuderen) zal echter niet al haar argumenten evalueren, maar slechts haar eerste argument en hier dan de

waarde van teruggeven. Van het tweede argument wordt nooit de waarde berekend.

Het verschil wordt duidelijk bij de volgende oproepen:



z. `{d(x) : x * 2; g(x) := x + 1; g(d(d(d(10))))}`