TDP019:
Projekt: Datorspråk

Grammatik
Lexikalisk analys
Parsning
Syntaxträd
Interpretering över syntaxträd

Olika tekniker för att hantera datorspråk

Beskrivning av datorspråk

Syntax:
Reguljära uttryck
BNF grammatik

Semantik:
Vanlig beskrivning i text:
”*man börjar med att beräkna …*”

Interpretatorn, evaluatorn

2

Lexikalisk analys

1 En sekvens med tecken -> en sekvens med tokens.
Dela upp i tokens

”value = -x-3.45+pi+(-4+upper)” ->

”if x>3 then x = sin(y) else x=10” ->

3

Lexikalisk analys

Definiera:
Identifierare

Integer (heltal)

Float (flyttal)

Sanningsvärden (logiska värden)

Kommentar

4

Lexikalisk analys

Teknik:

Egen teckenhantering

Använda reguljära uttryck

Använda tokenizer

Använda verktyg

5

Lexikalisk analys

Vissa konstruktioner kanske tas om hand på ett eget speciellt sätt, t ex

Kommentarer i programmet.
Strängar

Vad är separatorer? Blank-tecken, ny rad

Indentering som syntaktisk konstruktion

6

BNF-grammatik - Parsning

Övningar.

Läsa grammatik och se om konstruktioner uppfyller grammatiken eller ej.

Skriv grammatik för en repetitionsstruktur, t ex for, while, repeat …

7

Övning

Enkelt imperativt språk. Nyckelordstyrd. Skriv grammatik.

```
program
        läs antal;
        om (antal < 0)
                då skriv "Fel indata";
                annars skriv "ok";
        tilldela i  0;
        tilldela summa 0;
        repetera-tills (i > antal)
                tilldela summa summa + 1;
                tilldela i i-1;
        slut-repetera;
        skriv "summeringen blev: ";
        skriv summa;
slut-program
```

8

Övning

Skriv regler:

<program> ::=
<satser> ::=
<sats> ::=
<tilldelning> ::=
<utmatning> ::=
<inmatning> ::=
<villkor> ::=
<repetition> ::=
<uttryck> ::=
<aritm-uttryck> ::=
<predikat-uttryck> ::=
… ::=

9

Järnvägsalgoritmen

Överför infix-notation till postfix-notation.
(Se t ex wikipedia Järnvägsalgoritmen)

3+4-(5+6*7) ->
x = f(x+g(y, z)) + v ->

Postfixuttrycket kan antingen beräknas direkt eller bygga ett syntaxträd.

10

Syntaxträd

11

Parsning av andra konstruktioner

Olika typer av deklarationer, block och satser.

```
if x=4
        then {y = x+z; print(y)}
        else  if x=5 then y = x+10 endif
endif
```

12

Parsning av andra konstruktioner

Från tokens bygga syntaxträd.

Parsningstekniker:
Egen kod
Använda verktyg (t ex Rdparser från TDP007)

Klart svåraste momentet när man skall bygga ett språk.

13

Parsning av andra konstruktioner

Syntaxträd.

Varje grammatikregel, ger information hur en nod i syntaxträdet skall se ut.

villkor ::= IF predikatuttr THEN sats1 ELSE sats2

14

Implementering av syntaxträd

villkor ::= IF predikatuttr THEN sats1 ELSE sats2

En nod är en post eller ett objekt.

IF-objektet innehåller:
Referens till noderna för prediktatuttr, sats1 och sats2.

Metoder:
Konstruktor, skapa-if-objekt
Evaluera/exekvera/utföra
Typkontrollera konstruktionen
Skriv-ut / logg-utskrift

15

Interpretator

Teckensyntax -> syntaxträd

Evaluering över syntaxträdet.

Varje grammatik-regel har sin evaluerings-funktion/metod

Varje konstruktion beräknas relativt en omgivning ,dvs det aktuella läget på variabelbindningar.

print (x+y)
skriver ut 15 om x har värdet 10 och y värdet 5

16

Hur hanteras omgivning/variabeltabell

En omgivning består oftast av ett antal ramar (frames).

En ram skapas då man går in i ett block, funktion, metod etc, där lokala variabler/parametrar finns.

{int i, j; i = read; …… }

define f (int x, string s, float f) ……

En ram tas bort då man lämnar blocket/funktionen/metoden.

17

Hur hanteras omgivning/variabeltabell

I ett enkelt språk kan ramarna med variabelbindningar lagras i en stack.

Ramen för den aktuella funktionen man evaluerar ligger överst. En stackpekare håller reda på var den aktuella omgivningen börjar.

Ramar som tagits bort kan överlagras av andra på stacken.

Slå upp värde / tilldela värde kräver att man kan söka i omgivningen.

18

Hur hanteras omgivning/variabeltabell

Blockstruktur
Räckvidd (scope)
Statisk bindning
Variabler kan skymma varandra

{int i, int j; i=3; j=5
  {int j, int k; j=10; k=i+j …}
  i=j+2;
  …}

19

Anrop och återhopp från funktion

Vi måste lagra vart man skall återgå efter det att en funktion/metod har beräknats klart.

Hur skickas värdet tillbaka.

def f (x) x+3

5+f(1)+2

20

Statisk/dynamisk omgivning

def P(x)

  def Q(y) …. x+y …

  def R(x) … Q(x+1) …

Q(1)
R(2)

Vi anropar med P(1).
Man kan ha två tolkningar. Statisk resp. dynamisk omg.

21

Statisk/dynamisk omgivning

Man måste lägga in dynamiska och statiska länkar i stacken.
Se t ex Sebesta ”Concepts of Programming Languages”

Klarar man rekursivitet. Kan en funktion anropa sig själv?

22

Fria variabler i procedurer/lambda-uttryck

Closure

def f (x)
… returnerar t ex ett lambda-uttryck
 lambda y : x+y

g = f(1)
x=10
g(2) = 3 eller 12 ? Vilket x skall gälla.

Klarar man rekursivitet? När man anropar sig själv?

23

Kontroll av felaktig syntax.

Hur kontrollerar man.
Kan man ha syntaxkontrollerare som tar en del av ett syntaxträd och avgör om det underliggande trädet är ok eller ej?
Vid typning, hur avgör man att man har rätt typ på argumenten, t ex

int i, str s; … i + s …

Gå igenom syntaxträdet och för varje konstruktion gör ”typresonemang”.

24