Vereistes: leksikale ontleder

1. Verwyder blanko's en lyn-end karakters
2. Herken sleutelwoorde soos WHILE, DO, DIV, ens.
3. Herken name (letters en/of syfers wat begin met 'n letter)
4. Herken konstantes (stringe syfers). Ken waarde toe aan aan 'n spesiale veranderlike.
5. Herken spesiale simbole (":=", "[", ",", ens.)
6. Verwyder kommentaar.

Spesifikasie: leksikale ontleder

Indien ons die leksikale ontleder outomaties wil genereer, moet ons die vereistes *formeel* spesifiseer.

Die spesifikasie word geskryf as regulêre uitdrukkings. (Die program "lex" op Unix is 'n voorbeeld van hierdie benadering.)

Die vereistes is eenvoudig genoeg om 'n informele benadering te volg indien ons die leksikale ontleder self wil skryf.

Ontwerp: leksikale ontleder

Bedink 'n strategie om elke vereiste na te kom:

1. Blanko's en lyn-end karakters
2. Sleutelwoorde
3. Name
4. Konstantes
5. Spesiale simbole
6. Kommentaar


```
MODULE Scanner;
(* constant definitions for all symbols *)
VAR ch: CHAR; (* current character *)

PROCEDURE Get*(VAR sym: INTEGER);
(* ignore blanks *)
IF end of text THEN sym := eof
ELSE
  CASE ch OF
    "&": Read(ch); sym := and
  | "*": Read(ch); sym := times
  | (* similar for other legal characters *)
  | "0".."9": Number
  | "A".."Z", "a".."z": Ident
  | "~": Read(ch); sym := not
  ELSE Read(ch); sym := null
  END
END Get;

BEGIN
  (* enter keywords in table *)
END Scanner.
```

```
PROCEDURE Ident;
VAR i, k: INTEGER;
BEGIN i := 0;
  REPEAT
    IF i < IdLen THEN id[i] := ch; INC(i) END;
    Read(ch)
  UNTIL (ch < "0") OR (ch > "9") &
        (CAP(ch) < "A") OR (CAP(ch) > "Z");
  id[i] := 0X; k := 0;
  (* find keyword in table *)
  If keyword found THEN sym := keyTab[k].sym
  ELSE sym := ident
  END
END Ident;
```



```
PROCEDURE Number;
BEGIN val := 0; sym := number;
  REPEAT
    IF val small enough THEN
      val := 10 * val + (ORD(ch) - ORD("0"))
    ELSE Error(0); val := 0
    END;
    Read(ch)
  UNTIL (ch < "0") OR (ch > "9");
END Number;
```



```
PROCEDURE Error(nr: INTEGER);
BEGIN
  CASE nr OF
    0: Write("number too large")
  | 1: Write("message number 1")
  (* more messages *)
  ELSE Write("undefined error")
  END
END Error;
```

Voordele van hierdie benadering:

- Makliker om boodskappe te vertaal (na Afrikaans byvoorbeeld) omdat almal in een plek is
- Boodskappe kan in 'n lêer gestoor word om geheue te bespaar

Die programmeertaal C (kort oorsig)

Opdragte word *getermineer* met kommapunte. Saamgestelde opdagte (soos `if` of `while`) word nie getermineer met kommapunte nie.

Toekenningsopdragte, byvoorbeeld:

x = (x + y) * 3;

Rekenkundige operatore:

`+ - * / %`

Relasionele operatore:

`== != > < >= <=`

Boolese operatore:

```
&&  ||  !
```

Bitsgewyse operatore:

```
&  |  ^  <<  >>  ~
```

Verklarings aan begin van funksies, byvoorbeeld:

```
int x; /* x is of type integer */
char *p; /* p points to a character */
```

Algemene tipes:

```
int, short, char, float, double
```


if-opdrag:

```
if (x < 10) {
  y = 0; z = 0;
}
else {
  y = x - 1;
}
```

switch-opdrag:

```
switch (i) {
case 1: printf("i = 1\n"); break;
case 2: printf("i = 2\n"); break;
case 3: printf("i = 3\n"); break;
default: print("i not 1, 2 or 3\n"); exit(1);
}
```

while-opdrag:

```
while (x > 0) {
  x = x - 1;
}
```


Funksies is die basiese boublok. Byvoorbeeld:

```
int max(int a, int b) {

  if (a > b) {
    return a;
  }
  else {
    return b;
  }
}
```

'n Funksie wat geen waarde lewer nie—'n prosedure-se tipe is `void`. Hakies is verpligtend na die funksienaam, selfs al is daar geen parameters nie.

Koderingsriglyne vir projek

1. Konsekwente inkeping met behulp van "tab" karakters is verpligtend. Blanko's mag nie gebruik word nie.

2. Spesifieke formate is verpligtend vir inkeping. Byvoorbeeld:

   ```
   while (x > 0) {
     if (y > 20) {
       x = x + 1;
     }
     else {
       x = x - 1;
     }
   }
   ```

3. Vir leesbaarheid moet die konstantes TRUE (1) en FALSE (0) altyd verklaar word indien met Boolese waardes gewerk word.

4. Die toekenningsoperator in Boolese uitdrukkings en verkorte vorms van toekennings mag nie gebruik word nie. Byvoorbeeld, die volgende is onaanvaarbaar:

```
main(){
  int sum, i, n;

  printf("enter positive number\n");
  scanf("%d", &n);
  sum = i = 1;
  n *= 2;
  while ((i += 2) < n) sum += (i * i);
  printf("result = %d\n", sum);
  return 0;
}
```


Die volgende (ekwivalente) kode is aanvaarbaar (en meer leesbaar):

```
main(){
  int sum, i, n;

  printf("enter positive number\n");
  scanf("%d", &n);
  sum = 1; i = 3;
  n = n * 2;
  while (i < n) {
    sum = sum + (i * i); i = i + 2;
  }
  printf("result = %d\n", sum);
  return 0;
}
```


5. Verklarings van tipes en veranderlikes mag in geen ander posisie as aan die begin van funksies geplaas word nie.

6. Veranderlikes mag nie as deel van hulle verklarings geïnisialiseer word nie.

7. Geen ander lusse as `while` lusse mag gebruik word.

8. Gebruik altyd krulhakies in `if-then-else` opdragte, selfs al is dit nie streng gesproke nodig nie. Byvoorbeeld:

```
acceptable:
if (x == 10) {
  x = x - 1;
}
else {
  x = 10;
  y = 0;
}
```

```
not acceptable:
if (x == 10) x = x - 1;
else {
  x = 10;
  y = 0;
}
```

9. Verskaf 'n `default` vir elke `switch` opdrag.

10. Voorwaardelike uitdrukkings soos `a > b ? a : b` mag nie gebruik word nie.

Voorbeeld: lees van tekslêer

```
#include <stdio.h>

main(int argc, char *argv[]){
  FILE *sfile;
  char ch;

  sfile = fopen(argv[1], "r");
  if (sfile == NULL) {
    printf("file could not be opened\n");
    return 1; /* abnormal termination */
  }
  ch = fgetc(sfile);
  while (!feof(sfile)) {
    putchar(ch); ch = fgetc(sfile);
  }
  return 0; /* normal termination */
}
```


Komponente van 'n vertaler

Alle vertalers het die volgende komponente:

- leksikale ontleder: herken basiese simbole
- sintaksontleder: herken sinne
- simbooltabel: stoor inligting oor entiteite:
  - veranderlikes se name en tipes
  - konstantes en hulle waardes
  - prosedures se name, hulle parameters en dié se tipes
  - gebruiker-gedefinieerde tipes.
- kodegenerator: produseer masjienkode wat ooreenstem met die bronkode se funksionaliteit

Ontwikkeling van sintaksontleder

- Lei sintaksontleder direk af vanaf EBNF—die spesifikasie.
- Voordeel: dit verseker dat die spesifikasie bevredig word.
- EBNF moet aan beperkings voldoen om te verseker dat korrekte produksiereël direk gekies kan word:
  - Bereken twee versamelings `First(S)` en `Follow(S)` vir elke nie-terminaal.
  - Elke produksiereël moet aan spesifieke vereistes voldoen in terme van `First` en `Follow`.

EBNF vir Oberon-0

```
ident = letter {letter | digit}.
integer = digit {digit}.
selector = {"." ident | "[" expression "]"}.
number = integer.
factor = ident selector | number |
   "(" expression ")" | "~" factor.
term = factor {("*"|"DIV"|"MOD"|"&") factor}.
SimpleExpression = ["+" | "-"] term
   {("+" | "-" | "OR") term}.
expression = SimpleExpression
   [("=" | "#" | "<" | "<=" | ">" | ">=")
   SimpleExpression].
assignment = ident selector ":=" expression.
ActualParameters = "(" [expression
    {"," expression}] ")".
ProcedureCall =
   ident selector [ActualParameters].
```

```
IfStatement =
   "IF" expression "THEN" StatementSequence
   {"ELSIF" expression "THEN" StatementSequence}
   ["ELSE" StatementSequence] "END".
WhileStatement = "WHILE" expression "DO"
   StatementSequence "END".
statement = [assignment | ProcedureCall |
   IfStatement | WhileStatement].
StatementSequence = statement {";" statement}.
IdentList = ident {"," ident}.
ArrayType = "ARRAY" expression "OF" type.
FieldList = "RECORD" FieldList
    {";" FiledList} "END".
RecordType = "RECORD" FieldList
   {";" FieldList} "END".
type = ident | ArrayType | RecordType.
```



```
FPSection = ["VAR"] IdentList ":" type.
FormalParameters = "(" [FPSection
   {";" FPSection}] ")".
ProcedureHeading = "PROCEDURE" ident
    [FormalParameters].
ProcedureBody = declarations
   ["BEGIN" StatementSequence] "END" ident.
ProcedureDeclaration =
   ProcedureHeading ";" ProcedureBody.
declarations =
   ["CONST" [ident "=" expression ";"}]
   ["TYPE" {ident "=" type ";"}]
   ["VAR" {IdentList ":" type ";"}]
   {ProcedureDeclaration ";"}.
module = "MODULE" ident ";" declarations
   ["BEGIN" StatementSequence] "END" ident ".".
```


Berekening van First(S)

`First(S)` bevat alle terminale simbole wat afgelei kan word uit `S`.

- As `S` leeg is, is `First(S)` $= \emptyset$
- As `s` 'n enkele terminale simbool is, is `First(s)` die simbool self. Byvoorbeeld: `RecordType` $= \{$`"RECORD"`$\}$.
- As alle sinne van die vorm `E` nie-leeg is, moet alle sinne van die vorm `EF` begin met een van die simbole van `E`.

  Daarom is `First(EF)` $=$ `First(E)`. Byvoorbeeld:
  `First(factor{("*"|"DIV"|"MOD"|"&") factor}`
  $=$ `First(factor)` $= \{$`"(", "~", integer, ident`$\}$.

- As sommige sinne van die vorm `E` leeg is, moet alle sinne van die vorm `EF` begin met een van die simbole van `First(E)` $\cup$ `First(F)`. Byvoorbeeld,
  `First(["BEGIN" StatementSequence] "END")`
  $= \{$`"BEGIN", "END"`$\}$.
- Alle sinne van die vorm `E|F` moet begin met een van die simbole van `First(E)` $\cup$ `First(F)`. Byvoorbeeld,
  `First(ident | ArrayType | RecordType)`
  $= \{$`ident, "ARRAY", "RECORD"`$\}$

Berekening van Follow(S)

Om die versameling simbole te bereken wat kan volg na nie-terminale simbool `T`, beskou produksies van die vorms `N = STU`, `N = S[T]U` en `N = S{T}U`.

- As alle sinne van die vorm `U` nie-leeg is, sluit `Follow(T)` ook `First(U)` in.
- As sommige sinne van die vorm `U` leeg is, sluit `Follow(T)` ook `Follow(N)` in.
- Produksies van die vorms `N = ST`, `N = S[T]` en `N = S{T}` wys dat `Follow(T)` ook `Follow(N)` insluit.
- As `T` voorkom in die vorm `{T}`, sluit `Follow(T)` ook `First(T)` in.

Beperkings of EBNF

- Vir alle sinne van die vorm `N = E|F`, moet `First(E)` ∩ `First(F)` leeg wees.
- As 'n sin van die vorm `N` leeg kan wees, moet `First(N)` ∩ `Follow(N)` leeg wees.
- Links-rekursie is ontoelaatbaar. (Waarom?)

Afleiding van sintaksontleder

- Skryf 'n prosedure wat ooreenstem met elke nie-terminale simbool.
- 'n Prosedure om 'n reël van die vorm `ABC` te ontleed bevat die algoritme vir `A`, gevolg deur die algoritme vir `B`, gevolg deur die algoritme vir `C`.
- Wanneer 'n enkele terminale simbool `s` verwag word, gebruik die volgende algoritme:

```
IF symbol = s THEN Get(symbol)
ELSE Error(n)
END
```


- Gebruik die volgende algoritme vir 'n sin van die vorm `[E]`:

```
IF symbol IN First(E) THEN E END
```

- 'n Sin van die vorm `{E}` word herken deur die volgende algoritme:

```
WHILE symbol IN First(E) DO E END
```

- As geen sin van die vorms `A`, `B` of `C` leeg is nie, sal die volgende algoritme 'n sin van die vorm `A|B|C` herken:

```
IF symbol IN First(A) THEN A
ESLIF symbol IN First(B) THEN B
ELSEIF symbol IN First(C) THEN C
ELSE Error(n)
END
```

Wenke

- Onthou dat die sintaksontleder altyd slegs *een* simbool vooruit tas. Dis voldoende om die regte produksiereël te kies.
- Elke prosedure ontleed *een* produksiereël; gebruik die reël as kommentaar om onderhoud te vereenvoudig.
- Teken 'n diagram om die afhanklikhede tussen nie-terminale aan te dui. Dit sal help om die prosedures in 'n sinvolle volgorde te rangskik.
- Plaas 'n afvoer-opdrag aan die begin van elke prosedure om die naam van die prosedure te vertoon. Dit sal help om die sintaksontleder te ontfout.

Hoe om 'n sintaksontleder te toets

- Minimum vereiste: elke lyn kode moet ten minste een keer uitgevoer word.
- Skryf twee aparte toetsprogramme: een om ontleding van korrekte sinne te toets en 'n ander om die opsporig van sintaksfoute te toets.
- Sluit toetsgevalle in om elke opsie van elke if-opdrag ten minste een keer uit te voer.
- Sluit toetsgevalle in om elke while-lus nul kere uit te voer en ook om dit een of meer kere uit te voer.
- Bestudeer elke prosedure van die sintaksontleder apart om toetsgevalle op te stel.

Foutboodskappe

- Indien 'n sintaksfout opgespoor word, moet die sintaksontleder die lynnommer en 'n kort, sinvolle foutboodskap vertoon. Byvoorbeeld: "line 23: ']' expected". (Vraag: hoe gaan jy die lynnommer bereken?)
- Vir hierdie projek moet die vertaler stop wanneer die eerste fout opgespoor word. Die meeste produksie-vertalers sal verskeie foute probeer opspoor, maar dit vereis heelwat ekstra werk.
- Jou boodskappe behoort te wys jy is 'n professionele persoon. Boodskappe soos "I now stop processing this rubbish!" is onaanvaarbaar.