コンパイラ理論 3
字句解析と構文解析

櫻井彰人

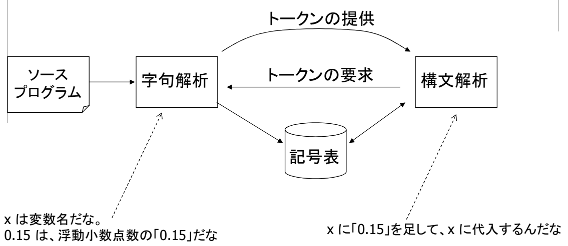
プログラミング言語の定義

- プログラミング言語は次の2階層で定義される。
  - 語彙（単語）とはどういうものか
  - （定義された語彙を用いて）どういう風にプログラムを書くのか？
  - 一気に（つまり、一階層で）定義することもできるのだが、分かり難くなる。

コンパイラの構造も

- その2階層に対応するように、コンパイラも作られる
  - 語彙の認識 ⇔ 語彙の定義
    - 字句解析という
    - 文字の並びをみて、単語に区切る
      - 例えば、変数と定数とを区別する
  - プログラムの認識 ⇔ プログラムの定義
    - 構文解析という
    - 例えば、これは「変数xと変数yの値を足してzに代入する」ことだな、と認識する

字句解析と構文解析



なぜ分けるか?

- 字句解析を構文解析から分ける理由:
  - 「字句」の定義は、正規文法でできる
    - 簡単な道具で済ませられるところは、簡単にすませよう！
  - 設計が単純になる
  - 効率（速度等）の向上が図れる
  - 可搬性がます
- 字句解析・構文解析それぞれによいツールが存在する

トークン・字句・パターン
(Tokens, Lexemes, Patterns)

- トークンは、キーワード（if, for, long,...）、演算子（+,*,...）、識別子、定数、文字列、区切り記号を含む、字句が属すクラスのことをいう
- 字句は、文字のある列であって、ソースプログラム内で意味をもつ最小の単位
- パターンは、（Lexで用いるが）あるトークンの生成規則

トークンの例

プログラム
MODULE HelloWorld;
BEGIN
WriteStr(“Hello World!“)
END HelloWorld .

トークン
MODULE
HelloWorld
;
BEGIN
WriteStr
(
"Hello World!“
)
END
HelloWorld
.

トークンとはプログラムの最小構成要素

文章だと、「単語」に相当
プログラムだと、「トークン」と言われる

「コンパイラ入門 C#で学ぶ理論と実践」より

トークンとは

- トークンは言語仕様の中で定義されている
  - ① ＜規則＞で定義されているもの～識別子、数、文字列
  - ② 予約語～MODULE、INTEGERなど
  - ③ その他の記号～2文字（＜＞、：＝）、1文字（；　．）など
  - BNFで使われているシンボルとの違いに注意

例1　<ident>は規則で定義（先頭が英字で、2文字目以降は英数字）
例2　MODULE, BEGIN, END等、プログラムの予約語
例3　「；」や「．」など埋め込まれている文字

```
<program> →  MODULE <ident> ; [ <decllist> ] BEGIN <statlist> END <ident> .
<ident>   →  <letter> ( <letter> | <digit> )*
<digit>   →  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<letter>  →  a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p |
             q | r | s | t | u | v | w | x | y | z |
             A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
             Q | R | S | T | U | V | W | X | Y | Z
```

語彙とは

- トークンの中で、特別に扱うもの
- ものすごく個数が多いので、「規則」を使って定義
  - 識別子（先頭は英字、2文字目以降は英数字）
  - 数　　（1桁（1文字）以上の数字）
  - 文字列（“と”に囲まれた文字～但し1行以内のもの）

```
<ident>   →  <letter> ( <letter> | <digit> )*                識別子
<integer> →  <digit>+                                         数
<string>  →  "<any character except EOF, EOL and ” >* "       文字列

<digit>   →  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<letter>  →  a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p |
             q | r | s | t | u | v | w | x | y | z |
             A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
             Q | R | S | T | U | V | W | X | Y | Z
```

語彙の定め方

- 予約語として
  - プログラムの中で予め使い方が決められている文字列
  - 予約語は変数名として使えないことが多い

```
<program>    →  MODULE <ident> ; [ <decllist> ] BEGIN <statlist> END <ident> .
<decllist>   →  VAR ( <identlist> : <type> ; )+
<type>       →  INTEGER | STRING
<statement>  →  <ident> := <expression>
                | IF <relation> THEN <statlist> [ ELSE <statlist> ] END
                | WHILE <relation> DO <statlist> END
                | <ident> "(" <literal> ")“
```

語彙の定め方

- その他の記号
  - 区切り記号
  - 演算子。言語によっては英数字を使うことがある。人間が読みやすいように、特殊な記号を用いることが多い

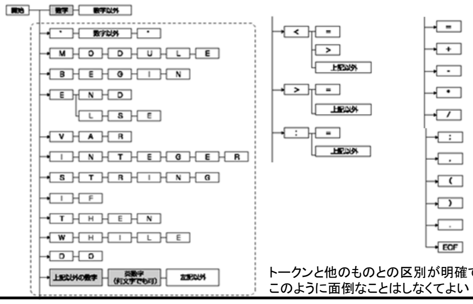
```
<program>    →  MODULE <ident> ; [ <decllist> ] BEGIN <statlist> END <ident> .
<decllist>   →  VAR ( <identlist> : <type> ; )+
<statlist>   →  <statement> ( ; <statement> )*
<identlist>  →  <ident> ( , <ident> )*
<statement>  →  <ident> := <expression>
                | IF <relation> THEN <statlist> [ ELSE <statlist> ] END
                | WHILE <relation> DO <statlist> END
                | <ident> "(" <literal> ")"
<relation>   →  <expression> <rel op> <expression>
<expression> →  <unary op> ] <term> ( <add op> [ <unary op> ] <term> )*
<term>       →  <factor> ( <mul op> <factor> )*
<factor>     →  <literal> | "(" <expression> ")"
<rel op>     →  = | < | <= | <> | > | >=
<unary op>   →  + | -
<add op>     →  + | -
<mul op>     →  * | /
```

トークン一覧例

分類	例	BNF	説明
数値	1, 123	<digit>+	1文字（桁）以上の数字
文字列	"Hello"	" <any…and> "	" "で囲まれた文字列（文字列にはEOF, EOL, "は含まれない）
識別子と予約語	MODULE	MODULE	MODULEの文字並び
	BEGIN	BEGIN	BEGINの文字並び
	END	END	ENDの文字並び
	VAR	VAR	VARの文字並び
	INTEGER	INTEGER	INTEGERの文字並び
	STRING	STRING	STRINGの文字並び
	IF	IF	IFの文字並び
	THEN	THEN	THENの文字並び
	ELSE	ELSE	ELSEの文字並び
	WHILE	WHILE	WHILEの文字並び
	DO	DO	DOの文字並び
	i variable ENDPROGRAM	<letter> [<letter>|<digit>]*	先頭が英字で始まる1文字以上の英数字
演算子	+	+	加算、符号
	-	-	減算、符号
	*	*	乗算
	/	/	除算
	=	=	比較（Equal）
	<	<	比較
	<=	<=	比較
	>	>	比較
	>=	>=	比較
	<>	<>	比較（Not equal）
	:=	:=	代入
区切り記号	:	:	コロン
	;	;	セミコロン
	,	,	カンマ
	(	(	左括弧
	)	)	右括弧
	.	.	ピリオド

トークン識別チャート

- 構文解析プログラムが書きやすいようにチャート化



トークンと他のものとの区別が明確であれば、
このように面倒なことはしなくてよい

言語の規則の階層１ – 字句

- 字句解析用
- パーサでは使用しない。定義済みとして考える。

```
<ident>    →  <letter> ( <letter> | <digit> )*
<integer>  →  <digit>+
<string>   →  "<any character except EOF, EOL and " >* "

<digit>    →  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<letter>   →  a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p |
              q | r | s | t | u | v | w | x | y | z |
              A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
              Q | R | S | T | U | V | W | X | Y | Z
```

言語規則の階層２ – 構文

```
<program>     →  MODULE IDENT ; [ <decllist> ] BEGIN <statlist> END IDENT .
<decllist>    →  VAR ( <identlist> : <type> ; )+
<statlist>    →  <statement> ( ; <statement> )*
<identlist>   →  IDENT ( , IDENT )*
<type>        →  INTEGER | STRING
<statement>   →  IDENT := <expression>
                 | IF <relation> THEN <statlist> [ ELSE <statlist> ] END
                 | WHILE <relation> DO <statlist> END
                 | IDENT "(" <literal> ")"
<relation>    →  <expression> <rel op> <expression>
<expression>  →  [ <unary op> ] <term> ( <add op> [ <unary op> ] <term> )*
<term>        →  <factor> ( <mul op> <factor> )*
<factor>      →  <literal> | "(" <expression> ")"
<literal>     →  IDENT | NUMBER | STR
<rel op>      →  = | < | <= | <> | > | >=
<unary op>    →  + | -
<add op>      →  + | -
<mul op>      →  * | /
```