コンパイラ理論 3
字句解析と構文解析

櫻井彰人

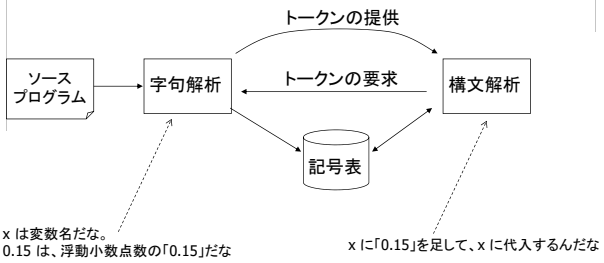
プログラミング言語の定義

- 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

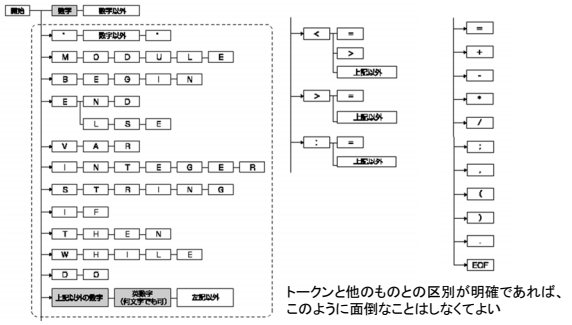
<program> → MODULE <ident> ; [<decllist>] BEGIN <statlist> END <ident> .
<decllist> → VAR (<identlist> : <type> ;)+
<statlist> → <statement> (; <statement>)*
<identlist> → <ident> (, <ident>)*
<statement> → <ident> := <expression>
| IF <relation> THEN <statlist> [ELSE <statlist>] END
| WHILE <relation> 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
	variable 例：PROGRAM	<letter> [<letter>|<digit>]+	先頭の英字で始まる 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
	(	(	各括弧
	)	)	各括弧
	.	.	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。

EBNF での書き方

<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> → * | /

3章　構文解析

- BNFと文法
- BNFとEBNF
- 言語仕様
- プログラムと言語仕様との関係

「コンパイラ入門 C#で学ぶ理論と実践」より

3.2 BNF（Backus Naur Form)

- BNF
  - 「文法」を記述する表記法
  - コンピュータ言語を表す為に使われることが多い
- 英文法
  - 単語と単語の構成・関係を表す
  - 5文型は単語の品詞から英文の型を表現している
- プログラム言語の文法
  - プログラムの最小構成要素の構成・関係を表す
  - 変数、キーワード、オペレータなどの関係
  - 代入文の①abc=123、②123=abc、どちらが正しい？

3.2 BNFの定義

- BNF
  - ターミナル（終端記号）
  - ノンターミナル（非終端記号）で< >と表記する
  - 左辺と右辺はターミナルとノンターミナルの集合体
  - 左辺 ::= 右辺
    - 本書では左辺はノンターミナルだけに制限する
- 例題
  - ＜文＞ ::= ＜主語＞ ＜動詞＞ ＜目的語＞
  - ＜主語＞ ::= I
  - ＜動詞＞ ::= Love
  - ＜目的語＞ ::= You
- 「::=」は置き換えるという意味、以後「→」を使用

3.2 BNFの定義

- BNFを用いて出来ること
  - 文法に合致した文字列を生成（導出という）すること
  - 文字列が文法に合致しているかどうかを「識別」すること
- 例題
  - ステップ1:「先頭」と定義したものから開始
    ＜文＞
  - ステップ2: ＜文＞を＜主語＞＜動詞＞＜目的語＞によって置き換えよう
    ＜主語＞＜動詞＞＜目的語＞
  - ステップ3: ＜主語＞は I によって置き換えよう
    I ＜動詞＞＜目的語＞
  - ステップ4: ＜動詞＞は Love によって置き換えよう
    I Love ＜目的語＞
  - ステップ5: ＜目的語＞は You によって置き換えよう
    I Love You
  - ステップ6: ＜文＞から **I Love You** を生成した

3.2 BNF例: 数

- 数
  - 1文字以上の数字からなる連なり
- BNF
  - <number> → <digit> <number1>
  - <number1> → ε
    | <digit> <number1>
  - <digit> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
  - 注: | （OR）は選択を示す（次スライド参照）
- 生成（識別）できる数の例
  - 1、1、2、3、5、8、13、21・・・
- 生成（識別）できない数の例
  - -123（マイナスは定義されていない）、abc(数字ではないものあり）

3.2 BNF例: 数

- BNF
  - <number> → <digit> <number1>
  - <number1> → ε | <digit> <number1>
  - <digit> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
- BNF（上記の記述と同義, 最も正統的な書き方）
  - <number> → <digit> <number1>
  - <digit> → 0
  - <digit> → 1
  - <digit> → 2
  - <digit> → 3
  - <digit> → 4
  - <digit> → 5
  - <digit> → 6
  - <digit> → 7
  - <digit> → 8
  - <digit> → 9
  - <number1> → ε
  - <number1> → <digit> <number1>

3.2 BNF例: 数

- 構文解析木
  - 開始から生成完了！に到るBNF（右辺と左辺）を表したもの
- 1を識別した場合の構文解析木
  - <number> → <digit> → **1**
    <number1> → ε
- 13の識別した場合の構文解析木
  - <number> → <digit> → **1**
    <number1> → <digit> → **3**
    <number1> → ε
- ε（エプシロン）の意味
  - 空文字列。空白ではなく、本当に「ない」ことを表す
  - 生成時には何も生成されないし、識別時には、その場所に何もなくてよいことを表す

3.2 BNF例: 数

- 構文解析木の表記について
  - 木としての形が一致していればどんな形式でもかまわない
  - 下記の解析「木」は同じ意味
- 横書きの場合（13を生成した場合）
  - <number> → <digit> → 1
    <number1> → <digit> → 3
    <number1> → ε
- 縦書きの場合（13を生成した場合）

<number>
↓
<digit> <number1>
↓ ↓
1 <digit> <number1>
↓ ↓
3 ε

3.2 BNF例: 識別子

- 識別子
  - 1文字以上の文字の列で色々な名前として使用される。プログラムでは識別子と呼ばれる（数字を含んでいいのだが、以下では、簡単のため、数字なし）
- BNF
  - <ident> → <letter> <ident1>
  - <ident1> → ε
    | <letter> <ident1>
  - <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
- 生成できる識別子の例
  - a、ab, abc, xyz, Hello・・
- 生成できない識別子の例
  - 123、abc123（数字は用いないと、定義している）

3.2 BNF例: 識別子

- Abを生成
  - ステップ1: **<ident>**
  - ステップ2: <ident>→**<letter>** **<ident1>**
  - ステップ3: <ident>→<letter>→**A** <ident1>
  - ステップ4: <ident>→<letter>→A <ident1>→**<letter>** **<ident1>**
  - ステップ5: <ident>→<letter>→A <ident1>→<letter> →**b** <ident1>
  - ステップ6: <ident>→<letter>→A <ident1>→<letter> →b <ident1>→**ε**

3.3 BNFとEBNF

- EBNF
  - Extended BNF(拡張されたBNF)
  - BNFよりコンパクトに記述できる
- ( )によるグルーピング
  - まとめて書ける
- ＊による0回以上の繰り返し
  - 再帰呼び出しが省略できる場合がある
- ＋による1回以上の繰り返し
  - 再帰呼び出しが省略できる場合がある
- [ ]により択一の選択
  - εを使わずに書ける。まとめて書ける

3.3 EBNF: ( )と＊

- BNF(識別子)
  - <ident> → <letter> <ident1>
  - <ident1> → ε
    → <lettter> <ident1>
    → <digit> <ident1>
  - <letter> → 前例と同じ(英小文字、英大文字)
  - <digit> → 前例と同じ(0～字、数値)
- 同等のEBNF
  - <ident> → <letter> (<letter> | <digit>)*
- ( )の効果
  - (<letter> | <digit>)によって2つの非終端記号をまとめて記述
- ＊の効果
  - (<letter> | <digit>)*により<ident1>の再帰呼び出しが不要

3.3 EBNF: [ ]

- BNF
  - <program> → MODULE <ident> ; <additional> BEGIN ･･･END <ident> .
  - <additional>→ ε
    → <decllist> (変数宣言の仕方を記述する非終端記号)
- 同等のEBNF
  - <program> --> MODULE <ident> ; [<decllist>] BEGIN ･･･END <ident> .
- 例題1: 変数宣言が無いということと[ ]内が選択されなかったことが対応する
  - MODULE PROGRAM;
  - ～ここに変数宣言が無い～
  - BEGIN
    ･･･
- 例題2: 変数宣言があるということは[ ]内が選択されたということ
  - MODULE PROGRAM;
    VAR I, J, K : INTEGER;
  - BEGIN
    ･･･

3.3 EBNF: ＋

- BNF
  - <decllist> → VAR <decilist1>
  - <decllist1> → <identlist> : <type> ; <decllist2>
  - <decllist2> → ε
    → <decilist1>
- 同等のEBNF
  - <decllist> → VAR (<identlist> : <type> ;)+
- 例題1: <decllist>が＋により1回選択された場合(前ページの例題2).
- 例題2: 変数宣言は1回以上何回書いてもよい(この例では3回)
  - MODULE PROGRAM;
    VAR I, J, K : INTEGER;
    VAR a, b : INTEGER;
    VAR z, z, x, y, z : INTEGER;
    ･･･
    BEGIN
    ･･･

3.4 言語仕様＝プログラムの設計図

```
<program>   →  MODULE <ident> ; [ <decllist> ] BEGIN <statlist> END <ident> .
<ident>     →  <letter> ( <letter> | <digit> )*
<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> | <integer> | <string>
<integer>   →  <digit>+
<rel op>    →  = | < | <= | <> | > | >=
<unary op>  →  + | -
<add op>    →  + | -
<mul op>    →  * | /
<digit>     →  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<string>    →  " <any character except EOF, EOL and "> "
<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
```

3.4 言語仕様の説明

- <program>は変数定義と文から構成される
  - <program> → MODULE <ident> ;
    [<decllist>]
    BEGIN
    <statlist>
    END <ident> .
  - <decllist> → VAR (<identlist> : <type> ;)+
  - <identlist> → <ident> (, <ident>)*
  - <type> → INTEGER | STRING
  - <statlist> → <statement> (; <statement>)*
  - <statement>→ <ident> := <expression>
    | IF <relation> THEN <statlist> [ELSE <statlist>] END
    | WHILE <relation> DO <statlist> END
    | <ident> "(" <literal> ")"

変数定義
例 VAR I,J,K: INTEGER;

文
例 WriteStr("HelloWorld!")

3.4 言語仕様の説明

- <ident>は識別子 ～ 先頭が英字で2文字目以降は英数字
  - <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

普通の識別子定義

- <string>は " と " で囲まれた文字、但し、EOF, EOL, " は除く
  - <string> → " <any character except EOF, EOL and "> "
  - EOL(End Of Line): 2行にまたがれない
  - EOF（End Of File): ファイルの終わりまで継続できない
  - 例: "Hello World!", "Input N>", etc
- <integer>による数値～1桁以上の数値
  - <integer> → <digit>+
  - <digit> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

3.4 言語仕様の説明（2/3）

- <unary op>は符号
  - <unary op>→ + | -

符号は演算子として定義するのが普通

- <add op>は加減演算
  - <add op> → + | -
- <mul op>は乗除演算
  - <mul op> → * | /
- 比較演算子
  - <rel op> → = （EQ）
    → < （LT）
    → <= （LE）
    → <> （NE）
    → > （GT）
    → >= （GE）

3.4 言語仕様の説明（3/3）

- <literal>は<ident>か<integer>か<string>
  - <literal> → <ident> | <integer> | <string>

「リテラル」とは字面通りとか書かれた通りといった意味。意味のある最小単位というものを呼ぶときによく用いられる言葉である

- <expression>は符号と加減乗除付の<literal>の式（再帰的に定義）
  - 例1：1+2*3 （加算、乗算）
  - 例2：-1+-2*-3 （符号付）
  - 例3：(1+2)*3 （括弧付の式～加算優先）
  - 例4：(((1)))
  - <expression>→ [<unary op>] <term> (<add op> [<unary op>] <term>)*
  - <term> → <factor> (<mul op> <factor>)*
  - <factor> → <literal> | "(" <expression> ")"
- <relation>は比較式～式と式を比較演算子で結合
  - 例：123<234
  - <relation> → <expression> <rel op> <expression>

3.5 言語仕様とプログラム

プログラム

```
MODULE HelloWorld;
BEGIN
    WriteStr ( "Hello World!" )
END HelloWorld .
```

プログラムと言語仕様の対応

```
MODULE <ident> ;
BEGIN
    <statlist>→<statement>→<ident> (  <literal> )
END <ident> .
```

相当する言語仕様 (該当する部分のみ)

```
<program>   → MODULE <ident> ; [ <decllist> ] BEGIN <statlist> END <ident> .
<statlist>      → <statement> ( ; <statement> )*
<statement> →  <ident> "(" <literal> ")"
```