5.2 ＣＩＴ（Ver0.9.0）構文規則

以下は、CIT構文規則となる。図5.1が範囲、定数、その他の定義で、図5.2が、CIT構文規則の記号定義である。C言語とCITの型定義の違いを示した例が図5.3となる。

```
<program>          →   <devers> <defuns>

<devers>           →   { <struct> | <union> | <dever> }

<defuns>           →   { <defun> }

<defun>            →   (DEFUN  <funcname>  <type>
                        <defgs>
                        <funcblock>
                       )

<funcname>         →   <identifier>

<funcblock>        →   "{" <devers>  <statementlist>  "}"

<block>            →   "{"   <statementlist>  "}"

<statementlist>    →   { <statement> }

<statement>        →   <if> | <switch> | <while> | <dowhile> | <for> |
                       <expression> | <return> | <break> | <continue> |
                       <label> | <goto>

<expression>       →   <op>

<struct>           →   (STRUCT <usertype> {<dememberr>}1 )

<union>            →   (UNION <usertype> {<dememberr>}1 )

<usertype>         →   <identifier>

<dememberr>        →   (<memname>  [ <type> | <bittype> ] )

<memname>          →   <identifier>

<dever>            →   (DEVER <varname> <type> <initval>opt)

<varname>          →   <identifier>

<defgs>            →   { <defg> }

<defg>             →   (DEFG <varname> <type>)

<type>             →   {  "()" | "*" | <arytype>  }
                       {  <usertype> | <typename>  }

<arytype>          →   "[" <arysize>opt  "]"
```

<bittype> → int : <bitsize>

<typename> → void | char | int | float | double

<fullvarname> → <varname> { <next> }

<next> → <suffix> | "*" | "&" | <nextmember>

<nextmember> → "." <memname>

<suffix> → "[" {<digit>} "]"

<initval> → <constant> | <initvals> | <funcname>

<initvals> → "{" <initval> {<delimiter> <initval> } "}"

<callfunc> → (<funcname> {<src>}) |
(<fullvarname> {<src>})

<if> → (IF <expression> <block> $\langle elseblock \rangle_{opt}$)

<elseblock> → (ELSE <block>)

<switch> → (SWITCH <fullvarname> { <case> | <default> })

繰り返しについて、<case>は、1つ以上となる。<default>は、1つ以下となる

<case> → (CASE <constant> <statementlist>)

<default> → (DEFAULT <statementlist>)

<while> → (WHILE <expression> <block>)

<dowhile> → (DOWHILE <block> <expression>)

<for> → (FOR <expression>$_A$ <expression>$_B$
<expression>$_C$
<block>
)

<return> → (RETURN <src>$_{opt}$)

<break> → (BREAK)

<continue> → (CONTINUE)

<label> → (LABEL <labelname>)

<labelname> → <identifier>

<goto> → (GOTO <labelname>)

```
<assignment>      →   [( T(=) <dst> <src> ) ]  |
                      [( T(=) <dst> <funcname> ) ]

<cast>            →   (CAST <src> <typename>)

<src>             →   <fullvarname> | <constant>  | <callfunc> | <op>

<dst>             →   <fullvarname>

<op>              →   <1op> | <2op> | <3op>  | <src>

<1op>             →   (T(POST++) <dst> ) | (T(POST--) <dst> ) |
                      (T(PRE++) <dst> ) | (T(PRE--) <dst> ) |
                      (T(!) <dst> ) | (T(~) <dst> )

<2op>             →   <assignment> | <cast> |
                      (T(+) <op>_L <op>_R ) | (T(-) <op>_L <op>_R ) |
                      (T(*) <op>_L <op>_R ) | (T(/) <op>_L <op>_R ) |
                      (T(%) <op>_L <op>_R ) |
                      (T(<<) <op>_L <op>_R ) | (T(>>) <op>_L <op>_R ) |
                      (T(==) <op>_L <op>_R ) | (T(!=) <op>_L <op>_R ) |
                      (T(<) <op>_L <op>_R ) | (T(>) <op>_L <op>_R )   |
                      (T(<=) <op>_L <op>_R ) | (T(>=) <op>_L <op>_R ) |
                      (T(&&) <op>_L <op>_R ) | (T(||) <op>_L <op>_R ) |
                      (T(&) <op>_L <op>_R ) |
                      (T(|) <op>_L <op>_R ) | (T(^) <op>_L <op>_R )

<3op>             →   ( T(?) <op> <fullvarname>_A <fullvarname>_B )

<identifier>      →   <alpha>  { <alpha> | <digit> }

<delimiter>       →   " " | "\t" | "\r\n"

<alpha>           →   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 |
                      _

<digit>           →   0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<constant>        →   <iconstant> | <cconstant> | <strconstant> |
                      <fconstant> | <dconstant>

<strconstant>     →   \" {<ascii_20_7e>}  \"

<cconstant>       →   ' <ascii_20_7e> '
```

<arysize>	1～65535 の数値
<bitsize>	1～32 の数値
<iconstant>	-2147483647 から 2147483647の範囲の数値。0xをつけることにより16進数での指定可能
<fconstant>	1.175494e-38から3.402823e+38の範囲の数値。明示的な指定は数値の末尾に ”f” を追加
<dconstant>	2.225074e-308から1.797693e+308の範囲の数値
<ascii_20_7e>	ASCIIのコード範囲0x20～0x7Fのうちどれか

図5.1 範囲、定数、その他の定義

x→r	:	x は rで置き換えられる
(x y z)	:	( ) は終端記号であるが、囲まれた記号間には、スペース、タブ、改行のうちどれかが一つ以上あるものとする。
x y	:	記号列xの後ろに記号列yを置いた記号列(連接)
x | y	:	記号列xまたは記号列y(選択)
{x}	:	記号列xの0回以上の繰り返し(反復)
$\{x\}_1$	:	記号列xの1回以上の繰り返し(反復)
x_{opt}	:	x | ε (換言すれば 記号列xは、指定しない場合がある。)
x_C	:	記号列xは、Cという特定のxである。
<n>	:	非終端記号
"t"	:	" で囲まれた記号 t は、終端記号となる
\"	:	" が終端記号となる
[x y]	:	[]で囲まれた部分をひとつにまとめてあつかうものとする
ε	:	空記号

図5.2 CIT構文規則の記号定義

```
//
// 文献[KR88 ,p122]に、記述されているものを、ほぼそのまま記述した
// C言語ソースとなる。文献[KR88 ,p122-126]は、CIT型定義の参考とした。
//
main()
{

 char **argv;         // argv: pointer to pointer to char

 int (*daytab1) [13]; // daytab: pointer to array[13] of int

 int *daytab2[13];    // daytab: array[13] of pointer to int

 void *comp1();       // comp: function returning pointer to void

 void (*comp2)();     // comp: pointer to function returning void

 char (*(*x1())[])();
  // x: function returning pointer to array[] of pointer to
  //    function returning char

 char (*(*x2[3])())[5];
  // x: array[3] of pointer to function returning pointer to
  //    array[5] of char

}
```

```
(DEFUN     main  void
 {
  (DEVER argv    **char)
  (DEVER daytab1 *[13]int)
  (DEVER daytab2 [13]*int)
  (DEVER comp1   ()*void)
  (DEVER comp2   *()void)
  (DEVER x1     ()*[]*()char)
  (DEVER x2     [3]*()*[5]char)
  }
)
```

図5.3 C言語とCITの型定義の違い