

アジェンダ

1. 改造部分の解説

1.1 パーシング処理

1.2 アクセッサの導入

1.3 バインド処理

2. 言語仕様検討事項

1. 改造部分の解説

1.1 パージング処理

1.2 アクセッサの導入

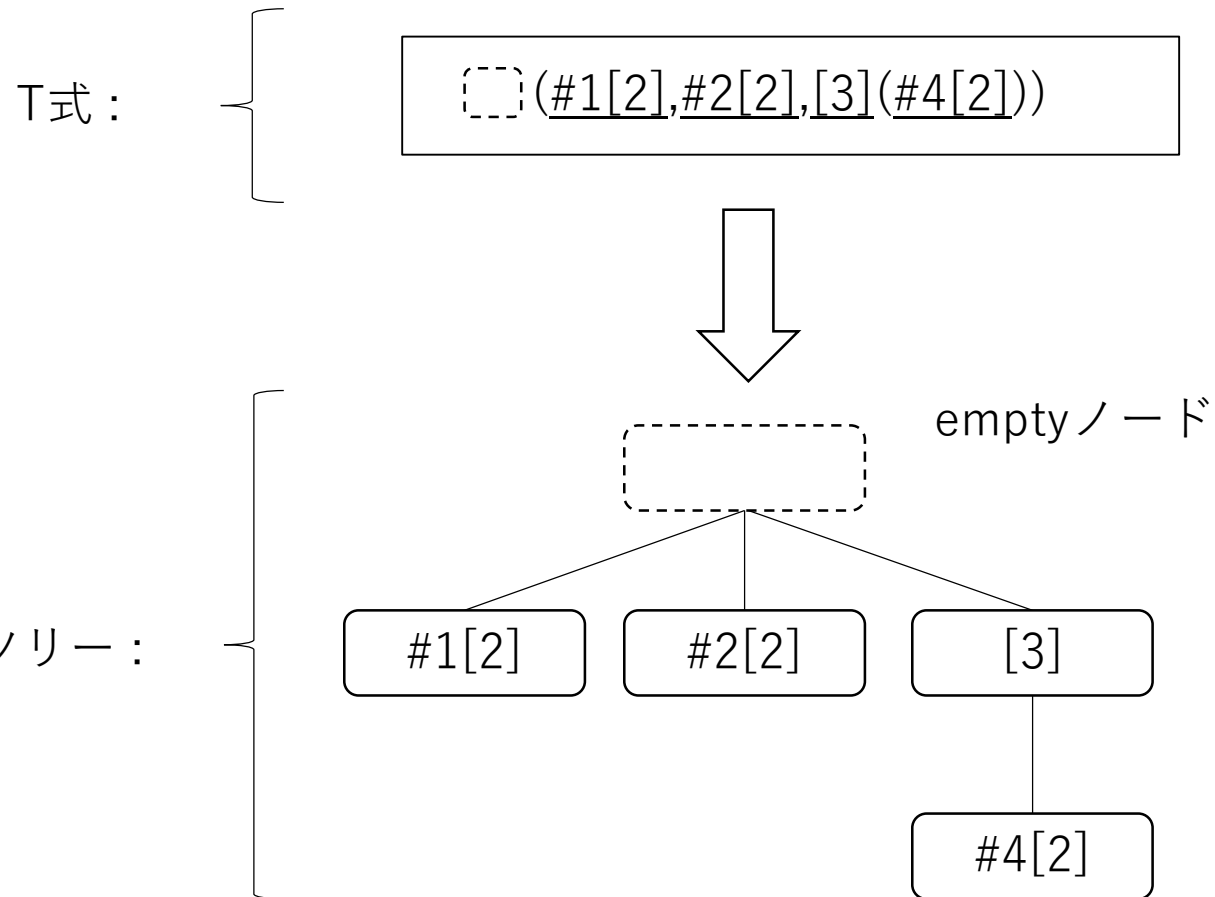
1.3 バインド処理

2. 言語仕様検討事項

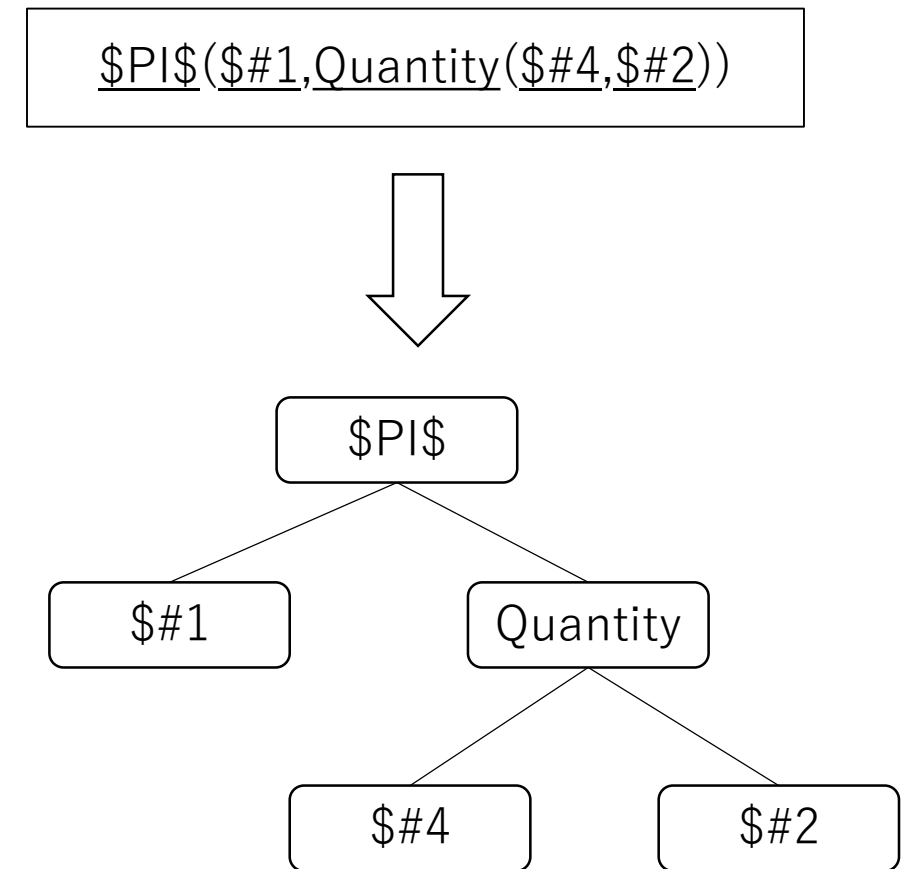
◎ パージングとは

- ・ T式 : tq言語による記述
- ・ パージング : T式からツリーを作成

(例1)



(例2)



◎ T式の文法(BNF記法)

- (1) $\langle \text{header} \rangle ::= \langle \text{empty} \rangle \mid \langle \text{name-str} \rangle$
(2) $\langle \text{T-exp} \rangle ::= \langle \text{header} \rangle \{ "(" [\langle \text{T-exp} \rangle \{ "," \langle \text{T-exp} \rangle] ")" \}$

$\langle \text{empty} \rangle ::=$ 空文字列

$\langle \text{name-str} \rangle ::=$ デリミタ以外の文字列

(凡例)

“...”: terminal

$\langle \rangle$: nonterminal

$\{ \}$: 0回以上

$[]$: 0回または1回

例

- ① A // header
- ② A(B,C(D)) // T-expの入れ子構造
- ③ (A,B,C) // 先頭にempty header
- ④ A(,D) // ,の前に //
- ⑤ 2(A,3) // 数字のみのheaderも可
- ⑥ A(B,C)(D) // Aの後に()の繰り返し

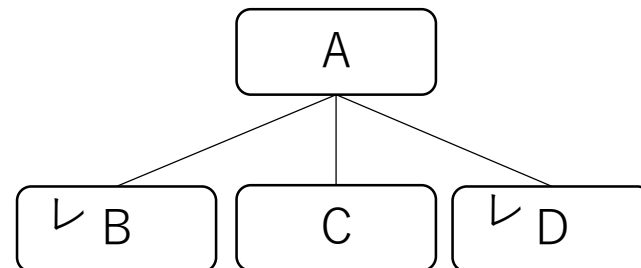
※⑥について

fは2変数関数であり、2つのパラメータ値を与えることで1変数関数になるように定義されているとする(例えば、 $f(x, y)(z) = x+y+z$)。

fをコールする記法は次のようになる。

$f(2,3)(3)$

レ: 先頭子ノード(conjugate flag)



◎ パーシング処理

- (1) <header> ::= <empty> | <name-str>
(2) <T-exp> ::= <header> { "(" [<T-exp> { "," <T-exp> }] ")" }

基本: 1 token 1 character look ahead

・ 処理関数との対応	
<header>	⇔ parse_header()
<T-exp>	⇔ parse_T()
token切出	⇔ next_token() skip()

{
 ■ : 構文要素<T-exp>
 ◆ : EOF文字

- current token
 - ① 構文要素の解析開始時: 構文要素の先頭
 - ② " 終了時: 構文要素の次
- current char
current tokenの次の文字

CharStream : A A A (B B B , C C C (D D D)) ◆

TokenStream : | (| , | (|)) E

current token	/ "CCC")
current character	(EOF

token一覧	
"I"	名前文字列(identifier)
"("	
")"	
","	
"E"	EOF

◎ ストリーム構造体(導入予定)

```
(a) struct CharStream {           // character stream
        FILE*    fp;              // source file ptr
        char     ch;              // current char
};
```

[主要関数]

- void initialize(struct CharStream* in, FILE* fp) // fpを入力ソースとして文字ストリームを初期化
- char peek(struct CharStream* in) // カレント文字をリターン
- char get(struct CharStream* in) // 1文字読み進めて新たなカレント文字をリターン

```
(b) typedef      char    TOKEN;
    Struct TokenStream {
        CharStream*  in;    // source stream
        Token        token; // current token
        char*        name   // name string of current token
    }
```

[主要関数]

- void initialize(struct TokenStream* in, struct CharStream* source) // inを入力ソースとしてトークンストリームを初期化
- Token token(struct TokenStream* in) // カレントトークンをリターン
- Token next(struct TokenStream* in) // 1トークン読み進めて新たなカレントトークンをリターン
- char* name(struct TokenStream* in) // カレントトークンの文字列をリターン

◎ parse_header()

(1) **<header> ::= <empty> | <name-str>**

(2) **<T-exp> ::= <header> { "(" [<T-exp> { "," <T-exp> }] ")" }**

// return tree for <header>

NODE parse_header(struct TokenStream* in)

{

 NODE node = alloc_node(); // allocate node for this <header>

 if(token(in) != 'I') {

 set_head(node, "");

 } else {

 set_head(node, name(in));

 skip('I', in);

 // set buff to <header> node

 // skip identifier token

 }

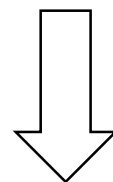
 return node;

}

current token: "I" / "AAA"

T式:

AAA



ツリー:

AAA

node

void skip(TOKEN tk, struct TokenStream* in)

{

 if(tk == token(in)) {

 next(in); // 次のトークンをカレント

 } else {

 error("syntax error");

 }

}

※赤フォントは導入予定

◎ parse_T()

// return tree for <T-exp>

NODE parseT(struct TokenStream* in)

{

 NODE root; // root node of tree for this <T-exp>

 NODE child; // child node of root

 root = parse_header(in); // node for <header>

 // child nodes

 while(token(in) == '(') {

 skip('(', in); // skip '('

 child = parseT(in); // 1st child after '('

 add_child(root, child); // add child to root

 set_conjugate(child, OFF); // mark as 1st child

 while(token(in) == ',') {

 skip(',', in); // skip ','

 child = parseT(in); // 2nd or later child

 add_child(root, child); // add child to root

 set_conjugate(child, ON); // mark as 2nd or later

 }

 skip(')', in); // skip ')'

 }

 return root; // root node of this T-exp

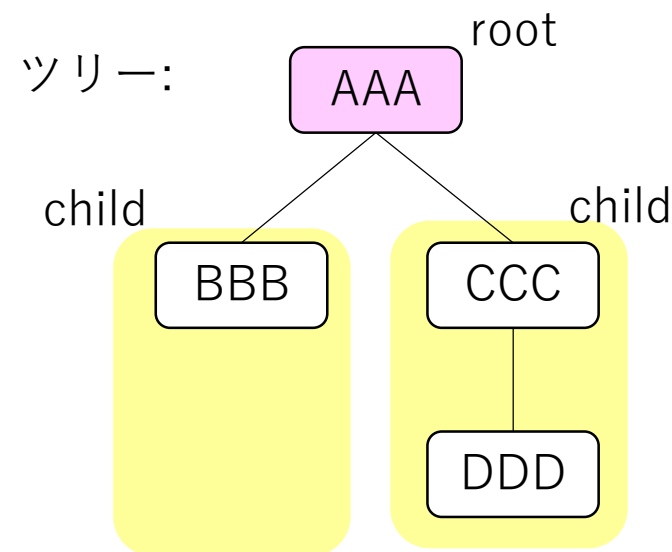
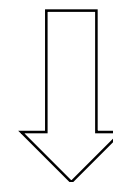
}

(1) <header> ::= <empty> | <name-str>

(2) <T-exp> ::= <header> { "(" [<T-exp> { "," <T-exp> }] ")" }

current token: "I" / "AAA"

T式: AAA(BBB,CCC(DDD))



◎ エスケープ処理

```
void skip_double_quote()  
{
```

```
    do {
```

```
        append_char(ch);           // ①②
```

```
        next_char();
```

```
        while(ch != EOF && ch != '"') {
```

```
            append_char(ch);       // 'a', 'b' / 'c', 'd'
```

```
            next_char();
```

```
        }
```

```
        if(ch != EOF) {
```

```
            append_char(ch);       // ①③
```

```
            next_char();
```

```
        } else {
```

```
            error("syntax error");
```

```
        }
```

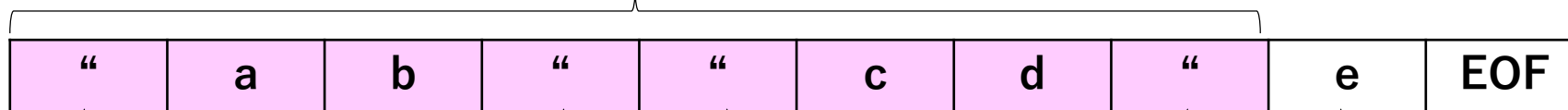
```
    } while(ch == '"');
```

```
    // ②->yes、④->no
```

```
}
```

エラー処理漏

1トークン



①

②

③

④

⑤

1. 改造部分の解説

1.1 パージング処理

1.2 アクセッサの導入

1.3 バインド処理

2. 言語仕様検討事項

◎ ツリーの内部構造(1/2)

メモリ削減対応
=> ツリー表現を変更

ポインタ表現



配列表現

構造体定義

ツリー全体

各ノード

```
typedef int      NODE; // node type
#define NO_NODE -1    // no node

struct LinkTable {
  int offset;
  int node_count; // number of nodes

  // int *ser;
  NODE *parent; // parent
  // int *level;
  // int *child_no;
  char **head; // head string
  int *conjugate; // 1st children / 2nd or later
  // char *label_type;
  // int *label;
  int *indicator_pos;
  char **dimension_str;
  int *value_count;
  int **value_poses;
  char **values_str; // bounded values
  int *child_count; // number of childrens
  NODE **children; // children
  NODE *ref_node; // referenced node
  int *extra_stat;
};
```

※ノード0はnull node

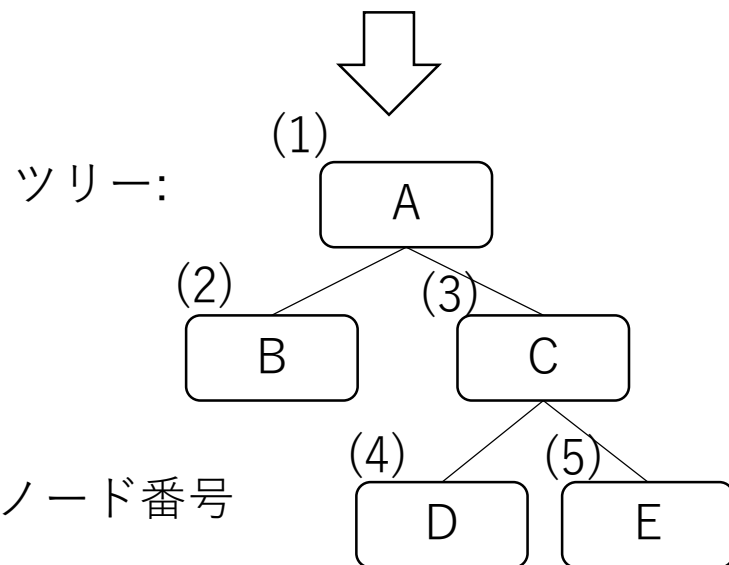
ノード番号:
(配列添字)

(配列添字) 0 1 2 3 4 5 6 7

	-1	1	1	3	3		
	\$NULL\$	A	B	C	D	E	
		OFF	OFF	ON	OFF	ON	
		2	0	2	0	0	

 2 3 4 5

A(B,C(D,E))



※():ノード番号

アクセス例:

```
((*LT).head)[n]
((*LT).parent)[n]
((( *LT).children)[n])[i]
((*LT).conjugate)[n] = ON
```

◎ アクセッサ

LT: LinkTableのアドレスを保持

要求	直コーディング	アクセッサ使用
ノードnのhead文字列	((*LT).head)[n]	head(n)
ノードnのparent	((*LT).parent)[n]	parent(n)
ノードnのi番目のchild	(((*LT).children)[n])[i]	child(n, i)
ノードnのconjugateをON	((*LT).conjugate)[n] = ON	set_conjugate(n, ON)

<定義> (C-functions.h内)

```
static struct LinkTable* LT;

// ith child of node n
NODE child(NODE n, int i)
{
    return (LT->children[n])[i];
}

// set conj flag of node n
void set_conjugate(NODE n, int conj)
{
    (LT->conjugate[n]) = conj;
}
```

<使用例>

```
ノードnをルートとするツリーの全ノードについて
head stringを表示する

int ptint_heads(NODE n)
{
    print_string(head(n));           // print self

    for(i=0; i<child count(n); i++) {
        print_heads(child(n, i));
    }
}
```

1. 改造部分の解説

1.1 パージング処理

1.2 アクセッサの導入

1.3 バインド処理

2. 言語仕様検討事項

◎ バインド処理とは

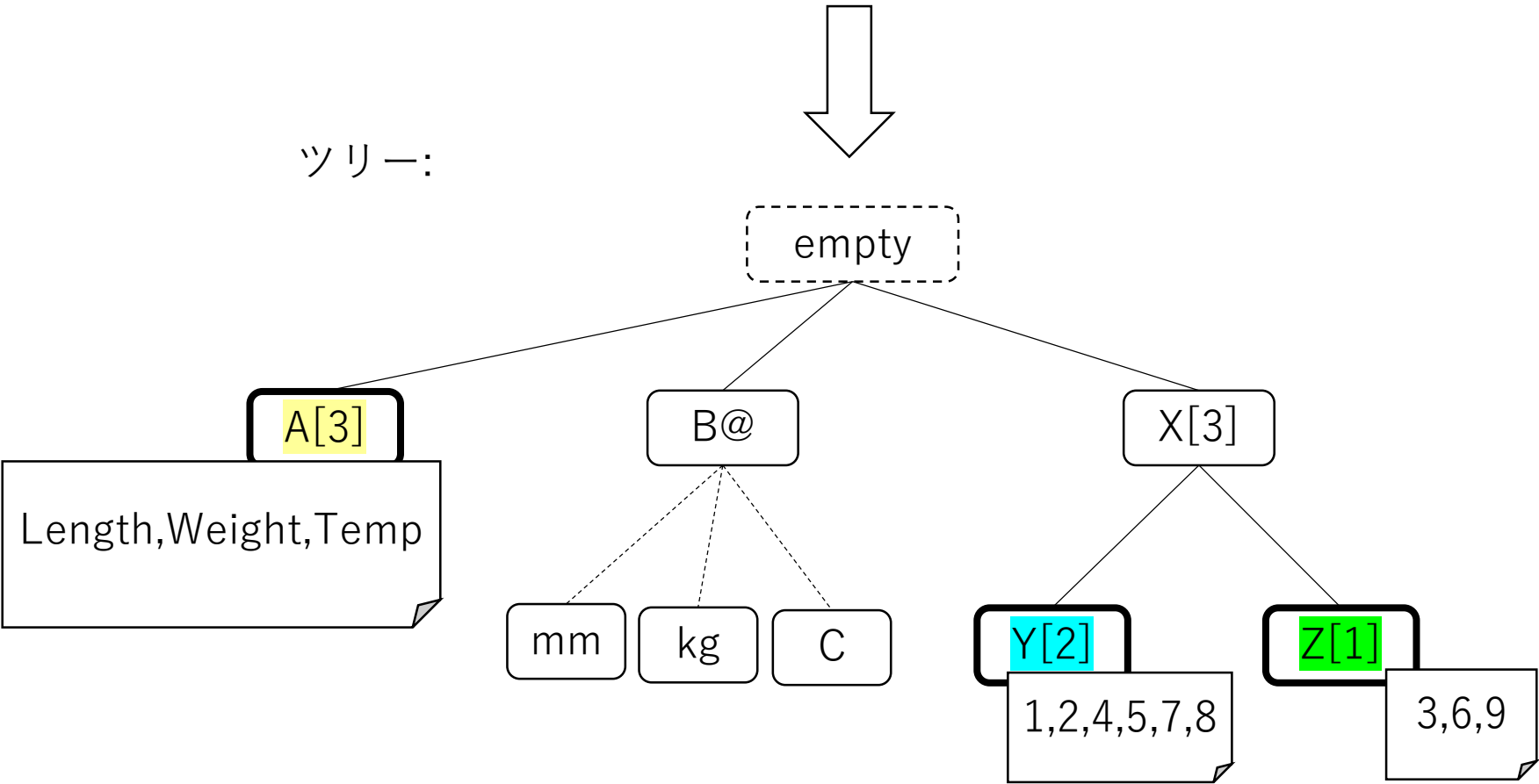
in(T式): (A[3], B@(mm,kg,C), X[3](Y[2], Z[1]))

data(CSV): Length,Weight,Temp,1,2,3,4,5,6,7,8,9

※2次元以上や任意回繰返も指定可

- ・ X[2,3,5] 3次元
- ・ X[] 任意回繰返 (現状1か所のみ可)

ツリー:





<値の割当順序>

ノード		値
X[3] (1回目)	Y[2]	1,2
	Z[1]	3
X[3] (2回目)	Y[2]	4,5
	Z[1]	6
X[3] (3回目)	Y[2]	7,8
	Z[1]	9

※ノード間リンク2種類

- ・ _____ : 親子関係
- ・ : 属性関係?

 : [~]付leaf(CSVデータのバインド対象)  : ノードにバインドされた値

◎ バインド処理のロジック

```
void bind_node(Node node, Stream* in)           // in: CSVファイル
{
    int num = array_size(node);                 // ノードnodeの配列サイズ、配列指定なしは-1;
                                                // (例)  A[3]  -> 3、 B -> -1
    if(is_leaf(node) && num == -1) {
        return;                                // leafであって配列指定なしの場合読み込みなし
    } else {
        for(int i=0; i<num; i++) {              // 配列サイズ分繰返す
            bind_primitive_node(node, in);
        }
    }
}
```

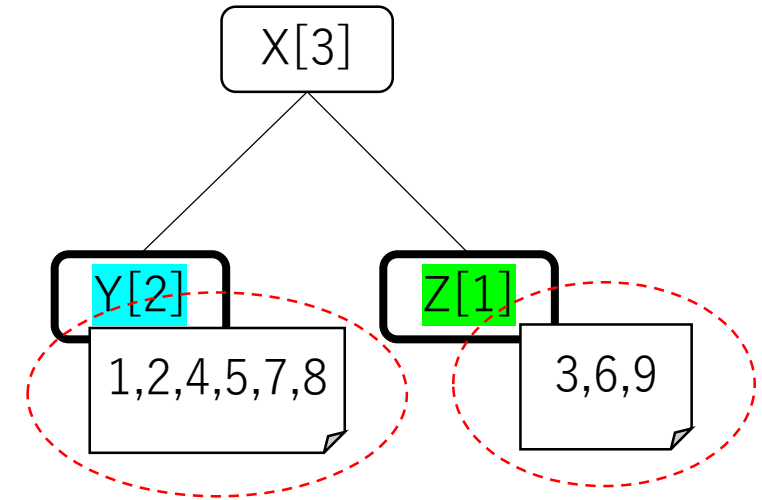
```
void bind_primitive_node(Node node, Stream* in)  // in: CSVファイル
{
    if(is_leaf(node)) {
        bind(node, in, dim);                    // inからdim個のデータを読み込んでnodeにバインド
    } else {
        for(int i=0; i<child_count(node); i++) {
            bind_node(child(node, i), in);
        }
    }
}
```

※2次元以上や任意回繰返の処理は省略

◎ バインドの2パス処理

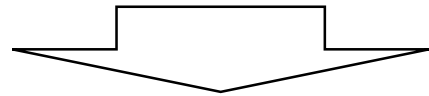
data(CSV): Length,Weight,Temp,1,2,3,4,5,6,7,8,9

値保持用のメモリ領域



(課題)メモリ領域のサイズが処理終了まで決定できない

- ・最初に大きめな領域確保 => メモリ量が無駄
- ・CSV値読み込みごとにメモリ領域割り当て => 性能上インパクト大



2パス方式

- ・1パス目: CSV値を読み込んで必要なメモリサイズを決定し、領域確保
- ・2パス目: CSV値を読み込んで、確保した領域に順次格納

1. 改造部分の解説

1.1 パージング処理

1.2 アクセッサの導入

1.3 バインド処理

2. 言語仕様検討事項

◎ tqとcqとの相違点(1/2)

#	項目	tq	cq	備考
1	CSVファイル中のescape	実装済	未実装	すぐに実装予定
2	[]の扱い	1	無限回	
3	配列指定のbind順序	配列指定数掛算による 最終要素数分	配列指定数分繰返し	
4	CSVデータ不足時*1	bindしない	不足分以外bind	
5	\$PI\$含むoutデータの出力*2	内積の枠	内積結果	

*1:

```
in=test_prd.1.ddf
    $(#1$1[2],#2$2[2],$3[3](#4$4[2]))
data=ssss.csv
    Length,Weight
    mm,kg
    1,2
    322,4
```

*2:次ページ

```
$ ./tq.o in=test_prd.1.ddf data=test_nummatch.csv -FT -C -Pin
=> $($1[2]@(Length,Weight),$2[2]@(mm,kg),$3[3]($4[2]))

$ ./cq.o in=test_prd.1.ddf data=test_nummatch.csv -FT -C -Pin
=> $($1[2]@(Length,Weight),$2[2]@(mm,kg),$3[3]($4[2]@(1,2,322,4)))
```

◎ tqとcqとの相違点(2/2)

```
in=test_prd.1.ddf
    $(#1$1[2],#2$2[2],$3[3](#4$4[2]))

out=test_prd.1.ddl
    $PI$($#1,$#2,$#4)
```

```
$ ./tq.o in=test_prd.1.ddf out=test_prd.1.ddl data=test.csv -FT -C -Pin -Pout -Pprod
=>
($1[2]@(Length,Weight),$2[2]@(mm,kg),$3[3]($4[2]@(1,2,322,4,5,68)))
(((,),(,)),((,),(,)),((,),(,)))
(((Length,mm,1),(Weight,kg,2)),((Length,mm,322),(Weight,kg,4)),((Length,mm,5),(Weight,kg,68)))

$ ./cq.o in=test_prd.1.ddf out=test_prd.1.ddl data=test.csv -FT -C -Pin -Pout -Pprod
=>
($1[2]@(Length,Weight),$2[2]@(mm,kg),$3[3]($4[2]@(1,2,322,4,5,68)))
(((Length,mm,1),(Weight,kg,2)),((Length,mm,322),(Weight,kg,4)),((Length,mm,5),(Weight,kg,68)))
(((Length,mm,1),(Weight,kg,2)),((Length,mm,322),(Weight,kg,4)),((Length,mm,5),(Weight,kg,68)))
```

◎ オペレータ評価の有無(1) tq/cq共通

#	対象	option		項目		
		-C	-Pprod	(a) ref-node	(b) 値bind	(c) \$x\$評価
1	-Pin	*	*	○	○	○
2		*	-			
3		-	*			×
4		-	-			
5	-Pout	*	*	○ (ref先->inのみ)	×	○
6		*	-			
7		-	*			×
8		-	-			

◎ オペレータ評価の有無(2) 内積

#	対象	option		(d) 内積(\$PI\$)	
		-C	-Pprod	tq	cq
1	-Pin	*	*	△	△(不良)
2		*	-	△	△(不良)
3		-	*	×	×
4		-	-	×	×
5	-Pout	*	*	△	○
6		*	-	△	○
7		-	*	×	×
8		-	-	×	×
9	-Pprod	*	*	○	○
10		*	-	-	-
11		-	*	×	×
12		-	-	-	-

```
in=sak-Pl.ddf
    $PI$(#1$1[2],#2$2[2],$3[3](#4$4[2]))
out=test_prd.1.ddl
    $PI$($#1,$#2,$#4)
```

```
$ ./tq.o in=sak-Pl.ddf out=test_prd.1.ddl data=test.csv -FT -Pin -Pout -C -Pprod
(((($1[2],$2[2],$3[3]($4[2])),($1[2],$2[2],$3[3]($4[2]))),(($1[2],$2[2],$3[3]($4[2])),($1[2],$2[2],$3[3]($4[2]))),(($1[2],$2[2],
$3[3]($4[2])),($1[2],$2[2],$3[3]($4[2]))))
(((,),(,)),((,),(,)),((,),(,)))
(((Length,mm,1),(Weight,kg,2)),((Length,mm,322),(Weight,kg,4)),((Length,mm,5),(Weight,kg,68)))
```

```
$ ./tq.o in=sak-Pl.ddf out=test_prd.1.ddl data=test.csv -FT -Pin -Pout -C
(((($1[2],$2[2],$3[3]($4[2])),($1[2],$2[2],$3[3]($4[2]))),(($1[2],$2[2],$3[3]($4[2])),($1[2],$2[2],$3[3]($4[2]))),(($1[2],$2[2],
$3[3]($4[2])),($1[2],$2[2],$3[3]($4[2]))))
(((,),(,)),((,),(,)),((,),(,)))
```

```
$ ./cq.o in=sak-Pl.ddf out=test_prd.1.ddl data=test.csv -FT -Pin -Pout -C -Pprod
(((Length$1[2],mm$2[2],$3[3](1$4[2])),(Weight$1[2],kg$2[2],$3[3](2$4[2]))),((Length$1[2],mm$2[2],$3[3](322$4[2])),(W
eight$1[2],kg$2[2],$3[3](4$4[2]))),((Length$1[2],mm$2[2],$3[3](5$4[2])),(Weight$1[2],kg$2[2],$3[3](68$4[2]))))
(((Length,mm,1),(Weight,kg,2)),((Length,mm,322),(Weight,kg,4)),((Length,mm,5),(Weight,kg,68)))
(((Length,mm,1),(Weight,kg,2)),((Length,mm,322),(Weight,kg,4)),((Length,mm,5),(Weight,kg,68)))
```

```
$ ./cq.o in=sak-Pl.ddf out=test_prd.1.ddl data=test.csv -FT -Pin -Pout -C
(((Length$1[2],mm$2[2],$3[3](1$4[2])),(Weight$1[2],kg$2[2],$3[3](2$4[2]))),((Length$1[2],mm$2[2],$3[3](322$4[2])),(W
eight$1[2],kg$2[2],$3[3](4$4[2]))),((Length$1[2],mm$2[2],$3[3](5$4[2])),(Weight$1[2],kg$2[2],$3[3](68$4[2]))))
(((Length,mm,1),(Weight,kg,2)),((Length,mm,322),(Weight,kg,4)),((Length,mm,5),(Weight,kg,68)))
```