Bison について

Bison は BNF とそれに対する動作記述から、 C 言語の構文解析系（パーサー）を自動生成するプログラムです。 Bison のソースファイル（通常 `.y` という拡張子をつける）は、次のように書きます。（これは通常の四則演算の式の文法です。この例では単項の「-」はサポートしていないので、 `-1+2` や `2*(-3)` のような式は構文解析できません。単項の「-」を扱う方法は yacc (bison) のマニュアルを調べて下さい。）

ファイル calc.y

```
%{
 /* C declarations */
#define YYSTYPE double
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
void yyerror(char* s) {
   printf("%s\n", s);
}

int yylex(void);
%}
 /* Bison declarations */
%token NUMBER
%left '+' '-'
%left '*' '/'

%%
 /* grammar rules */
input   :               {}
        | input line    {}
        ;
line    : '\n'          { exit(0); }
        | expr '\n'     { printf("\t%g\n", $1); }
        ;
expr    : NUMBER        { $$ = $1; }
        | expr '+' expr { $$ = $1 + $3; }
        | expr '-' expr { $$ = $1 - $3; }
        | expr '*' expr { $$ = $1 * $3; }
        | expr '/' expr { $$ = $1 / $3; }
        | '(' expr ')'  { $$ = $2; }
        ;
%%
 /* additional C code */
int yylex(void) {
  int c;

  do {
    c = getchar();
  } while (c == ' ' || c == '\t');

  if (isdigit(c) || c == '.') {
    ungetc(c, stdin);
    scanf("%lf", &yylval);
    return NUMBER;
```

```
46    } else if (c == EOF) {
47      return 0;
48    }
49    return c;
50  }
51
52  int main(void) {
53    printf("Exit with Ctrl-c\n");
54    yyparse();
55    return 0;
56  }
```

説明

最初の C 宣言部（C declarations というコメントの部分、「%{」と「%}」の間）には後述の動作記述の中で用いる関数の定義や宣言を書きます。 特に YYSTYPE という定数マクロには属性値（意味値）の型を指定します。ただし、属性値の型が int の場合はこのマクロの定義は必要ありません。

この例では exit 関数や isdigit 関数を使用するため、それぞれ stdlib.h, ctype.h というヘッダーをインクルードしています。また、エラーがあったときに呼ばれる関数として`void yyerror(char* s)` を定義しておきます。（この例では単に printf を呼び出しています。） また、yylex 関数は下で定義しているため、ここでプロトタイプ宣言してあります。

Bison 宣言部（Bison declarations というコメントの部分、最初の「%%」まで）には終端記号（トークン）（と非終端記号）に関する宣言を書きます。 %token はトークンを宣言します。トークンは出力の C プログラムでは定数マクロとして定義されます。 **下の優先順位の宣言や文法規則部でトークンとして使えるのはここで定義したマクロか文字リテラルです。**

Bison 宣言部には演算子の優先順位と結合性を宣言することもできます。 %left, %right, %nonassoc のいずれかのあとに演算子を表すトークンを空白で区切って並べて書きます。%left は左結合、%right は右結合、%nonassoc は非結合を表します。また、優先順位が高い演算子ほど下に書きます。同じ行に書かれている演算子は優先順位は同じです。上の例では、「+」「-」「*」「/」はすべて左結合で、「*」/」が「+」「-」よりも優先順位が高い、と宣言されています。

文法規則部（grammar rules というコメントの部分）は Bison の核心部分で、最初の「%%」から 2 つめの「%%」までが文法規則部です。 文法規則部には BNF とそれに付随するアクションを書きます。 BNF は教科書などでよく使われる記法の右矢印「→」の代わりにコロン「:」を使っていることに注意してください。また各 BNF の最後にセミコロン「;」を書きます。

特に指定しなければ開始記号 (start symbol) に関する BNF を最初に書きます。

アクションは**還元時**に実行されるプログラムのことです。アクションの中身は通常**属性値**（意味値）の計算です。属性値は解析木の各節（枝分れの部分）に関連付けられる“値”です。

生成規則の中では、終端記号（トークン）は、１文字からなるトークンの場合は通常、 **文字リテラル**そのまま、２文字以上からなるトークンの場合は `%token` で宣言された**マクロ**で表します。 Flex で生成される `yylex` 関数はトークン（文字リテラルまたはマクロ）を返します。

終端記号（トークン）の属性値は、字句解析器（`yylex` 関数）から **`yylval`** という大域変数に代入されて受け渡されます。

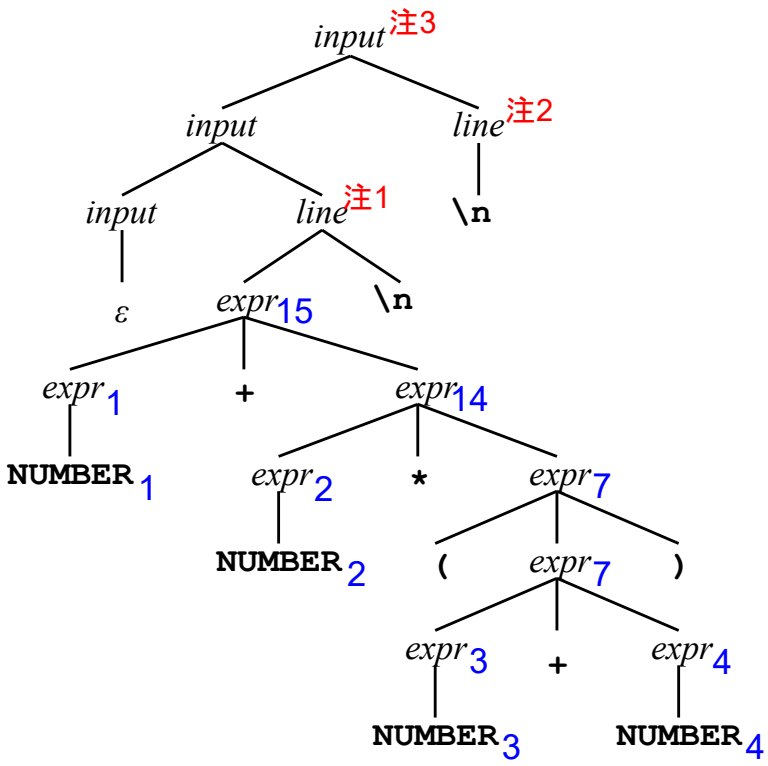
例えば、入力が "`(12+34)*56\n`" という文字列の場合、上の例の `yylex()` の戻り値とそのときの `yylval` の値は次のようになります。

	1回目	**2回目**	**3回目**	**4回目**	**5回目**	**6回目**	**7回目**	**8回目**
`yylex()`	`'('`	NUMBER	`'+'`	NUMBER	`')'`	`'*'`	NUMBER	`'\n'`
`yylval`		12.0		34.0			56.0	

還元時（つまり、解析木の節を作るとき）に対応するアクションが実行されます。

非終端記号の属性値は、各部分木の属性値から計算されます。 **$$** が還元される生成式の左辺の属性値、 **$1**, **$2**, ... が右辺の１番目、２番目、... の文法要素の属性値を表します。例えば、 `$$ = $1 * $3` というアクションでは、１番目と３番目の部分木の属性値の積が、節の非終端記号の属性値となります。

例えば、 `1 + 2 * ( 3 + 4 ) \n \n` というトークン列から、上の Bison プログラムは以下のような解析木を生成します。 青字で示されているのが各節の属性値です。



注 1 この節の属性値はないが、属性値を計算する副作用として “15” が出力される。
注 2 この節の属性値はないが、属性値を計算する副作用として、プログラムが強制終了（exit）される。
注 3 この節は実際には生成されない。（その前にプログラムが終了する。）

追加の C プログラム部（addtional C code というコメントの部分）は 2 つめの「%%」からファイルの最後までです。ここは文字通り追加の C プログラムを書きます。この部分は C のプログラムの末尾にそのままコピーされます。

この例では yylex と main 関数を定義しています。普通は yylex 関数は Flex などで定義しますが、この例では説明のため Bison プログラム中で定義しています。ここで yylex 関数の戻り値はトークンです。すなわち文字コードか、%token で定義した定数マクロ（この例では NUMBER）です。また、トークンの属性値（意味値）は yylval という大域変数に代入して返しています。マクロ YYSTYPE は、この yylval の型を表します。ファイルの終わりに到達するなど、これ以上トークンがないときは、 yylex 関数は 0 を返します。

また、この例では main 関数は Bison のプログラム中で用意していますが、通常は別の C のソースファイルに定義します。この例の main 関数は、単に yyparse を呼び出すだけです。この yyparse は、上の文法規則部から Bison が生成する関数です。

生成

このファイル（ファイル名を calc.y とする）から C ソースファイルを生成するには

```
bison calc.y
```

というコマンドを実行します。これで calc.tab.c という名前（.y ファイルの名前の後ろに .tab がつく）の C ソースファイルができます。また、 -o というオプションで、 C のファイル名を指定することができます。例えば、

```
bison -ocalc.c calc.y
```

で calc.c という名前の C ソースファイルができます。

この例の場合は、この C ソースファイルを普通にコンパイルすると、（警告 (Warning) がいくつか出ますが）実行可能ファイルができます。

Microsoft Visual Studio の場合は、

```
cl calc.c
```

GCC の場合は、

```
gcc calc.c
```

次のコマンドで実行できます。

```
.\calc
```