〈実践報告・資料〉

コンパイラ制作技法

—文法定義・字句解析・構文解析部—

北海道職業能力開発短期大学校　　澤　山　　力

Techniques of Compiler Design

Chikara SAWAYAMA

要約　コンパイラは、コンピュータ言語を翻訳するためのツールと言うだけでなく、各種の情報処理技術を集大成した大規模なソフトウェアである。本稿は、情報処理技術の実践習得を目的にサブセット版のC言語用コンパイラの制作を試みたので、その手順と問題点を報告する。

コンパイラ制作のプロセスを大きく分けると、言語の仕様決定、字句解析、構文解析ルーチンなど標準化が容易な前半部と意味解析・中間表現以降で、対象となるＣＰＵに大きく依存した後半部から構成される。本稿は、標準化を重視してUNIXのソフトウェアツールYACC（Yet Another Compiler Compiler）とＬＥＸ（Lexical Analyzer）を利用して制作した構文解析部までに関する報告であり、1992年度の専門二期研修の研修報告を加筆・再構成したものである。

Ⅰ　はじめに

計算機に仕事をさせるためには、指令書に相当するプログラムが必要である。コンピュータ言語は、プログラムを記述するための文法であり、人間の言語ほどではないまでも、多種類存在する。一般的には、プログラムの生産性を高め、人間の都合に重きを置いた高水準言語を用いてプログラムが作られる場合が多い。しかし、計算機を指令通り動作させるためには高水準言語を計算機が理解できる低水準言語に翻訳してやる必要があり、そのための道具がコンパイラである。コンパイラはコンピュータを使用する上で必然的なものであり、空気のような存在である。利用者の多くは、コンパイラを自ら制作するものとは考えていない。少なくとも私はこの前まではその様な利用者であった。コンパイラを制作するためには前提条件として、越えなくてはならない幾多の課題があり、これらを複合した形での理解力が要求される。このようなコンパイラの制作が短大2年間の情報処理教育で可能かどうか疑問を抱いていたが、卒業して数年でコンパイラを制作したとの話を聞くにつけ、避けて通ることができない。

従来の情報処理教育はコンピュータという格好のインターフェースがありながら、理論と実践の歯車のかみ合いが十分ではなかった。遊離しがちであった情報処理に関する各種技術の実践習得を目的として、「コンパイラ制作」実習を短期大学の学生が理解できる程度に展開する場合の手順と問題点を検討したので報告する。具体的には、ソフトウェアツールを使用してコンパイラを制作したが、紙面の都合上、C言語の基本的な仕様に対して特殊仕様を2点追加したことによる変更点を中心に報告する。

本稿は、1992年10月12日より11月20日までの40日間に及ぶ専門二期研修の研修報告「コンパイラの制作技法」を再構成した。コンパイラ制作のプロセスを大きく分けると、言語の仕様決定、字句解析、構文解析ルーチンなど汎用性を持った前半部と意味解析・中間表現以降で、対象となるＣＰＵに大きく依存した後半部から構成される。本稿は、標準化を重視してUNIXのソフトウェアツールYACC（Yet Another Compiler Compiler）とＬＥＸ（Lexical Analyzer）を利用して制作した前半部に関する報告であり、後半部は追って整理する。

Ⅱ　コンパイラを制作するための各種情報処理技術

コンパイラは各種情報処理技術の集大成である。コンパイラを制作する上で、理解を前提とする技術ある

論文受付日　1993.1.18

いは理解しているとスムーズに作業できる技術を以下に列挙する。

①YACCやＬＥＸなどのソフトウェアツールの使用法

②ソフトウェアツールの出力であるＣ言語の文法と構造体、共用体の使用法

③出力対象であるCPUの機械語の構造

④新たに定義しようとするコンピュータ言語の構造

⑤ブートストラッププログラムの付加

⑥字句解析、構文解析、意味解析、型推論などプログラムの構文と意味のためのアルゴリズムの定義

⑦構文木など中間表現のデータ構造の定義

⑧コード生成やコード最適化などプログラムの生成と変換法

⑨オートマトン理論、ハッシュによる高速ソーティング、逆ポーランド記法など

III　コンパイラ作成の手順

本稿は、Ｃ言語の基本的仕様に次の２点の特殊仕様を追加したことによる変更を通して、コンパイラ制作の一端を紹介する。

【追加仕様】

①インクリメント・デクリメント演算子に特殊形を追加する。つまり、Ｃ言語におけるインクリメント・デクリメント演算子は被演算子に１加えたり、引いたりする演算であるが、１以外の定数も対象とするものである。

例１　y = x＋＋3；
　xの値をyに代入した後、xに3を加える。

例２　y = 5＋＋x；
　xの値に5を加えた後、xの値をyに代入する。

例３　y = x--3;
　xの値をyに代入した後、xから3を引く。

例４　y = 5--x；
　xの値から5を引いた後、xの値をyに代入する。

②複文の区切り記号に特殊形を追加する。つまり、if・for・whileの複文が長くなった場合、複文の終了をそれぞれ}i・}f・}wとすることで、複文の開始と終了の対応を明確にする。もちろん}のみでもエラーとはならない。

```
例１　if(式) {　....;
                ......;
      }i
例２　for(式；式；式) {　...;
                ........;
      }f
例３　while(式) {　...;
                ......;
      }w
```

１．コンピュータ言語の文法を日本語で定義する

文法は文の構造、分解方法をリストアップしたものであり、如何なる文が形作られるかを説明することにより言語を定義することができる。つまり、一定の文法規則に従う入力を言語と呼ぶ。綴り（トークン）には、識別子・キーワード・定数・文字列・演算子その他の区切り子など６種類ある。スペース・タブ・復改・コメントは隣接した識別子・キーワード・定数を区切るのに必要である。文法の記述は構文解析部の規則の中で行う。注意すべき点は、文法は厳密にし、不必要に仕様を拡大することで曖昧な定義を含まないようにすることである。以下に機能追加にともなう仕様書の変更箇所を示す。

【追加前】

```
文：　複文
      式
      if(式)　文
      while(式)　文
      for(式；式；式)　文

複文：{宣言並び　文並び}

式：　＋＋左辺値
      --左辺値
      左辺値＋＋
      左辺値--
```

【追加後】

```
文１：　複文
        式
        if(式)　文１
        while(式)　文１
        for(式；式；式)　文１

文：　文１
        if(式)　複文i
        for(式；式；式)　複文f
        while(式)　複文w

複文：{宣言並び　文並び}
```

複文i：｛宣言並び　文並び｝i
複文f：｛宣言並び　文並び｝f
複文w：｛宣言並び　文並び｝w

式：　定数＋＋左辺値
　　定数--左辺値
　　左辺値＋＋定数
　　左辺値--定数

2．字句解析のための仕様書ファイルを作成する

字句解析は、あらかじめ与えられた規則に従い入力文字列からトークンを切り出す。空白・タブ・改行などC言語では意味を持たない文字を読み飛ばす。さらに、切出した各字句のトークン番号と属性を構文解析部に渡す。

字句解析用ツールであるＬＥＸは、必要以上に長いプログラムを生成し、かつ実行速度も遅いため、字句解析ルーチンは自作される場合が多い。これは、ＬＥＸが字句解析用としての機能以上のオーバースペックなことが原因している。本稿では学生の教材として、汎用性を重視し、あえてＬＥＸを使用した。ＬＥＸは、.l の拡張子ファイルを入力し、lex.yy.c の名のファイルを出力する。曖昧な仕様に対するＬＥＸの扱いは、２つ以上の規則にマッチする場合、マッチする文字数が最大になる規則を優先する。結果的に　＞と　＞＝の区別が付く。又、マッチする文字数が同じ場合は、仕様書ファイルにおいて先に書かれている規則を優先する。よって、予約語の規則は識別子よりも前に記述する。

２点の機能を追加したことによるＬＥＸの仕様書ファイルの変更は次の通りである。

①イリクリメント・デクリメント演算子のトークンは不変なので追加変更は無い。

②追加前では、}は other にマッチングしていたが、追加後では}i・}f・}w を新たなトークンとして付加する必要がある。

【追加前】

```
<INITIAL>"++"   return token(incop);
<INITIAL>"--"   return token(decop);
<INITIAL>{other} return token(yytext[0]);
```

【追加後】

```
<INITIAL>"++"   return token(incop);
<INITIAL>"--"   return token(decop);
<INITIAL>"}i"   return token(rbiop);
<INITIAL>"}f"   return token(rbfop);
<INITIAL>"}w"   return token(rbwop);
<INITIAL>{other} return token(yytext[0]);
```

3．構文解析のための仕様書ファイルを作成する

構文解析はあらかじめ与えられた文法に対して、トークンの並びがどのように対応しているかを調べる。字句解析と構文解析の分け方は自由である。構文解析部での仕事が多いので字句解析部でできることはそちらで行なう。構文解析部の作成は、YACC のようなツールの使用が一般的である。YACC の仕様書ファイルは宣言部、規則部、ユーザサブルーチン部から構成されている。規則部は、入出力インターフェースを十分理解し、整理展開する必要がある。機能追加にともなう YACC 仕様書ファイルの変更は以下の通りである。

①基本のインクリメント・デクリメント演算子はそのままとし、追加した機能は定数＋演算子＋識別子または、識別子＋演算子＋定数の書式であり、引数の数が変わるので新規に構文木を作る関数を追加する必要がある。その関数の中で識別子か定数かを判別する。

② if 文の複文の終わりとして {と}i は良いが}f と}w は使用不可とするためには、変更前の文(stmt)を stmt 1 として基本的な文をカバーする。さらに、追加機能も認めた（stmt）を設定する必要がある。

【追加前】

```
%token   ';' '(' ')' '[' ']' '{' '}' ','
%token<op> incop decop

stmt
  : compound-stmt
   | if-head stmt  { genlabel($1); }
   | while-head stmt  { genjump($1);
   genlabel($1 + 1); popLabels(); }
   | for-head stmt  { genjump($1);
   genlabel($1 + 2);  popLabels(); }

compound-stmt
  : '{'  { yyerrok; }      stmt-list rb  {  };

nc-expr
  : primary  { $$ = $1; }
   | incop nc-expr          /** pre ++ **/
   { $$ = expIncdec( op-PREINC , $2); }
   | decop nc-expr     /** pre -- **/
```

```
    { $$ = expIncdec( op-PREDEC, $2); }
    | nc-expr incop    /** post ++ **/
    { $$ = expIncdec( op-POSTINC , $1); }
    | nc-expr decop    /** post -- **/
    { $$ = expIncdec( op-POSTDEC, $1); }

rb : '} ' { yyerrok; } ;
```

【追加後】

```
%token   ';' '(' ')' '[' ']' '{' '}' ','
%token<op> incop decop
%token<op> rbiop rbfop rbwop

stmt 1
  : compound-stmt
    | if-head stmt 1  { genlabel ($1); }
    | while-head stmt 1 { genjump ($1);
    genlabel ($1 + 1);  popLabels(); }
    | for-head stmt 1 { genjump ($1);
    genlabel ($1 + 2);  popLabels (); }

stmt
  : stmt 1
    | if-head compound-dtmt-i  { genlabel ($1); }
    | while-head compound-stmt-w  { genjump ($1);
    genlabel ($1 + 1);    popLabels (); }
    | for-head compound-stmt-f  { genjump ($1);
    genlabel ($1 + 2);    popLabels (); }

compound-stmt
  : '{'  { yyerrok; }   stmt-list rb  { }  ;
compound-stmt-i
  : '{'  { yyerrok; }   stmt-list rbiop  { }  ;
compound-stmt-f
  : '{'  { yyerrok; }   stmt-list rbfop  { }  ;
compound-stmt-w
  : '{'  { yyerrok; }   stmt-list rbwop  { }  ;

nc-expr
  : primary  { $$ = $1; }
    | incop nc-expr    /** pre ++ **/
    { $$ = expIncdec( op-PREINC , $2); }
    | decop nc-expr    /** pre -- **/
    { $$ = expIncdec( op-PREDEC, $2); }
    | nc-expr incop    /** post ++ **/
    { $$ = expIncdec( op-POSTINC , $1); }
    | nc-expr decop    /** post -- **/
    { $$ = expIncdec( op-POSTDEC, $1); }
    | nc-expr incop nc-expr    /** 3++x   x++2 **/
    { $$ = expIncdec1( op-INC , $1,$3); }
    | nc-expr decop nc-expr    /** 2--y  y--3  **/
    { $$ = expIncdec1( op-DEC ,  $1,$3); }

rb : '}' { yyerrok; } ;
```

Ⅳ　おわりに

機能追加に伴うＬＥＸの仕様書ファイルの変更はわずかであった。又、ＹＡＣＣの仕様書ファイルは、日本語の文法を厳密に定義さえしていれば問題は少ないと考える。コンパイラのような大規模なソフトウェアを制作し、設計初期におけるデータ構造やアルゴリズムの検討の重要性と、それを支えるソフトウェアツールの必要性を痛感した。

研修として、このような機会を与えて頂きました職業能力開発大学校情報工学科の八田先生・玉井先生そして菅野先生に改めて深く感謝するものである。