

# コンパイラ構成

## XC-small が LL(1)文法でない

LL(1)文法は、講義資料 3-parse.pdf 92 ページより「(先読み記号が 1 文字で) うまく変換できる文法クラス」である。しかし、xcc-small.pdf に 4 ページにあるように、statement を IDENTIFIER ":" に展開するべきか、[ exp ] ";" に展開するべきかが 1 トークン先読みでは決定できず、exp か IDENTIFIER のどちらかを判別するために先読みをもう一度行う必要が生じる。そのため LL(1)文法の要件を満たしていない。

```
director(statement, IDENTIFIER ":") = first(IDENTIFIER ":") = {IDENTIFIER}
director(statement, [ exp ] ";") = first([ exp ] ";") = {INTEGER, CHARACTER, STRING,
IDENTIFIER, {}}
```

以上より

$\text{director}(\text{statement}, \text{IDENTIFIER ":"}) \neq \text{director}(\text{statement}, [\text{exp}] ";")$  であり、LL(1)文法ではない。

さらに

```
statement :
    (略)
    | "if" "(" exp ")" statement ["else" statement]
    (略)
```

についても LL(1)文法ではない。

なぜならば

else を if ( exp ) statement の statement の中に if がある際に、どちらの if と対応づけるかについて"あいまい"であり、1 トークン再帰読みによって 1 つの構文木に決定することができない。そのため LL(1)文法ではない。

## xcc-small

### 実装方針

構文解析 -> parse\_AST の順で実装を行なった。

構文解析では、xcc-small.pdf の LL(1)文法のページを参考に parse\_\*\* たちを実装した。

translation\_unit から順に、出てきた塊ごとに parse 関数を実装していくような形で実装を行なった。各関数では、`ast = create_AST("<name>", 0)` をトップとして、それらに AST を add\_AST により追加していく形で実装している。

translation\_unit, compound\_statement だけは add\_AST を 1 回だけでなく、複数回呼び出可能性がある形で実装しているが、それ以外では add\_AST は 1 回しか呼び出さない形で実装することで見通しを良くしている。

parse\_AST についても、構文解析にて作成した AST を show\_AST の結果を参考に、適切な形で parse できるように実装した。indentation 等は、そのまま depth の数字を渡す箇所と depth + 1 する箇所、depth = 0 とする箇所など出力結果が綺麗になるように工夫して実装した。

## 工夫点

実装する際に見通しが良くなるように、多くのコメントを残した。また、formatterを用いて統一的なコーディングスタイルを実現した。

またwarningsが極力出ないように、不要な関数などを消去した。最終的に以下のような状態になっている。

```
xcc-small.c:237:22: warning: comparison of integers of different signs:
'int' and 'enum token_kind' [-Wsign-compare]
    if (lookahead(1) == kind) {
        ~~~~~^~~~~
gcc-small.c:727:19: warning: declaration shadows a variable in the global
scope [-Wshadow]
    struct token *token_p = &tokens[tokens_index++];
        ^
gcc-small.c:121:22: note: previous declaration is here
static struct token *token_p; // for parsing
        ^
gcc-small.c:753:11: warning: unused variable 'ptr_start' [-Wunused-
variable]
    char *ptr_start = ptr;
        ^
gcc-small.c:862:23: warning: declaration shadows a variable in the global
scope [-Wshadow]
    struct token *token_p = &tokens[i];
        ^
gcc-small.c:121:22: note: previous declaration is here
static struct token *token_p; // for parsing
        ^
gcc-small.c:988:25: warning: declaration shadows a local variable [-
Wshadow]
        int is_not_compound_statement_flag = 0;
            ^
gcc-small.c:956:17: note: previous declaration is here
        int is_not_compound_statement_flag = 0;
            ^
gcc-small.c:164:20: warning: unused function 'create_leaf' [-Wunused-
function]
static struct AST *create_leaf(char *ast_type, char *lexeme) {
        ^
gcc-small.c:195:13: warning: function 'show_AST' is not needed and will
not be emitted [-Wunnneeded-internal-declaration]
static void show_AST(struct AST *ast, int depth) {
        ^
gcc-small.c:648:14: warning: unused function 'copy_string_region_ptr' [-
Wunused-function]
static void *copy_string_region_ptr(char *start, char *end) {
        ^
gcc-small.c:859:13: warning: unused function 'dump_tokens' [-Wunused-
function]
static void dump_tokens() {
```

9 warnings generated.

## アピールポイント

適切なインデントが実現されている点がアピールポイントである。

また `else if` についても

```
else {  
    if (...) {  
        ...  
    }  
}
```

のようにならず、

```
else if ()  
{  
    ...  
}
```

のように出力されるようになっている点もアピールポイントである。

## 既知のバグ

なし

## 改善可能点

```
if (exp) {  
    statement;  
} else {  
    statement;  
}
```

のようなネスト構造に改善することが可能である。

具体的には、compound\_statement "{" を出力する前の `if ( )` の時点で次の `statement->child[0]` が compound\_statement であるならば、`if ( )` の後改行しないという方法が考えられる。同様に、else 文についても考えられるが、私の実装方針であると、この方法を愚直に実装すると、compound statement の `printf_ns(depth, )` の問題が生じるため、時間的な問題から改善を断念した。

補足:

前述の `printf_ns(depth, )` の問題とは、compound\_statement の開始の `{` と終わりの `}` で `printf_ns(depth, )` に渡す、depth の値が異なり、場合分けが煩雑になるという問題である。

