

3 BNF と mini-C 言語

- ♣ BNF による構文記述法
- ♣ コンパイラ作成演習の言語 mini-C の概要

3.1 BNF 記法

- BNF 記法 (`Backus-Naur` form, `Backus normal` form とも)

プログラミング言語などの `構文規則` (syntax) の記述法

- 形式は

`構文要素 ::= 構文を表す式`

- 「構文を表す式」

(a) 接続

`代入文 ::= 変数 "=" 式 ";"`

「代入文は、変数、=、式、; をこの順に並べたものである」

(b) 選択

$x \mid y$ は 「 `x または y` 」 を表す

`文 ::= 代入文 | if 文 | while 文 | …`

「文は、代入文、if 文、while 文、…の `いずれか`

ϵ は `空列` (長さ 0 の文字列) を表す

`if 文 ::= "if" "(" 式 ")" 文 (ϵ | "else" 文)`

は

`if 文 ::= "if" "(" 式 ")" 文
| "if" "(" 式 ")" 文 "else" 文`

と等価

(c) 巾 (べき)

x^* は 「 x の `0 回以上の繰り返し`」 を表す

`複文 ::= "{" 文 * "}"`

1

¹ 「複文」とは `{x=10; c=4;}` のように、`{ }` の中に「文」が 0 回以上現れるものである。

※ BNF では **再帰** 的な式が許されており、これで繰り返しを表現することも多い。

複文 ::= "{" 文リスト "}"

文リスト ::= ϵ | 文 文リスト

3.2 mini-C 言語

3.2.1 特徴

1. データ型

- 基本型は整数 (`int`) 型, 文字 (`char`) 型およびその **ポインタ** と **配列** をサポート
 - 文字列, 構造体, 共用体, 型定義などはサポートしない

2. 式

- 演算は加減乗除算, **配列** 参照, **ポインタ** 修飾, **比較** 演算をサポート
 - 代入演算式, シフト演算, 論理演算はサポートしない
 - ポインタ修飾は変数に対してのみ適用可能 (一般の式には不可)

3. 文

- 代入文, 複文, **if** 文, **while** 文, **関数呼出し**, **return** 文をサポート
 - for 文, do-while 文, switch 文, break 文, goto 文などはサポートしない
 - 変数の動的割当て (`malloc`, `free`) はサポートしない

4. 関数

- 関数の型も `int` 型と `char` 型のみ
- 関数の型の宣言は省略不可
- **再帰** 呼出しをサポートする

5. 入出力

- 次の 4 つの関数をサポート
 - `char getchar()`: 標準入力より 1 文字入力し, その値を返す
 - `int getint()`: 標準入力より整数を 1 つ入力し, その値を返す
 - `char putchar(文字)`: 標準出力に文字データを出力し, その値を返す
 - `int putint(整数)`: 標準出力に整数データを出力し, その値を返す

3.2.2 プログラムの例

階乗計算プログラム (fact.mc)

```
1:  int fact(int k)
2:  {
3:      if (k==0) return 1;
4:      else      return k*fact(k-1);
5:  }
6:
7:  int main()
8:  {
9:      int n;
10:     int f;
11:     putchar('n');
12:     putchar('=');
13:     n = getint();
14:     f = fact(n);
15:     putint(n);
16:     putchar('!');
17:     putchar('=');
18:     putint(f);
19:     putchar('\n');
20:     return 0;
21: }
```

3.2.3 Mini-C 言語の BNF

1. 字句の定義

- (a) 数字 ::= "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
- (b) 英字 ::= "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"|"_"
- (c) 普通文字 ::= "\ " と "' ' 以外の文字
- (d) キーワード ::= "char" | "else" | "if" | "int" | "return" | "while"
- (e) ID ::= 英字 (英字 | 数字)*
- (f) INT ::= 数字 (数字)*
- (g) CHAR ::= "' ' 普通文字 "' | "'\n'" | "'\''" | "'\t'" | "'\\'"
- (h) 演算子 ::= "+" | "-" | "*" | "/" | "%" | "&" | "=" | "==" | "!=" | ">" | ">=" | "<" | "<="
- (i) 記号 ::= "," | ";" | "(" | ")" | "{" | "}" | "[" | "]"

2. 構文の定義

- (a) プログラム ::= (関数宣言 | 変数宣言 ";")*
- (b) 変数宣言 ::= 型 "*" ID ("[" INT "]")*
- (c) 関数宣言 ::= 型 "*" ID "(" (ϵ | 変数宣言 ("," 変数宣言)*) ")" 関数本体
- (d) 関数本体 ::= "{" (変数宣言 ";")* 文 * "}"
- (e) 型 ::= "int" | "char"
- (f) 文 ::= ";" | "{" 文 * "}" | if文 | while文 | return文 | 関数呼出し ";" | 代入文
- (g) if文 ::= "if" "(" 式 ")" 文 (ϵ | "else" 文)
- (h) while文 ::= "while" "(" 式 ")" 文
- (i) return文 ::= "return" 式 ";"
- (j) 代入文 ::= 左辺式 "=" 式 ";"
- (k) 左辺式 ::= "*" 変数名 ("[" 式 "]")*
- (l) 変数名 ::= ID
- (m) 式 ::= 式2 (("<" | ">" | "<=" | ">=" | "==" | "!=") 式2)*
- (n) 式2 ::= (ϵ | "+" | "-") 式3 (("+" | "-") 式3)*
- (o) 式3 ::= 式4 (("*" | "/" | "%") 式4)*
- (p) 式4 ::= "*" 式5
- (q) 式5 ::= INT | CHAR | "(" 式 ")" | 関数呼出し | 変数参照
- (r) 変数参照 ::= (ϵ | "&") 変数名 ("[" 式 "]")*
- (s) 関数呼出し ::= 関数名 "(" 引数リスト ")"
- (t) 関数名 ::= ID
- (u) 引数リスト ::= ϵ | 式 ("," 式)*

3.2.4 Mini-C の処理系 (演習)

- コンパイラ

- 中間コード (仮想スタックマシン VSM のアセンブリ) を生成

- 1 パス

- * 構文解析 をしながら コード生成 を行なう

- * 最適化 は行なわない

- 作成するプログラム

1. lex.c: 字句解析 ルーチン → 演習 L

2. mcc.c: コンパイラの 構文解析 部 + コード生成 部 → 演習 S1~S4

※ 仮想スタック機械 VSM のインタプリタは準備してある。ただし、準備として VSM のアセンブリプログラミングが 1 回ある → 演習 V



Nagisa ISHIURA