

プログラム言語論
-MiniC 言語について-
亀山幸義

1 MiniC 言語の構文

MiniC は C 言語のサブセットであり，図 1 に示す構文を持つ。

```
[識別子] ident ::= (英文字からはじまる英数字等の列)
[定数] const ::= true | false | 整数定数 (負数を含む)
[プログラム] prog ::= dec | dec prog
[宣言] dec ::= type ident(plist) statement
           | type ident ;
[型] type ::= int | bool
[仮引数の並び] plist ::= type ident | type ident , plist
[文] statement ::= ident = exp;
           | while exp statement
           | { prog stlist }
           | ;
           | print exp ;
           | show exp ;
           | return exp ;
           | if exp statement else statement
[文の並び] stlist ::= statement | statement stlist
[式] exp ::= ident | const | exp + exp | exp * exp
           | exp == exp | exp > exp | exp < exp
           | ident ( explist ) | ( exp )
[式の並び] explist ::= exp | exp , explist
```

図 1: MiniC の構文

MiniC のプログラム例を以下にあげる。

```
/* miniC 言語テストプログラム */
/* コメントは C 言語と同様に書ける。*/
int main () {
    int zero;
    int minus_one;
```

```

int sum;
int n;
zero = 0;
minus_one = -1;
sum = 0;
n = 10;
while (n > zero) {
    sum = sum + n;
    n = n + minus_one;
};
/* 次の行の print 文は C 言語にはないものである */
print sum; /* 引き数 sum の値を計算して、その値を出力する */
}

```

2 項演算子の $-$ がないので、「1 を引く」という操作のかわりに -1 を加えている。

また、型として `int` と `bool` の 2 つがある (それしかない)。さらに、`int` と `bool` は区別されているので、`int` 型の変数に `true` という `bool` 型の定数を代入するとエラーになる。

MiniC は、このように制限されているが、表現力はそれほど劣っているものではなく、たとえば、再帰呼び出しを使って階乗関数を記述できる。

```

int f (int x) {
    int result;
    if (x==0) result = 1;
    else result = x * f(x+(-1));
    return result;
}
int main () {
    print f(5);
}

```

ここで `print` 文のあとには、`int` 型か `bool` 型の式を 1 つだけ書けるようにした。また、全ての関数は、値を返さなければならない。(void 型がないため。) したがって、以下のプログラムはエラーである。

```

f (int x) {
    ...
}
int main () {
    f(5);
}

```

上記の構文の定義では、わかりやすさのために、「曖昧さのある構文」を与えた。たとえば、 $1 + 2 * 3$ という式は、 $(1 + 2) * 3$ という式なのか、 $1 + (2 * 3)$ という式なのか、上記の構文では曖昧である。

実際に演習で使う miniC 言語では、この点は C 言語と同様、演算子の優先度に従って処理され、たとえば、 $1 + 2 * 3$ は $1 + (2 * 3)$ と構文解析される。

2 MiniC 言語のプログラムの意味

MiniC 言語の意味論は講義を参照されたい。ここでは、要点を述べる。

2.1 大域変数、局所変数、ブロック構造

以下の例題からわかるように、MiniC では、変数の宣言があらわれる場所は、いくつかある。

```
int x;
int foo (int y) {
    int z;
    ...
    {int w; ...}
    ...
}
int main () {
    ...foo(10) ...
}
```

- 大域変数 (global variable): 上記の x のように、どの関数の中でもない場所で宣言された変数。
- 局所変数 (local variable): 上記の z や w のように、プログラム中の一部の場所のみで使えるよう宣言された変数。
- 関数の仮引数 (formal parameter): 上記の y のように、関数のパラメータ (仮引数, formal parameter) を表す変数で、関数に局所的な変数である。(参考: $foo(10)$ のように関数 foo の呼び出し側での 10 という引数を、実引数 (actual parameter) という。)

C 言語のように、ALGOL 言語の流れを組む言語 (現在存在する非常に多くの言語) は、プログラム (のテキスト) 全体が、ブロックとよばれる部分に分割される。ブロックについて、基本的な事実を列挙する。

- ブロックは、プログラムの構文上の概念である。(プログラムの構造に対応したものである。)
- ブロックは、入れ子構造をなす。
- 各々の変数宣言は、特定のブロックのみで有効である。(大域変数に対応するブロックは、プログラム全体と見なす。)

これらの事実から、環境をスタックに積んだ形式の処理系 (インタープリタ) が作成できる。(これについては、授業中に MiniC 言語に対する処理系として説明する。)

2.2 動的束縛 (dynamic binding)、静的束縛 (static binding)

変数束縛 (variable binding, 単に「束縛」とも言う) とは、プログラム中で使われる変数名が、どの変数 (どこで宣言された変数) に対応するか、を決めるものである。

C 言語の場合、変数束縛の関係は単純である。一方、MiniC 言語では、様々な仕組みを学ぶために、静的束縛以外のものを実装しており、内部のスイッチにより切り替えられるようになっている。

例として、次の MiniC プログラムを見てみよう。

```

int x;
int g (int y) {
    return x + y;
}
int f () {
    int x; x = 10;
    return (g(20));
}
int main () {
    x = 5; print f();
}

```

ここで問題となるのは、関数 `g` における変数 `x` の使用が、どの宣言に対応するものか、ということである。もし、プログラムの文面だけを見ていたら `g` の中の `x` は、大域変数である。この値は `main` 関数で 5 になっているので、`x=5` が使われると考えられる。一方、プログラムの実行を追ってみると、関数 `g` がよばれるのは関数 `f` からであり、`f` においては局所的に変数 `x` が宣言されているので、関数 `g` の中での `x` の値は 10 になる。これらは、両方ともあり得る考え方であり、以下の 2 つの方式に対応する。

- 静的束縛: 変数の使用と、変数の宣言の対応関係 (束縛) は、プログラムを書いた途端に (そのプログラムテキスト上の位置関係により) 決まる。実行時情報にはよらない。
- 動的束縛: 変数の使用と、変数の宣言の対応関係 (束縛) は、プログラムを書いた時ではなく、プログラムの実行時に決まる。

これらの違いに応じて、スタックの取り扱いも違ってくる。それについて、MiniC 言語処理系を用いて演習を行う。