記号処理・第 10 回

2018 年 6 月 19 日

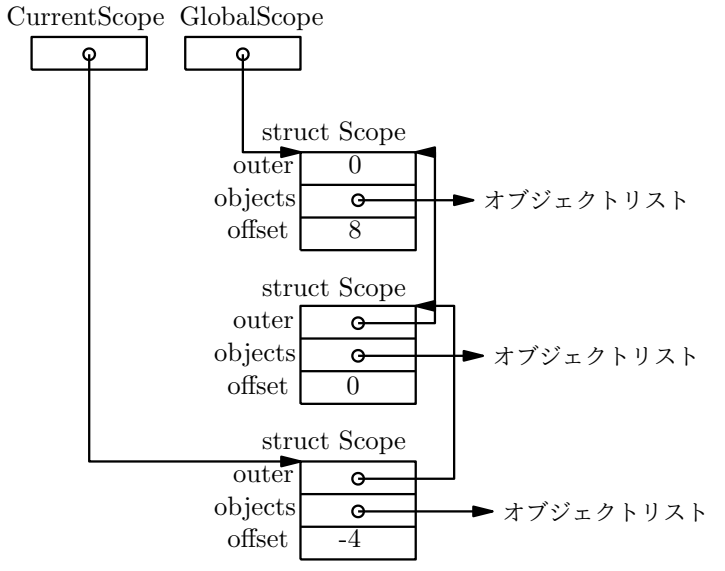
今回の内容

10.1 コンパイラによるオブジェクトの管理

前回、大域変数、関数やその実引数、局所変数が MVM のメモリ空間のおおよそどの位置に置かれるかについて解説しました。メモリ空間のある領域を占めることになるこれらの変数や関数を、ここでは一般に**オブジェクト**[1]と呼ぶことにします。Minimum C コンパイラが、ソースプログラム中に現れた変数の参照、代入、関数の呼び出し等に対して具体的な機械語命令を生成するためには、参照、代入、呼び出しなどの対象となっているオブジェクトが、それぞれ具体的にどのアドレスに置かれる (予定な) のかを知る必要があります。このため Minimum C コンパイラでは、ソースプログラム中で宣言・定義されたオブジェクトに関する情報を、次の 2 つのデータ構造を用いて管理することにします。

メモ

スコープリスト　スコープリストは、ソースプログラム中の各スコープ[2]の包含関係に関する情報を保持します。このリストの各要素 (`struct Scope`) は、そのスコープで宣言されたオブジェクトに関する情報を保持するオブジェクトリスト (`objects`) と、次に宣言される変数が割り当てられるであろう (メモリ中での) 相対位置 (`offset`) を保持しています。構文解析部が新しい関数定義や新しいブロックの内側に入る度に、新しい要素がスコープリストの先頭に追加され、そのスコープで宣言されたオブジェクトに関する情報がオブジェクトリストとして保持されます。ブロックや関数定



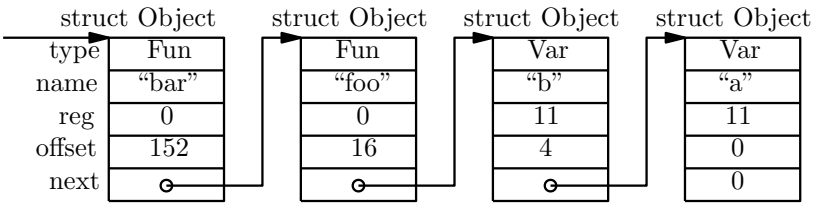
[1] いわゆる「オブジェクト指向プログラミング」で言うところの「オブジェクト」の意味ではありません

[2] ブロック ⟨*Block*⟩、関数定義 ⟨*FunDef*⟩、ソースファイル全体 ⟨*Prog*⟩ など、識別子 (変数名や関数名) の通用範囲

義を抜けると、そこに入るときに作られた要素はスコープリストから削除されます。スコープリストの先頭要素は現在構文解析中であるスコープに対応しており、リストの末尾は常に大域変数の宣言や関数の定義が行われるソースプログラムの一番外側のスコープに対応しています。Minimum C コンパイラは、これらをそれぞれ `CurrentScope` と `GlobalScope` という 2 つの大域変数に保持します。

メモ

オブジェクトリスト　オブジェクトリストは、各スコープ (ブロックなど) 中で宣言されたオブジェクト (変数や関数) に関する情報を保持するリストです。上に述べたように、スコープリスト中の要素がそれぞれオブジェクトリストを保持します。オブジェクトリストの各要素は、各オブジェクトの名前 (`name`)、型 (`type` — 変数かそれとも関数か) と参照方法 (基準となるアドレスの格納されているレジスタ番号 `reg` とそのアドレスからの変位 `offset`) を保持しています。コンパイラが構文解析中のスコープ内で、新しいオブジェクト (変数や関数) が宣言・定義される度に、新しい要素がオブジェクトリストの先頭に追加され、そのオブジェクトに関する情報がそこに記録されます。

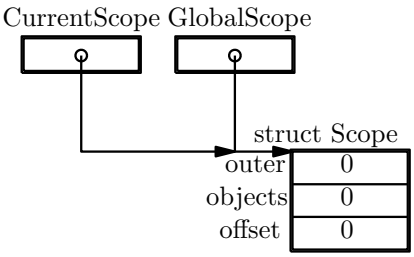


メモ

オブジェクト管理の例

以下は、コンパイラによってスコープリストやオブジェクトリストが管理されていく過程を、あるプログラムを例として図示したものです。

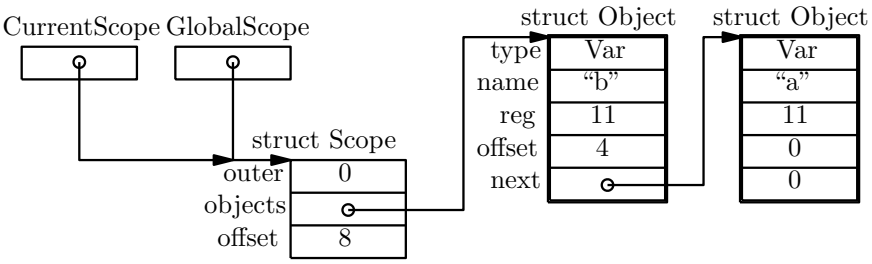
状態1: プログラム全体に対する構文解析を始める前に、ソースプログラムの一番外側のスコープが生成され、これが `GlobalScope` となります。`GlobalScope` の値はソースプログラム全体のコンパイルが終了するまで変りません。`GlobalScope` は、この時点での `CurrentScope` となります。



処理済

```
int a, b;

foo(x, y)
{
    int b;
    ⋮
```

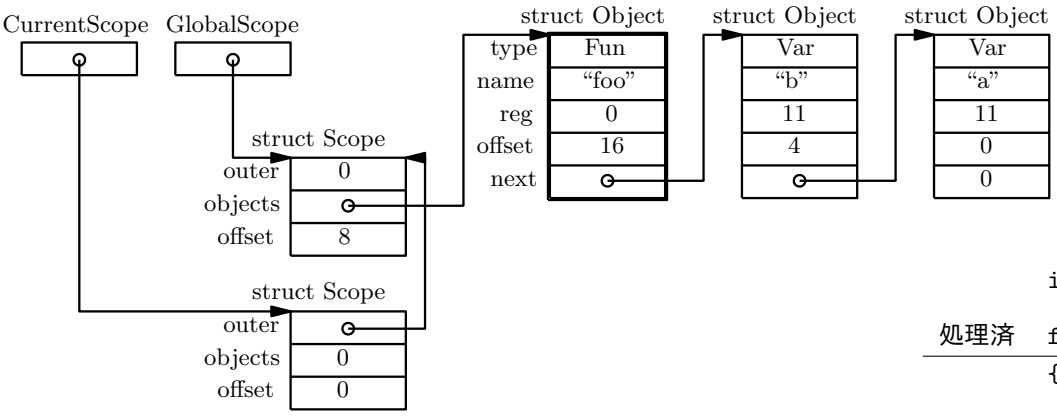
状態2: 大域変数 `a` と `b` がこのスコープのオブジェクトリストに追加されます。大域変数はレジスタ `R11` の値を基準として参照しますので、オブジェクトリストの各要素 (`struct Object`) の `reg` は 11 となり、新しい大域変数が現れる度に `offset` は 4 増えます。コンパイラが大域変数 `b` の処理を完了した時点では、`GlobalScope` (= `CurrentScope`) の `offset` の値は、さらに次の大域変数が割り当てられるであろう位置を表す 8 という値になっています。



処理済 `int a, b;`

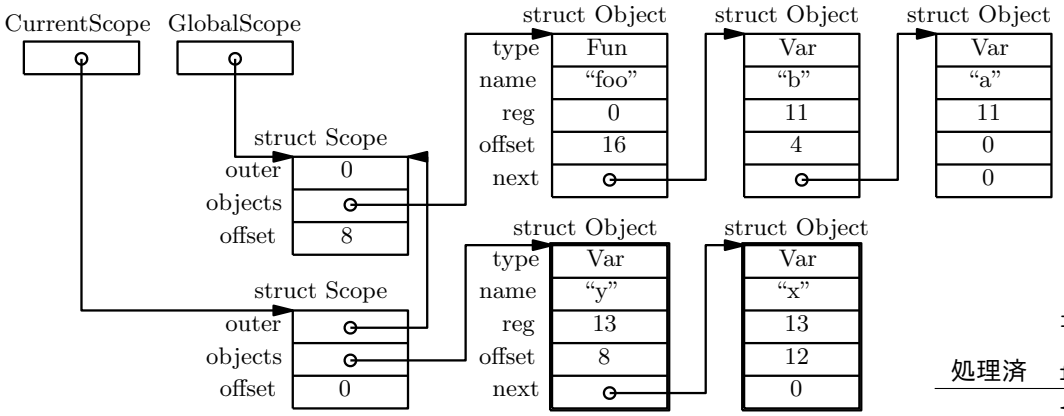
```
foo(x, y)
{
    int b;
    ⋮
```

状態3: 関数 `foo` がこのスコープのオブジェクトリストに追加されるとともに、その引数のためのスコープがスコープリストの先頭に追加され、`CurrentScope` となります。関数の場合、基準となるレジスタは `R0` です。`offset` は、`foo` の機械語命令の生成開始アドレスとなります。



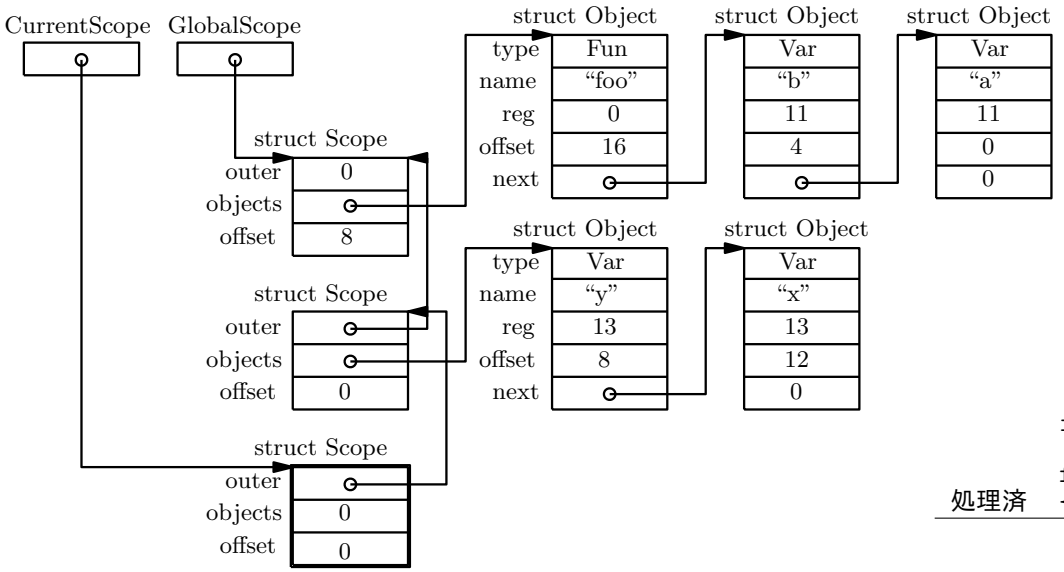

```
int a, b;

処理済 foo(x, y)
{
    int b;
    ⋮
```

状態4: 関数 `foo` の引数が `CurrentScope` のオブジェクトリストに追加されます。関数の実引数はフレームポインタ `R13` の値を基準として参照しますので `reg` は 13 となります。`offset` は新しい引数が現れる度に 4 減り、最後の引数で 8 になります。各引数の `offset` 値は、引数をすべて読み込んだ後でないと決められませんので、`struct Scope` の `offset` の値は使用されません。



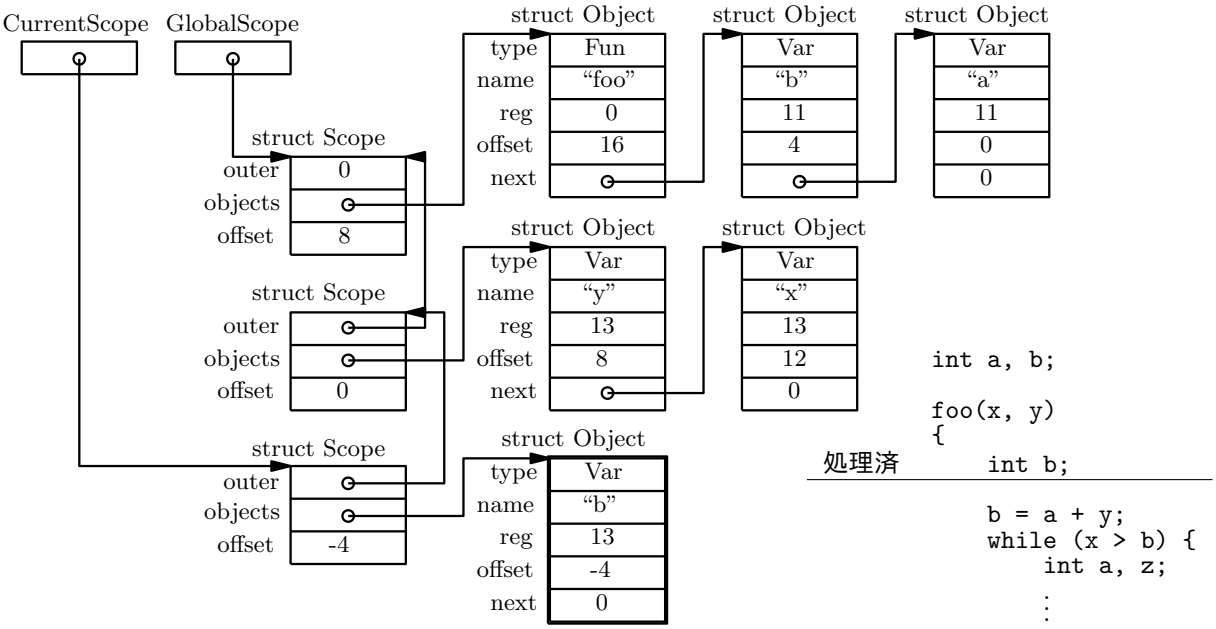

```
        int a, b
処理済  foo(x, y)
        {
            int b;
            ⋮
```

状態5: ブロックの始まりに対してスコープが 1 つ生成されます。



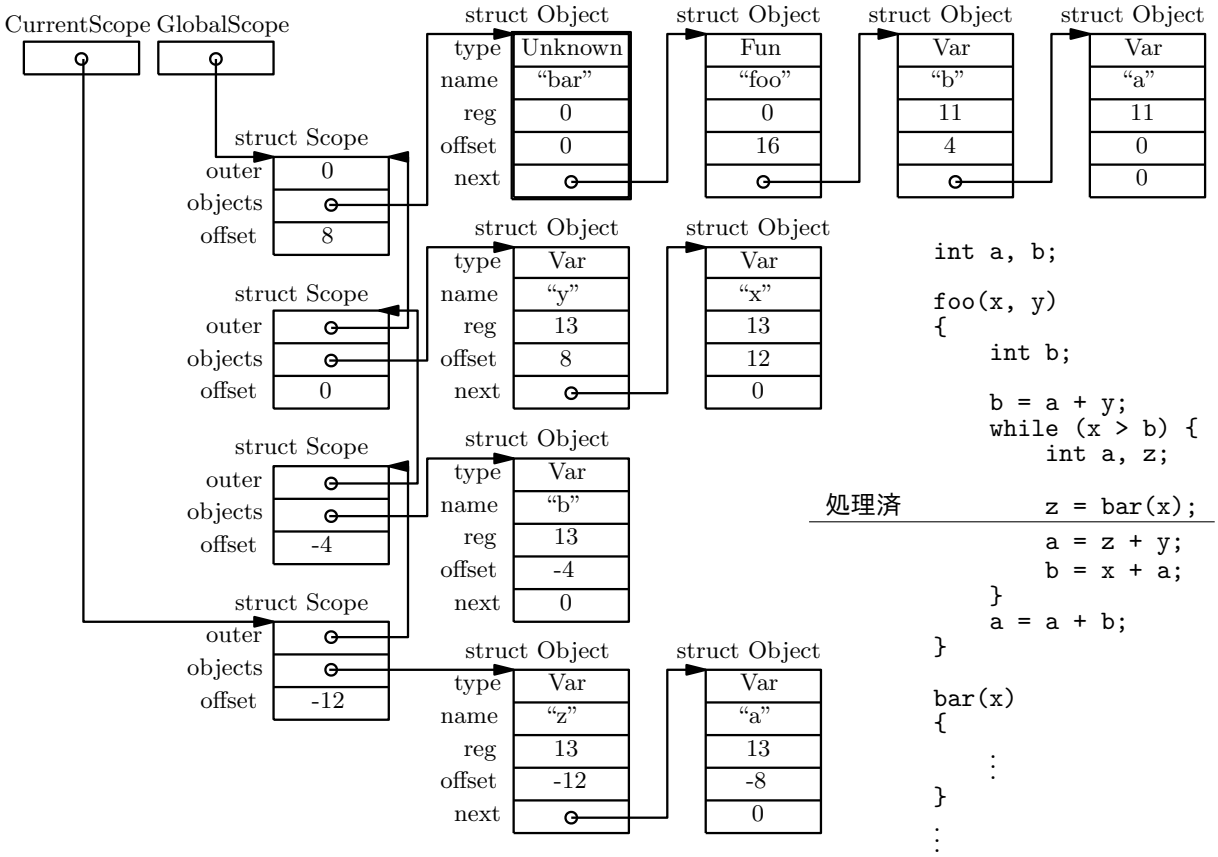

```
        int a, b

        foo(x, y)
処理済  {
            int b;
            ⋮
```

状態6: このブロックの局所変数 b がオブジェクトリストに追加されます。局所変数もフレームポインタ R13 の値を基準として参照しますので reg は 13 となります。offset は、最初の局所変数で −4 となり、新しい局所変数が現れる度に 4 減ります。局所引数のためのスコープ (struct Scope) の offset の値は、最後に現れた局所変数を割り当てた位置を保持します。

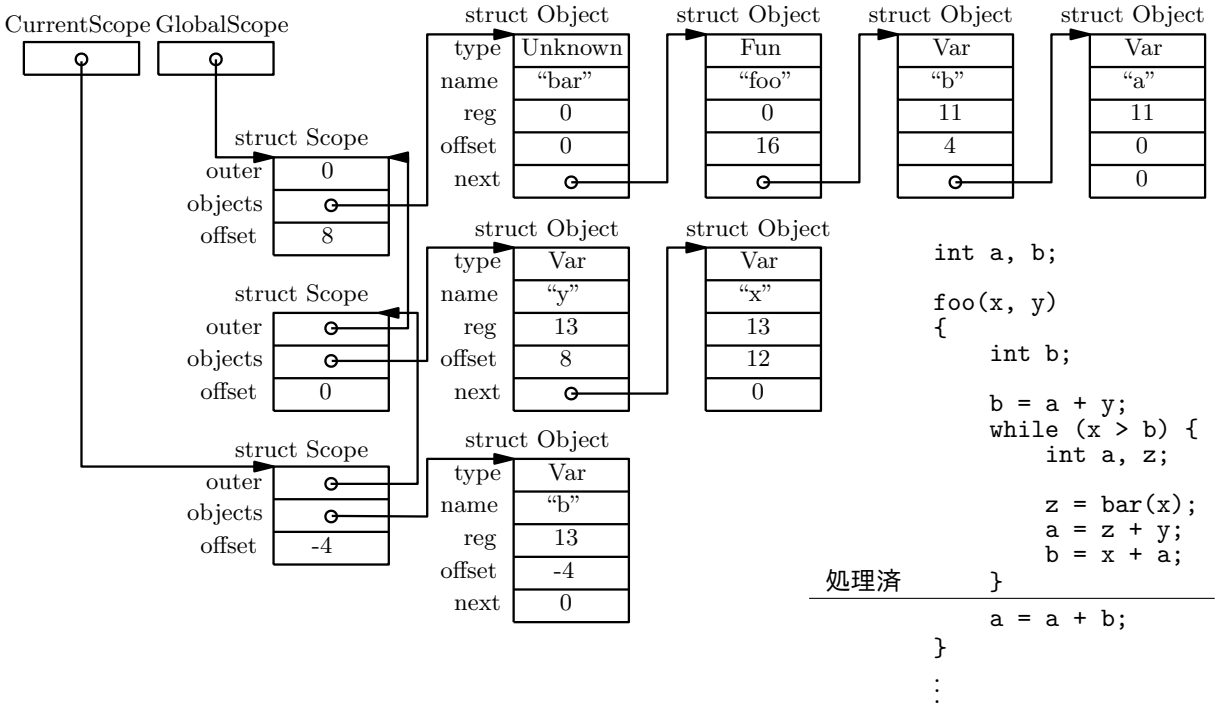


状態7: さらに内側のブロックが始まると、この新しいブロックのためのスコープが追加され、その局所変数 a と z がオブジェクトリストに登録されます。

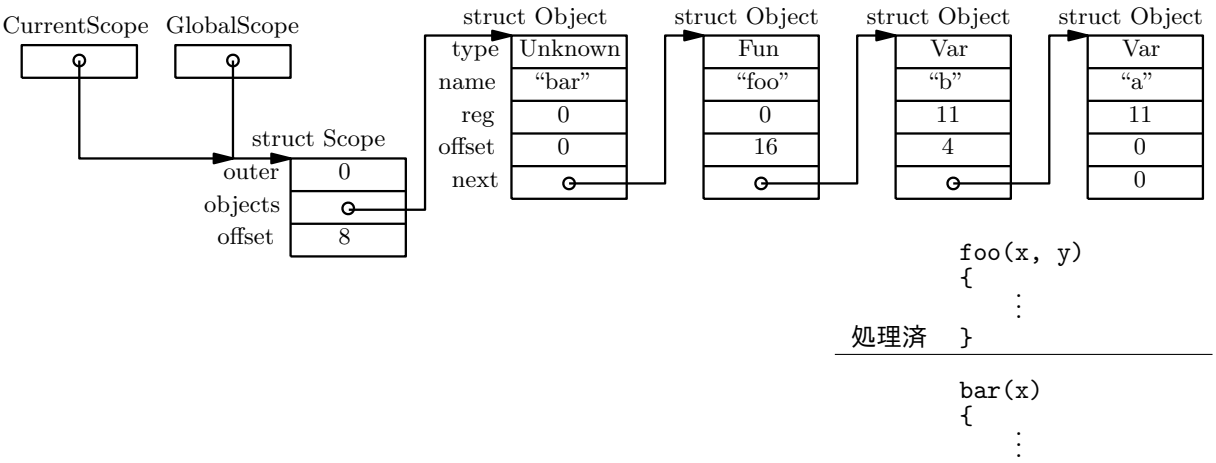
状態8: 未定義の関数が現れた場合は、`GlobalScope` のオブジェクトリストに `type` = Unknown としてこの関数を登録しておきます。未定義の関数の `offset` の値は、後にその定義が現れた時点で設定されます。



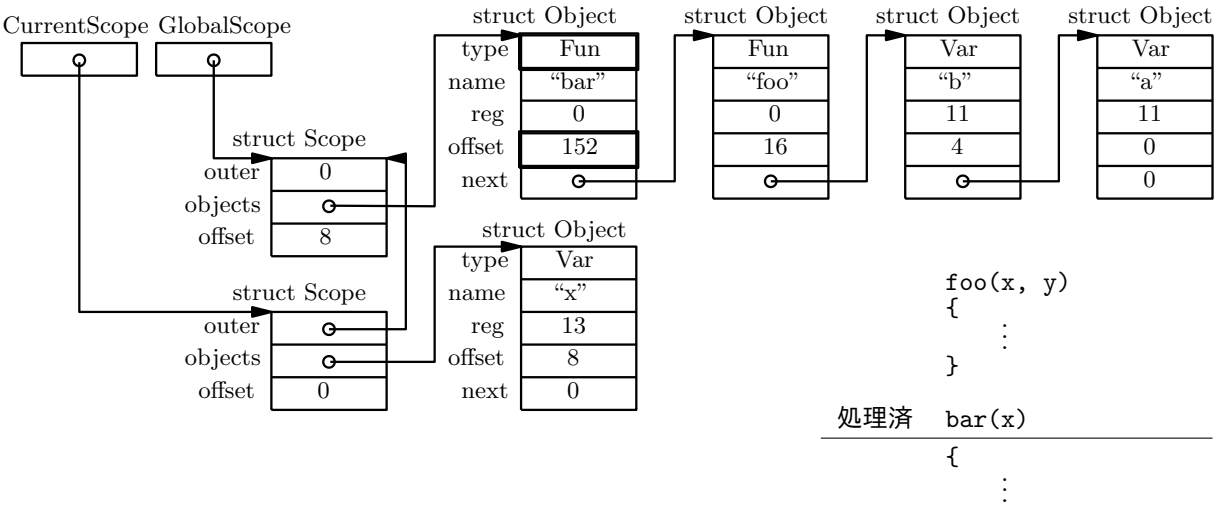
状態9: ブロックを抜けると、対応するスコープがスコープリストから取り除かれます。



状態10: 関数定義を終えると、スコープリストの要素は `GlobalScope` だけになります。



状態11: 関数 `bar` の定義が現れると Unknown だった `bar` の `type` が Fun となり、`offset` はこれから `bar` のための機械語命令の生成が始まるアドレスとなります。新しい関数の定義が始まると、スコープリストはまた伸び始めます。



10.2 オブジェクト管理部分を追加したパーザプログラム

以下は、第 7 回で紹介した文法チェックのみを行う構文解析部 (`parse0.c`) にオブジェクト管理部分を追加したもの (`parse1.c`) です。変更された行の先頭には * が付してあります。

parse1.c

```
   1 /*
*  2  *      parse1.c
   3  */
   4
   5 #include <stdio.h>
   6 #include <stdlib.h>
*  7 #include <string.h>
   8 #include "scan.h"
   9
* 10 #define MVM_R0  (0)
* 11 #define MVM_GP  (11)
* 12 #define MVM_FP  (13)
```

```
  13
* 14 typedef unsigned long Word;
* 15 typedef long Offset;
  16
* 17 typedef enum { Var, Fun, Unknown } ObjectType;
  18
* 19 typedef struct Object {
* 20     ObjectType      type;       /* オブジェクトの種類(変数/関数/未定義) */
* 21     char            name[MC_MAXIDLEN+1];        /* 変数名/関数名 */
* 22     int             reg;        /* 変数アドレスの基準となるレジスタ番号
* 23                                    関数の場合は R0、大域変数の場合は GP (R11)、
* 24                                    関数の仮引数や局所変数の場合は FP (R13) */
* 25     Offset          offset;     /* reg からのオフセット(相対位置) */
* 26     struct Object * next;       /* 同じスコープの次のオブジェクト */
* 27 } Object;
  28
* 29 typedef struct Scope {
* 30     struct Scope *  outer;      /* 一つ外側のスコープ */
* 31     Object *        objects;    /* このスコープのオブジェクトのリスト */
* 32     Offset          offset;     /* 新しい変数の次の割り当て(相対)位置 */
* 33 } Scope;
  34
* 35 static Scope * GlobalScope;     /* 大域(一番外側の)スコープ */
* 36 static Scope * CurrentScope;    /* 現在コンパイル中のスコープ */
  37
* 38 /* FindObject() : 指定のスコープのオブジェクリストからオブジェクトを探し
* 39                   出す。 見つからない場合は 0 を返す。 */
  40
* 41 static Object * FindObject(Scope *scope, char *name)
* 42 {
* 43     Object *obj;
  44
* 45     for (obj = scope->objects; obj != (Object *)0; obj = obj->next) {
* 46         if (strcmp(name, obj->name) == 0)
* 47             return obj;
* 48     }
* 49     return (Object *)0;
* 50 }
  51
* 52 /* NewObject() : 新しいオブジェクトを作成する。 */
  53
* 54 static Object * NewObject(ObjectType type, char *name)
* 55 {
* 56     Object *obj;
  57
* 58     obj = (Object *) malloc(sizeof(Object));
* 59     if (obj == (Object *)0)
* 60         Error("コンパイルするためのメモリが足りません");
* 61     strcpy(obj->name, name);
* 62     obj->type = type;
* 63     obj->next = (Object *)0;
* 64     return obj;
* 65 }
  66
* 67 /* NewVar() : 新しい変数を現在のスコープに追加する。*/
  68
* 69 static Object * NewVar(char *name)
* 70 {
* 71     Object *obj;
  72
* 73     if (FindObject(CurrentScope, name) != (Object *)0)
* 74         Error("変数名または引数名が衝突しています");
  75
```

```
* 76      obj = NewObject(Var, name);
* 77      obj->next = CurrentScope->objects;
  78
* 79      if (CurrentScope->outer == (Scope *)0) { /* 大域変数 */
* 80          obj->reg = MVM_GP;
* 81          obj->offset = CurrentScope->offset;
* 82          CurrentScope->offset += sizeof(Word);
* 83      }
* 84      else if (CurrentScope->outer->outer == (Scope *)0) { /* 仮引数 */
* 85          obj->reg = MVM_FP;
* 86          obj->offset = 0;        /* 後で FunDef() が設定する */
* 87      }
* 88      else { /* 局所変数 */
* 89          obj->reg = MVM_FP;
* 90          CurrentScope->offset -= sizeof(Word);
* 91          obj->offset = CurrentScope->offset;
* 92      }
  93
* 94      CurrentScope->objects = obj;
* 95      return obj;
* 96  }
  97
* 98  /* UseObject() : オブジェクトを探し出す。 見つからなければ、
* 99                   type=Unknown として大域スコープにオブジェクトを作る。 */
 100
*101  static Object * UseObject(char *name)
*102  {
*103      Scope *scope;
*104      Object *obj;
 105
*106      /* 内側のスコープから大域スコープに向って順にオブジェクトを探す */
*107      for (scope = CurrentScope; scope != (Scope *)0; scope = scope->outer) {
*108          obj = FindObject(scope, name);
*109          if (obj != (Object *)0)
*110              return obj;
*111      }
 112
*113      /* 大域スコープに新しいオブジェクトを作る */
*114      obj = NewObject(Unknown, name);
*115      obj->reg = MVM_R0;
*116      obj->offset = 0;
*117      obj->next = GlobalScope->objects;
*118      GlobalScope->objects = obj;
*119      return obj;
*120  }
 121
*122  /* NewScope() : 新しいスコープを作成し、それを CurrentScope とする。 */
 123
*124  static void NewScope(Offset offset)
*125  {
*126      Scope * scope;
 127
*128      scope = (Scope *) malloc(sizeof(Scope));
*129      if (scope == (Scope *)0)
*130          Error("コンパイルするためのメモリが足りません");
*131      scope->outer = CurrentScope;
*132      scope->objects = (Object *)0;
*133      scope->offset = offset;
*134      CurrentScope = scope;
*135  }
 136
*137  /* ExitScope() : CurrentScope を廃棄し、一つ外側のスコープに出る。 */
 138
```

```
*139 static void ExitScope(void)
*140 {
*141     char message[100+MC_MAXIDLEN];
*142     Scope *cur = CurrentScope;
*143     Object *obj, *next;
 144
*145     /* 未定義関数のチェック */
*146     for (obj = cur->objects; obj != (Object *)0; obj = next) {
*147         if (obj->type == Unknown) {
*148             sprintf(message, "関数 %s が未定義です", obj->name);
*149             Error(message);
*150         }
*151         next = obj->next;
*152         free((void *)obj);
*153     }
 154
*155     CurrentScope = CurrentScope->outer;
*156     free((void *)cur);
*157 }
 158
 159 static void Exp(void);
 160 static void Block(void);
 161
 162 static void FunArgs(void)
 163 {
 164     GetToken();
 165     if (token != mc_rparen) {
 166         Exp();
 167         while (token == mc_comma) {
 168             GetToken();
 169             Exp();
 170         }
 171         if (token != mc_rparen)
 172             Error("関数呼び出しの引数リストに ) がありません");
 173     }
 174     GetToken();
 175 }
 176
 177 static void Factor(void)
 178 {
*179     Object *obj;
 180
 181     switch (token) {
 182     case mc_ident:
 183         /* 変数: <Ident>、または、関数呼び出し: <Ident> <FunArgs> */
*184         obj = UseObject(ident);
 185         GetToken();
*186         if (token == mc_lparen) {
*187             if (obj->type == Var)
*188                 Error("変数に対する関数呼び出しはできません");
 189             FunArgs();
*190         }
*191         else if (obj->type != Var)
*192             Error("宣言されていない変数は使用できません");
 193         break;
 194     case mc_number:
 195         /* 定数: <Number> */
 196         GetToken();
 197         break;
 198     case mc_lparen:
 199         /* 括弧で囲まれた式: ( <Exp> ) */
 200         GetToken();
 201         Exp();
 202         if (token == mc_rparen)
```

```
 203             GetToken();
 204         else
 205             Error("閉じ括弧があるべき位置にありません");
 206         break;
 207     default:
 208         Error("式に文法誤りがあります");
 209         break;
 210     }
 211 }
 212
 213 static void Term(void)
 214 {
 215     Factor();
 216     while (token == mc_mult || token == mc_div || token == mc_mod) {
 217         GetToken();
 218         Factor();
 219     }
 220 }
 221
 222 static void Exp(void)
 223 {
 224     if (token == mc_plus || token == mc_minus) {
 225         GetToken();
 226         Term();
 227     }
 228     else
 229         Term();
 230     while (token == mc_plus || token == mc_minus) {
 231         GetToken();
 232         Term();
 233     }
 234 }
 235
 236 static void Cond(void)
 237 {
 238     Exp();
 239     if (mc_eq <= token && token <= mc_ge) {
 240         GetToken();
 241         Exp();
 242     }
 243     else
 244         Error("条件式に文法誤りがあります");
 245 }
 246
 247 static void Statement(void)
 248 {
*249     Object *obj;
 250
 251     switch (token) {
 252     case mc_ident:
 253         /* <Ident> = <Exp>; | <Ident> <FunArgs>; */
*254         obj = UseObject(ident);
 255         GetToken();
 256         switch (token) {
 257         case mc_assign:
*258             if (obj->type != Var)
*259                 Error("宣言されていない変数への代入はできません");
 260             GetToken();
 261             Exp();
 262             if (token != mc_semicolon)
 263                 Error("代入文に ; がありません");
 264             GetToken();
 265             break;
 266         case mc_lparen:
*267             if (obj->type == Var)
```

```
*268                Error("変数に対する関数呼び出しはできません");
 269            FunArgs();
 270            if (token != mc_semicolon)
 271                Error("関数呼び出し文に ; がありません");
 272            GetToken();
 273            break;
 274        default:
 275            Error("代入文の =、または、関数呼び出し文の ( がありません");
 276            break;
 277        }
 278        break;
 279    case mc_return:
 280        /* return <Exp>; */
 281        GetToken();
 282        Exp();
 283        if (token != mc_semicolon)
 284            Error("return 文に ; がありません");
 285        GetToken();
 286        break;
 287    case mc_input:
 288        /* input <Ident>; */
 289        GetToken();
 290        if (token != mc_ident)
 291            Error("input 文に変数名がありません");
*292        obj = UseObject(ident);
*293        if (obj->type != Var)
*294            Error("input 文の変数が宣言されていません");
 295        GetToken();
 296        if (token != mc_semicolon)
 297            Error("input 文に ; がありません");
 298        GetToken();
 299        break;
 300    case mc_print:
 301        /* print <Exp>; */
 302        GetToken();
 303        Exp();
 304        if (token != mc_semicolon)
 305            Error("print 文に ; がありません");
 306        GetToken();
 307        break;
 308    case mc_if:
 309        /* if ( <Cond> ) <Statement> [ else <Statement> ] */
 310        GetToken();
 311        if (token != mc_lparen)
 312            Error("if の後に ( がありません");
 313        GetToken();
 314        Cond();
 315        if (token != mc_rparen)
 316            Error("if の条件式の後に ) がありません");
 317        GetToken();
 318        Statement();
 319        if (token == mc_else) {
 320            GetToken();
 321            Statement();
 322        }
 323        break;
 324    case mc_while:
 325        /* while ( <Cond> ) <Statement> */
 326        GetToken();
 327        if (token != mc_lparen)
 328            Error("while の後に ( がありません");
 329        GetToken();
 330        Cond();
 331        if (token != mc_rparen)
```

```
 332             Error("while の条件式の後に ) がありません");
 333         GetToken();
 334         Statement();
 335         break;
 336     case mc_lcb:
 337         /* "{" [ <VarDef> ] { <Statement> } "}" */
 338         Block();
 339         break;
 340     case mc_semicolon:
 341         Error("空文は許されていません");
 342         break;
 343     default:
 344         Error("文を始めるべき記号に誤りがあります");
 345         break;
 346     }
 347 }
 348
 349 static void VarDef(void)
 350 {
 351     GetToken();
 352     if (token != mc_ident)
 353         Error("宣言される変数名がありません");
*354     NewVar(ident);
 355     GetToken();
 356     while (token == mc_comma) {
 357         GetToken();
 358         if (token != mc_ident)
 359             Error("宣言される変数名がありません");
*360         NewVar(ident);
 361         GetToken();
 362     }
 363     if (token != mc_semicolon)
 364         Error("変数宣言に , か ; がありません");
 365     GetToken();
 366 }
 367
 368 static void FunDef(void)
 369 {
*370     Offset offset;
*371     Object *obj;
 372
*373     obj = UseObject(ident);
*374     if (obj->type == Var)
*375         Error("変数名を関数名として再定義することはできません");
*376     if (obj->type == Fun)
*377         Error("関数を再定義することはできません");
*378     obj->type = Fun;
 379     GetToken();
 380
*381     NewScope((Offset)0);
 382
 383     if (token != mc_lparen)
 384         Error("関数定義の引数リストに ( がありません");
 385     GetToken();
 386     if (token != mc_rparen) {
 387         if (token != mc_ident)
 388             Error("関数定義の引数リストに文法誤りがあります");
*389         obj = NewVar(ident);
 390         GetToken();
 391         while (token == mc_comma) {
 392             GetToken();
 393             if (token != mc_ident)
 394                 Error("関数定義の引数リストに文法誤りがあります");
*395             obj = NewVar(ident);
 396             GetToken();
```

```
 397         }
 398         if (token != mc_rparen)
 399             Error("関数定義の引数リストに ) がありません");
 400     }
 401     GetToken();
 402
*403     offset = 2 * sizeof(Word);  /* FP から最後の引数までのオフセット */
*404     /* 最後の引数から順に、引数のオフセットを設定 */
*405     for (obj = CurrentScope->objects; obj != (Object *)0; obj = obj->next) {
*406         obj->offset = offset;
*407         offset += sizeof(Word);
*408     }
 409
 410     Block();
 411
*412     ExitScope();
 413 }
 414
 415 static void Block(void)
 416 {
 417     if (token != mc_lcb)
 418         Error("ブロックが { で始まっていません");
*419     NewScope(CurrentScope->offset);
 420     GetToken();
 421     if (token == mc_int)
 422         VarDef();
 423     while (token != mc_rcb)
 424         Statement();
 425     GetToken();
*426     ExitScope();
 427 }
 428
 429 static void Prog(void)
 430 {
*431     CurrentScope = (Scope *)0;
*432     NewScope((Offset)0);
*433     GlobalScope = CurrentScope;
 434
 435     while (token == mc_int || token == mc_ident) {
 436         if (token == mc_int)
 437             VarDef();
 438         else
 439             FunDef();
 440     }
 441
*442     ExitScope();
 443 }
 444
 445 void Parse(void)
 446 {
 447     GetToken();
 448     Prog();
 449     if (token != mc_eof)
 450         Error("変数宣言や関数定義として認識できません");
 451 }
 452
 453 int main (int argc, char *argv[])
 454 {
 455     if (argc < 2) {
 456         fprintf(stderr, "ソースファイルが指定されていません\n");
 457         exit(1);
 458     }
 459     if (OpenFile(argv[1]) != 0) {
 460         fprintf(stderr, "ソースファイルを読み込むことができません\n");
 461         exit(1);
```

```
462     }
463     Parse();
464     CloseFile();
465     return 0;
466 }
467
```

10.3 演習問題

第 9 回の 4 ページ右上の Minimum C プログラム (関数 `div` の定義) について、コンパイラが第 1 行、第 3 行、第 8 行、第 12 行、第 14 行、第 15 行、第 17 行の処理した完了した時点でのスコープリストと各スコープのオブジェクトリストの状態を、それぞれ今回の 2 ページから 6 ページにあるような図で表しなさい。ただし、関数 `div` の機械語命令は 16 番地から置かれていくものとする。

記号処理・第 10 回・終わり

付録：第 6 回演習問題の解答例

1. たとえば、`/**/*/` という文字列は、$\langle TokenSeq \rangle$ に対して、全体を 1 つのコメント ($\langle Token \rangle$) とする構文木と、`/**/` をコメント ($\langle Token \rangle$) として、残りを 2 つの $\langle Token \rangle$ とする 2 種類の構文木を持つ。

   また、文字列 `x/*y*/123*/` は `scan.c` によって、以下のような 5 つの $\langle Token \rangle$ として認識される。

x	/*y*/	123	*	/
mc_ident	コメント	mc_number	mc_mult	mc_div

2. たとえば、`xy` という文字列は、$\langle TokenSeq \rangle$ に対して、全体を 1 つの $\langle Ident \rangle$ とする構文木と、`x` と `y` をそれぞれ $\langle Ident \rangle$ とする 2 種類の構文木を持つ。

   また、文字列 `whilexyz` は `scan.c` によって、全体が 1 つの $\langle Ident \rangle$ (`mc_ident`) として認識される。

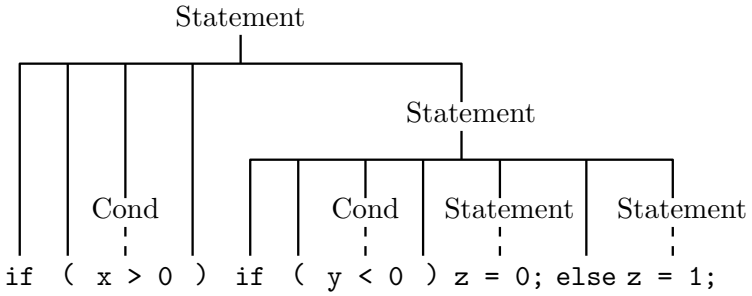
3. 第 7 回配布資料 `parse0.c`

付録：第 7 回演習問題の解答例

1. `else` は、より近い `if` と対応するものとして扱われるようになった。たとえば、

   ```
   if (x > 0) if (y < 0) z = 0; else z = 1;
   ```

は次のような構文木を持つものとして解釈される。



2. (1) 148 行目 `Error("if の条件式の後に ) がありません");`

   (2) 106 行目 `Error("関数呼び出し文に ; がありません");`

   (3) 47 行目 `Error("閉じ括弧があるべき位置にありません");`

   (4) 86 行目 `Error("条件式に文法誤りがあります");`

付録：第 8 回演習問題の解答例

1. 変数 x を 100 番地 (〜 103 番地) に割り当てると

```
 0  IN R1
 4  ST R1, R0, 100
 8  SUB R0, R1, R0
12  JPGE R15, 8
16  SUB R1, R0, R1
20  ST R1, R0, 100
24  OUT R1
28  EXIT R0
```

```
input x;

if (x < 0)

    x = -x;

print x;
```

2. 変数 i、n、sum を、それぞれ 100 番地 (〜 103 番地)、200 番地 (〜 203 番地)、300 番地 (〜 303 番地) に割り当てると

```
 0  IN R1
 4  ST R1, R0, 200
 8  ST R0, R0, 300
12  LDI R1, 1
16  ST R1, R0, 100
20  LD R2, R0, 200
24  SUB R0, R1, R2
28  JPGT R0, 68
32  LD R1, R0, 300
36  LD R2, R0, 100
40  MUL R2, R2, R2
44  ADD R1, R1, R2
48  ST R1, R0, 300
52  LD R1, R0, 100
56  ADDI R1, R1, 1
60  ST R1, R0, 100
64  JP R0, 20
68  LD R1, R0, 300
72  OUT R1
76  EXIT R0
```

```
input n;

sum = 0;
i = 1;

while (i <= n) {


    sum = sum + i*i;




    i = i + 1;


}
print sum;
```