ODC プログラミング言語仕様

佐藤周行

2020/4/22

1　はじめに

本文書は、プログラミング言語 ODC の仕様を定めるものである。DC は、その名前として (Desktop Calculator) にオリジンを持ち、基本的な整数演算をサポートするスクリプト言語である。加えて、関数とリスト、基本的な制御構造を実装している。ODC は、そのように定義される DC にオブジェクトを実装するものとして定義される。

2　用語

本文書で用いる用語のうち、**太文字**で「**しなければならない**」「**してはならない**」「**すべきである**」「**してもよい**」と記されているものは RFC2119[?] の記述に従って使用される。

さらに、以下の用語を用いる。

未定義 本仕様では、従うべき動作を定義しないことを表す。

実装依存 動作を実装ごとに定めて良いことを示す。

3　DCC のコンセプト

3.1　プログラム

プログラムは、行の並びとして定義される。この並びを**トップレベル**という。各行は、**式**、**関数定義**、**クラス定義**、**インポート文**のいずれかからなり、';' で区切られる。

```
1 lines: lines line
2      | line
3      | %empty
```

```
4 line: expr ';'
5     | defun ';'
6     | defclass ';'
7     | importdl ';'
8     | error ';'
```

実装として、行が; に続く改行で区切られることは想定していないが、改行が式の評価結果を返すのは、標準的な実装の一つである。

3.2 実行モデル

プログラムの各要素は、実行時の環境下で左から右に順に評価される。評価の結果、環境が変更されることがある。

変更は、その後の評価における環境に反映されなければならない。

3.3 データ型とクラス

ODC は、データとして整数型をとる。これを基本型とする。さらに関数型をとる。

3.4 リストとオブジェクト

基本型から派生型を定義することができる。

派生型としてリストをとることができる。リストは、連続したデータの並びとして定義される。データの並びは、同じ型を取る必要はない。リストは、その要素としてリストを取ることもできる。この機能によって多次元リストを実現することができる。リストは、全体、またはその要素に対して参照、代入ができる。

クラスは、基本型、関数型、または派生型の集合として定義される。各々の要素はメンバーとして参照、代入ができる。特に関数型を取るメンバーをメソッドと呼ぶ。メソッドは呼び出すことによって定義されている関数を呼び出すことができる。

3.5 データ参照と代入

データと関数は、変数に代入ができる。

3.6　スコープと引数のバインディング

トップレベルにあるかどうかに関わらず、変数は、**定義**された時点から参照可能になる。したがって、変数が **my** または関数仮引数として与えられておらず、定義された場合、その変数は大域変数となる。2 回目以降の代入において、代入されるデータは、それまでその変数によって参照可能であったデータと同じ型である必要はない。

式にはブロックを定義することができる。ブロックの中ではそのブロック内でしか代入、参照ができない変数を **my** によってあらかじめ与えることができる。これを局所変数という。

関数定義において、仮引数を与えた場合、その引数は、その関数の中でのみ参照代入ができる。関数が呼び出されるとき、実引数として与えた式の並びは、呼び出し時に評価され、その値が仮引数と結合され、仮引数の名前で参照できる。これをバインディングという。評価された値がリスト型またはクラスのオブジェクトであるとき、その要素またはメンバーに対する代入は、そのリストまたはオブジェクトの値を変更する。これを**参照渡し**と言い、整数型のデータの参照と区別される。

3.7　制御構造と関数

式は、評価される。式の評価は部分式を評価し、その値に対する演算によって行われる。部分式の評価の順序は規定しない。式には、制御構造を指定することができる。条件による分岐、ループ、ループからの exit によって制御構造を規定する。

式に関数呼び出しを書くことができる。関数をあらかじめ定義し、式の並びを実引数として与えることで、引数のバインディングを通して関数の実行が可能である。

4　データ型とクラス

データは、型を持つ。型には基本型として整数型がある。さらに関数を表現する関数型がある。さらに、それらから派生する型としてリスト型とクラスを持つ。

4.1　整数型

整数型は、一般的な計算機の整数型に対応する。整数型のデータの取り得る範囲は実装依存とする。

4.2 関数型

関数型は、関数を表現するデータ実体を表す型である。関数型のデータは、引数を伴う「呼び出し」を行うことにより、関数の本体を、式の並びと仮引数とを結合させて実行することができる。

4.3 リスト型

リスト型は、データの連続した並びを表現する型である。データの並びは、サイズを指定することができる。並びは、同じ型を取る必要はない。たとえば、下の例 1 は整数型、整数型、リスト型の並びからなる。

例 1 `a = list(3);`
`a[0] = 3;`
`a[1] = 4;`
`a[2] = list(5);`

4.4 クラス

クラスは、名前を持つ型の集合である。各型を持つ値は、名前によって参照できる。名前をメンバーという。関数型のメンバーを特にメソッドという。クラスには名前を付ける。型としてクラスを持つデータ実体をオブジェクトと言う。

例 2 以下のプログラムは、名前として `a` を持つクラスを定義する。`a` はメンバーとして `a1, a2,` メソッドとして `f` を持つ。

```
class a {
   val a1; val a2;
   val f(x) {this.a1+this.a2+x};
   init(x,y) {this.a1=x; this.a2=y}
}
```

5 データ実体

5.1 値

評価は、結果として**値**を生成する。**値**は、整数型、値のリスト、またはオブジェクトとして定義される。

データは変数に結合することができる。結合時の環境下では、その変数は結合された**値**を評価の結果として持た**なければならない**。

多次元のリストはデータである。ODC では、多次元リストは一つ下位の**値**のリストである。

例 3 以下のプログラム片を考える。

```
a=list(5, list(10));
a[3][4] = 5;
```

a[3] は、リスト型 list(10) を *5* 個連続領域に並べたリストの *3* 番目の **value** を示す。a[3][2] は、リスト型 list(10) を *5* 個連続領域に並べたリストの *3* 番目の **value** *(*list(10)*)* の *2* 番目の **value** を表す。

オブジェクトは、その型として与えられるクラスに定義されるメンバーに結合される値の集合である。オブジェクトのメンバーの値を代入するときに、それが同じクラスの他のオブジェクトのメンバーの値を変更**してはならない**。

例 4
```
class a {val a1; val a2; init(x,y) {this.a1 = x; this.a2 = y}};
class b {val b1; a b2}

w = new b();
b.b2 = new a (1,2);
```

w は、new によってクラス b を型として持つデータ実体として定義される。w のメンバー b2 には、クラス a を型として持つオブジェクトをわりつける。w.b2 は、クラス a の初期化ルーチンにより a1 に *1,* a2 に *2* を代入する。

5.2 関数

ODC では関数を定義することができる。関数は、その呼び出しごとに局所的な環境を構築して評価を行う。局所的な環境は、呼び出し時の環境を引き継ぎ、さらにそれにオーバーライドする形で仮引数と実引数の結合、関数定義に定めた局所環境を反映する。構築された局所環境をフレームと呼ぶ。

6 式

6.1 評価

式は、環境の下で評価されて**値**を生成する。

```
29 expr: assignment
30     | ifstatement
31     | loopstatement
```

```
32      | '{' compound '}'
33      | aexpr
34      | condexpr
35      | RE expr
36      | MY '(' varlist ')'
37      | DIGIT
38      | VARIABLE
39      | VARIABLE '(' exprlist ')'
40      | listref
41      | objref
42      | %empty
```

6.2 定数

```
DIGIT: [-][0-9]+
```

定数は、10 進数で表現された整数である。

6.3 変数

```
38      | VARIABLE
```

制約

1. 予約語を変数に使ってはならない。

予約語のリストは**??**を参照すること。

6.3.1 名前

変数は式である。変数が参照されると、それが格納されている値を値として持つ。

6.4 算術式

算術式は、式による「演算」か らなる。

```
11 aexpr: expr '+' expr
12      | expr '-' expr
13      | expr '*' expr
14      | '(' expr ')'
15      | '-' expr
```

6.4.1 演算

演算は、整数をオペランドとして取る 2 項加減乗除演算と符号を逆転させる – からなる。

6.4.2 算術式の評価

算術式は、その部分式を評価したものに対して演算を実行したものをその値として持つ。評価の際、演算は、)、*，/，-，+の順に高い優先度を持つ。

式の意味はそれぞれ整数間のそれぞれの演算とする。たとえば、 expr '+' expr は、2 つの expr それぞれが評価され、結果の**値**の足し算の結果が式の評価された**値**になる。2 つの部分式の評価において、一方の評価時に環境の変化が生じた場合、それが他方の評価時の環境に反映されるかどうかは、**実装依存**とする。

式の評価が、その型がカバーする値の範囲に収まらないときに、どのような値を取るかは**実装依存**とする。

6.5 条件

条件は、式である。

```
16 condexpr: expr GE expr
17         | expr GT expr
18         | expr LE expr
19         | expr LT expr
20         | expr EQ expr
21         | expr NE expr
22         | NOT expr

GE: '>='
GT: '>'
LE: '<='
LT: '<'
EQ: '=='
NE: '!='
NOT: '!'
```

6.5.1 条件の評価

条件は、評価され、それぞれの式の比較をして、比較が正しければ 1, 正しくなければ 0 を値として持つ。2 つの部分式の評価において、一方の評価時

に環境の変化が生じた場合、それが他方の評価時の環境に反映されるかどうかは、**実装依存**とする。

6.6 代入

代入は式であり、評価の結果、右辺を評価した値を値として持つ。

```
 23 lhs: VARIABLE
 24    | objref
 25    | listref

 26 assignment: lhs ASSIGN expr
 27           | lhs ASSIGN newobject
 28           | lhs ASSIGN listdef

ASSIGN: '='
      : ':='
```

以下の例 4 は代入文の例である。

例 5
```
a = 3+4;
w[3] = 4;
u.a1 = w[3];
```

6.6.1 代入式の評価

代入式を評価すると、(1) 右辺に現れる式がまず評価され、(2) 左辺に現われる値を格納する「場所」にその値が代入される。代入式が評価された後は、代入された値は、左辺の形により、変数、リスト、オブジェクトの参照によってそれぞれ参照することができる。

6.6.2 左辺

左辺とは、代入式の ASSIGN 演算子の左側に現れるものであり、変数 VARIABLE、リスト参照 listref、オブジェクト参照 objref のいずれかを取る。これらは、実行時に一定の大きさを持つ場所を占有し、値を格納できるものでなければならない。変数は、固有の名前を持つ。変数への代入が、他の名前の異なる変数に格納された値に影響してはならない。

6.6.3 リストとオブジェクトの生成

代入の特別な形としてリストの生成とオブジェクトの生成がある。生成されたリストまたはオブジェクトは、左辺に代入することで初めて参照することができるようになる。

6.7 条件式

```
43 ifstatement: '(' expr ')' '?' expr ':' expr
44            | IFIF '(' expr ')' expr
45            | IFIF '(' expr ')' expr ELSE expr

IFIF: 'if'
ELSE: 'else'
```

条件式は、(式 1)?式 2:式 3、if (式 1) 式 2、if (式 1)　式 2 else 式 3 の形式をとる。いずれも式 1 がまず評価され、その値が 0 以外の場合は式 2 を、0 でかつ式 3 が与えられている場合は式 3 を評価して全体の値とする。式 3 が与えられていない場合かつ式 1 の値が 0 である場合の値は未定義である。式 1 に条件を書く必要はなく、一般の式を取ることができる。

式 1 の値が 0 の場合、式 2 が評価されて**はならない**。式 1 の値が 0 以外の場合、式 3 が評価されて**はならない**。

6.8 ループ式

```
46 loopstatement: WH '(' expr ')' expr
47              | VARIABLE ':' WH '(' expr ')' expr
48              | BRK
49              | RDO
50              | BRK VARIABLE
51              | RDO VARIABLE

WH: 'while'
BRK: 'break'
RDO: 'redo'
```

ループ式の先頭に書く変数をラベルと言い、そのループ式のラベルと言う。ラベルは、break または redo で指定することができる。

制約

- break または redo で指定したラベルは、それが属するループ式のラベルの一つでなければならない。

6.9 ループ式の評価

ループ式は `while（式 1） 式 2` または `label: while（式 1） 式 2` の形をしている。この式が評価されると

1. まず式 1 が評価される。
2. 評価の値が 0 以外であれば、式 2 を評価する。評価の値を暫定的な結果の値とし、1. に戻る。
3. 評価の値が 0 であれば、今までの暫定的な結果の値を結果の値とする。

暫定的な結果の値が評価されずに式の評価が終了した場合の式の値は**未定義**とする。

式の評価は、他の場合と同じであるが、break と redo については以下のような意味を持つ。

6.9.1 break label

break label を評価すると、label をラベルとして持つループ式の評価をそのまま終了する。評価の値は、break を実行する直前の暫定的な結果の値とする。

例 6 以下のループ式を評価することを考える。

```
x = 3;
lab1: while (x > 0) {i = 3; x = x-1;
      lab2: while (i > 0) {i=i-1;if (x==2) {break lab1;}}
}
x;
```

`x` の値は *2* になる。

6.9.2 redo label

redo label を評価すると、label をラベルとして持つループ式の評価をはじめから再度開始する。ただし、それまでの式の評価の結果はそのまま反映されるものとする。

6.10 関数呼び出し

関数呼び出しは `VARIABLE '(' exprlist ')'` の形をとり、あらかじめ定義され変数 `VARIABLE` に割り当てられた関数に、式の並び `exprlist` を引数として与えて評価した値を値として持つ。

6.11 参照

6.11.1 変数参照

変数を評価すると、その変数に代入によって格納された値がそのまま値となる。

6.11.2 リスト参照

リスト、またはその要素を参照すると、添字の式がまず評価され、その値に対応する「場所」に格納された値が s のいまま値となる。この評価は再帰的に行われ、それで多次元のリストの参照が可能になる。

```
74 listref: VARIABLE '[' expr ']'
75        | listref '[' expr ']'
```

添字が、リストの生成時のサイズを超えるものであった場合の動作は**未定義**とする。

6.11.3 オブジェクト参照

```
67 objref: objref '.' VARIABLE
68       | objref '.' VARIABLE '(' exprlist ')'
69       | objref '.' listref
70       | THS
71       | VARIABLE
```

制約

1. (67) (68) VARIABLE は、objref で指定されたオブジェクトのクラスのメンバーでなければならない。
2. (69)listref 中の VARIABLE 部分は、objref で指定されたオブジェクトのクラスのメンバーでなければならない。
3. (71) VARIABLE は、クラスを型として持たなければならない。

```
THS: 'this'
```

オブジェクトは、それを格納した変数のクラスのメンバーを指定することで、オブジェクトのメンバーの値を参照することができる。メンバーが関数、つまりメソッドの場合、引数として与えられた式の並びが評価され、それがメソッドに関数引数として与えられて評価され、評価の結果を値として持つ。

`this` は、オブジェクト自身を指す。

6.12 複合式とブロック

式は';' で並べて、連続させて評価することができる。これを複合式と言う。

```
 9 compound: compound ';' expr
10         | expr

32     | '{' compound '}'
36     | MY '(' varlist ')'

MY: 'my'
```

複合式は、{} でくくることでブロックを作ることができる。ブロックの最初に my で、局所変数の並びを指定すると、そのブロックだけのスコープを持つ変数を宣言することができる。複合式の値は、最後に評価された式の値とする。したがって、例えば`{a=3;}` の式の値は 3 ではなく 0 である。

例 **7** 以下のプログラムの実行結果は *3* を値に持つ。

```
a = 3;
{my(a,b,c); a = 4;}
a;
```

7 定義と参照

7.1 変数

7.2 Name

変数名は任意長の alphanumeric 文字からなるな文字列とする。ただし、先頭文字は alpabet でなければならない。

7.3 定義とスコープ

変数に最初に値を代入することを「定義」と言う。ODC では、変数は宣言する必要はない。宣言されずに代入または参照が起こると、その変数はそれ以後、すべての式の評価で代入、参照ができる。これを「大域変数」と言う。

ブロック、または関数定義の中で `my` を用いて変数を宣言することができる。このように宣言された変数は、そのブロック、または関数の中だけで代入、参照ができる。これを「局所変数」と言う。

7.4 関数の参照と代入

関数は、定義とともに関数名と結合される。関数名は、変数として参照することができ、他の変数に代入することができる。

7.5 リストの参照と代入

リストは、定義とともに変数と結合される。リストは、その添字を [] で指定し、リスト内の要素を参照することができる。

リストが、listdef 中、n 重にネストされて定義されているとき、リストの次元は n であるという。下の定義に従えば、list のネスト数に対応する。

```
72 listdef: LIST '(' expr ')'
73        | LIST '(' expr ',' listdef ')'
```

この時、$m(< n)$ 重に添字を重ねると、部分リストを表す。この部分リストは参照することができて、値を代入することができる。

例 8 以下のプログラム片を考える。

```
x = list(10, list(10));
x[9] = 100;
```

ここで、x は通常 x[i][j] の形で値を参照することができるが、x[9] は、リストではなく、整数型の値 *100* を値として持つ。

このようにして、（各要素の型が異なるという意味で）ヘテロなリストを表現することができる。

7.6 オブジェクトの参照と代入

オブジェクトは、定義とともに変数と結合される。定義をするときには、対象となるクラスがあらかじめ定義されていなければならない。

オブジェクトの参照は、変数への参照で行う。オブジェクト内のメンバーの参照は、'.' に続けてクラスのメンバー名を指定することで可能である。クラスのメンバー名は、そのクラスに局所的であるとする。すなわち、異なるクラスで同じメンバー名を使っても良い。

8 関数

関数は、トップレベルで定義する。

```
 76 defun: DE VARIABLE '(' varlist ')' expr

 55 varlist: VARIABLE ',' varlist
 56        | VARIABLE
 57        | %empty

DE: 'def'
```

VARIABLE に相当する名前を関数名と言う。expr に相当する式を関数の本体と言う。

8.1 関数内でのスコープ

関数は、複合文と同じスコープを持つ。すなわち、my で定義することで関数呼び出しごとに局所的なスコープを持つ変数を宣言することができる。さらに、仮引数のリストで与えられた仮引数は、その関数に局所的なスコープを持つものとする。

関数名は、その本体内で有効なスコープを持つ。したがって、以下の例は、関数 fib の再帰的な宣言になっている。

例 9 `def fib(n) {if (n==1) 1 else {if (n==2) 1 else {fib(n-1)+fib(n-2)}}};`

8.2 関数呼び出し

関数は、() 内に式の並びを書いて、呼び出すことができる。

```
 39     | VARIABLE '(' exprlist ')'

 52 exprlist: expr ',' exprlist
 53         | expr
 54         | %empty
```

関数が呼び出されると、まず引数として与えた式の並びが評価される。並びをどの順序で評価するかは**実装依存**とする。評価することで変数の値が変化する場合、その変化がどのように反映されるかも実装依存である。

例 10 次の関数呼び出しを考える。

```
a=2;
f(a=a+1, a=a+1);
```

これが *f(3,4)* になるか、*f(4,3)* になるか、またある最適化の結果 *f(3,3)* になるかについては実装依存である。

8.3 return

関数呼び出し内で `return` expr が評価されると、expr が評価され、それが関数の評価の値とされる。関数は、それ以上本体部分の実行をすることはない。

トップレベルで return を実行してはならない。

9 オブジェクト

```
58 methoddef: VL VARIABLE
59          | VL VARIABLE '(' varlist ')' expr
60          | OBJINIT '(' varlist ')' expr
61          | VARIABLE VARIABLE
62          | VARIABLE VARIABLE '(' varlist ')' expr

63 methodlist: methoddef ';' methodlist
64           | methoddef
65           | %empty

66 newobject: NEW VARIABLE '(' exprlist ')'

77 defclass: CLASS VARIABLE '{' methodlist '}'

VL: 'val'
OBJINIT: 'init'
NEW: 'new'
```

9.1 オブジェクトの生成

オブジェクトは、以下のようにして左辺に結合する。変数に結合した後は、その左辺を参照することで参照できる。

```
assignment:...
       | lhs ASSIGN newobject
newobject: new classname '(' exprlist ')'
```

9.1.1 init

クラスの定義に init メソッドが指定されていた場合は、オブジェクトの生成時に同時に init が関数として実行される。

例 **11** 以下のプログラムを実行すると、変数 w には、メンバー a1 には *1,* メンバー a2 には *2* を値として持つオブジェクトが結合される。

```
class a {val a1; val a2; init(x,y) {this.a1 = x; this.a2 = y}}

w = new a(1, 2);
```

9.2 メンバーの参照

オブジェクト内のメンバーの参照は、'.' に続けてクラスのメンバー名を指定することで可能である。メンバーの型がクラスであるとき、さらに'.' に続けてメンバーの指定ができる。

9.3 メソッドの呼び出し

メンバーが関数定義の形をしているとき、これを特にメソッドと言う。メソッドには引数とともに「メソッド呼び出し」によって評価し、値を得ることができる。

メソッドがオブジェクトを値として持つことがある。この値は、いったん変数と結合しないと、それ以後参照ができなくなる。

例 **12** 以下のプログラムを考える。

```
class a {val a1};
class b {val b1; a b2; val f() {this.b2 = new a();this}};

w = new b();
w.f().b2.b1 = 3;
```

ここで代入された値 *3* は、w から w.b2.b2 として参照できる。

10 リスト

```
72 listdef: LIST '(' expr ')'
73        | LIST '(' expr ',' listdef ')'

74 listref: VARIABLE '[' expr ']'
75        | listref '[' expr ']'
```

LIST: 'list'

10.1　リストの生成

リストは、list により、定義によって左辺値と結合できる。

例 13 以下のプログラムは、連続した *10* 個の整数の領域を取るリストの「アドレス」を *10* 個並べることによって 10×10 の *2* 次元配列を実現している。連続した *100* 個の領域を確保するわけではない。

```
a = list(10, list(10));
```

10.2　リストの参照と定義

ここで、n は 0 から始まるとする。

リストの各要素は、添字に式を指定することにより、評価の結果（n とする）に対して、当該リストの n 番目の要素を参照することができる。

これは再帰的に適用され、結果的に多次元配列の要素の参照を可能にする。

例 14
```
x = list(10, list(10));
i = 0;
while (i < 10) {
 j = 0;
 while (j < 10) {
   x[i][j]= i+j;
   j = J+1;
 }
 i = i+1;
};
```

この例では、参照 x[i]/がまずなされ、その結果として与えられるリスト *verb/x[i]/*の j 番目の要素が参照される。

11　スコープと引数のバインディング

データを表す変数には、プログラム中、参照可能な場所が定義される。この場所の範囲をスコープという。

ODC では、値を表し、名前の異なる変数への代入によって変数に結合されている値が変更されることはない。

11.1　大域変数と局所変数

スコープがプログラム全体である変数を「大域変数」と言う。それ以外を「局所変数」と言う。変数が局所変数になるには、以下の場合がある。いずれの場合も変数は、代入または参照前に宣言されなければならない。

1. ブロックの先頭で my により宣言された場合。宣言した変数は、そのブロック内で参照可能である。同じ名前で、そのブロックの外にある変数は参照できなくなる。

2. 関数の仮引数。仮引数は、定義される関数の本体でのみ参照可能である。

11.2　変数の定義

大域変数は、最初に代入または参照された時点からその代入または参照が有効である。この最初の代入または参照をその大域変数の定義という。

関数名を表す変数は大域変数でなければならない。クラス名を表わす変数は大域変数でなければならない。

11.3　関数引数のバインディング

関数呼び出しは、関数名に実引数の並びをつけて行う。この時に、実引数を評価した値の並びと、関数定義中の仮引数の並びが結合されて本体中にその値が参照できる。

実引数の評価の結果としての値が整数型または関数型の場合、関数の本体の実行中に、その値が結合された仮引数に代入を行った場合、実引数として与えられたデータの実体が変更されてはならない。

実引数の評価の結果としての値がリストまたはオブジェクトの場合、関数の本体の実行中に、その値が結合された仮引数にリストの参照、またはオブジェクトの参照により代入を行った場合、実引数として与えられたデータの実体が変更される。

12　インポート

ODC は、他で定義された関数を利用することができる。この機能は import で実現される。この機能を用いて利用可能になる関数の集合をモジュールと言う。

```
78 importdl: IMPORT VARIABLE
```

制限

1. (78) インポートはトップレベルで行わなければならない。
2. (78) インポートするのは、関数に限る。

VARIABLE がモジュール名に対応する。モジュールは関数のみの集合でなければならない。

12.1 ODC 関数のインポート

ODC 関数を別ファイルで定義しているときに、その関数をファイル名を指定してインポートすることにより、利用可能になる。

例 **15** `ini.dc` が次のように定義されているとする。

```
def fac(n) {if (n==0) 1 else n*fac(n-1)};
```

このプログラムは、値 *6* を持つ。

```
import ini;
fac(3);
```

外部ファイル名とモジュール名の結合の仕方は実装依存とする。

12.2 C 関数のインポート

ODC は、C 関数とのインポートができる。インポートする関数は、プレフィックス `dc_`を持ち、第 1 引数は、引数の数、第 2 引数が引数の並びでなければならない。

例 **16** 以下の関数が *C* でコンパイルされ、`fac.so` として定義されているとする。

```
dc_factorial(int ac, char av[]);
{
  return myfactorial(av[0]);
}

myfactorial(int n)
{
  return n * myfactorial(n-1);
}
```

この関数は、以下のようにして参照可能である。

```
import fac;

factorial(3);
```

外部ファイル名とモジュール名の結合の仕方は実装依存とする。

13 終わりに

プログラミング言語 ODC を定義した。

A 文法

```
 1 lines: lines line
 2      | line
 3      | %empty

 4 line: expr ';'
 5     | defun ';'
 6     | defclass ';'
 7     | importdl ';'
 8     | error ';'

 9 compound: compound ';' expr
10         | expr

11 aexpr: expr '+' expr
12      | expr '-' expr
13      | expr '*' expr
14      | '(' expr ')'
15      | '-' expr

16 condexpr: expr GE expr
17         | expr GT expr
18         | expr LE expr
19         | expr LT expr
20         | expr EQ expr
21         | expr NE expr
```

```
22         | NOT expr

23 lhs: VARIABLE
24     | objref
25     | listref

26 assignment: lhs ASSIGN expr
27           | lhs ASSIGN newobject
28           | lhs ASSIGN listdef

29 expr: assignment
30     | ifstatement
31     | loopstatement
32     | '{' compound '}'
33     | aexpr
34     | condexpr
35     | RE expr
36     | MY '(' varlist ')'
37     | DIGIT
38     | VARIABLE
39     | VARIABLE '(' exprlist ')'
40     | listref
41     | objref
42     | %empty

43 ifstatement: '(' expr ')' '?' expr ':' expr
44            | IFIF '(' expr ')' expr
45            | IFIF '(' expr ')' expr ELSE expr

46 loopstatement: WH '(' expr ')' expr
47              | VARIABLE ':' WH '(' expr ')' expr
48              | BRK
49              | RDO
50              | BRK VARIABLE
51              | RDO VARIABLE

52 exprlist: expr ',' exprlist
53         | expr
54         | %empty
```

```
55 varlist: VARIABLE ',' varlist
56        | VARIABLE
57        | %empty

58 methoddef: VL VARIABLE
59          | VL VARIABLE '(' varlist ')' expr
60          | OBJINIT '(' varlist ')' expr
61          | VARIABLE VARIABLE
62          | VARIABLE VARIABLE '(' varlist ')' expr

63 methodlist: methoddef ';' methodlist
64           | methoddef
65           | %empty

66 newobject: NEW VARIABLE '(' exprlist ')'

67 objref: objref '.' VARIABLE
68       | objref '.' VARIABLE '(' exprlist ')'
69       | objref '.' listref
70       | THS
71       | VARIABLE

72 listdef: LIST '(' expr ')'
73        | LIST '(' expr ',' listdef ')'

74 listref: VARIABLE '[' expr ']'
75        | listref '[' expr ']'

76 defun: DE VARIABLE '(' varlist ')' expr

77 defclass: CLASS VARIABLE '{' methodlist '}'

78 importdl: IMPORT VARIABLE
```

B　予約語

以下の語は予約語とする。変数名に使用してはならない。

- while

- def
- my
- return
- if
- else
- for
- import
- list
- class
- new
- this
- init
- break
- redo