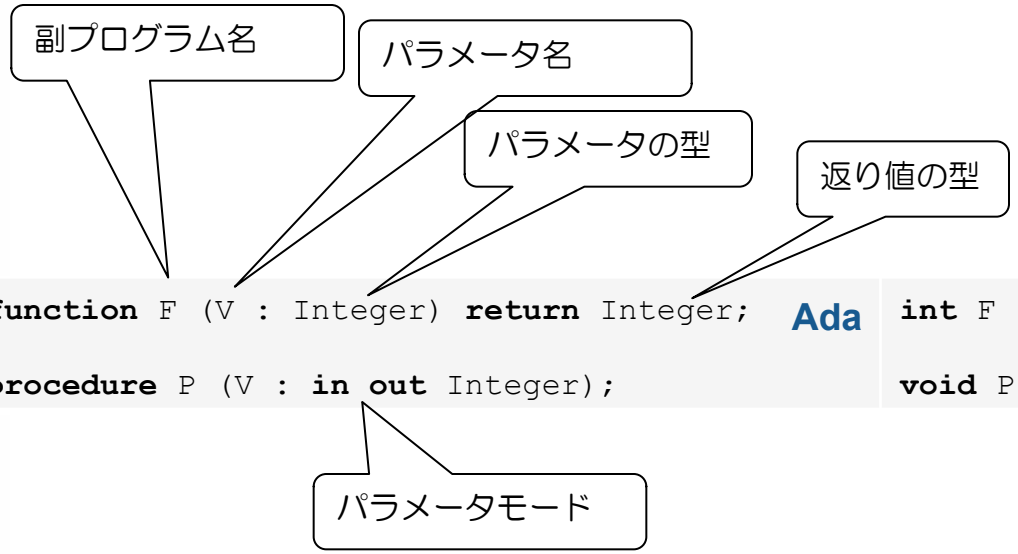
副プログラム

Presented by Quentin Ochem

university.adacore.com

Adaにおける副プログラム: 仕様部



Ada

```
function F (V : Integer) return Integer;

procedure P (V : in out Integer);
```

C++

```
int F (int V);

void P (int *V);
```

- Adaは，関数（値を返す）と手続き（値を返さない）を区別します

Adaにおける副プログラム: 宣言とボディ部

宣言

```
function F (V : Integer) return Integer;
```

ボディ部

```
function F (V : Integer) return Integer is
   R : Integer := V * 2;
begin
   R := R * 2;
   return R - 1;
end F;
```

- 宣言は必須ではありません．しかし，利用する前には，宣言が必要です
- 関数の返値を無視することはできません
- 完備化（completion）/ ボディ部の実体は，「is」以降に記述します

パラメータモード

- モード「in」
  - 実パラメータが，変更されないことを仕様化しています
  - 仮パラメータは，読み込みのみが可能です
  - デフォルト（パラメータ）モードになります
- モード「out」
  - 実パラメータは，更新される可能性があります
  - 仮パラメータは，書き出し用ですが，値の読み出しも可能です
  - 初期値は定義されていません
- モード「in out」
  - 実パラメータは，読み出しと変更が行われます
  - 仮パラメータは，読み込みと更新が行われます

```
function F (V : in out Integer) return Integer is
   R : Integer := V * 2;
begin
   V := 0;
   R := R * 2;
   return R - 1;
end F;
```

パラメータ受け渡しメカニズム

- コピー渡し（by-copy）と参照渡し（by-reference）
- コピー渡し
  - 仮パラメータは，実パラメータと異なるオブジェクトとなります
  - 実パラメータのコピーが呼び出し前に仮パラメータに設定されます
  - 仮パラメータのコピーは，呼び出し後，実パラメータにコピーされます
- 参照渡し
  - 仮パラメータは実パラメータと同じものを参照します
  - 仮パラメータの更新は，実パラメータに直接，影響を与えます
- パラメータの上記選択の制御機構
  - 相当するパラメータモードはありません

規格化されているパラメータ受け渡し規則

- 「コピー渡し」の型
  - スカラー型
  - アクセス型
  - 「コピー渡し」型として，完全に定義されている非公開型
- 「参照渡し」の型
  - タグ付き型
  - タスク型および保護型
  - 限定型
  - 参照渡しの要素型を持つ複合型
  - 「参照渡し」型として，完全に定義されている非公開型
- 実装定義の型
  - 「コピー渡し」要素のみを含んでいる配列型
  - 「コピー渡し」要素のみを含んでいる非限定レコード型
  - 効率的な方法による実装選択

サブプログラム呼び出し

- パラメータがない場合，括弧「（）」はつけません

```
function F return Integer;

V : Integer := F;
```

- 名前付き引数を利用可能です

```
procedure P (A, B, C : Integer);

P (B => 0, C => 0, A => 1);
```

- out モードと in out モードでは，オブジェクトが必要です

```
procedure P (X : out Integer);

V : Integer;

P (V);
```

デフォルト値

- 「in」パラメータは，デフォルト値を持つことができます

```
procedure P (A : Integer := 0; B : Integer := 0);
```

- デフォルト値は，動的な式です．明示的な式が与えられていなければ，呼び出し時点で評価されます

```
P;           -- A = 0, B = 0;
P (1);       -- A = 1, B = 0;
P (B => 2); -- A = 0, B = 2;
P (1, 2);    -- A = 1, B = 2;
```

不定パラメータと返り値

- サブプログラムは，不定パラメータと返り値を持つことができます

```
function Comment (Stmt : String) return String is
begin
   return "/*" & Stmt & "*/";
end Comment;

S : String := Comment ("a=0"); -- return /*a=0*/
```

- 制約は呼び出し時点で計算されます
- 境界に前提をおいてはいけません

```
   procedure Init (Stmt : in out String) is
   begin
      for J in 1 .. Stmt'Length loop -- Incorrect
         Stmt (J) := ' ';
      end loop;
   end Init;

   S : String := "ABCxxx";
begin
   Init (S (4 .. 6));
```

別名化

- Adaは，「明らかな」別名化のエラーを見つけることができます

```
   function Change (X, Y : in out Integer) return Integer is
   begin
      X := X * 2;
      Y := Y * 4;

      return X + Y;
   end;

   One : Integer := 2;
   Two : Integer := 4;

begin

   Two := Change (One, One);
   --  warning: writable actual for "X" overlaps with actual for "Y"

   Two := Change (One, Two) - Change (One, Two);
   --  warning: result may differ if evaluated after other actual in expression
```

オーバーロード(1/2)

- Adaでは，サブプログラムのオーバーロードを使用できます

```
procedure Print (V : Integer);
procedure Print (V : Float);
```

- 次の点で仕様が異なれば，オーバーロードを使用できます
  - パラメータの数
  - パラメータの型
  - 結果の型

```
subtype Positive is Integer range 1 .. Integer'Last;
procedure Print (V : Integer);
procedure Print (W : out Positive); -- NOK
```

- 仕様において下記のアスペクトの違いは，オーバーロードの対象となりません
  - パラメータの名前
  - パラメータの副型
  - パラメータモード
  - パラメータデフォルト式

オーバーロード(2/2)

- オーバーロードを用いると，呼び出し時点であいまいさが生じる場合があります
- 情報を付加することで，このあいまいさを取り除くことができます

```
   type Apples is new Integer;
   type Oranges is new Integer;


   procedure Print (Nb_Apples : Apples);
   procedure Print (Nb_Oranges : Oranges);

   N_A : Apples := 0;

begin

   Print (N_A);              -- OK
   Print (0);                -- NOK
   Print (Oranges'(0));      -- OK
   Print (Nb_Oranges => 0); -- OK
```

演算子オーバーロード

- デフォルト演算子（=, /=, *, /, +, -, >, <, >=, <=, and, or…）は，型に関して，オーバーロードしたり・加えたり・取り除くことができます

```
type Distance is new Float;
type Surface is new Float;

function "*" (L, R : Distance) return Distance is abstract; -- removes "*"
function "*" (L, R : Surface) return Surface is abstract;   -- removes "*"
function "*" (L, R : Distance) return Surface;              -- adds "*"

type Rec is record
   Unimportant_Field : Integer;
   Important_Field : Integer;
end record;

function "=" (L, R : Rec) return Boolean is
begin
   return L.Important_Field = R.Important_Field;
end "=";
```

- 「=」のオーバロードは，自動的に関連する「/=」のオーバーロードを生成します

副プログラムのネスト化と広域変数へのアクセス

- 全てのスコープで副プログラムをネスト化することができます
- ネスト化された副プログラムは，親の副プログラムパラメータや既に宣言されている変数にアクセスできます

```
procedure P (V : Integer) is
   W : Integer;

   procedure Nested is
   begin
      W := V + 1;
   end Nested;
begin
   W := 0;
   Nested;
```

ヌル（null）手続きと式関数

- ボディ部を持たない副プログラムは，ヌルとして宣言することができます（実際の利用法は後述します）

```
procedure P (V : Integer) is null;
```

- 式の評価のみからなる関数は，仕様部レベルで完結することができます．これを式関数と呼びます

```
function Add (L, R : Integer) return Integer is (L + R);
```

- これは，事前／事後条件を記述するための便利な方法です

Quiz

正しいですか　(1/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
   function F (V : Integer) return Integer is
   begin
      Put_Line (Integer'Image (V));
      return V + 1;
   end F;

begin

   F (999);
```

正しいですか (1/10)

いいえ

```
function F (V : Integer) return Integer is
begin
   Put_Line (Integer'Image (V));
   return V + 1;
end F;

begin

   F (999);
```

関数 F は呼ばれますが，返り値を保持していません

正しいですか (2/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
procedure P (V : Integer) is
begin
   V := V + 1;
end P;
```

正しいですか (2/10)

いいえ

```
procedure P (V : Integer) is
begin
   V := V + 1;
end P;
```

コンパイルエラー
V は，（デフォルトの） in モードであり，更新することはできません

正しいですか (3/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
function F () return Integer is
begin
   return 0;
end F;

V : Integer := F ();
```

正しいですか (3/10)

いいえ

```
function F () return Integer is
begin
   return 0;
end F;

V : Integer := F ();
```

コンパイルエラー
パラメータがない場合，括弧を用いることはできません

正しいですか (4/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
procedure P (V : Integer) is
   procedure Nested is
   begin
      W := V + 1;
   end Nested;

   W : Integer;
begin
   W := 0;
   Nested;
```

正しいですか (4/10)

いいえ

```
procedure P (V : Integer) is
   procedure Nested is
   begin
      W := V + 1;
   end Nested;

   W : Integer;
begin
   W := 0;
   Nested;
```

コンパイルエラー
Wは，この時点ではまだ可視性がありません

正しいですか (5/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
function F return String is
begin
   return "A STRING";
end F;

V : String (1 .. 2) := F;
```

正しいですか (5/10)

いいえ

```
function F return String is
begin
   return "A STRING";
end F;

V : String (1 .. 2) := F;
```

実行時エラー
Vのサイズと F が返す値があっていません

正しいですか (6/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
procedure P (Data : Integer);
procedure P (Result : out Integer);
```

正しいですか (6/10)

いいえ

```
procedure P (Data : Integer);
procedure P (Result : out Integer);
```

パラメータ名とモードの違いだけでは，オーバーロードするには十分ではありません

正しいですか (7/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
   procedure P (V : Integer := 0);
   procedure P (V : Float := 0.0);
begin
   P;
```

正しいですか (7/10)

いいえ

```
   procedure P (V : Integer := 0);
   procedure P (V : Float := 0.0);
begin
   P;
```

コンパイルエラー
単にPと呼び出すだけではあいまいです

正しいですか (8/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
   procedure Multiply
      (R : out Integer; V : Integer; Times : Integer)
   is
   begin
      for J in 1 .. Times loop
         R := R + V;
      end loop;
   end Multiply;

   Res : Integer := 0;
   X : Integer := 10;
begin
   Multiply (Res, X, 50);
```

正しいですか (8/10)

いいえ

```
   procedure Multiply
      (R : out Integer; V : Integer; Times : Integer)
   is
   begin
      for J in 1 .. Times loop
         R := R + V;
      end loop;
   end Multiply;

   Res : Integer := 0;
   X : Integer := 10;
begin
   Multiply (Res, X, 50);
```

誤りやすい
Multiply 手続きに入るとき，R は初期化されていません

正しいですか (9/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
   type My_Int is new Integer;

   function "=" (L, R : My_Int) return Boolean;

   function "=" (L, R : My_Int) return Boolean is
   begin
      if L < 0 or else R < 0 then
         return True;
      else
         return L = R;
      end if;
   end "=";

   V, W : My_Int := 1;
begin
   if V = W then
...
```

正しいですか (9/10)

いいえ

```
   type My_Int is new Integer;

   function "=" (L, R : My_Int) return Boolean;

   function "=" (L, R : My_Int) return Boolean is
   begin
      if L < 0 or else R < 0 then
         return True;
      else
         return L = R;
      end if;
   end "=";

   V, W : My_Int := 1;
begin
   if V = W then
...
```

「＝」演算子に関して，無限の再帰となります
変更することで，問題を解決することができます．例えば，Integer (L) = Integer (R) です

正しいですか　(10/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
   type Rec is record
      A, B : Integer;
   end record;

   function "=" (L : Rec; I : Float) return Boolean;

   function "=" (L : Rec; I : Float) return Boolean
is ...

   A : Rec;
begin
   if A /= 0.0 then
   ...
```

正しいですか　(10/10)

はい

```
   type Rec is record
      A, B : Integer;
   end record;

   function "=" (L : Rec; I : Float) return Boolean;

   function "=" (L : Rec; I : Float) return Boolean
is ...

   A : Rec;
begin
   if A /= 0.0 then
   ...
```

問題ありません
「=」の宣言は，暗黙に，「/=」の宣言を作ります

university.adacore.com