

## 16.2 - C

Cから呼びされる関数のある多くのユーティリティが提供するので、AdaからCをどのように呼び出すかをしておくのはいいアイデアです。このセクションは、プログラマーが関数のようなレベルでCについて知っていることを示しています。 ;  
もしプログラマーがCについて知らないのであれば、このセクションは読みすだけでいいです。

次に、ここでAdaとCをどのようにさせるかについての一般的なルールを示します。これは、[RM B.3\(63\)](#)を参照しています。

1. Adaのプロシージャは、voidを返すC関数にマッピングします。
2. Adaの関数は、voidでない返値を返すC関数にマッピングします。
3. Adaの関数は、Cのポインターの関数のマッピングにマッピングします。
4. 数値型はinteger、floatsとアクセス/ポインター型は、Cの関数のマッピングにマッピングします。

## Ada

95は、Cとのインターフェースを提供するためのパッケージのセットを提供しています。このパッケージは、「Interfaces.C」で、AdaにおけるCのマッピングを提供しています。Cのint、long、unsigned、doubleのマッピングが提供されています。Cのfloatは、「C\_float」とAdaでは提供されており、それは、AdaのFloatと異なるようにするためです。(AdaのFloatとCのfloatは、Cのように見えますが、異なるわけではありません)。

「char\_array」は、Cのcharacterをマッピングします。多くのCのマッピングは、characterが異なるキャラクター「nul」で終わっていることを示しています。Cでは「\0」と書かれます。Adaのマッピングは、\0、ヌルターミネートではないので、To\_CとTo\_AdaがAdaのマッピングとCのchar\_arrayのマッピングをマッピングします。

Interfaces.C.StringsとInterfaces.C.PointersというCスタイルのマッピングとポインターのためのマッピングを提供するためのパッケージが提供されます。次に、「Interfaces.C.Strings」は、「chars\_ptr」を提供しており、これはCのマッピングな「char\*」にマッピングし、Cのマッピングへのポインターにマッピングされます。たとえば、キャラクターへのポインターなどです。

このパッケージは、またももしています。:

1. `Null_Ptr`、これはCの`(char*)NULL`にします。
2. プロシージャ `Free`、これはCの`free()`にします。
3. `Value`、これは`chars_ptr`をにとり、のAdaをします。このは、ヌルポインタがされたには、`Dereference_Error`を上げます。

のをして、プログラマーがこれらをにどのようにするかをていくことにしましょう。このは、「[package CGI](#)」からとったもので、AdaをWWWのコモン・ゲートウェイ・インターフェースCGIとみわせたものです。プログラマーがオペレーティングシステムからのをりたいとっているとしましょう。そして、プログラマーは、のCがこれをすることをじて、のをろうとしていることにします。Cでは、このは「`getenv`」とばれており、これはのようにされています。(のによります [Kernighan and Ritchie 1988, edition 2, page 253]):

```
char *getenv(char *name);
```

これは、かなりそのままAdaにすることが出来ます。:

```
function getenv(Variable : chars_ptr) return chars_ptr; pragma Import(C, getenv);
```

これはしますが、Adaプログラムにおいてinとoutの「`chars_ptr`」のされたをしなければならぬにはです。おそらく、ラッパープログラムをいて、AdaをC(`chars_ptr`)に、まさそのをするようにするがよりいでしょう。では、そのようにするAdaのをします。:

```
with Interfaces.C.Strings; use Interfaces.C.Strings; -- ...
function Get_Environment(Variable : String) return String is
  -- Return the value of the given environment variable.
  -- If there's no such environment variable, return an empty string.
  function getenv(Variable : chars_ptr) return chars_ptr; pragma Import(C, getenv);
  -- getenv is a standard C library function; see K&R 2, 1988, page 253.
  -- it returns a pointer to the first character; do NOT free its results.
  Variable_In_C_Format : chars_ptr := New_String(Variable);
  Result_Ptr : chars_ptr := getenv(Variable_In_C_Format);
  Result : String := Value_Without_Exception(Result_Ptr);
begin
  Free(Variable_In_C_Format);
  return Result;
end Get_Environment;
```

セクションにおいて、たくさんのがすることをてておいてください。これは、のをするためであり、シンプルなAdaはそれらがされるとのさをつためなのです。`Value_Without_Exception`とばれるのへのびしがします。;これは、ヌルのCのポインタをにしようとしたときに、ががるためです。そして、たちはそれをのにしたいとえたわけです。このことは、たちがそのようなをしなければならないということをします。;ここで、そのをします。:

```
function Value_Without_Exception(S : chars_ptr) return String is -- Translate S from a
C-style char* into an Ada String. -- If S is Null_Ptr, return "", don't raise an exception. begin
if S = Null_Ptr then return ""; else return Value(S); end if; end Value_Without_Exception;
pragma Inline(Value_Without_Exception);
```

さて、私たちはAdaにおいてREQUEST\_METHODという環境変数の値を取得できるようにしました。例えば、REQUEST\_METHODという環境変数の値を取得したいのであれば、このようにします。:

```
Request_Method_Text : String := Get_Environment("REQUEST_METHOD");
```

私たちがカバーしていない点に、Cのポインタがあります。AdaのレコードとCのポインタは、はっきりと区別されています。しかし、どのように区別されているのでしょうか。Ada RMは、Adaのレコードがポインタとして扱われるべきとアドバイスをしていますが、それはしていません。

このため、ポインタはありますが、Cのポインタをポインタとして扱っているようなケースへのポインタの代わりにコピーが扱われる点において、プログラマーは、ポインタをポインタに扱うような新しいCのポインタを定義し、そして、Adaからその新しいCのポインタを呼び出すようにします。しかしながら、これは良いアドバイスです。GNATコンパイラは、このアドバイスに従っていません。 -

その代わりに、GNATはAdaのレコードをポインタとしてコピーします。どちらのアプローチも良いものですが、良いことにそれらは区別されています。Adaのレコードを扱うための良いアプローチは、ポインタに「access to record」のポインタを定義することです。 -

それらが、スカラーなのであれば、それらはAdaコンパイラにおいて区別されることなく扱われています。

参照: <http://www.adahome.com/Tutorials/Lovelace/s16s2.htm>