Ada2012 Language Update

Notification

2011年春時点の話です
最終的にどうなるかは
わかりません
あとヲチしてるだけなんで
勘違いが含まれるかも
私見願望も含まれてますが
勘弁ください

Ada2012 Language Update

- C++で言えばADL
- C++で言えばoperator []
- C++で言えばrange-based-for
- C++で言えば[[]]
- Eiffel化！？
- 関数型言語化！？
- ...

Ada2012 Language Update

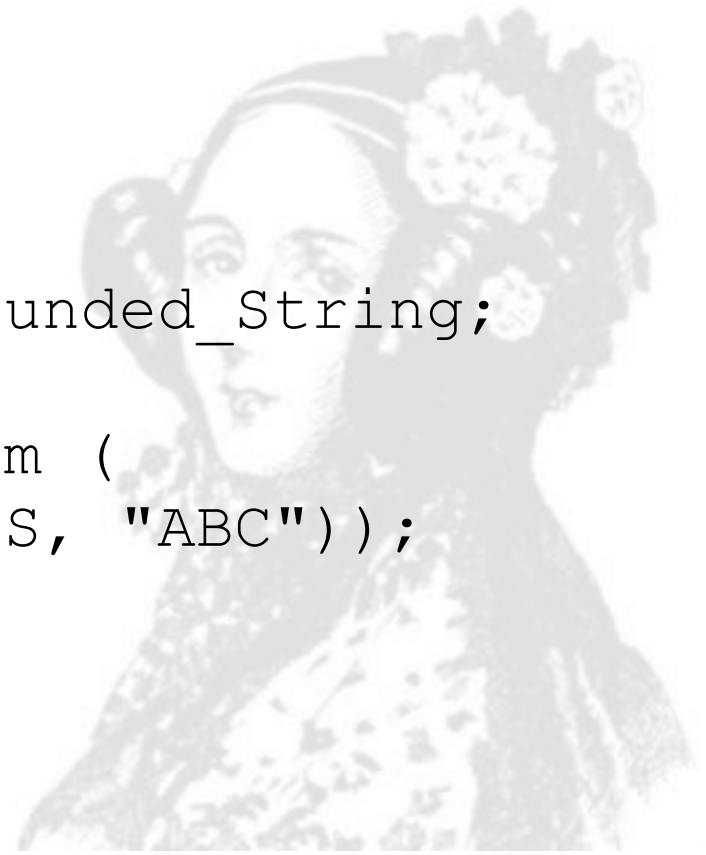
- C++で言えばADL
- C++で言えばoperator []
- C++で言えばrange-based-for
- C++で言えば[[]]
- Eiffel化！？
- 関数型言語化！？
- ...

C++で言えばADL

経緯

パッケージ名が長くてうざい。
たとえば↓を短くしたい。

```
with Ada.Strings.Unbounded;
procedure Test is
   S : Ada.Strings.Unbounded.Unbounded_String;
begin
   S := Ada.Strings.Unbounded.Trim (
      Ada.Strings.Unbounded."&" (S, "ABC"));
end Test;
```



C++で言えばADL

Ada83の解法 (1) ……using namespace

```
with Ada.Strings.Unbounded;
procedure Test is
    use Ada.Strings.Unbounded;
    S : Unbounded_String;
begin
    S := Trim (S & "ABC"); -- OK
end Test;
```

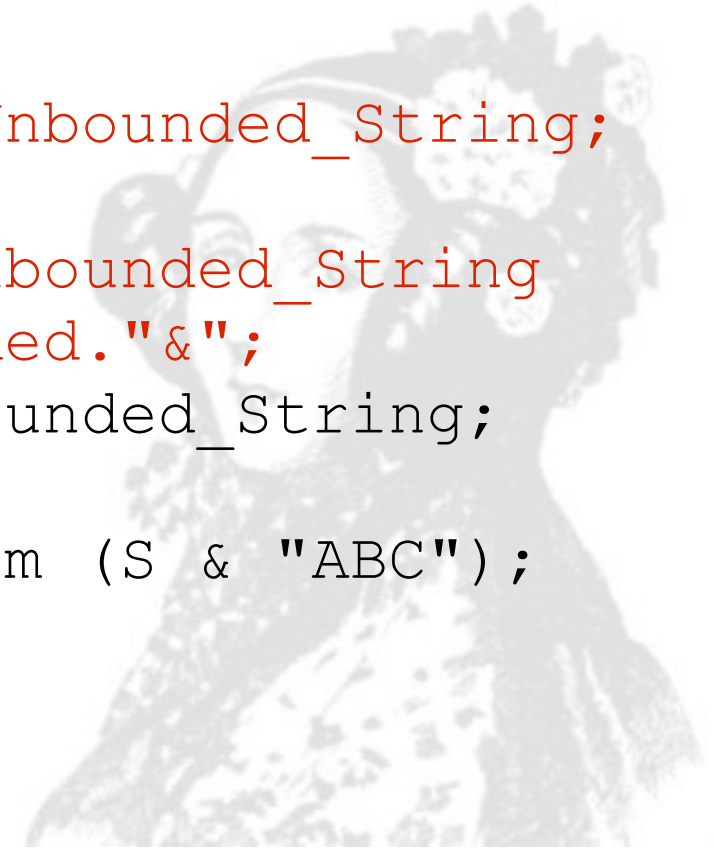
C++でusing namespaceの多用が好ましくないのと同じ理由で、通常useは避けられます。

C++で言えばADL

Ada83の解法 (2) ……個別にrenames

```
with Ada.Strings.Unbounded;
procedure Test is
   function "&" (
      L : Ada.Strings.Unbounded.Unbounded_String;
      R : String) return
         Ada.Strings.Unbounded.Unbounded_String
      renames Ada.Strings.Unbounded."&";
   S : Ada.Strings.Unbounded.Unbounded_String;
begin
   S := Ada.Strings.Unbounded.Trim (S & "ABC");
end Test;
```

めんどい。

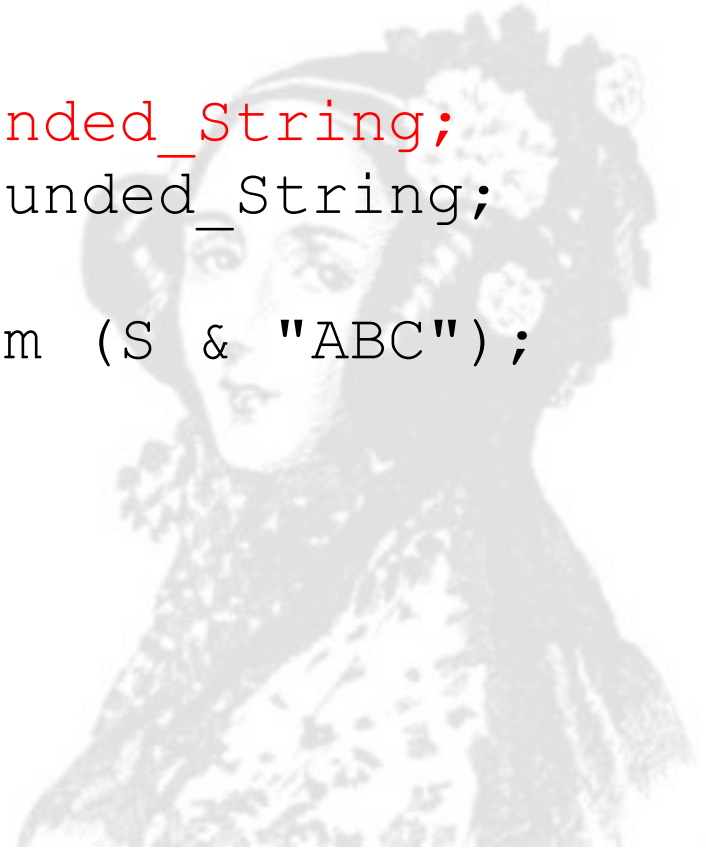


C++で言えばADL

Ada95の解法 ……use type

```
with Ada.Strings.Unbounded;
procedure Test is
    use type
        Ada.Strings.Unbounded.Unbounded_String;
    S : Ada.Strings.Unbounded.Unbounded_String;
begin
    S := Ada.Strings.Unbounded.Trim (S & "ABC");
end Test;
```

演算子のみ引っ張ってくる。



C++で言えばADL

Ada95の解法 ……use type
しかしgeneric関数を書こうとすると？

```
generic
   type T is private;
   function "<" (L, R : T) return Boolean is <>;
   function Trim (Item : T) return T is <>;
procedure My_Func (C : in out C_Type);

use type Ada.Strings.Unbounded.Unbounded_String;
procedure My_Instance is new My_Func (
   T => Ada.Strings.Unbounded.Unbounded_String,
   -- "<"は見えてるので省略可
   Trim => -- Trimは見えてないので省略できない。
    Ada.Strings.Unbounded.Unbounded_String.Trim);
```

C++で言えばADL

Ada2005では……進展なし

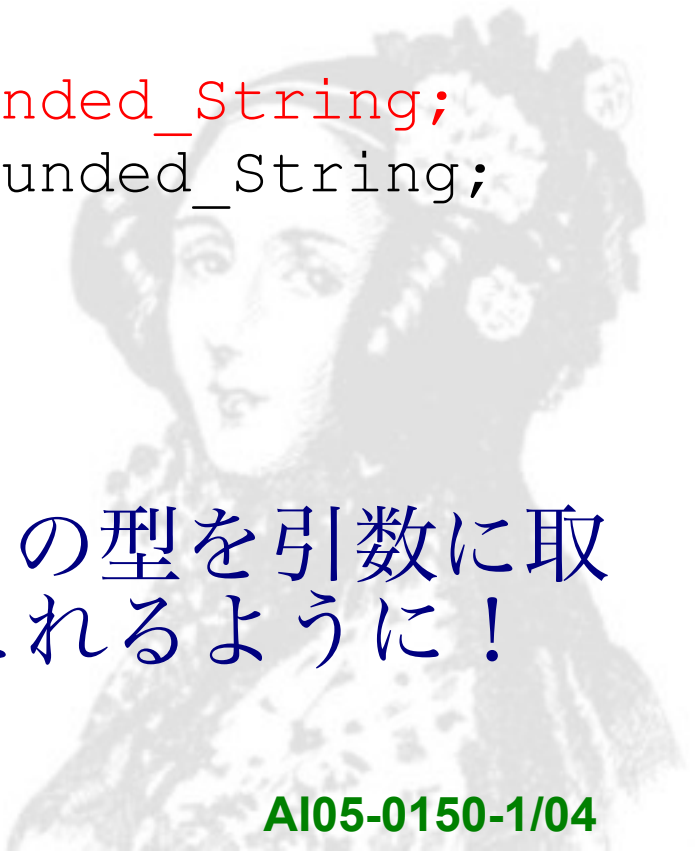
強いていえば、標準のコンテナライブラリの追加でgenericの引数めんどい問題が多発するようになった。

C++で言えばADL

Ada2012では……use all type

```
with Ada.Strings.Unbounded;
procedure Test is
   use all type
      Ada.Strings.Unbounded.Unbounded_String;
   S : Ada.Strings.Unbounded.Unbounded_String;
begin
   S := Trim (S & "ABC");
end Test;
```

同じパッケージで宣言された、その型を引数に取るor返す関数全部を引っ張ってこれるように！

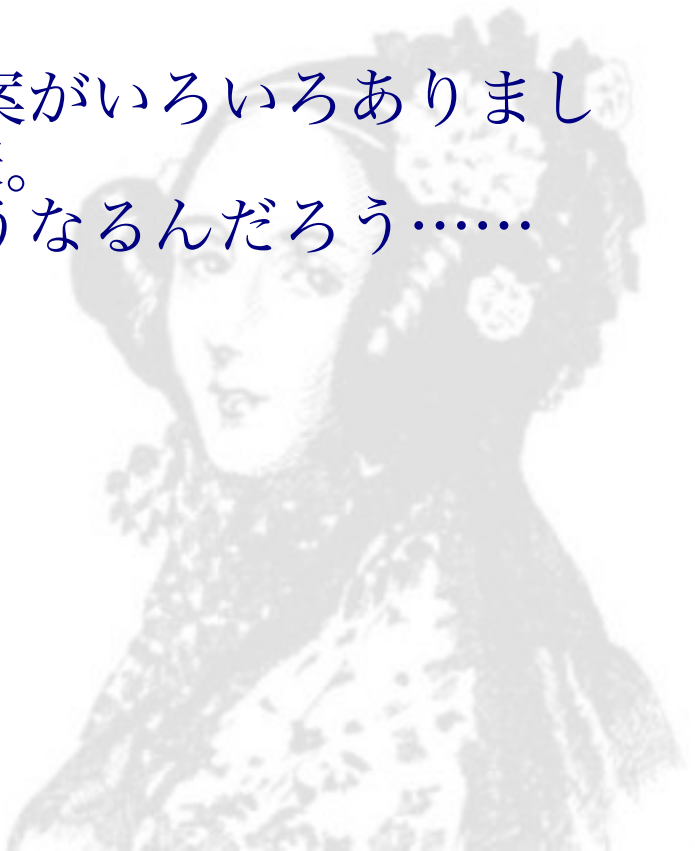


C++で言えばADL

Ada2012では ……use all type

ADLは邪悪じゃないよ！素晴らしいよ！

ちなみにuse all typeという構文については案がいろいろありましたが、結局、既存の予約語を再利用した模様。
(今後更に上位の機能が追加されるときはどうなるんだろう……
use all others of type！？)



Ada2012 Language Update

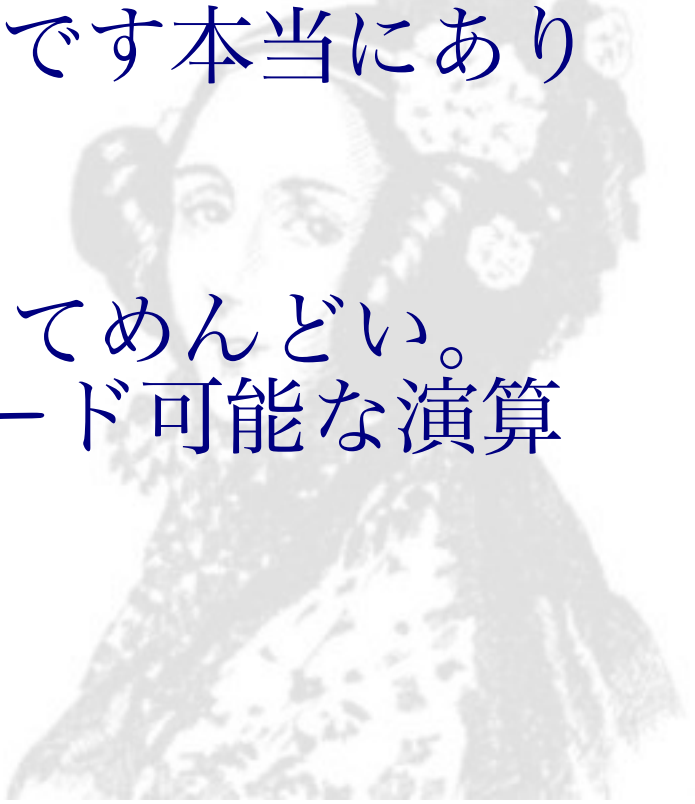
- C++で言えばADL
- C++で言えばoperator []
- C++で言えばrange-based-for
- C++で言えば[[]]
- Eiffel化！？
- 関数型言語化！？
- ...

C++で言えばoperator []

経緯

Ada2005でコンテナライブラリが追加された。
(イテレータの枠組みはあっても<algorithm>もない。どこからどう見ても劣化STLです本当にありがとうございました。)

これに必要な記述が長ったらしくてめんどい。
特にoperator []相当のオーバーロード可能な演算子が無いのが一番痛い。



C++で言えばoperator []

Ada2005では (1)

```
Item := Element (Container, Position);
Func (Item); -- Itemを加工
Replace_Element (Container, Position, Item);
```

値全部をコピーするため、要素型のサイズが大きかったり同一性が大事だったりすると好ましくない。

C++で言えばoperator []

Ada2005では (2)

```
declare
   procedure Process (Item : in out Integer) is
   begin
      Func (Item); -- Itemを加工
   end Process;
begin
   Update_Element (Container, Position,
      Process'Access);
end;
```

めんどくさい。

C++で言えばoperator []

C++では

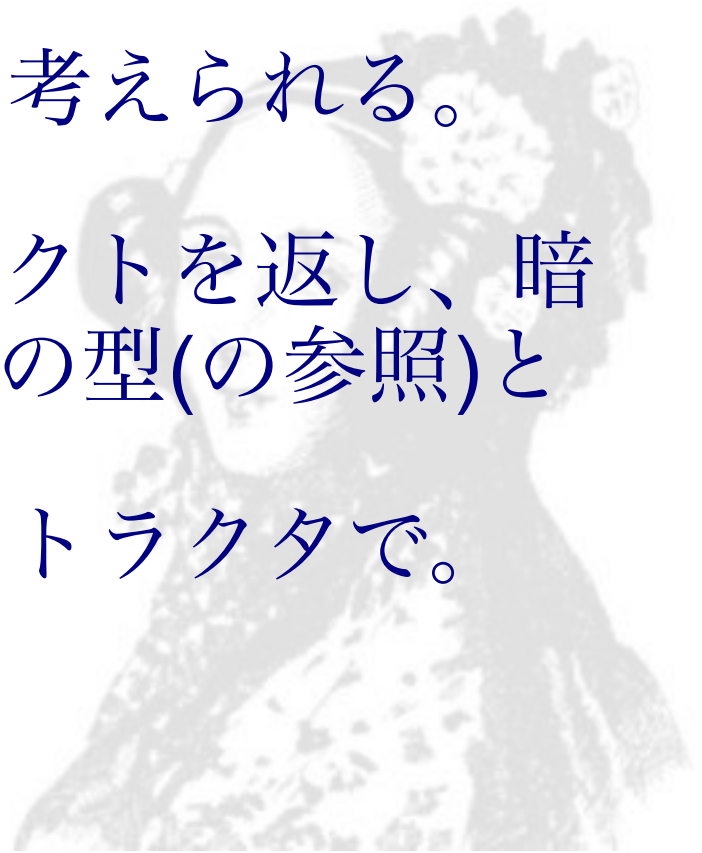
```
Func(Container[Position]);
```

C++で言えばoperator []

この差は何事！？

コンテナによってはアクセス後の後始末を要するかもしれない。(ロック解除等)
一時的な追加データがあることも考えられる。

C++はそういう時は一時オブジェクトを返し、暗黙のキャスト演算子によって目的の型(の参照)として使わせる。
後始末は一時オブジェクトのデストラクタで。



C++で言えばoperator []

この差は何事！？

Adaには

- operator []がない
- 参照がない
- 暗黙のキャスト演算子(operator 型)が無い
- ~~なによりも速さがry~~

結果、値を丸ごとコピーするか、クロージャでアクセス範囲を絞るかの二択だった。
構文も冗長に。

C++で言えばoperator []

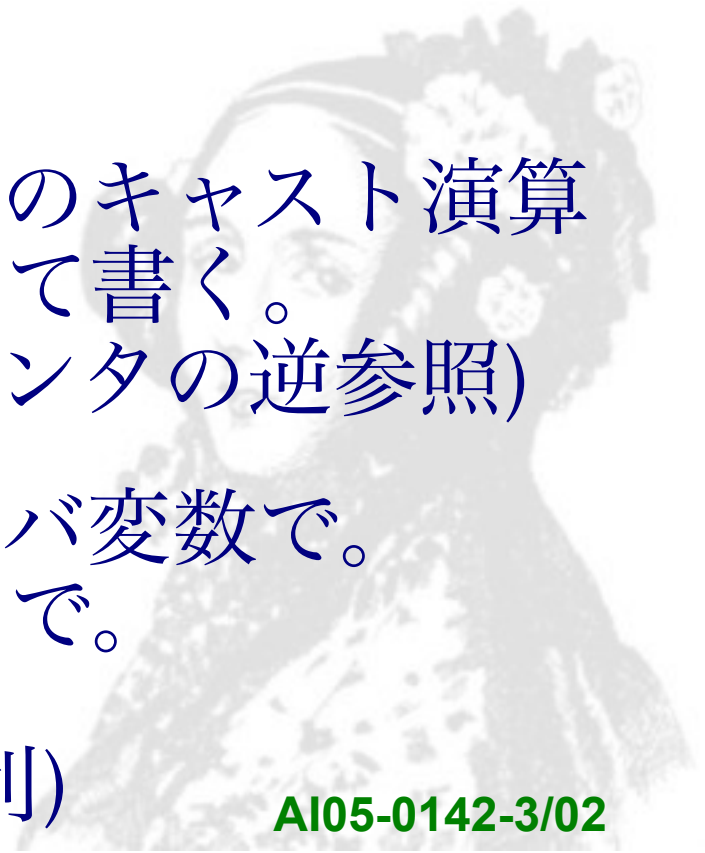
こういうテクニックを思いついた人がいた。

```
Func (Reference (
    Container,
    Position).Element.all);
```

要するにラッパーを返して、暗黙のキャスト演算子の代わりに明示的に.Elementって書く。
参照もないので.allも必要。(ポインタの逆参照)

追加データはラッパーの隠しメンバ変数で。
後始末はラッパーのデストラクタで。

(この案は関数の参照返し案に勝利)



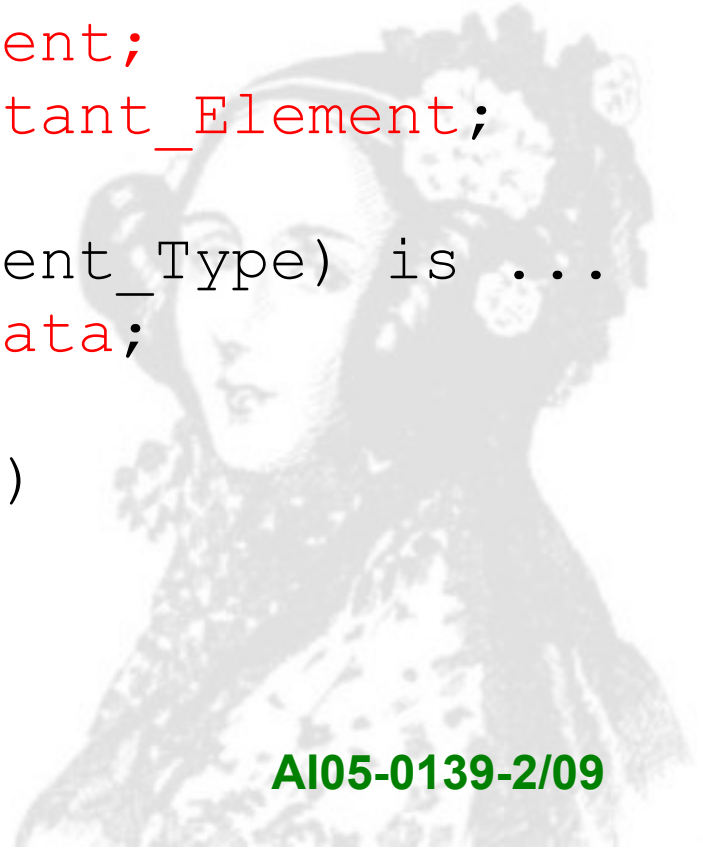
C++で言えばoperator []

ついでに暗黙のキャスト演算子が
入ってしまいそうな嫌な予感がする……。

```
type Container is tagged limited private
   with Variable_Indexing => Element;
        Constant_Indexing => Constant_Element;

type Ref_T (Element : access Element_Type) is ...
   with Implicit_Dereference => Data;

function Element (Obj : Container)
   return Reference;
```



C++で言えばoperator []

全部入ればAda2012ではこうなる。

```
Func (Container (Position)));
```

見た目配列と同じに見えてめでたしめでたし。

Ada2012 Language Update

- C++で言えばADL
- C++で言えばoperator []
- C++で言えばrange-based-for
- C++で言えば[[]]
- Eiffel化！？
- 関数型言語化！？
- ...

C++で言えばrange-based-for

コンテナの話の続きです。

コンテナをループ (インデックス版)

```
for I in
   First_Index (Container) ..
   Last_Index (Container)
loop
   ...
end loop;
```

めんどい。

C++で言えばrange-based-for

コンテナをループ (イテレータ版)

```
declare
   Position : Cursor := First (Container);
begin
   while Has_Element (Position) loop
      ...
      Next (Position);
   end loop;
end;
```

インデックスの使えないコンテナ(vector以外全部)は輪をかけてめんどい。

C++で言えばrange-based-for

流石にIterator must goと言いたくなる。

C++で言えばrange-based-for

Ada2012では

```
for I in Container loop
   ... -- Iはイテレータ
end loop;

for E of Container loop
   ... -- Eは要素の参照
end loop;
```

かんたん。

AI05-0139-2/09

C++で言えばrange-based-for

だが(C++0xと異なり)後付け不可能な実装方法になってしまった……

```
type Forward_Iterator is limited interface;
function First(Object : Forward_Iterator)
   return Cursor;
function Next (
   Object : Forward_Iterator;
   Position : Cursor) return Cursor;

-- Reversible_Iteratorもある
```

こういうinterfaceを実装しないといけない。

C++で言えばrange-based-for

とりあえず実に手っ取り早い後付け方法が想定されているらしい。(ラッパー挟むだけ)

```
for I in Iterate (Container) loop
   ...
end loop;
```

(↑は本の虫(江添様)の「range-based forに対する意見求む」の案4に相当する。)

http://cpplover.blogspot.com/2011/02/range-based-for.html

結論：ADLでもなんでも後付け機構が考えられているだけC++0xは素晴らしい

C++で言えばrange-based-for

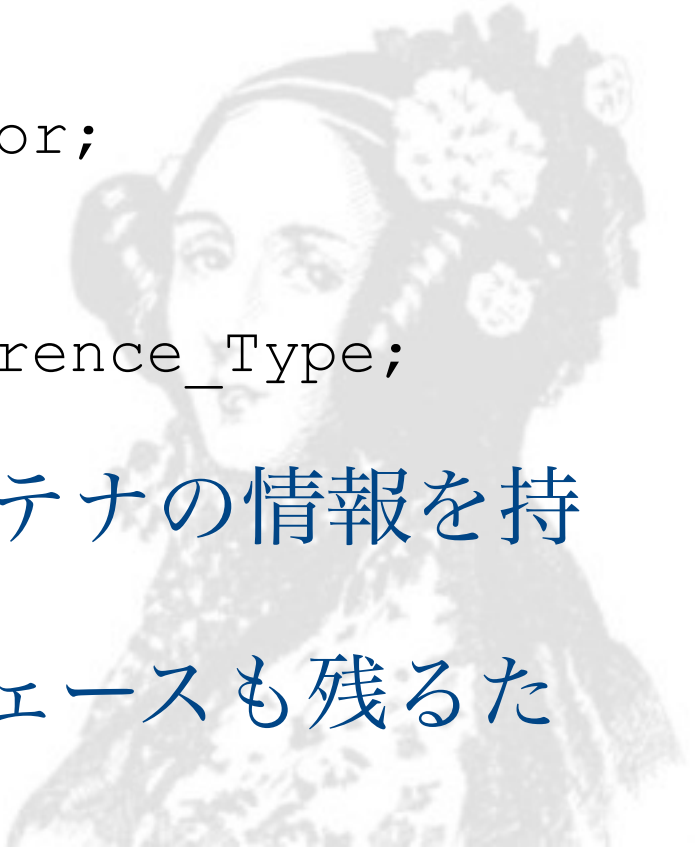
ちなみに、先のoperator[]と組み合わせてこう展開される。

```
declare
   Ite : Iterator := Iterate (Container);
   I : Cursor := First (Ite);
begin
   while I /= No_Element loop
      -- E renames
      --    Reference (Container, I).Element.all;
      I := Next (Ite, I);
   end loop;
end;
```

C++で言えばrange-based-for

この展開は、Next(++)、Reference([])の両方にコンテナの情報が渡るところが素晴らしい。

```
function Next (
   Object : Forward_Iterator;
   Position : Cursor) return Cursor;
function Reference (
   Object : Container_Type;
   Position : Cursor) return Reference_Type;
```



もうCursor(イテレータ)にコンテナの情報を持たせる必要が無くなった。
(実際にはAda2005のインターフェースも残るため最適化はできないけど)

C++で言えばrange-based-for

なんで従来イテレータがコンテナの情報を持っている必要があったかといいますと……

例)Ada2005ではVectorのCursor(イテレータ)はインデックスとVector本体へのポインタの組になっておりレジスタに乗りませんでした……。

Q.なんで(C++のように)実体の配列上を指すポインタじゃないの？
A.最後を示すNo_Elementが定数だから。

結論：最後の判定に ::end() を使うSTLは賢い

Ada2012 Language Update

- C++で言えばADL
- C++で言えばoperator []
- C++で言えばrange-based-for
- C++で言えば[[]]
- Eiffel化！？
- 関数型言語化！？
- ...

C++で言えば[[]]

inline等は今までpragmaで書いていた。

```
package P is
   pragma Pure (P);
   function F (X : Integer) return Integer;
   pragma Inline (F);
end P;
```

名前を2回書かないといけない。
めんどい。

C++で言えば[[]]

Ada2012では

```
package P with Pure is
   function F (X : Integer) return Integer
      with Inline;
end P;
```

C++で言えば[[]]

単に互換性を無くす「だけ」の文法のため、GNATの中の人がまだAda95を使い続けてる顧客が多いとごねてます。
Robert Dewar vs Tucker Taftをヲチしよう。(AI05-0228-1)

あと先の暗黙のキャストやfor I of Container loop...の裏で使われる関数もこの構文で指定するようになる模様。

```
type Container is ... with
  Variable_Indexing => ...,
  Constant_Indexing => ...,
  Default_Iterator  => ...,
  Iterator_Element  => ...;
```

AI05-0183-1/09

Ada2012 Language Update

- C++で言えばoperator []
- C++で言えばADL
- C++で言えばrange-based-for
- C++で言えば[[]]
- Eiffel化！？
- 関数型言語化！？
- ...

Eiffel化！？

事前条件、事後条件

```
function Sin (X : Float) return Float;
pragma Precondition (0.0 <= X
   and then X <= 2.0 * Pi);
pragma Postcondition (-1.0 <= Sin'Result
   and then Sin'Result <= +1.0);
```

或いは`function ... with Pre => ..., Post => ...;`

Ada2005のとき全く同じ提案があって却下されたはず……。(型チェックで解決するのがAda流)

AI05-0145-1/02

Eiffel化！？

ちなみにsinの例に限ればこれでOK

```
subtype Arg is Float range 0.0 .. 2.0 * Pi;
subtype Sine_Range is Float range -1.0 .. +1.0;
function Sin (X : Arg) return Sine_Range;
```

Ada2012 Language Update

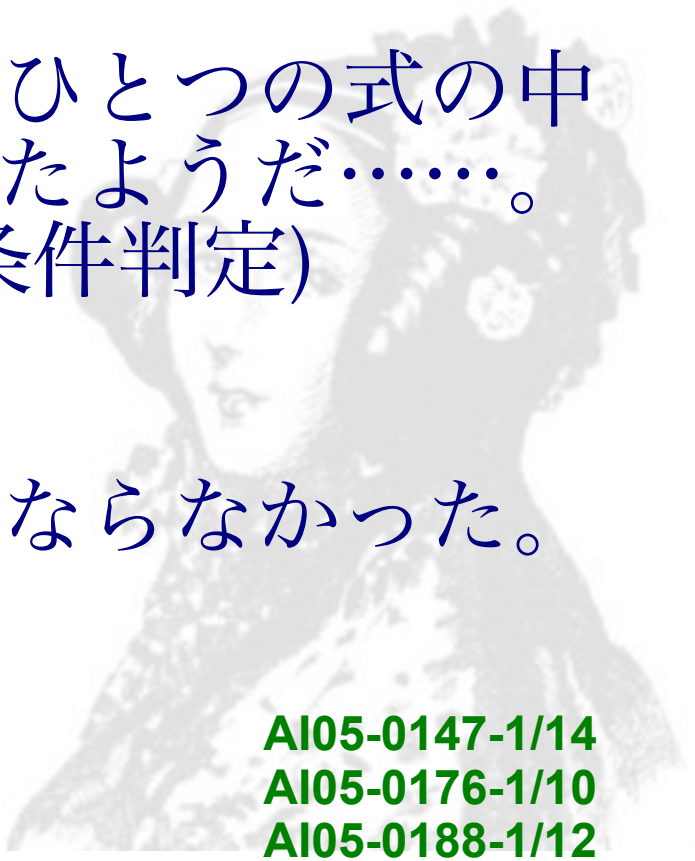
- C++で言えばoperator []
- C++で言えばADL
- C++で言えばrange-based-for
- C++で言えば[[]]
- Eiffel化！？
- 関数型言語化！？
- ...

関数型言語化！？

```
A := (if B then C else D);
A := (case B is when C => D, when others => E);
A := (for all B in C .. D => E (B) = F);
A := (for some B in C .. D => E (B) = F);
```

事前事後条件を書こうとすると、ひとつの式の中で分岐が欲しいってんで追加されたようだ……。
(後ろ二つは内包表記ではなくて条件判定)

someは新しい予約語。
コンテキスト依存キーワードにはならなかった。
(一安心)



AI05-0147-1/14
AI05-0176-1/10
AI05-0188-1/12

Ada2012 Language Update

- C++で言えばoperator []
- C++で言えばADL
- C++で言えばrange-based-for
- C++で言えば[[]]
- Eiffel化！？
- 関数型言語化！？
- …

Ada2012 Language Update

目次も2ページ目に入ります。
残るは細かいことだけですが……。

- 空文の必要が減った
- コンテナ多数追加
- 使えないライブラリ多数追加
- vectorを入れ子にできない問題解決なるか？

Ada2012 Language Update

- 空文の必要が減った
- コンテナ多数追加
- 使えないライブラリ多数追加
- vectorを入れ子にできない問題解決なるか？

空文の必要が減った

ラベルの後ろに文がいらなくなった。

```
while ... loop
   if ... then
     goto Continue; -- Adaにはcontinueが無いので
   end if;
   ...
   <<Continue>>
   -- 今まではここに空文が必要だった
end loop;
```

これはC言語に対する優位性として語り継がれるべき(嘘)

空文の必要が減った

pragmaを書いたら空文が不要になった。
(前提知識:ブロックの中には最低ひとつ文が必要)

```
begin
    if A then
        pragma Assert (B = C);
        -- 今まではここに空文が必要だった
    end if;
end;
```

Assertがpragmaになっているから、というだけで他にこの書き方ができて嬉しいpragmaは無さそう。

AI05-0163-1/03

Ada2012 Language Update

- 空文の必要が減った
- コンテナ多数追加
- 使えないライブラリ多数追加
- vectorを入れ子にできない問題解決なるか？

コンテナ多数追加

Ada2005まで: 同じコンテナが2種類あった。

- Ada.Containers.Vectors
- Ada.Containers.Indefinite_Vectors

Adaの型は実行時までサイズが確定しない型があるため、コンテナも少々めんどい実装が必要になる。

でも可変サイズ対応コンテナばかりだと普通の型で効率が悪いので、2種類に。

(templateの特殊化ができない言語の末期症状)

コンテナ多数追加

Ada2012: もう一種類追加。

- Ada.Containers.Vectors
- Ada.Containers.Indefinite_Vectors
- Ada.Containers.Bounded_Vectors

あらかじめ決めた要素数までしか持てない固定サイズのコンテナ。領域は静的に確保。

(そもそもVectorsがC++のようにアロケータを指定できる設計になっていれば要らんのでは……？)

コンテナ多数追加

各種キュー

Synchronized / Priority / Bounded or Unbounded / Indefniteの組み合わせでものすごい種類が……

アトミックにtask間通信を行う機能付き。

(もちろん普通にキューが欲しい時は
Double_Linked_Listsでも充分)

コンテナ多数追加

Indefnite_Holders

可変サイズ型の値を固定サイズ変数に入れるコンテナ。

(これが最初からあればIndefnite_Vectorsも要らなかったですよね……)

コンテナ多数追加

Multiway_Trees

木構造を表現するコンテナ。
(具体的な話は後述します)

AI05-0136-1/07

Ada2012 Language Update

- 空文の必要が減った
- コンテナ多数追加
- 使えないライブラリ多数追加
- vectorを入れ子にできない問題解決なるか？

使えないライブラリ多数追加

- UTF-8/16BE/16LEを変換する関数群
  (この設計がまたすごく酷い)
- パス文字列のディレクトリ階層を切り貼りする関数群
- ロケールを取得する関数
- CPU数を取得する関数
  CPUにtaskを割り振るpragmaも追加
- etc.....

AI05-0137-2/03
AI05-0049-1/04
AI05-0127-2/05
AI05-0167-1/07

Ada2012 Language Update

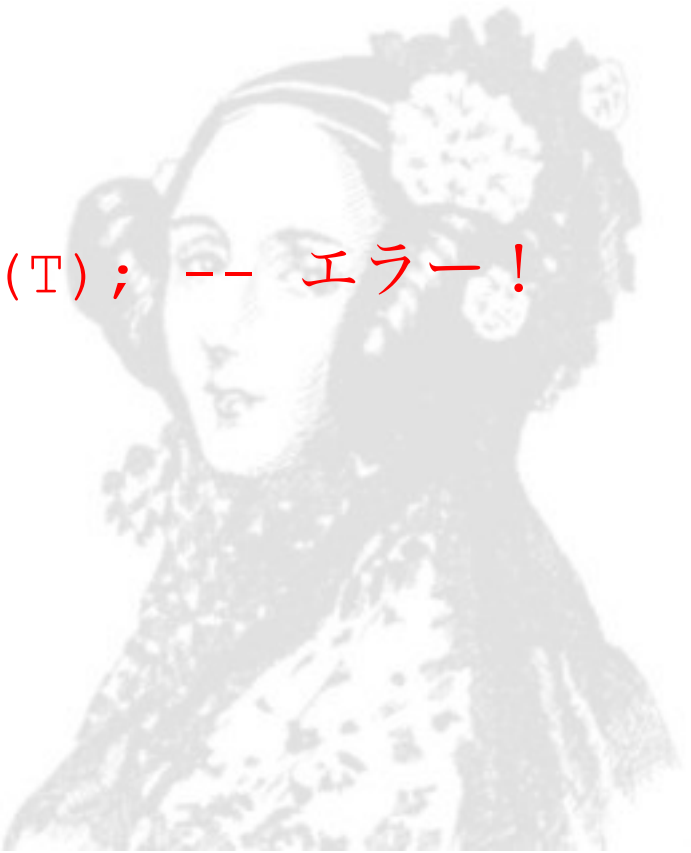
- 空文の必要が減った
- コンテナ多数追加
- 使えないライブラリ多数追加
- vectorを入れ子にできない問題解決なるか？

vectorを入れ子にできない問題

Adaでは↓ができない。

```
stuct T {
    vector<T> children;
};

type T;
package T_Vectors is new Vectors (T); -- エラー！
type T is record
    children : T_Vectors.Vector;
end record;
```



vectorを入れ子にできない問題

理由:

- record宣言とinstantiationの構文が別になっていて、record宣言中にinstantiationできない。
- incomplete type(Cで言うところのopaque type)はgenericをインスタンス化するときに使えない。(←これはAda2012で緩和されるかも？)
- そもそもvectorに必要なのはちゃんと実体のある型なので、opaque typeが使えたところでどうにもならない。

vectorを入れ子にできない問題

いろいろ案が出ているものの、まだ決着は付いていません。

with private案
limited new案
end private案
pragma May_Be_Partial案
private new案その1その2

AI05-0011-1
AI05-0074-*/**
AI05-0151-1/08
AI05-0162-1/04
AI05-0213-1/01

(説明は省きますが、基本どれもinstantiationを2段階に分けて行うと思ってくだされば)

とりあえず木構造だけ実現するMultiway-treeコンテナが追加されました。ちゃんと解決された暁にはゴミと化す予感……本来要らないはずのコンテナをどれだけ増やせば気が済むんだろう……

Ada2012 Language Update

おしまい

書いてないやつメモ

- aliased parameter ……引数のアドレス取れるよ
- expression functions ……一行関数が短く書けるよ
- generalized membership tests ……subtypeの範囲が連続じゃなくてもOK
- 同期用オブジェクトいろいろ追加
- X'Overlaps_Storage
- “Integrated” nested packages (未確定だけどすごく欲しい)