Ada and C++

Martyn Pike

university.adacore.com

Introduction

- **CPP Convention**
- **C++ Name Mangling**
- **Methods for Address tricky issues of C++ Name Mangling**
  - Using Link_Name with hardcoded linker symbol
  - extern “C”
  - Use g++ -fdump-ada-spec
- **Interfacing at the C++ class level**
  - Constructors and Multiple Inheritance of Abstract Classes
- **Exporting Ada tagged types as classes**
- **Handling C++ Exceptions**
- **Ada 2005 pragmas**

CPP Convention

- **GNAT supports C++ specific import conventions**
  - CPP
  - C_Plus_Plus

```
with Interfaces.C; with Interfaces.C.Strings;

procedure Main is

   function getRef return Interfaces.C.int;
   pragma Import(CPP, getRef, [..]);

   function getRefwithString(aString : Interfaces.C.Strings.chars_ptr) return Interfaces.C.int;
   pragma Import(C_Plus_Plus, getRefwithString, [..]);

begin
   null;
end Main;
```

C++ Name Mangling

```
const int x = 30;

const int getRef(void)
{
   return x;
}

const int getRef(char* aString)
{
   return x;
}
```

```
$ g++ -c cpplib.cpp -o cpplib.o
```

```
$ nm cpplib.o
```

```
00000000 b .bss
00000000 d .data
00000000 r .eh_frame
00000000 r .rdata
00000000 t .text
0000000c T __Z6getRefPc
00000000 T __Z6getRefv
00000000 r __ZL1x
```

Using Link_Name

- **Link_Name argument for pragma Import**
  - Needs hard coded C++ mangled name

```
function getRef return Interfaces.C.int;
pragma Import(CPP, getRef, Link_Name => "__Z6getRefPc");

function getRefwithString(aString : Interfaces.C.Strings.chars_ptr) return Interfaces.C.int;
pragma Import(CPP, getRefwithString, Link_Name => "__Z16getRefWithStringPc");
```

```
function getRef return Interfaces.C.int;
pragma Import(CPP, getRef, Link_Name => "__Z16getRefWithStringPc");

function getRefwithString(aString : Interfaces.C.Strings.chars_ptr) return Interfaces.C.int;
pragma Import(CPP, getRefwithString, Link_Name => "__Z6getRefPc");
```

- **Increases maintenance costs**
- **Reduces compiler independence**

extern “C”

```
const int x = 30;

const int getRef(char* aString)
{
   return x;
}

extern "C" {
   const int getRefWithString(char* aString)
   {
      return x;
   }
}
```

```
$ g++ -c cpplib.cpp -o cpplib.o
$ nm cpplib.o

00000000 b .bss
00000000 d .data
00000000 r .eh_frame
00000000 r .rdata
00000000 t .text
00000000 T __Z6getRefPc
00000000 r __ZL1x
0000000c T _getRefWithString
```

```
with Interfaces.C; with Interfaces.C.Strings;

procedure Main is

   function getRef return Interfaces.C.int;
   pragma Import(CPP, getRef, Link_Name => "__Z6getRefPc");

   function getRefwithString(aString : Interfaces.C.Strings.chars_ptr) return Interfaces.C.int;
   pragma Import(CPP, getRefwithString, External_Name => "getRefWithString");

begin
   null;
end Main;
```

Using –fdump-ada-spec

```
const int x = 30;

const int getRef(void)
{
   return x;
}

const int getRef(char* aString)
{
   return x;
}
```

```
$ g++ -c -fdump-ada-spec cpplib.cpp
```

```
[..]

package cpplib_cpp is

   x : aliased int;
   pragma Import (CPP, x, "_ZL1x");

   function getRef (aString : Interfaces.C.Strings.chars_ptr) return int;
   pragma Import (CPP, getRef, "_Z6getRefPc");

   function getRefWithString (aString : Interfaces.C.Strings.chars_ptr) return int;
   pragma Import (CPP, getRefWithString, "_Z16getRefWithStringPc");

end cpplib_cpp;
```

Ada Limited Types

- Features of Ada limited types
- Limited types can be used to represent C++ classes

```
package Class_AClass is

   type AClass is limited record
      lastCharacter : aliased char;
      firstCharacter : aliased char;
   end record;
   pragma Import (CPP, AClass);

end Class_AClass;
```

- Assignments between objects is prohibited
- No predefined equality operator for limited typed objects

Note on forthcoming simplifications

- **On the following slides, except when useful:**
  - with / use clauses will be omitted
  - pragma import / export will be omitted
  - Only bound subprograms of interest will be displayed
  - Nested packages generated by the binding generator will be omitted
- **Full examples will be provided along with the course**

Importing C++ Classes

```
//aclass.cpp
class AClass
{
public:
   AClass(const char *name);
   ~AClass();
private:
   char lastCharacter;
protected:
   char firstCharacter;
};
```

```
$ g++ -c -fdump-ada-spec aclass.cpp
```

```
type AClass is limited record
   lastCharacter : aliased char;
   firstCharacter : aliased char;
end record;
pragma Import (CPP, AClass);

function New_AClass (name : Interfaces.C.Strings.chars_ptr) return AClass;
pragma CPP_Constructor (New_AClass, "_ZN6AClassC1EPKc");

procedure Delete_AClass (this : access AClass);
pragma Import (CPP, Delete_AClass, "_ZN6AClassD1Ev");
```

C++ Constructors

```
type AClass is limited record
  lastCharacter : aliased char;
end record;
pragma Import (CPP, AClass);

function New_AClass return AClass;
pragma CPP_Constructor (New_AClass, "_ZN6AClassC1Ev");
```

- **GNAT –gnatG option to produce intermediate output**

```
[…]
procedure main is
   use aclass_cpp.aclass_cpp__class_aclass;
   x : aliased aclass_cpp__class_aclass__aclass;
   _ZN6AClassC1Ev (x);
Begin
[…]
```

Using C++ Classes

```
package Class_AClass is
   type AClass is limited record
      firstCharacter : aliased char;
   end record;
   pragma Import (CPP, AClass);

   -- Assigns firstCharacter the value ‘B’
   function New_AClass return AClass;
   pragma CPP_Constructor (New_AClass, "_ZN6AClassC1Ev");

   -- Assigns firstCharacter the first character of the name
   function New_AClass (name : Interfaces.C.Strings.chars_ptr) return AClass;
   pragma CPP_Constructor (New_AClass, "_ZN6AClassC1EPKc");

   function getFirstChar (this : access AClass) return char;
   pragma Import (CPP, getFirstChar, "_ZN6AClass12getFirstCharEv");
end Class_AClass;
```

```
with aclass_h;
with Ada.Text_IO; use Ada.Text_IO;
with Interfaces.C.Strings; use Interfaces.C.Strings;

procedure Main is
   use aclass_h.Class_AClass;
   X : access AClass := new AClass'(New_AClass(New_String("B")));
begin
   Put_Line(getFirstChar(X)'Img);
end Main;
```

Extending C++ Classes

```
class Base {
public:
  Base () {};

  virtual void P1 ()
  {
     cout << "P1 FROM C++" << endl;
  };

  int F;
};
```

```
type Base is tagged limited record
   F : aliased int;
end record;

function New_Base return Base;
procedure P1 (this : access Base);
```

```
package Extensions is

   type Child is limited new Base with record
      F2 : Integer;
   end record;

   function New_Base return Child;

   overriding
   procedure P1 (This : access Child);

end Extensions;
```

```
package body Extensions is

   function New_Base return Child is
   begin
      return (Base'(New_Base) with F2 => 0);
   end New_Base;

   overriding
   procedure P1 (This : access Child) is
   begin
      Put_Line ("P1 FROM Ada");
   end P1;

end Extensions;
```

Cross-Language Dispatching

```
void CallFromCpp (Base * obj) {
   obj->P1();
}
```

```
procedure CallFromCpp (Obj : access Base'Class);
```

```
procedure Main is
   procedure CallFromAda (O : in out Base'Class) is
   begin
      O.P1;
   end CallFromAda;

   type Acc is access all Base'Class;

   O1 : Acc := new Base'(Base'(New_Base));
   O2 : Acc := new Child'(Child'(New_Base));
begin
   CallFromAda(O1.all);
   CallFromAda(O2.all);

   CallFromCpp(O1);
   CallFromCpp(O2);
end Main;
```

```
P1 FROM C++
P1 FROM ADA
P1 FROM C++
P1 FROM ADA
```

C++ Abstract Classes as Abstract Types

```
class Base {
public:
      virtual void P1 () = 0;
      virtual int P2 () {return 0;}
      int F;
};
```

```
type Base is abstract tagged limited record
   F : aliased int;
end record;

procedure P1 (this : access Base) is abstract;
function P2 (this : access base) return int;
```

```
class Concrete : public Base {
   public:
      virtual void P1 () {}
};
```

```
type Concrete is limited new Base with record
   null;
end record;

procedure P1 (this : access Concrete);
function P2 (this : access Concrete) return int;
```

C++ Abstract Classes as Interfaces

```
class I1{
   public:
      virtual void P1 () = 0;
      virtual int P2 () = 0;
};
```

```
type I1 is limited interface;
procedure P1 (this : access I1) is abstract;
function P2 (this : access I1) return int is abstract;
```

```
class I2{
   public:
      virtual void P3 () = 0;
};
```

```
type I2 is limited interface;
procedure P3 (this : access I2) is abstract;
```

```
class Concrete : public I1, I2{
   public:
      virtual void P1 () {}
      virtual int P2 () {return 0;}
      virtual void P3 () {}
};
```

```
type Concrete is limited new I1 and I2 with record
   null;
end record;

procedure P1 (this : access Concrete);
function P2 (this : access Concrete) return int;
procedure P3 (this : access Concrete);
```

Exporting Ada Tagged Types

```
with Interfaces.C;

package ALib is

   type Animal is tagged record
      The_Age : Interfaces.C.int;
   end record;
   pragma Convention (CPP, Animal);

   function New_Animal return Animal'Class;
   pragma Export(CPP, New_Animal);

   function Age(X : Animal) return Interfaces.C.int;
   pragma Export(CPP, Age);

end ALib;
```

```
package body ALib is

   function New_Animal
      return Animal'Class is
   begin
      return Animal'(The_Age => 20);
   end New_Animal;

   function Age(X : Animal)
      return Interfaces.C.int is
   begin
      return X.The_Age;
   end Age;

end ALib;
```

```
// animal.h
class Animal {
public:
   virtual int age();
};
```

```
#include <iostream>
#include "animal.h"

extern "C" {
  void adainit (void);
  void adafinal (void);
  Animal* new_animal();
}

int main(void) {
   adainit();
   std::cout << new_animal()->age() << std::endl;
   adafinal();
   return 0;
};
```

Extending Exported Ada Tagged Types

```ada
with Interfaces.C;

package ALib is

   type Animal is tagged record
      The_Age : Interfaces.C.int;
   end record;
   pragma Convention (CPP, Animal);

   function New_Animal return Animal'Class;
   pragma Export(CPP, New_Animal);

   function Age(X : Animal) return Interfaces.C.int;
   pragma Export(CPP, Age);

end ALib;
```

```ada
package body ALib is

   function New_Animal
      return Animal'Class is
   begin
      return Animal'(The_Age => 20);
   end New_Animal;

   function Age(X : Animal)
      return Interfaces.C.int is
   begin
      return X.The_Age;
   end Age;

end ALib;
```

```cpp
// animal.h
class Animal {
public:
   virtual int age();
};
```

```cpp
// dog.h
class Dog {
public:
   Dog();
   void writeAge(void);
protected:
   Animal* m_animal;
};
```

```cpp
// dog.cpp
#include <iostream>
#include "dog.h"

extern "C" {
  Animal* new_animal();
}

Dog::Dog() :
   m_animal(new_animal()) {}

void Dog::writeAge(void) {
   std::cout <<
   this->m_animal->age() <<
   std::endl;
}
```

```cpp
// main.cpp
#include "dog.h"

extern "C" {
  void adainit (void);
  void adafinal (void);
}

int main(void) {
   adainit();
   Dog* theDog = new Dog();
   theDog->writeAge();
   adafinal();
   return 0;
};
```

C++ Exceptions

```
bool isOK(void) throw(int) {
   throw 20;
};
```

```
with Interfaces.C.Extensions;
with Ada.Text_IO;

procedure Main is
   function IsOK return Interfaces.C.Extensions.Bool;
   pragma Import (CPP, isOK, "_Z4isOKv");
   Res : Interfaces.C.Extensions.bool;
begin
   Res := isOK;
exception
   when others =>
      Ada.Text_IO.Put_Line("C++ Exception raised");
end Main;
```

Quiz

Is this correct? (1/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
with Interfaces.C;

procedure Main is

   function getRef return Interfaces.C.int;
   pragma Import(C++, getRef, "_Z6getRefv");

   X : Interfaces.C.int := getRef;

begin
   null;
end Main;
```

Is this correct? (1/10)

NO

C++ is not a valid convention

```
with Interfaces.C;

procedure Main is

   function getRef return Interfaces.C.int;
   pragma Import(C++, getRef, "_Z6getRefv");

   X : Interfaces.C.int := getRef;

begin
   null;
end Main;
```

```
function getRef return Interfaces.C.int;
pragma Import(CPP, getRef, "_Z6getRefv");

function getRef return Interfaces.C.int;
pragma Import(C_Plus_Plus, getRef, "_Z6getRefv");
```

Is this correct? (2/10)

✓ **YES**
(click on the check icon)

NO
(click on the error location(s))

```
// cpplib.cpp
int myfunc(void)
{
   return 20;
}
```

```
with Interfaces.C;

procedure Main is

   function MyFunc return Interfaces.C.int;
   pragma Import(CPP, MyFunc, "myfunc");

   X : Interfaces.C.int := MyFunc;

begin
   null;
end Main;
```

Is this correct? (2/10)

NO

```
// cpplib.cpp
int myfunc(void)
{
   return 20;
}
```

Not a mangled C++ name

```
with Interfaces.C;

procedure Main is

   function MyFunc return Interfaces.C.int;
   pragma Import(CPP, MyFunc, "myfunc");

   X : Interfaces.C.int := MyFunc;

begin
   null;
end Main;
```

```
$ nm cpplib.o

00000000 b .bss
00000000 d .data
00000000 r .eh_frame
00000000 t .text
00000000 T __Z6myfuncv
```

A mangled C++ name

Is this correct? (3/10)

✓

YES
(click on the check icon)

NO
(click on the error location(s))

```
// cpplib.cpp
extern “C” {
  int myfunc(void)
  {
    return 20;
  }
}
```

```
with Interfaces.C;

procedure Main is

   function MyFunc return Interfaces.C.int;
   pragma Import(CPP, MyFunc, "myfunc");

   X : Interfaces.C.int := MyFunc;

begin
   null;
end Main;
```

Is this correct? (3/10)

YES

```
// cpplib.cpp
extern “C” {
  int myfunc(void)
  {
    return 20;
  }
}
```

```
with Interfaces.C;

procedure Main is

   function MyFunc return Interfaces.C.int;
   pragma Import(CPP, MyFunc, "myfunc");

   X : Interfaces.C.int := MyFunc;

begin
   null;
end Main;
```

```
$ nm cpplib.o

00000000 b .bss
00000000 d .data
00000000 r .eh_frame
00000000 t .text
00000000 T _myfunc
```

Is this correct? (4/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
class Base {
public:
      virtual void P1 () = 0;
      virtual int P2 () {return 0;}
      int F;
};
```

```
type C1 is tagged limited record
   F : aliased int;
end record;

procedure P1 (this : access C1) is abstract;
function P2 (this : access C1) return int;
```

Is this correct? (4/10)

NO

Type must be Abstract

```
type C1 is tagged limited record
   F : aliased int;
end record;

procedure P1 (this : access C1) is abstract;
function P2 (this : access C1) return int;
```

```
type C1 is abstract tagged limited record
   F : aliased int;
end record;

procedure P1 (this : access C1) is abstract;
function P2 (this : access C1) return int;
```

Is this correct? (5/10)

✓ **YES**
(click on the check icon)

NO
(click on the error location(s))

```
// aclass.h
class AClass {
public:
  AClass();
  int m_attribute;
};
```

```
// aclass.cpp
#include "aclass.h"

AClass::AClass() : m_attribute(10) {};
```

```
with Interfaces.C;

procedure Main is

   type AClass is limited record
      m_attribute : Interfaces.C.int;
   end record;
   pragma Import(CPP,AClass);

   function AClass_Constructor return AClass;
   pragma Import(CPP, AClass_Constructor, "_ZN6AClassC1Ev");

   X : AClass;

begin
   null;
end Main;
```

? Is this correct? (5/10)

NO

```
// aclass.h
class AClass {
public:
  AClass();
  int m_attribute;
};
```

```
// aclass.cpp
#include "aclass.h"

AClass::AClass() : m_attribute(10) {};
```

No CPP_Constructor defined

```
with Interfaces.C;

procedure Main is

   type AClass is limited record
      m_attribute : Interfaces.C.int;
   end record;
   pragma Import(CPP,AClass);

   function AClass_Constructor return AClass;
   pragma Import(CPP, AClass_Constructor, "_ZN6AClassC1Ev");

   X : AClass;

begin
   null;
end Main;
```

Is this correct? (6/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
class Base{
public:
   virtual void P1 () = 0;
   virtual int P2 () = 0;
};
```

```
type Base is limited interface;
procedure P1 (this : access Base) is abstract;
function P2 (this : access Base) return int is abstract;
```

Is this correct? (6/10)

✓ YES

```
class Base{
public:
   virtual void P1 () = 0;
   virtual int P2 () = 0;
};
```

```
type Base is limited interface;
procedure P1 (this : access Base) is abstract;
function P2 (this : access Base) return int is abstract;
```

? Is this correct? (7/10)

✓ **YES**
(click on the check icon)

NO
(click on the error location(s))

```
// aclass.h
class AClass {
public:
  AClass();
  int m_attribute;
};
```

```
// aclass.cpp
#include "aclass.h"

AClass::AClass() : m_attribute(10) {};
```

```
with Interfaces.C;

procedure Main is

   type AClass is limited record
      m_attribute : Interfaces.C.int;
   end record;
   pragma Import(CPP,AClass);

   function AClass_Constructor return AClass;
   pragma CPP_Constructor(AClass_Constructor, "_ZN6AClassC1Ev");

   X : AClass;
   Y : AClass := X;

begin
   null;
end Main;
```

Is this correct? (7/10)

NO

```
// aclass.h
class AClass {
public:
  AClass();
  int m_attribute;
};
```

```
// aclass.cpp
#include "aclass.h"

AClass::AClass() : m_attribute(10) {};
```

Unable to assign limited types

```
with Interfaces.C;

procedure Main is

   type AClass is limited record
      m_attribute : Interfaces.C.int;
   end record;
   pragma Import(CPP,AClass);

   function AClass_Constructor return AClass;
   pragma CPP_Constructor(AClass_Constructor, "_ZN6AClassC1Ev");

   X : AClass;
   Y : AClass := X;

begin
   null;
end Main;
```

Is this correct? (8/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```cpp
// i_dds.h
class I_DDS {
public:
   virtual
     void printMe(void) = 0;
};
```

```ada
with Ada.Text_IO;

procedure Main is

   package Class_I_DDS is
      type I_DDS is limited interface;
      pragma Import (CPP, I_DDS);
      procedure printMe (this : access I_DDS) is abstract;
   end Class_I_DDS;

   package SubClass_I_DDS is
      type Sub_I_DDS is new Class_I_DDS.I_DDS with record
         An_Attribute : Integer := 20;
      end record;

      overriding
      procedure printMe (this : access Sub_I_DDS);
   end SubClass_I_DDS;

   package body SubClass_I_DDS is
      procedure printMe (this : access Sub_I_DDS) is
      begin
         Ada.Text_IO.Put_Line(this.An_Attribute'Img);
      end printMe;
   end SubClass_I_DDS;

begin
   SubClass_I_DDS.printMe(new SubClass_I_DDS.Sub_I_DDS);
end Main;
```

Is this correct? (8/10)

✓ YES

```
// i_dds.h
class I_DDS {
public:
   virtual
     void printMe(void) = 0;
};
```

```
with Ada.Text_IO;

procedure Main is

   package Class_I_DDS is
      type I_DDS is limited interface;
      pragma Import (CPP, I_DDS);
      procedure printMe (this : access I_DDS) is abstract;
   end Class_I_DDS;

   package SubClass_I_DDS is
      type Sub_I_DDS is new Class_I_DDS.I_DDS with record
         An_Attribute : Integer := 20;
      end record;

      overriding
      procedure printMe (this : access Sub_I_DDS);
   end SubClass_I_DDS;

   package body SubClass_I_DDS is
      procedure printMe (this : access Sub_I_DDS) is
      begin
         Ada.Text_IO.Put_Line(this.An_Attribute'Img);
      end printMe;
   end SubClass_I_DDS;

begin
   SubClass_I_DDS.printMe(new SubClass_I_DDS.Sub_I_DDS);
end Main;
```

? Is this correct? (9/10)

YES
(click on the check icon)

✓

NO
(click on the error location(s))

```
with Interfaces.C;

package ALib is

   type Animal is tagged record
      The_Age : Interfaces.C.int;
   end record;
   pragma Convention (CPP, Animal);

   function New_Animal return Animal'Class;
   pragma Export(CPP, New_Animal);

   function Age(X : Animal) return Interfaces.C.int;
   pragma Export(CPP, Age);

end ALib;
```

```
package body ALib is

   function New_Animal
      return Animal'Class is
   begin
      return Animal'(The_Age => 20);
   end New_Animal;

   function Age(X : Animal)
      return Interfaces.C.int is
   begin
      return X.The_Age;
   end Age;

end ALib;
```

```
// animal.h
class Animal {
public:
   virtual int age();
};
```

```
#include <iostream>
#include "animal.h"

extern "C" {
  void adainit (void);
  void adafinal (void);
  Animal* new_animal();
}

int main(void) {
   adainit();
   std::cout << new_animal()->age() << std::endl;
   adafinal();
   return 0;
};
```

Is this correct? (9/10)

✓ YES

```ada
with Interfaces.C;

package ALib is

   type Animal is tagged record
      The_Age : Interfaces.C.int;
   end record;
   pragma Convention (CPP, Animal);

   function New_Animal return Animal'Class;
   pragma Export(CPP, New_Animal);

   function Age(X : Animal) return Interfaces.C.int;
   pragma Export(CPP, Age);

end ALib;
```

```ada
package body ALib is

   function New_Animal
      return Animal'Class is
   begin
      return Animal'(The_Age => 20);
   end New_Animal;

   function Age(X : Animal)
      return Interfaces.C.int is
   begin
      return X.The_Age;
   end Age;

end ALib;
```

```cpp
// animal.h
class Animal {
public:
   virtual int age();
};
```

```cpp
#include <iostream>
#include "animal.h"

extern "C" {
  void adainit (void);
  void adafinal (void);
  Animal* new_animal();
}

int main(void) {
   adainit();
   std::cout << new_animal()->age() << std::endl;
   adafinal();
   return 0;
};
```

(10/10)

```cpp
// raiseException.cpp
void raiseException(void)
   throw (int)
{
   throw (int)20;
}
```

```ada
with Ada.Text_IO;

procedure Main is

   cpp_exception : Exception;

   procedure RaiseException;
   pragma Import(CPP, RaiseException, "_Z14raiseExceptionv");

begin
   RaiseException;
exception
   when cpp_exception =>
      Ada.Text_IO.Put_Line("C++ Exception");
end Main;
```

(10/10)

```
// raiseException.cpp
void raiseException(void) throw (int)
{
   throw (int)20;
}
```

```
with Ada.Text_IO;

procedure Main is

   cpp_exception : Exception;

   procedure RaiseException;
   pragma Import(CPP, RaiseException, "_Z14raiseExceptionv");

begin
   RaiseException;
exception
   when cpp_exception =>
      Ada.Text_IO.Put_Line("C++ Exception!!");
end Main;
```

The Answer is the program crashes.
A default Exception handler was required.

university.adacore.com