

**Seminar demonstration files**

---

**Overlays (II)**

**Denis Girou**

**June 2002**

With Acroread, **CTRL-L** switch  
between full screen and window mode

1 – Introduction . . . . .	3
2 – Equations with (cumulative) annotations . . . . .	4
3 – Listing with (progressive) annotations . . . . .	5
4 – (Cumulative) listing with (cumulative) annotations (I) . . . . .	9
5 – Listing with (cumulative) annotations (II) . . . . .	10

## 1 – Introduction

- ☞ If the talk is related to computing science, we must often show the contents of some programs. The ‘listings’ package is very useful and powerful for such tasks, used alone or both with the ‘fancyvrb’ one as we do here.
- ☞ This is also useful to be able to use **overlays** to emphasize some lines of the codes. This is possible both with ‘fancyvrb’ and ‘listings’<sup>a</sup>, using their escape mechanisms to execute some commands put inside verbatim material.
- ☞ We also add a macro to be able to put annotations on a text previously shown. This is specially useful to comment interactively things like equations and codes, putting them also in overlays. We give examples for this two cases.

---

<sup>a</sup>Thanks to Carsten HEINZ to have added in his package a special mechanism to interact with the overlay feature of Seminar.

## 2 – Equations with (cumulative) annotations

A formula for  $\Pi$  from Leonhard Euler

$\Pi$

End of slide

## 2 – Equations with (cumulative) annotations

A formula for  $\Pi$  from Leonhard Euler

$$\Pi = \sqrt{6}$$

= 2.44949


End of slide

## 2 – Equations with (cumulative) annotations

A formula for  $\Pi$  from Leonhard Euler

$$\Pi = \sqrt{6} \times \sqrt{\quad}$$

$= 2.44949$



## 2 – Equations with (cumulative) annotations

A formula for  $\Pi$  from Leonhard Euler

$$\Pi = \sqrt{6} \times \sqrt{1}$$

The diagram illustrates the calculation of Pi using the formula  $\Pi = \sqrt{6} \times \sqrt{1}$ . Two arrows originate from the formula: one points from the  $\sqrt{6}$  term to a box containing the value  $= 2.44949$ , and another points from the  $\sqrt{1}$  term to a box containing  $1 \implies 2.44949$ .

End of slide

## 2 – Equations with (cumulative) annotations

A formula for  $\Pi$  from Leonhard Euler

$$\Pi = \sqrt{6} \times \sqrt{1 + \frac{1}{4}}$$

The diagram illustrates the calculation of  $\Pi$  using the formula  $\Pi = \sqrt{6} \times \sqrt{1 + \frac{1}{4}}$ . Annotations show the cumulative value of the expression as terms are added:

- The value  $1$  (from the term  $1 + \frac{1}{4}$ ) is associated with the value  $2.44949$ .
- The value  $1.25$  (from the term  $1 + \frac{1}{4}$ ) is associated with the value  $2.73861$ .
- The final result of the entire expression is  $2.44949$ .

End of slide



## 2 – Equations with (cumulative) annotations

A formula for  $\Pi$  from Leonhard Euler

$$\Pi = \sqrt{6} \times \sqrt{1 + \frac{1}{4} + \frac{1}{9}}$$

The diagram illustrates the cumulative calculation of the formula for  $\Pi$ . It shows the formula  $\Pi = \sqrt{6} \times \sqrt{1 + \frac{1}{4} + \frac{1}{9}}$  with arrows pointing from the terms  $1$ ,  $\frac{1}{4}$ , and  $\frac{1}{9}$  to boxes containing their cumulative values and the resulting  $\Pi$  value.

Cumulative Value	Resulting $\Pi$
1	2.44949
1.25	2.73861
1.36111	2.85774

End of slide

## 2 – Equations with (cumulative) annotations

A formula for  $\Pi$  from Leonhard Euler

$$\Pi = \sqrt{6} \times \sqrt{1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16}}$$

The diagram illustrates the cumulative calculation of the formula for  $\Pi$ . It shows the formula  $\Pi = \sqrt{6} \times \sqrt{1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16}}$  with arrows pointing from the terms  $1$ ,  $\frac{1}{4}$ ,  $\frac{1}{9}$ , and  $\frac{1}{16}$  to boxes containing intermediate values and their corresponding  $\Pi$  values. The final result is shown in a box at the top right:  $\Pi = 2.44949$ .

Term	Intermediate Value	$\Pi$ Value
$1$	$1$	$2.44949$
$1 + \frac{1}{4}$	$1.25$	$2.73861$
$1 + \frac{1}{4} + \frac{1}{9}$	$1.36111$	$2.85774$
$1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16}$	$1.42361$	$2.92261$

End of slide

## 2 – Equations with (cumulative) annotations

A formula for  $\Pi$  from Leonhard Euler

$$\Pi = \sqrt{6} \times \sqrt{1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \dots}$$

The diagram illustrates the cumulative values of the series  $1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \dots$  and the resulting approximations of  $\Pi$ . The cumulative values are shown in yellow boxes, and the corresponding  $\Pi$  values are shown in blue boxes. Arrows indicate the relationship between the terms and the cumulative values.

Cumulative Value	$\Pi$ Value
1	2.44949
1.25	2.73861
1.36111	2.85774
1.42361	2.92261

End of slide

## 2 – Equations with (cumulative) annotations

A formula for  $\Pi$  from Leonhard Euler

$$\Pi = \sqrt{6} \times \sqrt{1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \dots}$$

		<b>= 2.44949</b>
	<b>1</b>	<b>⇒ 2.44949</b>
	<b>1.25</b>	<b>⇒ 2.73861</b>
	<b>1.36111</b>	<b>⇒ 2.85774</b>
	<b>1.42361</b>	<b>⇒ 2.92261</b>

$$= \left( 6 \sum_{n=1}^{\infty} \frac{1}{n^2} \right)^{\frac{1}{2}}$$

End of slide

## 3 – Listing with (progressive) annotations

- ➡ From a manual to introduce to parallel programming with the MPI library
- ➡ First with overlays but without annotations, just using the features of the 'fancyvrb' package

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6
7
8
9
10
11   print *, 'I am process ',rank,' among ',nb_procs
12
13
14 end program WhoAmI
```

End of slide

## 3 – Listing with (progressive) annotations

- ➡ From a manual to introduce to parallel programming with the MPI library
- ➡ First with overlays but without annotations, just using the features of the 'fancyvrb' package

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT(code)
7
8
9
10
11   print *, 'I am process ',rank,' among ',nb_procs
12
13
14 end program WhoAmI
```

End of slide

## 3 – Listing with (progressive) annotations

- ➡ From a manual to introduce to parallel programming with the MPI library
- ➡ First with overlays but without annotations, just using the features of the 'fancyvrb' package

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT(code)
7
8
9
10
11  print *, 'I am process ',rank,' among ',nb_procs
12
13  call MPI_FINALIZE(code)
14 end program WhoAmI
```

End of slide

## 3 – Listing with (progressive) annotations

- ➡ From a manual to introduce to parallel programming with the MPI library
- ➡ First with overlays but without annotations, just using the features of the 'fancyvrb' package

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT(code)
7
8   call MPI_COMM_SIZE(MPI_COMM_WORLD,nb_procs,code)
9
10
11  print *, 'I am process ',rank,' among ',nb_procs
12
13  call MPI_FINALIZE(code)
14 end program WhoAmI
```

End of slide



## 3 – Listing with (progressive) annotations

- ➡ From a manual to introduce to parallel programming with the MPI library
- ➡ First with overlays but without annotations, just using the features of the 'fancyvrb' package

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT(code)
7
8   call MPI_COMM_SIZE(MPI_COMM_WORLD,nb_procs,code)
9   call MPI_COMM_RANK(MPI_COMM_WORLD,rank,code)
10
11  print *, 'I am process ',rank,' among ',nb_procs
12
13  call MPI_FINALIZE(code)
14 end program WhoAmI
```

End of slide

➡ Then the same code, but using both the features of the 'fancyvrb' and 'listings' packages, with an automatic *pretty printing* of the code, after definition of a new language to emphasize the MPI-1 constants and subroutines (see the file `lstlang0.sty`)

➡ We could also use the 'listings' package alone

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6
7
8
9
10
11   print *, 'I am process ',rank,' among ',nb_procs
12
13
14 end program WhoAmI
```

End of slide

➡ Then the same code, but using both the features of the 'fancyvrb' and 'listings' packages, with an automatic *pretty printing* of the code, after definition of a new language to emphasize the MPI-1 constants and subroutines (see the file `lstlang0.sty`)

➡ We could also use the 'listings' package alone

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT(code)
7
8
9
10
11   print *, 'I am process ',rank,' among ',nb_procs
12
13
14 end program WhoAmI
```

End of slide

➡ Then the same code, but using both the features of the 'fancyvrb' and 'listings' packages, with an automatic *pretty printing* of the code, after definition of a new language to emphasize the MPI-1 constants and subroutines (see the file `lstlang0.sty`)

➡ We could also use the 'listings' package alone

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT (code)
7
8
9
10
11  print *, 'I am process ',rank,' among ',nb_procs
12
13  call MPI_FINALIZE (code)
14 end program WhoAmI
```

End of slide

☞ Then the same code, but using both the features of the 'fancyvrb' and 'listings' packages, with an automatic *pretty printing* of the code, after definition of a new language to emphasize the MPI-1 constants and subroutines (see the file `lstlang0.sty`)

☞ We could also use the 'listings' package alone

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT (code)
7
8   call MPI_COMM_SIZE (MPI_COMM_WORLD ,nb_procs ,code)
9
10
11   print *, 'I am process ',rank, ' among ',nb_procs
12
13   call MPI_FINALIZE (code)
14 end program WhoAmI
```

End of slide

☞ Then the same code, but using both the features of the 'fancyvrb' and 'listings' packages, with an automatic *pretty printing* of the code, after definition of a new language to emphasize the MPI-1 constants and subroutines (see the file `lstlang0.sty`)

☞ We could also use the 'listings' package alone

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT (code)
7
8   call MPI_COMM_SIZE (MPI_COMM_WORLD ,nb_procs ,code )
9   call MPI_COMM_RANK (MPI_COMM_WORLD ,rank ,code )
10
11  print *, 'I am process ',rank, ' among ',nb_procs
12
13  call MPI_FINALIZE (code)
14 end program WhoAmI
```

End of slide

➡ And now always the same code, but adding external annotations, using PStricks nodes. This time, all annotations are shown together, without using overlays.

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT (code)
7
8   call MPI_COMM_SIZE ( MPI_COMM_WORLD ,nb_procs ,code )
9   call MPI_COMM_RANK ( MPI_COMM_WORLD ,rank ,code )
10
11  print *, 'I am process ',rank,' among ',nb_procs
12
13  call MPI_FINALIZE (code)
14 end program WhoAmI
```

The diagram shows four annotations with arrows pointing to specific parts of the code:

- Initialization of MPI environment** (blue text in a white box) points to the `MPI_INIT` call on line 6.
- Number of processes for the current execution** (red text in a yellow box) points to the `nb_procs` argument in the `MPI_COMM_SIZE` call on line 8.
- Rank of the process among all of them** (red text in a white box) points to the `rank` argument in the `MPI_COMM_RANK` call on line 9.
- Exit of MPI environment** (blue text in a white box) points to the `MPI_FINALIZE` call on line 13.

- ➔ Then finally the same code again, with the same annotations, but shown in a progressive way, using overlays

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT (code)
7
8   call MPI_COMM_SIZE ( MPI_COMM_WORLD ,nb_procs ,code )
9   call MPI_COMM_RANK ( MPI_COMM_WORLD ,rank ,code )
10
11  print *, 'I am process ',rank,' among ',nb_procs
12
13  call MPI_FINALIZE (code)
14 end program WhoAmI
```



- ➡ Then finally the same code again, with the same annotations, but shown in a progressive way, using overlays

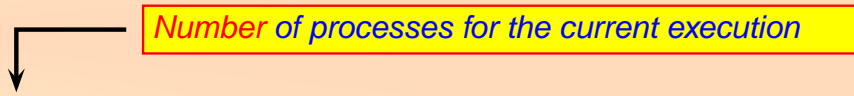
```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT (code)
7
8   call MPI_COMM_SIZE ( MPI_COMM_WORLD ,nb_procs ,code )
9   call MPI_COMM_RANK ( MPI_COMM_WORLD ,rank ,code )
10
11  print *, 'I am process ',rank,' among ',nb_procs
12
13  call MPI_FINALIZE (code)
14 end program WhoAmI
```

Initialization of MPI environment

End of slide

- ➔ Then finally the same code again, with the same annotations, but shown in a progressive way, using overlays

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT (code)
7
8   call MPI_COMM_SIZE ( MPI_COMM_WORLD , nb_procs , code )
9   call MPI_COMM_RANK ( MPI_COMM_WORLD , rank , code )
10
11  print *, 'I am process ',rank,' among ',nb_procs
12
13  call MPI_FINALIZE (code)
14 end program WhoAmI
```



- ➡ Then finally the same code again, with the same annotations, but shown in a progressive way, using overlays

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT (code)
7
8   call MPI_COMM_SIZE ( MPI_COMM_WORLD ,nb_procs ,code )
9   call MPI_COMM_RANK ( MPI_COMM_WORLD ,rank ,code )
10
11   print *, 'I am process ',rank, ' among ',nb_procs
12
13   call MPI_FINALIZE (code)
14 end program WhoAmI
```

↑ Rank of the process among all of them

- ➔ Then finally the same code again, with the same annotations, but shown in a progressive way, using overlays

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT (code)
7
8   call MPI_COMM_SIZE ( MPI_COMM_WORLD ,nb_procs ,code )
9   call MPI_COMM_RANK ( MPI_COMM_WORLD ,rank ,code )
10
11  print *, 'I am process ',rank,' among ',nb_procs
12
13  call MPI_FINALIZE (code) ← Exit of MPI environment
14 end program WhoAmI
```

## 4 – (Cumulative) listing with (cumulative) annotations (I)

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6
7
8
9
10
11   print *, 'I am process ',      , ' among ',
12
13
14 end program WhoAmI
```

End of slide

## 4 – (Cumulative) listing with (cumulative) annotations (I)

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT(code)
7
8
9
10
11   print *, 'I am process ', rank, ' among ', nb_procs
12
13
14 end program WhoAmI
```

End of slide

## 4 – (Cumulative) listing with (cumulative) annotations (I)

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT(code)
7
8
9
10
11  print *, 'I am process ', rank, ' among ', nb_procs
12
13
14 end program WhoAmI
```

*Initialization of MPI environment*



End of slide

## 4 – (Cumulative) listing with (cumulative) annotations (I)

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT(code)
7
8
9
10
11  print *, 'I am process ', rank, ' among ', nb_procs
12
13  call MPI_FINALIZE(code)
14 end program WhoAmI
```

*Initialization of MPI environment*



End of slide



## 4 – (Cumulative) listing with (cumulative) annotations (I)

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT(code)
7
8
9
10
11  print *, 'I am process ', rank, ' among ', nb_procs
12
13  call MPI_FINALIZE(code)
14 end program WhoAmI
```

*Initialization of MPI environment*

*Exit of MPI environment*

End of slide

## 4 – (Cumulative) listing with (cumulative) annotations (I)

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT (code)
7
8   call MPI_COMM_SIZE ( MPI_COMM_WORLD ,nb_procs ,code )
9
10
11   print *, 'I am process ', rank , ' among ', nb_procs
12
13   call MPI_FINALIZE (code)
14 end program WhoAmI
```

*Initialization of MPI environment*

*Exit of MPI environment*

End of slide

## 4 – (Cumulative) listing with (cumulative) annotations (I)

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT (code)
7
8   call MPI_COMM_SIZE ( MPI_COMM_WORLD , nb_procs , code )
9
10
11  print *, 'I am process ', rank , ' among ', nb_procs
12
13  call MPI_FINALIZE (code)
14 end program WhoAmI
```

*Initialization of MPI environment*

*Number of processes for the current execution*

*Exit of MPI environment*

End of slide

## 4 – (Cumulative) listing with (cumulative) annotations (I)

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT (code)
7
8   call MPI_COMM_SIZE ( MPI_COMM_WORLD , nb_procs , code )
9   call MPI_COMM_RANK ( MPI_COMM_WORLD , rank , code )
10
11  print *, 'I am process ', rank , ' among ', nb_procs
12
13  call MPI_FINALIZE (code)
14 end program WhoAmI
```

Initialization of MPI environment

Number of processes for the current execution

Exit of MPI environment

End of slide

## 4 – (Cumulative) listing with (cumulative) annotations (I)

```
1 program WhoAmI
2   implicit none
3   include 'mpif.h'
4   integer :: nb_procs,rank,code
5
6   call MPI_INIT (code)
7
8   call MPI_COMM_SIZE ( MPI_COMM_WORLD , nb_procs , code )
9   call MPI_COMM_RANK ( MPI_COMM_WORLD , rank , code )
10
11  print *, 'I am process ', rank , ' among ', nb_procs
12
13  call MPI_FINALIZE (code)
14 end program WhoAmI
```

Initialization of MPI environment

Number of processes for the current execution

Rank of the process among all of them

Exit of MPI environment

End of slide

## 5 – Listing with (cumulative) annotations (II)

➔ From a manual to introduce to distributed programming with CORBA

```
1 #include <iostream.h>
2 #include <fstream.h>
3
4 #include <OB/CORBA.h>
5 #include <export_skel.h>
6
7 class ClassMatrix : virtual public POA_Exporte {
8
9     private :
10         TypeMatrix A;
11
12     public :
13         ClassMatrix(double init);
14         ~ClassMatrix();
15
16         virtual void MultiplyVector(CORBA::Double alpha,
17                                     TypeVector_slice *vector)
18             throw(CORBA::SystemException);
19 };
```

End of slide

## 5 – Listing with (cumulative) annotations (II)

➔ From a manual to introduce to distributed programming with CORBA

```
1 #include <iostream.h>
2 #include <fstream.h>
3
4 #include <OB/CORBA.h>
5 #include <export_skel.h>
6
7 class ClassMatrix : virtual public POA_Exporte {
8
9     private :
10         TypeMatrix A;
11
12     public :
13         ClassMatrix(double init);
14         ~ClassMatrix( );
15
16         virtual void MultiplyVector(CORBA::Double    alpha,
17                                     TypeVector_slice *vector)
18             throw(CORBA::SystemException);
19 };
```

*File of CORBA required headers*

End of slide

## 5 – Listing with (cumulative) annotations (II)

➔ From a manual to introduce to distributed programming with CORBA

```
1 #include <iostream.h>
2 #include <fstream.h>
3
4 #include <OB/CORBA.h>
5 #include <export_skel.h>
6
7 class ClassMatrix : virtual public POA_Exporte {
8
9     private :
10         TypeMatrix A;
11
12     public :
13         ClassMatrix(double init);
14         ~ClassMatrix( );
15
16         virtual void MultiplyVector(CORBA::Double    alpha,
17                                     TypeVector_slice *vector)
18             throw(CORBA::SystemException);
19 };
```

*File of CORBA required headers*

*File of headers relative to the skeleton generated from the IDL interface by the IDL compiler*

End of slide



## 5 – Listing with (cumulative) annotations (II)

➔ From a manual to introduce to distributed programming with CORBA

```
1 #include <iostream.h>
2 #include <fstream.h>
3
4 #include <OB/CORBA.h>
5 #include <export_skel.h>
6
7 class ClassMatrix : virtual public POA_Exporte {
8
9     private :
10         TypeMatrix A;
11
12     public :
13         ClassMatrix(double init);
14         ~ClassMatrix( );
15
16         virtual void MultiplyVector(CORBA::Double    alpha,
17                                     TypeVector_slice *vector)
18             throw(CORBA::SystemException);
19 };
```

*File of CORBA required headers*

*File of headers relative to the **skeleton** generated from the IDL interface by the IDL compiler*

*The class **ClassMatrix** must now be known inside the CORBA POA*

End of slide

## 5 – Listing with (cumulative) annotations (II)

➔ From a manual to introduce to distributed programming with CORBA

```
1 #include <iostream.h>
2 #include <fstream.h>
3
4 #include <OB/CORBA.h>
5 #include <export_skel.h>
6
7 class ClassMatrix : virtual public POA_Exporte {
8
9     private :
10         TypeMatrix A;
11
12     public :
13         ClassMatrix(double init);
14         ~ClassMatrix();
15
16         virtual void MultiplyVector(CORBA::Double alpha,
17                                     TypeVector_slice *vector)
18             throw(CORBA::SystemException);
19 };
```

*File of CORBA required headers*

*File of headers relative to the skeleton generated from the IDL interface by the IDL compiler*

*The class ClassMatrix must now be known inside the CORBA POA*

*Constructor*

End of slide

## 5 – Listing with (cumulative) annotations (II)

➔ From a manual to introduce to distributed programming with CORBA

```
1 #include <iostream.h>
2 #include <fstream.h>
3
4 #include <OB/CORBA.h>
5 #include <export_skel.h>
6
7 class ClassMatrix : virtual public POA_Exporte {
8
9     private :
10         TypeMatrix A;
11
12     public :
13         ClassMatrix(double init);
14         ~ClassMatrix();
15
16         virtual void MultiplyVector(CORBA::Double alpha,
17                                     TypeVector_slice *vector)
18             throw(CORBA::SystemException);
19 };
```

File of CORBA required headers

File of headers relative to the *skeleton* generated from the IDL interface by the IDL compiler

The class *ClassMatrix* must now be known inside the CORBA POA

Constructor

Destructor

End of slide

## 5 – Listing with (cumulative) annotations (II)

➔ From a manual to introduce to distributed programming with CORBA

```
1 #include <iostream.h>
2 #include <fstream.h>
3
4 #include <OB/CORBA.h>
5 #include <export_skel.h>
6
7 class ClassMatrix : virtual public POA_Exporte {
8
9     private :
10         TypeMatrix A;
11
12     public :
13         ClassMatrix(double init);
14         ~ClassMatrix();
15
16         virtual void MultiplyVector(CORBA::Double    alpha,
17                                     TypeVector_slice *vector)
18             throw(CORBA::SystemException);
19 };
```

*File of CORBA required headers*

*File of headers relative to the skeleton generated from the IDL interface by the IDL compiler*

*The class ClassMatrix must now be known inside the CORBA POA*

*Constructor*

*Destructor*

*Definition of a service to multiply a matrix by a scalar and a vector, with a management of the exceptions done by CORBA*

End of slide

```
20 // Implementation of the methods
21
22 ClassMatrix::ClassMatrix(double cste) {
23     long long i, j;
24
25     for (i = 0; i < N; i++) {
26         for (j = 0; j < N; j++) {
27             A[i][j] = 0.0;}}
28     for (i = 0; i < N; i++) {
29         A[i][i] = cste; }
30 }
31
32 ClassMatrix::~ClassMatrix() {
33     cout << "Destruction of the object" << endl;
34 }
35
36 void ClassMatrix::MultiplyVector(CORBA::Double alpha,
37     TypeVector_slice *vector)
38     throw(CORBA::SystemException) {
39
40     long long i, j;
41     TypeVector tmp;
42
43     for (i = 0; i < N; i++) {
44         tmp[i] = 0.0;
45         for (j = 0; j < N; j++) {
46             tmp[i] = tmp[i] + alpha * A[i][j] * vector[j];
47         }
48     }
49     for (i = 0; i < N; i++) vector[i] = tmp[i];
50 }
```

End of slide

```
20 // Implementation of the methods
```

```
21 ClassMatrix: :ClassMatrix(double cste) {
```

```
22     long long i, j;
```

```
23     for (i = 0; i < N; i++) {
```

```
24         for (j = 0; j < N; j++) {
```

```
25             A[i][j] = 0.0;}}
```

```
26     for (i = 0; i < N; i++) {
```

```
27         A[i][i] = cste;}
```

```
28     }
```

```
29 }
```

```
30 ClassMatrix: :~ClassMatrix() {
```

```
31     cout << "Destruction of the object" << endl;
```

```
32 }
```

```
33 void ClassMatrix: :MultiplyVector(CORBA: :Double alpha,
```

```
34     TypeVector_slice *vector)
```

```
35     throw(CORBA: :SystemException) {
```

```
36     long long i, j;
```

```
37     TypeVector tmp;
```

```
38     for (i = 0; i < N; i++) {
```

```
39         tmp[i] = 0.0;
```

```
40         for (j = 0; j < N; j++) {
```

```
41             tmp[i] = tmp[i] + alpha * A[i][j] * vector[j];
```

```
42         }
```

```
43     }
```

```
44     for (i = 0; i < N; i++) vector[i] = tmp[i];
```

```
45 }
```

Implementation of the constructor:  
initialization of the matrix to the identity matrix

```
20 // Implementation of the methods
```

```
21 ClassMatrix::ClassMatrix(double cste) {
```

```
22     long long i, j;
```

```
23     for (i = 0; i < N; i++) {
```

```
24         for (j = 0; j < N; j++) {
```

```
25             A[i][j] = 0.0;}}
```

```
26     for (i = 0; i < N; i++) {
```

```
27         A[i][i] = cste;}
```

```
28     }
```

```
29 }
```

```
30 }
```

```
31
```

```
32 ClassMatrix::~ClassMatrix() {
```

```
33     cout << "Destruction of the object" << endl;
```

```
34 }
```

```
35
```

```
36 void ClassMatrix::MultiplyVector(CORBA::Double alpha,
```

```
37     TypeVector_slice *vector)
```

```
38     throw(CORBA::SystemException) {
```

```
39     long long i, j;
```

```
40     TypeVector tmp;
```

```
41     for (i = 0; i < N; i++) {
```

```
42         tmp[i] = 0.0;
```

```
43         for (j = 0; j < N; j++) {
```

```
44             tmp[i] = tmp[i] + alpha * A[i][j] * vector[j];
```

```
45         }
```

```
46     }
```

```
47     for (i = 0; i < N; i++) vector[i] = tmp[i];
```

```
48 }
```

```
49 }
```

Implementation of the **constructor**:  
initialization of the matrix to the identity matrix

Implementation of the **destructor**:  
generally memory deallocation

End of slide

```
20 // Implementation of the methods
```

```
21 ClassMatrix: :ClassMatrix(double cste) {
```

```
22     long long i, j;
```

```
23     for (i = 0; i < N; i++) {
```

```
24         for (j = 0; j < N; j++) {
```

```
25             A[i][j] = 0.0;}}
```

```
26     for (i = 0; i < N; i++) {
```

```
27         A[i][i] = cste;}
```

```
28     }
```

```
29 }
```

```
30 }
```

```
31
```

```
32 ClassMatrix: :~ClassMatrix() {
```

```
33     cout << "Destruction of the object" << endl;
```

```
34 }
```

```
35
```

```
36 void ClassMatrix: :MultiplyVector(CORBA: :Double alpha,
```

```
37     TypeVector_slice *vector)
```

```
38     throw(CORBA: :SystemException) {
```

```
39     long long i, j;
```

```
40     TypeVector tmp;
```

```
41     for (i = 0; i < N; i++) {
```

```
42         tmp[i] = 0.0;
```

```
43         for (j = 0; j < N; j++) {
```

```
44             tmp[i] = tmp[i] + alpha * A[i][j] * vector[j];
```

```
45         }
```

```
46     }
```

```
47     for (i = 0; i < N; i++) vector[i] = tmp[i];
```

```
48 }
```

```
49 }
```

Implementation of the **constructor**:  
initialization of the matrix to the identity matrix

Implementation of the **destructor**:  
generally memory deallocation

Service to compute the product of a matrix  
(multiplied by a constant) with a vector, with  
a management of the exceptions done by  
CORBA

End of slide



```
51 // Main program
52
53 int main(int argc, char* argv[])
54 {
55     // Initialization of the CORBA ORB and POA
56     CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
57     CORBA::Object_var poaObj = orb -> resolve_initial_references("RootPOA");
58     PortableServer::POA_var RootPOA = PortableServer::POA::_narrow(poaObj);
59
60     ClassMatrix Matrix( (double) (1.0));
61
62     // Writing of the "universal pointer" IOR in the file "reference"
63     CORBA::String_var str = orb -> object_to_string( Matrix._this() );
64     ofstream out("reference");
65     out << str << endl;
66     out.close();
67
68     RootPOA -> the_POAManager() -> activate();
69     orb -> run();
70
71     orb -> destroy();
72 }
```

End of slide

```
51 // Main program
52
53 int main(int argc, char* argv[])
54 {
55     // Initialization of the CORBA ORB and POA
56     CORBA::ORB_var orb = CORBA::ORB_init(argc, argv); ← Declaration and initialization of the ORB
57     CORBA::Object_var poaObj = orb -> resolve_initial_references("RootPOA");
58     PortableServer::POA_var RootPOA = PortableServer::POA::_narrow(poaObj);
59
60     ClassMatrix Matrix( (double) (1.0));
61
62     // Writing of the "universal pointer" IOR in the file "reference"
63     CORBA::String_var str = orb -> object_to_string( Matrix._this() );
64     ofstream out( "reference" );
65     out << str << endl;
66     out.close();
67
68     RootPOA -> the_POAManager() -> activate();
69     orb -> run();
70
71     orb -> destroy();
72 }
```

End of slide

```
51 // Main program
52
53 int main(int argc, char* argv[])
54 {
55     // Initialization of the CORBA ORB and POA
56     CORBA::ORB_var orb = CORBA::ORB_init(argc, argv); ← Declaration and initialization of the ORB
57     CORBA::Object_var poaObj = orb -> resolve_initial_references("RootPOA");
58     PortableServer::POA_var RootPOA = PortableServer::POA::_narrow(poaObj); } ← Declaration and initialization of the POA
59
60     ClassMatrix Matrix( (double) (1.0));
61
62     // Writing of the "universal pointer" IOR in the file "reference"
63     CORBA::String_var str = orb -> object_to_string( Matrix._this() );
64     ofstream out( "reference" );
65     out << str << endl;
66     out.close();
67
68     RootPOA -> the_POAManager() -> activate();
69     orb -> run();
70
71     orb -> destroy();
72 }
```

End of slide

```
51 // Main program
52
53 int main(int argc, char* argv[])
54 {
55     // Initialization of the CORBA ORB and POA
56     CORBA::ORB_var orb = CORBA::ORB_init(argc, argv); ← Declaration and initialization of the ORB
57     CORBA::Object_var poaObj = orb -> resolve_initial_references("RootPOA"); } ← Declaration and initialization of the POA
58     PortableServer::POA_var RootPOA = PortableServer::POA::_narrow(poaObj);
59
60     ClassMatrix Matrix( (double) (1.0) ); ← Creation of an object Matrix
61
62     // Writing of the "universal pointer" IOR in the file "reference"
63     CORBA::String_var str = orb -> object_to_string( Matrix._this() );
64     ofstream out( "reference" );
65     out << str << endl;
66     out.close();
67
68     RootPOA -> the_POAManager() -> activate();
69     orb -> run();
70
71     orb -> destroy();
72 }
```

End of slide

```
51 // Main program
52
53 int main(int argc, char* argv[])
54 {
55     // Initialization of the CORBA ORB and POA
56     CORBA::ORB_var orb = CORBA::ORB_init(argc, argv); ← Declaration and initialization of the ORB
57     CORBA::Object_var poaObj = orb -> resolve_initial_references("RootPOA"); } ← Declaration and initialization of the POA
58     PortableServer::POA_var RootPOA = PortableServer::POA::_narrow(poaObj);
59
60     ClassMatrix Matrix( (double) (1.0) ); ← Creation of an object Matrix
61
62     // Writing of the "universal pointer" IOR in the file "reference"
63     CORBA::String_var str = orb -> object_to_string( Matrix._this() ); ← Local characters string, used to store the generated reference
64     ofstream out("reference");
65     out << str << endl;
66     out.close();
67
68     RootPOA -> the_POAManager() -> activate();
69     orb -> run();
70
71     orb -> destroy();
72 }
```

End of slide

```
51 // Main program
52
53 int main(int argc, char* argv[])
54 {
55     // Initialization of the CORBA ORB and POA
56     CORBA::ORB_var orb = CORBA::ORB_init(argc, argv); ← Declaration and initialization of the ORB
57     CORBA::Object_var poaObj = orb -> resolve_initial_references("RootPOA"); } ← Declaration and initialization of the POA
58     PortableServer::POA_var RootPOA = PortableServer::POA::_narrow(poaObj);
59
60     ClassMatrix Matrix( (double) (1.0) ); ← Creation of an object Matrix
61
62     // Writing of the "universal pointer" IOR in the file "reference"
63     CORBA::String_var str = orb -> object_to_string( Matrix._this() ); ← Local characters string, used to
64     ofstream out( "reference" ); } store the generated reference
65     out << str << endl; ← Writing of this server reference in the file
66     out.close(); } reference
67
68     RootPOA -> the_POAManager() -> activate();
69     orb -> run();
70
71     orb -> destroy();
72 }
```

End of slide

```
51 // Main program
52
53 int main(int argc, char* argv[])
54 {
55     // Initialization of the CORBA ORB and POA
56     CORBA::ORB_var orb = CORBA::ORB_init(argc, argv); ← Declaration and initialization of the ORB
57     CORBA::Object_var poaObj = orb -> resolve_initial_references("RootPOA"); } ← Declaration and initialization of the POA
58     PortableServer::POA_var RootPOA = PortableServer::POA::_narrow(poaObj);
59
60     ClassMatrix Matrix( (double) (1.0) ); ← Creation of an object Matrix
61
62     // Writing of the "universal pointer" IOR in the file "reference"
63     CORBA::String_var str = orb -> object_to_string( Matrix._this() ); ← Local characters string, used to
64     ofstream out( "reference" ); } store the generated reference
65     out << str << endl;
66     out.close(); } ← Writing of this server reference in the file
67                                     reference
68     RootPOA -> the_POAManager() -> activate(); } ← Activation of the ORB (which will "listen")
69     orb -> run();
70
71     orb -> destroy();
72 }
```

End of slide

```
51 // Main program
52
53 int main(int argc, char* argv[])
54 {
55     // Initialization of the CORBA ORB and POA
56     CORBA::ORB_var orb = CORBA::ORB_init(argc, argv); ← Declaration and initialization of the ORB
57     CORBA::Object_var poaObj = orb -> resolve_initial_references("RootPOA"); } ← Declaration and initialization of the POA
58     PortableServer::POA_var RootPOA = PortableServer::POA::_narrow(poaObj);
59
60     ClassMatrix Matrix( (double) (1.0) ); ← Creation of an object Matrix
61
62     // Writing of the "universal pointer" IOR in the file "reference"
63     CORBA::String_var str = orb -> object_to_string( Matrix._this() ); ← Local characters string, used to
64     ofstream out( "reference" ); } store the generated reference
65     out << str << endl;
66     out.close(); } ← Writing of this server reference in the file
67                                     reference
68     RootPOA -> the_POAManager() -> activate(); } ← Activation of the ORB (which will "listen")
69     orb -> run();
70
71     orb -> destroy(); ← Destruction of the ORB (it will never occur
72 } here)
```

End of slide