

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’^a, 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.

^aThanks 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 Π from Leonhard Euler

Π

End of slide

2 – Equations with (cumulative) annotations

A formula for Π from Leonhard Euler

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


= 2.44949

End of slide

2 – Equations with (cumulative) annotations

A formula for Π from Leonhard Euler

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



= 2.44949

2 – Equations with (cumulative) annotations

A formula for Π from Leonhard Euler

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

$\sqrt{6}$ is annotated with $= 2.44949$

$\sqrt{1}$ is annotated with $1 \Rightarrow 2.44949$

2 – Equations with (cumulative) annotations

A formula for Π from Leonhard Euler

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

The diagram illustrates the calculation of Π using Euler's formula. It shows the formula $\Pi = \sqrt{6} \times \sqrt{1 + \frac{1}{4}}$ with arrows pointing to the numerical results of each part of the calculation:

- The final result Π is **= 2.44949**.
- The intermediate result $1 + \frac{1}{4}$ is **1 \Rightarrow 2.44949**.
- The intermediate result $\frac{1}{4}$ is **1.25 \Rightarrow 2.73861**.

End of slide

2 – Equations with (cumulative) annotations

A formula for Π from Leonhard Euler

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

$1 \Rightarrow 2.44949$

$1.25 \Rightarrow 2.73861$

$1.36111 \Rightarrow 2.85774$

$= 2.44949$

End of slide

2 – Equations with (cumulative) annotations

A formula for Π from Leonhard Euler

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

Term	Sum	Π
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

The final result of the formula is $\Pi = 2.44949$.

End of slide

2 – Equations with (cumulative) annotations

A formula for Π from Leonhard Euler

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

Cumulative Sum	Resulting Π
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 Π from Leonhard Euler

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

Diagram illustrating the cumulative calculation of the series for Π :

- Initial value: $1 \Rightarrow 2.44949$
- Adding $\frac{1}{4}$: $1.25 \Rightarrow 2.73861$
- Adding $\frac{1}{9}$: $1.36111 \Rightarrow 2.85774$
- Adding $\frac{1}{16}$: $1.42361 \Rightarrow 2.92261$

The final result is shown as:

$$= 2.44949$$

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

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
```

Annotations:

- Initialization of MPI environment (points to `MPI_INIT`)
- Number of processes for the current execution (points to `nb_procs`)
- Rank of the process among all of them (points to `rank`)
- Exit of MPI environment (points to `MPI_FINALIZE`)

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
```

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
```

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
```

Number of processes for the current execution

- ☞ 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                                     ↑ Rank of the process among all of them
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) ← 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 ',      , ' 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 ',      , ' among ',
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 ',      , ' among ',
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 ',      , ' among ',
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 ',      , ' 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 ',      , ' 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         {
19             throw(CORBA::SystemException);
20         };
21 }
```

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         }
29     }
30
31     for (i = 0; i < N; i++) {
32         A[i][i] = cste;
33     }
34 }
35
36 ClassMatrix::~ClassMatrix() {
37     cout << "Destruction of the object" << endl;
38 }
39
40 void ClassMatrix::MultiplyVector(CORBA::Double alpha,
41                                     TypeVector_slice *vector)
42     throw(CORBA::SystemException) {
43     long long i, j;
44     TypeVector tmp;
45
46     for (i = 0; i < N; i++) {
47         tmp[i] = 0.0;
48         for (j = 0; j < N; j++) {
49             tmp[i] = tmp[i] + alpha * A[i][j] * vector[j];
50         }
51     }
52     for (i = 0; i < N; i++) vector[i] = tmp[i];
53 }
```

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 }
```

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

```
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 }
```

End of slide

// Implementation of the methods

ClassMatrix::ClassMatrix(double cste) {

long long i, j;

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

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

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

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

A[i][i] = cste;}

}

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

ClassMatrix::~ClassMatrix() {

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

}

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

void **ClassMatrix::MultiplyVector**(CORBA::Double alpha,
TypeVector_slice *vector)

throw(CORBA::SystemException) {

long long i, j;

TypeVector tmp;

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

tmp[i] = 0.0;

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

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

}

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

}

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         }
29     }
30 }
```

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

```
31
32 ClassMatrix::~ClassMatrix() {
33     cout << "Destruction of the object" << endl;
34 }
```

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

```
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 }
```

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");
58     PortableServer::POA_var RootPOA = PortableServer::POA::_narrow(poaObj); } ← Declaration and initialization of the POA
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");
58     PortableServer::POA_var RootPOA = PortableServer::POA::_narrow(poaObj); } ← Declaration and initialization of the POA
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 store the generated reference

```
64     ofstream out( "reference" );
```

```
65     out << str << endl;
```

```
66     out.close();
```

Writing of this server reference in the file *reference*

```
67  
68     RootPOA -> the_POAManager() -> activate();
```

```
69     orb -> run();
```

```
70  
71     orb -> destroy();
```

```
72 }
```

```
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();
```

Writing of this server reference in the file *reference*

```
67  
68 RootPOA -> the_POAManager() -> activate();
```

```
69 orb -> run();
```

Activation of the ORB (which will "listen")

```
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();
```

Writing of this server reference in the file *reference*

```
67  
68     RootPOA -> the_POAManager() -> activate();
```

```
69     orb -> run();
```

Activation of the ORB (which will "listen")

```
70  
71     orb -> destroy();
```

Destruction of the ORB (it will never occur here)

```
72 }
```

End of slide