With Acroread, CTRL-L switch between full screen and window mode
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’¹, 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.

¹Thanks 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

$$\Pi$$
2 – Equations with (cumulative) annotations

A formula for $\Pi$ from Leonhard Euler

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

= 2.44949
2 – Equations with (cumulative) annotations

A formula for $\Pi$ from Leonhard Euler

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

$= 2.44949$
A formula for $\Pi$ from Leonhard Euler

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

= 2.44949

1 $\implies$ 2.44949
A formula for \( \Pi \) from Leonhard Euler

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

\( = 2.44949 \)

\( 1 \rightarrow 2.44949 \)

\( 1.25 \rightarrow 2.73861 \)
A formula for $\Pi$ from Leonhard Euler

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

$= 2.44949$

$1 \Rightarrow 2.44949$

$1.25 \Rightarrow 2.73861$

$1.36111 \Rightarrow 2.85774$
A formula for $\Pi$ from Leonhard Euler

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

$= 2.44949$

1 $\Rightarrow 2.44949$
1.25 $\Rightarrow 2.73861$
1.36111 $\Rightarrow 2.85774$
1.42361 $\Rightarrow 2.92261$
A formula for $\Pi$ from Leonhard Euler

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

$\Pi = 2.44949$
2 – Equations with (cumulative) annotations

A formula for $\Pi$ from Leonhard Euler

\[
\Pi = \sqrt{6} \times \sqrt{1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \cdots}
\]

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

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

```fortran
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs,rank,code

  print *,’I am process ’,rank,’ among ’,nb_procs

end program WhoAmI
```

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

```fortran
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs, rank, code

  call MPI_INIT(code)

  print *, 'I am process ', rank, ' among ', nb_procs

end program WhoAmI
```
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

```fortran
program WhoAmI
    implicit none
    include 'mpif.h'
    integer :: nb_procs, rank, code

    call MPI_INIT(code)

    print *, 'I am process ', rank, ' among ', nb_procs

    call MPI_FINALIZE(code)
end program WhoAmI
```
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

```
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs, rank, code

  call MPI_INIT(code)

  call MPI_COMM_SIZE(MPI_COMM_WORLD, nb_procs, code)

  print *, 'I am process ', rank, ' among ', nb_procs

  call MPI_finalize(code)
end program WhoAmI
```
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

```fortran
program WhoAmI
    implicit none
    include 'mpif.h'
    integer :: nb_procs, rank, code

    call MPI_INIT(code)

    call MPI_COMM_SIZE(MPI_COMM_WORLD, nb_procs, code)
    call MPI_COMM_RANK(MPI_COMM_WORLD, rank, code)

    print *, 'I am process ', rank, ' among ', nb_procs

    call MPI_FINALIZE(code)
end program WhoAmI
```

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.

```plaintext
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs, rank, code

  print *, 'I am process ', rank, ' among ', nb_procs
end program WhoAmI
```
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

```fortran
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs, rank, code

  call MPI_INIT(code)

  print *, 'I am process ', rank, ' among ', nb_procs

end program WhoAmI
```
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.

```
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs, rank, code

  call MPI_INIT(code)

  print *, 'I am process ', rank, ' among ', nb_procs

  call MPI_FINALIZE(code)
end program WhoAmI
```
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

```fortran
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs, rank, code

  call MPI_INIT(code)

  call MPI_COMM_SIZE(MPI_COMM_WORLD, nb_procs, code)

  print *, 'I am process ', rank, ' among ', nb_procs

  call MPI_FINALIZE(code)
end program WhoAmI
```
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

```fortran
program WhoAmI
implicit none
include 'mpif.h'
integer :: nb_procs, rank, code

call MPI_INIT(code)

call MPI_COMM_SIZE(MPI_COMM_WORLD, nb_procs, code)
call MPI_COMM_RANK(MPI_COMM_WORLD, rank, code)

print *, 'I am process ', rank, ' among ', nb_procs

call MPI_FINALIZE(code)
end program WhoAmI
```
And now always the same code, but adding external annotations, using PSTricks nodes. This time, all annotations are shown together, without using overlays.

```
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs, rank, code
  call MPI_INIT (code)
  call MPI_COMM_SIZE (MPI_COMM_WORLD, nb_procs, code)
  call MPI_COMM_RANK (MPI_COMM_WORLD, rank, code)
  print *, 'I am process ', rank, ' among ', nb_procs
  call MPI_FINALIZE (code)
end program WhoAmI
```
Then finally the same code again, with the same annotations, but shown in a progressive way, using overlays.

```fortran
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs, rank, code

  call MPI_INIT(code)

  call MPI_COMM_SIZE(MPI_COMM_WORLD, nb_procs, code)
  call MPI_COMM_RANK(MPI_COMM_WORLD, rank, code)

  print *, 'I am process ', rank, ' among ', nb_procs

  call MPI_FINALIZE(code)
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
Then finally the same code again, with the same annotations, but shown in a progressive way, using overlays.

```fortran
program WhoAmI
    implicit none
    include 'mpif.h'
    integer :: nb_procs, rank, code

    call MPI_INIT(code)

    call MPI_COMM_SIZE(MPI_COMM_WORLD,nb_procs,code)
    call MPI_COMM_RANK(MPI_COMM_WORLD,rank,code)

    print *, 'I am process ', rank, ' among ', nb_procs

    call MPI_FINALIZE(code)
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
Then finally the same code again, with the same annotations, but shown in a progressive way, using overlays

program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs, rank, code

  call MPI_INIT(code)

  call MPI_COMM_SIZE(MPI_COMM_WORLD, nb_procs, code)
  call MPI_COMM_RANK(MPI_COMM_WORLD, rank, code)

  print *, 'I am process ', rank, ' among ', nb_procs

  call MPI_FINALIZE(code)
end program WhoAmI
Then finally the same code again, with the same annotations, but shown in a progressive way, using overlays

```fortran
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs, rank, code

  call MPI_INIT(code)

  call MPI_COMM_SIZE(MPI_COMM_WORLD, nb_procs, code)
  call MPI_COMM_RANK(MPI_COMM_WORLD, rank, code)

  print *, 'I am process ', rank, ' among ', nb_procs

  call MPI_FINALIZE(code)
end program WhoAmI
```

Initialization of MPI environment
Number of processes for the current execution
Rank of the process among all of them

Rank of the process among all of them

Seminardemonstrationfiles–Overlays(II)
Version 1.0–June2002

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

```fortran
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs, rank, code

  call MPI_INIT(code)

  call MPI_COMM_SIZE(MPI_COMM_WORLD, nb_procs, code)
  call MPI_COMM_RANK(MPI_COMM_WORLD, rank, code)

  print *, 'I am process ', rank, ' among ', nb_procs

  call MPI_FINALIZE(code)
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
program WhoAmI
   implicit none
   include 'mpif.h'
   integer :: nb_procs, rank, code

   print *, 'I am process ', rank, ' among ',

end program WhoAmI
program WhoAmI
    implicit none
    include 'mpif.h'
    integer :: nb_procs, rank, code

    call MPI_INIT(code)

    print *, 'I am process ', ', ', ' among ',

end program WhoAmI
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs, rank, code

  call MPI_INIT(code)

  print *, 'I am process ', ', ', ' among ',

end program WhoAmI

Initialization of MPI environment
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs, rank, code

  call MPI_INIT(code)

  print *, 'I am process ', rank, ' among ', nb_procs

  call MPI_FINALIZE(code)
end program WhoAmI
Demonstration of overlays (ii)

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

```fortran
program WhoAmI
   implicit none
   include 'mpif.h'
   integer :: nb_procs, rank, code

   call MPI_INIT(code)

   print *, 'I am process ', rank, ' among ', nb_procs

   call MPI_FINALIZE(code)
end program WhoAmI
```

Initialization of MPI environment

Exit of MPI environment

Seminar demonstration files – Overlays (II)
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs, rank, code

  call MPI_INIT(code)

  call MPI_COMM_SIZE(MPI_COMM_WORLD, nb_procs, code)

  print *, 'I am process ', rank, ' among ', nb_procs

  call MPI_FINALIZE(code)
end program WhoAmI
Demonstration of overlays (ii)

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

```fortran
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs, rank, code

  call MPI_INIT(code)

  call MPI_COMM_SIZE(MPI_COMM_WORLD, nb_procs, code)

  print *, 'I am process ', rank, ' among ', nb_procs

  call MPI_FINALIZE(code)

end program WhoAmI
```

- **Initialization of MPI environment**
- **Number of processes for the current execution**
- **Exit of MPI environment**
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs, rank, code

  call MPI_INIT (code)

  call MPI_COMM_SIZE (MPI_COMM_WORLD, nb_procs, code)
  call MPI_COMM_RANK (MPI_COMM_WORLD, rank, code)

  print *, 'I am process ', rank, ' among ', nb_procs

  call MPI_FINALIZE (code)
end program WhoAmI
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs, rank, code

  call MPI_INIT (code)  ! Initialization of MPI environment

  call MPI_COMM_SIZE (MPI_COMM_WORLD, nb_procs, code)  ! Number of processes for the current execution
  call MPI_COMM_RANK (MPI_COMM_WORLD, rank, code)      ! Rank of the process among all of them

  print *, 'I am process ', rank, ' among ', nb_procs

  call MPI_FINALIZE (code)  ! Exit of MPI environment

end program WhoAmI
5 – Listing with (cumulative) annotations (II)

From a manual to introduce to distributed programming with CORBA

```cpp
#include <iostream.h>
#include <fstream.h>

#include <OB/CORBA.h>
#include <export_skel.h>

class ClassMatrix : virtual public POA_Exporte {

private:
    TypeMatrix A;

public:
    ClassMatrix(double init);
    ~ClassMatrix();

    virtual void MultiplyVector(CORBA::Double alpha,
                                TypeVector_slice *vector)
        throw(CORBA::SystemException);
};
```

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
5 – Listing with (cumulative) annotations (II)

From a manual to introduce to distributed programming with CORBA

```c++
#include <iostream.h>
#include <fstream.h>
#include <OB/CORBA.h>
#include <export_skel.h>

class ClassMatrix : virtual public POA_Exporte {

private:
    TypeMatrix A;

public:
    ClassMatrix(double init);
    ~ClassMatrix();
    virtual void MultiplyVector(CORBA::Double alpha,
                                 TypeVector_slice *vector)
        throw(CORBA::SystemException);
};
```

File of CORBA required headers

File of headers relativete 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
From a manual to introduce to distributed programming with CORBA

```cpp
#include <iostream.h>
#include <fstream.h>

#include <OB/CORBA.h>
#include <export_skel.h>

class ClassMatrix : virtual public POA_Exporte {

private:
  TypeMatrix A;

public:
  ClassMatrix(double init);
  ~ClassMatrix();

  virtual void MultiplyVector(CORBA::Double alpha,
                              TypeVector_slice *vector)
    throw(CORBA::SystemException);

};
```

File of CORBA required headers
File of headers relative to the skeleton generated from the IDL interface by the IDL compiler
5 – Listing with (cumulative) annotations (II)

From a manual to introduce to distributed programming with CORBA

```cpp
#include <iostream.h>
#include <fstream.h>

#include <OB/CORB_A.h>
#include <export_skel.h>

class ClassMatrix : virtual public POA_Exporte {

private:
    TypeMatrix A;

public:
    ClassMatrix(double init);
    ~ClassMatrix();

    virtual void MultiplyVector(CORBA::Double alpha,
                                TypeVector_slice *vector)
        throw(CORBA::SystemException);

};
```

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
From a manual to introduce to distributed programming with CORBA

```cpp
#include <iostream.h>
#include <fstream.h>

#include <OB/CORBA.h>
#include <export_skel.h>

class ClassMatrix : virtual public POA_Exporte {

private:
  TypeMatrix A;

public:
  ClassMatrix(double init);
  ~ClassMatrix();

  virtual void MultiplyVector(CORBA::Double alpha,
                             TypeVector_slice *vector)
    throw(CORBA::SystemException);

};
```

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
5 – Listing with (cumulative) annotations (II)

From a manual to introduce to distributed programming with CORBA

```cpp
#include <iostream.h>
#include <fstream.h>

#include <OB/CORBA.h>
#include <export_skel.h>

class ClassMatrix : virtual public POA_Exporte {

private:
    TypeMatrixA;

public:
    ClassMatrix(double init);
    ~ClassMatrix();

    virtual void MultiplyVector(CORBA::Double alpha,
                                TypeVector_slice *vector)
    throw(CORBA::SystemException);

};
```

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
From a manual to introduce to distributed programming with CORBA

```cpp
#include <iostream.h>
#include <fstream.h>

#include <OB/CORBA.h>
#include <export_skel.h>

class ClassMatrix : virtual public POA_Exporte {

private:
    TypeMatrix A;

public:
    ClassMatrix(double init);
    ~ClassMatrix();

    virtual void MultiplyVector(CORBA::Double alpha,
                                 TypeVector_slice *vector)
    throw(CORBA::SystemException);

};
```

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
// 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;
    }
}

ClassMatrix::~ClassMatrix() {
    cout << "Destruction of the object" << endl;
}

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

ClassMatrix::~ClassMatrix() {
    cout << "Destruction of the object" << endl;
}

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++) {
        }
    }

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


// Implementation of the methods

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

ClassMatrix::~ClassMatrix() {
    cout << "Destruction of the object" << endl;
}

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

Implementation of the constructor:初始化矩阵为单位矩阵

Implementation of the destructor:一般情况下是内存释放

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

```cpp
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;
    }
}

ClassMatrix::~ClassMatrix () {
    cout << "Destruction of the object" << endl;
}

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

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
// Main program

int main(int argc, char* argv[])
{
    // Initialization of the CORBA ORB and POA
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
    CORBA::Object_var poaObj = orb -> resolve_initial_references("RootPOA");
    PortableServer::POA_var RootPOA = PortableServer::POA::_narrow(poaObj);

    ClassMatrix Matrix((double) (1.0));

    // Writing of the "universal pointer" IOR in the file "reference"
    CORBA::String_var str = orb -> object_to_string( Matrix._this() );
    ofstream out("reference");
    out << str << endl;
    out.close();

    RootPOA -> the_POAManager() -> activate();
    orb -> run();

    orb -> destroy();
}
// Main program

int main(int argc, char* argv[])
{
    // Initialization of the CORBA ORB and POA
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
    CORBA::Object_var poaObj = orb -> resolve_initial_references("RootPOA");
    PortableServer::POA_var RootPOA = PortableServer::POA::_narrow(poaObj);

    ClassMatrix Matrix((double) (1.0));

    // Writing of the "universal pointer" IOR in the file "reference"
    CORBA::String_var str = orb -> object_to_string( Matrix._this() );
    ofstream out("reference");
    out << str << endl;
    out.close();

    RootPOA -> the_POAManager() -> activate();
    orb -> run();

    orb -> destroy();
}
Demonstration of overlays (ii)

// Main program

int main(int argc, char* argv[])
{
    // Initialization of the CORBA ORB and POA
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
    CORBA::Object_var poaObj = orb -> resolve_initial_references("RootPOA");
    PortableServer::POA_var RootPOA = PortableServer::POA::_narrow(poaObj);

    ClassMatrix Matrix((double)(1.0));

    // Writing of the "universal pointer" IOR in the file "reference"
    CORBA::String_var str = orb -> object_to_string(Matrix._this());
    ofstream out("reference");
    out << str << endl;
    out.close();

    RootPOA -> the_POAManager() -> activate();
    orb -> run();

    orb -> destroy();
}
// Main program

int main(int argc, char* argv[])
{
    // Initialization of the CORBA ORB and POA
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
    CORBA::Object_var poaObj = orb -> resolve_initial_references("RootPOA");
    PortableServer::POA_var RootPOA = PortableServer::POA::_narrow(poaObj);

    ClassMatrix Matrix((double)(1.0));

    // Writing of the "universal pointer" IOR in the file "reference"
    CORBA::String_var str = orb -> object_to_string( Matrix._this() );
    ofstream out("reference");
    out << str << endl;
    out.close();

    RootPOA -> the_POAManager() -> activate();
    orb -> run();

    orb -> destroy();
}
```c++
int main(int argc, char* argv[]) {

    // Initialization of the CORBA ORB and POA
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
    CORBA::Object_var poaObj = orb -> resolve_initial_references("RootPOA");
    PortableServer::POA_var RootPOA = PortableServer::POA::_narrow(poaObj);

    ClassMatrix Matrix((double)(1.0));

    // Writing of the "universal pointer" IOR in the file "reference"
    CORBA::String_var str = orb -> object_to_string( Matrix._this() );
    ofstream out("reference");
    out << str << endl;
    out.close();

    RootPOA -> the_POAManager() -> activate();
    orb -> run();
    orb -> destroy();
}
```
// Main program

int main(int argc, char* argv[])
{
    // Initialization of the CORBA ORB and POA
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
    CORBA::Object_var poaObj = orb->resolve_initial_references("RootPOA");
    PortableServer::POA_var RootPOA = PortableServer::POA::_narrow(poaObj);

    ClassMatrix Matrix((double) (1.0));

    // Writing of the "universal pointer" IOR in the file "reference"
    CORBA::String_var str = orb->object_to_string( Matrix._this() );
    ofstream out("reference");
    out << str << endl;
    out.close();

    RootPOA->the_POAManager()->activate();
    orb->run();

    orb->destroy();
}
```c++
int main(int argc, char* argv[]) {
    // Initialization of the CORBA ORB and POA
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
    CORBA::Object_var poaObj = orb -> resolve_initial_references("RootPOA");
    PortableServer::POA_var RootPOA = PortableServer::POA:_narrow(poaObj);

    ClassMatrix Matrix((double) (1.0));

    // Writing of the "universal pointer" IOR in the file "reference"
    CORBA::String_var str = orb -> object_to_string( Matrix._this() );
    ofstream out("reference");
    out << str << endl;
    out.close();

    RootPOA -> the_POAManager() -> activate();
    orb -> run();

    orb -> destroy();
}
```

- **Declaration and initialization of the ORB**
- **Declaration and initialization of the POA**
- **Creation of an object Matrix**
- **Local characters string, used to store the generated reference**
- **Writing of this server reference in the file reference**
- **Activation of the ORB (which will "listen")**

Seminar demonstration files – Overlays (II)
Demonstration of overlays (ii)

```
// Main program

int main(int argc, char* argv[])
{
    // Initialization of the CORBA ORB and POA
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
    CORBA::Object_var poaObj = orb -> resolve_initial_references("RootPOA");
    PortableServer::POA_var RootPOA = PortableServer::POA::_narrow(poaObj);

    ClassMatrix Matrix((double) (1.0));

    // Writing of the "universal pointer" IOR in the file "reference"
    CORBA::String_var str = orb -> object_to_string( Matrix._this() );
    ofstream out("reference");
    out << str << endl;
    out.close();

    RootPOA -> the_POAManager() -> activate();
    orb -> run();
    orb -> destroy();
}
```