ENSIETA

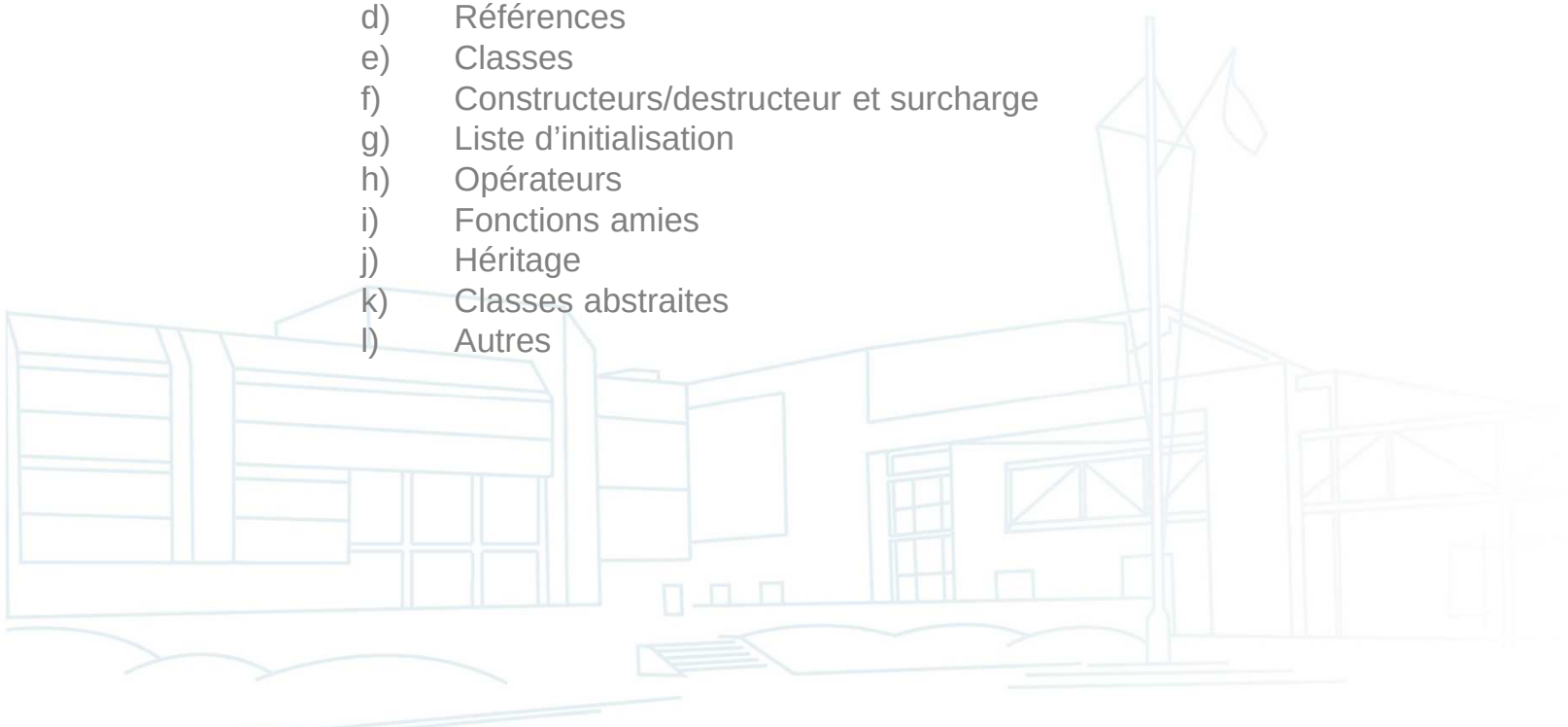
ÉCOLE NATIONALE SUPÉRIEURE D'INGÉNIEURS

Introduction au langage C++

But

- Programmer une classe en C++ permettant de faire du calcul par intervalles
- L'utiliser pour résoudre un problème de localisation « range only » (EX : plusieurs émetteurs et récepteurs radar qui tentent de localiser un cible)

> Sommaire



Introduction au langage C++

Présentation

- Le C est inclus (à 99%) dans le C++
- Le C++ rajoute des notions de programmation orientée objet (classe, héritage, polymorphisme… comme en Java), ainsi que des facilités d'écriture (surcharge d'opérateurs…)

Hello World!

```
#include <iostream>

int main(int argc, char* argv[])
{
    std::cout << "Hello\n";

    return 0;
}
```

Notion de namespace

Opérateur

Namespace

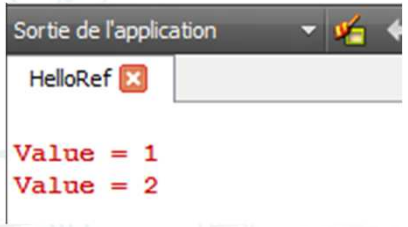
Plus besoin d'utiliser std::

```
#include <iostream>

using namespace std;

int main(int argc, char* argv[])
{
    cout << "Hello\n";

    return 0;
}
```

Références

```
#include <iostream>

void functionWithRef(int& value)
{
    value = 2;
}

int main(int argc, char* argv[])
{
    int a = 1;

    std::cout << "Value = " << a << "\n";

    functionWithRef(a);

    std::cout << "Value = " << a << "\n";

    return 0;
}
```

Paramètre passé par référence



Classes

```
#ifndef TESTCLASS_H
#define TESTCLASS_H

class TestClass
{
private:
    double value;
public:
    // Member functions (EX : getters and setters)
    double getValue() const;
    void setValue(double);
};

#endif // TESTCLASS_H
```

TestClass.h

```
#include "TestClass.h"

// Member functions (EX : getters and setters)
double TestClass::getValue() const
{
    return value;
}

void TestClass::setValue(double value)
{
    this->value = value;
}
```

TestClass.cpp

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

using namespace std;

int main(int argc, char* argv[])
{
    TestClass tst;

    tst.setValue(2.1);

    cout << tst.getValue() << endl;

    return 0;
}
```

Main.cpp

Constructeurs/Destructeur et surcharge

```
#ifndef TESTCLASS_H
#define TESTCLASS_H

#include <iostream>

class TestClass
{
private:
    double value;
public:
    // Constructors
    TestClass();
    TestClass(const TestClass&);
    TestClass(const double&);

    // Destructor
    ~TestClass();
};

#endif // TESTCLASS_H
```

TestClass.h

```
#include "TestClass.h"

using namespace std;

TestClass::TestClass()
{
    value = 0;
}

TestClass::TestClass(const TestClass& a)
{
    value = a.value;
}

TestClass::TestClass(const double& value)
{
    this->value = value;
}

// Destructor
TestClass::~TestClass()
{
    value = 0;
}
```

TestClass.cpp

Liste d'initialisation

```
TestClass::TestClass(const TestClass& a) :
        value(a.value),
        tmpValue(1)
        // Initialization list
{

}
```

L'initialisation des membres dans le constructeur peut être faite de cette façon

Opérateurs

```cpp
#ifndef TESTCLASS_H
#define TESTCLASS_H

class TestClass
{
private:
    double value;
public:
    // Member functions (EX : getters and setters)
    double getValue() const;
    void setValue(double);

    // Member operators (EX : unary operator =)
    TestClass& operator=(const TestClass&);
};

#endif // TESTCLASS_H
```

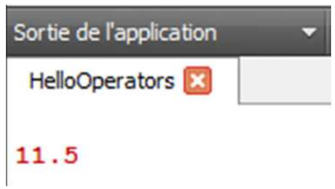
TestClass.h

```cpp
#include "TestClass.h"

// Member functions (EX : getters and setters)
double TestClass::getValue() const
{
    return value;
}

void TestClass::setValue(double value)
{
    this->value = value;
}

// Member operators (EX : unary operator =)
TestClass& TestClass::operator=(const TestClass& a)
{
    value = a.value+10;

    return *this;
}
```

TestClass.cpp

Opérateurs

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

using namespace std;

int main(int argc, char* argv[])
{
    TestClass tst1;
    TestClass tst2;

    tst2.setValue(1.5);

    tst1 = tst2;

    cout << tst1.getValue() << endl;

    return 0;
}
```



Résultat

Main.cpp

Fonctions amies

```
#ifndef TESTCLASS_H
#define TESTCLASS_H

#include <iostream>

class TestClass
{
private:
    double value;
public:
    // Member functions (EX : getters and setters)
    double getValue() const;
    void setValue(double);

    // Friend functions
    friend TestClass compute(const TestClass&, const TestClass&);
};

#endif // TESTCLASS_H
```

TestClass.h

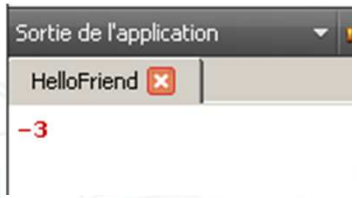
Fonctions amies

```cpp
#include "TestClass.h"

// Member functions (EX : getters and setters)
double TestClass::getValue() const
{
    return value;
}

void TestClass::setValue(double value)
{
    this->value = value;
}

// Friend functions
TestClass compute(const TestClass& a, const TestClass& b)
{
    TestClass c;

    c.setValue(a.value + 2* b.value);

    return c;
}
```

TestClass.cpp

Fonctions amies

```
#include "TestClass.h"

using namespace std;

int main(int argc, char* argv[])
{
    TestClass tst1;
    TestClass tst2;

    tst1.setValue(1);
    tst2.setValue(-2);

    TestClass tst = compute(tst1, tst2);

    cout << tst.getValue() << endl;

    return 0;
}
```



Main.cpp

Héritage

Classe de base

Classe dérivées

```
// constructors and derived classes
#include <iostream>
using namespace std;

class mother {
  public:
    mother ()
      { cout << "mother: no parameters\n"; }
    mother (int a)
      { cout << "mother: int parameter\n"; }
};

class daughter : public mother {
  public:
    daughter (int a)
      { cout << "daughter: int parameter\n\n"; }
};

class son : public mother {
  public:
    son (int a) : mother (a)
      { cout << "son: int parameter\n\n"; }
};

int main () {
  daughter cynthia (0);
  son daniel(0);

  return 0;
}
```

```
Sortie de l'application
HelloInheritance
mother: no parameters
daughter: int parameter

mother: int parameter
son: int parameter
```

Appel à un constructeur spécifique de la classe de base

Classes abstraites

Classe de base abstraite

Classe dérivées

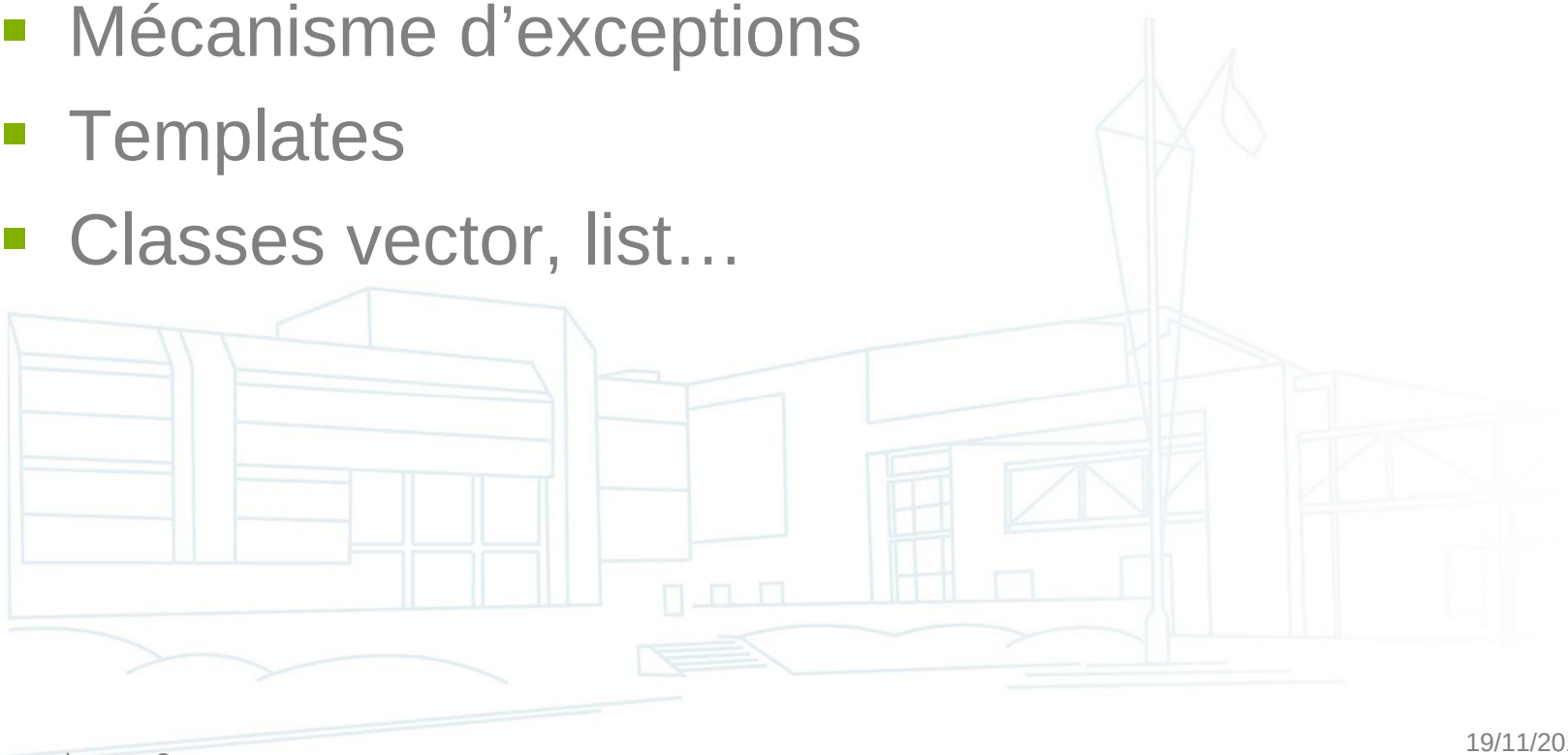
```
#include <iostream>
using namespace std;

class CPolygon {
protected:
    int area;
public:
    virtual int computeArea(void) = 0;
};

class CRectangle : virtual public CPolygon {
private:
    int width, height;
public:
    CRectangle(int w, int h) : width(w), height(h) {}
    int computeArea(void)    {
        area = width * height;
        return area;
    }
};

class CSquare : virtual public CPolygon {
private:
    int width;
public:
    CSquare(int w) : width(w) {}
    int computeArea(void)    {
        area = width * width;
        return area;
    }
};
```

Classes abstraites

```
int main() {
    CRectangle rect = CRectangle(3,4);
    CSquare square = CSquare(2);
    CPolygon * pPoly1 = &rect;
    CPolygon * pPoly2 = &square;
    cout << pPoly1->computeArea() << endl;
    cout << pPoly2->computeArea() << endl;
    return 0;
}
```

Autres apports du C++ par rapport au C

- new, delete et new[], delete[] en C++ à la place de malloc() et free() en C pour utiliser des pointeurs et tableaux à taille variable
- Mécanisme d'exceptions
- Templates
- Classes vector, list…



Annexes

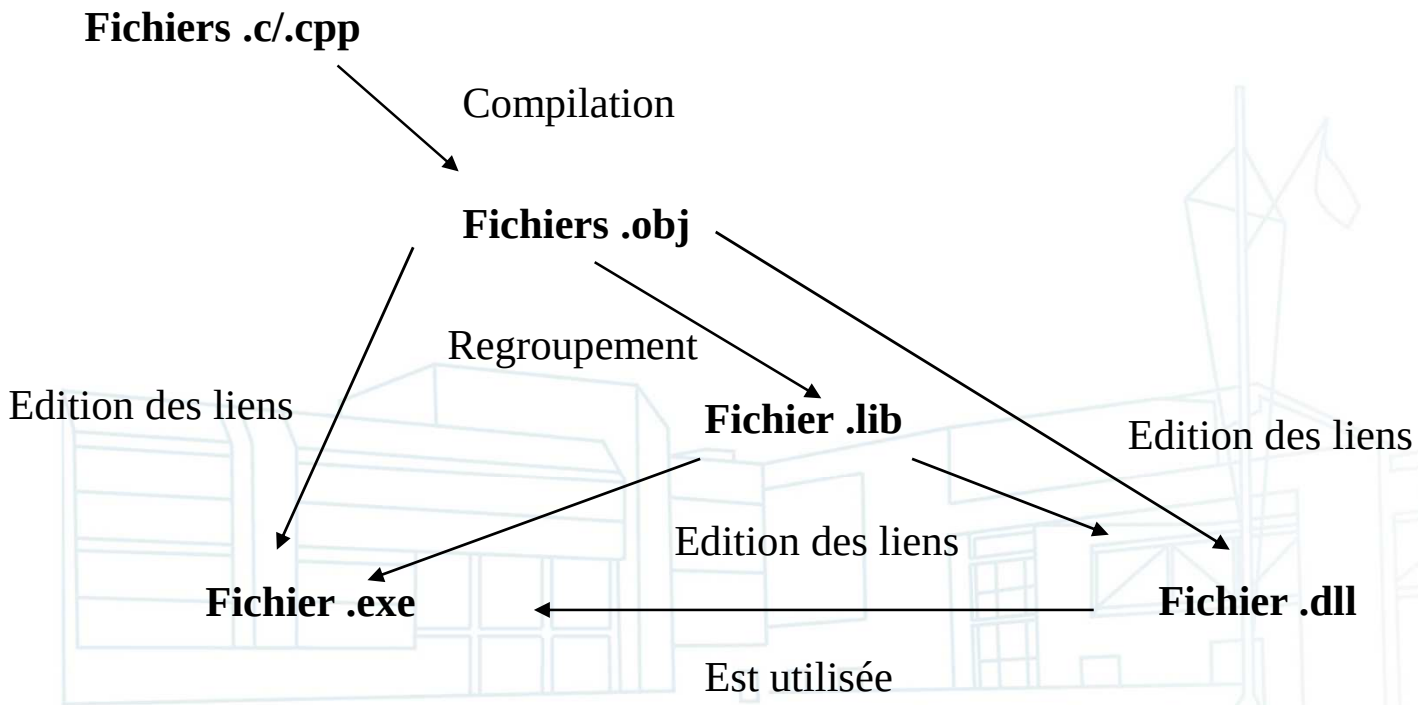
- Le C est inclus (à 99%) dans le C++ : quand on fait du C, on fait aussi du C++ mais l'inverse n'est pas forcément vrai
- Du code C ou C++ peut être écrit dans un fichier .cpp mais seul du code C peut être écrit dans un fichier .c
- Dans un .h, on peut écrire du C ou du C++, mais il faut que les .c n'incluent que des .h avec du C

- Différences entre les compilateurs Windows et Linux
  - Linux
    - Le compilateur C le plus utilisé est GCC
    - Son équivalent C++ est G++
  - Windows
    - GCC/G++ existent avec Cygwin et MinGW
    - Différents IDE existent et fournissent leurs propres compilateurs
      - Microsoft Visual Studio avec CL
      - Borland C++ Builder / Turbo C++ / Borland Developper Studio avec BCC32
      - Code Blocks / Dev-C++ avec MinGW

- Différences entre les compilateurs Windows et Linux

Equivalences Linux / Windows		
	Linux/GCC	Windows/Visual C++
Fichiers objets	.o	.obj
Bibliothèque statique	.a	.lib
Bibliothèque dynamique	.so	.dll
Exécutable	-	.exe

- Utilisation de bibliothèques de fonctions externes



- Utilisation de bibliothèques de fonctions externes
  - Cas où on possède des fichiers .h et .c/.cpp
    - Exemple : on a Main.cpp qui doit appeler des fonctions déclarées dans Lib.h et définies dans Lib.cpp
      - On met #include « Lib.h » dans Main.cpp
      - On copie Lib.h et Lib.cpp dans le dossier de Main.cpp
      - On compile et lie Lib.cpp et Main.cpp en les ajoutant au projet

- Utilisation de bibliothèques de fonctions externes
  - Cas où on possède des fichiers .h, .lib et .dll
    - Exemple : on a Main.cpp qui doit appeler des fonctions déclarées dans Lib.h et définies dans Lib.lib et Lib.dll
      - On met #include « Lib.h » dans Main.cpp
      - On ajoute le dossier de Lib.h dans les chemins de recherche de fichiers .h du projet
      - On ajoute le dossier de Lib.lib dans les chemins de recherche de fichiers .lib du projet
      - On compile Main.cpp et lie avec Lib.lib en les ajoutant au projet
      - On ajoute le dossier de Lib.dll à la variable d'environnement PATH du système