Les exceptions

Comment gérer les erreurs potentielles et avérées.

La définition d'une exception

Si lors de l'exécution d'une fonction, un événement pouvant conduire à l'interruption de l'exécution se produit, cet événement est une exception.

- Tout dysfonctionnement est constitutif d'une exception
- Toute exception n'entraîne pas systématiquement un dysfonctionnement

Une liste non exhaustive d'exceptions

- Exécution d'une opération produisant une condition anormale détectée par le système (division par 0)
- Appel d'une routine qui échoue
- Violation d'une pré-condition
- Violation d'une post-condition
- Violation d'un invariant de classe
- Violation d'un invariant de boucle
- Déclenchement par le logiciel d'une exception

Causes d’un échec d’une fonction

Une fonction échoue si et seulement si une exception se produit durant son exécution et cette exception n’est pas gérée par la fonction

- Il est possible de capturer des exceptions afin de récupérer de cet état d’erreurs
- La définition d’échec et d’exception dépendent mutuellement l’une de l’autre

Le déclenchement d'une exception en C++

En C++

```
throw Expression
```

Où Expression est d'un type quelconque

La capture d'une expression en C++

```
try
   { …
   }
catch(type1 E)
   {…}
catch(type2 E)
   {…}
```

- `try` : début du bloc surveillé
- `catch` : capture les exceptions qui dérivent de *type1* et effectue le code qui suit

Contrôler les exceptions (C++14,17 et 20)

- Ajouter une annotation qui indique si une fonction génère une exception
  - double divide(double theNumerator, double theDivisor) noexcept(false);
  - void doSomething() noexcept;

TD – Partie 1

- Manipuler des exceptions en C++

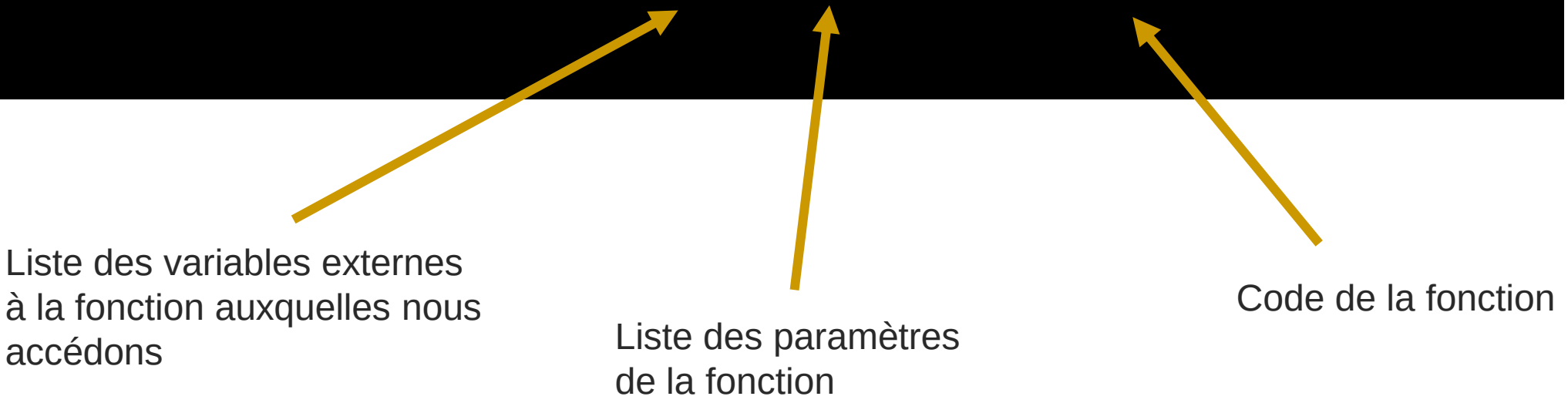
Les fonctions anonymes

Où comment manipuler des fonctions comme des données.

Les fonctions anonymes en C++

Une fonction est un objet de type : std::function

```
std::function<int(int)> func2 = [](int i) { return i + 4; };
```



Les fonctions anonymes en C++

Intérêt des fonctions anonymes :

elles peuvent être passées en argument.

```
std::vector<int> v = {1, 2, 3, 4, 5, 6, 7};
int x = 5;
v.erase(std::remove_if(v.begin(), v.end(),
        [x](int n) { return n < x; }), v.end());
```

L’inférence de types en C++

Comment se simplifier l’écriture de code
et augmenter l’expressivité du langage.

L‘inférence de type en C++

- auto x = expr;

Le type de x est déterminée à partir du type inférée pour l’expression ‘expr’.

- const auto& i = expr;

Le type ‘TE’ de l’expression ‘expr’ est calculée et on détermine s’il existe un type ‘T’ tel que :

‘TE’ puisse être convertie en const T&

```
auto lambda = [](int x) { return x + 3; };
```

Intérêt de ‚auto‘ en C++

Simplifier l‘écriture de code :

```
std::vector<int> myVec = {1, 2, 3};
for(std::vector<int>::const_iterator it = myVec.begin(); it != myVec.end()
; it++)
{ ... }
```

qui se réécrit en :

```
std::vector<int> myVec = {1, 2, 3};
for(auto it = myVec.begin(); it != myVec.end(); it++)
{ ... }

auto lambda = [](int x) { return x + 3; };
```

Intérêt de ‚auto‘ en C++

Généraliser les définitions des „template“ et des fonctions anonymes:

```
template<auto n> struct B
{
  auto value = n;
};

auto glambda = [](auto a) { return a; };
```