Chapitre 8 – Exceptions en C++

1. Pourquoi les exceptions ?

En C++, les exceptions permettent de **gérer les erreurs** proprement.

Sans exception :

- Il faut vérifier les codes de retour partout
- On risque d'oublier
- Le programme devient illisible

Avec les exceptions :

✓ séparation claire entre **code normal** et **code d'erreur**

✓ propagation automatique de l'erreur

✓ arrêt propre et contrôlé du programme

2. Définition d'une exception

Une exception est un événement anormal qui interrompt l'exécution normale d'une fonction.

Une exception = un objet lancé pendant l'exécution

Elle peut être :

- un type standard (int, string, bool)
- Un objet de type std::exception
- Une classe personnalisée

Exemple basique :

```
throw "Erreur grave !";
```

3. Lancer une exception : throw

Syntaxe :

```
throw expression;
```

Exemple :

```
if (x == 0)
   throw std::runtime_error("Division par zéro !");
```

Le mot-clé throw arrête la fonction actuelle et cherche un gestionnaire (catch).

4. Intercepter une exception : try / catch

Structure :

```
try {
```

```
    // code susceptible d'échouer
}
catch (typeErreur e) {
    // traitement de l'erreur
}
```

Exemple :

```
try {
    int x = 0;
    if (x == 0) throw std::runtime_error("Division par zéro");
    int y = 10 / x;
}
catch (std::runtime_error& e) {
    cout << "Erreur attrapée : " << e.what() << endl;
}
```

5. Interception multiple

On peut attraper plusieurs types d'erreurs différentes :

```
try {
    fonctionCritique();
}
catch (int e) {
    cout << "Erreur int : " << e << endl;
}
catch (std::string& msg) {
    cout << "Exception string : " << msg << endl;
}
catch (...) {
    cout << "Erreur inconnue !" << endl;
}
```

catch (...) attrape **n'importe quoi**.

6. Exceptions standard (std::exception)

Toutes les exceptions standards héritent de std::exception.

Exemples :

Classe	Description
std::runtime_error	erreur à l'exécution
std::out_of_range	index invalide
std::invalid_argument	argument incorrect
std::bad_alloc	échec de new
std::logic_error	erreur de logique

Syntaxe :

```
throw std::out_of_range("Index hors limites");
```

Accès au message :

```
catch (const std::exception& e) {
   cout << e.what();
}
```

7. Créer une exception personnalisée

C'est souvent utile en conception orientée objet.

Exemple :

```
class MonException : public std::exception {
private:
   std::string msg;

public:
   MonException(const std::string& m) : msg(m) {}

   const char* what() const noexcept override {
      return msg.c_str();
   }
};
```

Utilisation :

```
throw MonException("Erreur dans mon module");
```

Interception :

```
catch (const MonException& e) {
   cout << e.what();
}
```

8. Exceptions et RAII

En C++, la gestion de ressources repose sur :

- destructeurs

- objets temporaires
- smart pointers

Les destructeurs sont **toujours** appelés en cas d'exception.

Exemple :

```
{
   vector<int> v;
   throw 10;
}
// v est automatiquement détruit
```

9. Cas d'erreurs typiques

Situation	Bonne pratique
Division par zéro	throw std::runtime_error
Accès hors tableau	throw std::out_of_range
Fichier introuvable	throw std::runtime_error
Donnée invalide	throw std::invalid_argument
Allocation échoue	throw std::bad_alloc