

Les exceptions

- Les exceptions forment un mécanisme d'interruption à l'exécution du programme
 - elles permettent une **gestion simplifiée et plus sûre des erreurs**
- Principe
 - une fonction qui détecte une erreur "**lance**" une exception (un objet) : **cela nécessite donc de détecter l'erreur et de lancer l'exception**, ce qui n'est pas automatique
 - **un gestionnaire d'exception** adapté **est consulté** (il "attrappe" l'exception) **et un traitement approprié est exécuté**
 - lorsqu'une exception est lancée, **la pile d'appels de fonctions est remontée jusqu'à ce qu'elle soit rattrapée**

Les exceptions

> Try / catch

- Les exceptions sont “surveillées” dans un bloc particulier `try {}`
- Dans le bloc `try`, les exceptions sont “lancées” grâce à l’instruction `throw`
 - les exceptions sont typées : elles peuvent être d’un type de base (`int`, `double...`) ou un objet ou encore un pointeur sur un objet quelconque
 - ex : `long a; throw a;` ou même `throw Vector3(2.0);`
- Les exceptions sont “attrapées” et traitées dans un bloc particulier qui est protégé contre l’exception : `catch() {}`
 - un bloc `catch() {}` est écrit pour un type d’exception particulier

```
MyException e0("I/O Error"), e1("Memory allocation error");
```

```
try { // début de zone sous surveillance
    switch(...) {
        case 0: {
            ...
            throw e0;
        } break;
        case 1: {
            ...
            throw e1;
        }
        case 2: {
            ...
            throw 0x01;
        }
        ...
    }
}
```

on lance une exception particulière pour
dérouter l'exécution vers le gestionnaire
d'exception qui traitera le problème

les exceptions sont des instances
d'objets ou bien de types simples
(entiers, réels etc.)

**les exceptions peuvent être
directement lancées par throw
ou par n'importe quelle fonction
appelée à l'intérieur du bloc try**

```
catch (MyException& me) { // après try
    cout << "Exception attrapée = " << me.name() << endl;
}
catch (int ie) { // intercepter les exceptions de type int
    cout << "Exception avec entier = " << ie << endl;
}
catch (...) { // les ... font partie de la syntaxe !
    // toutes les autres exceptions
    cout << "Oups, exception non prévue..." << endl;
}
```

le (ou les) gestionnaire(s) d'exceptions
apparaissent dans les blocs catch() {}
après le bloc de surveillance try{}

Les exceptions

> Propagation d'une exception

- Une exception lancée "remonte" la pile d'appels des fonctions jusqu'à trouver un gestionnaire adapté

```
void func3(int ie) { throw ie; }  
void func2(int ie) { func3(ie); }  
void func(int ie) { if (!ie) func2(ie); else throw ie; }
```

```
int main() {  
    try {  
        func(0);  
        func(3);  
    }  
    catch (int ie) {  
        cout << "Exception levée avec id = " << ie << endl;  
    }  
    cout << "Fin" << endl;  
    return 0;  
}
```

cet appel n'est jamais effectué si une exception est lancée lors de l'exécution de l'appel précédent...



> Exception levée avec id = 0
> Fin

Les exceptions

> Relancer une exception

- Il est possible de relancer une exception (faire suivre), par exemple pour attraper une exception sans réellement effectuer le traitement nécessaire

```
try { ... }  
  
catch (exception_type e) {  
    throw f; // on relance une nouvelle exception à l'appelant  
}  
  
catch (exception_type e) {  
    throw; // on relance l'exception e à l'appelant  
}
```

Les exceptions

> Exception non prévue ?

- Si aucun gestionnaire adapté n'est trouvé pour une exception lors de la remontée de la pile de fonctions, la fonction `terminate()` est appelée : l'exécution du programme est interrompue

```
void func3(int ie) { throw 12.7; }  
void func2(int ie) { func3(ie); }  
void func(int ie) { if (!ie) func2(ie); else throw ie; }
```

```
int main() {  
    try {  
        func(0);  
        func(3);  
    }  
    catch (int ie) {  
        cout << "Exception levée avec id = " << ie << endl;  
    }  
    return 0;  
}
```

on lance une exception de type `double`
alors qu'on n'a prévu aucun gestionnaire
adapté dans le programme

> `terminate` called after throwing
an instance of 'double'
> Abort trap

la fonction `terminate` est appelée quand aucun
gestionnaire adapté n'est trouvé, elle quitte le
programme avec la f° C standard `abort()`