

Informatique Appliquée (IA)  
Programmation Objet Avec C++

Realisé Par:

**ABDELLAH ELMILOUDI**

**Groupe: A42**

Encadré par:

**PROF. ALI OUACHA**

**PROF. SARA LAGHMATI**

COMPTE RENDU

# **TP5**

# **Exceptions**

---

12/05/2025

LIEN VERS LE DÉPÔT GITHUB

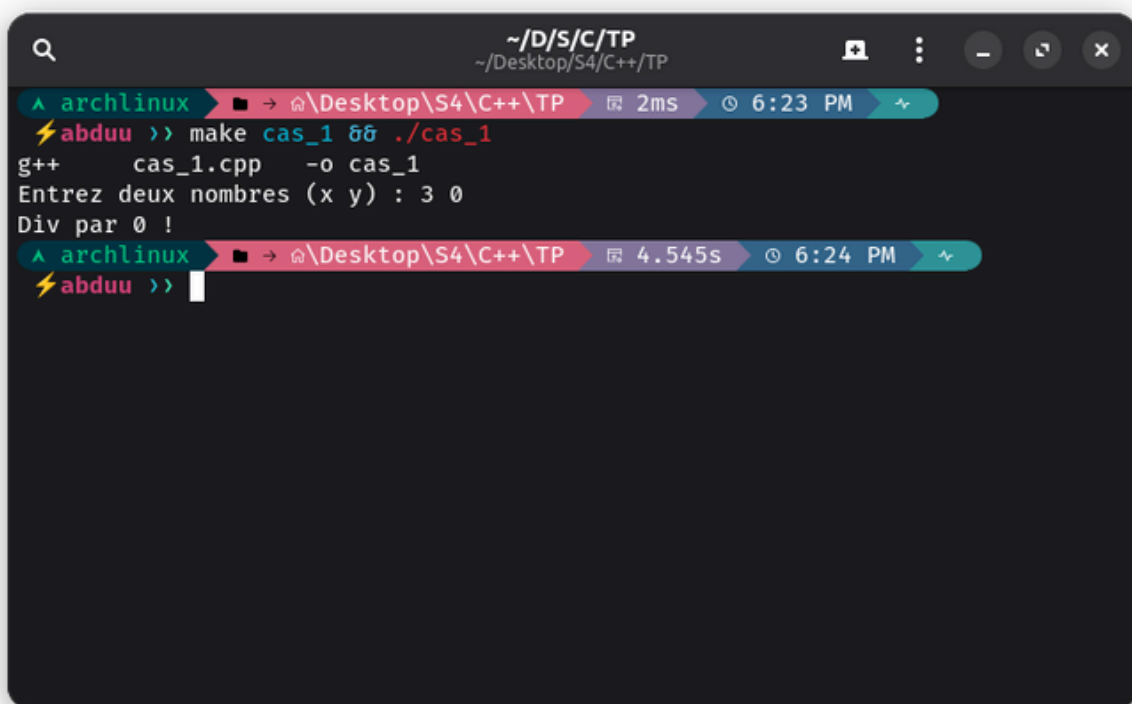
[https://github.com/ItsAbduu/OOP\\_Cpp\\_tp5-IA-2025](https://github.com/ItsAbduu/OOP_Cpp_tp5-IA-2025)

# Objectif

Ce TP a eu pour objectif d'explorer la gestion des exceptions en C++, d'abord à travers trois cas d'étude différents, puis en l'appliquant dans le cadre du projet développé lors du TP4.

## Exercice 1

- **Cas 1: Exception string**



```
~/D/S/C/TP
~/Desktop/S4/C++/TP
archlinux → 2ms 6:23 PM
abduu >> make cas_1 00 ./cas_1
g++      cas_1.cpp      -o cas_1
Entrez deux nombres (x y) : 3 0
Div par 0 !
archlinux → 4.545s 6:24 PM
abduu >>
```

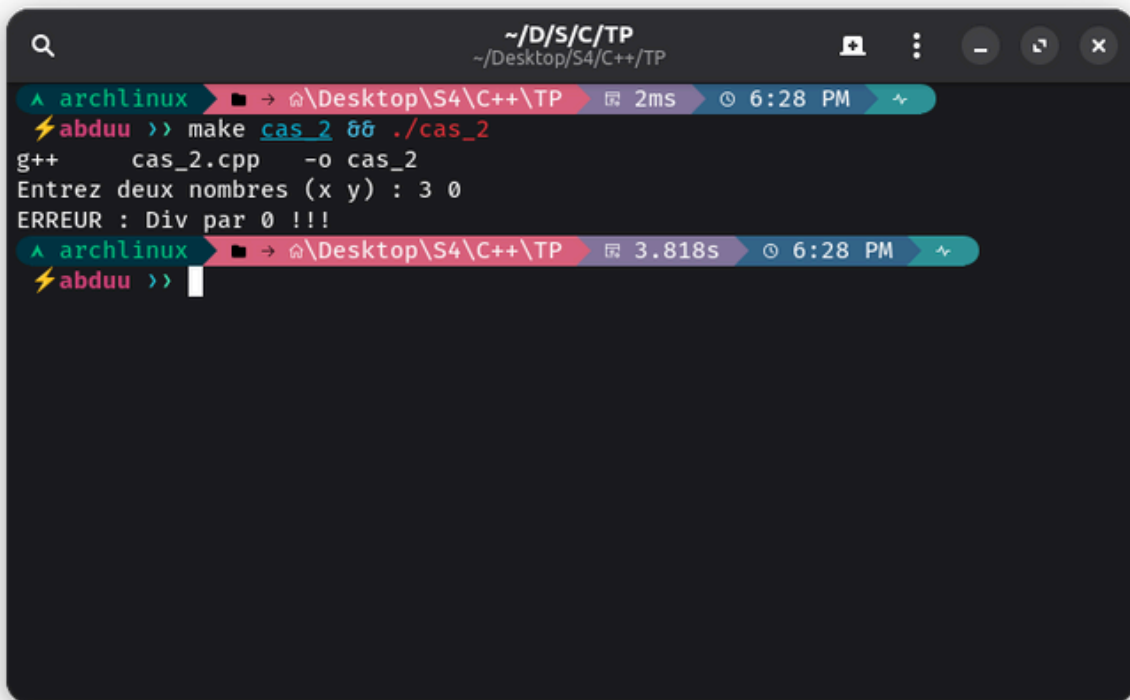
### Résultats avec les données 3 et 0

Lorsqu'on entre x=3 et y=0, le programme affiche: Div par 0 !

### Explication des instructions

- Ligne 6 (`throw string("Div par 0 !");`): Lance une exception de type string contenant un message d'erreur quand le diviseur est zéro.
- Ligne 16 (`try {...}`): Délimite le bloc de code susceptible de générer une exception.
- Ligne 20 (`catch (string const &s) {...}`): Capture l'exception de type string et récupère le message d'erreur pour l'afficher.

## • Cas 2: Exception avec classe



```
~/D/S/C/TP
~/Desktop/S4/C++/TP
archlinux → 2ms 6:28 PM
abduu >> make cas_2 00 ./cas_2
g++    cas_2.cpp    -o cas_2
Entrez deux nombres (x y) : 3 0
ERREUR : Div par 0 !!!
archlinux → 3.818s 6:28 PM
abduu >>
```

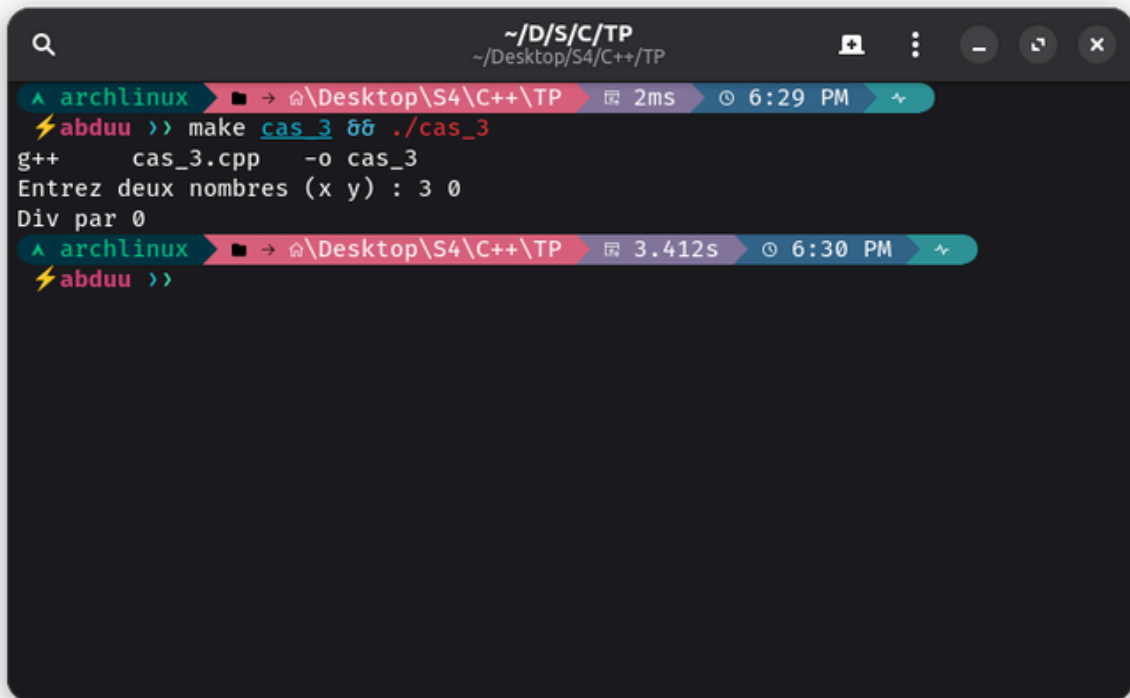
### Résultats avec les données 3 et 0

Lorsqu'on entre  $x=3$  et  $y=0$ , le programme affiche: ERREUR : Div par 0 !!!

### Explication des instructions

- Ligne 14 (`throw (me);`): Lance une exception de type `MonException`, classe créée spécifiquement pour la gestion des erreurs.
- Ligne 25 (`try {...}`): Délimite le bloc de code pouvant générer une exception.
- Ligne 30 (`catch (MonException & me) {...}`): Capture l'exception de type `MonException` et appelle la méthode `getMsg()` pour récupérer et afficher le message d'erreur.

- **Cas 3: Exception avec héritage de la classe exception**



```
~/D/S/C/TP
~/Desktop/S4/C++/TP
archlinux → Desktop/S4/C++/TP 2ms 6:29 PM
abduu >> make cas_3
g++ cas_3.cpp -o cas_3
Entrez deux nombres (x y) : 3 0
Div par 0
archlinux → Desktop/S4/C++/TP 3.412s 6:30 PM
abduu >>
```

### Résultats avec les données 3 et 0

Pour x=3 et y=0, le programme affiche: Div par 0

### Explication des instructions

- Ligne 25 (throw er;): Lance une exception de type Erreur, classe qui hérite de std::exception.
- Ligne 35 (try\_{...}): Délimite le bloc de code susceptible de générer une exception.
- Ligne 40 (catch(Erreur & e)\_{...}): Capture l'exception de type Erreur et appelle la méthode what() (héritée de std::exception mais surchargée) pour afficher le message d'erreur.

## • Synthèse

- **Cas 1 :** Utilise un type simple (comme une chaîne de caractères) pour lancer une exception.
- **Cas 2 :** Crée une classe d'exception personnalisée. Cela permet plus de contrôle (par exemple, ajouter des méthodes ou stocker des informations).
- **Cas 3 :** Crée une classe d'exception qui hérite de `std::exception`. Elle s'intègre bien avec la bibliothèque standard et bénéficie de ses méthodes, comme `what()`.

## Exercice 2

### Implémentation requise

#### Classe OperationException

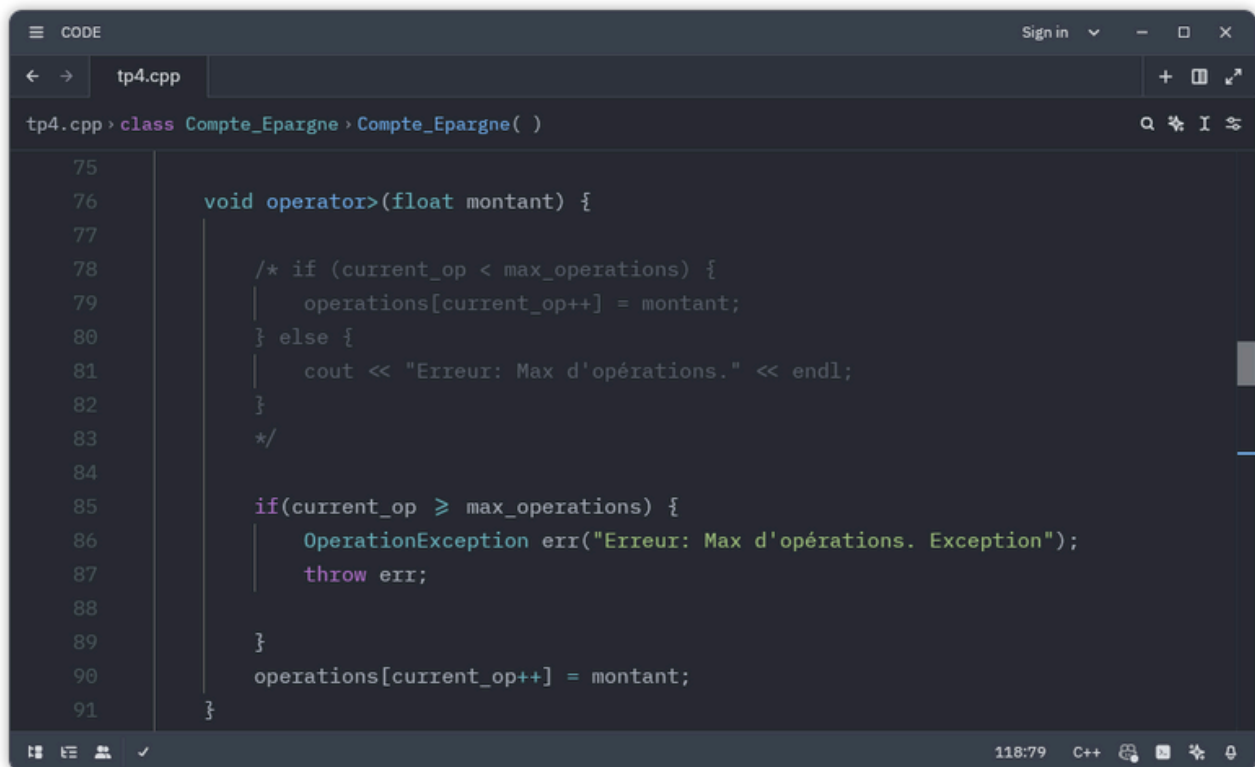
- Classe personnalisée héritant de `std::exception`.
- Gère deux types d'erreurs: accès hors limites et dépassement de capacité.



```
CODE
tp4.cpp
tp4.cpp > class Compte_Epargne > Compte_Epargne( )
4   using namespace std;
5
6   class OperationException: public exception {
7   private:
8       string msg;
9   public:
10      OperationException(string const& s="") throw() : msg(s) {}
11
12
13      virtual const char* what() const throw() {
14          return msg.c_str();
15      }
16
17      ~OperationException() throw() {}
18
19  };
20
```

## Modification de Compte Depot

- Ajout de l'exception lors de l'ajout d'une opération dans un tableau plein.



```
CODE tp4.cpp
tp4.cpp > class Compte_Epargne > Compte_Epargne( )

75
76 void operator>(float montant) {
77
78     /* if (current_op < max_operations) {
79         operations[current_op++] = montant;
80     } else {
81         cout << "Erreur: Max d'opérations." << endl;
82     }
83     */
84
85     if(current_op ≥ max_operations) {
86         OperationException err("Erreur: Max d'opérations. Exception");
87         throw err;
88     }
89
90     operations[current_op++] = montant;
91 }
```

- Ajout de l'exception lors de l'accès à un indice invalide via l'opérateur [].



```
CODE tp4.cpp
tp4.cpp > class Compte_Epargne > Compte_Epargne( )

92
93 float operator[](int idx) {
94     /* if (idx ≥ 0 && idx < current_op) {
95         return operations[idx];
96     } else {
97         cout << "Erreur: invalide idx." << endl;
98         return 0.0f;
99     } */
100
101     if(idx < 0 || idx ≥ current_op) {
102         OperationException err_idx("Erreur: invalide idx. Exception");
103         throw err_idx;
104     }
105
106     return operations[idx];
107 }
108 };
```

# Gestion des exceptions dans main

```
CODE
tp4.cpp
tp4.cpp > int main()

159     try {
160         Compte_Epargne ce3(201, 5000, 0.05, -3);
161     } catch (OperationException &err_d) {
162         cerr << "\033[31m" << err_d.what() << "\033[0m" << endl;
163     }

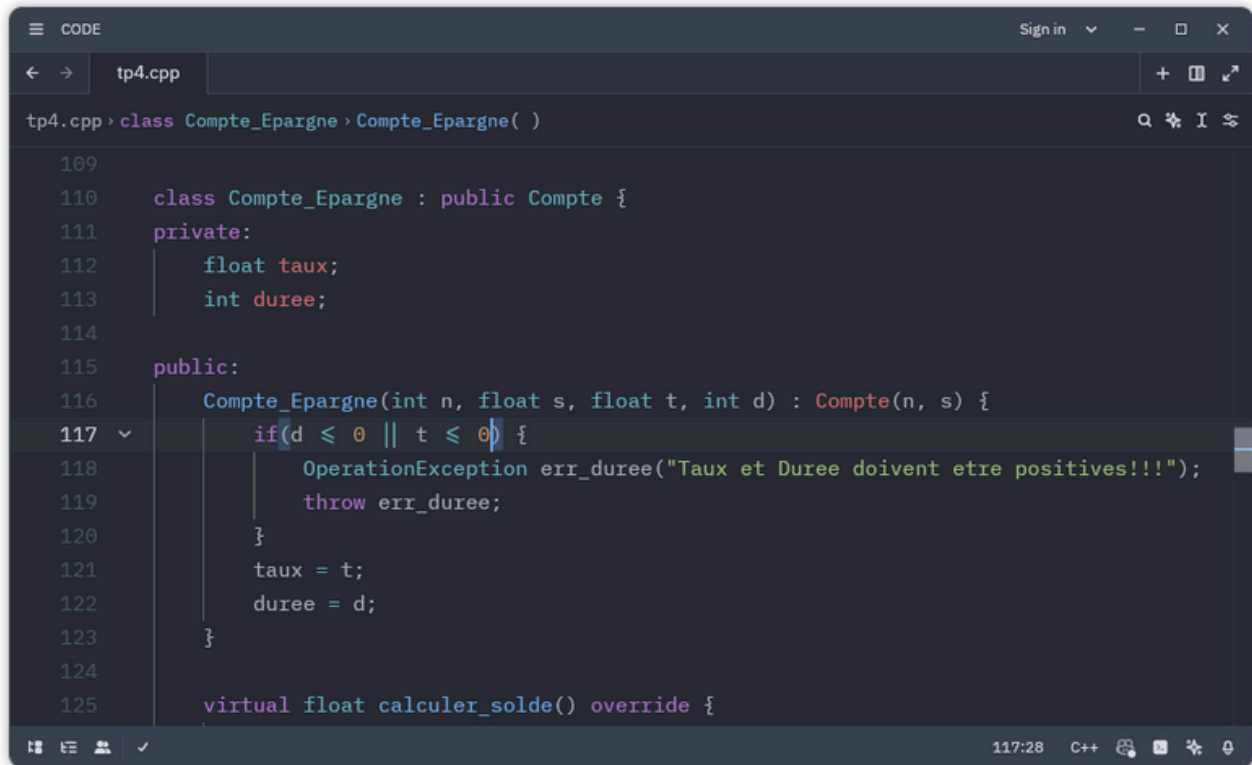
164
165     try {
166         cd1 > 500;
167         cd1 > -200;
168         cd1 > 300;
169         cd1 > 700;
170         cd1 > 800;
171         //cd1 > 100;
172     } catch (OperationException &e_idx) {
173         cerr << "\033[31m" << e_idx.what() << "\033[0m" << endl;
174     }
175
```

```
CODE
tp4.cpp
tp4.cpp > int main()

190
191     try {
192         cout << cd1[9] << endl;
193     } catch (OperationException &e) {
194         cerr << "\033[31m" << e.what() << "\033[0m" << endl;
195     }

196
197     cout << "\033[33m";
198     cout << "\n---- Solde moyen ----" << "\033[0m" << endl;
199     cout << "Solde moyen: " << "\033[1;32m"
200     << calculer_solde_moyen(comptes, 4) << endl;
201     cout << "\033[0m";
202
203     cout << "==== Fin du programme ==== \n" << endl;
204
205     return 0;
206
```

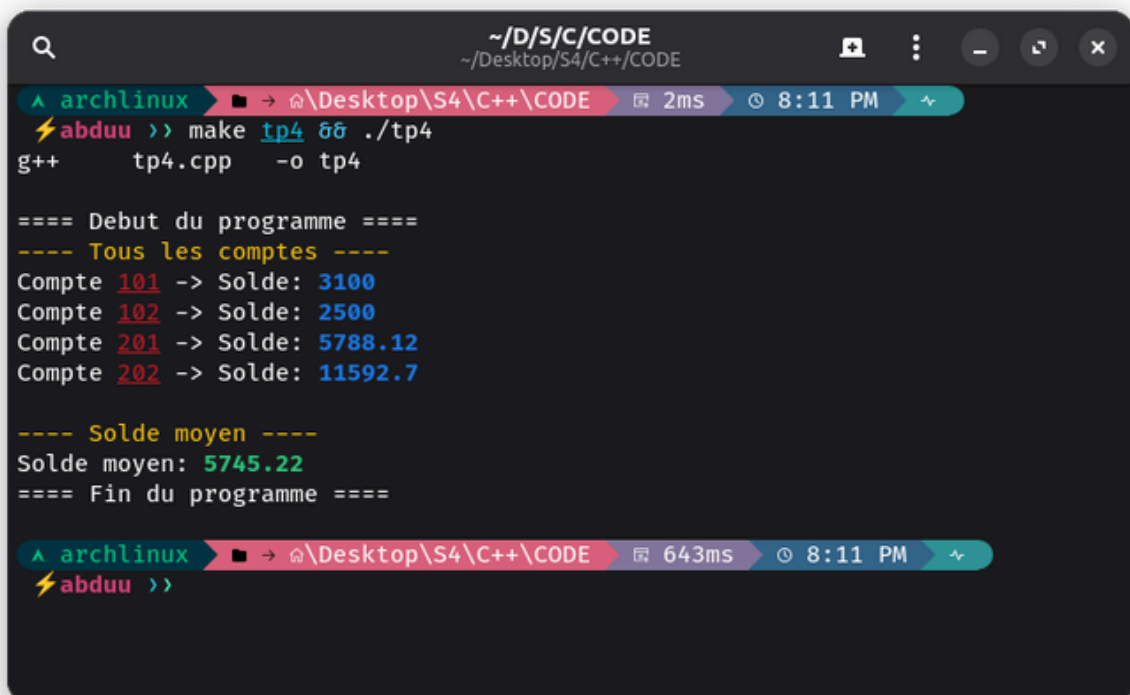
# Vérification dans Compte\_Epargne



```
CODE
tp4.cpp
tp4.cpp > class Compte_Epargne > Compte_Epargne( )
109
110 class Compte_Epargne : public Compte {
111 private:
112     float taux;
113     int duree;
114
115 public:
116     Compte_Epargne(int n, float s, float t, int d) : Compte(n, s) {
117         if(d <= 0 || t <= 0) {
118             OperationException err_duree("Taux et Duree doivent etre positives!!!");
119             throw err_duree;
120         }
121         taux = t;
122         duree = d;
123     }
124
125     virtual float calculer_solde() override {
```

## Résultats

Voici le programme s'exécutant sans erreurs.



```
~/D/S/C/CODE
~/Desktop/S4/C++/CODE 2ms 8:11 PM
archlinux >> make tp4 66 ./tp4
g++ tp4.cpp -o tp4

==== Debut du programme ====
---- Tous les comptes ----
Compte 101 -> Solde: 3100
Compte 102 -> Solde: 2500
Compte 201 -> Solde: 5788.12
Compte 202 -> Solde: 11592.7

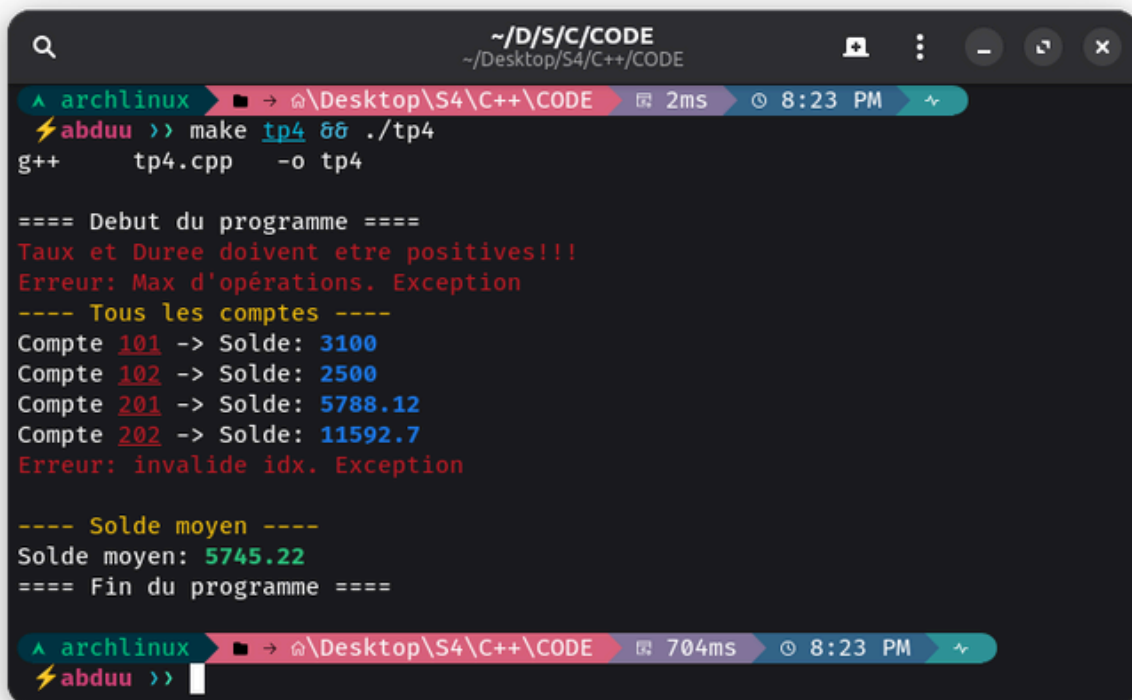
---- Solde moyen ----
Solde moyen: 5745.22
==== Fin du programme ====

archlinux >> 643ms 8:11 PM
```



Voici l'exécution du programme après l'introduction des erreurs prévues :

1. Création d'un Compte\_Epargne avec une durée négative (-3).
2. Tentative d'effectuer une sixième opération sur cd1, alors que seulement 5 sont autorisées : `cd1 > 100`.
3. Tentative d'accès à l'opération numéro 9 sur cd1, alors que seulement 5 opérations sont présentes : `cd1[9]`.



```
~/D/S/C/CODE
~/Desktop/S4/C++/CODE 2ms 8:23 PM
abduu >> make tp4 55 ./tp4
g++ tp4.cpp -o tp4

==== Debut du programme ====
Taux et Duree doivent etre positives!!!
Erreur: Max d'opérations. Exception
---- Tous les comptes ----
Compte 101 -> Solde: 3100
Compte 102 -> Solde: 2500
Compte 201 -> Solde: 5788.12
Compte 202 -> Solde: 11592.7
Erreur: invalide idx. Exception

---- Solde moyen ----
Solde moyen: 5745.22
==== Fin du programme ====

archlinux >> ~/Desktop/S4/C++/CODE 704ms 8:23 PM
abduu >>
```

## Conclusion

Ce TP a permis d'explorer en profondeur la gestion des exceptions en C++, d'abord avec des exemples simples puis en l'appliquant à un cas pratique de gestion bancaire. Les différentes approches pour créer et gérer des exceptions ont été comparées, montrant l'intérêt d'utiliser l'héritage de `std::exception` pour une meilleure intégration avec la bibliothèque standard.