

République du Cameroun

Paix – Travail - Patrie

Ministère des travaux Publics

Ministère de l'Enseignement

Supérieur

Ecole Nationale Supérieure des Travaux



THE
ICT
UNIVERSITY

Republic of Cameroon

Peace – Work - Fatherland

Ministry of Public Works

Ministry of Higher Education

National Advanced School of Public Works

CAHIER DES CHARGES ASDD II

PROJET :

**VÉRIFICATEUR DE
PARENTHESES ÉQUILIBRÉES**

Noms des exposants:

- **BOUCHEL PONKOUA EXCEL PAGUIEL**
- **DJUIKEM TANGUE YOLAINE SHEILLA**
- **FOSSE KOAGNE DAVID RYAN**

Sous la supervision de :

Ing ATANGANA Guy Martial

SOMMAIRE

INTRODUCTION

I. DESCRIPTION GENERALE.....

II. EXIGENCES

III. DEPLOIEMENT DU SYSTEME.....

IV. AUTRES EXIGENCES DU PROJET

CONCLUSION

INTRODUCTION

Dans le cadre du développement d'outils informatiques facilitant la vérification de la syntaxe, ce projet vise à concevoir un vérificateur de parenthèses équilibrées. Les expressions contenant des parenthèses sont omniprésentes en programmation, en mathématiques et dans divers formats de données, et une mauvaise structuration peut entraîner des erreurs bloquantes. L'objectif principal est de fournir une solution capable d'analyser automatiquement une expression donnée afin de déterminer si les parenthèses sont correctement équilibrées. Ce vérificateur pourra être utilisé par les étudiants, les développeurs et les professionnels manipulant des structures nécessitant un respect rigoureux des règles syntaxiques. Ce document présente le vérificateur des parenthèses équilibrées, ses objectifs et son utilité pour les développeurs et étudiants. Il détaille son environnement d'exploitation, ses fonctionnalités clés et son intégration avec d'autres systèmes. Les besoins utilisateurs et les objectifs stratégiques sont aussi abordés, ainsi que les méthodes de feedback.

Quelques acronymes et définitions:

Acronymes :

API : Application Programming Interface.

IDE : Integrated Development Environment.

CI/CD : Continuous Integration/ Continuous Deployment.

Définitions :

Git : Système de gestion de versions distribuée qui permet aux développeurs de suivre l'évolution de leur code et de collaborer efficacement.

Feedback : (ou rétroaction) est une réponse où un retour d'information donnée à une personne ou un système pour l'aider à s'améliorer.

Crash : Se produit lorsqu'un programme ou un système informatique s'arrête brusquement en raison d'une erreur, souvent sans possibilité de récupération immédiate.

Pipeline : Processus qui permet de traiter des données ou des tâches en plusieurs étapes successives.

Workflow : (flux de travail) c'est une suite de tâches et d'étapes organisées pour atteindre un objectif précis.

Parentheses mal imbriquées : Mauvaise organisation des parenthèses dans une expression, ce qui peut rendre le code invalide ou le calcul incorrect.

PyCharm : Environnement de développement intégré (IDE) conçu pour le langage Python.

Docker : Outil permettant de créer, déployer et exécuter des applications dans des « conteneurs ».

Vérificateur de parenthèses équilibrées : Outil essentiel pour assurer la validité des expressions mathématiques, des codes sources et autres formats nécessitant une syntaxe correcte.

I. DESCRIPTION GENERALE

1. Contexte d'utilisation

Ce produit sera utilisé principalement dans des environnements où la vérification de la syntaxe est cruciale, comme :

- Développement logiciel : les programmeurs et ingénieurs peuvent l'intégrer dans leur workflow* pour s'assurer que leur code respecte les bonnes règles de syntaxe.
- Education et formation : les étudiants en programmation pourront l'utiliser pour tester leurs exercices et apprendre les bonnes pratiques
- Automatisation des tests : il peut être intégré dans des pipelines¹ CI/CD pour vérifier automatiquement la validité des expressions avant déploiement.

2. Environnement d'exploitation

Le vérificateur pourra fonctionner dans plusieurs environnements :

- Application Web : Accessible via une interface utilisateur simple pour une analyse rapide.
- Extension d'éditeur de code : compatible avec des IDE comme VS code, permettant une vérification en temps réel
- Application mobile : Une version légère pour vérifier la syntaxe directement depuis un téléphone.

3. Relation avec d'autres systèmes :

- ✚ Des compilateurs et interpréteurs peuvent être utilisés en complément pour éviter des erreurs de syntaxe avant exécution du code.
- ✚ Base des données : peut stocker l'historique des vérifications pour analyse
- ✚ Outils de documentation : peut générer rapports détaillés sur les erreurs et recommandations.

4. Besoins des utilisateurs finaux

Les utilisateurs potentiels de cet outil recherchent :

- Fiabilité et précision : un vérificateur qui identifie correctement les erreurs des parenthèses pour éviter les bugs.
- Simplicité d'utilisation : une interface intuitive permettant une analyse rapide sans complexité excessive.
- Feedback détaillée : des images claires et compréhensibles expliquant les erreurs détectées et comment les corriger.

5. Objectifs commerciaux et stratégiques

Les objectifs clés visés sont :

- Améliorer la productivité des développeurs en évitant des erreurs de syntaxe qui pourraient entraîner des dysfonctionnements.
- Réduire le temps de débogage en fournissant des vérifications automatisées.
- Créer un produit évolutif qui pourra être enrichi avec d'autres vérifications syntaxiques à l'avenir.

6. Scenarios d'utilisation de haut niveau

Développeur en entreprise

Intègre l'outil dans un pipeline CI/CD pour vérifier automatiquement les erreurs syntaxiques avant le déploiement d'une application

Etudiant en informatique

Utilise l'application pour tester des exercices de programmation et apprendre à éviter les erreurs de parenthèses.

Editeurs de texte avancé :

Un IDE intègre l'outil pour aider les utilisateurs à corriger leur syntaxe en temps réel.

Application Web éducative :

Une plateforme en programmation inclut cet outil pour aider les débutants à comprendre la gestion des parenthèses.

7. Contraintes

a. budgétaire

- ❖ Coût des développements (IDE, Serveurs, services cloud)
- ❖ Budget limité pour les tests et la validation du produit
- ❖ Ressources humaines nécessaires (développeurs, testeurs experts en sécurité).

b. Contrainte temporelles

- ❖ Délais de livraison du produit
- ❖ Temps nécessaire pour les tests et la correction des bugs
- ❖ Eventuelles modifications en cours de développement qui rallongent le projet.

8. Caractéristiques du produit

Voici quelques caractéristiques essentielles de notre projet :

Gestion des types de parenthèses : Il doit prendre en compte différents types de parenthèses comme (), {} et [].

Utilisation d'une pile (structure de données) : Une pile est souvent utilisée pour suivre l'ouverture et la fermeture des parenthèses de manière ordonnée.

Détection d'erreurs : Le programme doit pouvoir identifier si une parenthèse est manquante, mal placée ou si l'ordre d'imbrication est incorrect.

9. Hypothèses et dépendances

a) Hypothèses :

Les entrées utilisateurs sont bien formatées : on suppose que les données fournies respectent les règles syntaxiques attendues.

Le langage de programmation choisi est adapté : l'algorithme sera implémentée dans un langage offrant une gestion efficace des structures de données (ex : Python, Java, C++).

Les erreurs seront gérées correctement : le système doit identifier et signaler les erreurs de parenthesage sans toutefois provoquer de crash.

b) Dépendances externes :

Dépendances technologiques : compatibilité avec les systèmes d'exploitation et les environnements d'exécution

Dépendances des ressources :

- disponibilité des développeurs et des experts en validation logicielle.
- accès aux outils de tests et aux plateformes de déploiement.

Dépendances budgétaires :

- coût des outils et des infrastructures nécessaires.
- budget alloué aux tests et à la maintenance.

II. EXIGENCES

Voici une analyse détaillée des exigences spécifiques, fonctionnelles et non fonctionnelles de notre projet de vérification des parenthèses équilibrées.




1. Exigences spécifiques

Ces exigences définissent les besoins précis du projet en fonction de son usage et de ses objectifs.

Objectifs du projet :

- ✓ Fournir un outil fiable permettant de vérifier l'équilibre des parenthèses dans un texte ou un code source.
- ✓ Offrir une solution rapide et accessible aux développeurs et étudiants en informatique.
- ✓ Intégrer des mécanismes de retour d'information détaillé pour aider les utilisateurs à corriger leurs erreurs.

Public cible :

-  Développeurs : Utilisation dans des environnements de programmation pour éviter les erreurs syntaxiques.
-  Étudiants : Apprentissage et correction de syntaxe dans les exercices de programmation.
-  Automatisation : Intégration dans les outils CI/CD pour une vérification automatique avant déploiement.

2. Exigences fonctionnelles

Fonctionnalités précises que notre système offre :

- ✓ Vérification des parenthèses équilibrées
- Le système doit analyser une chaîne de caractères contenant des parenthèses ((), {}, []).

- Il doit détecter les erreurs telles que des parenthèses ouvertes sans fermeture ou des parenthèses mal imbriquées.
- Il doit afficher un message d'erreur clair pour guider l'utilisateur vers la correction.

✓ Feedback et gestion des erreurs

- Lorsque des erreurs sont détectées, le système doit fournir un message détaillé indiquant l'emplacement et la nature de l'erreur.
- Un historique des vérifications doit être disponible pour suivre les erreurs passées et les corrections apportées.

✓ Formats de saisie et compatibilité

- Le système doit accepter différents formats de données :

■ Texte brut : Entrée manuelle par l'utilisateur.

📄 Code source : Compatible avec plusieurs langages de programmation (Python, JavaScript, C++, etc.).

📁 Fichiers : Chargement et analyse de fichiers .txt ou .code.

✓ Intégration et accessibilité

- Le système doit être accessible via :

🌐 Interface web : Application en ligne pour une vérification simple et rapide.

✎ Extension pour IDE : Intégration dans des éditeurs de code comme VS Code ou PyCharm.

🔗 API : Une interface permettant aux autres applications d'appeler la fonctionnalité de vérification.

🏗️ CI/CD : Automatisation pour vérifier les erreurs avant le déploiement.

3. Exigences non fonctionnelles

Voici quelques exigences non fonctionnelles de notre système.

✓ Performance et rapidité

- L'analyse des parenthèses doit être instantanée, même sur de grandes quantités de texte.
- Le temps de réponse doit être inférieur à 200ms pour une bonne fluidité.

✓ Sécurité et fiabilité

- Le système doit garantir l'intégrité des données sans altérer le texte fourni par l'utilisateur.
- Les historiques et résultats doivent être sécurisés contre toute modification non autorisée.

✓ Scalabilité et évolutivité

- L'outil doit être extensible, avec la possibilité d'ajouter de nouvelles règles de vérification (ex: guillemets "", ").
- Il doit pouvoir s'adapter à une forte demande en cas d'utilisation massive.

III. DEPLOIEMENT DU SYSTEME

Le processus de déploiement du système dans l'environnement de production suit une approche méthodique garantissant stabilité, performance et maintenabilité. Voici les principales étapes :

1. Préparation de l'Environnement de Production

Avant le déploiement effectif, il est essentiel de s'assurer que l'environnement cible est correctement configuré :

- Infrastructure : Serveur dédié ou cloud (AWS, Azure, ou autre).
- Système d'exploitation : Linux ou Windows, selon les exigences.

- Dépendances : Installation des bibliothèques et Framework nécessaires (Python, Node.js, etc.).
- Base de données (si requise) : Configuration et optimisation des accès aux données.

2. Automatisation du Déploiement

L'intégration et le déploiement continu (CI/CD) sont mis en place pour assurer une mise en production fluide :

- Git et gestion de versions : Le code source est maintenu sur un dépôt Git.
- Pipeline CI/CD : Automatisation via Docker pour garantir un déploiement cohérent.
- Tests unitaires et fonctionnels : Validation du bon fonctionnement avant mise en production.

3. Déploiement Effectif

Une fois l'environnement prêt et les tests validés, le déploiement suit ces étapes :

- Mise en production : Déploiement progressif via une mise à jour contrôlée.
- Monitoring et logs : Implémentation d'un système de surveillance pour détecter d'éventuelles anomalies.
- Optimisation des performances : Ajustement des configurations pour un fonctionnement fluide.

4. Maintenance et Support

Après le déploiement, un suivi est assuré pour garantir le bon fonctionnement du système :

- Gestion des erreurs et bugs : Corrections rapides via mises à jour.
- Capabilité : Possibilité d'adaptation aux besoins évolutifs.
- Feedback utilisateurs : Prise en compte des retours pour améliorations continues.

IV. AUTRES EXIGENCES DU PROJET :

1) Exigences de Formation

Pour garantir une utilisation efficace et une administration fluide du système, voici les besoins en formation pour les différents acteurs :

Formation des utilisateurs

1. Introduction à l'outil : Comprendre son utilité, son fonctionnement et ses limites.
2. Utilisation pratique : Manipulation de l'interface, saisie de données et interprétation des résultats.
3. Types de parenthèses prises en charge : Parenthèses (), crochets [], accolades {}.
4. Cas d'erreurs courantes : Parenthèses mal fermées, déséquilibrées ou mal imbriquées.
5. Exploitation des fonctionnalités avancées (si disponibles) : Options de correction automatique ou suggestions d'amélioration.

Formation des administrateurs

1. Installation et configuration du système : Paramétrage selon les besoins spécifiques des utilisateurs.
2. Gestion des accès et des permissions : Contrôle des niveaux d'accès et de la sécurité des données.
3. Maintenance et dépannage : Diagnostic et résolution des problèmes techniques.
4. Optimisation des performances : Amélioration de la rapidité et de l'efficacité du vérificateur.
5. Mises à jour et évolutivité : Intégration de nouvelles fonctionnalités et adaptation aux besoins futurs.

Une formation adaptée permet aux utilisateurs d'exploiter pleinement l'outil et aux administrateurs de garantir un fonctionnement optimal.

2) Exigences de Documentation

Voici une structure pour chaque document :

1. Manuel Utilisateurs

Introduction

Le vérificateur de parenthèses équilibrées est un outil permettant de vérifier si une expression contient des parenthèses correctement ouvertes et fermées. Il est utile pour la validation de code source, d'expressions mathématiques et bien d'autres applications.

Installation

a) Prérequis

- Un ordinateur avec un système d'exploitation Windows, macOS ou Linux.
- Un interpréteur Python (si l'outil est basé sur Python) ou tout autre langage utilisé.
- Éventuellement, un éditeur de texte ou un IDE (comme VS Code, PyCharm).

b) Procédure d'installation

1. Téléchargement : Accédez au dépôt GitHub ou au site officiel pour télécharger l'outil.
2. Installation :
 - Si l'outil est en Python : exécutez `pip install nom_du_programme`
 - Si c'est un exécutable, suivez les instructions d'installation.
3. Lancement : Ouvrez le programme et commencez à l'utiliser.

Utilisation

a) Interface utilisateur

L'interface peut être en ligne de commande ou graphique. Selon le cas :

- En ligne de commande : exécutez `python verifier.py` et entrez l'expression.
- Interface graphique : cliquez sur le champ de saisie, entrez l'expression et appuyez sur "Vérifier".

b) Exemples d'utilisation

Expression	Résultat
-----	-----
$(a + b) * (c - d)$	✓ Parenthèses équilibrées
$((a + b)$	✗ Parenthèses non équilibrées
$\{[a + b] * (c - d)\}$	✓ Parenthèses équilibrées

c) Messages d'erreur

Si les parenthèses ne sont pas équilibrées, le programme affiche un message indiquant où se trouve l'erreur.

Dépannage

Si le programme ne fonctionne pas correctement :

- Vérifiez que tous les fichiers nécessaires sont présents.
- Assurez-vous que votre environnement Python est à jour.
- Consultez la documentation ou le support technique.

Contact et support

Pour toute question ou problème :

- Consultez la documentation officielle.
- Contactez le support via email ou sur GitHub

2. Guide d'installation

Prérequis

Avant de commencer, assurez-vous d'avoir :

- Un ordinateur sous Windows, macOS ou Linux.
- Un interpréteur adapté au langage utilisé.
- Un éditeur de texte ou un IDE (Visual Studio Code, PyCharm, etc.).
- Git (le projet est récupéré depuis un dépôt GitHub).

Téléchargement du projet

Si le projet est hébergé sur GitHub :

1. Ouvrir un terminal ou une invite de commandes.
2. Cloner le dépôt avec la commande :

```
bash  
git clone https://github.com/nom_du_repositoire.git
```

3. Accéder au dossier du projet :

```
bash  
cd nom_du_repositoire
```

Si le projet est sous forme d'archive ZIP :

1. Télécharge le fichier ZIP.
2. Extrayez-le dans un dossier de ton choix.

Installation des dépendances

Si le projet utilise des bibliothèques externes, les installer avec :

```
bash  
pip install -r requirements.txt
```

Exécution du programme

Selon l'interface disponible :

- Mode ligne de commande :

```
bash  
python verifier.py
```

Entrez une expression à analyser et obtenez le résultat.

- Mode interface graphique :

1. Lance l'application en exécutant l'exécutable fourni.
2. Entrez votre expression dans le champ prévu et appuyez sur "Vérifier".

Vérification du bon fonctionnement

Teste le programme avec plusieurs expressions :

- $(a + b) * (c - d) \rightarrow \checkmark$ Parenthèses équilibrées.
- $((a + b) \rightarrow \times$ Parenthèses non équilibrées.

3. Documentation technique

- API et intégrations : Interfaces pour connecter le système à d'autres applications.
- Maintenance et évolutivité : Stratégies de mise à jour et d'amélioration.
- Sécurité : Protocoles de protection des données et des utilisateurs.

Règles de gestion

- Les parenthèses prises en charge sont : $()$, $[]$, $\{\}$.
- Toute parenthèse fermante doit correspondre à la dernière ouvrante.
- Le contenu non-parenthétique est ignoré.

Jeux de tests :

Cas de tests	Expression	Résultat attendu
Test vide	“ “	Vrai
Simple équilibre	“([)]”	Vrai
Manque de fermeture	“((()”	Faux
Ordre incorrect	“([)]”	Faux
Parenthèses imbriquées	“({[]})”	Vrai

3) Exigences de Maintenance

La maintenance future de notre **vérificateur de parenthèses équilibrées** est essentielle pour garantir son bon fonctionnement, sa fiabilité et son évolutivité. Voici les principales spécifications pris en compte :

Maintenance préventive

Cette maintenance permet d'anticiper les éventuels problèmes avant qu'ils ne surviennent.

- Surveillance des performances : Analyse régulière des temps de traitement et optimisation des algorithmes.
- Mises à jour du logiciel : Ajout de nouvelles fonctionnalités et correction des bugs.
- Compatibilité avec les nouvelles versions des systèmes* : Vérification de la compatibilité avec les dernières versions de Windows, Linux ou Mac.

Maintenance corrective

Elle intervient en cas de problème détecté par les utilisateurs ou administrateurs.

- Correction des erreurs signalées : Analyse et résolution des bugs identifiés.
- Amélioration de la gestion des erreurs : Affichage plus précis des messages d'erreur et suggestions de correction.
- Tests de non-régression : Vérification que les corrections n'introduisent pas de nouveaux problèmes.

Maintenance adaptative

Elle permet d'adapter le système aux évolutions technologiques et aux nouveaux besoins des utilisateurs.

- Prise en charge de nouveaux langages ou formats : Amélioration du support pour divers langages de programmation ou formats de texte.
- Optimisation du moteur d'analyse : Réduction du temps de traitement et amélioration des algorithmes de vérification.
- Ajout de fonctionnalités avancées : Possibilité d'intégration avec des éditeurs de code ou plateformes tierces.

Maintenance évolutive

Elle vise à améliorer continuellement le système en fonction des retours des utilisateurs et des avancées technologiques.

- Automatisation de la correction : Proposition de solutions intelligentes pour ajuster les erreurs détectées.
- Interface utilisateur améliorée : Optimisation de l'ergonomie et de l'expérience utilisateur.
- Sécurité renforcée : Protection contre les intrusions et sécurisation des données traitées.

Documentation et suivi

Un suivi rigoureux est nécessaire pour garantir une maintenance efficace.

- Base de connaissances : Documentation mise à jour sur les fonctionnalités et corrections apportées.
- Système de gestion des bugs : Plateforme où les utilisateurs peuvent signaler des problèmes et proposer des améliorations.
- Évaluation périodique : Tests de performance et enquête de satisfaction auprès des utilisateurs.

Conclusion:

En résumé, le projet de vérificateur de parenthèse équilibré est un outil utile qui peut aider les utilisateurs à vérifier les parenthèses dans les expressions de manière efficace et précise. L'outil offre plusieurs avantages, notamment le gain de temps, la réduction des erreurs et l'amélioration de la productivité. Les perspectives pour l'amélioration et l'extension de l'outil sont nombreuses et peuvent inclure l'intégration avec d'autres outils, l'extension à d'autres types de parenthèses et l'amélioration de l'interface utilisateur.