Séquence 2 : évaluation d'expressions arithmétiques

Compte Rendu

Nom :TAVAKOL

Prénom :Zahra

Numéro étudiant :22300070

Choix d'Implémentation

a) Tokenisation

- La fonction stringToTokenQueue(const char* expression) transforme une chaîne de caractères en file de tokens.
- Les caractères sont analysés un à un. Les espaces sont ignorés. Les nombres et symboles sont identifiés.
- Les tokens sont directement insérés dans une file, sans allocations superflues.

b) Algorithme de Shunting Yard

- La fonction shutingYard() prend une file en notation infixe et renvoie une file postfixée.
- Utilisation d'une pile d'opérateurs et d'une file de sortie.
- Les priorités et associativités des opérateurs sont correctement gérées.
- Les parenthèses sont traitées rigoureusement avec détection d'erreurs.

c) Évaluation Postfixe

- Une pile est utilisée pour stocker les opérandes.
- Lorsqu'un opérateur est rencontré, deux opérandes sont extraits, et un nouveau token contenant le résultat est créé.
- Le résultat est récupéré en fin de file.

Problèmes rencontrés et solutions

Appel de stack_top() sur pile vide

Erreur : segmentation fault lors de l'évaluation de lignes vides.
Solution : ajout de vérification de ligne vide dans computeExpressions() et vérification de pile non vide avant tout accès.

Fuite mémoire

Erreur : certains tokens ou structures n'étaient pas libérés.
Solution : suppression explicite de chaque token et structure (queues, stacks) à la fin de chaque étape.

Lecture de lignes vides avec getline()

Erreur : lors de la lecture du fichier d'entrée, getline() retourne parfois des lignes contenant uniquement des espaces ou retours à la ligne, provoquant un traitement inutile voire une erreur d'évaluation.

Solution : ajout d'un bloc dans computeExpressions() pour détecter et ignorer les lignes blanches ou vides :

```
bool is_blank = true;

for (size_t i = 0; buffer[i] != '\0'; i++) {

  if (buffer[i] != ' ' && buffer[i] != '\n' && buffer[i] != '\t') {

    is_blank = false;

    break;

  }

}

if (is_blank) continue;
```

Cette condition garantit que seules les lignes contenant des données valides sont traitées.

Conclusion

Le programme est fonctionnel, respecte les spécifications et gère les cas classiques et les erreurs. Il est optimisé pour éviter les redondances et fuites mémoire. Les structures de données sont utilisées de manière efficace.

Des extensions sont possibles (gestion des opérateurs unaires, constantes comme pi, etc.), mais non implémentées ici pour rester concentré sur l'essentiel du TP.