P31 - TP noté

Consignes

- Vous avez 1h30 pour réaliser le TP.
- Vous avez le droit d'accéder à vos TP précédents.
- Vous avez le droit d'utiliser un crayon et une feuille de papier.
- Le TP est à réaliser **seul** et tous les outils de communication sont **interdits**.
- Vous **n'avez pas le droit** à un navigateur web.
- Vous **n'avez pas le droit** d'utiliser des d'outils de génération de code tels que `Copilot` ou `ChatGPT`.
- Le rendu se fait par l'intermédiaire de votre dépot `git` au sein du dossier `tp_note`.

> La partie 1 sur les *trie (arbre préfixe)* est la plus importante.
> La partie 2 sur la vérification d'expressions bien parenthésées, est indépendante de la partie 1.

> Quelques tests sont déjà présents dans le fichier `main.c`.
> Libre à vous d'en écrire d'autres pour vous assurer que votre implémentation est correcte.

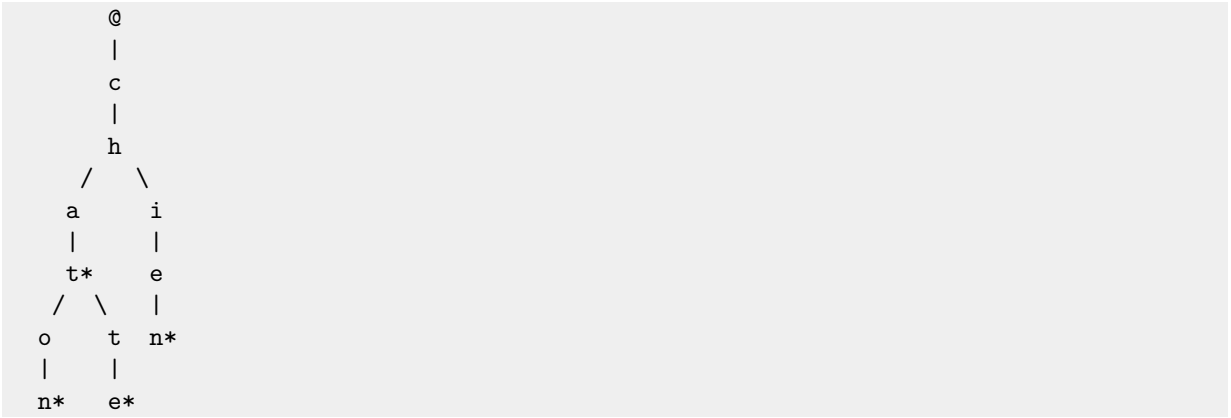
---

1 - Trie (arbre préfixe)

Un trie est une structure de données qui permet de stocker des mots et de les rechercher efficacement. On peut l'utiliser notamment pour analyser des textes et fournir efficacement des suggestions de mots (autocomplétion).

Un trie est un arbre dans lequel chaque noeud possède 26 fils (un par lettre de l'alphabet), représentant chacun la prochaine lettre d'un mot. Chaque noeud possède également un booléen indiquant s'il correspond à la fin d'un mot. La racine représente la chaîne vide. Toutes les feuilles du trie représentent nécessairement des mots.

Par exemple, le trie suivant contient les mots `chat`, `chaton`, `chatte`, et `chien` (`@` indique la racine et les `*` indiquent les noeuds correspondant à la fin d'un mot) :



L'état d'un trie ne dépend pas de l'ordre dans lequel les mots ont été insérés.

Implémentation demandée

Dans le fichier `containers/trie.h`, compléter la définition de la structure `trie_node` représentant un noeud du trie. Elle doit contenir les champs suivants :

- `is_word` : un booléen indiquant si le noeud correspond à la fin d'un mot
- `children` : un tableau de 26 pointeurs vers les fils du noeud (un par lettre de l'alphabet)

Compléter ensuite la définition de la structure `trie` représentant un trie. Elle contient :

- `root` : un pointeur vers le noeud racine du trie.

Attention: on fera l'hypothèse que les mots manipulés sont uniquement composés des 26 lettres minuscules de l'alphabet latin (entre `'a'` et `'z'`).

> Implémenter toutes les fonctions dans le fichier `containers/trie.c`.
> Des commentaires sont présents au-dessus des fonctions pour préciser ce qui est attendu.

Indications

Pour obtenir un `index` entre 0 et 25 à partir d'un caractère `c` dont la valeur est comprise entre `'a'` et `'z'`, on peut utiliser :

```
size_t index = c - 'a';
```

De même, pour obtenir un caractère `c` dont la valeur est comprise entre `'a'` et `'z'` à partir d'un `index` entre 0 et 25, on peut utiliser :

```
char c = index + 'a';
```

Pour parcourir une chaîne `str` caractère par caractère, on peut utiliser par exemple :

```
for (size_t i = 0; str[i] != '\0'; i++)
{
    char c = str[i];
    // ...
}
```

---

2 - Expressions bien parenthésées

Une expression est dite bien parenthésée si chaque symbole ouvrant correspond à un symbole fermant situé plus loin dans l'expression, et si l'expression située entre ces deux symboles est elle-même bien parenthésée.

Par exemple, si on utilise les symboles `()` et `[]`, les expressions suivantes sont bien parenthésées :

- `(abc)`
- `(a[b]c)`
- `([a(b)](c))`
- `[(a)(b)(c)]`

Alors que les expressions suivantes ne le sont pas :

- `abc)`
- `(abc]`
- `[a(b]c)`
- `a)([b](c)`

On considère la chaîne vide comme étant bien parenthésée.

Principe général

Pour vérifier si une chaîne de caractères contient une expression bien parenthésée, on peut utiliser une pile. Chaque symbole ouvrant rencontré est ajouté sur la pile. Dès qu'un symbole fermant est rencontré, on récupère l'élément ouvrant situé au sommet de la pile :

- s'il correspond bien au symbole fermant, on continue
- s'il n'y a plus d'élément ouvrant à dépiler (la pile était vide), ou si celui-ci ne correspond pas au symbole fermant rencontré, alors on peut s'arrêter et l'expression n'est pas bien parenthésée

Si on arrive au bout de la chaîne et que la pile est vide, alors l'expression est bien parenthésée.

Tous les caractères rencontrés autres que les symboles ouvrants et fermants considérés sont simplement ignorés.

Implémentation demandée

Dans le fichier `algo/parentheses.h`, compléter la définition de la fonction `check_parentheses` qui prend en paramètre une chaîne `str` et qui retourne un booléen indiquant si la chaîne est bien parenthésée ou non.

Indications

Une implémentation fonctionnelle de pile (`stack`) est déjà fournie dans le fichier `containers/stack.h`.

Comme pour l'exercice précédent, on peut parcourir une chaîne `str` caractère par caractère de la façon suivante :

```
for (size_t i = 0; str[i] != '\0'; i++)
{
    char c = str[i];
    // ...
}
```