L2 Algo – TP noté 1

Équipe pédagogique du L2 Algorithmique

1 Introduction

Dans ce TP noté il vous est demandé de développer 2 fonctions (**calc** et **shunting_yard**) dans le fichier **polonaise.c** et de déposer ce fichier et **uniquement ce fichier** sur moodle, **sans modifier son nom** : ce fichier sera évalué automatiquement en utilisant des scripts. Afin de maintenir de la clarté dans votre code et de simplifier vos développements, il vous est conseillé de définir et d'utiliser des fonctions auxiliaires à l'intérieur du fichier **polonaise.c**. Les scripts Python et fichiers C qui seront utilisés pour évaluer votre travail vous sont fournis afin que vous puissiez tester "en condition réelle" votre fichier **polonaise.c**. Vous pouvez évidemment les lire et les modifier si vous le souhaitez, mais l'évaluation finale de votre travail se fera en utilisant les scripts et fichiers originaux non modifiés. Assurez-vous donc avant de déposer votre fichier **polonaise.c** que celui-ci est bien compatible avec l'outillage d'évaluation tel qu'il vous est fourni. Par ailleurs les fichiers de test qui vous sont fournis se verront étoffés de cas de tests supplémentaires pour la notation.

2 Calculatrice en polonaise inversée

L'objectif de cet exercice est de réaliser une calculatrice d'expressions notées en polonaise inversée. La *polonaise inversée* ou *notation postfixe* consiste à d'abord écrire les opérandes d'une opération puis son opérateur. Ainsi, l'addition "$1+2$" s'écrira "1 2 +", l'expression "$3\times(2+5)$" s'écrira "3 2 5 + ×" et l'expression "$(1+2)\times 3$" deviendra "1 2 + 3 ×".

L'avantage de la notation en *polonaise inversée*, outre qu'elle ne nécessite pas de parenthèse, est qu'elle est très facile à programmer si on dispose d'une pile :

— si le terme courant est un nombre, on l'empile ;

— si le terme courant est un opérateur, on dépile les opérandes et on empile le résultat.

Le code à réaliser prendra en entrée une chaîne de caractère représentant l'expression en *polonaise inversée* où chaque caractère aura la signification suivante :

chiffre *c* nombre dont la valeur est c[1].

espace ignoré.

'+' opération d'addition (binaire).

'-' opération de soustraction (binaire).

'*' opération de multiplication (binaire).

'/' opération de division (binaire).

'^' opération de puissance (binaire).

',' opération de négation (unaire).

'!' opération de calcul de factorielle (unaire).

À faire (a) : écrire la fonction **calc** de calcul d'expression en *polonaise inversée* qui récupère une chaîne de caractère saisie au clavier et affiche le résultat s'il n'y a pas d'erreur. La fonction **calc** à développer se situe dans le fichier **polonaise.c**. Cette fonction utilisera la pile que vous avez développée lors des TPs précédents (fichier **pile.c** qu'il faudra ajouter au dossier courant afin de compiler). La fonction **main()** qui appelle **calc** pour la tester et la noter est située dans le fichier **polonaise_test.c** et **ne doit pas** être modifiée. Un script permettant de lancer des tests sur votre fonction **calc** vous est fourni : **test_polonaise.py**. La notation du TP utilisera exactement le même script, mais les fichiers contiendront des tests différents.

1. On se souviendra qu'en ASCII les chiffres sont codés de manière successive à partir du caractère '0'.

Expressions erronées : votre fonction **calc** doit être capable de détecter une expression mal formée et de renvoyer un code particulier lors de la détection des différents cas d'expressions mal formées :

— si l'expression comporte trop d'opérateurs d'affilée, **calc** doit renvoyer la valeur **-1**,
— si l'expression comporte trop d'opérandes (de chiffres), **calc** doit renvoyer la valeur **-2**.

Pour tester : On pourra commencer par tester le programme en tappant directement au clavier les expression suivantes :

— "1 2 3 + + =" → 6
— "6 5 * 3 / =" → 10
— "1 1 + 2 2 + * =" → 8
— "2 3 ′ 8 + 4 / =" → 4
— "1 , 4 ! + =" → 23

Après avoir vérifié que ces expressions sont correctement interprêtées par le programme, on pourra utiliser le script **test_polonaise.py** pour tester sur le reste des exemples, dans les conditions du processus de notation. Ce script récupère des fichiers de test dans le dossier **test** et les passe en entrée de la fonction **main()**.

3 Algorithme shunting-yard

On désire maintenant que l'utilisateur puisse saisir une expression sous forme infixe. Pour ce faire, on va lire les caractères tapés au clavier et construire la chaîne de caractère en polonaise inversée. Pour cela, on va utiliser l'algorithme Shunting-yard.

On dispose d'une chaîne de caractère résultat, e (initialement vide), et d'une pile des opérateurs, p (initialement vide).

A chaque caractère c lu,

— si c est un chiffre, il est ajouté à e
— si $c =$ '(', il est ajouté à p,
— si $c =$ ')', on dépile les éléments de p et on les ajoute à e (en les séparant par un espace) jusqu'à trouver '(', qu'on dépile sans l'ajouter à e,
— si c est un opérateur,

1. on dépile les éléments de p qu'on ajoute à e (en les séparant par un espace) jusqu'à trouver un opérateur moins prioritaire,
2. on ajoute c à p

À la fin, on vide la pile p et on concatène ses éléments à e (en les séparant par un espace). La chaîne e doit se terminer par un chiffre ou un opérateur (et pas par un espace). Comme toute chaîne de caractère en C, on concatène le caractère '\0' en fin de chaîne.

La priorité des opérateurs est la suivante :

Opérateur	Priorité
'('	0
'+', '-'	1
'*', '/'	2
'^'	3
',', '!'	4

À faire (b) : écrire la fonction **shunting_yard** qui récupère une chaîne de caractères représentant un calcul sous forme infixe saisie au clavier et qui la transforme en sa version postfixe (polonaise inversée) en utilisant l'algorithme shunting-yard. Cette fonction prend en entrée un pointeur vers une chaîne de caractères (a.k.a. un tableau de caractères) initialement vide e, dans laquelle il s'agit de produire la chaîne de caractères solution, à partir d'une autre chaîne de caractères (contenant une expression en notation infixe) tappée au clavier par l'utilisateur. On considèrera que la chaîne de caractères tappée par l'utilisateur est toujours bien formée (on ne prendra par conséquent pas en compte d'éventuels cas pathologiques). La fonction **shunting_yard** à développer se trouve dans le fichier **polonaise.c**. Comme pour la fonction de calcul en notation polonaise inversée, cette fonction utilisera la pile développée dans les TPs précédents. La fonction **main()** qui appelle **shunting_yard** est définie dans le fichier **test_sy.c** et **ne doit pas** être modifiée.

Pour tester : Un script de test/notation vous est fourni : **test_sy.py**. Ce script récupère des fichiers de test dans le dossier **test_sy** et les passe en entrée de la fonction **main()**.