Travaux Pratiques de C n°3

Cours de langage et programmation

—L3 Informatique—

On souhaite écrire un programme permettant de calculer des expressions données sous la forme de petites instructions élémentaires, à raison d'une par ligne. Pour simplifier, la calculatrice ne fonctionnera que sur des entiers. On souhaite cependant pouvoir définir des noms symboliques représentant des constantes entières, pour pouvoir les utiliser dans les expressions. Voici un exemple de ce que pourra faire la calculatrice :

```
   add 3 2
   add $ 8
   add x $
   print
@x=3
```

Voici comment il faut lire l'exemple. Partout, le nom symbolique désigne la constante 3. Le nom symbolique $ désigne toujours la valeur calculée. La première ligne fait la somme de 3 et 2. Juste après la première ligne, $ vaut donc 5. On fait la somme de 5 et de 8, ce qui donne 13. Après avoir fait cette somme, $ vaut donc 13. Enfin, la dernière addition ajoute 3 et 13, ce qui fait 16. Finalement, le `print` affiche la valeur de $, donc 16.

En plus de la somme, la petite calculatrice devra pouvoir faire le quotient, la différence et le produit d'entiers.

Un des problèmes rencontrés dans l'implémentation de la calculatrice sera d'associer une valeur à chaque nom symbolique. On voit bien, dans l'exemple, qu'un nom `x` est employé, mais qu'au moment de sa première utilisation, si on lit les instructions ligne à ligne, on ne connaît pas encore sa valeur : elle n'est définie qu'après.

Pour résoudre ce problème, nous allons employer la technique de la double passe (ou double lecture) : on fait une première lecture de toutes les lignes, dont le but n'est pas de calculer mais de construire une structure de données, la *table des symboles*, dans laquelle on associe à chaque nom symbolique sa valeur.

Après cette première lecture, on suppose qu'à chaque nom est associé une valeur. On peut alors commencer la seconde lecture (passe), qui elle a pour objectif de réaliser les calculs. À chaque fois qu'elle rencontrera un nom symbolique dans une opération, elle ira chercher sa valeur dans la table des symboles.

Les fonctions de l'exercice 1 seront regroupées dans un même fichier source (créer le fichier d'entêtes correspondant). De même pour les fonctions de l'exercice 3. Les fonctions permettant de traiter la ligne de commandes seront elles aussi regroupées.

▸ **Exercice 1.**
Définir en C *une structure de données pour la table des symboles. Écrire des fonctions*

`initSymbole()`, `addSymbole()`, `findSymbole()`, `printSymboleTable()` *permettant d'initialiser la table des symboles, d'y ajouter un nouveau symbole, d'y rechercher la valeur associée à un symbole, et d'afficher la table (ensemble des symbole et leur valeurs). Les fonctions retourneront toutes un entier indiquant leur bon déroulement ou non.*

Un autre problème est lié à l'analyse syntaxique des arguments des opérations : en effet, à partir d'une chaîne de caractères de la forme "3" ou "8" ou "`x`*" il faut réussir à extraire les valeurs correspondantes, éventuellement en cherchant dans la table des symboles quand c'est nécessaire. La fonction de la bibliothèque standard* `strtol()` *permet de récupérer, à partir d'un chaîne de caractères représentant un entier, la valeur de cet entier.*

▸ **Exercice 2.**
Écrire une fonction `recupereArgument(const char *arg, int *r)` *prenant en argument une chaîne de caractère contenant soit un nom symbolique, soit un entier, et qui met sa valeur dans* `*r`*. Encore une fois, la fonction retournera un entier indiquant si elle s'est bien déroulée ou non.*

Nous allons maintant écrire des fonctions réalisant les opérations de la calculatrice.

▸ **Exercice 3.**
Écrire une fonction `add_f(const char *,int *r)` *dont le premier argument est une chaîne de caractères de la forme "3 2", "$ 8" ou bien encore "x $", et qui réalise la somme de ces arguments. Le résultat est mis dans* `*r`*. Même chose pour les autres opérations arithmétiques et pour* `print`*.*

Nous savons maintenant faire des additions et gérer la table des symboles. Il nous reste à lire l'ensemble des lignes et à réaliser les opérations associuées. D'autre part, nous savons différencier les types des lignes en fonction de leur premier caractère :

- `@` *: la ligne contient la définition d'une constante ;*
- `;` *: la ligne contient un commentaire ;*
- `tab` *: la ligne contient une instruction.*

▸ **Exercice 4.**
Écrire une fonction `traiteConstante(const char *ligne)` *qui prend en argument une ligne entière, privée de son premier caractère et dont on sait qu'elle contient la définition d'une constante, donc que la ligne (sans son premier caractère) est de la forme "toto=234". La fonction aura pour rôle d'ajouter* `toto` *dans la table des symboles s'il n'y est pas déjà (erreur sinon). Comme d'habitude, elle retourne un entier indiquant si elle s'est bien déroulée ou non.*

▸ **Exercice 5.**
Écrire une fonction `traiteInstruction(const char *ligne)` *qui prend en argument une ligne entière, privée de son premier caractère, et dont on sait qu'elle contient une instruction, c'est-à-dire qu'elle est de la forme "add x 3". la fonction aura pour rôle d'appeler correctement* `add_f`, `sub_f`, `mul_f`, `quo_f` *ou* `print_f` *selon le cas. Comme d'habitude elle retourne un entier indiquant si elle s'est bien déroulée ou non.*

▸ **Exercice 6.**
Écrire une fonction `analyse()` *contenant une boucle lisant ligne à ligne un fichier ou l'entrée standard, et appelant, suivant le premier caractère de la ligne, soit la fonction* `traiteConstante()`, *soit* `traiteInstruction()`, *soit tout autre fonction que vous jugerez nécessaire.*

NOM

strtol - Convertir une chaîne en un entier long.

SYNOPSIS

```
#include <stdlib.h>
long int strtol (const char *nptr, char **endptr, int base);
```

DESCRIPTION

La fonction strtol() convertit la chaîne nptr en un entier long en fonction de l'argument base, qui doit être dans l'intervalle 2 à 36 (bornes comprises), ou avoir la valeur spéciale 0.
(...)

VALEUR RENVOYÉE

La fonction strtol() renvoie le résultat de la conversion, à moins qu'un débordement superieur (overflow) ou inférieur (underflow) se produise. Si un dépassement inférieur se produit, strtol() renvoie LONG_MIN. Si un dépassement supérieur se produit, strtol() renvoie LONG_MAX. Dans les deux cas, errno contient le code d'erreur ERANGE.
(...)

CONFORMITÉ

strol() : SVID 3, BSD 4.3, ISO 9899 (C99), POSIX.
(...)

##

NOM

strchr - Rechercher un caractère dans une chaîne.

SYNOPSIS

```
#include <string.h>
char *strchr (const char *s, int c);
```

DESCRIPTION

La fonction strchr() renvoie un pointeur sur la première occurence du caractère c dans la chaîne s.
(...)

VALEUR RENVOYÉE

La fonctions strchr() renvoit un pointeur sur le caractère correspondant, ou NULL si le caractère n'a pas été trouvé.
(...)

CONFORMITÉ

SVID 3, POSIX, BSD 4.3, ISO 9899