Université de Montréal

Rapport TP 1

Philippe Caron
Gabriel Lemyre

Travail remis à l'intention de:
Marc Feelay

Lundi 17 Octobre 2016

Fonctionnement général du programme

Ce programme est une calculatrice à précision infinie à l'intérieur d'une console. Le terminal affiche en premier la chaîne "> " afin d'inciter l'utilisateur à faire une entrée. Ces entrées doivent être sous forme postfixe.

Le format standard est "n1 n2 o" où 'n1' et 'n2' sont des nombres et 'o' est un opérateur binaire accepté. Le nombre 'n1' est toujours l'opérande de gauche et 'n2' l'opérande de droite. Si l'on veut effectuer une opération plus longue, il suffit d'ajouter un autre nombre suivi d'un opérateur comme suit: "n1 n2 o1 n3 o4". Les opérateurs binaires acceptés sous ce format standard sont '+', '-' et '*'.

'+' Permet l'addition;
'-' permet la soustraction;
'*' permet la multiplication.

Le second format accepté est "n1 u" où 'n1' est un nombre et u est un opérateur unaire accepté. Les opérateurs unaires acceptés sont '?' et '=variable' où variable est une lettre de [a-z] et où '?' n'accepte que les variables initialisées. Une variable ayant été initialisée par l'opérateur '=' est considérée comme un nombre.

'?' Permet de vérifier le compteur de référence d'une variable;
'=variable' permet l'initialisation d'une variable avec une valeur.

Il est possible de combiner les deux formats afin d'effectuer des opérations plus compliquées telles "451 741 * 4015 - =x". Si l'on veut, on peut utiliser le résultat calculé précédemment pour dans le calcul présent. Le résultat de l'opération est retourné précédé de la chaîne "(ANS) ". Après cela, il est à nouveau possible de faire une nouvelle opération.

Deux commandes textuelles sont aussi utilisables:

"free" Permet de libérer l'espace occupé par les variables et ainsi les réinitialiser;
"exit" permet à l'utilisateur de quitter le programme.

Le programme utilise un système d'empilage et dépilage afin de pouvoir gérer des nombres arbitrairement grands avec aisance.

Représentation des nombres et variables

La représentation des nombres est faite avec l'utilisation de trois structures.

La première est **digits** et possède deux membres:

int digit;
struct Digits *next.

La structure **digits** est une pile de chiffres. L'entier digit étant la valeur du chiffre de 0 à 9 contenu à la position dans la pile (unité, dizaine, centaines...) et struct **Digits** *next étant le pointeur vers le prochain élément de la pile représentant le nombre.

La seconde structure est **bingint** et possède deux membres:

int flags;
digits *value.

La structure **bigint** est la représentation d'un nombre. L'entier flags permet de faire deux choses. Il permet de vérifier le compteur de référence en faisant un décalage logique à gauche d'un bit et il permet d'évaluer la négativité du nombre avec un et bit-à-bit lorsqu'utilisé avec le chiffre 1. Le pointeur *value de type **digits** pointe vers l'élément au sommet de la pile de chiffres.

La troisième structure est **stack** et possède trois membres:

bigint **ptr;
int len;
int cap.

La structure **stack** est la représentation d'une pile de nombres. Le membre **ptr est un pointeur vers l'adresse du bigint qui est au dessus de la pile. Le membre de type **int** len représente la taille actuelle de la pile et cap représente sa capacité. Il n'y a qu'un **stack** d'utilisé.

Les variables sont représentées par un pointeur sur le tableau variables de taille 52 et de type **bigint**. La position 0 est assignée à la variable 'A' et la position 51 à 'z'.

Analyse par ligne et calcul de la réponse

Afin de lire une ligne de taille arbitraire, nous utilisons la fonction getchar() qui nous retourne le prochain caractère. Depuis ce caractère, nous vérifions si c'est un chiffre, un opérateur autorisé, un espace vide, une tabulation, un 'end of file' ou encore une variable.

Si le caractère est un chiffre, le programme itère jusqu'à trouver le dernier chiffre formant le nombre et créé un **digits** pour chacun des chiffres qui pointe vers le prochain élément formant la pile du nombre. Le pointeur value d'un **bigint** initialisé reçoit ensuite la valeur au sommet de la pile de **digits**. Le **bigint** est ensuite empilé sur le **stack** principal.

Lorsqu'il s'agit d'un opérateur autorisé à l'exception de '=', la méthode de l'opération désirée est appelée directement. Toutefois, quand il s'agit de '=', le caractère suivant est récupéré et le programme s'assure que la variable fait bien partie de l'ensemble des lettres majuscules ou minuscules.

S'il s'agit d'un espace vide ou d'une tabulation, ils sont ignorés.

Quand le caractère devient 'end of file', la pile est vidée et libérée tout comme le tableau de variables. Le programme est ensuite quitté.

Lorsqu'il s'agit d'autre chose, le programme détermine si c'est une lettre et si elle est utilisée en tant que variable. La lecture d'une variable non utilisée se fait à l'aide de l'opérateur '='.

Après l'exécution d'une opération, le résultat est empilé et utilisé comme opérande de gauche de la prochaine opération.

Le résultat est affiché lorsqu'il n'y a plus d'opérateurs et qu'il ne reste qu'un élément dans la pile principale.

Gestion mémoire

La gestion de la mémoire est effectuée dans chacune des parties importantes du programme: dans le "main", à l'intérieur des procédures opératoires et dans les méthodes du **stack**. Tous les "malloc" sont suivi de "sizeof()" appropriés pour leurs types. Un "free" est toujours utilisé sur les pointeurs ayant des valeurs différentes de "NULL" lorsqu'un "malloc" ou "realloc" retourne "NULL". En cas d'échec du programme, la pile est aussi vidée et libérée.

En premier lieu, trois "malloc" sont effectués dans le "main" afin d'accorder de l'espace pour un pointeur sur un **bigint**, un pointeur sur un **digits** et le pointeur du **stack**.

Ensuite, pour les opérations '+', '-', '*' et '?', un pointeur sur une structure **bigint** et un autre sur une structure **digits** sont initialisés en début de méthode et à chaque ajout de chiffre au résultat, le pointeur sur **digits**->next fait un "malloc" afin d'augmenter la taille de la pile de chiffres. La mémoire des nombres utilisés pour l'opération est ensuite libérée.

L'opération "push" sur le **stack** utilise "realloc" afin d'ajouter de l'espace sur la pile et l'opération "pop" permet d'en enlever.

Implémentation des algorithmes

L'addition est implémentée en parcourant les **digits** des deux **bigint** du **stack** et additionne les unités avec les unités, les dizaines avec les dizaines, etc. Jusqu'à ce que les deux piles soient vides. Si le résultat de l'addition est supérieur ou égal à 10, la prochaine itération se verra ajouter une valeur de 1. Le résultat de chaque addition est empilé sur le pointeur de type **bigint** du résultat. Si les deux piles sont vides, mais que la prochaine itération est sensée recevoir une valeur de 1, le 1 est empilé sur le pointeur de type **bigint** du résultat. Aussi, l'addition appelle la soustraction si l'un des deux éléments est négatif.

La soustraction s'effectue comme l'addition avec certaines exceptions. D'abord, lorsque la soustraction de deux éléments donne un résultat plus petit que 0, on ajoute 10 à cette valeur et l'on retire 1 au résultat de la prochaine itération. La seconde différence est l'inversion des valeurs si le nombre résultant est négatif. Ceci est effectué en parcourant la pile de chiffres du pointeur de type **bigint** en modifiant les valeurs pour qu'elles soient égales à "10 - chiffre - ret" où 'ret' est la valeur de 0 à 1 à retirer à la prochaine itération. (Ce chiffre est emprunté au prochain élément). Aussi, la soustraction fait appel à l'addition si un des deux éléments est négatif.

La multiplication est différente. Le programme boucle sur l'opérande de droite et multiplie chacun des chiffres avec celui de l'opérande de gauche actuel. Lorsque tous les chiffres ont été parcourus, le résultat est empilé sur le **stack** principal et l'opérande de gauche prend la valeur du prochain chiffre et recommence la boucle sur l'opérande de droite en ajoutant avant une valeur de 0 sur la pile du résultat par niveau d'itération (donc aucun pour les unités, un pour les dizaines, deux pour les centaines, etc.). Une addition est ensuite effectuée sur les résultats empilés par la multiplication.

Une vérification de négativité s'effectue à la fin de chacune de ces opérations.

Traitement des erreurs

Une procédure "alert" a été implémentée afin d'afficher les messages d'erreurs appropriés lorsqu'une erreur survient. Si tel est le cas, la procédure vide et libère la pile puis affiche la ligne d'entrée suivante.

Les erreurs gérées sont les suivantes:

Mémoire insuffisante lors d'un "malloc" ou "realloc";
Variable différente d'une lettre.
Format de l'opération non conforme au standard postfixe;
Utilisation d'une variable sans l'avoir préalablement définie;
Tentative de définition de variable avec plusieurs caractères.