Projet C - Arabica

Quand les autres cafés sont indisponibles.

Introduction

Maintenant que vous avez acquis de bonnes connaissances en C, nous allons nous exposer à de nouvelles problématiques : la compilation, et l'interprétation du langage machine.

Notre objectif est donc de créer l'Arabica compiler, pour créer des binaires interprétés par l'Arabica VM (Fournie).

L'Arabica est un langage de programmation qui ressemble quelque peu à son cousin l'assembleur.

Il utilise une stack lors de l'exécution du code, ainsi que des registres pour stocker les variables.

Bytecode

Instr.	Args	Pop	Description
01	4	0	LOAD_VAL <valeur> : Empile une valeur numérique (int32) sur la pile.
02	1	0	READ_VAR <identifiant> : Empile la valeur de la variable spécifiée (uint8 pour l'ID de la variable).
03	1	1	STORE_VAR <identifiant> : Stocke la valeur du sommet de la pile dans la variable (uint8 pour l'ID de la variable).
04	0	2	ADD : Additionne les deux valeurs numériques (int32) au sommet de la pile.
05	0	2	SUB : Soustrait les deux valeurs numériques (int32) au sommet de la pile.
06	0	2	MUL : Multiplie les deux valeurs numériques (int32) au sommet de la pile.
07	0	2	DIV : Divise les deux valeurs numériques (int32) au sommet de la pile.

08	0	2	MOD : Modulo des deux valeurs numériques (int32) au sommet de la pile.
09	4	0	JMP <offset> : Saute à l'instruction en se décalant de la valeur de l'offset (int32).
0A	4	1	JMP_IF_ZERO <offset> : Saute à l'instruction en se décalant de la valeur de l'offset si la valeur du sommet est zéro (int32).
0B	4	1	JMP_IF_TRUE <offset> : Saute à l'instruction en se décalant de la valeur de l'offset si la valeur du sommet est non nulle(int32).
0C	0	2	EQ : Empile 1 si les deux valeurs au sommet (int32) sont égales, sinon 0.
0D	0	2	NEQ : Empile 1 si les deux valeurs au sommet (int32) ne sont pas égales, sinon 0.
0E	0	2	GT : Empile 1 si la valeur en sous-sommet (int32) est plus grande que celle au sommet, sinon 0.
0F	0	2	LT : Empile 1 si la valeur en sous-sommet (int32) est inférieure à celle au sommet, sinon 0.

10	0	2	GTE : Empile 1 si la valeur en sous-sommet (int32) est plus grande ou égale à celle au sommet, sinon 0.
11	0	2	LTE : Empile 1 si la valeur en sous-sommet (int32) est inférieure ou égale à celle au sommet, sinon 0.
12	0	1	PRINT_VAL : Dépile une valeur de la pile (int32) et l'imprime.
13	0	0	INPUT_VAL : Lit une valeur numérique (int32) de l'entrée utilisateur et l'empile.
14	0	0	HALT : Arrête l'exécution de la machine virtuelle.
15	1+var.	0	LOAD_STR <taille, chaîne> : Empile une chaîne de caractères (uint8 pour sa taille, suivi de la chaîne elle-même, pas terminée par un NUL).
16	0	1	PRINT_STR : Dépile une référence à une chaîne et l'imprime.
17	0	0	INPUT_STR : Lit une chaîne de l'entrée utilisateur et empile une référence à celle-ci (uint8).
18	0	1	STR_LEN : Dépile une référence à une chaîne et empile la longueur de celle-ci (int32).

19	0	2	CONCAT : Dépile deux références à des chaînes, les concatène et empile la référence au résultat.
1A	0	1	GET_CHAR : Dépile une référence à une chaîne (uint8) et un indice (uint8), puis empile le caractère à cet indice (char).
1B	3	0	SET_CHAR : Dépile un caractère (char), un indice (uint8) et une référence à une chaîne (uint8), et insère le caractère à l'indice spécifié.
1C	0	2	STR_CMP : Dépile deux références à des chaînes, les compare et empile le résultat.

Le fichier compilé doit avoir un entête de la forme :
4 octets "CODE"
4 octets indiquant la taille du code lui-même (entête exclu).
16 octets pour le nom du programme.

Donc 24 octets d'entête.

La stack

Quand une instruction empile une valeur, celle-ci est placée au-dessus de la stack.
Une instruction qui dépile une valeur va donc l'utiliser, et si nécessaire empiler une nouvelle valeur.

Une valeur dépilée n'est plus accessible du tout, il est donc important de la stocker dans un registre s'il est nécessaire de la réutiliser.

Les registres

Vos variables peuvent être stockées dans des registres, qui doivent être nommés dans le code Arabica, et convertis en valeur numérique (représentant un index de registre) lors de la compilation.

Les labels

Des labels peuvent être utilisés dans le code Arabica, pour faire référence à une position dans le code. A la compilation, ceux-ci sont transformés en offsets.

Exemple

```
          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000  43 4F 44 45 00 00 00 13 48 45 4C 4C 4F 00 00 00
00000010  00 00 00 00 00 00 00 00 01 00 00 00 2A 15 0B 48
00000020  45 4C 4C 4F 20 57 4F 52 4C 44 16
```

- L'entête :
  CODE (4 octets)
  19 (taille total du code, entête exclue) (sur 4 octets)
  HELLO, suivi de 11 octets vides

- Le code :
  01 : LOAD_VAL
  00 00 00 2A : 42
  15 : LOAD_STR
  0B : 11
  HELLO WORLD (la string, qui fait 11 caractères)
  16 : PRINT_STR

- En Arabica :

```
HELLO.abc
1    LOAD_VAL 42
2    LOAD_STR "HELLO WORLD"
3    PRINT_STR
```

Attention : on peut voir ici que la VM est gros-boutiste.

Soutenance / Rendu

Votre code doit être compilé automatiquement, et avec les flags -Wall -Wextra -Werror.

L'exécutable généré doit s'appeler arabica.

Le programme doit être stable, ne doit pas crash, et ne pas avoir de fuites de mémoire.

Le projet sera réalisé sur une période de deux semaines de présence, par groupes de 3 ou 4. La soutenance aura lieu la semaine de présentiel suivante.

La soutenance sera divisée en deux étapes, sur 30 minutes : Présentation de l'équipe et du projet (conception, algorithmes utilisés, problèmes rencontrés...), puis analyse du code, et test du programme. Pour le test du programme, vous devriez avoir au moins un programme simple pouvant être compilé et exécuté.

Compétences visées

- C
- Parsing
- Compilation

Base de connaissances

- Lexing/parsing
- Boutisme

La VM

- Windows
- Linux
- Mac