Scale for project Abstract VM

Git repository

;sphere@vogsphere.42.fr:intra/2015/activities/abstract_vm/kpoirier

Introduction

Nous vous demandons, pour le bon déroulement de cette notation :

- De rester courtois, poli, respectueux, constructif, lors de cet échange. Le lien de confiance entre la communauté 42 et vous en dépend.
- De bien mettre en évidence auprès de la personne notée (ou du groupe) les dysfonctionnements éventuels.
- D'accepter qu'il puisse y avoir parfois des différences d'interprétation sur les demandes du sujet ou l'étendue des fonctionnalités. Restez ouvert d'esprit face a la vision de l'autre (a-t-il raison ou tort ?), et notez le plus honnêtement possible.

Bonne soutenance à tous !

Guidelines

RAPPELEZ VOUS QUE VOUS NE DEVEZ CORRIGER QUE CE QUI SE TROUVE SUR LE DÉPÔT DE RENDU DE L'ÉTUDIANT.

Il s'agit de faire un \"git clone\" du dépôt de rendu, et de corriger ce qui s'y trouve.

Ratings

Define the type of error (if there is an error), which ended the correction.

Ok | Empty work | Incomplete work | No author file | Invalid compilation | Norme | Cheat | Crash

Attachments

Subject

Sections

Préliminaires

Tests préliminaires

Vérifiez d'abord les éléments suivants :

- Il y a bien un rendu (dans le dépôt git)
- Pas de triche, les étudiants doivent pouvoir expliquer leur code.

Si un élément n'est pas conforme au sujet, la notation s'arrête la. Vous êtes encouragés à continuer de débattre du projet, mais le barème n'est pas appliqué.

Yes | No

Tests de fonctionnalité

Test 1

Testez le programme suivant :

```
push int32(42)
push int32(33)
add ;poney
push float(44.55)
mul
push double(42.42)
;commentaire de ouf
push int32(42)
dump
pop
assert double(42.42)
exit
```

Le programme s'exécute-il correctement ?

Yes No

Test 2

Testez le programme suivant :

```
push int32(32)
push int32(0)
div
exit
```

Le programme s'arrête-il proprement en erreur à cause de la division par 0 ?

Yes No

Test 3

Testez le programme suivant :

```
push int16(99999999999999999999999999)
exit
```

Le programme s'arrête proprement en erreur à cause de l'overflow ?

Yes No

Test 4

Testez le programme suivant :

```
push int16(32 ;)
pu int(32))
exit
```

Le programme s'arrête sur une erreur de syntaxe ?

Yes No

Test 5

Testez le programme suivant :

```
pop
exit
```

Le programme s'arrête proprement en erreur à cause du pop sur stack vide ?

Yes No

Yes No

Test 6

Testez le programme suivant :

```
push int32(42)
assert int32(0)
exit
```

Le programme s'arrête proprement en erreur sur l'assert ?

Yes No

Test 7

Testez le programme suivant :

```
push int32(42)
add
exit
```

Le programme s'arrête proprement en erreur sur l'opérande manquante ?

Yes No

Test 8

Testez le programme suivant :

```
push int8(33) ;!
push int8(112) ;p
push int8(111) ;o
push int8(108) ;l
push int8(112) ;p
print
pop
print
pop
print
pop
print
pop
print
pop
exit
```

Le programme s'exécute et affiche "plop!" ?

Yes No

Test custom

Testez un programme de votre invention. Par exemple, faites des operations avec croisement de types avec de très grands et de très petits nombres (hors overflow/underflow).

Ça fonctionne comme vous vous y attendiez ?

Yes No

Test custom difficile

Essayez de tester un programme difficile de votre invention (genre vicieux quoi).

Ça fonctionne comme vous vous y attendiez ?

Yes No

Implémentation

Entrées

La VM doit pouvoir lire son entrée depuis un fichier, ou sur l'entrée standard (avec ;; pour délimiter la fin de l'entrée).

Yes No

Pile

La VM contient une "pile". Elle peut ne pas être une std::stack sauf si cela est justifié rigoureusement (std::stack n'est pas itérable, elle peut servir de classe de base au mieux).

Yes No

Opérandes polymorphiques

Les Operand sont manipulées de façon polymorphique à travers IOperand *.
Dans le cas contraire, le rendu est hors sujet. Cliquez sur le
flag \"crash\", la notation n'est plus appliquee, mais vous etes
encourages a continuer de discuter.

Yes No

Opérandes factory

Il doit exister une "factory" d'Operand implémentant la fonction suivante :

IOperand * SomeClass::createOperand(eOperandType type, const std::string & value);

Yes No

Gestion de précision

La VM gere la précision de façon non triviale - pas de batteries d'ifs ou autres saletés. Une enum est tout à fait acceptable par exemple.

Yes No

Parseur

La VM a un parsing propre et complet ?

Yes No

Exceptions

La VM doit utiliser des exceptions pour gérer certaines erreurs.

Sélectionnez la graduation correspondante :
- Pas d'exceptions : 0
- Exceptions scalaires (string, char*, int, ...) : 1
- Présence d'exceptions pré-faites (uniquement std::exception ou autre) : 2
- Présence d'exceptions custom héritant de std::exception : 3
- Présence d'exceptions custom heritant d'une classe plus specialisée que std::exception : 4

Rate it from 0 (failed) through 5 (excellent)

Bonus

Vérification complète

La VM est capable de donner toutes les erreurs d'un fichier, et ne s'arrête pas à la première erreur rencontrée (hors interprétation).

Yes No

Parsing avancé

Le parsing est bien structuré, à savoir couple lexer / parser avec des rôles bien définis tels qu'ils sont censés être dans la realité.

Yes No

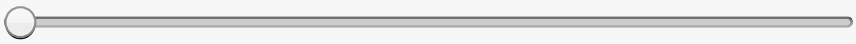
D autres bonus

Comptez dans cette partie s'il y a d'autres bonus. Vous pouvez noter jusqu'à 5 bonus disctincts.

Chaque bonus doit être :
- Un minimum utile (à votre discrétion)
- Bien réalisé et 100% fonctionnel

1. mode verbose
2. interactif
3. couleur
4. instruction (max, min, avg, pow)
5. instructions bitwise: ET/OR/XOR
6. log (verbose strings)

Rate it from 0 (failed) through 5 (excellent)



Conclusion

Leave a comment on this correction