SCALE FOR PROJECT ABSTRACT VM (/PROJECTS/ABSTRACT-VM)

Introduction

Please respect the following rules:

- Remain polite, courteous, respectful and constructive
throughout the correction process. The well-being of the community
depends on it.

- Identify with the person (or the group) graded the eventual
dysfunctions of the work. Take the time to discuss
and debate the problems you have identified.

- You must consider that there might be some difference in how your
peers might have understood the project's instructions and the
scope of its functionalities. Always keep an open mind and grade
him/her as honestly as possible. The pedagogy is valid only and
only if peer-evaluation is conducted seriously.

Guidelines

- Only grade the work that is in the student or group's
GiT repository.

- Double-check that the GiT repository belongs to the student
or the group. Ensure that the work is for the relevant project
and also check that "git clone" is used in an empty folder.

- Check carefully that no malicious aliases was used to fool you
and make you evaluate something other than the content of the
official repository.

- To avoid any surprises, carefully check that both the correcting
and the corrected students have reviewed the possible scripts used
to facilitate the grading.

- If the correcting student has not completed that particular
project yet, it is mandatory for this student to read the
entire subject prior to starting the defence.

- Use the flags available on this scale to signal an empty repository,
non-functioning program, a norm error, cheating etc. In these cases,
the grading is over and the final grade is 0 (or -42 in case of
cheating). However, with the exception of cheating, you are
encouraged to continue to discuss your work (even if you have not
finished it) in order to identify any issues that may have caused
this failure and avoid repeating the same mistake in the future.

Attachments

Subject (/uploads/document/document/999/abstract-vm.en.pdf)

Preliminaries

Preliminaries tests

Check firstly the following elements:

- There is something in the git repository.
- No cheating (All functions are authorised, the student can explain
the code, ...)

If an element isn't implemented as explained in the subject, the
grading ends. Use the appropriate flag. You're allowed to debate
some more.

✓ Yes ✗ No

Feature's testing

Test 1

Run the following program:

```
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
```

Does the program execute properly?

✓ Yes ✗ No

Test 2

Run the following program:

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

Does the program stops properly because of the 0 division?

✓ Yes ✗ No

Test 3

Run the following program:

```
push int16(9999999999999999999999999999)
exit
```

Does the program stops properly because of the overflow error?

✓ Yes ✗ No

Test 4

Run the following program:

```
push int16(32 ;)
```

```
pu int(32))
exit
```

Does the program stops properly because of a syntax error?

✓ Yes ✗ No

Test 5

Run the following program:

```
pop
exit
```

Does the program stops properly because of an empty stack?

✓ Yes ✗ No

Test 6

Run the following program:

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

Does the program stops properly on an assert error?

✓ Yes ✗ No

Test 7

Run the following program:

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

Does the program stops properly on a missing operand?

✓ Yes ✗ No

Test 8

Run the following program:

```
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
```

Does the program run properly and display the following output?

```
p
l
o
p
!
```

✓ Yes ✗ No

Custom test

Run your own tests. For example, run operation with mixed types, really big or really small numbers (overflow and underflow excluded).

Does the program run as expected?

✓ Yes ✗ No

Difficult custom test

Run a really complicated program of your invention (a vicious test basically).

Does the program run as expected?

✓ Yes ✕ No

Implementation

Inputs

The VM must be able to read either from a file or from the standard input
(with a ;; to end the input)

✓ Yes ✕ No

Stack

The VM countains a "stack". It can't be a std::stack except if
rigorously justified (std::stack isn't iterable, it can at best
be used as a base class).

✓ Yes ✕ No

Polymorphic operands

Are operand manipulated polymorphicaly through IOperand *.
If not, the project is off topic. Click on the "crash" flag,
the grading stops but you're allowed to debate some more.

✓ Yes ✕ No

Operand factory

There must be an operand "factory" implementing the following function:

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

✓ Yes ✕ No

Precision management

The VM manages precision in a non trivial way - An if forest or any
other disgusting thing. An enum is totally acceptable for example.

✓ Yes ✕ No

Parser

The VM has a clean and clomplete parsing?

✓ Yes ✕ No

Exceptions

The VM must use exceptions to manage errors.

Select the corresponding grade:
- No exceptions: 0
- Scalar exceptions (string, char*, int, ...): 1
- Use of pre-made exceptions (only std::exception ou autre): 2
- Use of custom exceptions custom inheriting from std::exception: 3
- Use of custom exceptions custom inheriting from a more specific class
than std::exception: 4

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

○

Bonus

Complete verification

The VM is capable of ouputing every error in a file, and doesn't stop at the first error met (interpretation excluded).

✓ Yes ✕ No

Advanced parsing

The parsing is well structured, more specificaly a lexer / parser combo with well defined roles as it should be in reality.

✓ Yes ✕ No

Other bonus

Count in this section the different bonuses. You can grade up to 5 distinctive bonuses.

Each bonus must be :
- At the very least useful (up to you)
- Well implemented and 100% functional

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

Ratings

Don't forget to check the flag corresponding to the defense

✓ Ok ★ Outstanding project

Empty work Incomplete work No author file Invalid compilation

Norme Cheat Crash Incomplete group Forbidden function

Conclusion

Leave a comment on this evaluation

Preview!!!