Napiszmy sobie interpreter

Zadania kwalifikacyjne

WWW19

Michał Horodecki

Wstęp

Nie trzeba robić wszystkiego, ale zachęcam żeby zrobić jak najwięcej ;)

Zadania proszę wysyłać przez **stronę warsztatów** w dowolnym (rozsądnym) formacie.

Będę się starał oceniać rozwiązania na bieżąco, tak abyście mogli dosłać poprawki w terminie.

W razie jakichkolwiek pytań i wątpliwości śmiało piszcie maila na `michalhorodecki2002+www@gmail.com`

Powodzenia!

Zadanie 0 - Ankieta [1pkt]

Napisz krótko coś o sobie. Dlaczego chcesz iść na te warsztaty i co spodziewasz się zobaczyć? Jakie jest Twoje doświadczenie z językami programowania a w szczególności z Pythonem? Czy pisał_ś lub myślał_ś kiedyś nad własnym językiem programowania, a jeśli tak to jakim?

Zadanie 1 - Python [20pkt]

Celem tego zadania jest sprawdzenie Waszej znajomości Pythona, w którym będziemy pisać nasz projekt, a przy okazji da Wam przedsmak tego, co będziemy robić na warsztach :).

Do zadań przygotowałem testy: `https://github.com/mhorod/www19-quals`, w razie niejasności polecam do nich zajrzeć, a jeśli nawet tam nie ma odpowiedzi to zachęcam do napisania maila.

Przy ocenianiu będę przede wszystkim patrzeć na to ile testów przechodzi oraz na styl i czytelność kodu.

Maszyna

Wyobraźmy sobie bardzo prostą maszynę – może ona:

- zapisywać/odczytywać dane z pamięci adresowanej kolejnymi liczbami począwszy od zera
- wykonywać proste skoki warunkowe
- dodawać i odejmować 1
- wypisywać zawartość pamięci

Aby napisać program na taką maszynę potrzebujemy dać jej ciąg instrukcji zrozumiałych dla niej. Z drugiej strony fajnie jakby program był też w miarę zrozumiały dla człowieka.

Twoim zadaniem jest zaimplementować maszynę, która wykonuje instrukcje podane w odpowiedniej formie, oraz „tłumacza", który potrafi wygenerować ciąg instrukcji mając na wejściu jedynie program jako ciąg znaków.

Do zaimplementowania są dwie funkcje w pliku `machine.py`:

1. `parse_program`, która dostaje na wejściu program jako ciąg znaków i zwraca listę instrukcji.
2. `execute_instructions`, która dostaje na wejściu listę instrukcji, wykonuje je, a następnie zwraca wyjście maszyny.

Poniżej specyfikacja zadania:

Instrukcje maszyny

W pliku `machine_language.py` jest zdefiniowany język maszyny. Pokrótce:

1. Maszyna dostaje listę instrukcji typu `Instruction`. Każda instrukcja składa się z rodzaju instrukcji, oraz argumentów specyficznych dla różnych rodzajów. Dla uproszczenia, argumentem zawsze będzie obiekt typu `Address`
2. Adres `Address` składa się z trybu adresowania `AddressMode` oraz liczby.

Adresowanie

Mamy trzy rodzaje adresów:

1. natychmiastowy – `IMMEDIATE` – nawet nie sięgamy do pamięci; po prostu zwracamy wartość adresu
2. bezpośredni – `DIRECT` – odczytujemy wartość z komórki w pamięci o numerze adresu
3. pośredni – `INDIRECT` – odczyt następuje z komórki o numerze odczytanym z komórki o numerze adresu

Instrukcje

1. `INC [adres]` – zwiększa wartość pod danym adresem o 1
2. `DEC [adres]` – zmniejsza wartość pod danym adresem o 1

3. `STOP` – przerywa wykonanie programu
4. `PRINT [adres]` – wypisuje wartość spod adresu
5. `JZERO [adres warunku] [adres docelowy]` – przechodzi do instrukcji wskazywanej przez adres docelowy, jeśli wartość pod adresem warunku wynosi zero i kontynuuje stamtąd program
6. `JNZERO [adres warunku] [adres docelowy]` – tak samo, ale gdy wartość nie jest równa zero

Człowiek → Maszyna

Adresowanie

Dostęp do pamięci odbywa się poprzez adresy, które mają jedną z trzech postaci:

- `.` – `IMMEDIATE`
- `@` – `DIRECT`
- `*` – `INDIRECT`

Specyfikacja języka

- `inc [adres]` – `INC [adres]`
- `dec [adres]` – `DEC [adres]`
- `stop` – `STOP`
- `print [adres]` – `PRINT [adres]`
- `label [napis]` – tworzy etykietę o zadanej nazwie, która wskazuje na kolejną instrukcję. Używane jedynie w skokach.
- `jzero [adres] [napis]` – jeśli odczyt spod zadanego adresu jest równy zero to skocz do pierwszej instrukcji pod etykietą o zadanej nazwie
- `jnzero [adres] [napis]` – jak wyżej, ale tylko gdy odczyt nie jest równy zero

Uwaga! Zauważ, że maszyna sama w sobie nie posiada etykiet, więc trzeba je przetłumaczyć odpowiednio na język maszyny.

Zadanie 2 - Języki Programowania [10pkt]

Każde pytanie jest warte 2pkt

- Na przykładzie wybranego języka wyjaśnij czym są typy, w jaki sposób i dlaczego są używane.
- Czym różni się programowanie obiektowe od funkcyjnego? Jakie są wady i zalety obu tych podejść?
- Jaka jest różnica między językiem kompilowanym a interpretowanym? Czy dałoby się skompilować Pythona i zinterpretować C++ ? Jakie zmiany należałoby wprowadzić do tych języków aby to było możliwe?
- Podaj dwa rodzaje błędów jakie mogą wystąpić podczas kompilowania/interpretowania programu. Krótko wyjaśnij na czym polegają i do każdego podaj przykład.
- Czym się różni wyrażenie infiksowe od prefiksowego? Jakie są wady i zalety tych notacji?

Zadanie 3 - Programowanie Funkcyjne [10pkt]

Aby było nam nieco prościej, napiszemy język funkcyjny. W związku z tym, fajnie będzie jak będziecie mieć jakieś pojęcie z czym to się je.

Zasady

1. Język może być dowolny, ale zachęcam do spróbowania jakiegoś języka funkcyjnego np. Haskell, Lisp, Elm. Nie trzeba ich instalować, są dostępne np. na stronie `https://replit.com/`
2. Zakaz używania pętli
3. Zakaz używania `goto`
4. Zakaz używania gotowych funkcji rozwiązujących zadania
5. Każda zmienna powinna być stała tj. nie wolno w żaden sposób modyfikować raz przypisanych wartości; jeśli wybrany język nie posiada stałych to należy upewnić się, że nie przypisujemy wartości dwa razy
6. Zachęcam do korzystania z rekurencji i list comprehensions

Napisz program który

1. znajdzie największy element listy [2pkt]
2. znajdzie n-tą liczbę Fibbonaciego [2pkt]
   Uwaga: Należy zadbać o złożoność tzn. program powinien działać szybko nawet dla $n \geq 1000$
3. sprawdzi czy liczba jest pierwsza [2pkt]
4. posortuje listę [2 lub 4 pkt]
   Wersja $\mathcal{O}(n^2)$ [2pkt]
   Wersja $\mathcal{O}(n \log n)$ [4pkt]