Interpreter - Robert Suchocki

Sposób uruchomienia

Po wykonaniu make w katalogu pojawi się wykonywalny plik interpreter, program akceptuje ścieżkę pliku podaną jako argument uruchomienia lub czyta z wejścia, gdy ścieżka nie zostanie podana. Obie poniższe komendy są równoważne

- ./interpreter ./folder/program
- ./interpreter < ./folder/program

Krótki opis

Interpreter korzysta z monad Reader, Except, State i IO. W monadzie Reader przechowywane jest środowisko zmiennych oraz funkcji, które może ulec zmianie tylko i wyłącznie w części deklaracji bloku, która wypełnia środowisko zmiennymi, a następnie w tym środowisku uruchamia resztę bloku, czyli instrukcje. Monada State przechowuje “store” z wartościami zmiennych, na początku zapisywane są do niego wartości inicjalizujące zmienne w czasie deklaracji, a później może (i często ulega) zmianie. Środowisko obecne podczas deklaracji funkcji jest wraz z jej argumentami i blokiem zapamiętywane i odtwarzane na czas wykonania tej funkcji, podczas gdy “store” jest globalny i wszystkie zmiany wykonane w dowolnym momencie w nim pozostają. Do tego monady Except używam przy błędach czasu wykonania programu, które zostają przy pomocy tej monady rzucone, a następnie przerywają wykonanie reszty programu i zostają wypisane. Na końcu jest monada IO służąca do obsługi funkcji print i komunikatów diagnostycznych. Do tego wykonanie programu poprzedza faza weryfikacji typów zmiennych, która podobnie jak program idzie po drzewie instrukcji programu i wykrywa błędy w typowaniu przed wykonaniem samego programu

Zmiany względem deklaracji gramatyki

- Rezygnacja z dwóch sposobów przekazywania parametrów na rzecz dodania operacji przerywających pętle while
- Podział bloku na listę deklaracji i instrukcji, brak możliwości deklarowania między instrukcjami (powodują błąd parsowania)
- Zmiana arrayów na listy
    - Nie jest podawana długość przy tworzeniu listy
    - Można pobierać wartości na indeksach 0..len-1
    - Można zapisywać na indeksach 0..len, gdzie zapis na len zapisuje na koniec listy i ją wydłuża o jeden
- Zmiana składni dict z {} na {{}} celem uniknięcia konfliktów w bnfc

Cechy języka Breve

- Typy zmiennych: int, bool, str, list, dict, statycznie typowane
- Zmienne, operacje przypisania, działania arytmetyczne i logiczne, porównania, wyrażenia z efektami ubocznymi (‘++’, ‘--’)
- Instrukcje if, if/else, while, “pascalowy” for
- Funkcje z parametrami przez wartość i zmienną, z rekurencją i zagnieżdżaniem (z przesłanianiem, z zachowaniem poprawności statycznego wiązania identyfikatorów)
- Obsługa dynamicznych błędów wykonania, wbudowane funkcje do wypisywania na wyjście, funkcje rzutujące między typami int i str

Zakres

Wszystkie (nie rozwijające się na listę podpunktów do wyboru) wymagania na 24 punkty

Z punktu 6. (3)

~~a) dwa sposoby przekazywania parametrów (przez zmienną / przez wartość)~~

b) pętla for w stylu Pascala

c) typ str, literały napisowe, wbudowane funkcje pozwalające na rzutowanie między napisami a liczbami

d) wyrażenia z efektami ubocznymi (przypisania, operatory języka C ++, += itd)

Z punktu 11. (3)

b) tablice indeksowane int lub coś à la listy

c) tablice/słowniki indeksowane dowolnymi porównywalnymi wartościami; typ klucza należy uwzględnić w typie słownika

e) operacje przerywające pętlę while - break i continue

Szczegóły konstrukcji rozszerzających Latte

- Array (indeksowany wartościami int)
  - Typ [typ_wartości]
  - Tworzenie array = []
  - Przypisanie array[indeks] = wartość
  - Dostęp array[indeks]
  - Funkcje length
- Dict
  - Typ {{typ_klucza, typ_wartości}}
  - Tworzenie dict = {{}}
  - Przypisanie dict{{klucz}} = wartość
  - Dostęp dict{{klucz}}
  - Funkcje has_key, delete_key

- “Pascalowy” for
    - Składnia for i = first..last do statement
    - Zarówno inkrementacyjny jak i dekrementacyjny
- Operacje przerywające pętlę while (oraz pętlę for)
    - break;
    - continue;
- Zagnieżdżanie funkcji