Tempest

Opis języka

Jakub Grondziowski
jg417886

Język Tempest jest językiem imperatywnym w pełni zainspirowany językiem latte. Ma standardową składnię i nie zawiera żadnych niestandardowych konstrukcji. Program napisany w języku Tempest powinien mieć rozszerzenie .tms. Pełny opis wybranych przeze mnie podpunktów do zrealizowania:

01. trzy typy:
    Tak jak w latte, język Tempest posiada trzy typy - int, boolean oraz string.

02. literały, arytmetyka, porównania:
    Żadnych zmian względem latte. Można ze sobą porównywać wszystkie trzy typy. Dodatkowo można konkatenować ze sobą typy string.

03. zmienne, przypisanie:
    Standardowa składnia postaci Typ Nazwa “=” Wyrażenie. W Tempeście nie ma możliwości deklarowania zmiennych bez podania ich wartości.

04. print:
    W Tempest istnieją trzy rodzaje printów: printString, printBool oraz printInt, każdy przewidziany na odpowiedni typ. Wszystkie te procedury są procedurami wbudowanymi.

05. while, if:
    Z uwagi na możliwe konflikty Tempest wymaga stosowania bloków zamiast instrukcji (blok to instrukcja otoczona nawiasami wąsatymi) zarówno w instrukcjach while, jak i w instrukcjach if (nie tyczy się warunków).

06. funkcje lub procedury, rekurencja:

    Standardowa składnia deklaracji funkcji postaci ZwracanyTyp Nazwa(ListaArgumentów) {Instrukcja}. ZwracanyTyp może być jednym z 3 dostępnych typów lub “void” dla procedur. ListaArgumentów to lista postaci Typ Nazwa, gdzie każdy element listy jest rodzielony przecinkami.

07. przez zmienną / przez wartość / ~~in/out~~:

    Przekazywanie argumentów do funkcji przez wartość to standardowa deklaracja funkcji opisana w 06. Aby przekazać argument przez zmienną, należy dodać między Typem a Nazwą argumentu znak “@” (w deklaracji funkcji).

08. –

09. przesłanianie i statyczne wiązanie:

    Standardowe.

10. obsługa błędów wykonania:

    Interpreter obsługuje trzy standardowe błędy wykonania - błąd arytmetyczny (dzielenie przez zero), błąd braku instrukcji powrotu (brak napotkania “return” po wyjściu z funkcji) oraz błąd spowodowany wywołaniem wbudowanej procedury error.

11. funkcje zwracające wartość:

    Opisane w 06.

12. statyczne typowanie:

    Standardowe, z Type Checkerem.

13. funkcje zagnieżdżone ze statycznym wiązaniem:

    Wewnątrz funkcji będzie możliwe tworzenie funkcji zagnieżdżonych o składni tej samej co ta opisana w 06.

Dodatkowo:

- Interpreter można uruchomić w dwóch trybach:
  W trybie czytania z wyjścia, gdzie interpreter będzie czytać podany przez użytkownika program aż do napotkania znaku końca linii (ctrl+d), po czym rozpocznie jego interpretację.
  W trybie czytania z pliku, wtedy interpreter przeczyta podany plik i rozpocznie jego interpretację.

- Interpreter obsługuje komentarze. Podając znak “//”, interpreter zignoruje wszystkie pozostałe znaki w tej linii. Podając znak “/*” interpreter zignoruje wszystkie następne znaki aż do napotkania znaku “*/”

- Oprócz trzech wbudowanych procedur printInt, printString i printBool interpreter obsługuje jeszcze jedną wbudowaną procedurę bez argumentową error, która kończy działanie programu, wyrzucając błąd wykonania informujący o wywołaniu tej funkcji.

- Każdy program musi się gdzieś zacząć, dlatego każdy program musi posiadać bezargumentową procedurę main. Brak procedury main lub błędne jej zdefiniowanie skutkuje błędem.

- Każda zadeklarowana funkcja ma swój zwracany typ. Tak jak w latte, każda funkcja musi posiadać instrukcję return, która zwróci wyrażenie odpowiedniego typu. W przypadku procedury wystarczy podać instrukcję return bez wyrażenia. Jeśli w czasie wykonania funkcja nie natrafi na returna, rzucany jest błąd wykonania.

- Dozwolona jest reinicjalizacja zmiennych w ramach jednego bloku - np. zmienna “x” może zostać na początku bloku zadeklarowana jako int, po czym zostać ponownie zadeklarowana z innym typem. Deklaracja z początku jest wtedy zapomniana, a zmienna “x” jest podlinkowana pod nową deklarację.