

Implementacja interpretera języka programowania ogólnego przeznaczenia z wbudowanym typem danych przedstawiającym datę.

Daniel Górniak

18 listopada 2021

1 Zasady działania języka.

- obsługuje liczby całkowite i ułamki
 - obsługuje operacje matematyczne o różnym priorytecie wykonania
 - obsługuje operacje logiczne i porównania o różnym priorytecie wykonania
- obsługuje typ znakowy
 - obsługuje konkatencje napisów tylko z innymi napisami
 - może zawierać dowolne znaki, też wyróżnik napisu (""')
- obsługa operatorów porównania dwóch dat
- typ timeDiff, powstający w wyniku odjęcia daty większej od mniejszej
- w języku tym będzie można pisać komentarze
- obsługuje tworzenie zmiennych, przypisywanie do nich wartości oraz je odczytywać
 - typowanie statyczne
 - typowanie słabe dla int i float
 - mutowalne
 - zmienne będą miały zakresy lokalne
- obsługiwana będzie pętla warunkowa if else
- obsługiwana będzie bazowa pętla while
- obsługiwana będzie możliwość wołania i definiowania własnych funkcji ze zmiennymi lokalnymi, gdzie argumenty będą przekazywane przez wartość
- obsługa rekursywnych wywołań funkcji
- wbudowana funkcja print przyjmująca jeden argument typu napis
- zmienna boolowska false zastąpiona jest zmienną int i float o wielkości 0, inne dają wynik true

2 Struktura projektu.

- projekt zostanie napisany w c++
- będzie to aplikacja okienkowa do której będzie się podawało skrypt, który będzie poddany interpretacji
- testowanie za pomocą testów jednostkowych z użyciem biblioteki Boost
- Moduły:

- moduł analizatora leksykalnego, czyta ciąg znaków i tworzy kolejne tokeny po prośbie analizatora składniowego; wykrywa nieprawidłowe tokeny i sygnalizuje to modułowi obsługi błędów
- moduł analizatora składniowego, prosi analizator leksykalny i tworzy z nich drzewa rozbioru, wykrywa błędy składniowe; wykrywa nieprawidłową składnię kolejnych tokenów i sygnalizuje to modułowi obsługi błędów
- moduł analizatora semantycznego, sprawdzając utworzone drzewa rozbioru sprawdza czy mają one sens, czy nie ma w nich błędów takich jak na przykład operator + dla niewłaściwych typów
- moduł obsługi błędów, przy nieudanym sparsowaniu ciągu znaków podaje kolejne wykryte błędy użytkownikowi
- tablica identyfikatorów razem z tymi zarezerwowanymi przez język i jej zarządzanie; używana przez moduły analizatora leksykalnego, składniowego oraz semantycznego

3 Gramatyka.

```

program          ::= ( functionDefinition | declaration ) *
functionDefinition ::= 'fun' type id '(' parameters ')' ':' body
parameters       ::= empty
                  | type id (',' type id) *
body             ::= statement +
statement        ::= assignment
                  | if
                  | while
                  | declaration
                  | return
                  | functionCall
assignment       ::= id '=' expression
return          ::= 'return' expression?
declaration      ::= type id
                  | type id '=' expression
if              ::= 'if' condition ':' body
                  | 'if' condition ':' body 'else:' body
while           ::= 'while' condition ':' body
condition        ::= relationalCondition (logicalOperator, relationalCondition) *
relationalCondition ::= basicCondition, (relationalOperator, basicCondition) *
basicCondition   ::= negCondition
                  | '(' condition ')'
                  | expression
negCondition     ::= '!' ( '(' condition ')' | expression )
expression       ::= advancedExpression ([+-] advancedExpression) *
advancedExpression ::= basicExpression ([*/] advancedExpression) *
basicExpression  ::= '"' allchar* '"'
                  | timeDiff
                  | date
                  | number
                  | '-' number
                  | id
                  | '-' id
                  | '(' expression ')'
                  | '-' '(' expression ')'
                  | functionCall
                  | '-' functionCall
functionCall     ::= id '(' arguments ')'
arguments        ::= empty
                  | expression (',' expression) *
id              ::= letter (digit | letter) *

```

```

number                ::= int
                        | float
int                    ::= nonzeroDigit digit*
                        | '0'
float                  ::= int '.' digit+
digit                  ::= nonzeroDigit
                        | '0'
nonzeroDigit           ::= [1-9]
relationOperator       ::= '=='
                        | '!='
                        | '<'
                        | '>'
                        | '>='
                        | '<='
logicalOperator        ::= 'and'
                        | 'or'
letter                 ::= [A-Za-z_]
timeDiff               ::= {int'y'int'm'int'd'}
type                   ::= 'int'
                        | 'float'
                        | 'date'
                        | 'timeDiff'
                        | 'string'
allchar                ::= ?all characters?
date                   ::= '[' [0-9]{4}':' [0-9]{2} ':' [0-9]{2} ']'
empty                  ::= ''

```

4 Przykład kodu.

```

1)
fun int start():
    string napis = "A"
    if napis == "A":
        print("Tak")
    else:
        int zmienna = 1
        while(zmienna < 5):
            print(i)
            zmienna++
    return 0

2)
fun int eatNumber(int x):
    int result
    if(x > 0):
        result = eatNumber(x-1)
    return result+1
fun int start():
    return eatNumber(5)

3)
fun int start():
    date a = (1999:05:05)
    if (2000:01:01) - a > 0y1m0d:
        print('Większa')
    else:
        print('Mniejsza')
    return 1

```

