

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

Daniel Górniak

16 stycznia 2022

1 Opis języka

Język ten czerpie zarówno z Pythona jak i z C++. Zakresy zmiennych i przynależność instrukcji do bloków i funkcji określa wcięcie typowo widziane w takich językach jak python. Dzięki temu pozbywamy się niepotrzebnych nawiasów.

Z drgiej strony zmiennym należy określić typ przy definicji z uwagi na większą czytelność kodu, ponieważ wiesz po przeczytaniu go co się w niej znajduje.

2 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 (""")
- obsługuje typ bool
- 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

3 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 o kolejne tokeny 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, wykonuje tak dostarczony program
 - 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

4 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 ('else:' body)?
while            ::= 'while' condition ':' body
condition        ::= relationalCondition (logicalOperator, relationalCondition)*
relationalCondition ::= basicCondition, (relationalOperator, basicCondition)*
basicCondition   ::= negCondition
                  | '(' condition ')'
negCondition     ::= '!' ( '(' condition ')' | expression)
expression       ::= advancedExpression ([+-] advancedExpression)*
advancedExpression ::= basicExpression ([*/] advancedExpression)*
basicExpression  ::= '"' allchar* '"'
                  | timeDiff
                  | date
                  | '-'? number
                  | bool
                  | '-'? id
```

```

        | '-'? '(' expression ')'
        | '-'? functionCall
functionCall ::= id '(' arguments ')'
arguments   ::= empty
              | expression (',' expression)*
id          ::= letter (digit | letter)*
number      ::= int
              | float
bool        ::= 'True'
              | 'False'
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'
                  | 'bool'
allchar         ::= ?all characters?
date            ::= '[' [0-9]{4}':' [0-9]{2} ':' [0-9]{2} ','
empty           ::= ''

```

5 Przykład kodu.

```

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

```

6 Kompilacja projektu

6.1 Kompilacja

Żeby skompilować projekt należy uruchomić skrypt `make.sh`, który znajduje się w głównym folderze projektu.

6.2 Uruchomienie

Aby uruchomić uprzednio zbudowany projekt należy wejść do podkatalogu `build` i uruchomić powstały program z terminalu. Plikiem wejściowym jest plik `input.txt` znajdujący się podkatalog wyżej.

7 Instrukcja użycia

7.1 zmienne

Zmienne można deklarować zarówno globalnie jak i w funkcjach. Deklaracja zmiennych wygląda następująco. `<typ> <nazwa> = <wyrażenie arytmetyczne>` Dostępne typy zmiennych to : `int`, `float`, `string`, `bool`, `date`, `timeDiff` Przy deklaracji zmiennych nie trzeba od razu podawać wartości (można ominąć `= <wyrażenie arytmetyczne>`). Przykłady:

- `int a = 1`
- `float b = (1.1 + 2)*3`
- `string c = "napis"`
- `bool g = True`
- `date d = [2020:12:12]`
- `timeDiff e = {1y1m1d}`

W napisie może też wystąpić znak `"` poprzez poprzedzenie go znakiem `\`, jak np. `"przy\"klad"`. Przy różnicy czasu zawsze należy podawać liczbę lat, miesięcy i dni, nawet jeżeli wynosi ona 0.

7.2 wyrażenia arytmetyczne

Na zmiennych typu `int` oraz `float` można wykonywać podstawowe operacje: dodawanie, odejmowanie, mnożenie, dzielenie

Zachowana jest prawidłowa kolejność wykonywania działań, tzn. najpierw mnożenie i dzielenie, a następnie dodawanie i odejmowanie. Kolejność wykonywania działań można zmienić za pomocą nawiasów.

Zmienną typu `string` można konkatelować z dowolnym inną zmienną, np. dodając do niej zmienną typu `date`. `string a = "teraz jest: "+[2022:01:16]`

Od daty można odjąć datę aby otrzymać różnicę czasu, przy czym odejmowana data musi być mniejszą datą. Od daty można też odjąć różnicę czasu aby otrzymać datę mniejszą.

7.3 porównania i wyrażenia logiczne

W języku mamy podstawowe operatory porównania "==" - równe, "!=" - różne, "<" - mniejsze, ">" - większe, "<=" - nie większe, ">=" - nie mniejsze.

Język udostępnia klasyczne operatory logiczne: negacja "!", koniunkcja "and", oraz alternatywa "or". Można je stosować tylko, kiedy lewa i prawa strona są typu bool.

7.4 instrukcja warunkowa if

Instrukcja if jest tutaj typową instrukcją tego typu dla języków programowania.

```
if True:
    return 1
```

Instrukcja else jest opcjonalna i musi się ona znajdować bezpośrednio po zakończonym bloku instrukcji if.

```
if True:
    return 1
else:
    return 2
```

Warunek w instrukcji if może przyjmować bardziej złożony wygląd. Można korzystać zarówno z wyrażeń logicznych, porównań jak i wyrażeń arytmetycznych. Np:

```
if (1 - 2)*2 > 1 and !([2022:10:02] - [2000:01:01] > {2y0m0d}):
    return 1
```

7.5 pętla while

Tak samo jak instrukcja if, jest to typowa konstrukcja pętli while w językach programowania.

```
int a = 5
while a > 1:
    a = a - 1
```

7.6 funkcje

Użytkownik może tworzyć własne funkcje. Muszą się one rozpocząć słówkiem "fun" oraz mieć zerowe wcięcie w dokumencie. Funkcja "main" nie może mieć parametrów. Każda funkcja musi zwracać zadeklarowany typ zmiennej. Parametry funkcji podajemy po przecinku razem z ich typem, np: int a.

Funkcje przyjmują argumenty jako kopie wartości, dlatego nie są one nadpisywane poza funkcją do której zostały one podane.

```
fun <typ> <nazwa> (<parametry>) :
    ciało funkcji
```

7.7 funkcja wbudowana

Dla użytkownika dostępna jest funkcja wbudowana "print". Przyjmuje ona jeden argument typu string, który wypisuje na ekran. W celu podania na wyjście innych typów zmiennych niż string należy wymusić zmianę typu, poprzez konkatencję z pustym stringiem.

```
int a = 1
print("" + a)
```