

# Interpreter własnego języka Cute



Akademia Górniczo-Hutnicza w Krakowie

Wydział Informatyki, Elektroniki i Telekomunikacji

Projekt wykonany na przedmiot Metody i Algorytmy Kompilacji

Martyna Olszewska

[martynao@student.agh.edu.pl](mailto:martynao@student.agh.edu.pl)

Jakub Domogała

[domogala@student.agh.edu.pl](mailto:domogala@student.agh.edu.pl)

2022/2023

|   |           |
|---|-----------|
| <b>1. Wstęp</b>                                   | <b>3</b>  |
| <b>2. Wybór architektury aplikacji</b>            | <b>3</b>  |
| <b>3. Gramatyka</b>                               | <b>3</b>  |
| <b>4. Produkcje</b>                               | <b>5</b>  |
| <b>5. Opis zasad</b>                              | <b>6</b>  |
| 5.1. Zmienne                                      | 6         |
| 5.2. Operacje arytmetyczne                        | 6         |
| 5.3. Operacje logiczne                            | 7         |
| 5.4. Bloki  | 7         |
| 5.5. Pętla oraz if.                               | 7         |
| 5.6. Print  | 8         |
| 5.7. Komentarze                                   | 8         |
| <b>6. Testowanie i sprawdzenie poprawności</b>    | <b>8</b>  |
| 6.1. Ciąg Fibonacciego                            | 8         |
| 6.2. LoopGame                                     | 9         |
| 6.3. Test operacji matematycznych oraz logicznych | 10        |
| <b>7. Instrukcja instalacji i użytkowania</b>     | <b>11</b> |

## 1. Wstęp

Celem naszego projektu było stworzenie Interpretera do języka Cute, który został wymyślony przez nas samych. Język ten nie jest przeznaczony do użytku na co dzień. Jego składnia jest interesująca i dosyć trudna, więc nie byłby on przeznaczony do codziennego użytkowania na większą skalę. Można go traktować jako ciekawostkę. Wynikiem działania naszego programu jest interpretacja kodu zapisanego w języku .cute

## 2. Wybór architektury aplikacji

Do stworzenia parsera wybraliśmy generator ANTLR4. Generowane pliki są w języku Python.

## 3. Gramatyka

Gramatyka naszego języka łączy elementy języka Python i Haskell.

Każda linia musi być zakończona znakiem |<3|. Tabulatory, znaki puste są domyślnie pomijane.

Poniżej znajduje się lista słów kluczowych wraz z ich znaczeniem.

| Słowo kluczowe w naszym języku | Odpowiednik domyślny |
|--------------------------------|----------------------|
| drukareczka                    | print                |
| waruneczek                     | if                   |
| powielanko                     | while                |
| zwrocik                        | return               |
| bezprzecinek                   | Int                  |
| zerojedynek                    | Bool                 |
| zprzecinek                     | Double               |
| prawda                         | true                 |
| nieprawda                      | false                |
| oraz                           | and                  |

|               |    |
|---------------|----|
| lub           | or |
| kropkawkropke | == |
| innyod        | != |
| mniejszyod    | <  |
| wiekszyod     | >  |

Tabela ze znakami zarezerwowanymi:

|    |                                 |
|----|---------------------------------|
| +  | dodawanie                       |
| -  | odejmowanie                     |
| *  | mnożenie                        |
| /  | dzielenie                       |
| /  | dzielenie całkowitoliczbowe     |
| *  | potęgowanie                     |
| %  | modulo                          |
| ^  | max                             |
| v  | min                             |
| {  | początek bloku                  |
| }  | koniec bloku                    |
| () | priorytet wyrażeń               |
| -> | definicja zmiennej              |
| <- | przypisanie wartości zmiennej   |
| <3 | znak końca linii oraz komentarz |

Inne tokeny:

|            |                          |
|------------|--------------------------|
| NAME       | [a-zA-Z_-][a-zA-Z0-9_-]* |
| INT        | (-)?[0-9][0-9]*          |
| WHITE_SIGN | (' ' '\n' '\r' '\t')+    |
| COMMENT    | ' <3 ' ~[\r\n]*          |

## 4. Produkcje

```
program      : block;

block        : (stat)+;

// All Statements
stat         : define_stat
             | assign_stat
             | print_stat
             | if_stat
             | while_stat;

// Create new Variable statement
define_stat  : TYPE Var_define NAME Semicolon
             | TYPE Var_define NAME Val_assign value=expr Semicolon
             ;

// Assign value to Variable
assign_stat  : NAME Val_assign value=expr Semicolon;

// Print variable or sth
print_stat   : Print Open_Parenthesis valorname=term Close_Parenthesis
Semicolon;
```

```

// If statement
if_stat      : If Open_Parenthesis valorname=expr Close_Parenthesis
              (Open_Bracket block Close_Bracket);

// While statement
while_stat   : While Open_Parenthesis valorname=expr Close_Parenthesis
              (Open_Bracket block Close_Bracket);

// All expressions
expr         : left=expr Operator_sign right=expr           # operat
              | Open_Parenthesis mid=expr Close_Parenthesis # parentise
              | term                                         # terminal
              | Operator_sign mid=expr                      # negate
              ;

// All terms ( identifiers and s )
term         : NAME          #TermName
              | Int          #TermInt
              | Double       #TermDouble
              | Bool         #TermBool
              ;

```

## 5. Opis zasad

### 5.1. Zmienne

Język jest typowany, więc potrzebujemy deklaracji każdej zmiennej przed jej wywołaniem. Mamy dostępne 3 typy zmiennych: *zerojedynek* - wartość boolowska, *bezprzecinek* - liczba całkowita i *zprzecinek* - liczba wymierna. Wartości liczbowe mogą być dodatnie jak i ujemne. Przy deklaracji zmiennej musimy użyć między jej typem, a nazwą zmiennej symbolu `->`, a jeśli chcemy przypisać do niej wartość to możemy to zrobić po symbolu `<-`. Oczywiście nie zapominamy o znaku końca linii. W nazwie zmiennej może występować symbol `"_"`, ale nie może znajdować się na początku.

```

bezprzecinek -> count <- 0 |<3|
zerojedynek  -> flag_to_Print |<3|

```

## 5.2. Operacje arytmetyczne

W naszym języku mamy dostępne operacje arytmetyczne: dodawanie, odejmowanie, mnożenie i dzielenie. Ze znakami odpowiednio: +, -, \* i /. Postanowiliśmy wykonywać operacje tylko na jeśli typy są takie same, więc nie jest możliwe dodanie *bezprzecinka* do *zprzecinka*.

Dostępna jest również standardowa operacja modulo - `/%`, dzielenie całkowitoliczbowe - `//`, maximum - `/^/` i minimum - `/v/` dwóch liczb oraz potęgowanie - `/**/`.

```
bezprzecinek -> nazwazmiennej <- 5 |^| 1 |^| 10 |<3|
count <- (5 + 3) |%| (4 |v| 1) |<3|
```

## 5.3. Operacje logiczne

Dostępne są dwie operacje logiczne oraz i lub. Możemy porównywać wartości, słowem *kropkawkropke* sprawdzamy czy dane wartości są takie same. Słowem *innyod* sprawdzamy czy wartości są od siebie różne. Możemy również sprawdzić która z wartości jest większa lub mniejsza, słowami *wiekszyod* i *mniejszyod*. Możemy użyć również zaprzeczenia wartości logicznej lub wyrażenia przez zastosowanie symbolu *not* przed.

```
var <- count mniejszyod amount_to_print |<3|
val <- 5 kropkawkropke 3 |<3|
```

## 5.4. Bloki

Ograniczone są w nawiasach "{ }". Wewnątrz takiego bloku możemy wykonywać pętle, deklarować zmienne, przypisywać wartości, sprawdzać warunki oraz wypisywać wartości. Bloki mogą być zagnieżdżane w sobie.

```
powielanko(a1 mniejszyod 10) {
  warunekczek(1 kropkawkropke nieprawda) {
    drukareczka(a1) |<3|
  }
}
```

## 5.5. Pętla oraz if.

Możemy wywołać pętlę *powielanko* z warunkiem logicznym w nawiasie "( )". Po warunku następuje instrukcja w bloku. Podobnie możemy sprawdzać warunki

poprzez słowo *warunek*. Po nim znajduje się warunek w nawiasie “( )” oraz instrukcja w bloku.

```
powielanko(a1 mniejszyod 10) {  
  . . .  
}  
  
warunek(1 kropkawkropke nieprawda) {  
  . . .  
}
```

## 5.6. Print

Dostępne jest wypisywanie wartości na konsolę poprzez słowo *drukareczka*. Wartość którą chcemy wypisać musi być w nawiasie “( )”. Jeśli chcemy wypisać wartość zmiennej wcześniej zadeklarowanej podajemy jej nazwę w nawiasie w wyniku dostaniemy również informacje o jej typie.

```
drukareczka(a1) |<3|
```

## 5.7. Komentarze

Możemy pisać komentarze jednolinijkowe poprzez wystąpienie symbolu `|<3|` na początku linii. Po tym symbolu może występować dowolny ciąg.

```
|<3| to jest komentarz
```

# 6. Testowanie i sprawdzenie poprawności

Aby przetestować poprawność działania naszego interpretera stworzyliśmy kilka przykładowych plików z kodem.

## 6.1. Ciąg Fibonacciego

Poniżej znajduje się zaimplementowany algorytm do wypisywania 20 pierwszych wyrazów ciągu Fibonacciego w naszym języku.

```
bezprzecinek -> amount_to_print <- 20 |<3|  
bezprzecinek -> count <- 0 |<3|  
bezprzecinek -> n1 <- 0 |<3|  
bezprzecinek -> n2 <- 1 |<3|
```



```

bezp przecinek -> n3 |<3|

powielanko(count mniejszyod amount_to_print) {
  drukareczka(n1) |<3|
  n3 <- n1 + n2 |<3|
  n1 <- n2 |<3|
  n2 <- n3 |<3|
  count <- count + 1 |<3|
}

```

Wynik powyższego programu jest następujący:

|                      |                            |
|----------------------|----------------------------|
| 😎 n1 ma wartość 0    | i jest typu bezprzecinek 😊 |
| 😊 n1 ma wartość 1    | i jest typu bezprzecinek 😊 |
| 😎 n1 ma wartość 1    | i jest typu bezprzecinek 😊 |
| 😊 n1 ma wartość 2    | i jest typu bezprzecinek 😊 |
| 😊 n1 ma wartość 3    | i jest typu bezprzecinek 😊 |
| 😎 n1 ma wartość 5    | i jest typu bezprzecinek 😊 |
| 😊 n1 ma wartość 8    | i jest typu bezprzecinek 😊 |
| 😊 n1 ma wartość 13   | i jest typu bezprzecinek 😊 |
| 😊 n1 ma wartość 21   | i jest typu bezprzecinek 😊 |
| 😊 n1 ma wartość 34   | i jest typu bezprzecinek 😊 |
| 😊 n1 ma wartość 55   | i jest typu bezprzecinek 😊 |
| 😍 n1 ma wartość 89   | i jest typu bezprzecinek 😊 |
| 😊 n1 ma wartość 144  | i jest typu bezprzecinek 😊 |
| 😊 n1 ma wartość 233  | i jest typu bezprzecinek 😊 |
| 😊 n1 ma wartość 377  | i jest typu bezprzecinek 😊 |
| 😎 n1 ma wartość 610  | i jest typu bezprzecinek 😊 |
| 😊 n1 ma wartość 987  | i jest typu bezprzecinek 😊 |
| 😊 n1 ma wartość 1597 | i jest typu bezprzecinek 😊 |
| 😊 n1 ma wartość 2584 | i jest typu bezprzecinek 😊 |
| 🔧 n1 ma wartość 4181 | i jest typu bezprzecinek 🔧 |

Dostaliśmy pierwsze dwadzieścia wyrazów ciągu Fibonacciego i do tego nawet poprawnych. Zatem możemy stwierdzić, że nasz interpreter działa poprawnie.

## 6.2. LoopGame

Poniższy algorytm wypisuje w pętli wartość aktualnego poziomu, który za każdym razem jest zwiększany o jeden, ale jest printowany tylko jeśli poziom jest parzysty.

```

bezp przecinek -> current_level <- 0 |<3|
bezp przecinek -> final_level <- 11 |<3|

```

```

zerojedynka -> game_completed <- prawda |<3|

powielanka(current_level mniejszyod final_level) {
  warunek((current_level % 2 ) kropkawkropka 0) {
    drukareczka(current_level) |<3|
  }
  current_level <- current_level + 1 |<3|
}

```

|                               |                            |
|-------------------------------|----------------------------|
| 😊 current_level ma wartość 0  | i jest typu bezprzecinek 😊 |
| 😊 current_level ma wartość 2  | i jest typu bezprzecinek 😊 |
| 😊 current_level ma wartość 4  | i jest typu bezprzecinek 😊 |
| 😊 current_level ma wartość 6  | i jest typu bezprzecinek 😊 |
| 😊 current_level ma wartość 8  | i jest typu bezprzecinek 😊 |
| 😊 current_level ma wartość 10 | i jest typu bezprzecinek 😊 |

Jak widać tutaj również dostaliśmy prawidłowy wynik.

### 6.3. Test operacji matematycznych oraz logicznych

Przetestowaliśmy również poprawność działania operacji matematycznych oraz logicznych.

```

bezprzecinek -> a <- 5 |<3|
bezprzecinek -> b <- 20 |<3|

bezprzecinek -> wynik <- a |v| b |<3|
drukareczka(wynik) |<3|

wynik <- a |^| b |<3|
drukareczka(wynik) |<3|

wynik <- a % b |<3|
drukareczka(wynik) |<3|

wynik <- b || a |<3|
drukareczka(wynik) |<3|

wynik <- b |*| a |<3|
drukareczka(wynik) |<3|

zerojedynka -> val <- b kropkawkropka a |<3|
drukareczka(val) |<3|

```

```
val <- b innyod a |<3|  
drukareczka(val) |<3|
```

Wynik jest następujący

😁 wynik ma wartość 5 i jest typu bezprzecinek 😊  
😁 wynik ma wartość 20 i jest typu bezprzecinek 🥰  
😁 wynik ma wartość 5 i jest typu bezprzecinek 😎  
😁 wynik ma wartość 4 i jest typu bezprzecinek 😊  
😁 wynik ma wartość 3200000 i jest typu bezprzecinek 🥰  
😁 val ma wartość nieprawda i jest typu zerojedynek 😊  
😁 val ma wartość prawda i jest typu zerojedynek 😊

## 7. Instrukcja instalacji i użytkowania

Aby uruchomić interpreter dla zadanego pliku należy pobrać [repozytorium z projektem](#). Następnie w folderze z pobranym projektem uruchamiamy terminal i wpisujemy komendę: `python main.py ./nazwaPliku.cute` - Gdzie nazwaPliku to plik z kodem napisanym w języku Cute