Stanisław Czobot

295797

TKOM - Dokumentacja końcowa

Język do operacji na walutach

1. Zmiany w gramatyce względem projektu wstępnego.

Podczas budowy parsera została przerobiona gramatyka przedstawiona w projekcie wstępnym, ponieważ wcześniejsza nie wspierała poprawnej kolejności operatorów.

Aktualna gramatyka:

program = { function };

function = signature parameters code_block;

signature = data_type id;

parameters = ‘(‘ [signature { ‘,‘ signature }*] ‘)’;

code_block = ‘{‘ statement [statement]* ‘}’;

statement = function_call | declaration | if_statement | while_statement | return_statement | assign_statement | print_statement;

declaration = data_type id [‘=’ expression][‘,‘ id [‘=’ expression]]*;

expression = priority_operation [low_priority_operator priority_operation]*;

priority_operation = base_expression [high_priority_operator base_expression]*;

base_expression = value_instance | higher_expression;

higher_expression = ‘(’ expression ‘)’;

low_priority_operator = ‘+’ | ‘-’ ;

high_priority_operator = ‘*’ | ‘/’;

value_instance = id | value | funcion_call;

if_statement = ‘if’ ‘(’ condition ‘)’ single_or_block;

while_statement = ‘while’ ‘(’ condition ‘)’ single_or_block;

single_or_block = statement ’;’ | code_block;

condition = condition_operation [condition_linker condition_operation]*;

condition_operation = (expression condition_operator expression) | bool_value | expression;

condition_operator = ‘==’ | ‘!=’ | ‘<’ | ‘>’ | ‘<=’ | ‘>=’;

condition_linker = ‘&&’ | ‘||’;

function_call = id ‘(’ call_elements ‘)’;

call_elements = expression [‘,’ expression]*;

data_type = ‘int” | ‘double’ | ‘bool’ | ‘PLN’ | ‘USD’ | ‘EUR’ | ‘GBP’ | ‘CHF’ | ‘RUB’ | user_currency;

user_currency = upper_case_letter upper_case_letter upper_case_letter;

bool_value = ‘true’ | ‘false’;

value = number | ‘true’ | ‘false’;

number = not_zero_digit [digit];

not_zero_digit = ‘1’ | ‘2’ | ‘3’ | ‘4’ | 5’ | 6’ | ‘7’ | ‘8’ | ‘9’;

digit = ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | 5’ | 6’ | ‘7’ | ‘8’ | ‘9’;

id = letter { digit | letter };

letter = upper_case_letter | lower_case_letter;

upper_case_letter = | ‘A’ | ‘B’ | ‘C’ | ‘D’ | ‘E’ | ‘F’ | ‘G’ | ‘H’ | ‘I’ | ‘J’ | ‘K’ | ‘L’ | ‘M’ | ‘N’ | ‘O’ | ‘P’ | ‘R’ | ‘S’ | ‘T’ | ‘U’ | ‘V’ | ‘w’ | ‘X’ | ‘Y’ | ‘Z’;

lower_case_letter = ‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘g’ | ‘h’ | ‘i’ | ‘j’ | ‘k’ | ‘l’ | ‘m’ | ‘n’ | ‘o’ | ‘p’ | ‘r’ | ‘s’ | ‘t’ | ‘u’ | ‘v’ | ‘w’ | ‘x’ | ‘y’ | ‘z’;

Dodatkowo względem projektu wstępnego nie została zaimplementowana funkcjonalość: amount(currency), która pierwotnie miała zwracać ilość waluty podanego argumentu. Problemem w implementacji okazało się niepoprawne dopasowanie tej funkcjonalności jako amount_statement. W projekcie wszystkie statement można traktować jako oddzielny blok kodu, ale niezaimplementowana funkcjonalność powinna zachowywać się jako wywołanie funkcji, np. aby zwróconą wartość użyć do obliczeń. W kolejnym kroku spróbowałem dodać tą funkcjonalość jako dodatkową funkcję środowiska wykonania programu, niestety z negatywnym skutkiem.

2. **Tutorial języka**
    a. Główny funkcja programu, w jej środku kolejno wykonują się wszystkie zapisane komendy:

```
int main() {
    ...
    return 0;
}
```

    b. Dostępne typy danych:

        int - liczby całkowitoliczbowe

        double - liczby zmiennoprzecinkowe

        bool - wartość binarna true/false

    Oraz wszystkie walutowe typy danych zdefiniowane w podanym pliku wejściowym.

    c. Definiowanie funkcji:

```
bool myFunction() {
    return true;
}

int dodawanie(int a, int b) {
    return a + b;
}
```

    Definiować funkcje można powyżej oraz poniżej funkcji main ważne, żeby znajdowały się w tym samym pliku.

    d. Definiowane zmiennych:

```
bool warunek;
int liczba;
PLN zlotowki;
```

    e. Przypisywanie wartości zmiennym:

```
liczba = 1;
liczbaZmiennoprzecinkowa = 2.5;
warunek = false;
zlotowki = 5;
zlotowki = 5.5;
zlotowki = zlotowki + 1;
```

Przypisanie wartości zmiennej można połączyć z jej definiowaniem:

```
int liczba = 1;
```

Wszystkie rzutowania zmiennych typów walutowych dokonują się automatycznie, np.:

```
EUR euro = 1;
PLN zloty = 1;
USD dolary = euro + zloty;
```

Wynikiem takiej operacji zostanie zmiennej dolary zostanie przypisana wartość odpowiednia przeliczonym wartościom zmiennych euro oraz zloty na typ USD, w tym przypadku wyniósłby on odrobinę ponad 1.

f. Funkcja print:

Służy do wypisywania komunikatów, może przyjmować wiele argumentów, oddzielonych przecinkami. Argumenty mogą być różnych typów, ponieważ każdy z nich ma zaimplementowaną metodę toString().

```
int x= 10;
print("Zmienna x ma wartosc: ", x);
USD dolar = 1;
print(dolar);
```

g. Warunki logiczne:

Warunki logiczne można tworzyć z:

- Wartości typu bool
- Porównania zmiennych (za pomocą operatorów porównania: ==, <=, >=, !=, >, <)
- Negacji warunku (operatorem !)
- Połączenia kilku warunków (operatorem || - odpowiednik logicznego lub, && - odpowiednikiem logicznego i)

```
true || false
1 >= 0 && zmienna == 3
!warunek
```

h. Komenda warunkowa if / if...else:

```
int zero = 0;
if(zero == 0)
    print("Zero jest równe zero");
```

```
else
  print("Ten tekst się nie wyświetli");
```

i. Pętla warunkowa while:

```
int licznik = 5;
while(licznik > 0) {
  licznik = licznik - 1;
}
```

j. Wywoływanie funkcji:

```
double odejmij(double a, int b) {
  return a - b;
}
int a = 0;
double b = 1.5;
double wynik = odejmij(a, b);
```

Specjalnym przypadkiem wywoływania funkcji są wszystkie funkcje, które zwracają bądź mają za argument zmienną typu walutowego. Dzięki automatycznemu rzutowaniu użytkownik nie musi się przejmować jakiego typu dane podaje oraz jakiego typu dane funkcja zwraca, ponieważ wartości liczbowe za każdym razem przeliczane są według dostępnych przeliczników i końcowo dopasowywane do docelowego typu.

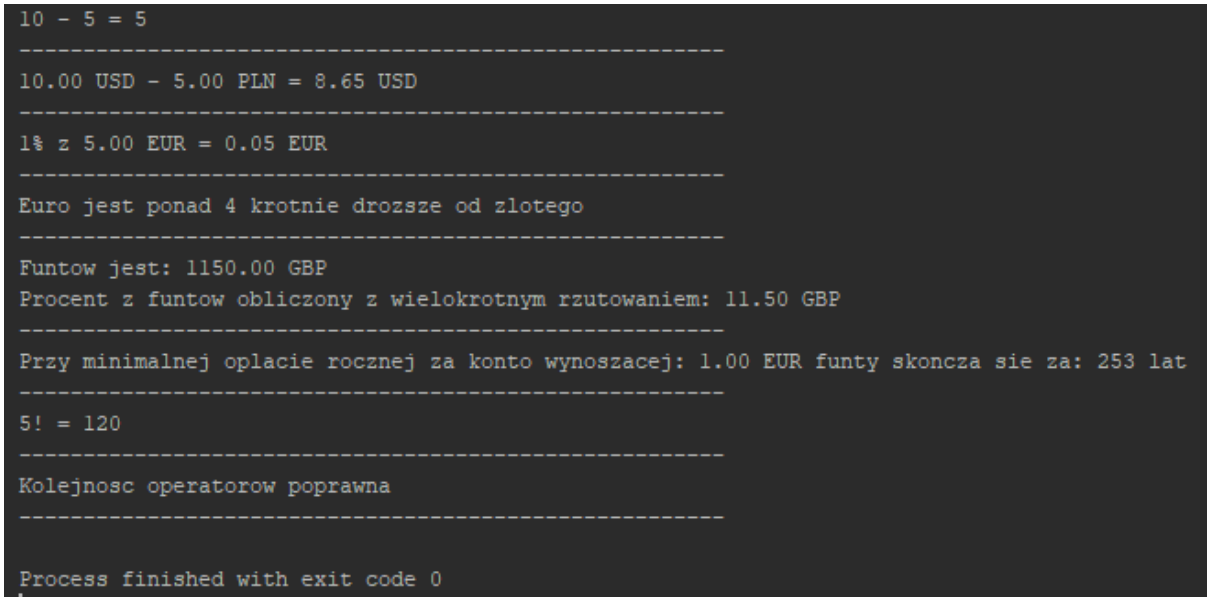
```
EUR procent(PLN argument) {    return argument * 0.01;}
GBP funt = 1;
RUB rubel = procent(funt);
```

3. Przykładowy program

```
int main() {
    int a = 10;
    int b = 5;
    int c = sub(a, b);
  print(a, " - ", b, " = ", c);
  print("-----------------------------------------------");

  USD dollars = 10;
  PLN polishZloty = 5;
  print(dollars, " - ", polishZloty, " = ", dollars-polishZloty);
  print(" -----------------------------------------------");

  EUR euro = 5;
  EUR currency;    currency = procent(euro);
  print("1% z ", euro," = ", currency);
```

```
  print(" ------------------------------------------------");

  EUR jedenEuro = 1;
  PLN jedenZloty = 1;
  if(jedenEuro > jedenZloty * 4)
    print("Euro jest ponad 4 krotnie drozsze od zlotego");
  print(" ------------------------------------------------");

  EUR minOplata = 1;
  GBP funt = 1150;
    int yearCounter = 0;
    print("Funtow jest: ", funt);
    print("Procent z funtow obliczony z wielokrotnym rzutowaniem:
", procent(funt));
  print(" ------------------------------------------------")

  while(procent(funt) > minOplata) {
  funt = funt - procent(funt);
        yearCounter = yearCounter + 1;
    }
    print("Przy minimalnej oplacie rocznej za konto wynoszacej: ",
minOplata, " funty skoncza sie za: ", yearCounter, " lat");
  print(" ------------------------------------------------");

  print("5! = ", silnia(5));
  print(" ------------------------------------------------");

  int sprawdzeniePoprawnosciKolejnosci = 2 + 2 * 2 - 2 / 2;
    int wynik = 5;
  if(sprawdzeniePoprawnosciKolejnosci == wynik)
    print("Kolejnosc operatorow poprawna");
  else
    print("Niepoprawna kolejnosc =operatorow");
    print(" ------------------------------------------------")

return 0;
}

int sub(int a, int b) {    return a - b;}

EUR procent(EUR e) {    return e * 0.01;}

int silnia(int x) {
    if(x == 1)
    return 1;
  else
     return x * silnia(x-1);
}
```

Efekt jego działania:



4. Ograniczenia języka

Najważniejsza zaleta języka jest również jego największą wadą, mianowicie rzutowanie. Jednym z założeń podczas budowy języka było, że walutowe typy danych będą miały tylko dwa miejsca po przecinku, zatem każde rzutowanie wymaga zastosowania przybliżenia (dla wartości z zakresu 0.005(włącznie) do 0.0099... przybliżenie wynosi 0.01, dla wartości z zakresu od 0.000(włącznie) do 0.0049... przybliżenie wynosi 0.00) do dwóch miejsc po przecinku. W skutek tego wielokrotne rzutowanie tej samej wartości może doprowadzić do jej zmiany, dlatego zalecane byłoby używanie tej funkcjonalności tylko w koniecznych przypadkach, np.: do tworzenia tylko jednej funkcji dla wielu typów walutowych, natomiast użytkownik podczas operacjach na typach walutowych powinien być świadomy zagrożenia.

5. Instrukcja obsługi projektu

Abu uruchomić projekt należy:

- Sklonować repozytorium gitlab : SCzobot_tkom_currency_lang, komendą:

```
git clone https://gitlab-stud.elka.pw.edu.pl/sczobot/tkom_currency_lang
```

- Wejść do folderu komendą:

```
Cd tkom_currency_lang/src
```

- Skompilować klasę Main komendą:

```
javac -d . Main.java
```

- Uruchomić skompilowaną klasę komendą:

```
java Main <ścieżka do pliku z programem> <ścieżka do pliku z macierzą walut>
```

W przypadku przykładowego programu i macierzy walut będzie to komenda:

```
java Main ../inputs/program.txt ../inputs/currencies.txt
```