

[TKOM] Interpreter prostego języka

Bartosz Świtalski

kwiecień 2021

Opis funkcjonalny

Celem projektu jest stworzenie interpretera prostego języka, umożliwiającego operacje na zmiennych typu `date` oraz `time`. W ramach projektu stworzę interpreter języka przypominającego [C](#), który zostanie wzbogacony o dwa dodatkowe typy danych i będzie umożliwiał proste operacje na zmiennych tych typów.

Elementy języka

- typ daty
- typ czasu
- typ numeryczny do reprezentacji liczbowej
- operatory arytmetyczne
- operatory logiczne (`&&`, `||`, `!`)
- operatory relacji (`<`, `>`, `==`, `!=`)
- arytmetyka, nawiasowanie
- instrukcje warunkowe (`if`, `else`)
- pętla `while`
- wbudowana funkcja `print()`

Szczegóły języka

- Interpreter będzie wykrywał błędy i informował o ich wystąpieniu użytkownika
- Zostaną zaimplementowane następujące typy zmiennych:
 1. typ numeryczny - `num`
 2. typ daty - `date`
 3. typ czasu - `time`
- W kodzie programu musi wystąpić pojedyncza funkcja `main`, która jest typu `void` i nie przyjmuje żadnych argumentów i jest punktem wyjścia dla wykonania się programu
- Język umożliwia pisanie własnych funkcji, które mogą zwracać jeden z trzech implementowanych typów lub żaden z nich (`void`)
- Język umożliwia dodawanie komentarzy po znaku `//` - każdy znak, występujący po `#`, aż do znaku nowej linii, jest traktowany jako komentarz
- Każda zdefiniowana zmienna ma swój zasięg i nie jest widoczna poza nim (funkcja lub pętla)

Operacje arytmetyczne

- Dla zmiennych numerycznych dozwolone są standardowe operacje dodawania, odejmowania, mnożenia oraz dzielenia
- Zmienne typu `date` oraz `time` **nie mogą** być elementami mnożenia ani dzielenia
- Język będzie umożliwiał określone operacje na implementowanych typach:
 1. `date_1 - date_2 = time`
 2. `date_1 + time = date_2`
 3. `date_1 - time = date_2`
 4. `time_1 + time_2 = time_3`
 5. `time_1 - time_2 = time_3`

Operatory relacji

- Dla zmiennych numerycznych dozwolone są standardowe operacje porównania
- Dla implementowanych typów także dozwolone są standardowe operacje porównania - będzie można porównywać daty oraz czasy

Przykłady języka

1

```
num func(num a) {
    if (a < 10) {
        return a;
    }
    else {
        return 10;
    }
}

void main() {
    num result = func(10.01);
    print("result: ", result);
}
// output: result: 10
```

2

```
void func(date d) {
    if (!(d < 01.01.2000.00:00:00)) {
        print("not 1999");
    }
}

void main() {
    date d = 01.01.2021.15:25:59;
    func(d);
}
// utput: not 1999
```

3

```
// function
date change(date d, time t){
    num i = 0;
    while (i < 5) {
        d = d + t;
        i = i + 1;
    }
    return d;
}

// main
void main() {
    date now = 01.01.2021.23:59:59;
    time add = 24:00:00;

    date later = change(now, add);
    print("after change: ", later);
}
// output: after change: 06.01.2021.23:59:59
```

4

```
time dif(date d1, date d2) {
    if (d1 >= d2) {
        return d1 - d2;
    }
    else {
        return d2 - d1;
    }
}

void main() {
    time ref = 00:00:00;
    date d1 = 01.01.1999.00:00:00; // komentarz
    date d2 = 01.01.1999.00:00:00;
    date d3 = 01.01.2021.00:00:00;

    if (d3 > d1 && dif(d1, d2) == ref) {
        print("everything is ok");
    }
}
// output: everything is ok
```

5

```
void check(date d1, date d2, num n1, num n2) {
    if (d1 >= d2 && n1 > n2 != n1 < n2){
        print("priorities hierarchy ok");
    }
    else {
        print("oops.. something wrong");
    }
}

void main() {
    date d1 = 01.01.1999.11:11:11;
    date d2 = 01.01.1999.11:11:11;
    num n1 = 12.34;
    num n2 = 5;

    check(d1, d2, n1, n2);
}
// output: priorities hierarchy ok
```

6

```
void main() {
    num n1 = 1;

    while (n < 5) {
        num n2 = 0;
        n = n + 1;
    }

    print(n2);
}
// output: error: ... no such variable n2
```

Gramatyka języka (EBNF)

programme	= { functionDef } ;
functionDef	= signature, "(", parameters, ")" , block ;
functionCall	= id, "(", arguments, ")" ;
signature	= type, id ;
parameters	= [signature, { ",", signature }] ;
block	= "{", { statement }, "}" ;
statement	= block ifStatement whileStatement printStatement returnStatement initStatement assignStatement (functionCall, ";") ;
ifStatement	= "if", "(", orCondition, ")" , statement, ["else", statement] ;
whileStatement	= "while", "(", "orCondition", ")" , block ;
printStatement	= "print", "(", printable, { ",", printable }, ")" , ";" ;
returnStatement	= "return", expression, ";" ;
initStatement	= signature, [assignmentOp, expression], ";" ;
assignStatement	= id, assignmentOp, expression, ";" ;
printable	= expression string ;
expression	= term, { addOp, term } ;
parenthExpr	= "(", expression, ")" ;
term	= factor, { multOp, factor } ;
factor	= ["-"], (number date time id parenthExpr functionCall) ;
orCondition	= andCond, { orOp, andCond } ;
andCond	= equalCond, { andOp, equalCond } ;
equalCond	= relationCond, [equalOp, relationCond] ;
relationCond	= primaryCond, [relationOp, primaryCond] ;
primaryCond	= [negationOp], (parenthCond expression) ;
parenthCond	= "(", orCondition, ")" ;

negationOp	= "!" ;
assignmentOp	= "=" ;
orOp	= " " ;
andOp	= "&&" ;
equalOp	= "==" ;
relationOp	= ">" "<" ">=" "<=" ;
addOp	= "+" "-" ;
multOp	= "*" "/" ;
number	= (digit, [".", digit, {digit}]) (naturalDigit, {digit}, [".", digit, {digit}]) ;
date	= digit, digit, ".", digit, digit, ".", digit, [digit, [digit, [digit]]], ".", time ;
time	= digit, digit, ":", digit, digit, ":", digit, digit ;
id	= letter, { digit letter "_" } ;
letter	= "a" ... "z" "A" ... "Z" ;
digit	= "0" ... "9" ;
naturalDigit	= "1" ... "9" ;
type	= "num" "date" "time" ;
string	= """ , { (anyChar - """) " " }, """ ;
anyChar	= ? all visible characters ?

Przyjęte założenia techniczne

wykorzystywane narzędzia

Projekt zostanie wykonany w języku **Java**. Do obsługi formatu `datetime` zostanie wykorzystany pakiet `java.time`. Do testów użyję narzędzia **JUnit**.

przewidywana struktura pakietów projektu

```

datetime-processing-language-main
├── doc
├── programme
│   └── inputCode.txt
├── src
│   ├── main
│   │   ├── interpreter
│   │   ├── lexer
│   │   ├── parser
│   │   └── source
│   └── test
│       ├── interpreter
│       ├── lexer
│       └── parser

```

└─ source

konwencja wejścia/wyjścia

Przy uruchomieniu programu na wejście będzie trzeba podać nazwę pliku z kodem źródłowym, znajdującego się w folderze **programme**. Na standardowym wyjściu zostanie wypisany wynik wykonania programu.

Repozytorium projektowe

Prowadzący został dodany do repozytorium na gitlabie wydziałowym.