Dokumentacja wstępna TKOM

Temat projektu: „Język z czasem”

SPIS TREŚCI

1. OPIS TEMATU PROJEKTU

1.1. OPIS „JĘZYKA Z CZASEM”

Główną, charakterystyczną funkcjonalnością oferowaną przez tworzony język jest **przetwarzanie zmiennych i wartości związanych z czasem** – **momentami w czasie** (np.: data) czy **okresami w czasie** (np.: sekunda). Język obsługuje również liczby całkowite i rzeczywiste oraz łańcuchy znaków (tylko w funkcji do wyświetlania tekstu na ekranie). Przetwarzanie danych możliwe jest dzięki zdefiniowanym operatorom, instrukcji warunkowej i pętli oraz mechanizmom tworzenia zmiennych czy własnych funkcji. Język oferuje również kilka funkcji wbudowanych – np.: do wypisywania tekstu na ekran czy podania aktualnej daty. Z wyjątkiem spacji w łańcuchu znaków, wszelkie białe znaki w kodzie są ignorowane przez język.

1.2. ELEMENTY JĘZYKA

1.2.1. TYPY W JĘZYKU

Implementowany język oferuje obsługę wartości o następujących typach:

- Typ liczbowy (bez jednostki): `20`, `3.14`, `(-8)`, ...;
- **Typ czasowy – moment w czasie:**
  - godzina – `H`, np.: `[H]17:42:36`
  - data – `D`, np.: `[D]08/11/2021`
- **Typ czasowy – okres w czasie:**
  - sekunda – `s`, np.: `2[s]`, `8.4[s]`, `(-16)[s]`, ...;
  - minuta – `min`, np.: `2[min]`, `8.4[min]`, `(-16)[min]`, ...;
  - godzina – `h`, np.: `2[h]`, `8.4[h]`, `(-16)[h]`, ...;
  - dzień – `d`, np.: `2[d]`, `8.4[d]`, `(-16)[d]`, ...;
- Typ łańcucha znaków (używany jedynie w funkcji wyświetlającej tekst na ekranie): `„Ala ma kota”`, `„Witaj”`, ...;

Jak widać z powyższych przykładów, wartości niektórych kategorii wymagają podania wprost swojego typu. Jednostkę należy umieścić w nawiasach klamrowych, bezpośrednio przed lub po wartości liczbowej – zależy to od typu.

1.2.2. DEFINIOWANIE ZMIENNYCH

„Język z czasem” pozwala tworzyć zmienne, przypisywać im wartość oraz ją odczytywać. Język wykorzystuje **typowanie dynamiczne**. Nie trzeba z góry określać typ zmiennej przy jej tworzeniu. Zdefiniowanie zmiennej następuje w momencie przypisania do niej jakiejś wartości – służy do tego **operator =**. Do zmiennych odwołuje się poprzez ich identyfikatory. Przykłady:

```
zmienna_liczbowa = 7;
zmienna_czasowa_okres_1 = 14[s];
zmienna_czasowa_okres_2 = (-5)[h];
zmienna_czasowa_moment_1 = [D]15/04/1990;
zmienna_czasowa_moment_2 = [H]11:12:43;
```

1.2.3. WYRAŻENIA ARYTMETYCZNE

Tworzony język umożliwia konstruowanie wyrażeń arytmetycznych przy użyciu określonych operatorów: *****, **/**, **+**, **–** oraz nawiasów **( )**. Niektóre operatory można użyć jedynie z określonymi kombinacjami typów operandów ze względu na sensowność operacji. Poniższa tabelka pokazuje dozwolone operacje dla danych typów operandów:

Operand 2 / Operand 1	Typ liczbowy	Typ czasowy - okres	Typ czasowy - moment
Typ liczbowy	`*,/,+,-`	`*`	BRAK
Typ czasowy - okres	`*,/`	`+,-`	BRAK
Typ czasowy - moment	BRAK	BRAK	-

Operand operatora unarnego **–** musi mieć typ liczbowy lub czasowy – okres.

Przy obliczaniu wyrażeń algebraicznych, typ wyniku będzie wyznaczany na podstawie typów operandów. W przypadku typów czasowych – moment w czasie różnica będzie wyrażona w dniach dla typu `[D]` oraz w sekundach dla typu `[H]`. W przypadku typów czasowych – okres w czasie typ wyniku będzie taki sam, jak bardziej podstawowy spośród typów operandów tzn. dla operacji `[h] + [s]` wynik będzie w `[s]`; dla operacji `[d]-[min]` wynik będzie w `[min]` itp. W celu konwersji wartości do innego typu, należy użyć odpowiedniej funkcji wbudowanej.

Przykłady wyrażeń arytmetycznych:

```
wyr1 = 12*(6+2.4);
wyr2 = 14[s]/2;
wyr3 = ([D]11/11/2021-[D]10/11/2021)/wyr1;
```

1.2.4. WYRAŻENIA LOGICZNE

Język umożliwia również konstruowanie wyrażeń logicznych przy użyciu określonych operatorów: **==, =\=, <, >, <=, >=** , **&** *(i)*, **|** *(lub)* oraz nawiasów **( )**. Operatory logiczne można używać z operandami, które mają ten sam typ lub który można sprowadzić do tego samego typu, tzn. operacje pomiędzy `[D]` i `[D]`, `[d]` i `[h]`, `[h]` i `[s]` są poprawne, ale np.: `[D]` i `[h]` już nie. Przykłady wyrażeń logicznych zostaną pokazane w części *Instrukcja warunkowa* i *Definiowanie Funkcji*.

1.2.5. INSTRUKCJA WARUNKOWA

W języku istnieje instrukcja warunkowa, która może wystąpić w jednej z dwóch postaci:

- `if(wyrazenie_logiczne){`wykonaj jeśli wyrażenie prawdziwe`}`
- `if(wyrazenie_logiczne){`wykonaj jeśli wyrażenie prawdziwe`}` `else {`wykonaj jeśli wyrażenie fałszywe`}`

Przykład:

```
if(zmienna >= 6[min])
{
  zmienna = zmienna/10;
}
else
{
  zmienna = zmienna-5[min];
}
```

1.2.6. INSTRUKCJA PĘTLI

Język udostępnia instrukcję pętli, która umożliwia wielokrotne wykonanie fragmentu kodu tak długo jak spełniony jest określony warunek.

```
while(warunek_czyli_wyrazenie_logiczne)
{
  powtarzany kod
}
```

Przykład:

```
i = 4;
while(zmienna >= 5[min] & i=\=0)
{
  zmienna = zmienna/5;
  i = i-1;
}
```

1.2.7. DEFINIOWANIE I WYWOŁYWANIE FUNKCJI

„Język z czasem” umożliwia tworzenie własnych funkcji. Deklaracja i definicja funkcji odbywają się w tym samym momencie – nie ma możliwości ich rozdzielenia. W nawiasach **()** można podać nazwy argumentów funkcji. Argumenty są przekazywane do funkcji poprzez wartość. W ciele funkcji można użyć **RET()** do ustalenia wartości zwracanej. Zmienne stworzone w ciele funkcji są widoczne tylko w jej obrębie. Sama definicja powinna się rozpoczynać od **FUNC:** .

`FUNC: nazwa_funkcji(arg1, arg2, ..., argN) {` ciało funkcji`}`

Przykład:

```
FUNC: podzielPrzez5(zmienna)
{
  RET(zmienna/5);
}
```

Aby wywołać funkcję, należy użyć jej nazwy, np.:

```
inna_zmienna = podzielPrzez5(5[s]);
```

1.2.8. FUNKCJE WBUDOWANE

Język oferuje kilka funkcji wbudowanych, które można wywołać tak jak każdą inną funkcję:

- `show(arg1, arg2, ..., argN);` - wypisuje tekst na ekran. Argumentem funkcji może być łańcuch znaków, zmienna lub wyrażenie arytmetyczne.
- `getTime();` - zwraca aktualny czas – wartość typu `[H]`.
- `getDate();` - zwraca aktualną datę – wartość typu `[D]`.
- funkcje do konwersji typów – zostaną one wyspecyfikowane w trakcie trwania projektu w tej sekcji dokumentacji.

1.3. PRZYKŁADOWY PROGRAM

```
FUNC: zwrocIleDniTemu(data)
{
  dzis = getDate();
  RET(dzis - data);
}

dataUrodzin = [D]01/01/2000;
dniOdUrodzin = zwrocIleDniTemu(dataUrodzin);
show(„Ilosc przezytych dni: ”, dniOdUrodzin);

i = 1;
iloscLat = 0;
while(i == 1)
{
    if(dniOdUrodzin >= 365)
    {
          dniOdUrodzin = dniOdUrodzin - 365[d];
          iloscLat = iloscLat + 1;
    }
    else
    {
          i = 0;
    }
}

show(„Ilosc przezytych lat: ”, iloscLat);
```

2. GRAMATYKA JĘZYKA

2.1. SKŁADNIA

```
program              = polecenie, {polecenie};

polecenie            = definicja funkcji
                     | instrukcja;

definicja funkcji    = 'FUNC:', identyfikator, '(',
                     [identyfikator, {',', identyfikator}],
                     ')', blok instrukcji;

blok instrukcji      = '{', {instrukcja}, '}';

instrukcja           = (przypisanie
                     | instrukcja warunk
                     | pętla
                     | wywołanie funkcji
                     | wyświetlenie tekstu
                     | zwrócenie wartości), ';';

przypisanie          = identyfikator, '=', (wyrażenie arytm
                     | wywołanie funkcji);

instrukcja warunk    = 'if', warunek, blok instrukcji
                     [ 'else', blok instrukcji ];

pętla                = 'while', warunek, blok instrukcji;

warunek              = '(', wrażenie logiczne, ')';

wywołanie funkcji    = identyfikator, '(', [wyrażenie arytm,
                     {',', wyrażenie arytm } ], ')';

wyświetlenie tekstu  = 'show', '(',(string
                     | wyrażenie arytm ), {',', (string
                     | wyrażenie arytm ) }, ')';

zwrócenie wartości   = 'RET', '(', wyrażenie arytm , ')';

wyrażenie logiczne   = podwyrażenie logic, {operator logiczny,
                     wyrażenie logiczne };

podwyrażenie logic   =  wyrażenie arytm, operator porównania,
                     wyrażenie arytm
                     || '(', wyrażenie logiczne, ')';

wyrażenie arytm      = składnik, { operator addytywny,
                     wyrażenie arytm };

składnik             = czynnik, { operator multiplikatywny,
                     składnik };
```

```
czynnik              = identyfikator
                     | '(', '-', identyfikator, ')'
                     | stała
                     | '(', wyrażenie arytm, ')'
                     | '(', '-', '(', wyrażenie arytm,')',')';
```

2.2. LEKSYKA

```
identyfikator            = [a-zA-Z][a-zA-Z0-9_]*

operator logiczny        = \| | &

operator porównania      = == | =\\= | < | > | <= | >=

operator addytywny       = \+ | -

operator multiplikatywny = \* | /

string                   = „[a-zA-Z0-9[ \t]]*„

stała                    = moment | liczba jednostka?

liczba                   = rzeczywista nieujemna
                         | \(-rzeczywista nieujemna\)

rzeczywista nieujemna    = liczba nieujemna(\.[0-9]*)?

liczba nieujemna         = 0|([1-9][0-9]*)

jednostka                = \[(s | min | h | d)\]

moment                   = data | godzina;

data                     =\[D\]liczba nieujemna/liczba
                         nieujemna/liczba nieujemna

godzina                  =\[H\]liczba nieujemna:liczba
                         nieujemna:liczba nieujemna
```

3. TECHNICZNE ASPEKTY REALIZACJI PROJEKTU

3.1. PRODUKT KOŃCOWY

Celem projektu jest zaimplementowanie interpretera dla opisanego wcześniej języka. Interpreter będzie pracował w trybie wsadowym. Danymi wejściowymi będzie plik *.txt* z kodem źródłowym „języka z czasem”. Będą one zaciągane przez specjalny moduł dotyczący źródła danych. Z modułem tym będzie komunikował się lekser, który będzie przekazywał ciąg rozpoznanych tokenów parserowi. Parser z kolei będzie budował drzewa składniowe, na podstawie których będzie wykonywany kod przez moduł interpretacyjny. Translacja kodu źródłowego będzie więc przebiegać w ramach współpracy poszczególnych modułów interpretera.

3.2. MECHANIZM OBSŁUGI BŁĘDÓW

Integralną częścią implementowanego interpretera będzie również moduł gromadzący dane dotyczące błędów odnalezionych podczas translacji. Będą one zgłaszane na różnych etapach tego procesu:

- analiza leksykalna – np.: nieznane ciągi znaków,
- analiza składniowa – np.: brak średnika na końcu niektórych instrukcji,
- analiza semantyczna – np.: sprawdzanie poprawności/dopuszczalności niektórych operacji.

3.3. WYBRANE NARZĘDZIA

Narzędzia wybrane do realizacji projektu:

- Język Java,
- Do testów jednostkowych – JUnit 5 i AssertJ,
- Maven,
- Git i Gitlab,
- ...