Politechnika Warszawska, WEiTI

Dokumentacja do projektu z TKOM

Język do generowania wizualizacji danych z JSON do HTML

Ahata Valiukevich

15 czerwca 2017

1 Treść zadania

Napisać język do generowania wizualizacji. Program powinien przyjmować dane w formacie JSON i wizualizować je zgodne z definiowanymi szablonami plików HTML.

2 Opis

Projekt zostanie zaimplementowany w języku *C++*.
Program sprawdza poprawność wprowadzanych danych. Rozbiór odbywa się poprzez analizę kolejnych linii skryptu i wyodrębnianie z nich tokenów. Język jest jest typowany dynamicznie, wyposażony w dwie podstawowe konstrukcje sterujące: pętla oraz instrukcja warunkowa, istnieje możliwość definiowania funkcji w tym rekurencyjnie. Dodatkowo język obsługuje wyrażenia matematyczne uwzględniając priorytet operatorów.

3 Wymagania funkcjonalne do projektu

1. Poprawne wykonanie poprawnie sformułowanego kodu
2. Wyświetlanie błędów oraz wskazanie miejsca, gdzie te błędy występują
3. Wyświetlanie zawartości pliku JSON zgodnie z szablonami języka HTML

4 Wymagania niefunkcjonalne

1. Informacje o błędach powinny być możliwie dokładne

5 Przykładowe uruchomienie

```
mkdir build
cd build
cmake ..
make
./build/runner < ./src/test.lang
```

W przypadku budowania i uruchomienia w środowisku CLion, wystarczy w pliku main.cpp podać ścieżkę do pliku zawierającego przykładowy skrypt.

6 Struktura programu

Program będzie złożony z modułów, których odpowiedzialnością będzie wykonanie kolejnych etapów kompilacji skryptu. Oprócz tego zawierać będzie moduły niezwiązane bezpośrednio z kompilacją, lecz będące użyteczne do budowy pozostałych modułów. Proces kompilacji i wykonania skryptu będzie się odbywać w następujących etapach:

1. Analiza leksykalna
2. Analiza składniowa
3. Wykonanie

Analiza leksykalna

Odpowiedzialnością modułu lexera będzie podział skryptu wejściowego na tokeny, które następnie trafią do modułu parsera.

Analiza składniowa

Moduł parsera będzie ładował kolejne tokeny za pomocą interfejsu udostępnianego przez moduł lexera. Będzie odpowiedzialny za sprawdzenie, czy wejściowy układ tokenów należy do gramatyki języka. Na jego wyjściu będzie drzewo rozbioru skryptu, które trafi do modułu analizatora semantycznego. Moduł będzie współpracował z modułem obsługi błędów.

Wykonanie

Moduł interpretera wykonuje kod przekazany przez użytkownika (uzyskany w fazie kompilacji) w postaci drzewa. Na tym etapie mogą wystąpić błędy związane z niepoprawnymi operacjami oraz inne błędy czasu wykonania. Błędy są zgłaszane przez oddzielny moduł błędów czasu wykonania i powodują zakończenie wykonania skryptu.

Dodatkowe moduły

1. Moduł obsługi błędów - tłumaczy błędy z poszczególnych modułów na postać czytelną dla człowieka.
2. Moduł obsługi błędów czasu wykonania - tłumaczy błąd na postać czytelną dla człowieka i dostarcza podstawowych informacji o stanie programu (np. callstack).

Biblioteka standardowa

1. array() - konstruuje tablicę
2. len(array) - zwraca długość tablicy
3. push(array, value) - dodaje element na końcu tablicy
4. print()

7 Lista tokenów

```
 "function", "(", ")", "{", "}", "{{", "}}", "</", ";", ",", ":", "if", "else",
"for", 'return', "let", " " ", "+", "-", "*", "/", "<", ">", "==", "!=", "<=",
">=", "=", "&&", "||", "[", "]", "true", "false"
```

8 Gramatyka

```
visualisation = { functionDef }
functionDef = "function" ident [ "[" htmlTemplate "]" ] functionArgs functionBody
htmlTemplate = "<" ident attrsList ">" { htmlTemplate | textContent } "</" ident ">"
attrsList = { attr }
attr = ident "=" attrValue
attrValue = injectedValue | String
textContent = { injectedValue | any_text }
injectedValue = "{{" ident "}}"
functionArgs = "(" ident [ {"," ident } ] ")"
functionBody = "{" { expr } "}"
block = expr | "{"expr"}"
expr = declExpr | assignExpr | logicExpr | forExpr| ifExpr | returnExpr ";"
declExpr = "let" ident [ "=" logicExpr ]
forExpr = "for" "(" declExpr ";" logicExpr ";" assignExpr ")" block
assignExpr = var "=" logicExpr
ifExpr = "if" "(" logicExpr ")" block [ "else" block ]
returnExpr = "return" logicExpr
logicExpr = andExpr { orOp andExpr }
andExpr = equalityExpr { andOp equalityExpr }
equalityExpr = relationalExpr { equalOp relationalExpr }
relationalExpr = mathExpr { relationalOp mathExpr }
mathExpr = mulExpr { additiveOp mulExpr }
mulExpr = expr { multiplicativeOp expr }
baseMathExpr = [ unaryMathOp | unaryLogicOp ] ( literal | var | parenthExpr
| objectLiteral | funcCall )
funcCall = ident '(' [ logicExpr { ',' logicExpr } ] ')'
parenthExpr = "(" mathExpr ")"
objectLiteral = "{" [ objectField { "," objectField } ] "}"
objectField = indent ":" logicExpr
unaryMathOp = "-"
unaryLogicOp = "!"
additiveOp = "+" | "-"
multiplicativeOp = "*" | "/"
orOp = "||"
andOp = "&&"
equalOp = "==" | "!="
relationalOp = "<" | ">" | "<=" | ">="
literal = numberLiteral | String | booleanLiteral
numberLiteral = [ "-" ] digit { digit } [ "." { digit } ]
booleanLiteral = "true" | "false"
ident = (letter | specialSymbol) { letter | specialSymbol | digit }
var = ident { indexExpr }
```

```
indexExpr = "[" expr "]"
any_text = { any_char }
any_char = letter | digit | specialSymbol | " "
specialSymbol = "_"
letter = "a".."z" | "A".."Z"
digit = "0".."9"
```

9 Przykładowy skrypt

```
function map(items, func) {
    result = array();
    for (let i = 0; i < len(items); i=i+1)
        { push(result, func(items[i])); }
    return result ;
}

function Nodes[
    <table>
        <thead>
            <tr>
                <td>Nazwa</td>
                <td>Inne dane</td>
                <td>Dzieci</td>
            </tr>
        </thead>

        {{ nodes }}
    </table>
](nodes) {
    return {
        nodes: map(nodes, Node)
    }
}

function Node[
    <tr>
        <td> {{ name }} </td>
        <td> {{ additionalData }} </td>
        {{ children }}
    </tr>
](node) {
```

```
    let children = Empty()
    if (len(node.children) > 0) {
        children = map(node.children, Node)
    }
    return {
        name: node.name,
        additionalData: node.additinalData,
        children: children
    }
}

function Main[
    <html>
        <head>
        </head>

        <body>
     {{ nodes }}
</body>

    </html>
](root) {
    return {
        nodes: Nodes(root.nodes)
    }
  }
```