Iga Słotwińska

Projekt TKOM

Opis

Projekt ma umożliwiać działania na plikach w formacie svg (grafiki wektorowe). Stworzony język powinien zapewniać możliwość wyszukiwania pewnych obiektów, zapisywania ich jako nowe grafiki i zmieniania ich atrybutów (np. kolor). Jako błędne będą traktowane próby pracowania na plikach innych niż svg, np. xml. Zakładana jest poprawność podawanych atrybutów dla danego elementu.

Tokeny

(,) , ” , { , } , , , ; , if , else , while , . , = , == , < , > , <= , >= , != , + , - , * , / , && , || , val, element , list

Operacje

- save(element, path-to-file) // stworzenie lub dodanie elementu do pliku
- save(list, path-to-file) // stworzenie lub dodanie elementu do pliku
- read(path-to-file) // wczytanie elementów pliku do kolekcji
- create() //stworzenie pustej kolekcji
- create(element) // stworzenie jednoelementowej kolekcji
- create(list) // stworzenie kolekcji z obiektami z istniejącej kolekcji
- create(tag-name) // stworzenie elementu o danym tagu
- (list).add(element) // dodanie elementu do kolekcji
- (list).add(list) // dodanie grupy elementów do kolekcji
- (list).remove(element) // usunięcie elementu z kolekcji
- (list).remove(list) // usunięcie grupy elementów z kolekcji
- (list).filterByTag(tag-name) // zwrócenie nowej przefiltrowanej kolekcji z zadanym tagiem
- (list).filterByAttr(attribute-name, value) // zwrócenie nowej przefiltrowanej kolekcji z zadanym atrybutem
- (element).setAttr(attribute-name, value) // ustawienie wartości atrybutu dla danego elementu
- (element).deleteAttr(attribute-name) // usunięcie atrybutu dla danego elementu
- (element).getAttr(attribute-name) // zwrócenie wartości atrybutu
- print(list) // wypisanie elementów w liście
- print(element) // wypisanie atrybutów elementu
- (list).size() // wypisanie wielkości listy

Wymagania funkcjonalne

1. Wypisanie elementów z dokumentu
    1. Wypisanie wszystkich elementów
    2. Wypisanie elementów spełniających pewne wymogi (filtrowanie)
        1. Według tagu np. <rect …/>
        2. Według atrybutu np. <rect id=„2” …/>
2. Wypisanie atrybutów elementów z dokumentu
    1. Wszystkich elementów
    2. Przefiltrowanej grupy elementów (według tagu lub atrybutu)
3. Zmiana atrybutu danego elementu
    1. Zmiana wartości atrybutu
    2. Dodanie nowego atrybutu
    3. Usunięcie istniejącego atrybutu
4. Zapisanie elementów do pliku svg
    1. Utworzenie nowego pliku
    2. Dopisanie do istniejącego dokumentu

Budowa projektu

Proces analizy i wykonywania skryptów będzie odbywał się w następujących etapach:

1. Obsługa źródeł - wczytanie pliku ze skryptem
2. Analiza leksykalna - rozbicie tekstu wejściowego na tokeny, przekazywanie rozpoznanego tokenu do parsera
3. Analiza składniowa - sprawdzenie poprawności przychodzących tokenów ze zdefiniowaną gramatyką, tworzenie drzewa rozbioru
4. Analiza semantyczna - sprawdzenie poprawności programu utworzonego przez analizator składniowy drzewa i jego instrukcji

Dodatkowo, zdefiniowane zostaną moduły do obsługi błędów, które mogą występować na każdym z wyżej wymienionych etapów, moduł do parsowania plików svg oraz moduł z dostępnymi funkcjami - akcjami użytkownika.
Program może przyjąć na wejście plik tekstowy ze skryptem.
W celu sprawdzenia poprawności działania programu zostaną stworzone także testy automatyczne.

Przykład

```
list mlist = read(„./image.svg”);
val color = „#000000”;
if( mlist.size() > 1) {
        mlist[1].setAttr(„fill”, color).setAttr(„opacity”, „0.1”);
}
list rectangles = mlist.filterByTag(„rect”);
Save(rectangles, „./only-rectangles.svg”);
Mlist.remove(rectangles);
save(mlist, „./without-rectangles”);
Val i = 0;
while( i < mlist.size() ) {
        mlist[i].deleteAttr(„stroke”);
        i = i + 1;
}
```

Gramatyka

Statement = IfStmt | initListStmt | initElemStmt | initValStmt | MathStmt | OnVarStmt | PrintStmt | SaveDocStmt | WhileStmt ;

IfStmt = „if” „(“ Expression „)”„{„ { Stmt } „}” [„else” „{„ { Stmt } „}”] ;
WhileStmt = „while” „(„ Expression „)” „{“ { Stmt } „}” ;
MathStmt = [valVar] id assignOp NumberArgument mathOp NumberArgument „;” ;
SaveDocStmt = „save” „(„ id, textArgument „)” „;” ;
PrintStmt= „print” „(„ TextArgument „)” „;” ;

initListStmt = [listVar id assignmentOp] initListFun [{ modListFun } | { filterFun }] „;” ;
onVarStmt = id onVarFun „;” ;
onVarFun = { modListFun } | { filterFun } | { modAttrFun } | getAttrFun | sizeFun ;
initListFun = createFun | readFun ;
createFun = „create” „(„ [id] „)” ;
readFun = „read” „(„ textArgument „)” ;
filterFun = FilterAttrFun | FilterTagFun ;
FilterAttrFun = exOp „filterByAttr” „(„ textArgument , argument „)” ;
FilterTagFun = exOp „filterByTag” „(„ textArgument „)” ;
modListFun = exOp modListOp „(„ id „)” ;
SizeFun = exOp „size” „(„ „)” ;

initElemStmt = [elemVar id assignmentOp] initElemFun [setAttrFun] „;” ;
initElemFun = createElemFun | getElemFun ;
createElemFun = „create” „(„ textArgument „)” ;
getElemFun = id „[„ numberArgument „]” ;
modAttrFun = setAttrFun | deleteAttrFun ;
setAttrFun = exOp „setAttr” „(„ textArgument , argument „)” ;
deleteAttrFun = exOp „deleteAttr” „(„ textArgument „)” ;
getAttrFun = exOp „getAttr” „(„ textArgument „)” ;

initValStmt = valVar id assignOp argument „;” ;

Expression = numberArgument expressionOp numberArgument [{ andOp numberArgument expressionOp numberArgument }] ;

assignOp = „=„ ;
andOp = „&&” | „||”;
expressionOp = „<" | „>" | „<=" | „>=" | „==" | „!=";
mathOp = „+" | „-" | „*” | „/„ ;
modListOp = „add” | „remove” ;
exOp = „.” ;

Argument = textArgument | number ;
TextArgument = text | id ;
NumberArgument = number | id ;

listVar = „list”;
valVar = “val”;
elemVar = “element”;

id = letter [{ zero | digit | letter }];
letter = „a”.."z" | „A”.. „Z" ;
Text = „”” id „””
Number = digit { digit | zero } ;
digit = „1”.. ”9";
Zero = „0” ;