Tomasz Wiaderek

283782

Dokumentacja końcowa

1. **Cel projektu**

   Język skryptowy pozwalający na edycję utworów muzycznych.

2. **Rodzaje obiektów**
   1. Song
   2. Effect
      a. Extend
      b. Shorten
      c. Rise
   3. Effects

   Obiekt **Song** będzie reprezentował utwór muzyczny w formacie MIDI. Użytkownik będzie mógł wczytać piosenkę o podanej nazwie, jak również zapisać ją potem do pliku. Obiekt Song będzie zawierał dwa parametry tempo (**tempo)** i długość (**length**). **Effect** będzie implementował efekty, które będą dodawane do grupy **Effects**, która będzie nakładana na utwór i od razu będzie go modyfikować. Modyfikowanie piosenki zwraca obiekt **Song** jako wynik modyfikacji. Dla obiektu **Effect** będzie można określić typ oraz parametry, które są zdefiniowane w ramach danego typu. Typ **Shorten** będzie umożliwiał skrócenie utworu o zadany w postaci liczby całkowitej procent (analogicznie **Extend** będzie wydłużał utwór). **Rise** będzie podnosił tempo utworu o zadany w postaci liczby całkowitej procent. Język będzie

umożliwiał użytkownikowi definiowanie własnych funkcji, przyjmowanych i zwracanych przez nie wartości (argumenty są referencją).

3. Dostępne prymitywy

- int
- string

4. Dostępne operacje

- Song
  - **i.** „==”, „!=” „=”
  - **ii.** void load(„nazwa_utworu”),
    1. Metoda ustawia tytuł piosenki na zadany i losowo ustawia parametry piosenki.
  - **iii.** void save(„nazwa_pliku”),
    1. Metoda zapisuje parametry zadanej piosenki do pliku: Target/nazwa_pliku.
  - **iv.** int getTempo(),
    1. Metoda zwraca wartość parametru tempo danej piosenki (wartość jest wyrażona w [Bpm]).
  - **v.** int getLength(),
    1. Metoda zwraca wartość parametru length danej piosenki (wartość jest wyrażona w [s]).
  - **vi.** Song modify(Effects e).
    1. Metoda aplikuje kolejno wszystkie modyfikacje zawarte w grupie efektów. Metoda zwraca modyfikowaną piosenkę, nie edytuje obiektu na którego rzecz została wywołana.

- Effect
    i. „==”, „!=”
    ii. void setType(string type),
        1. Metoda ustawia typ obiektu na typ o podanej nazwie (dostępne typy: „Extend”, „Rise”, „Shorten”)
    iii. void setParam(string paramName, int value),
        1. Metoda ustawia wartość zadanego nazwą parametru. (dostępne parametry: „value”).
    iv. string getType(),
        1. Metoda zwraca nazwę typu modyfikacji (początkowo typ jest ustawiony na UNDEFINED, wtedy metoda zwróci pusty napis).
    v. int getParamValue(string name),
        1. Metoda zwraca wartość zadanego nazwą parametru (Po ustawieniu typu wszystkie parametry mają wartość 0, dla niezdefiniowanego typu obiekt nie posiada żadnych parametrów)
- Effects
    i. „==”, „!=”,
    ii. void add(Effect e),
        1. Metoda dodaje do listy efektów podany efekt.
    iii. int getCount(),

1. Metoda zwraca rozmiar listy dodanych efektów.

iv. Effect getAt(int index),

1. Metoda zwraca efekt z listy na zadanym indeksie. Jeśli rozmiar listy efektów jest mniejszy od podanego indexu rzuca wyjątek.

v. void delete(int index),

1. Metoda usuwa efekt o zadanym indeksie z listy. Jeśli rozmiar listy efektów jest mniejszy od podanego indexu rzuca wyjątek.

- int
  i. „=”, „+”, „-”, „*”, „/”, „==”, „!=”;
- string
  i. „==”, „!=” , „=”, „+”(konkatenacja dopuszczana również z liczbami);
- Operatory logiczne
  i. „!”, „||”, „&&”;
- Funkcje wbudowane:
  i. void PRINT(string text)
    1. Wyświetla na ekranie podany tekst;
- Dostępne konstrukcje
  i. IF

```
IF (i == 0)
 {
        PRINT(„zero”);
  }
  ELSE {
        PRINT(„not zero”);
 }
```

ii. FOR

```
FOR i = 0 TO 5 STEP 1
{
  PRINT(i);
}
```

5. Gramatyka

KONWENCJE LEKSYKALNE:

Ident = litera, {litera, cyfra};

Liczba = [„-”] cyfra_bez_zera, {cyfra};

Napis = {litera|cyfra};

Komentarz = „//”… newline

GRAMATYKA:

Program = {FuncDecl};

FuncDecl = „FUNC”, RetFuncDecl | VoidFuncDecl;

RetFuncDecl = Type, FuncName, „(”, [Arguments], „)”, RetBlock;

FuncName = Ident;

VoidFuncDecl = „void”, FuncName, „(”, [Arguments], „)”, Block;

Arguments = Argument, { „,”, Argument};

Argument = Type, Ident;

Type = SimpleType | ObjectType;

SimpleType = „string” | „int”;

ObjectType = „Song” | „Effect” | „Effects”;

Block = „{”, {VarDecl | Statement}, „}”;

RetBlock = “{“ , {VarDecl | Statement} , “return” , RetVal , “;” , “}”;

RetVal = ([VarName , “.”] , VarName) | FuncExpr | MathExpr | liczba;

VarDecl = ObjectDecl | SimpleVarDecl, „;”;

ObjectDecl = ObjectType, VarName;

SimpleVarDecl = SimpleType, VarName, { „,”, VarName};

VarName = Ident;

Statement = SimpleStatement | ComplStatement;

SimpleStatement = Assigment | FuncStatement | OutPutStatement;

Assigment = VarName, „=”, MathExpr , „;”;

MathExpr = Parameter, {MathOp, Parameter};

Parameter = ([VarName , “.”] , VarName) | FuncExpr | Liczba | („””, napis, „””);

MathOp = „+”| „-”| „*” | „/”;

FuncExpr = [VarName , “.”] , FuncName , “(“ , [TypelessArguments] , “)”;

TypelessArguments = Parameter, { „,”, Parameter};

FuncStatement = FuncExpr. „;”;

OutPutStatement = „PRINT(”, RetVal, „);”

ComplStatement = IfStatement | ForStatement;

IfStatement = „IF”, „(” Condition, „)” „{” Block „}” [„ELSE”, „{”, Block, „}”];

Condition = Comparision, {LogicalOp, Comparision};

LogicalOp = „&&” | „||”;

Comparision = NegativeComp | MathExpr , {CompOp , MathExpr};

CompOp = “==” | “<” | “>” | “<=” | “!=” | “>=”;

NegativeComp = „!”, „(”, Comparision, „)”;

ForStatement = „FOR”, Assigment, „TO”, NumMathExpr [„STEP”, NumMathExpr] „{” BLOCK „}”;

NumMathExpr = NumParameter, {MathOp, NumParameter};

NumParameter = ([VarName , “.”] , VarName) | FuncExpr | Liczba;

Tokeny:

Jednoznakowe:

„(”, „)”, „{”, „}”, „,”, „.”, „+”, „-”, „*”, „;”, „””;

Jedno lub dwuznakowe:

„!”, „<”, „>”, „=”, „/”, „!=”, „<=”, >=„”, „==”, „//”;

Literały:

Identifier, String, Number;

Słowa kluczowe:

„Song”, „Effect”, „Effects”, „&&”, „||”, „IF”, „ELSE”, „FOR”, „TO”, „STEP”, „string”, „int”, „RETURN”, „FUNC”;

6. Składnia

- Składnia języka opiera się na składni języków obiektowych takich jak java i c++. Składnia wymaga, żeby program zawierał jedną funkcję main typu void, która nie przyjmuje, żadnych parametrów wejściowych. Program rozpocznie wykonywanie właśnie od funkcji.
- Język wymaga aby funkcje były deklarowane i definiowane w tym samym miejscu.
- Deklaracja i inicjalizacja zmiennych musi się odbywać osobno, w dwóch oddzielnych instrukcjach.
- Zmienne przekazywane do funkcji są przekazywane jako referencja, a nie kopia.
- Funkcje mogą rekurencyjnie wywoływać inne funkcje niezależnie od miejsca ich deklaracji.
- Język pozwala na „nawiasowanie” wyrażeń.

7. Przykłady kodu

==========1==========

```
FUNC void printEffects(Effects e) {
        int i;
        FOR i = 0 TO e.getCount() STEP 2 {
                Effect ef;
                ef = e.getAt(i);
                PRINT(ef.getParamValue("value"));
        }
}

FUNC void main() {
    Effects e;
    Effect ex;
    ex.setType("Extend");
    ex.setParam("value", 4);

    e.add(ex);

    printEffects(e);
}
```

==========2==========

```
FUNC int power(int i) {
    int sum;
    sum = 1;
    IF (i != 0) {
        int tmp;
        int tmpi;
        tmpi = i - 1;
        tmp = power(tmpi);
        sum = tmp + sum;
        RETURN sum;
    }
    RETURN sum;
}

FUNC void main() {
    int i;
    i = (1 + 2) * 3 - 4;
    int res;
    PRINT(i);
    res = power(i);
    string msg;
    msg = "2 to power " + i + " equals: " + res;
    PRINT(msg);
}
```

8. Moduły

Projekt składa się z kilku modułów:

- Scanner + Source
- Parser
- SemCheck
- Executor
- Interpreter

Scanner na żądanie pobiera ze **źródła (source)** kolejne znaki, aż do zbudowania tokenu. **Parser** pobiera ze scanner'a kolejne tokeny i składa z nich drzewo rozbioru, sprawdzając zgodność z gramatyką języka. **Parser** aby móc stworzyć poprawne drzewo musi czasem przeanalizować kolejny token (kolejny w przód). **SemCheck** odpowiada za analizę semantyczną. Sprawdza zgodność typów i operacji, czy program posiada odpowiednią funkcję main. **Executor** znajduje funkcję main i zaczyna wykonanie od niej. **Interpreter** jest kontenerem dla wszystkich tych operacji, odpowiada za uruchomianie kolejnych modułów jeśli poprzedni nie zgłosił żadnych błędów.

9. Testowanie

- Projekt zawiera testy jednostkowe, które sprawdzają poprawność działania danych komponentów.

- Przykłady ręcznych testów funkcjonalnych:

KOD:

```
FUNC void main() {
    int i;

    FOR i = 0 TO 10 STEP 2 {
        PRINT (i);
    }
}
```

WYNIK:

PRINT: 0
PRINT: 2
PRINT: 4
PRINT: 6
PRINT: 8

KOD:

```
FUNC int power(int i) {
  int sum;
  sum = 1;
  IF (i != 0) {
      int tmp;
      int tmpi;
      tmpi = i - 1;
      tmp = power(tmpi);
      sum = tmp + sum;
      RETURN sum;
  }
  RETURN sum;
}

FUNC void main() {
    int i;
    i = 5;
    int res;
    res = power(i);
    string msg;
    msg = "2 to power " + i + " equals: " + res;
    PRINT(msg);

}
```

WYNIK:

PRINT: 2 to power 5 equals: 32

KOD:

```
FUNC void printEffectsAndDelete(Effects e) {
        int i;
        FOR i = 0 TO e.getCount() STEP 1 {
                Effect ef;
                ef = e.getAt(0);
                string msg;
                string type;
                type = ef.getType();
                int value;
                value = ef.getParamValue("value");
                msg = "type: "  + type + ", value: " + value;
                PRINT(msg);
                e.delete(0);
        }
}

FUNC void main() {
    Effect rise;
    Effect shorten;
    Effect extend;
    rise.setType("Rise");
    rise.setParam("value", 1);
    shorten.setType("Shorten");
    shorten.setParam("value", 2);
    extend.setType("Extend");
    extend.setParam("value", 3);

    Effects ef;
    ef.add(rise);
    ef.add(shorten);
    ef.add(extend);
    PRINT(ef.getCount());
    printEffectsAndDelete(ef);
    PRINT(ef.getCount());

}
```

WYNIK:

PRINT: 3
PRINT: type: Rise, value: 1
PRINT: type: Shorten, value: 2
PRINT: type: Extend, value: 3
PRINT: 0

KOD:

```
FUNC void main() {
    int i;
    Song song;
    song.load("mama mia");
    i = 2 + song;

}
```

WYNIK:

java.lang.Exception: Incompatible types! Requierd [INT] get [SONG]

KOD:

```
FUNC void main() {
    Effect ef;
    ef.setType("another");
}
```

WYNIK:

java.lang.IllegalArgumentException: There is no Effect type [another]