Interpreter prostego języka z wbudowanym typem do obsługi grafiki wektorowej

Dokumentacja

Łukasz Wolanin

25 maja 2020

1 Opis funkcjonalny

W ramach projektu powstał interpreter prostego języka przypominającego C, wyposażony w podstawowe instrukcje sterujące (instrukcje if, else i while). Język ten będzie umożliwi definiowanie funkcji przez użytkownika oraz posiada wbudowany typ do obsługi grafiki wektorowej 2D, oraz typ koloru. Język spełnia standardy takie jak: zachowywanie priorytetów operatorów arytmetycznych, obsługa zasięgów zmiennych. Język posiada bibliotekę standardową z funkcjami do obsługi typu graficznego takie jak rysowanie linii, koła, itd. skalowanie ich, rotację itp.

1.1 Wymagania funkcjonalne

1. Możliwość deklarowania zmiennych typów int i string
2. Możliwość deklarowania zmiennych typu specjalnego: graphic
3. Możliwość definiowania swoich kolorów za pomocą typu: color
4. Możliwość deklarowania własnych funkcji przyjmujących argumenty poprzez wartość, oraz zwracających wartość o zadanym typie
5. Poprawne obliczanie wyrażeń arytmetycznych z zachowaniem kolejności operatorów
6. Język umożliwia wykorzystanie instrukcji warunkowej if - else
7. Język udostępnia pętlę while
8. Język oblicza wartość wyrażeń logicznych zgodnie z zasadami logiki
9. W wyrażeniach logicznych można używać operatorów porównania
10. Operowanie typem specjalnym za pomocą wbudowanych funkcji
    - funkcje to rysowania linii, okręgu, wielokątów
    - funkcja do nie wypełniania kolorem
    - funkcje do przemieszczania, skalowania, oraz obracania
11. Możliwość umieszczania komentarzy w kodzie
12. Język umożliwia wyświetlanie lub zapisywanie wygenerowanego obrazu
13. Każdy program musi zawierać funkcję main
14. Język ułatwia operację na obiektach graficznych przechowywanych w postaci wektorowej
15. W wyrażeniach logicznych typ int jest konwertowany na wartość logiczną jak w języku C

16. Typy inne niż int nie mają swojej wartości logicznej
17. Położenie centrum każdego obiektu jest podawane względem środka obiektu nadrzędnego
18. Uruchomienie programu z niepoprawnymi parametrami informuje użytkownika o możliwych parametrach startowych

1.2 Wymagania niefunkcjonalne

1. Każdy obiekt typu graficznego posiada parametry jak:
   - wysokość i szerokość
   - pozycja, rotacja i skala (transform)
2. Wielkość pliku źródłowego nie jest bezpośrednio ograniczona przez interpreter
3. Komunikaty o błędach analizy plików są przejrzyste i ułatwiają wyszukanie błędu
4. Program powinien działać w środowisku Windows, ale też łatwo dać się przenieść do środowiska Linuxowego

2 Formalny opis gramatyki

2.1 Formalny opis gramatyki

```
program = {function_def};
function_def = data_type, identifier, call_def, body;
call_def = "(", [data_type, identifier, { ",", data_type, identifier }], ")";
call_operator = "(", [right_val, {",", right_val}], ")";
body = "{", {instruction}, "}";
instruction = if_statement | while_loop | asign_statement |
   init_statement | function_exec | return_statement | "break;";
if_statement = "if", "(", condition, ")", (body | instruction), [ "else",
(body | instruction)];
while_loop = "while", "(", condition, ")", (body | instruction);
asign_statement = identifier, "=", right_val, ";";
init_statement = data_type, identifier, ["=", right_val], {",", identifier,
   "=", right_val}, ";";
function_exec = function_call, ";";
return_statement = "return", right_val, ";";
right_val = literal | identifier | function_call | expression;
expression = [-], multiplicative_expr, {("+"|"-"), multiplicative_expr };
multiplicative_expr = primary_expr, {("*"|"/"), primary_expr };
primary_expr = ( integer_literal | identifier | function_call | bracket_expr );
bracket_expr = "(" expression ")";
condition = and_cond, {"||", and_cond };
and_cond = relation_cond, {"&&", relation_cond };
relation_cond = primary_cond, [ ("==" | "!=" | relation_oper), primary_cond ];
primary_cond = ["!"], ( braces_cond | right_val );
braces_cond = "(", condition, ")";
function_call = identifier, call_operator;
data_type = "int" | "string" | "graphic" | "color";
identifier = (letter|"_"){letter|digit|"_"};
literal = string_literal | integer_literal;
string_literal = '"', {?visible character? | ?white space? }, '"';
integer_literal = "0" | ["-"], non_zero_digit, {digit};
letter = "a"| "b"| "c"| "d"| "e"| "f"| "g"| "h"| "i"| "j"| "k"| "l"| "m"|
   "n"| "o"| "p"| "q"| "r"| "s"| "t"| "u"| "v"| "w"| "x"| "y"| "z"| "A"| "B"|
   "C"| "D"| "E"| "F"| "G"| "H"| "I"| "J"| "K"| "L"| "M"| "N"| "O"| "P"|
   "Q"| "R"| "S"| "T"| "U"| "V"| "W"| "X"| "Y"| "Z";
non_zero_digit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
digit = "0"| non_zero_digit;
relation_oper = "<" | ">" | "<=" | ">=";
```

2.2 Biblioteka standardowa

Język został wyposażony w bibliotekę standardową pozwalającą na tworzenie obiektów graficznych, kolorów i ich modyfikację:

1. int print(string) - funkcja służy do wyświetlania komunikatów na konsoli
2. int show(graphic) - funkcja służy do wyświetlania obrazów (używa biblioteki OpenGL)
3. graphic blank(int, int) - funkcja zwraca pusty obraz na którym można malować
4. graphic triangle(int, int) - funkcja zwraca trójkąt równoramienny o zadanej wysokości i szerokości
5. graphic rectangular_triangle(int, int) - funkcja zwraca trójkąt prostokątny o zadanych długościach przyprostokątnych
6. graphic rectangle(int, int) - funkcja zwraca prostokąt o zadanych wymiarach
7. graphic circle(int) - funkcja zwraca koło o zadanym promieniu
8. graphic line(int) - funkcja zwraca linię o zadanej długości
9. graphic add(graphic, graphic) - funkcja dodaje do pierwszego obiektu drugi i zwraca pierwszy
10. graphic translate(graphic, int, int) - funkcja zwraca obiekt przesunięty o zadany wektor
11. graphic scale(graphic, int, int) - funkcja zwraca obiekt przeskalowany o zadane wartości wyrażone w procentach
12. graphic rotate(graphic, int) - funkcja zwraca obiekt obrócony o zadany kąt wyrażony w stopniach
13. graphic unfill(graphic) - funkcja zwraca obiekt który podczas rysowania nie będzie wypełniony kolorem
14. color color_rgb(int, int, int) - funkcja zwraca kolor o odpowiednich wartościach składowych RGB
15. graphic set_color(graphic, color) - funkcja zwraca obiekt o zadanym kolorze

3 Opis techniczny realizacji

Projekt został zrealizowany w języku C++ w standardzie C++17 w środowisku MS Visual Studio 2019. Do implementacji testów jednostkowych została wykorzystana biblioteka boost, natomiast do realizacji funkcji bibliotecznej show została wykorzystana biblioteka OpenGL

3.1 Definicja produktu końcowego

Interpreter jest aplikacją konsolową przyjmującą jako argumenty ścieżkę do pliku z kodem w wyżej zdefiniowanym języku, który będzie interpretowany, a informacje o wyniku poszczególnych etapów analizy oraz wyjście programu są wyświetlane na konsoli. Interpreter powiadamia użytkownika o błędach wykrytych w przekazanym do interpretacji skrypcie.

3.2 Budowa interpretera

Na interpreter składają się moduły odpowiedzialne za poszczególne etapy procesu interpretacji

Analiza leksykalna Scanner jest odpowiedzialny za generację tokenów z ciągu znaków jaki otrzyma na wejściu. Osobny moduł jest odpowiedzialny za stworzenie abstrakcji strumienia danych przekazywanych do scannera tak aby nie uzależniać go od konkretnej implementacji źródła kodu. Scanner jest połączony z pomocnymi strukturami jak tablica tokenów. Wygenerowane tokeny na bieżąco przekazuje do parsera.

Analiza składniowa Parser na wejściu otrzymuje tokeny przekazane przez scanner. Następnie budował z tych tokenów drzewo składniowe jednocześnie sprawdzając czy wszystkie tokeny są ułożone zgodnie ze zdefiniowaną gramatyką języka. Parser ma dostęp do tablicy symboli i na jej podstawie dodaje nowe symbole lub zgłasza błąd.

Analiza semantyczna i wykonanie kodu Egzekutor kodu sprawdza to czego nie sprawdzono w poprzednich etapach: poprawność używanych identyfikatorów, zgodność kontekstów operacji logicznych, zgodność kontekstów operacji arytmetycznych, zgodność ilości parametrów wywołań funkcyjnych, brak nadpisywania identyfikatorów funkcji. Moduł przechowuje tablicę symboli i wywołuje rekurencyjną procedurę wykonania kodu przekształconego już do drzewa składniowego.

Tablica symboli Tablica symboli przechowuje wszystkie symbole zdefiniowane przez użytkownika oraz symbole reprezentujące funkcje biblioteczne. Symbole są przechowywane albo w globalnym zasięgu, albo w odpowiedniej warstwie stosu zasięgów lokalnych. W zależności gdzie zmienna lub funkcja była zdefiniowana

3.3 Testowanie

Zostały napisane testy jednostkowe dla:

- Klasy Reader czytającej źródło kodu
- Klasy Scannera
- Klasy Parsera
- Tablicy Symboli
- Własnej klasy wyjątków służącej do zgłaszania błędów w kodzie przekazanym do interpretacji
- Egzekutora kodu, oraz metod ewaluacji i wykonania węzłów drzewa programu przekazanego do interpretacji
- klas reprezentujących węzły drzewa składniowego
- funkcji biblioteki standardowej języka

3.4 Przykłady kodu

Przykład całego programu

```
graphic spruce(int levels)
{
graphic base = rectangle(50,100);
base = set_color(base, color_rgb(210, 105, 30));
graphic level = triangle(120, 150);
level = set_color(level, color_rgb(2, 100, 64));
while(levels > 0)
{
level = translate(level, 0, 50 - levels * 5);
level = scale(level, 80, 80);
base = add(base, level);
levels = levels -1;
}
return base;
}

graphic forest(int composition)
{
graphic tree = spruce(5), tree2 = spruce(4);
tree2 = translate(tree2, -100, 0);
tree = add(tree, tree2);
tree2 = spruce(3);
if(composition)
```

```
tree2 = translate(tree2, 150, 0);
else
{
tree2 = translate(tree2, 90, 0);
tree2 = rotate(tree2, 5);
}
tree = add(tree, tree2);
return tree;
}


int main()
{
graphic root = blank(800, 600), sun;
graphic ground = rectangle(800,350);
ground = set_color(ground, color_rgb(144, 238, 144));
sun = circle(60);
sun = set_color(sun, color_rgb(255, 255, 161));
sun = translate(sun, 300, 200);
root = add(root, sun);
ground = translate(ground, 0, -150);
root = add(root, ground);
root = set_color(root, color_rgb(212, 235, 242));
root = add(root, forest(0));
show(root);
return 0;
}
```

Przykład definicji funkcji

```
graphic drawArrow()
{
graphic arrow = rectangle(50,100);
graphic arrow3 = triangle(60, 60);
arrow3 = translate(arrow3, 50);
arrow = add(arrow, arrow3);
return arrow;
}
```

Przykład wykorzystania pętli while i zdefiniowanej funkcji

```
graphic root = blank(800, 600);
int i = 0;
while(i < 5)
{
graphic arrow = drawArrow();
```

```
arrow = translate(arrow, -300 + 80*i, 0);
root = add(root, arrow);
i = i + 1;
}
```

3.5 Przykład wyniku wykonania poprawnego kodu przez interpreter