TKOM - Projekt Wstępny

Krzysztof Blankiewicz

November 24, 2020

1 Opis projektu

Celem projektu jest stworzenie języka (właśc. kompilatora języka) opisu prostych animacji 2D. Przez proste animacje 2D rozumiem sekwencje ruchów figur geometrycznych na płaszczyźnie. Animacja składa się z klatek. Klatka (analogicznie jak w filmie) to obraz płaszczyzny ekrazu w danym momencie czasu. Jako że nie wymyśliłem jeszcze satysfakcjonującej nazwy języka, na potrzeby tego opisu nazywam go po prostu "XXX".

Zatem program napisany w języju "XXX" będzie definiował pewne obiekty składające się z figur geometrycznych (bądź będące figurami geometrycznymi) oraz opisywał ich poruszanie się w pewnych ramach czasowych po płaszczyźnie ekranu.

Także zasadnicze cele jakie stawiam przed tworzonym językiem to:

- możliwość zdefiniowania wyglądu obiektów graficznych
- możliwość opisania ruchu obiektów
- możliwość opisania modyfikacji wyglądu obiektów

Program będący efektem końcowym projektu będzie napisany w C++, a w ramach swojego działania poza kompilacją programu w "XXX" będzie również generował zadaną animację.

2 Opis języka

"XXX" będzie językiem syntaktycznie zblizonym do C++. Semantycznie będzie jego podzbiorem, rozszerzonym o wbudowane typy graficzne i ich obsługę (w szczególności metody służące ich reprezentacji graficznej). Również o nieco zmodyfikowanym przepływie sterowania - np. pewne metody będą wywoływane cyklicznie, "co klatkę".

2.1 Struktura programu

Program w "XXX" składa się z 5 części:

2.1.1 Inicjalizacje

Wskazanie parametrów dotyczących całości animacji jak czas trwania, prędkość, wielkość ekranu. Nie mam jeszcze dokładnego wyobrażenia jakie te parametry będą i jak będzie się je inicjowaćiem, ten element języka będzie się istotnie zmieniać w trakcie pracy.

2.1.2 Definicje klas

"Klasa" w "XXX" przypomina strukturę w C. Jest zgrupowaniem obiektów (typów wbudowanych lub innych klas) z zestawem metod operujących na tych obiektach. Obiekt będący elementem klasy nazywamy polem tej klasy.
Kluczowa jest metoda draw(), która musi być zdefiniowana dla każdej klasy. Jest ona wywoływana co klatkę. Przede wszystkim ma określać sposób obrazowania obiektu klasy na ekranie. W ramach draw() można wywołać inne metody zmieniające położenie/parametry/wygląd pól obiektu klasy. Tu warto zaznaczyć, że każdy układ klas zawierających kolejne klasy ostatecznie da się sprowadzić do pewnej ilości obiektów typów wbudowanych, dla których metoda draw() jest zaimplementowana.

```
class domek
{
  int x = 4;      // nale y pami ta  ,  e  nie istniej
    konstruktory dla klas
  rectangle field1;
  rectangle field2;
  triangle field3;

  draw(){
        field1.draw();
        field2.draw();
        field3.draw();
    }
}

class miasteczko
{
  rectangle ulica;
  domek dom[3];
    bool kwarantanna[3];
    int podatekOdCovida = 0;

    draw()
    {
        ulica.draw();
        for(int i = 0; i < 3; ++i)
            dom[i].draw();
            if(kwarantanna[i] == true)
            {
                podatekOdCovida += 30000;
            }
    }
}
```

2.1.3 Definicje obiektów występujących w animacji

Analogicznie jak w językach obiektowych, wskazanie obiektów (typów wbudowanych lub własnych klas), które pojawią się w animacji. Również obiektów, które nigdy nie będą widoczne, a będą potrzebne w obliczeniach. Możliwe jest tworzenie tablic. Jako że nie istnieją konstruktory, w tej sekcji można

inicjować pola obiektów. Warto zwrócić uwagę, że przy odpowiednio zdefiniowanych klasach, w tej części może pojawić się definicja tylko jednego obiektu klasy reprezentującej całą klatkę animacji. Obiekty są rysowane, a właściwie co klatkę są wywoływane metody draw() na rzecz obiektów, w kolejności wskazanej w tym miejscu. Zmiana kolejności wyświetlania jest możliwa tylko jeśli interesujące nas obiekty ujmiemy w klasę i w odpowiednio zaimplementujemy jej metody.

```
objects{
    miasteczko warszawa;
    domek mojDom;
    nographic int x[10];
    //"nographic" oznacza obiekt, ktorego sie nie rysuje
    nographic triangle pomocniczyTrojkat;

    pomocniczyTrojkat.visible = false;
}
```

2.1.4 Definicje funkcji

Funkcje są metodami dostepnymi globalnie. Mogą być wywoływane zarówno w metodach klas, jak i w funkcji main(). Mogą przyjmować obiekty wystepujące w animacji jako argumenty. Wyglądają dokładnie jak w języku C++.

2.1.5 Funkcja main()

Jest funkcją globalną, której nie można wywołać z innego miejsca programu. Jej rola jest istotnie różna od tej z języków obiektowych. Funkcja main() jest wołana co klatkę przed wywołaniem metod draw() wszystkich obiektów. Tylko w tym miejscu możliwe jest np. stworzenie rozróżnienia między obiektami tej samej klasy. Wygląda dokładnie jak w języku C++ poza brakiem zwracanej wartości.

2.2 Gramatyka

W załączeniu znajduje się opis gramatyki języka w postaci pliku ebnf.

2.3 Typy wbudowane

- int
- bool
- double
- char
- rectangle
- triangle

- circle
- line

Cztery ostatnie są typami graficznymi, nieco bardziej złożonymi. Posiadają pola typu bool "visible" i "filled" oraz odpowiednią ilość pól typu double określających położenie na ekranie.

2.4 Pola klas

Każdy obiekt tworzonych przez użytkownika klas ma domyślnie pola x i y oznaczających pozycję na ekranie. Do pól obiektu dostać się można operatorem ".".

3 Opis sposobu realizacji

Schemat ponizej obrazuje zależności modułów projektu.

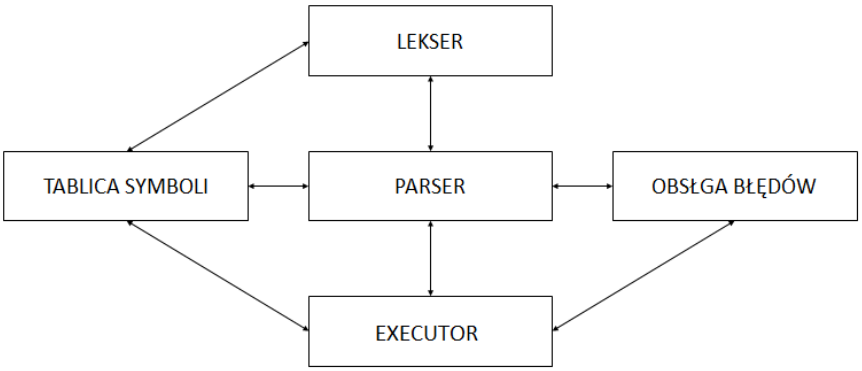

Figure 1: Moduły projektu

3.1 Lekser

Lekser będzie modułem wywoływanym przez parser (za każdym razem gdy parser potrzebuje tokenu). Będzie przechowywał uchwyt do pliku z programem i zwracał kolejne tokeny. Zawartość części tokenów będzie przechowywana w tablicy symboli.

3.1.1 Tokeny

Rozróżniane będą natępujące typy tokenów: liczba, symbol (czyli dowolny ciąg znaków alfanumerycznych), operator, endOfFile, invalidToken. Słowa kluczowe to również symbole. Symbole i operatory będą przekazywane przez tablice symboli. Operatory ze względu na dwuznakowe operatory, które trudniej by było przekazywać bezpośrednio w tokenie.

3.2 Parser

Będzie parserem RD (rekurencyjnym, zstępującym). Na podstawie efektów działania leksera będzie tworzył drzewo wyprowadzenia. Implementując parser zamierzam stworzyć jedynie szkielety klas reprezentujących węzły drzewa. Wystarczy to do kontrolowania poprawności składniowej, a dopiero na etapie tworzenia executora dodana zostanie tam zasadnicza treść.

3.3 Executor

(Zachowałem angielską pisownię, jako że "egekucja" ma dośc negatywne konotacje.)
Ten moduł będzie odpowiedzialny za stworzenie animacji. Na tym etapie realizacji jeszcze nie w pełni mam wyobrażenie o jego działaniu. Na pewno będzie on w pewnym sensie warstwą obstrakcji dla parsera. Właściwie warstwą abstrakcji dla metod klas tworzących drzewo wyprowadzenia - udostępni interfejs pod którym dopiero będę korzystać z konkretnej biblioteki graficznej. Biblioteką tą będzie Allegro5.

4 Testowanie

Lekser będzie testowany prostym programem wywołującym go i wypisującym na konsolę kolejne tokeny. Newralgicznym wydaje się testowanie parsera, jako że efektem jego działania jest potencjalnie bardzo złożone drzewo. Do tego celu stworzyłem pomocniczy program przedstawiający odpowiednio skontruowane drzewo w postaci graficznej z możliwością interaktywnego poruszania się po węzłach. Wydaje mi się, że będzie to wystarczające do efektywnego "debugowania" parsera - nie zamierzałem pisać dodatkowych testów.