

# Programowanie obiektowe – dokumentacja projektu „Arkusz kalkulacyjny”

Paweł Laskoś-Grabowski  
nr indeksu 169827

19 czerwca 2006

## 1 Wstęp

Program „Arkusz kalkulacyjny” służy do obliczania wartości wyrażeń arytmetycznych złożonych z czterech działań arytmetycznych, nawiasów, literałów stało- i zmiennopozycyjnych oraz nazw zmiennych.

Program został napisany w języku C# 2.0, kompilowany i testowany na platformie Mono 1.1.15. Instalacja na komputerze wyposażonym w system operacyjny Linux oraz platformę Mono polega na wykonaniu polecenia `make` w katalogu, w którym znajdują się pliki źródłowe `librpn.cs`, `libdict.cs`, `libexpr.cs`, `sheet.cs` oraz plik sterujący `Makefile`. W przypadku braku pliku sterującego należy wykonać polecenia

```
gmcs -t:library librpn.cs
gmcs -t:library libdict.cs
gmcs -r:librpn.dll,libdict.dll -t:library libexpr.cs
gmcs -r:librpn.dll,libdict.dll,libexpr.dll sheet.cs
```

Program uruchamia się poleceniem `mono sheet.exe`.

Instalacja na komputerze wyposażonym w system operacyjny MS *Windows* oraz platformę .NET 2.0 instalacja polega na wykonaniu w folderze, w którym znajdują się ww. pliki źródłowe poleceń

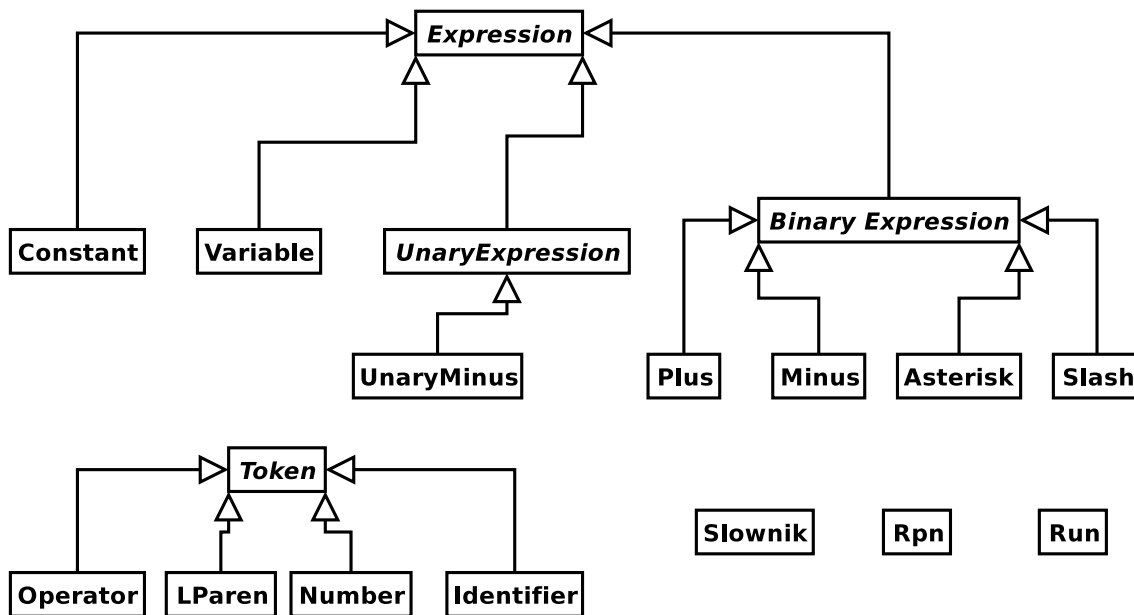
```
csc /t:library librpn.cs
csc /t:library libdict.cs
csc /r:librpn.dll,libdict.dll /t:library libexpr.cs
csc /r:librpn.dll,libdict.dll,libexpr.dll sheet.cs
```

Program uruchamia się poleceniem `sheet`. Póki standardowe wejście nie jest puste, program czyta z niego linie. Poprawna linia wejścia jest postaci `<nazwa>:<wyrażenie>`.

*Uwaga!* Znak - (minus) w wykładnikach literałów zmiennopozycyjnych zastąpić należy ~ (tyldą). Użycie . lub , jako kropki dziesiętnej zależy od ustawień lokalnych komputera, na którym program był kompilowany.

Działanie programu może zakończyć się niepowodzeniem, jeśli wystąpi m.in. zapełnienie definicji, brak definicji, lub błąd rozbioru wyrażenia. Wtedy wypisywany jest na standardowe wyjście stosowny komunikat, a program zwraca niezerowy kod wyjścia. W przypadku wystąpienia kilku definicji jednej nazwy, wiążąca jest ostatnia.

Działanie programu kończy się powodzeniem, jeśli udało się wyliczyć wartości wszystkich nazw. Zostają one wypisane na standardowe wyjście w kolejności alfabetycznej, a pro-



Rysunek 1: Hierarchia klas projektu „Arkusz kalkulacyjny”

gram kończy obliczenia zwracając kod wyjścia 0.

## 2 Analiza obiektowa

Hierarchię klas graficznie przedstawiono na rys. 1.

### 2.1 Biblioteka librpn

Przestrzeń nazw: *RpnParser*

Klasy: *Token*, *Operator*, *LParen*, *Identifier*, *Number*, *Rpn*

Wyjątki: *ParenMismatch*, *InvalidExpression*

**Token** Abstrakcyjna. Szczyt hierarchii klas żetonów, z których składać będzie się wyrażenia w odwrotnej notacji polskiej.

**Operator** Dziedziczy po *Token*. Zawiera pole wyliczeniowe *type*, przechowujące informację o typie operatora, jaki reprezentuje obiekt tej klasy. Konstruktor jednoargumentowy – przypisuje polu jego wartość.

**LParen** Dziedziczy po *Token*. Nie zawiera pól – służy jedynie jako wartownik reprezentujący lewy nawias w konstrukcji wyrażenia w ONP. Stąd też brak analogicznej klasy dla prawych nawiasów.

**Identifier** Dziedziczy po *Token*. Zawiera pole napisu *name*, przechowujące nazwę zmiennej, reprezentowanej przez obiekt tej klasy. Konstruktor jednoargumentowy – przypisuje polu jego wartość.

**Number** Dziedziczy po *Token*. Zawiera pole zmiennopozycyjne *val*, przechowujące liczbę reprezentowaną przez obiekt tej klasy. Konstruktor jednoargumentowy – przypisuje polu jego wartość.

**Rpn** Nie zawiera pól. Posiada metodę statyczną *Translate*, rozkładającą napis zadany

jej w pierwszym parametrze na podciąg, z których pierwszy przypisuje się pod drugi parametr (nazwa), zaś pozostałe (wyrażenie) przetwarza się zgodnie z algorytmem „stacji rozrządowej” Dijkstry do wyrażenia w ONP, reprezentowanego jako kolejka żetonów (biblioteczna klasa `System.Collections.Generic.Queue`), i zwraca. Metoda rzuca wyjątek `ParenMismatch` w przypadku błędu rozstawienia nawiasów, lub `InvalidExpression` w przypadku innych błędów przetwarzania wyrażenia.

## 2.2 Biblioteka `libdict`

Przestrzeń nazw: `RbtDictionary`

Klasy: `Słownik<K,V>`

**Słownik<K,V>** Implementuje generyczny słownik na drzewie czerwono-czarnym. Zawiera następujące pola:

- `key` klasy `K` (implementującej generyczny interfejs `System.IComparable<K>`), przechowujące klucz węzła drzewa,
- `val` klasy `V`, przechowujące wartość węzła,
- `parent`, `left`, `right` klasy `Słownik<K,V>`, przechowujące odpowiednio rodzica, lewego i prawego syna węzła,
- boolowskie `color` (reprezentujące kolor) oraz `isempty` (techniczne, mówiące o pustoci węzła).

Metody implementują operacje sprawdzania, wstawiania i usuwania na drzewie czerwono-czarnym zgodnie z opisem<sup>1</sup>. Publiczne metody `Add`, `ContainsKey`, `TryGetValue`, `Remove` oraz właściwość `this` zgodne są z interfejsem `System.Collections.Generic.IDictionary<K,V>`. Metoda publiczna `inorderWalk` przegląda drzewo w porządku infiksowym, wykonując funkcję przekazaną przez delegata w parametrze, biorącą za argumenty klucz i wartość węzła.

Konstruktor bezargumentowy, inicjalizuje pola `isempty` i `parent`.

## 2.3 Biblioteka `libexpr`

Przestrzeń nazw: `ExpressionTrees`

Klasy: `Expression`, `Constant`, `Variable`, `UnaryExpression`, `BinaryExpression`, `UnaryMinus`, `Plus`, `Minus`, `Asterisk`, `Slash`

Wyjątki: `SelfReferenceExpression`

**Expression** Abstrakcyjna. Szczyt hierarchii klas węzłów drzew wyrażen. Zawiera następujące pola:

- wyliczeniowe `iseval`, opisujące stan obliczenia drzewa wyrażenia o drzewie zakorzenionym w węźle,
- zmiennopozycyjne `val`, przechowujące (po obliczeniu) wartość tego wyrażenia.

Metoda wirtualna `eval` wymusza istnienie w podklasach metod wyliczających wartość wyrażenia. Metoda statyczna `FromRpn` buduje drzewa wyrażen z ich zapisu w ONP, danego kolejką żetonów w pierwszym parametrze, implementując typowy algorytm ze stosom. Wyrażenie jest upraszczane w trakcie konstrukcji (tj. podwyrażenia nie zawierające zmiennych są obliczane w biegu). Drugi parametr jest klasy

---

<sup>1</sup>[http://en.wikipedia.org/wiki/Red-black\\_trees](http://en.wikipedia.org/wiki/Red-black_trees)

*Słownik* $\langle$ *string*,*Expression* $\rangle$  i służy do konstrukcji nowych obiektów podklasy *Variable*.

Konstruktor bezargumentowy, inicjalizuje pole `iseval` na `State.No`.

**Constant** Dziedziczy po *Expression*. Nie zawiera własnych pól. Metoda `eval` nie wykonuje obliczeń. Konstruktor jednoargumentowy – przypisuje polu `val` jego wartość, oraz ustawia `iseval` na `State.Yes`.

**Variable** Dziedziczy po *Expression*. Zawiera następujące pola własne:

- napis `name`, przechowujący nazwę zmiennej,
- `sl` klasy *Słownik* $\langle$ *string*,*Expression* $\rangle$ , przechowujący słownik, w którym zmienna jest przechowywana.

Metoda `eval` rzuca wyjątek *SelfReferenceException* jeśli `iseval` wynosi `State.Pending` – oznacza to odwołania cykliczne. W przeciwnym przypadku ustawia `iseval` na `State.Pending`, oblicza wartość wyrażenia znajdującego się w słowniku `sl` na pozycji `name`, przypisuje ją polu `val` i ustawia `iseval` na `State.Yes`. Konstruktor dwuargumentowy – przypisuje polom `name`, `sl` ich wartości.

**UnaryExpression** Abstrakcyjna, dziedziczy po *Expression*. Zawiera własne pole `sub` klasy *Expression*, przechowujące drzewo podwyrażenia, do którego zaaplikowany jest operator unarny. Konstruktor jednoargumentowy – przypisuje polu `sub` jego wartość.

**BinaryExpression** Abstrakcyjna, dziedziczy po *Expression*. Zawiera własne pola `left`, `right` klasy *Expression*, przechowujące drzewa podwyrażeń, do których zaaplikowany jest operator binarny. Konstruktor dwuargumentowy – przypisuje polom `left`, `right` ich wartość.

**UnaryMinus** Dziedziczy po *UnaryExpression*. Nie zawiera własnych pól. Metoda `eval` wylicza `sub.eval()` i ze zmienionym znakiem zapisuje do `val`.

**Plus, Minus, Asterisk, Slash** Dziedziczą po *BinaryExpression*. Nie zawierają własnych pól. Metoda `eval` wylicza `left.eval()`, `right.eval()`, wykonuje na nich odpowiednią operację arytmetyczną, wynik zapisując do `val`.

## 2.4 Program sheet

Klasy: *Run*

**Run** Nie zawiera pól. Posiada metodę statyczną `Main` – *entry point* programu. Metoda wczytuje linie ze standardowego wejścia, póki nie jest ono puste, przetwarza je metodą `Rpn.Translate`, umieszcza w słowniku, po czym wywołuje na nim metodę `inorderWalk` z parametrem będącym delegatem powstałym z funkcji obliczającej wartość wyrażenia i drukującej ją wraz z kluczem. Metoda łapie wyjątki zarówno przy rozbiorze, jak i obliczaniu wartości wyrażeń, wypisując odpowiednie komunikaty na ekranie i kończąc pracę z różnymi kodami wyjścia.

## 3 Zastosowania i rozwój

Elementami projektu są klasy przetwarzające wyrażenia arytmetyczne z formy czytelnej dla człowieka na postaci czytelne i łatwo obliczalne dla maszyny – postfiksową i drzewiastą.

Inną ważną klasą jest słownik polimorficzny oparty o drzewo zbalansowane. Zastosowania takich klas są wprost niezliczone.

Projekt jako aplikacja użytkowa jest w stadium bardzo mało zaawansowanym, jednak jest łatwy w rozwijaniu. Chodzi tu przede wszystkim o implementację większej gamy wyrażeń (potęgowanie, funkcje) – co uzyskać można przez poszerzenie hierarchii klas żetonów i wyrażeń, oraz proste modyfikacje funkcji `Rpn.Translate` i `Expression.FromRpn`. Kolejnym stadium mogłoby być zmodyfikowanie `Rpn.Translate` tak, można jej było zadać zbiór operatorów wraz z ich łącznością i priorytetem wiązania jako parametr z zewnątrz. Z kolei klasa `Słownik<K,V>` wymagałaby dopisania metod, które pozwoliłyby jej implementować interfejs `System.Collections.Generic.IDictionary<K,V>`.