Zaawansowane C++: Zadanie II
Diagram klas

Kamila Korzec

6 kwietnia 2018

Follow up: Zagadnienia do rozważenia

Zdecydowano się na wyodrębnienie dwóch iteracji aplikacji - wersji podstawowej (*MVP*) i rozszerzonej (*nice-to-have*). W pierwszej kolejności zaimplementowane zostanie MVP, które, w zależności od pozostałego czasu i możliwości, będzie następnie rozszerzane o dodatkowe funkcjonalności.

1. **Jaki tokenizer?**

   Zdecydowano się na użycie biblioteki C++ *Boost Tokenizer*.

   Dokumentacja biblioteki: *http://www.boost.org/doc/libs/1_36_0/libs/tokenizer/index.html*

2. **Rozpoznawanie i obsługa stałych (e, π)**

   W podstawowej wersji użycie stałej we wzorze funkcji będzie zwracało błąd. W rozszerzonej wersji zakodowane zostaną standardowe stałe możliwe do użycia - wystąpienie tej stałej będzie skutkowało przekonwertowaniem jej na zakodowaną liczbę podczas obliczania wartości funkcji w punkcie.

3. **Rozpoznawanie zmiennej - x czy dowolna litera?**

   Zaimplementowanie funkcjonalności umożliwiającej jednoczesne wprowadzanie stałych przez użytkownika i różnych oznaczeń zmiennych jest wykonalne, ale wymaga odpowiedniego poinformowania użytkownika o jego możliwościach i ograniczeniach. W programie jedyną rozpoznawaną zmienną będzie x. Dopuszczalny format wzoru (użyte stałe, symbol zmiennej) zostanie przedstawiony użytkownikowi w GUI.

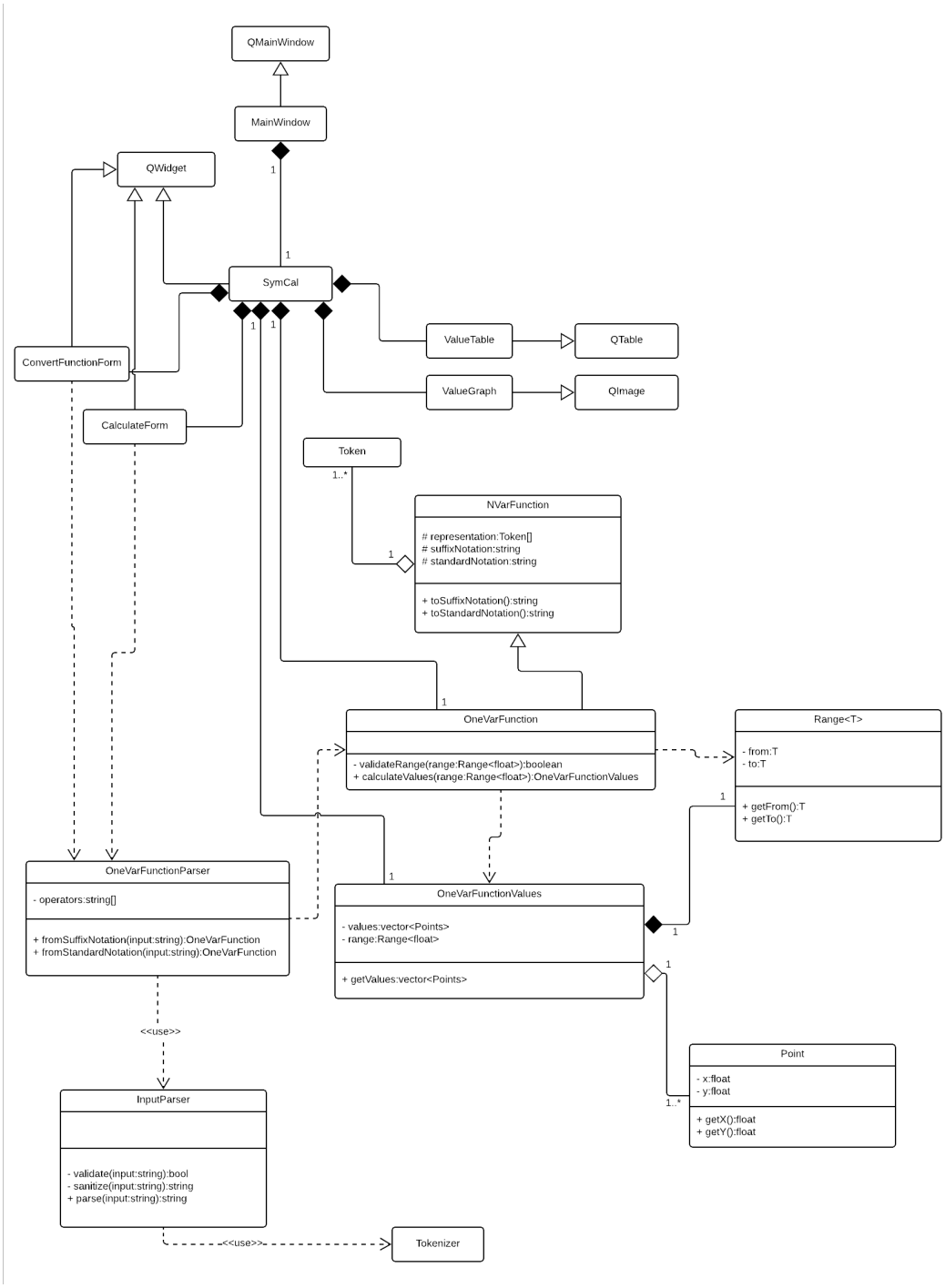
4. **Rozpoznawanie niestandardowych wyrażeń (potęgi, pierwiastki)**

   Wspierane działania: dodawanie, odejmowanie, dzielenie, mnożenie, potęga ($**$), pierwiastek kwadratowy (*sqrt*). Nieplanowane jest wspieranie pozostałych rodzajów wyrażeń (np. funkcji trygonometrycznych)

5. **Wyświetlanie błędów - osobne okienko, stałe miejsce w UI czy zależne od kontekstu?**

   Wszystkie komunikaty błędu będą wyświetlane w tym samym miejscu - rozwijanym czerwonym pasku na górze interfejsu. Dodatkowo, w przypadku nieprawidłowych danych wprowadzonych przez użytkownika, (jeśli możliwa do ustalenia) kontrolka z nieprawidłowymi danymi zostanie podświetlona na czerwono.

Diagram klas



Opis klas

Metody i właściwości klas (poza klasami z bibliotek i klasą Token, której implementacja nie została jeszcze zaplanowana) zostały zawarte w diagramie klas. Poniżej znajdują się podstawowe odpowiedzialności klas przedstawionych na diagramie:

- **Klasy związane z interfejsem graficznym**: Klasy dziedziczące po klasach z biblioteki Qt odpowiedzialne za obsługę interfejsu graficznego. Struktura zależności tych klas na diagramie została uproszczona jako, że stanowi ona szczegół implementacyjny wynikający z wykorzystania biblioteki Qt. Główna klasa - **SymCal** - jest odpowiedzialna za synchronizację pomniejszych klas (widgetów) oraz połączenie z warstwą logiki.

- **Token**: obecnie klasa/wydmuszka - dokładna implementacja klasy będzie znana w przyszłości. Reprezentacja funkcji będzie przechowywana jako tablica tokenów (w najprostszej wersji - stringów), jednoznacznie charakteryzująca daną funkcję. Na podstawie tej reprezentacji tworzone są następnie postacie funkcji w obu notacjach, oraz obliczane są wartości funkcji w punkcie.

- **NVarFunction**: klasa abstrakcyjna reprezentująca funkcję n zmiennych. Zawiera uniwersalny format postaci funkcji (tablica tokenów) oraz dwa formaty notacji.

- **OneVarFunction**: klasa dziedzicząca po VarFunction, stanowiąca reprezentację funkcji jednej zmiennej oraz dodatkową metodę używaną do obliczania wartości funkcji w punkcie.

- **OneVarFunctionValues**: funkcja przechowująca wartości funkcji dla zadanego przedziału.

- **Point**: prosta klasa przechowująca współrzędne punktu (x, y).

- **Range<T>**: szablon prostej klasy przechowującej zakres, dla którego wyliczana jest wartość funkcji w punkcie. W programie wykorzystana będzie klasa Range<float>, przechowująca zakres od-do w postaci dwóch floatów.

- **OneVarFunctionParser**: parser zawierający dwie statyczne metody, tworzące instancję klasy OneVarFunction w zależności od wprowadzonego formatu, oraz przechowujący listę możliwych operatorów w postaci tablicy stringów.

- **InputParser**: parser przygotowujący wprowadzony przez użytkownika format, aby był możliwy do użycia w Tokenizerze.

- **Tokenizer**: zewnętrzna klasa z biblioteki Boost (patrz: punkt 1. w *Follow-up*).