



**POLITECHNIKA
RZESZOWSKA**
im. IGNACEGO ŁUKASIEWICZA

PROJEKT NA PRZEDMIOT
LabView – akwizycja danych pomiarowych

Treść zadania:

„Zaimplementować interpreter funkcji”

Wykonane przez:

Vitalii Morskyi

Kacper Mosoń

Kacper Piędel

Miłosz Ostasz

1. Słownik pojęć

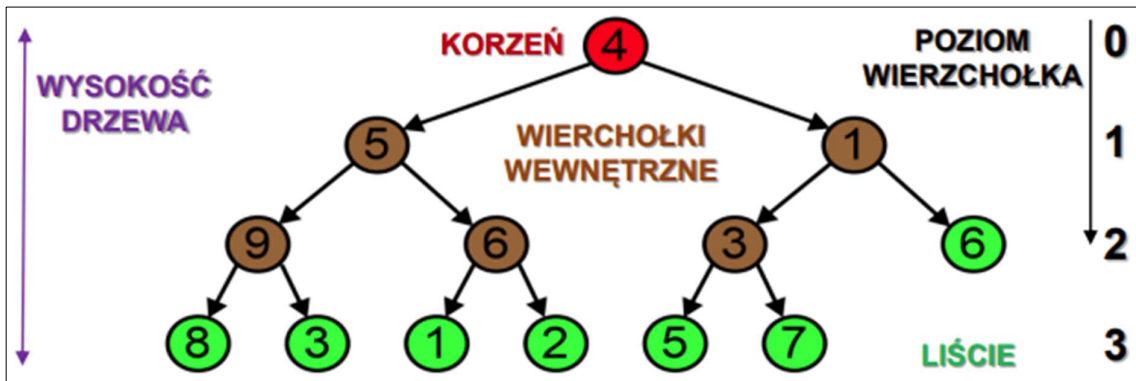
Interpreter – program, który zajmuje się tłumaczeniem kodu języka programowania na język maszynowy i jego wykonaniem.

Funkcje – fragmenty kodu zamknięte w określonym przez programistę symbolu, mogące przyjmować parametry oraz mogące zwracać wartości.

Zmienne – symbole definiowane i nazywane przez programistę, które służą do przechowywania wartości, obliczeń na nich i odwoływaniu się do wartości przez zdefiniowaną nazwę.

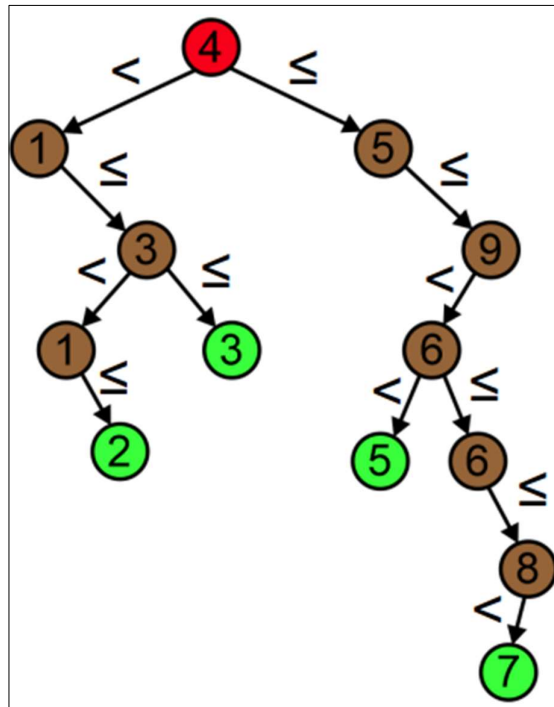
2. Drzewo binarne

Drzewo to struktura danych składająca się z wierzchołków (węzłów) i krawędzi, przy czym krawędzie łączą wierzchołki w taki sposób, iż istnieje zawsze dokładnie jedna droga pomiędzy dowolnymi dwoma wierzchołkami.



Wierzchołki w drzewach przedstawiamy w postaci warstwowej, tzn. każdy wierzchołek w drzewie znajduje się na jakimś poziomie. Poziom wierzchołka w drzewie jest równy długości drogi łączącej go z korzeniem. Korzeń drzewa jest na poziomie 0. Wysokość drzewa równa jest maksymalnemu poziomowi drzewa, czyli długości najdłuższej spośród ścieżek prowadzących od korzenia do poszczególnych liści drzewa. Wierzchołki mogą posiadać rodzica, który jest umieszczony na wyższym poziomie oraz dzieci, które są umieszczone na niższym poziomie. Niektóre dzieci nie posiadają własnych dzieci i są liśćmi. Dzieci jednego rodzica nazywamy rodzeństwem. Wierzchołki, które nie posiadają ani jednego dziecka nazywamy liśćmi. Przodkami są rodzice oraz rekurencyjnie rodzice rodziców. Potomkami są dzieci oraz rekurencyjnie dzieci dzieci. Wierzchołki posiadające zarówno rodzica jak i przynajmniej jedno dziecko nazywamy wierzchołkami wewnętrznymi. Każde drzewo posiada wyróżniony, nie posiadający rodzica wierzchołek, który nazywamy korzeniem.

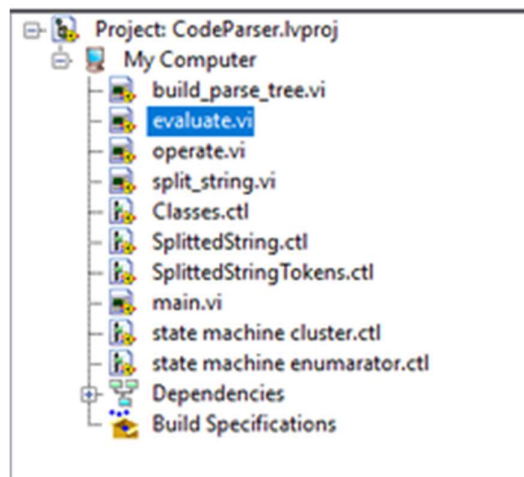
Binarne drzewo poszukiwań (BST - binary search tree) to binarne drzewo reprezentujące elementy multizbioru w taki sposób, iż każdy wierzchołek ma po lewej elementy mniejsze, a po prawej większe od reprezentowanej wartości klucza przez ten wierzchołek:



Odnajdywanie miejsca wstawienia nowego elementu polega na przechodzeniu od korzenia po węzłach drzewa w taki sposób, iż jeśli wartość jest mniejsza, wtedy idzie się w lewo, a jeśli większa lub równa w prawo. Przechodzenie po drzewie jest dokonywane tak długo, dopóki nie natrafimy na sytuację, gdy węzłowi brak węzła lub liścia w pożądaną stronę. Tam jest dodawany nowy element. Drzewa BST nie posiadają żadnego mechanizmu wyważania, więc w pesymistycznym przypadku może zostać zbudowana lista – czyli trywialna postać drzewa.

3. Budowa programu

Program zbudowany jest z czterech subVI:



Rysunek 1 Budowa programu

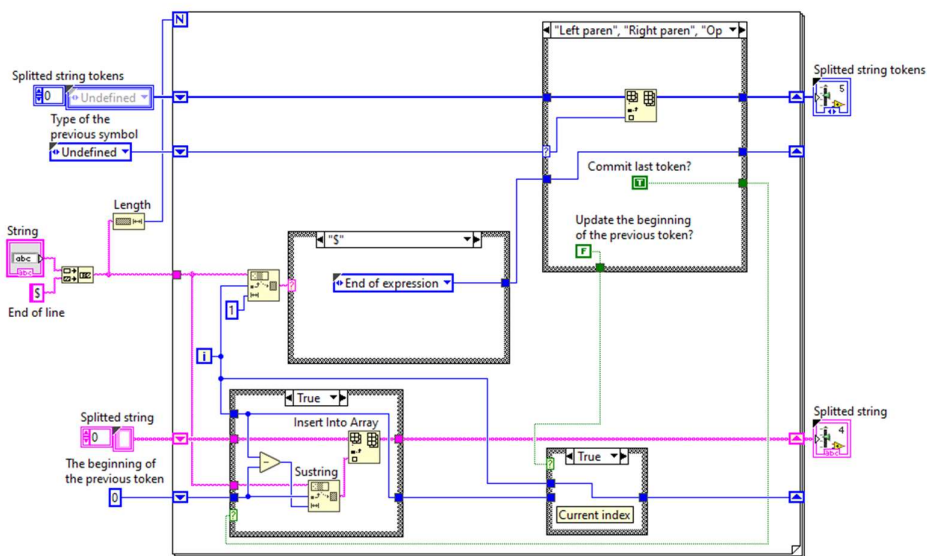
- **build_parse_tree.vi** – odpowiada za budowę drzewa binarnego
- **evaluate.vi** – odpowiada za obliczenie wartości zadanej funkcji
- **operate.vi** – wykonuje najmniejszą atomową operację
- **split_string.vi** – rozpoznaje każdy znak wprowadzony do funkcji i określa jego typ
- **substitute.vi** – rysuje wykres zadanej funkcji

Opis poszczególnych subVI:

- **split_string.vi**

W tym subVI program określa typ danego znaku. Rozpoznaje on czy dany znak jest:

- number – liczba
- operator arytmetyczne – operator potęgowania (^), operator mnożenia (*), operator dzielenia (/), operator dodawania (+), operator odejmowania (-)
- left paren – nawias otwierający
- right paren – nawias zamykający
- variable – zmienna, np. x, y, alamaKota
- high level function – funkcje trygonometryczne (tj. sin, cos, tg, ctg) oraz exp



Rysunek 2 split_string.vi

Przykład analizowania zadanej funkcji przez subVI **split_string.vi**:

Wprowadzamy funkcję:

$$(3.5 + 6)$$

Etapy analizy przez program:

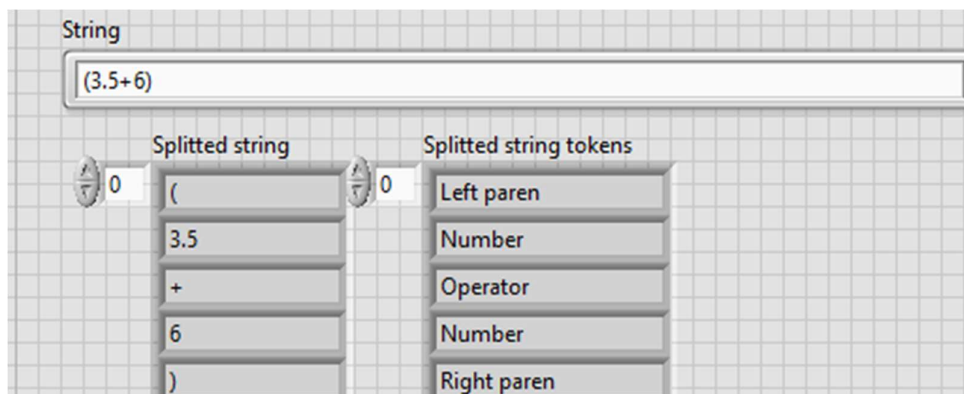
K1: 3 → liczba

K2: 3. → liczba

K3: 3.5 → liczba

K4: 3.5 + → program zapisuje liczbę 3.5 i rozpoznaje operator dodawania

K5 3.5 + 6 → program zapisuje operator dodawania i rozpoznaje liczbę 6



Działanie programu z funkcją, która zawiera High Level Function i zmienne:

Wprowadzamy funkcję:

$\exp((x + 45))$

Etapy analizy przez program:

K1: $e \rightarrow$ zmienna

K2: $ex \rightarrow$ zmienna

K3: $\exp \rightarrow$ zmienna

K4: $\exp(\rightarrow$ High Level Function

K5: $\exp((\rightarrow$ program zapisuje HLF $\exp()$ i rozpoznaje lewy nawias

K5 $\exp((x \rightarrow$ program zapisuje lewy nawias i rozpoznaje zmienną

K7: $\exp((x+ \rightarrow$ program zapisuje zmienna i rozpoznaje operator dodawania

K8: $\exp((x+4 \rightarrow$ program zapisuje operator i rozpoznaje liczbę

K9: $\exp((x+45 \rightarrow$ liczba

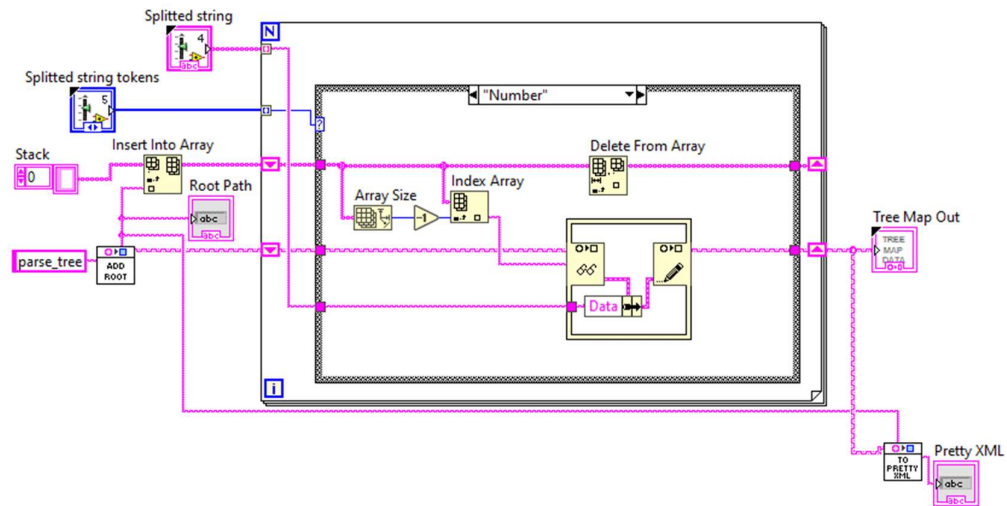
K10: $\exp((x+45) \rightarrow$ program zapisuje liczbę i rozpoznaje prawy nawias

K11: $\exp((x+45)) \rightarrow$ program zapisuje prawy nawias i rozpoznaje prawy nawias

String		
<input type="text" value="exp((x+45))"/>		
	Splitted string	Splitted string tokens
0	exp(0 High level function
	(Left paren
	x	Variable
	+	Operator
	45	Number
)	Right paren
)	Right paren

- ***build_parse_tree.vi***

Ten subVI odpowiada za generowanie drzewa binarnego.



Rysunek 3 *build_parse_tree.vi*

Dla przykładu wprowadzamy działanie algebraiczne:

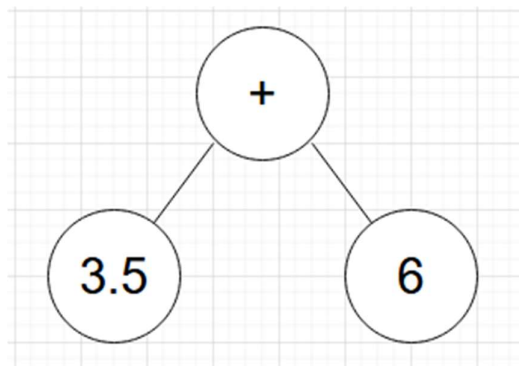
a)

$$(3.5 + 6)$$

Program tworzy kod drzewa binarnego w języku XML dzięki funkcji *to_pretty_xml*, która znajduje się w bibliotece *tree_max*. Poniżej przedstawiamy wynik tej operacji.

```
<parse_tree>
+
  <left>3.5</left>
  <right>6</right>
</parse_tree>
```

Rysunek 4 *parse_tree_XML*

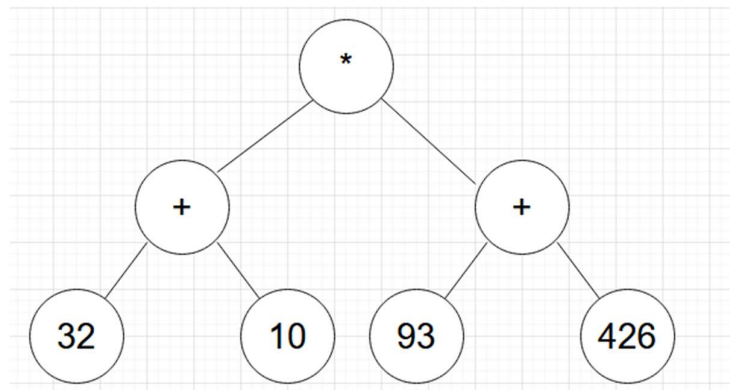


b)

$$((32 + 10) * (93 + 426))$$

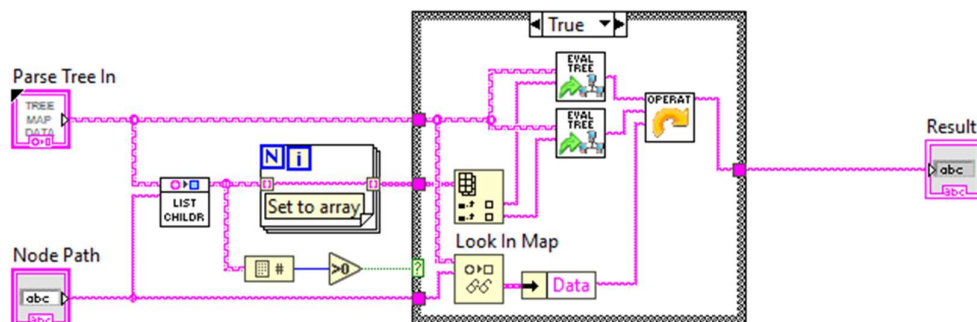
```
Pretty XML
<parse_tree>
*
  <left>
  +
    <left>32</left>
    <right>10</right>
  </left>
  <right>
  +
    <left>3</left>
    <right>426</right>
  </right>
</parse_tree>
```

Rysunek 5 parse_tree_XML



- **evaluate.vi**

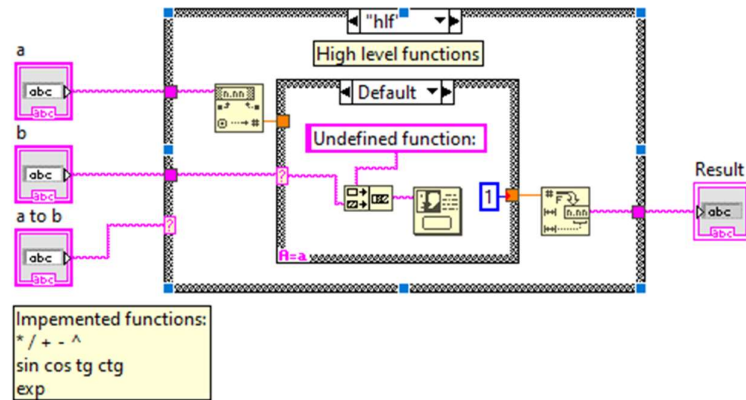
Zadaniem subVI **evaluate.vi** jest sprowadzenie stworzonego drzewa zadanej funkcji do stanu gdy zostanie tylko do wykonania najmniejsza atomowa operacja, która wykonywana jest już dzięki funkcji **operate.vi**.



Rysunek 6 evaluate.vi

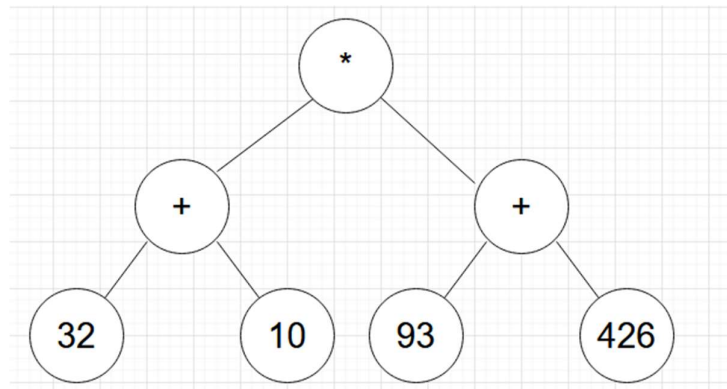
- **operate.vi**

Ten subVI wykonuje najmniejszą atomową operację zadanej funkcji.



Rysunek 7 operate.vi

Przykład:



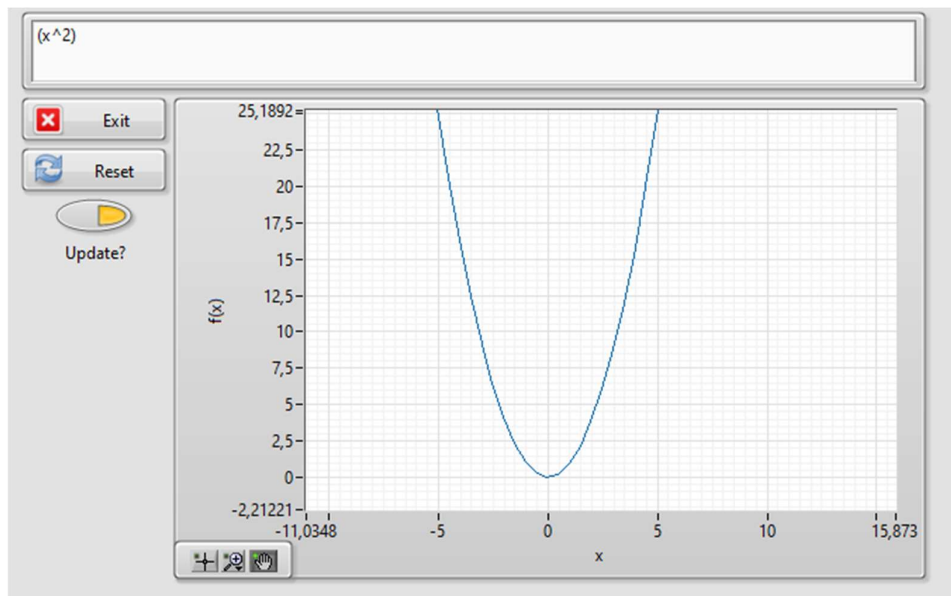
Dla tego przykładu najmniejszą atomową operacją funkcji jest mnożenie.

- ***substitute.vi***

W programie dla lepszego przedstawiania wartości funkcji został stworzony wykres, który używa granic wykresu w aktualnej pozycji. Dzięki temu łatwo zobaczyć dokąd wykres został narysowany.

Przykład dla zadanych funkcji:

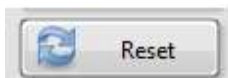
a) $f(x) = (x^2)$



Rysunek 8 Wykres dla funkcji $f(x)=x^2$



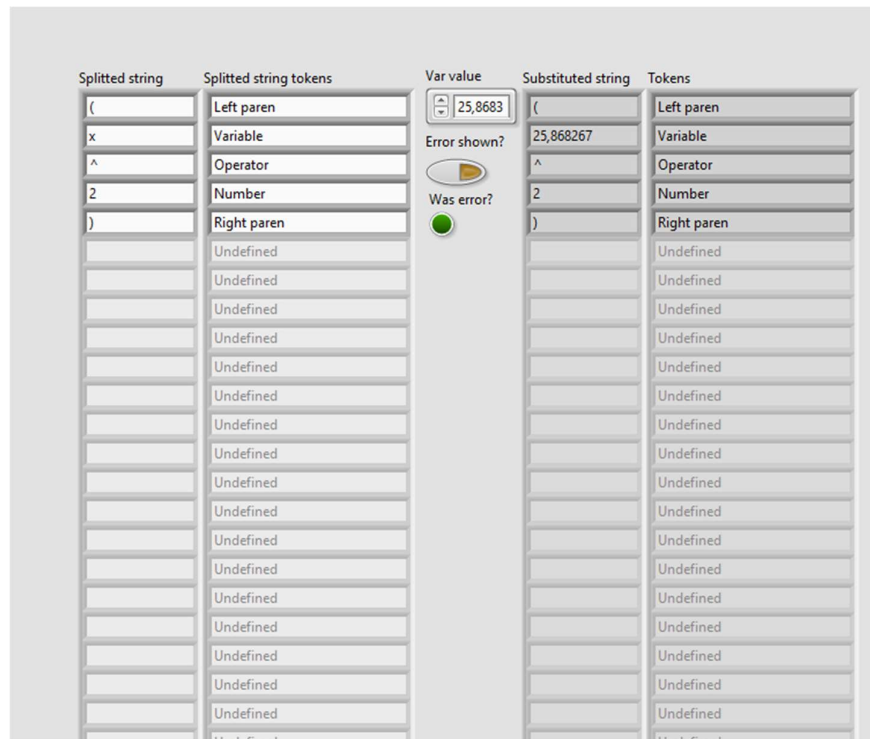
- wyjście/zakończ



- ustawia zakres x i y na przedziale $\langle -1, 1 \rangle$



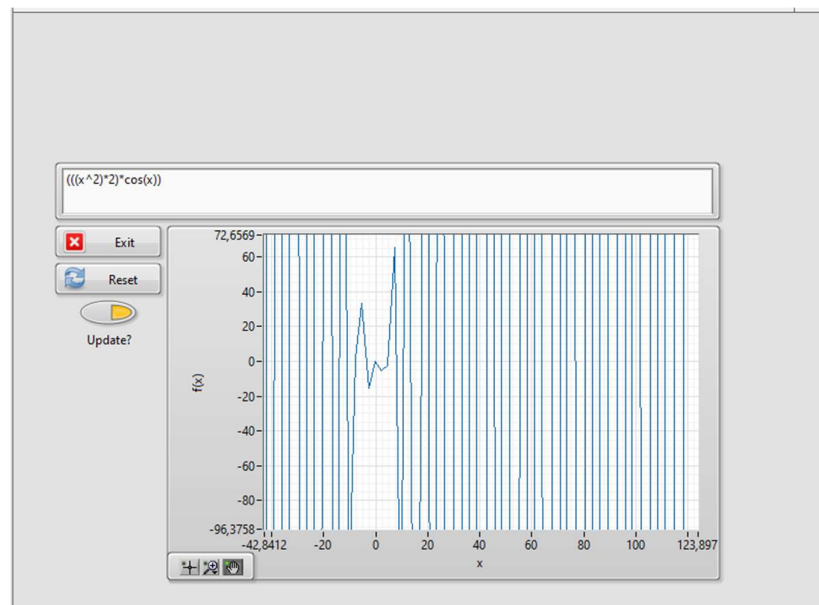
- przycisk może mieć dwa stany on/off, w jednym i drugim stanie dane są pobierane i obliczenia się nadal wykonują lecz wykres się nie aktualizują (off)





Rysunek 9 substitute.vi dla funkcji $f(x) = x^2$

Jak możemy zauważyć po lewej stronie widoku wyświetlone są poszczególne znaki, z jakich składa się zadana funkcja, dzięki subVI **split_string.vi**. Natomiast po prawej stronie w miejsce rozpoznanej zmiennej podstawiona zostaje górna granica aktualnego przedziału.

c) $f(x) = (((x^2) \cdot 2) \cdot \cos(x))$



Rysunek 10 Wykres dla funkcji $f(x) = 2x^2 \cdot \cos(x)$

Splitted string	Splitted string tokens	Var value	Substituted string	Tokens
(Left paren	117,271	(Left paren
(Left paren	Error shown?	(Left paren
(Left paren		(Left paren
x	Variable	Was error?	117,271149	Variable
^	Operator		^	Operator
2	Number		2	Number
)	Right paren)	Right paren
*	Operator		*	Operator
2	Number		2	Number
)	Right paren)	Right paren
*	Operator		*	Operator
cos(High level function		cos(High level function
x	Variable		117,271149	Variable
)	Right paren)	Right paren
)	Right paren)	Right paren
	Undefined			Undefined
	Undefined			Undefined
	Undefined			Undefined
	Undefined			Undefined
	Undefined			Undefined
	Undefined			Undefined
	Undefined			Undefined

Rysunek 11 substitue.vi dla funkcji $f(x) = 2x^2 \cdot \cos(x)$