



**KTH Computer Science
and Communication**

SimpleGraphPlotter v1.6

Programkonstruktion för F, DD1342
Laboration 4A

JIM HOLMSTRÖM
JIMHO@KTH.SE

Teacher: Ann Bengtson

Contents

1	Introduction	1
1.1	Requirements	1
1.2	Scope	1
1.3	Assistance	2
2	Structure	3
2.1	Parser	3
2.1.1	interface iparser	3
2.1.2	class parser	3
2.1.3	function_container	4
2.1.4	interface iexpression	4
2.1.5	class constant	4
2.1.6	class variable	4
2.1.7	class unary_operation	4
2.1.8	class binary_operation	5
2.2	Plotter	5
2.2.1	class function	5
2.2.2	class plot_drawingarea	5
2.2.3	class function_list_controller	5
3	Results and Discussion	9
3.1	Results	9
3.2	Discussion	9

Chapter 1

Introduction

In the following part firstly the problem will be explained and secondly the requirements for a basic plotter will be enlisted. A plotter is a program that can plot functions from strings which defines the functions by ordinary math syntax. This project uses `C++` programming language and the `gtkmm`¹ wrapper for the `GTK+`² toolkit to generate the graphical user interface. It is compiled with the `GNU gcc` compiler.

1.1 Requirements

A few basic things is needed to have a functioning math plotter:

1. Define a function given ordinary math syntax.
2. Parse the inputed function and plot it accordingly.
3. Add/Remove functions from plotarea.
4. Plotarea should be scrollable both vertical and horizontal.
5. Range should be fixed to the unit-cube.³
6. Display axis of the plot.
7. Parser must be properly tested.

1.2 Scope

The amount of functionality that is possible to put in a system like this is almost endless so a few delimitations has to be made in order to complete the project. The

¹Documentation, binaries and source can be found at: www.gtkmm.org

²Documentation, binaries and source can be found at: www.gtk.org

³This restriction will be handled in section 1.2

currently biggest restriction to the plotter is the lack of ability to zoom or change the range from the unit-cube. No support for parametric nor complex functions.⁴

1.3 Assistance

Besides the reference manuals for `gtkmm` no external help for this project was received.

⁴ Since no native support in `C++` for complex numbers which means all the basic math functions would have to be rewritten in order for this to work.

Chapter 2

Structure

An basic overview of the structure can be seen in figure 2.1, all public non-self-explanatory parts will then be enlisted and explained in a `javadoc` like manner. In the actual code the definition and implementation was separated into `.h` and `.cpp`-files respectively as long as possible,¹ in a `C++` manner.

2.1 Parser

The parser code can be divided into to parts the algorithm code, that is the actual parser, and the data structure in the form of a parse tree.^{2.2}

2.1.1 interface iparser

2.1.2 class parser

The parser is an implementation of a *recursive descent parser*. To types of methods are used in the parsing, `is-a`² and `read-it`³. The `is-a` is used for lookahead to determine which type of expression lays ahead, while `read-it` is used to do the actual syntactic information gathering from the expression fragment.

The EBNF syntax for the parsing made by this algorithm is as follows:

```
plots = term-(-1),[';',',',expression-(-1)],'\n' (* no support in this
implementation *)
expression-i = [unary-i],expression-(i+1),[op-(i+1),expression-i]  \\\
(* -1 is the lowest order expression *) \\\
(* either unary-(i+1) or op-(i+1), unary (since on the left) \\\
has higher priority *)
```

¹Some small trivial methods were left out from this distinction as well as a few things that is hard or impossible to separate.

²Starts with `is_`

³Starts with `read_`

```

term-n = var | num | [function],(,term-(-1),) \\
(* n is the number of the highest order operator *) \\
(* if function is left out it will be handled as the unit function *)

op-0 = '>' | '<'
op-1 = '+' | '-'
op-2 = '*' | '/' | '%'
op-3 = '^'
unary-3 = '+' | '-' | '*'
num = ? all numbers ?
var = 'x'
function = cos | sin | tan | acos | asin | atan | cosh \\
| sinh | tanh | exp | log | log10 | sqrt | ceil | abs \\
| floor | pi | e (* where pi and e are constant
functions *)

```

public parse(expr : std::string) Parses the string `expr` to generate a parse tree that represents the math expression in `expr`.

Parameters:

`expr` - The string to be parsed.

Returns:

`iexpression*` - Returns a pointer to the root of the parse tree.

Public <basic description of the function> TODO should i perhaps move the arguments/return as i doxygen to their own posts?

2.1.3 function_container

what is this used for? compare to GUI>function and use ref.

2.1.4 interface iexpression

2.1.5 class constant

2.1.6 class variable

2.1.7 class unary_operation

public unary_operation(op : unary_op, left : iexpression*) Constructor ...
blabla

Parameters:

`op` - «» an `unary_op`⁴ .
`left` - «» .

⁴Typedefined to be a function pointer: `*binary_op(double):double`.

2.2. PLOTTER

Public <basic description of the function> TODO should i perhaps move the arguments/return as i doxygen to their own posts?

2.1.8 class binary_operation

2.2 Plotter

... <images with the different parts highlighted with a red border, that is the parts being described at the moment> especially point out the inheritance in the custom widgets.

2.2.1 class function

Acts as a view for one function

2.2.2 class plot_drawingarea

2.2.3 class function_list_controller

CHAPTER 2. STRUCTURE

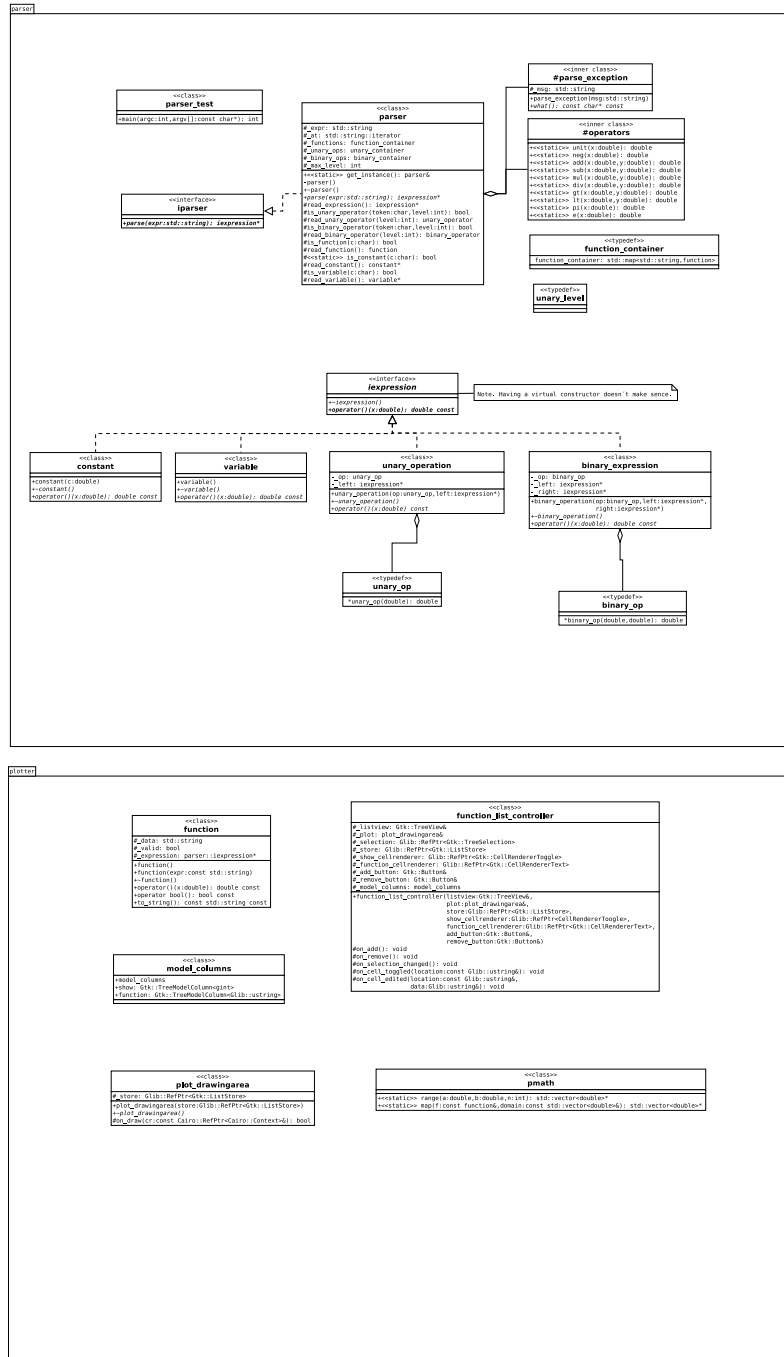


Figure 2.1. An UML showing the structure and the enclosure.

2.2. PLOTTER

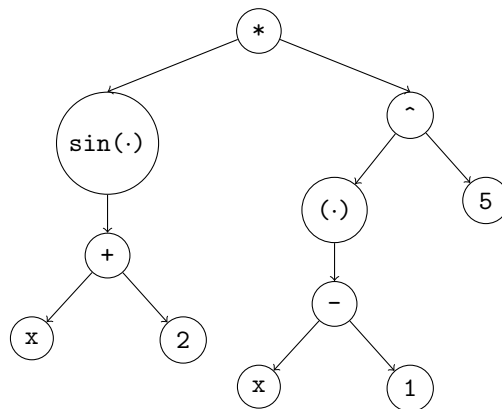


Figure 2.2. An example of the parse tree for the expression $\sin(x+2)*(x-1)^5$. Trivial nodes where left out.

Chapter 3

Results and Discussion

3.1 Results

«screenshots» Runned trough valgrind, results?.

3.2 Discussion

- = Problems with the unofficial C++wrapper `gtkmm`, only used it to avoid missing out inheritance, polymorphism and to get it compatible with the standard C++Library.
- = Easy to miss combinations in the parser and have bugs.