KTH Computer Science
and Communication

SimpleGraphPlotter v1.6

Programkonstruktion för F, DD1342
Laboration 4A

JIM HOLMSTRÖM
JIMHO@KTH.SE

Teacher: Ann Bengtson

Contents

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

[1]Documentation, binaries and source can be found at: www.gtkmm.org
[2]Documentation, binaries and source can be found at: www.gtk.org
[3]This 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.

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 parser

The parser is an implementation of a *recursive descent parser.* To types of methods are used in the parsing, `is-a` and `read-it`. 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  *)
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?

parse_exception

operators

2.1.2 unary_level

what is this used for?

2.1.3 iexpression...

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.

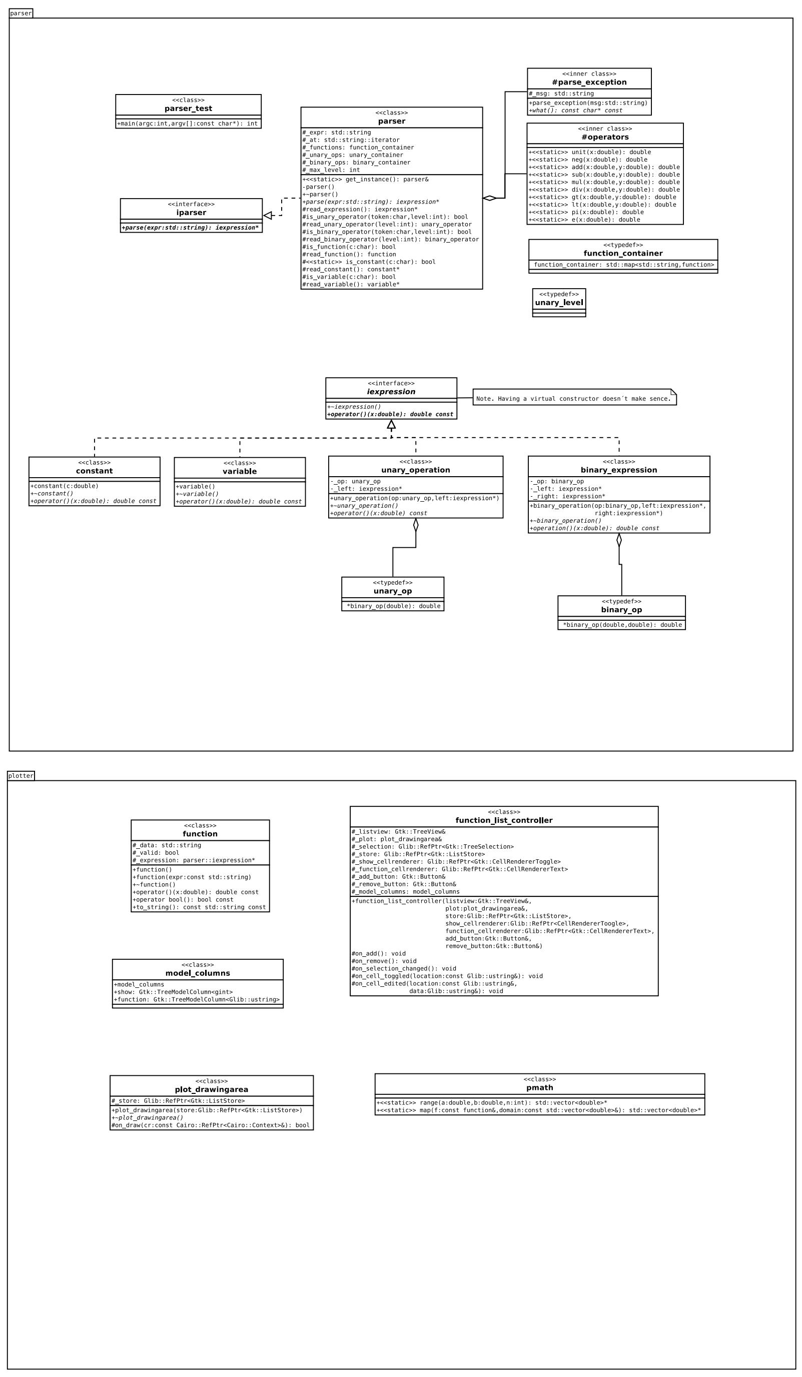
parser
<<class>>
parser_test
+main(argc:int,argv[]:const char*): int
<<interface>>
iparser
+parse(expr:std::string): iexpression*
<<class>>
parser
#_expr: std::string
#_at: std::string::iterator
#_functions: function_container
#_unary_ops: unary_container
#_binary_ops: binary_container
#_max_level: int
+<<static>> get_instance(): parser&
-parser()
+~parser()
+parse(expr:std::string): iexpression*
#read_expression(): iexpression*
#is_unary_operator(token:char,level:int): bool
#read_unary_operator(level:int): unary_operator
#is_binary_operator(token:char,level:int): bool
#read_binary_operator(level:int): binary_operator
#is_function(c:char): bool
#read_function(): function
#<<static>> is_constant(c:char): bool
#read_constant(): constant*
#is_variable(c:char): bool
#read_variable(): variable*
<<inner class>>
#parse_exception
#_msg: std::string
+parse_exception(msg:std::string)
+what(): const char* const
<<inner class>>
#operators
+<<static>> unit(x:double): double
+<<static>> neg(x:double): double
+<<static>> add(x:double,y:double): double
+<<static>> sub(x:double,y:double): double
+<<static>> mul(x:double,y:double): double
+<<static>> div(x:double,y:double): double
+<<static>> gt(x:double,y:double): double
+<<static>> lt(x:double,y:double): double
+<<static>> pi(x:double): double
+<<static>> e(x:double): double
<<typedef>>
function_container
function_container: std::map<std::string,function>
<<typedef>>
unary_level
<<interface>>
iexpression
+~iexpression()
+operator()(x:double): double const
Note. Having a virtual constructor doesn´t make sence.
<<class>>
constant
+constant(c:double)
+~constant()
+operator()(x:double): double const
<<class>>
variable
+variable()
+~variable()
+operator()(x:double): double const
<<class>>
unary_operation
-_op: unary_op
-_left: iexpression*
+unary_operation(op:unary_op,left:iexpression*)
+~unary_operation()
+operator()(x:double) const
<<class>>
binary_expression
-_op: binary_op
-_left: iexpression*
-_right: iexpression*
+binary_operation(op:binary_op,left:iexpression*, right:iexpression*)
+~binary_operation()
+operation()(x:double): double const
<<typedef>>
unary_op
*binary_op(double): double
<<typedef>>
binary_op
*binary_op(double,double): double
plotter
<<class>>
function
#_data: std::string
#_valid: bool
#_expression: parser::iexpression*
+function()
+function(expr:const std::string)
+~function()
+operator()(x:double): double const
+operator bool(): bool const
+to_string(): const std::string const
<<class>>
function_list_controller
#_listview: Gtk::TreeView&
#_plot: plot_drawingarea&
#_selection: Glib::RefPtr<Gtk::TreeSelection>
#_store: Glib::RefPtr<Gtk::ListStore>
#_show_cellrenderer: Glib::RefPtr<Gtk::CellRendererToggle>
#_function_cellrenderer: Glib::RefPtr<Gtk::CellRendererText>
#_add_button: Gtk::Button&
#_remove_button: Gtk::Button&
#_model_columns: model_columns
+function_list_controller(listview:Gtk::TreeView&, plot:plot_drawingarea&, store:Glib::RefPtr<Gtk::ListStore>, show_cellrenderer:Glib::RefPtr<CellRendererToogle>, function_cellrenderer:Glib::RefPtr<Gtk::CellRendererText>, add_button:Gtk::Button&, remove_button:Gtk::Button&)
#on_add(): void
#on_remove(): void
#on_selection_changed(): void
#on_cell_toggled(location:const Glib::ustring&): void
#on_cell_edited(location:const Glib::ustring&, data:Glib::ustring&): void
<<class>>
model_columns
+model_columns
+show: Gtk::TreeModelColumn<gint>
+function: Gtk::TreeModelColumn<Glib::ustring>
<<class>>
plot_drawingarea
#_store: Glib::RefPtr<Gtk::ListStore>
+plot_drawingarea(store:Glib::RefPtr<Gtk::ListStore>)
+~plot_drawingarea()
#on_draw(cr:const Cairo::RefPtr<Cairo::Context>&): bool
<<class>>
pmath
+<<static>> range(a:double,b:double,n:int): std::vector<double>*
+<<static>> map(f:const function&,domain:const std::vector<double>&): std::vector<double>*

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

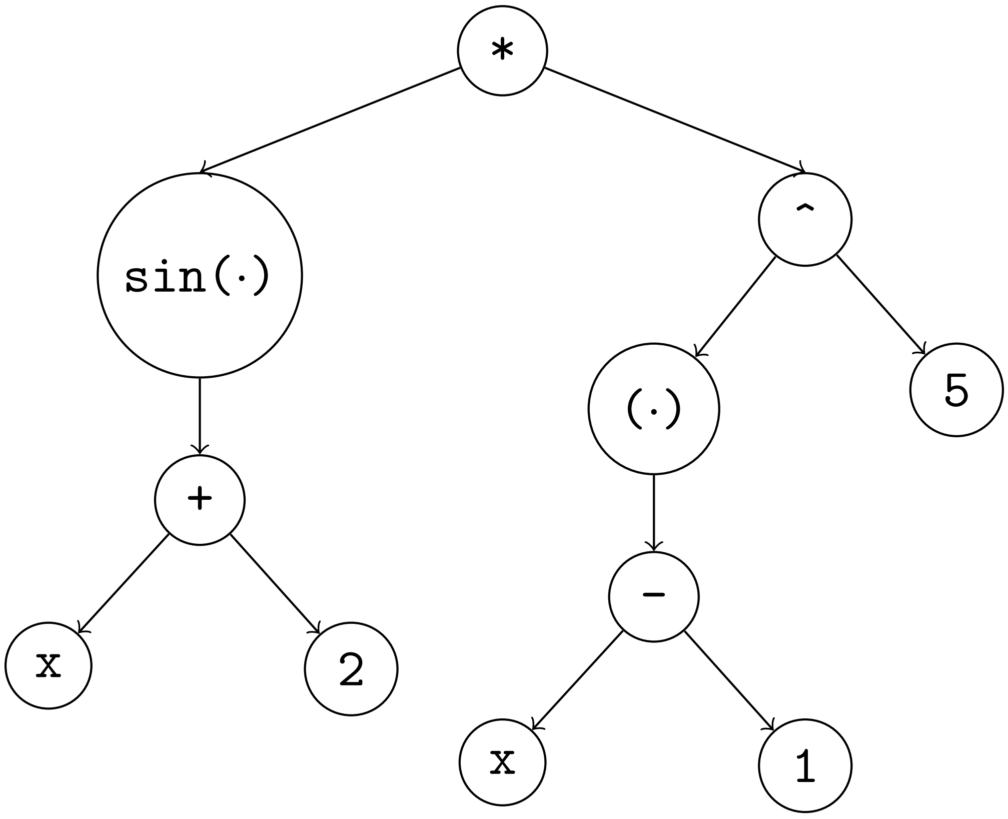

Figure 2.2. An example of the parse tree for the expression `sin(x+2)*(x-1)^5`. Trivial nodes where left out.

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.