Cairo charts

Graded ASsignment 2 - System Programming

Francesco Saverio Zuppichini

Project Structure

The project is composed by 5 source files and 4 headers:

- main.c
- my_string.c
- my_string.h
- sentinel_linked_list.c
- sentinel_linked_list.h
- parse_helper.c
- parse_helper.h
- draw_helper.c
- draw_helper.h

main.c

In this files the main functions from the others source files are called. It creates a pointer to *generic sentinel linked list* called *floatstdsll* and to *cairocharts_payload* called *my_payload*.
In detail *my_payload* is passed to *parse_helper.c* that parse and store in it the command lines parameters, on the other and *cairopointsll*, as the name suggest, is used to store the *cairo_point* struct collected from the standart input. All the std is stored into a *my_string* variable. This last parsing is done in *main.c* inside the function *store_float_into_sll*. This function converts every float value from the std into *cairo_point* that is defined in *draw_helper*
If the user select the *smoothing average* an addictional function is called, *smoothing_average*, that allocated a new *sentinel list* with the values of the smoothed point in *float_std_sll* and then it changes the pointer to that list.
After the two parsing are happened succesfully, *main.c* calls *create_cairocharts* from draw_helper.c in order to actually draw the graph.
In the end *free_memory* is called.

parse_helper.c

The porpuse of this file is parsing all the data from the command lines parameters (argv). This is done by

create_cairocharts that calls *add_default_params* and *add_command_line_params*.
The first function just add the default valus defined in your pdf, the second one parse and store into *my_payload* all the argv.
In order to do that an array of two *my_string* named *curr_param* is allocated. It will holds the params name in the first elemenet and the value in the second. The color parsing is more trickly and a special function *store_and_parse_color* is called.

draw_helper.c

This is the main file, it draws the graph. This is done in differente step and they dipends on the grahp type. The main function is *create_cairocharts* that select the correct draw function according to the type.
A special enum *cairochart_Type* defined into *parse_helper.c* is used to store the type. Before actually draw the points some action are done in *create_cairocharts*, all the cairo variable are created and the *origin* is calculated, then the variable needed for the scaling of the points are calculated too.

sentinel_linked_list.c

This is a personal implementation of a sentinel linked list that is composed by a *sll_node* where i can store a void pointed. For a more detail explanation you can see:

https://github.com/FrancescoSaverioZuppichini/Generic-Sentine-Linked-List-C

my_string.c

This is a personal implementation of a String object, it is used to store the std and all the dynamic reallocation of the memory is done into it. For a more detail explanation you can see:

https://github.com/FrancescoSaverioZuppichini/String-Implementation-C

Line plot

Line plot is used for the standard and the xplot graph type. It is created by function *draw_line_plot*. This function iterate all over the linked list and for every point into is normalized by calling *normalize_point*. Then it draws the graph using cairo API

Histogram

In the histogram we do more or less the same action done in *lineplot* but the witdh is incremented by "one" scaled point and, of course, the rectangle are drawn.

Axis and ticks

Axis are drawn into *draw_axis* and tick into *draw_lines*.

Error handling

In my implementation the returning value *1* means true and *0* means false. Booth commandline errors and std input are handled by shutting down the program. This are the error handled:

- parsing float error (e.g with=a) in argv and std
- partial command line (e.g with=)
- no points
- memory allocation errors

I did an expection for the output, if a user doesn't put the ".pdf" the program will add it automatically

Debug

You can pass the macro *DEBUG* in order to see usefull debug print

Make

I also provide a *makefile* that can be usefull to speed up the correction. You only need to run

```
make
```

inside this folder. If you want to switch to debug mode just type

```
make debug
```