

# 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 values 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 hold the params name in the first element and the value in the second. The color parsing is more tricky 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 different steps and they depend on the graph type.

The main function is *create\_cairocharts* that selects 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 drawing the points some actions are done in *create\_cairocharts*, all the cairo variables are created and the *origin* is calculated, then the variables 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 pointer. For a more detailed explanation you can see:

<https://github.com/FrancescoSaverioZuppichini/Generic-Sentinel-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 detailed 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 iterates all over the linked list and for every point it is normalized by calling *normalize\_point*. Then it draws the graph using the cairo API. You can pass "type=lineplot" for a normal graph or "type=xplot" for the x and y version.

## **Histogram**

In the histogram we do more or less the same action done in *lineplot* but the width is incremented by "one" scaled point and, of course, the rectangles are drawn. You can pass "type=histogram" to trigger this graph.

## **Axis and ticks**

Axes are drawn into *draw\_axis* and ticks into *draw\_lines*.

## Error handling

In my implementation the returning value *1* means true and *0* means false. Both 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 exception 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
```