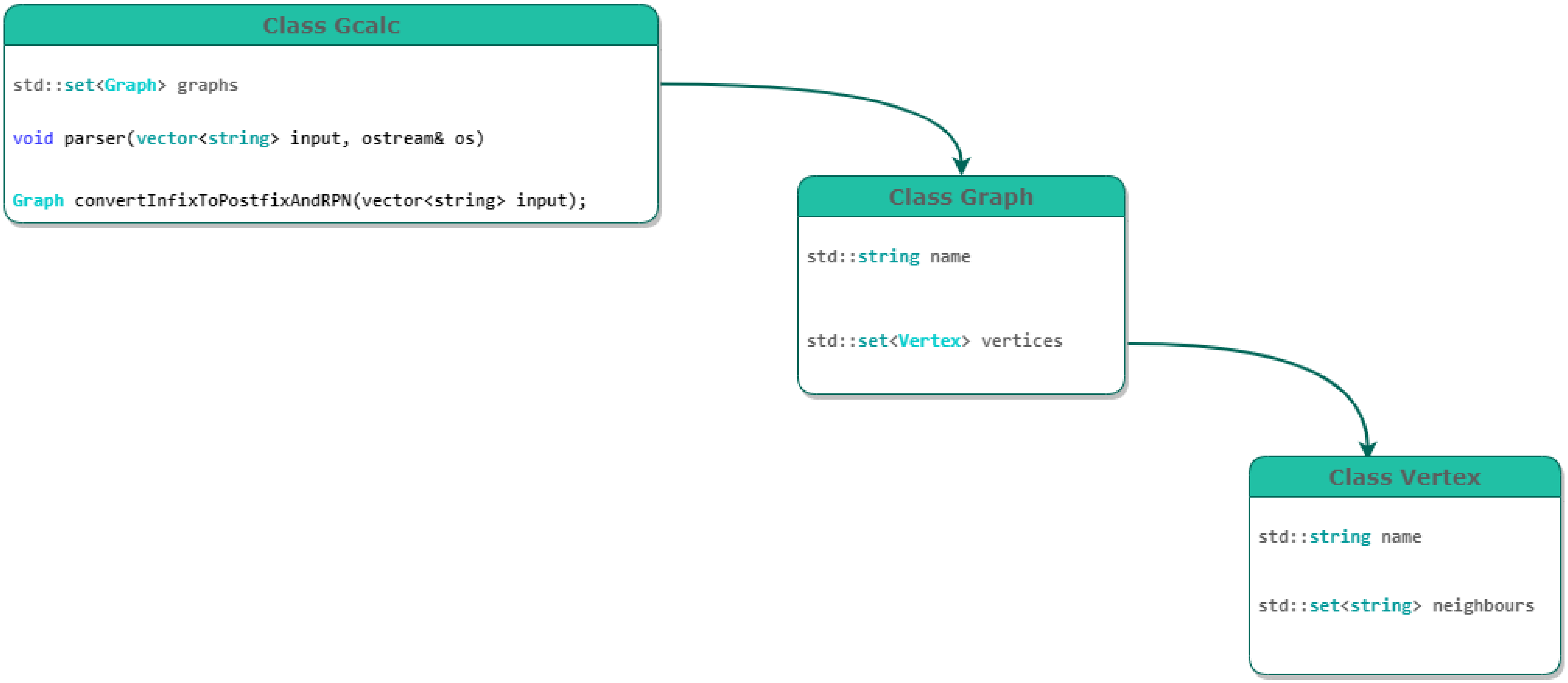
<u>The design :</u>

“Gcalc” is a container which holds all graphs created in a scope before the first “reset” command, each graph in graphs is of type “Graph”. Graph is another container which holds the name of the current graph and it’s set of vertices where each vertex is of type “Vertex”. Vertex is yet another container which holds the name of the current vertex and it’s set of neighbours (the set holds the names of the goal vertices – the other vertices which our current vertex sends an edge to)



```
std::set<Graph> graphs

void parser(vector<string> input, ostream& os)

Graph convertInfixToPostfixAndRPN(vector<string> input);
```

```
std::string name

std::set<Vertex> vertices
```

```
std::string name

std::set<string> neighbours
```

<u>How the calculator works:</u>

When the input string is read by std::`getline()`, the first thing done is *tokenization* of the string.

The tokenization process: first identify each special keys such as operators / brackets ( ) { } < > / commas , then add spaces before and after each key in the input string, this updated input string is sent to another function which splits the string to substrings by a `delimiter = isspace` and inserts the substrings into a `vector<string>` which is the input we will be dealing with from now on.
The function `gcalc::parser()` takes vector of strings and executes commands according to the tokens inside it.
for example the command “print”/”save”/”who”/… will be identified in the first token of the `vector<string>` and the function which executes the command will be called, the parser also identifies special keys as “=” and calls `gcalc::converInfixToPostfixAndRPN()` function which defines/redefines/loads a graph.
This function uses a refined version of the Shunting-yard algorithm which converts an infix expression to a postfix one and the definition is executed by the Reverse Polish Notation (RPN).