# Software
GF2 - First Interim Report

SOFTWARE DESIGN TEAM 1
Quang-Thinh Ha - *qth20*; Edgar Dakin - *ed408*; Konstantinos Kyriakopoulos - *kk492*

## 1 Introduction and General Approach

The aim of this project is to develop a logic simulation program in C++. The project involves all five major phases of the software engineering life cycle: specification, design, implementation, testing and maintenance.
This report will include a general approach from the team, including teamwork planning and our agreed EBNF for syntax, along with analysing all possible semantic errors and errors handling.
The system is named Elea, after the Ancient Greek colony of , home to early logicians such as Parmenides and Zeno.

## 2 Teamwork Planning

The following tasks have been allocated to the members of the team, with their corresponding deadlines for the remaining allowance time of the project.

| **Week** | **Member** | **Activity** |
|---|---|---|
| *2* | Edgar (E) | Semantic analysis and error handling. |
| | Konstantinos (K) | Finalise EBNF for syntax. |
| | Quang (Q) | Start GUI design. |
| *3* | E and K | Design and implement `names, scanner, parser classes`. |
| | Q | Design and implement `gui classes` |
| *4* | E, K and Q | Integration and final testing. |
| | | Maintenance. |

## 3 Language Description

### 3.1 Syntax

The circuit description language, named Nyaya[1], is to consist of 5 high level commands, each beginning with an identifying keyword and terminating in a semi-colon, for robust LL(1) parsing.

[1] from the Sanskrit word  meaning logic

**CREATE:** Defines and names the individual devices in the circuit, specifying relevant parameters for each one. A single CREATE statement can create multiple devices, delimiting devices of the same type by commas and different types by ampersands. The declaration of each device is reminiscent of constructors in object-oriented programming languages, taking the form *type name* (*comma−delimited parameters*).

**CONNECT:** Creates a connection between an output pin and an input pin.

**DEFINE:** Defines a new type of device containing instances of previously defined types, added using CREATE and connected using CONNECT, and accepting specified parameters, which can be passed to the internal devices. Pins of the internal devices are selected as inputs and outputs of the new device.

**MONITOR:** Specifies output pins to be monitored.

**INCLUDE:** Links to a library file containing additional DEFINE commands to be parsed before parsing the rest of the current file.

The complete syntax of Nyaya in EBNF follows below.

```
nyayagrammar = { include | create | connect | define | monitor };
libgrammar = { define };
include = "INCLUDE", lib, ";";
create = "CREATE", typecreate, {"&", typecreate}, ";";
typecreate = type, init, {",", init};
init = name, "(", [ param, {",", param} ], ")";
param = int_param | ref_param;
connect = "CONNECT", in, "->", out, ";";
in = device, [".", input]; out = device, [".", output];
define = "DEFINE", new_type, "(", [param_name, {",", param_name, }, ")", "{",
inputs = "INPUT", increate, {"," , increate}; outputs = "OUTPUT", outcreate,
increate = in_name, "=", in; outcreate = out_name, "=", out;
monitor = "MONITOR", out, {"," , out};
```

### 3.2 Semantics

The following semantic constraints apply to the language and must be validated when a Nyaya file is read in.

- *lib* must point to a file which is a valid libgrammar
- *type* must correspond to the name of a primitive type or a type previously defined
- *name* must be an alphanumeric string that is not already the name of a device in the current context (ie. within a given define or outside defines)
- the number of *param* in init must equal that required by the type

- *int_param* must be an integer which satisfies the validation required by the type *ref_param* must correspond to one of the *param_names* within the current define statement
- *device* must be the name of one of the devices previously created within the current context (ie. within a given DEFINE or outside DEFINEs) input must be present unless the device has exactly one input input must be one of the input names for that device (eg. I1, I3, data, clock) output must be present unless the device has exactly one output output must be one of the output names for that device (e.g. Q, Qbar)
- *new_type* must be an alphanumeric string that is not already the name of a primitive type or a type previously defined *param_name* must be an alphanumeric string that does not correspond to a previous *param_name* in the same define
- *in_name* must be an alphanumeric string that does not correspond to a previous *in_name*
- *out_name* must be an alphanumeric string that does not correspond to a previous *out_name*

# 4 Semantic Errors and Handling