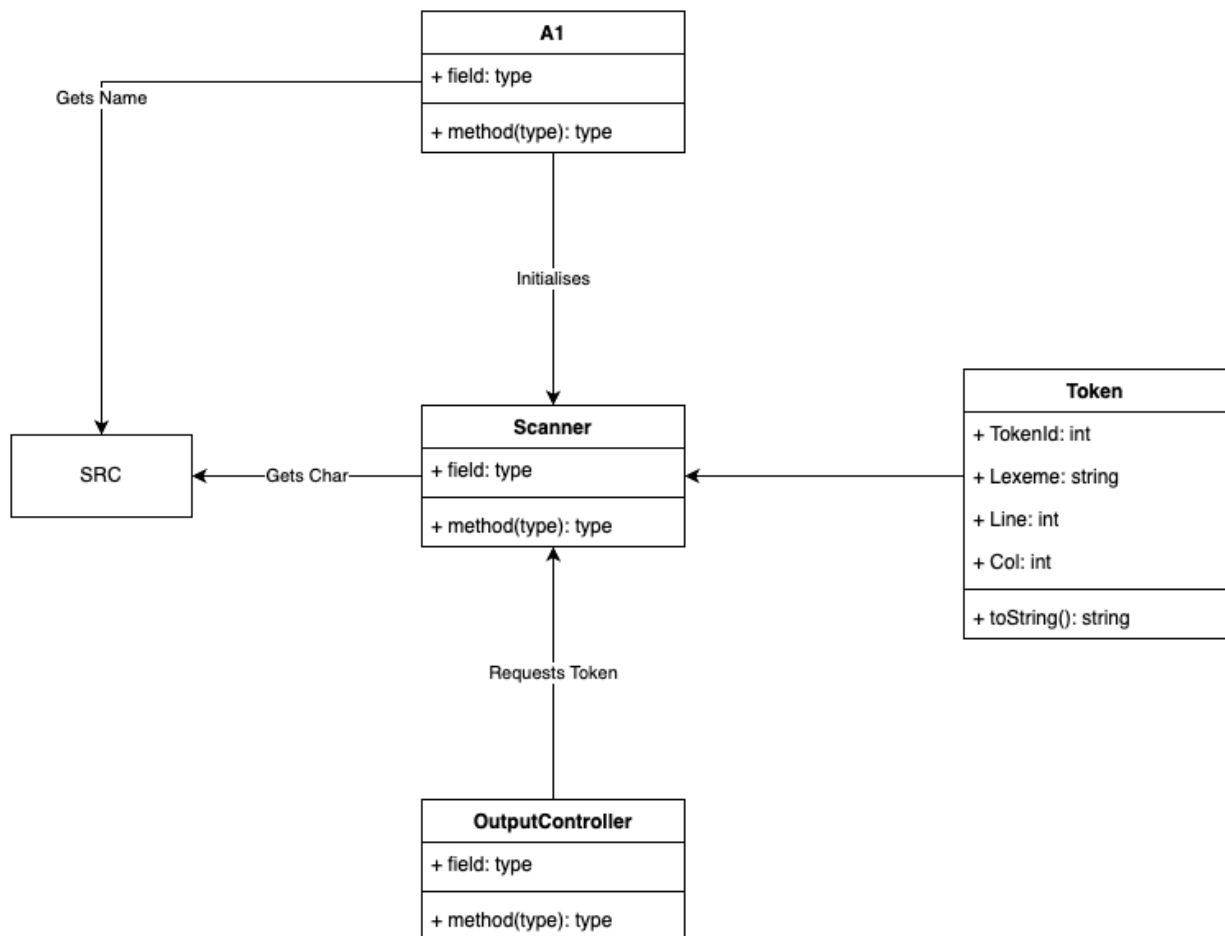# Assignment 1

## Part A - Writing Programs in CD24

- ☑ Two programs that fail lexically
- ☑ Two programs that fail syntactically (succeeds lexically)
- ☑ Two programs that fail semantically (succeeds lexically and syntactically)
- ☑ The simplest working program
- ◯ Several Random programs - lets go for 5
    - ☑ One that has a main program and simple variables
    - ☑ Simple arrays program
    - ☑ Vet program - needs to pass all different types once (esp arrays by reference)
- ◯ Grading Program
    - ☑ Define all functions
    - ◯ Need to write main logic

## Issues found

- typeid and structid are defined to match the same pattern (id) causing ambiguity
- Ambiguity in parsing booleans - is not left or right associated

## Part B - Scanner Planning

Class Diagram

```
          A1
+ field: type
+ method(type): type
```

Gets Name

Initialises

```
        Scanner
+ field: type
+ method(type): type
```

```
   SRC
```

Gets Char

```
        Token
+ TokenId: int
+ Lexeme: string
+ Line: int
+ Col: int
+ toString(): string
```

Requests Token

```
     OutputController
+ field: type
+ method(type): type
```

## Pseudocode of Main Logic

```
Init readBuffer
loop until EOF
    get nextChar
    loop until stateChange
    tokenise
    add to arrayList of token
```

## Scanning Example

Starting code - `abc = 10 + xyz; EOF`

1. First Read -> `a` -> continues to read char by char until state change -> `abc` with `=` held (state change occurred here)
2. `abc` is tokenised -> `TIDNT abc` and stored in token arraylist
3. Resumes from held char -> `=` -> next char is read until state change -> `=` with `1` held.
4. `=` is tokenised -> `TEQUL` and stored in token array list

5. Resumes from held char -> `1` continues to read char by char until state change -> `10` with `+` held.
6. `10` is tokenised -> `TILIT 10` and stored in token arraylist
7. Resumes from held char -> `+` continues to read char by char until state change -> `+` with `x` held.
8. `+` is tokenised -> `TPLUS` and stored in token arraylist
9. Resumes from held char -> `x` continues to read char by char until state change -> `xyz` with `;` held.
10. `xyz` is tokenised -> `TIDNT xyz` and stored in token arraylist
11. Resumes from held char -> `;` continues to read char by char until state change -> `;` with `EOF` held.
12. `;` is tokenised -> `TSEMI` and stored in token arraylist
13. Resumes from held char -> `EOF` and stops reading, adding the `TEOF` token to the arraylist

## Other Notes

- Will probably use a hashmap for storage of the tokens.