## Tokens

A token has the following structure:

```
struct InternalToken
{
    std::string repr;
    int cc;
    int cl;
    InternalTokenType type;
```

With cc referring to the column the token appears on, and cl referring to the line.

Tokens can have the following types:

```
enum class InternalTokenType
{
    INVALID,
    IDENTIFIER,
    CONSTANT,
    KEYWORD,
    SEPARATOR,
    OPERATOR
};
```

## Scanning algorithm

The scanner algorithm takes a program as a string and a list of tokens with their respective codes as per the language specification, and tries to parse & classify them into tokens.

The algorithm performs several passes:

```
auto tks = splitByWhitespace(program);
std::vector<InternalToken> splitTokens;
for (auto& t : tks)
{
    auto str = splitBySeparatorOperator(t);
    int n = 0;
    for (auto& s : str)
    {
        int cc = t.cc + n;
        splitTokens.emplace_back(InternalToken(s, cc, t.cl));
        n += s.length();
    }
}
classifyTokens(splitTokens);
disambiguateTokens(splitTokens);
insertTokens(splitTokens);
```

First of all, it removes all comments – any line starting with '//' will be ignored.

Then, the string is separated by whitespace elements (space/newline/tab) into pseudo-tokens that will further be analyzed and classified.

Then, since we can have different token types NOT separated by whitespace, we need to correctly identify and separate our pseudo-tokens into keywords, separators, operators, identifiers and constants.

After we've done this, we can attempt to classify each token. Any token that cannot be classified will throw an exception and terminate the scanning algorithm. A message containing the line and column of the unclassifiable token will be reported.

After classifying our tokens, we need to do one final pass through the tokens to account for some special cases, such as numeric signs that were identified as binary arithmetic operators, and full stop separators in floating-point numbers that were identified as the member access operator.

Finally, we insert our tokens into the program internal form, which is a list of pairs of form (type, code). The type part is an indicator that classifies tokens into identifiers (0), constants (1), and other types – keywords/separators/operators (-1). The code refers to the tokens code read from a file in the case of keywords/separators/operators, or the position of the token in the symbol table in the case identifiers/constants.

The symbol table uses a hash table with linear probing. The keys are represented by the string representation of the constant/identifiers, and the values are the positions of the tokens within the table.

```
void insertTokens(std::vector<InternalToken>& tks)
{
    for (auto& t : tks)
    {
        if (t.type == InternalTokenType::IDENTIFIER)
        {
            st.add(t);
            pif.emplace_back(std::pair<int, int>(0, st.get(t)));
        }
        else if (t.type == InternalTokenType::CONSTANT)
        {
            st.add(t);
            pif.emplace_back(std::pair<int, int>(1, st.get(t)));
        }
        else
        {
            pif.emplace_back(std::pair<int, int>(-1, tokens[t.repr]));
        }
    }
}
```

## Class diagram