Stage 0 - Command-Line Argument Parser (CLAP)

This is the first stage of the ByteFrost Assembler pipeline. The **Command-Line Argument Parser (CLAP)** takes as input the command-line arguments received from running the ByteFrost Assembler on a command-line shell and writes to a `CommandLineArguments` object which is stored by the `Assembler` (the `Assembler` passes a reference to this object to the CLAP).

Definitions

`CLToken`s

The CLAP recognizes the following Command-Line Argument Token (`CLToken`) types, which are the values of the `CLTokenType` enum:

```
enum CLTokenType = {FLAG, NUMBER, FILE, TEXT, INVALID}
```

`FLAG`

A command-line argument string `w` is mapped to `CLTokenType.FLAG` if

1. The length of `w` is `>= 2`.
2. `w[0] = '-'`.
3. `w[1]...w[len(w) - 1]` $\in TEXT$

`NUMBER`

A command-line argument string `w` is mapped to `CLTokenType.NUMBER` if

1. `w` $\in NUMBER$.

`FILE`

A command-line argument string `w` is mapped to `CLTokenType.FILE` if

1. `w` $\in FILE$.

`TEXT`

A command-line argument string `w` is mapped to `CLTokenType.TEXT` if

1. `w` $\in TEXT$.

`INVALID`

A command-line argument string `w` is mapped to `CLTokenType.INVALID` if

1. `w` cannot be mapped to any `CLTokenType` that is not `CLTokenType.INVALID`.

`CLFlag`

The `CLFlag` struct is defined as follows:

```
struct CLFlag {
    string flagName;
    vector<CLTokenType> expected_pattern;
    bool isSet = false;
};
```

The following are flags (`CLFlag`s) that are recognized by the CLAP:

`-b` Binary Flag

- Use: When set, this flag tells the ByteFrost Assembler to produce a `.bin` file.
- Flag name: `b`.
- Number of arguments: 0.
- Expected pattern: `{}`.

`-o` Output File Name Flag

- Use: When set, this flag tells the ByteFrost Assembler to produce an output file with the name specified by the following argument (`CLToken`).
- Flag name: `o`.
- Number of arguments: 1.
- Expected pattern: `{FILE}`.

Command-Line Argument Parsing

The CLAP is given the command-line arguments in the form of the tuple `(int argc, char ** argv)`. This is similar to the `vector<string> tokenStrings` sub-input in the Parser (Stage 1) parsing of `.asm` file lines.

The CLAP will first convert the command-line argument strings into a list of `CLToken`s (i.e., a `std::vector<CLToken>`).

Each command-line argument string `w` is mapped to a `CLTokenType` and then a corresponding `CLToken` object is created with the contents of `w` and the specified `CLTokenType` which is then added to a `std::vector<CLToken>`.

If an `CLTokenType.INVALID` token is encountered, the CLAP throws an error.

Once the `std::vector<CLToken>` list is generated, the CLAP will iterate over this list and treat each token as a flag, a flag argument, or as the required input argument which is the input file name.

The first non-flag and non-flag argument argument is understood to be the input_file_name.

When a flag argument is encountered, the CLAP will check whether the flag is a CLAP-recognized flag by checking whether the argument string exists as a key of the flag name to `CLFlag` hashmap `std::unordered_map<string, CLFlag>` in the `CommandLineArguments` object passed in as a reference

from the `Assembler` when the CLAP constructor was called. If the flag is not recognized (key doesn't exist), the CLAP will throw an error. If the flag does exist, then the corresponding `CLFlag` object will contain a `vector<CLTokenType> expected_pattern` which details the number and types of the following expected arguments. If there aren't enough `CLToken`s after this one or if their types won't match, the CLAP will throw an error. The CLAP will also set the `CLFlag` argument value fields to the values in the following `CLToken`s (e.g., if we have `-size 3`, then the `CLSizeFlag` object (which derives from `CLFlag`) will have its `size` field set to the integer `3`).

Once the CLAP finishes iterating through the `CLToken` vector and no errors have occurred, the CLAP will return and so stage 0 of the ByteFrost Assembler completes its execution.