Stage 2 - Preprocessor

This is the third stage of the ByteFrost Assembler pipeline.

The **Preprocessor** takes as input the `std::vector<Line *>` vector generated by the `Parser`. It then does two things:

1. Handle every `Line` that is a `Directive` statement by updating its or the `Assembler`'s state.
2. Replace every `TokenType::TEXT` token in `InstructionLine`s with a `TokenType::IMMEDIATE` token (as these were assumed by the `Parser` to be preprocessor constants defined in `.define` `Directive` `Line`s; the preprocessor therefore attempts to replace these with immediate tokens that contain the constant's defined value.)

Definitions

The `Preprocessor` Class

The `Preprocessor` is defined as the following class:

```
class Preprocessor {
public:
    Preprocessor();
    void run(std::vector<Line *> & lines, CommandLineArguments & args);
private:
    void handleDirective(DirectiveLine * line);
}
```

`DirectiveType`

Each directive that the Preprocessor recognizes has its own `DirectiveType` enum value:

```
enum class DirectiveType {define, start, ram, rom, ...};
```

`Directive`s

Each `Preprocessor` directive is encoded as a `Directive` struct in the following way:

```
struct Directive {
    string name;
    DirectiveType type;
    vector<TokenType> expected_param_types;
}
```

Running the `Preprocessor`

The `Preprocessor.run()` method implements the following loop:

```
for (Line * line : lines) {
    //  Iterate through each line in the vector<Line *> lines generated by the
    //  Parser. The current line is line.
    //  Case 1: line is a DirectiveLine.
    //      Handle the directive in the DirectiveLine.
    //  Case 2: line is an InstructionLine.
    //      Iterate through the line's Token vector.
    //          If a Token in the Token vector has type TokenType::TEXT or
    //          TokenType::BYTE_CONSTANT then attempt to find a Preprocessor
    //          constant (defined with a .define Preprocessor Directive) with
    //          that name.
    //              If such a constant was found, replace the token's value with
    //              an immediate string that contains the constant's value and
    //              replace the token's type with TokenType::IMMEDIATE.
    //              If not, throw an error (undefined constant).
    switch (line.type) {
        case LineType::DirectiveLine: {
            DirectiveLine * directive_line = (DirectiveLine *)line;
            handleDirective(directive_line);
            break;
        }
    }
    if (line.type == LineType::DirectiveLine) {
        DirectiveLine * directive_line = (DirectiveLine * )line;
        handleDirective(directive_line);
    }
    else if (line.type == )
}
```