PBL

- Parses 8085 assembly code using ANTLR4.
- Checks for **syntactic** and **semantic** errors.
- Outputs a list of parsed instructions.
- Warns about things like unknown opcodes or missing labels.
- Generates a **parse tree graph** for visualization.

```
class CollectingErrorListener(ErrorListener):
    def __init__(self):
        self.errors = []

    def syntaxError(self, recognizer, offendingSymbol, line, column, msg
        self.errors.append(f"line {line}:{column} {msg}")
```

Extends the **ErrorListener** class provided by the Antlr. Added own error field in the class , and override the **syntaxError** function to append the errors line , column , message into error array.

```
class InstructionVisitor(Assembly8085Visitor):
    ....................
    self.instructions = []
    self.errors = []
    ....................
```

```
    def visitInstruction(self, ctx):
        opcode = ctx.opcode().getText().upper() if ctx.opcode() else "<mi
        operands = [op.getText() for op in ctx.operand()] if ctx.operand() 

        if opcode not in VALID_OPCODES:
            self.errors.append(f"Line {ctx.start.line}: Unknown or missing o

        return {"opcode": opcode, "operands": operands}
```

Extends the **Assembly8085Visitor** class and declares function **visitInstruction** to extract the opcode and operand from each line, for invalid opcode , append the errors else return the operand and operator.

VisitProgram visits the program and returns the instructions in each line.

```
class SemanticAnalyzer(Assembly8085Visitor):
    def __init__(self):
        self.labels = set()
        self.used_labels = []
        self.errors = []
```

This have own visit Instruction function that visit the parse tree and check the semantic meaning of it . Whether our op code have valid number of operands or not, whether labels have been re-used.

MAIN FUNCTION

1. Pass the input file name as argument
2. Open the input file and re-write content with removed \r
3. Open the input stream , feed the input stream to the lexer.
4. Lexer returns token stream , feed that stream to parser.
5. Add custom error listener into the class
6. Start parsing from program rule defined in the grammar rules . Generate a parse tree
7. If there are errors visit the tree and collect the invalid opcodes
8. Again visit the tree and collect the semantics errors this time