USER
MANUAL

Two Pass Assembler Documentation

A Comprehensive Guide to Two-Pass Assemble Functionality

Bhadra J

Roll no:68

1. Introduction

The Two Pass Assembler is a graphical user interface (GUI) application designed to convert assembly language into object code using the two-pass assembly process. This application helps users input source code, generate intermediate text, symbol table, and opcode table (Optab), and output the final object code.

2. User Requirements

Input Source Code Area

Generate Intermediate Text

View Symbol Table

Opcode Table (Optab

Object Code Generation

Clear Button

Error Handling

3. Developer Requirements

Programming Language: The application is built using Java, specifically utilizing the Java Swing library to create the GUI elements.

Development Tools: need a Java Development Kit (JDK), version 8 or later, to compile and run the program.

Functional Requirements : GUI Design, Two-Pass Assembly Process, File Handling,Error Handling & Validation.

Dependencies & Libraries : The user interface is developed using the **Java Swing** framework. The GUI uses the **Lucida Console** font for displaying code to maintain clarity.

4. Features

- Input Source Code Area: Allows users to input the assembly code manually.
- Intermediate Text Generation: Displays the intermediate output after Pass 1.
- Symbol Table (Symtab): Shows a table of symbols with their addresses generated in Pass 1.
- Opcode Table (Optab): Displays the opcode table used in generating object code.
- Object Code Output Area: Displays the final machine code after the second pass.
- Clear Functionality: Resets all fields for a new assembly operation.

5. User Interface Description

5.1 Layout Overview

The interface is divided into two main sections:

- Input Section (Left): Where users enter their assembly code.
- Output Section (Right): Where the intermediate text, symbol table, opcode table, and final object code are displayed.

5.2 Components

- Input Source Code Field: A large text area for users to input the assembly code manually.
    - This area supports multiline inputs and can be cleared for new entries using the ‘Clear’ button.

- Buttons:
    - Intermediate Text Button: Displays the intermediate code output from Pass 1.
    - Symtab Button: Displays the symbol table after Pass 1.
    - Optab Button: Displays the opcode table.
    - Assemble Button: Starts the assembly process and generates the final object code.
    - Clear Button: Clears both the input field and the output fields.
- Output Section:
    - Object Code Field: A large text area where the final object code is displayed after the assembly process.

6. Functionality

- Two-Pass Process:
    - Pass 1: Reads the source code, generating the intermediate file and symbol table.
    - Pass 2: Using the symbol table and opcode table (Optab), it generates the final object code.
- Error Detection: The assembler also includes error detection, providing users with messages when encountering undefined symbols or syntax errors.

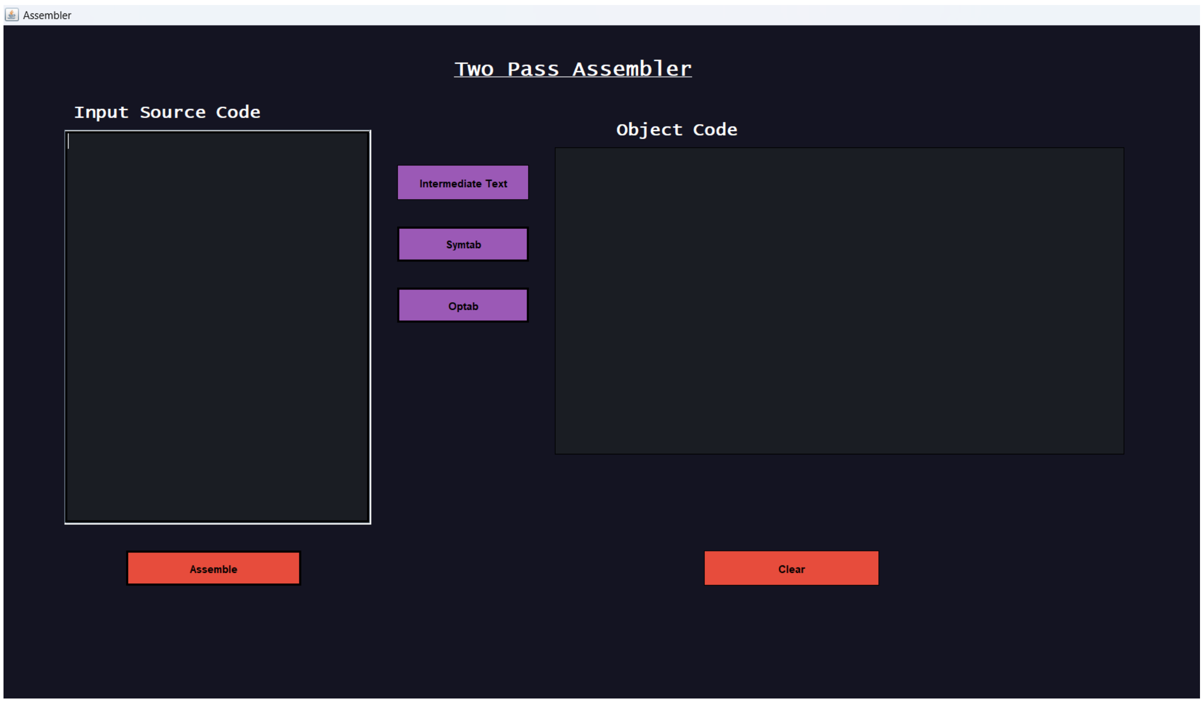
7. Steps for Use

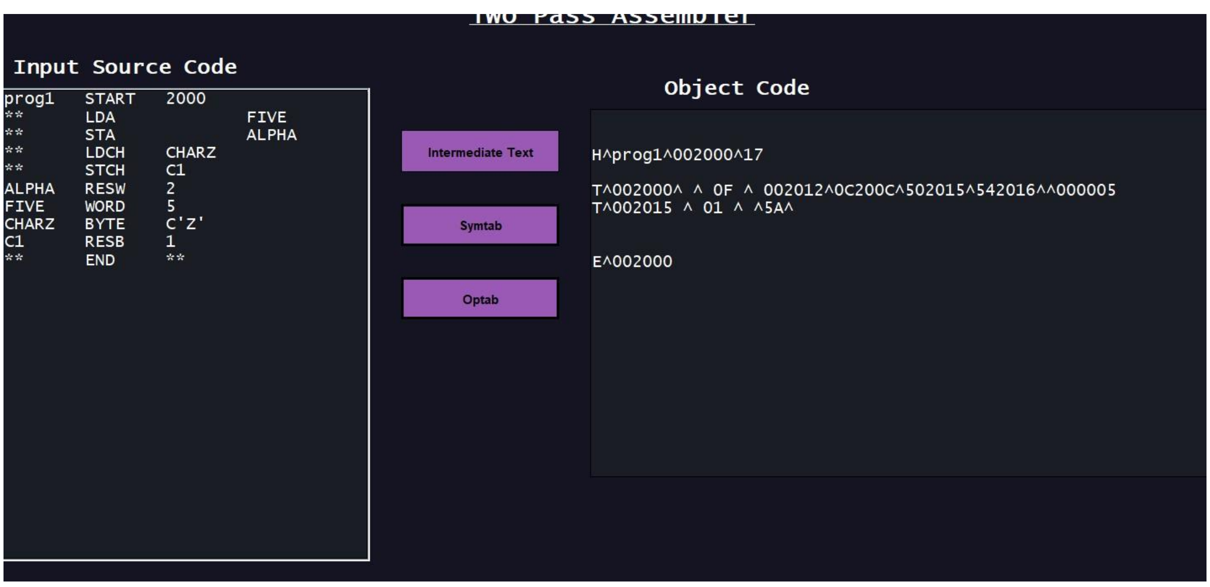
1. Input Assembly Code: Type or paste the assembly code into the input area.
2. Generate Intermediate Text: Click on the 'Intermediate Text' button to generate and view the intermediate output.
3. View Symbol Table: Click on 'Symtab' to view the generated symbol table.
4. View Opcode Table: Click on 'Optab' to view the opcode table.
5. Assemble: Click the 'Assemble' button to generate the final object code.

6. Clear: To reset all fields, click the 'Clear' button.

8. Example Usage

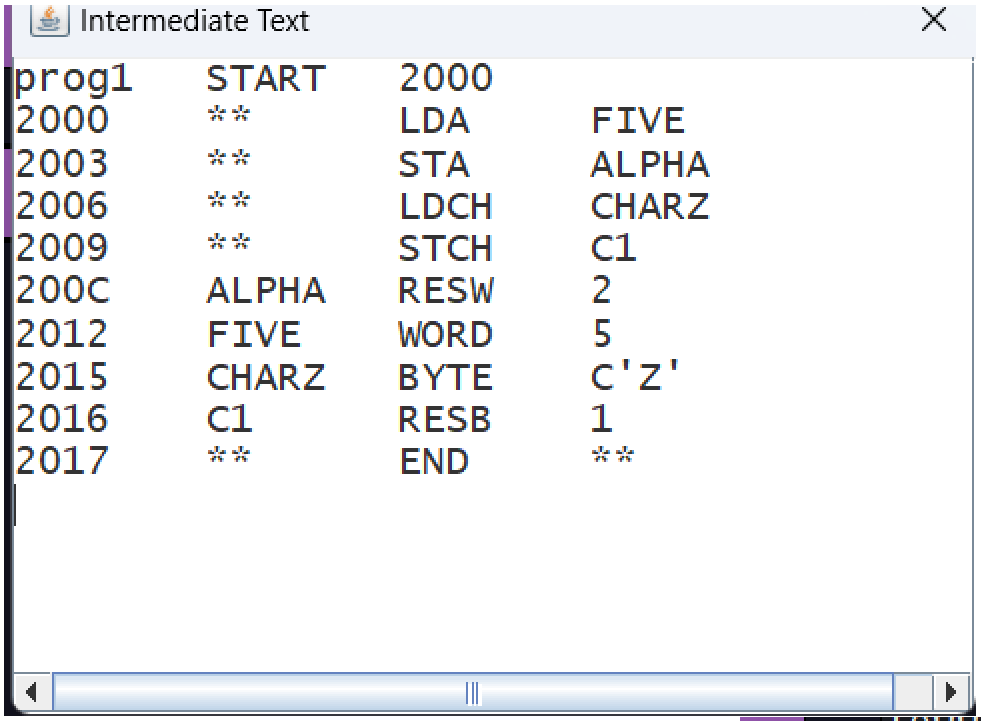
Step 1: Enter the source code in the left-hand box.
Step 2: Click the 'Assemble' button to process the code through both Pass 1 and Pass 2.
Step 3: Once the assembly is complete, you can view the generated Intermediate Text, Symbol Table (Symtab), and Opcode Table (Optab).
Step 4: The final object code will be displayed in the output area.



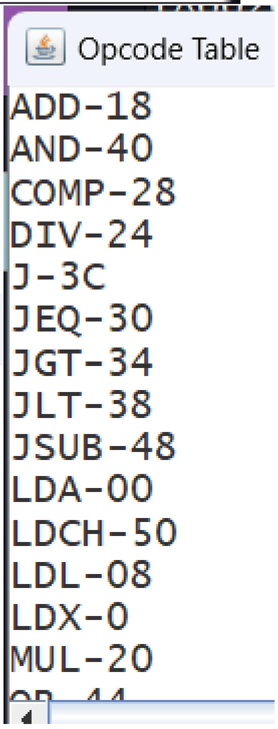


```
prog1   START   2000
**      LDA             FIVE
**      STA             ALPHA
**      LDCH    CHARZ
**      STCH    C1
ALPHA   RESW    2
FIVE    WORD    5
CHARZ   BYTE    C'Z'
C1      RESB    1
**      END     **
```

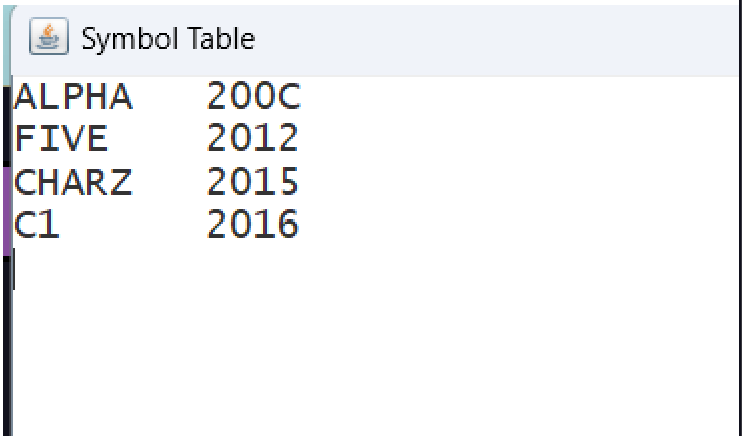
```
H^prog1^002000^17

T^002000^ ^ 0F ^ 002012^0C200C^502015^542016^^000005
T^002015 ^ 01 ^ ^5A^


E^002000
```



```
prog1     START     2000
2000      **        LDA       FIVE
2003      **        STA       ALPHA
2006      **        LDCH      CHARZ
2009      **        STCH      C1
200C      ALPHA     RESW      2
2012      FIVE      WORD      5
2015      CHARZ     BYTE      C'Z'
2016      C1        RESB      1
2017      **        END       **
```



```
ADD-18
AND-40
COMP-28
DIV-24
J-3C
JEQ-30
JGT-34
JLT-38
JSUB-48
LDA-00
LDCH-50
LDL-08
LDX-0
MUL-20
```



```
ALPHA     200C
FIVE      2012
CHARZ     2015
C1        2016
```

9. Error Handling

Errors detected during the assembly process will be displayed in the output section, helping users to debug their assembly code.

10. Dependencies and Requirements

- Java Swing: The GUI is built using Java Swing components.
- Font: The interface uses 'Lucida Console' for a clear and consistent display.
- Execution Environment: Requires Java Development Kit (JDK) to run.

GitHub repository link : bhadraaa/TwopassGui (github.com)

Eg code

```
COPY  START 1000
**      LDA    ALPHA
**      ADD    ONE
**      SUB    TWO
**      STA    BETA
ALPHA          BYTE C'CZXE'
ONE    RESW 2
TWO   WORD          5
BETA RESW 1
**      END    -
```

```
prog1 START 2000
**      LDA             FIVE
**      STA             ALPHA
**      LDCH  CHARZ
**      STCH  C1
ALPHA          RESW           2
FIVE   WORD          5
CHARZ          BYTE C'Z'
C1      RESB  1
**      END    **
```