SYSTEM SOFTWARE
20114086
SHAURYA RANA

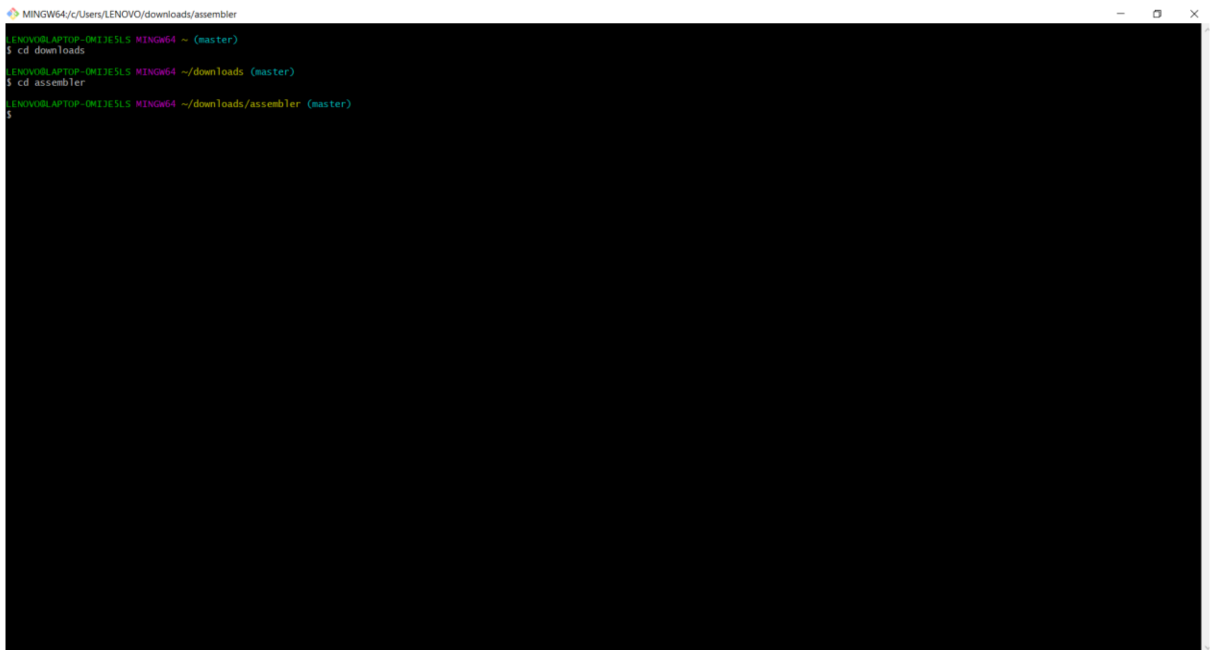
Design of 2 pass SIC/XE assembler

Assembler overview:

The SIC/XE assembler is a two-pass assembler in which an input file is served to the assembler and the assembler generates an object code, table file, intermediate file and an error file. The input file is written in assembly language. The working of the assembler takes in two phases which is pass1 and pass2. In pass1 the assembler generates a symbol table, an intermediate file for pass 2. In pass 2 an object program along with an error file is generated and a listing file for containing the input assembly code and address, block number, object code of each instruction. The assembler takes into account all the SIC/XE instructions all machine independent assembler features like literals, program blocks, expressions and symbol defining statements.

Steps to compile and execute the assembler for a test program:

In order to compile and execute we need to go to the Assembler directory



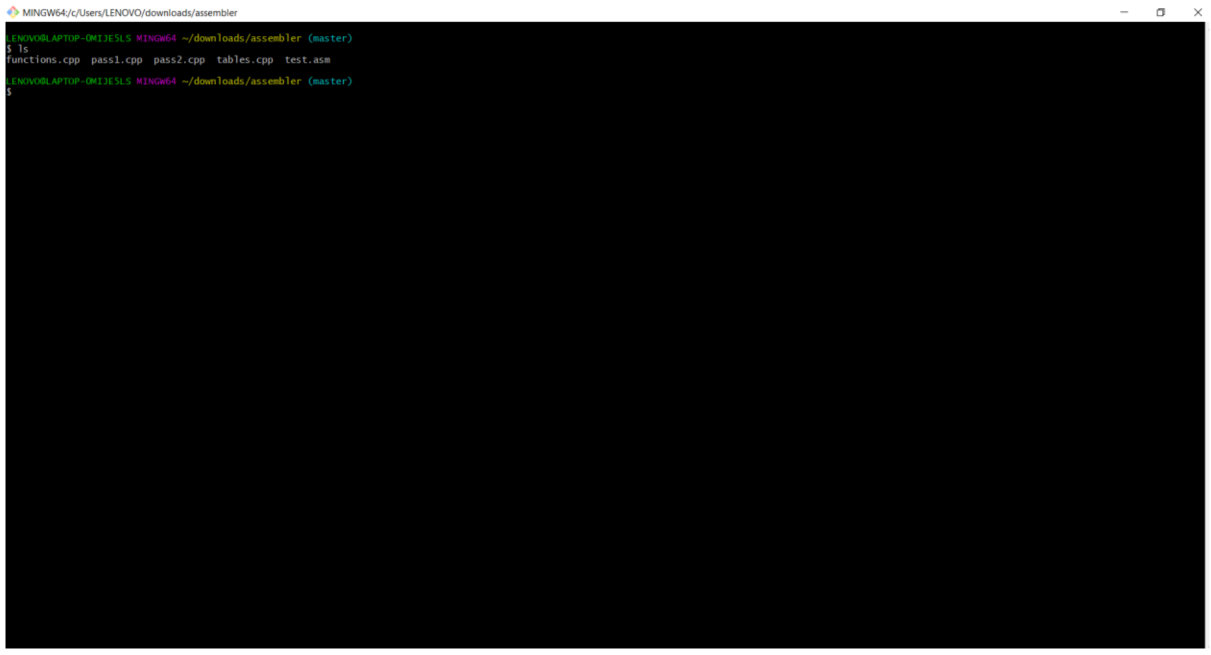
```
MINGW64:/c/Users/LENOVO/downloads/assembler

LENOVO@LAPTOP-OMIJE5LS MINGW64 ~ (master)
$ cd downloads

LENOVO@LAPTOP-OMIJE5LS MINGW64 ~/downloads (master)
$ cd assembler

LENOVO@LAPTOP-OMIJE5LS MINGW64 ~/downloads/assembler (master)
$
```

In the present directory initially we have 4 files for assembler including pass1.cpp, pass2.cpp, functions.cpp and tables.cpp. Also we have the required input files to be given to assembler (in this case test.asm).

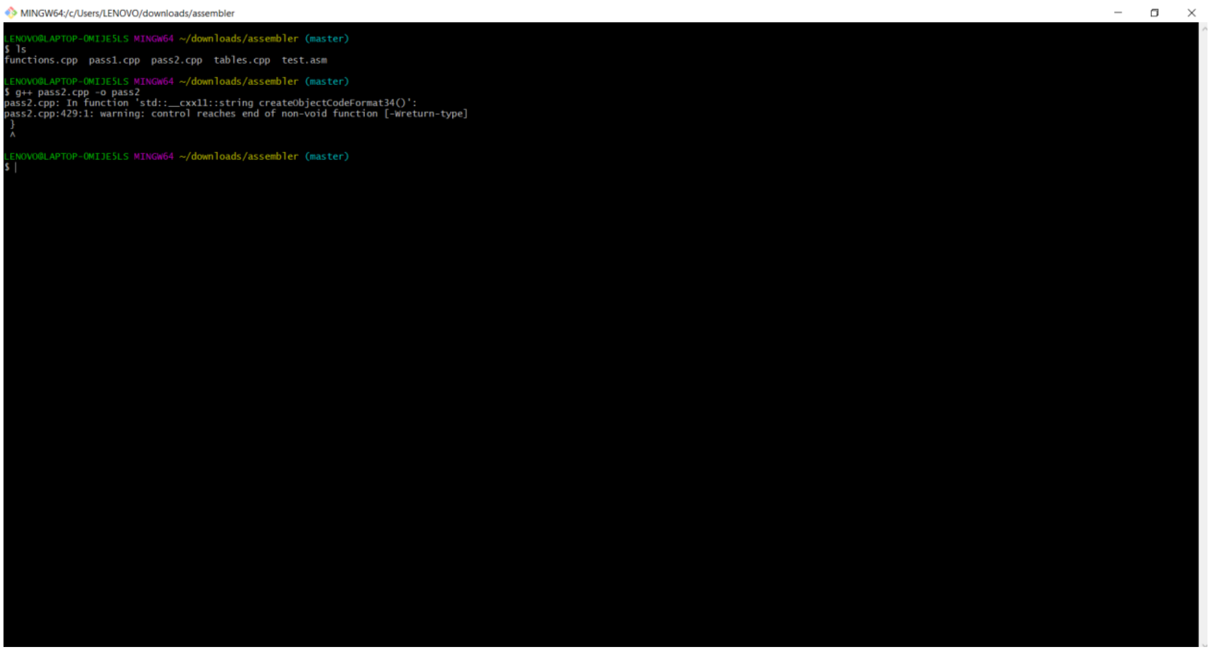


```
MINGW64:/c/Users/LENOVO/downloads/assembler

LENOVO@LAPTOP-OMIJE5LS MINGW64 ~/downloads/assembler (master)
$ ls
functions.cpp  pass1.cpp  pass2.cpp  tables.cpp  test.asm

LENOVO@LAPTOP-OMIJE5LS MINGW64 ~/downloads/assembler (master)
$
```

The assembler is designed in such a way that pass2 file makes call to all the other required files in assembler like pass1 , functions and tables. So, first we need to compile pass2.cpp

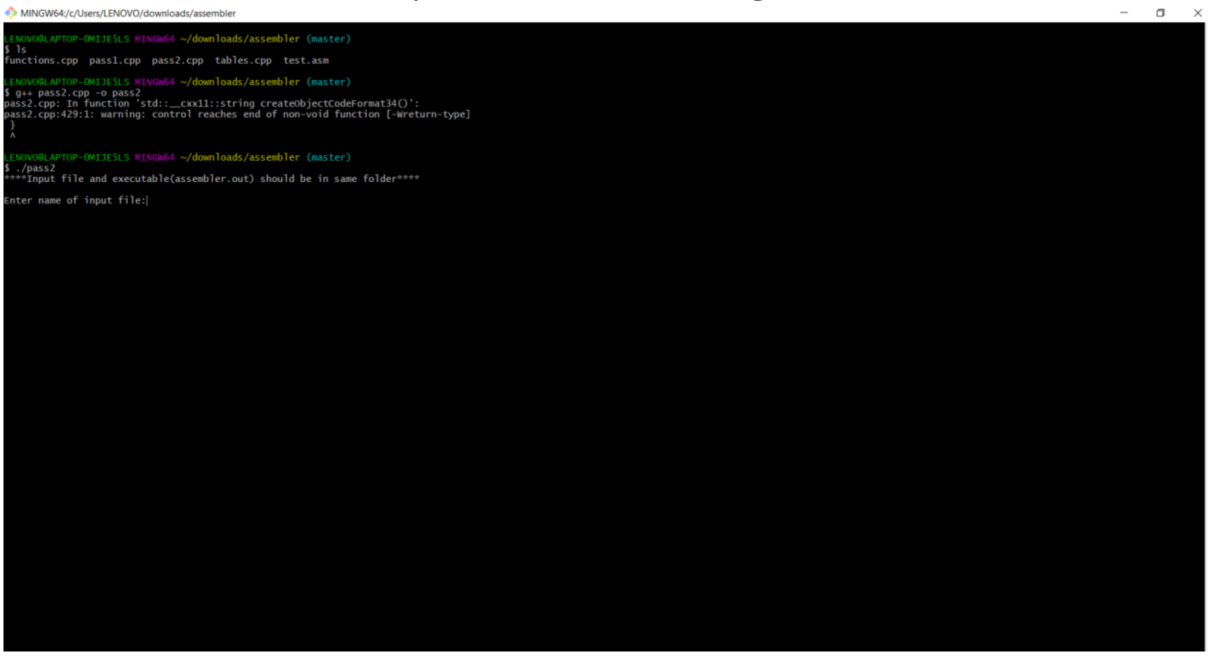


```
LENOVO@LAPTOP-OMIJE5LS MINGW64 ~/downloads/assembler (master)
$ ls
functions.cpp  pass1.cpp  pass2.cpp  tables.cpp  test.asm

LENOVO@LAPTOP-OMIJE5LS MINGW64 ~/downloads/assembler (master)
$ g++ pass2.cpp -o pass2
pass2.cpp: In function 'std::__cxx11::string createObjectCodeFormat34()':
pass2.cpp:429:1: warning: control reaches end of non-void function [-Wreturn-type]
 }
 ^

LENOVO@LAPTOP-OMIJE5LS MINGW64 ~/downloads/assembler (master)
$ |
```

After compiling pass2.cpp we need to execute it. When we execute the file it asks for the input file we want to give.



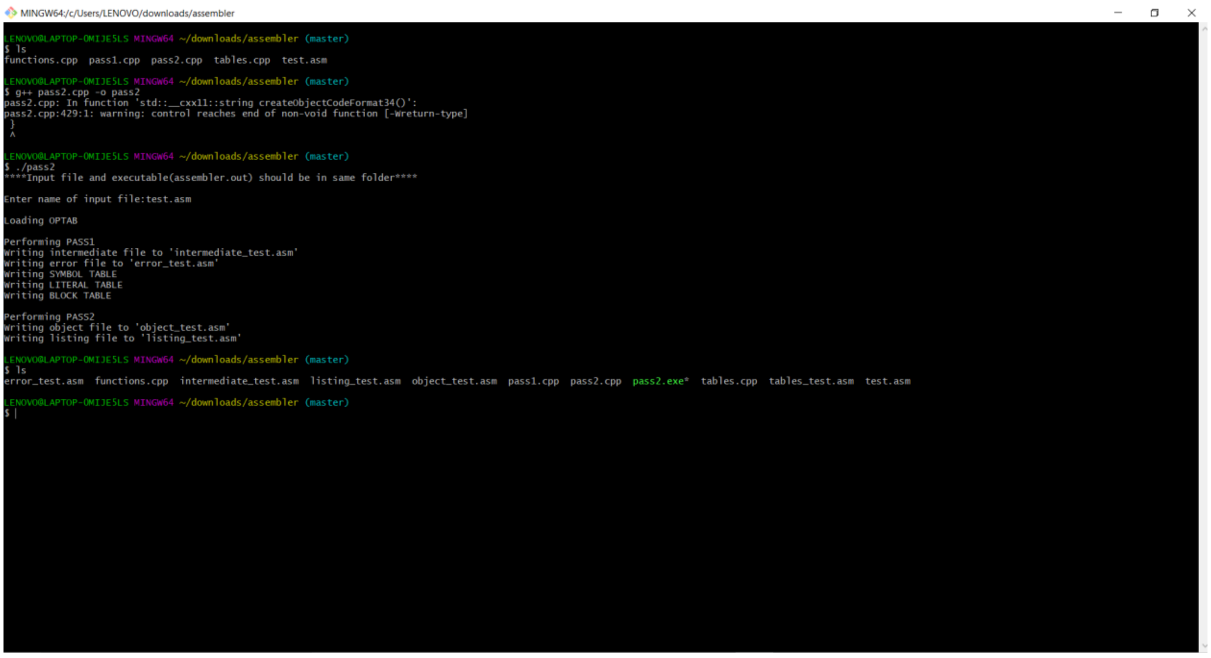
```
LENOVO@LAPTOP-OMIJE5LS MINGW64 ~/downloads/assembler (master)
$ ls
functions.cpp  pass1.cpp  pass2.cpp  tables.cpp  test.asm

LENOVO@LAPTOP-OMIJE5LS MINGW64 ~/downloads/assembler (master)
$ g++ pass2.cpp -o pass2
pass2.cpp: In function 'std::__cxx11::string createObjectCodeFormat34()':
pass2.cpp:429:1: warning: control reaches end of non-void function [-Wreturn-type]
 }
 ^

LENOVO@LAPTOP-OMIJE5LS MINGW64 ~/downloads/assembler (master)
$ ./pass2
****Input file and executable(assembler.out) should be in same folder****

Enter name of input file:|
```

In this case we give test.asm as input and run the assembler. The assembler finally gives literal, symbol, block tables as output.



```
MINGW64:/c/Users/LENOVO/downloads/assembler
LENOVO@LAPTOP-OMIJE5LS MINGW64 ~/downloads/assembler (master)
$ ls
functions.cpp  pass1.cpp  pass2.cpp  tables.cpp  test.asm

LENOVO@LAPTOP-OMIJE5LS MINGW64 ~/downloads/assembler (master)
$ g++ pass2.cpp -o pass2
pass2.cpp: In function 'std::__cxx11::string createObjectCodeFormat34()':
pass2.cpp:429:1: warning: control reaches end of non-void function [-Wreturn-type]
 }
 ^

LENOVO@LAPTOP-OMIJE5LS MINGW64 ~/downloads/assembler (master)
$ ./pass2
****Input file and executable(assembler.out) should be in same folder****

Enter name of input file:test.asm

Loading OPTAB

Performing PASS1
Writing intermediate file to 'intermediate_test.asm'
Writing error file to 'error_test.asm'
Writing SYMBOL TABLE
Writing LITERAL TABLE
Writing BLOCK TABLE

Performing PASS2
Writing object file to 'object_test.asm'
Writing listing file to 'listing_test.asm'

LENOVO@LAPTOP-OMIJE5LS MINGW64 ~/downloads/assembler (master)
$ ls
error_test.asm  functions.cpp  intermediate_test.asm  listing_test.asm  object_test.asm  pass1.cpp  pass2.cpp  pass2.exe*  tables.cpp  tables_test.asm  test.asm

LENOVO@LAPTOP-OMIJE5LS MINGW64 ~/downloads/assembler (master)
$ |
```

Clearly, we can see that after running test.asm file we got intermediate_test.asm, listing_test.asm, object_test.asm, error_test.asm, tables_test.asm

Data Structure used in implementation:

Map: Maps are associative containers that store elements in a mapped fashion. This data structure is used in the implementation to define different tables. The structure of the tables is defined using 'struct' and later using map we form the tables like SYMBOL, LITTERAL, BLOCKS etc.

Architecture of the Assembler:

Pass1: Intermediate and error file are written using input file (source file). If the intermediate or source file does not open we write the corresponding error to the error file, if the error file does not open we print it to the console. The global variables are defined in the beginning of pass1. The assembler then reads the first line of the input and checks for the line if it is comment line or not. If it is a comment line we write it to the intermediate file and update the line number. The assembler continues this process until it encounters a line other than comment. Once when it happens it checks for the opcode, if it is start we update the line number, LOCCTR and start address if not found, we initialize start address and LOCCTR as 0. Next, we use a single while loop which terminates when we encounter opcode as ‘END’. In the loop we check if line is a comment. If we find a comment we print it to the intermediate file and update line number, and take in the next input line. If it is not a comment we check if there is a label in the line, if found we check it in the SYMTAB, if found we print error saying ‘Duplicate symbol’ in the error file or else we assign the name, address and other required values to the symbol and store it in SYMTAB. The next step is to check if the opcode is present in the OPTAB, if it is there we find it’s format and then accordingly increment the LOCCTR. If it is not found in the OPTAB we check for other opcodes like ‘RESW’, ‘BYTE’, ‘RESBYTE’, ‘WORD’, ‘LTORG’, ‘ORG’, ‘BASE’, ‘USE’, ‘EQU’. Accordingly we insert the symbols in the map defined in the tables file. For instance whenever we encounter ‘USE’ other than the case of default we insert a new BLOCK entry in the BLOCK map, for ‘ORG’ we point out the LOCCTR to the operand value given, for ‘LTORG’ we call handle_LTORG() function which is defined in pass1.cpp, for EQU we check for the operand is an expression and then check if it is a valid expression, if valid we enter symbol in the SYMTAB. If after all this still the opcode does not match then we print an error message in the error file. After the while loop ends we store the program length and then print the LITTAB and the SYMTAB. Next we head on to the pass2 with the intermediate file.

<u>Description about the functions used in pass1:</u>
<u>evaluateExp</u>ression(): This function utilises pass by reference. The while loop is used to fetch the symbols from the expression. If the symbol which are fetched are not present in SYMTAB we put the error message in the error file. In order to have a check on the nature of expression i.e. if it is absolute or relative , a variable named pairCount is used as we know that in absolute expression the relative

terms appear in pairs. If the pairCount gives some unexpected value, an error message is printed.

Functions:
This file consists of all the useful functions which are utilised in pass 1 and pass 2 of the assembler. Below is description about some of the functions which are utilised by the assembler.
checkforComment(): This function is used to check if the current line is a comment line or not. It takes string as input returns the Boolean value of true or false depending on the result found.

fetchFlagFormat(): This function returns the flag bit if present in the input string or else it returns null string.

getString(): This function takes character as input and returns a string.

checkforSpace(): This function is used to check if there is space or not. It’s return type is Boolean.

stringHextoInt(): This function converts the hexadecimal string to integer and the return type is integer.

writeToFile(): This function takes a file and a string as input. The string is written on the file .

readFirstNonWhiteSpace(): This function is used to read in the current line the first non white space character. This function is a pass by reference so it also updates the index of string where non white space character is encountered.

Tables:

In this file the description is made about all the tables which are generated by the assembler. The structure of all the tables is defined in this file and using map the tables are defined. We have SYMBOL, OPCODE, REG, LIT, BLOCK table. In SYMBOL TABLE we place all the symbols of the input file, In OPCODE TABLE we have the description of all the SIC/XE opcodes, In the LIT TABLE we define all the literals and the BLOCK TABLE is used to list all the blocks of the input file.

Pass2: The intermediate file which was generated by the pass1 is read by the pass 2 of the assembler and a listing file and object program is generated. If the intermediate file or object file is unable to open we print error message in error file. If everything opens successfully then we read the first line of the intermediate file. We check for the comment lines if they are found we write them to intermediate file else we check for the opcode. If we get START we initialise our start address as the LOCCTR and write the line into the listing file.First we write the header record .Next we read from the intermediate file until we encounter the opcode as END. Using the textrecord() function we write on the object program and update the listing file. The object code is written on the basis of the types of formats used in instruction. For format 3 and 4 we use createObjectCodeFormat34() function. After completing all the text records we write the end record.

Description about the functions used in pass2:
readIntermediateFile(): This function will take line number, LOCCTR, opcode, operand, label and input output files. This function checks for the various situations in which the opcode can be and then taking into consideration the operand and the half bytes calculates the object code for the instruction. It also generates the modification record if required.

Description of Assembler working:
It is important for all input test files to have START and END labels as the assembler is designed in such a way that it starts with START

label and ends with END label. Pass1 of assembler operates in such a way that it recognises first the comment lines and print them directly to intermediate file. The while loop iterates in such a way that until it encounters opcode as ‘END’ it keeps on iterating in the lines of the input file. For various opcodes it searches for the flag format if it is format 3 or format 4. It checks for the assembler keywords and accordingly operates. All this is written to the intermediate file and if any error is encountered it is written to the error file. For pass2 the intermediate file is taken as input and it again operates on while loop until it gets ‘END’ as opcode. The pass2 takes help from the Symbol table in order to assign the address and generate the object code. Using combination of if else statements we check for the different opcodes that are possible.

Object Program for the programs:

a. Object Program for question 3 of section 2.2 (The input file is present in the folder with the name test.asm). After completing the execution the corresponding output files will be(error file-error_test.asm, intermediate file-intermediate_test.asm, listing file- lisitng_test.asm, object program- object_test.asm, tables file- tables_test.asm)

```
H^SUM   ^000000^002F03
T^000000^1D^0500000100006910179 01BA0131BC0002F
200A3B2FF40F102F004F0000
M^000007^05
M^000017^05
E^000000
```

b. Object Program for a test program(The input file is present in the folder with name program.asm). After completing the execution the corresponding output files will be(error file-error_program.asm, intermediate file-intermediate_program.asm, listing file- lisitng_program.asm, object program- object_program.asm, tables file-tables_program.asm)

```
H^COPY  ^000000^001071
T^000000^1E^1720634B2021032060290000332006 4B20
3B3F2FEE0320550F2056010003
T^00001E^09^0F20484B20293E203F
T^000027^1D^B410B400B44075101000E32038332FFADB
2032A00433200857A02FB850
T^000044^09^3B2FEA13201F4F0000
T^00006C^01^F1
T^00004D^19^B410772017E3201B332FFA53A016DF2012
B8503B2FEF4F0000
T^00006D^04^454F4605
E^000000
```

(The error files for both are empty which indicates no error).

c. To check the assembler for an input program with errors(program1.asm). The program in part (b) is modified in which the opcode WD is changed to W and RETADR is twice defined also, LENGTH is not defined but is used in the program. The error file is with the name error_program1.asm After completing the execution the corresponding output files will be(error file-error_program1.asm, intermediate file-intermediate_program1.asm, listing file- lisitng_program1.asm, object program- object_program1.asm, tables file-tables_program1.asm)

```
PASS1 errors are:
Duplicate symbol for 'RETADR'. Previously defined at 00000
Invalid OPCODE. Found W
```

PASS2 error are:
Symbol doesn't exists. Found LENGTH

CONCLUSION:

The two pass assembler successfully detects errors in case of an incorrect input file and writes it in the error file. In case of correct files there is nothing written in the error file and all the other files like object, listing, intermediate files are successfully generated.

The folder contains
4 cpp files:
Pass1.cpp
Pass2.cpp
Functions.cpp
Tables.cpp

1Report File
3 Input Files
test.asm (Sample Input File)
program.asm(Correct Input File)
program1.asm(Incorrect Input File)

1 pass2.exe executable file

3 folders each containing output files corresponding to 3 input files.
Each folder contain files: error_filename.asm, intermediate_filename.asm, listing_filename.asm, object_filename.asm, tables_filename.asm
These 3 folders contain output files corresponding to input programs when run for verification of assembler.

In order to run the input files again we need to follow steps to compile and execute.