Name: Ammaar Ahmad
Roll No: 1801CS08
Course: CS321/CS322

The assembler is written in language C++
Running the assembler in 2 steps for any test file
g++ asm.cpp -o asm
./asm filename.asm

On running this file. 3 files are generated by the assembler if there are no errors in the code

filename.l - Listing file showing instruction memory dump along with instruction
filename.log - error file listing out all errors in the assembly code.
filename.o - Binary file generated only if there are no errors

Apart from these files there is **opcode.txt** file which asm.cpp uses to read opcode and store in a map. This file contains opcodes in **HEX** format. There is another file **instructions.txt** which give details of type of operand each instruction accepts

Warnings:
It is generated if there is any unused label in the program. This allows programmers to recheck if there are spelling errors though the program assembles successfully. It gives indication that output might not be what is expected

Errors: Common errors that are reported
ERROR: A duplicate label was found on line X
ERROR: Extra information on line X, maybe there is mistype error
ERROR: Invalid label name on line X
ERROR: Illegal mnemonic on line X
ERROR: A non existent label was found on line X
ERROR: A non numerical value present, data or set instruction requires numeric value on line X
ERROR: Invalid Octal number starting with 0 on line X
ERROR: Invalid hexadecimal number starting with 0x on line X
ERROR: A non decimal value present on line 7
ERROR: operand expected on line X
ERROR: Unexpected operand present on line X

Assumptions:
sum data 0 - As a human we think that assembler should report missing colon after sum. But assembler reports an error: Illegal mnemonic on line X

This is because it reads the first token and checks if it ends with : then only it can be labeled. It doesn't understand whether there are 3 parts in code so it’s likely to be a label. Since sum is not a valid mnemonic error occurs

ldc 5 :load A=5
This reports an error: Extra information on line X, maybe there is mistype error
Assembler expects only one operand after mnemonic if at all so anything extra leads to error. Error can be due to misspelled ‘;’ to ‘:’ so it reports a possible typing error. Content after ‘;’ is automatically rejected by assembler

There is no segregation in data and code segments which was not required according to the project. So it’s the programmer's responsibility to write code accordingly. File is read from the beginning as instructions and the emulator (to be completed) executes instructions from the first line itself. So data segment should be mentioned by programmers after HALT instruction. Emulator stops the execution of instructions at the HALT statement. After the HALT statement there should be a data segment (if any) to store values in variables.
If convention is not followed. Then there can be undesired execution of instruction. For example

1.
2. 00000000 label:
3. 00000000 00000000 ldc 0
4. 00000001 FFFFFB00 ldc -5
5. 00000002 00000500 ldc +5
6. 00000003 FFFFFF11 loop: br loop
7. 00000004 00000011 br next
8. 00000005 next:
9. 00000005 00000300 ldc loop
10. 00000006 00000700 ldc var1
11. 00000007 00000005 var1: data 5
12. 00000008 00000012 halt

This code when executed by the emulator reads line 11 and interprets as stnl 00 because it wasn't actually instruction to execute. Only data was stored in the memory but emulator have no clue to differentiate it.
Some instruction like
val2: SET 55 - Opcode: 00000037
This opcode if fetched by the emulator will not execute anything because 37 in hex is not an opcode of any mnemonic hence it will just continue.
So we see there is a chance that execution of data as instruction and it's programmers responsibility to check this for correct implementation.

Assembler

1. Read the file in 1st pass and store it in data structure in (label,mnemonic,operand) format. It uses stored code in 2nd pass
2. Detect label errors
3. Properly Formatted and commented with proper function names
4. Advanced Listing file: Memory dump as well as assembly code is written line by line
5. Instruction.txt contains information of instruction mnemonics along with expected operands
6. It assembles given test program (test1 to test4)
7. It assembles extra test programs (test5, test6, test7). test5 has errors and is duly reported in test5.log. test6 is the corrected code of test5.
8. SET instruction is implemented and can be demonstrated
9. Numerical values can be Decimal, Octal and Hexadecimal. It's implemented for all 3 types

Filename	Log file	List file	Object file	Warnings
test1.asm	No error	generated	generated	1
test2.asm	Errors	empty	No	1
test3.asm	No error	generated	generated	0
test4.asm	No error	generated	generated	0
test5.asm	Errors	empty	No	3
test6.asm	No error	generated	generated	2
test7.asm	No error	generated	generated	0
bubblesort .asm	No error	generated	generated	0

Emulator:

The assembler is written in language C++

g++ emu.cpp -o emu

./emu -trace/-after/-before/-isa filename.o

1. It loads object file correctly
2. It has 4 options
    a. -isa -> Printing Instruction Set
    b. -trace -> Executes line by line showing changes in PC, SP, A and B
    c. -before -> Memory dump before execution
    d. -after -> Memory dump after execution
3. All test programs are executed.
    - test4.o, test6.o, test7.o executes correctly
    - As expected, test1.o has an infinite loop
    - test3.o stops execution due to unknown opcode
4. It detects errant program for example test3.o