Hack Assembler

Task_1:

Write down a Hack Assembler program that opens and reads a text file (with .asm extension) containing valid Hack assembly instructions (having no symbols) and each instruction separated by a new line character. The hack assembler generates a text output file containing the corresponding 16-bit machine code for each instruction (with .hack extension). Hack assembler program performs translation with following characteristics:

- Ignore all the whitespaces, Empty lines /indentation, Line and In-line comments.
- Assume that no builtin symbols, labels and variables are being used (Break symbolic instruction into its underlying fields).
- For each instruction use A-Instruction and C-Instruction
  - A-Instruction: translate the decimal value into a binary value.
  - C-instruction: for each field in the instruction, generate the corresponding binary code.
  - Consider the table given below to verify the binary code against C-instructions
  - Combines the binary codes into16-bit instructions.
- Write the translated instruction in the output file with the same input file name but with .hack extension
- The name of the code file should be task1.c/.cpp.

comp		c1	c2	c3	c4	c5	c6
0		1	0	1	0	1	0
1		1	1	1	1	1	1
-1		1	1	1	0	1	0
D		0	0	1	1	0	0
A	M	1	1	0	0	0	0
!D		0	0	1	1	0	1
!A	!M	1	1	0	0	0	1
-D		0	0	1	1	1	1
-A	-M	1	1	0	0	1	1
D+1		0	1	1	1	1	1
A+1	M+1	1	1	0	1	1	1
D-1		0	0	1	1	1	0
A-1	M-1	1	1	0	0	1	0
D+A	D+M	0	0	0	0	1	0
D-A	D-M	0	1	0	0	1	1
A-D	M-D	0	0	0	1	1	1
D&A	D&M	0	0	0	0	0	0
D|A	D|M	0	1	0	1	0	1
a==0	a==1						

dest	d1	d2	d3	effect: the value is stored in:
null	0	0	0	The value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register
MD	0	1	1	RAM[A] and D register
A	1	0	0	A register
AM	1	0	1	A register and RAM[A]
AD	1	1	0	A register and D register
AMD	1	1	1	A register, RAM[A], and D register

jump	j1	j2	j3	effect:
null	0	0	0	no jump
JGT	0	0	1	if out > 0 jump
JEQ	0	1	0	if out = 0 jump
JGE	0	1	1	if out ≥ 0 jump
JLT	1	0	0	if out < 0 jump
JNE	1	0	1	if out ≠ 0 jump
JLE	1	1	0	if out ≤ 0 jump
JMP	1	1	1	Unconditional jump

Task 2:

Extend the Hack Assembler program written in Task-1 that can handle the symbols as well. Hack assembler program performs translation with following characteristics:

- Hack Assembler uses the **Two Pass** approach to translate an assembly program.
- **In the first pass, it creates an empty Symbol Table and initializes it with symbols and their values**.
- Assembly program contains symbols of 3 types:
  - **Pre-Defined Symbols:** Start with "@" and occur only in A-instructions.
    - Initialize the table with 23 pre-defined symbols.
    - Consider the given table for 23 pre-defined symbol values.
  - **Label Symbols:** Use to label destination of goto command
    - Label declarations are not translated and generate no code.
    - Replace the label symbol with the address of memory location holding the next instruction.
    - Read the source file and look for label declaration only, and on encountering a label declaration, enter the label name with its corresponding address in the symbol table
- **In the second pass. It replaces the variable symbols with their corresponding values**
  - **Variable Symbols:** Any symbol appearing in the program which is not pre-defined and not used to refer to goto commands.
    - If seen for the first time, assign a unique memory address starting from 16
    - Replace symbol with the address in the symbol table
  - For A-Instruction translate the decimal value into a binary value.
  - For C-instructions consider the table given in Task1.
- It handles White Spaces.
- Write the translated instructions to the output file with the name same as the input assembly file but with .hack extension.
- The name of the code file should be task2.c/.cpp.

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4