Assembler task tutorial

Name: Hoda Khaled Yehya

Section: 14

First: How to use the program

1) Run the program
2) You will be asked to enter 1 for entering code or 2 to exit
3) If you enter 1 you will be asked to enter your code
4) If you press 2 the program will exit
5) If you enter your code the program will realize the end of it from the "end" key word, then 2 tables will appear analyzing your code and you will be asked again whether for a new code or close the program.

The program initially read its input from a file. If you want to enter your input you will whether enter it in the file or comment the line `freopen( "output.out", "r", stdin )` in the beginning of the function main.

Second: analysis of functions

1) Inc_add function: this function takes the address of the processed instruction and increment it. The address is a string as it may contains alphabet (a, b, c, d, e, f) as well as numbers.
2) Format_string function: this function takes each line and decomposes it into instructions and labels (if they exist) then transfer them all into lower case strings (as the instructions saved in the program are all in low case).
3) Get_address function: given a label name will return its address from address symbol table or return "not found" if it doesn't exist.
4) Begin function: this function runs in the beginning of the program where it reads all instructions from file (input.in) and saves it in the program to use it during the analysis of code.
5) hexaToDecimal function: this function takes the hexa decimal generated for each in the code and generate its corresponding binary code.
6) First_pass function: this function passes by the whole code, catches the labels and save them in the address symbol table and then show it to the user.
7) Second_pass function: this function analyze the whole code and generate for each line its hexa and binary codes then show the table containing the whole analysis.

Third: Types of errors caught by the program:

1) If you enter a reserved word as a label. Example: STA, LDA Y an error message will appear like that:
```
Error at line 2
Variable can't be a reserved word.
```
2) If you enter a wrong instruction. Example: Addz Z an error message will appear like that:
```
Error at line 3.
Instruction not recognized.
```
3) If you use a variable which wasn't generated in the address symbol table. Example: STA Y an error message will appear like that
```
Error at line 3.
Undefined variable.
```
4) If you enter any character or string instead of I for indirect address. An error message will appear like that:
```
Error at line 3.
Expected I for indirect address.
```
5) If you enter any other string after I in memory reference instructions. An error message will appear like that:
```
Error at line 3.
Excess labels.
```
6) If you forgot to enter a variable for the memory reference instruction. An error message will appear like that:
```
Error at line 3
variable is missing.
```
7) If you write a label name or any other string next to a register reference instruction or input/output instruction. An error message will appear like that:
```
Error at line 5
This instruction needn't any memory reference.
```

If you run the program with the input files sent with the program file all of this errors will show up after the analysis of 2 right codes.