CSC 236 Programming Assignment

KEY

Summary

The key assignment is worth 4% of your grade. For this assignment, you will be submitting a file named **key.ans**. This file is created by the grading system when you grade your KEY program. The grade is based on correct answers, documentation and the number of executable instructions used.

If you have questions about the assignment, feel free to use our EdStem forum, but be sure not to post any code from your solution in a public post (even code that's not working). You can also stop by during office hours to get help from anyone on the teaching staff. Remember, if you need help, you need to make sure your code is documented, including block comments above sections of your code and line comments saying what you are trying to do with each instruction. We'll also expect you to have a design for the program that you're trying to follow.

Specifications

The KEY reads printable characters (20h-7Fh) from the Standard Input. This can be used to read characters actually typed in by the user or to read from a file via redirected standard input. The program won't automatically echo every key, so you will use DOS system call ah=08h and int 21h. The program should process characters one at a time, immediately, as they are read. For each character read, it will perform the following:

- If the character is an upper case letter (A-Z) then write it to the standard output (ah=02h, dl=char, int 21h).
- If the character is a lower case letter (a-z) then convert it to upper case and write it to the standard output. You can perform this conversion by subtracting 20h from the code for a lower-case letter. That's the difference between the start of the capital letters and the lower-case letters in ASCII.
- If the character is a blank (20h) or period (2Eh) then write it to the standard output.
- If the character is anything else then do not write it to the standard output; just throw the character away.

After processing the character, if the character is a period (2Eh) then end your program's execution.

Example user input:

```
Hello, testing 123.
```

Example program output:

```
HELLO TESTING .
```

To help simplify your program, consider the following:

- You only need to handle the printable ASCII characters in the range of 20h-7Fh.
- All grading data is guaranteed to have the terminating period.

- You do not need to handle the special extended ASCII characters such as the function keys F1 through F12, Home key, PgUp, PgDn, Del, etc. These keys will generate two character codes each. The first call returns 00h and the second call returns a special extended code. See the section of the notes on File Input / Output for additional information.
- Do not write anything else to the output stream such as prompts or messages.

Step 1 : Create a design

A design converts the English language statement of a problem into a series of logical steps that can be coded. It assures the problem is understood well enough to be solved. A good design for an assembly language program is a working program in Java or C/C++.

- If you would like to see a basic design for KEY then see the section of the notes titled How To Design A Program.
- See the last page of this document for One Way To Approach Coding In assembly language.
- If you need help from the staff, we expect you have a design you can share with us.

For sample C and Java code that reads and writes the keyboard and display, see the READFILE course locker.

For most students, developing a working version of KEY is not difficult. However, for most students, the new variable in the equation is coding for efficiency. It will require reasonable thought and initiative to achieve full credit for efficiency. Using compares and jumps, the most common solutions have 22-24 instructions. To reach 20 instructions using compares and jumps requires an innovative approach to the logic used to solve the problem. There is an alternative to using just compares and jumps. Reading the section of the notes that cover Assembler Opcodes and Directives can help you think about this. The staff can provide feedback on your planned techniques and guidance once you have a working program. The earlier you start, the greater the opportunity for success.

Step 2 : Code your solutions

Use a text editor to convert your design into an assembly language program. Your source code file must be named **key.asm** and the executable program will be named **key.exe**.

For editing your source file, you can use any text editor you'd like on your host. Just be sure to save your source file in the KEY subdirectory of your P23X shared directory. For example, if you're working on a MacOS system, you could use BBEdit or Sublime Text to write your source file and then save it under KEY. If you're using Windows, you could edit your source file with notepad++ or Visual Studio Code and save it under KEY. If your DOSBox program is running while you edit your source file, remember that you may have to enter the **rescan** command in DOSBox to get it to see changes to files from outside DOSBox. Really, I've only needed to use rescan to get new files to show up in the directory listing. I've seen changes to the contents of a file seem to show up in DOSBox even when I don't re-run rescan. Still, running an extra rescan doesn't hurt.

Remember you must use these directives in your code:

```
.model small          ; 64k code and 64k data
.8086                 ; only allow 8086 instructions
```

Use these commands to assemble and then link your program.

```
     ml    /c   /Zi  /Fl  key.asm
     link  /CO  key.obj
```

Retrieve the testing and grading files. All files are packed together in one self-extracting file named **unpack.exe**. It's with the rest of the materials for the KEY assignment on our Moodle page. You can click on the unpack.exe link to download this executable file and save it on your hard drive in the KEY subdirectory of your shared P23X directory. Or, you can change directory to the KEY subdirectory and then use curl to pull down a copy of this archive with the following command.

```
curl -O https://people.engr.ncsu.edu/dbsturgi/236/files/key/unpack.exe
```

Then, start up DOSBox and run the usual commands to get ready to work. Then, change to the KEY directory and unpack the archive you just downloaded. Depending on where you put your shared directory, it will probably be something like the following:

```
mount e ~/P23X
E:
dbset
cd KEY
unpack
```

Step 3 : Test and debug

Testing is not the same as grading. Testing is the work you do to determine whether there are defects in your code. Grading is the university equivalent of a real world customer's acceptance test. It's as if you have shipped your product to the customer and they are now using that product. In our class, after the first grading run, every subsequent one that you need to get the program to work correctly, is like a bug that was shipped to your customer.

To test KEY follow these instructions:

- Decide what input lines should be typed into KEY to assure KEY functions correctly. These should include the spectrum of possible ASCII characters (upper case letters, lower case letters, numbers, punctuation, etc.).

- Run these tests against your program, using the testkey procedure, to determine if there are any defects.

  The **testkey** program is a batch procedure. It is required to use the testkey procedure for testing. Grading cannot occur unless testing is done using testkey.

  To run a test, type the following DOS batch command: **testkey**

  The testkey procedure will run your KEY program with the input coming from the keyboard.

- Isolate and fix any defects found.
  1. Try to find the bug in your design and your assembly source code.
  2. If you cannot find the bug in the source code then you must now use the debugger. Determine the shortest input sequence that fails. This is important since tracing code in the debugger is time consuming, and you do not want to be tracing through the code for long input sequences. Activate the debugger (cv key) and trace each instruction that processes your input sequence. You need to determine if the result of each instruction is correct. This will take time and patience. There is no silver bullet.
  3. Repeat running your tests until the program works correctly.
- There is one bug, the NT bug described in the *File Input / Output* section of the notes, for which you need a special test.

  To test for the NT bug, type the following DOS batch command: **testkey nt**

  If the system hangs then you probably have the NT bug in your code.
- Before grading, you can get a count of the number of instructions written to solve KEY and you can tell if the system likes your documentation.

Type: **graddoc key.asm**	This will provide documentation information.
Type: **gradmkey key.asm**	This will give you statistics on lines of code written.

If you need help from instruction staff to find a bug or improve efficiency, you can ask a private question on piazza, or you can visit a member of the teaching staff during office hours. Remember, we're going to expect you to have a documented design and your implementation will need to be documented.

Step 4 : Grading

KEY is 100% self-grading. This means you run the grading program to determine how many points your program has earned. This is to your advantage. There are no surprises. You will know your grade when you submit your solution.

The system needs access to your source file and executable file. Assure current versions of **key.asm** and **key.exe** are located in the current directory. The **gradkey** procedure will run the grading program, give you a grade and report defects.

Type the following DOS command: **gradkey**

Status information is written to a file named **results**. If a defect is found then the results file will tell you which test failed.

Fix any defects and re-run the grading program. Your KEY implementation must work for all input values 20-7F hex to receive any credit.

Grading is not a replacement for your testing, so the grading program will count the number of times it was executed. The results file will contain the chronology of your testing and grading.

Your final grade will be based on the following three components.

- 60 points for getting the correct answer.
  Your KEY code must work for all input values between 20h and 7F hex to receive any credit.
- 20 points for the number of executable instructions written to correctly complete this assignment. The goal is an efficient program. Your source code will be analyzed and you will be awarded 0 to 20 points, with the most points for a program with the fewest instructions. You can find the target count in the lecture material from lecture 9.
- 20 points for documentation of a program that functions correctly.

For grading, efficiency and documentation are only a concern after the code works correctly. Documentation grading will only occur after your code passes the functional test. Efficiency grading will only occur after your code passes the functional test.

How do you improve efficiency or documentation without being accused of running the grading program too often? After you have KEY working correctly, you may make additional grading runs to improve efficiency or documentation. To mark these as grading runs that were only done to improve efficiency or documentation, execute this command, one time, after your program has earned the points for getting the correct answers: **testkey mark**

All subsequent grading runs will be marked as only being done to improve efficiency or documentation.

Step 5 : Submit your solutions

The file you electronically submit is: **key.ans**

This is the only acceptable file. Although the grading system creates the file, you are responsible to assure its content is correct. This file should contain two other files concatenated together.

- The first file is the results file.
- The second file is key.asm, which is your source code.

When you look at key.ans to verify its contents, you should find this line of text with your grade.

```
++ Grade ++ nnn = Total grade generated by the Grading System.
```

If that line is not in the file then do not submit it. You have a problem. You must resolve that problem.

Sample key.ans

```
Test date: 04/19/09  08:39:56
The number of times you used the grading program is 1
In the business world, your customer replaces the grading program.
The goal is a correct program with the fewest grading attempts.
```

```
Key Grading System Version 4.1
Student: Last=Lasher               First=Dana                 Type=Individual
Grading and testing history log
       User Test.    Date: 04/19/09  08:38:00  keyboard     test
       User Test.    Date: 04/19/09  08:38:10  keyboard     test
       User Test.    Date: 04/19/09  08:38:18  keyboard     test
  *** Grading Run. Date: 04/19/09 08:39:56 Running test
Test ran correctly.
++ Grade ++ 60 = Points earned for correct answers.

-> A major reason for programming in assembler is to maximize efficiency. This can
be instructions written and/or instructions executed.
Please read the section in the class notes titled Code Complexity.

We are now analyzing your source code. You will be graded on
the number of instructions you wrote to solve this problem.
The target for the number of executable instructions written is:
20 points for   0 -  20
15 points for  21 -  22
10 points for  23 -  24
05 points for  25 -  26
00 points for  27+
Your actual counts are:
Lines .......  90    Comments ...  59    Labels .......   5
Directives ..   6    Variables ..   0    Executables ..  20    McCabe ..   7
The number of executable instructions written is considered best
++ Grade ++   20 = Points earned for code written efficiency.

The McCabe number for your KEY.ASM is 7 which is considered
fine for the KEY assignment and indicates reasonable program structure.

-> Analyzing program documentation.
The heuristic algorithms used may have problems with your specific code. Contact
the instructor if you detect a problem with your grade.
DOCPH-0032 The program header format is adequate.
DOCHB-0.75 Use of code block headers is adequate.
DOCLC-0.95 Use of line comments is adequate.
++ Grade ++ 20 = Points earned for correct documentation.

-> Calculating the total grade for the program.
This grade is subject to adjustment by the instructor.
This grade will be reduced by any late submission penalty.
++ Grade ++ 100 = Total grade generated by the Grading System.

-> Statistics on testing and grading
     3  = Number of user tests.
     1  = Number of grading runs to get the program working.
     0  = Number of grading runs to improve efficiency or documentation.

+- atyC<6&}xC86&k~C6&juiC86&zuzC766&Zkyz&jgzk@&6:57?56?&&6>@9?@;<&

Building the file key.ans for electronic submission…
;----------------------------------------------------------------------
; Prog:    KEY
```

Make sure the file has this line with grade information.

One Way to Approach Coding in Assembly Language

The KEY assignment is your first opportunity to write assembler code. If you are having trouble writing assembler, this suggestion may help you. There is a way to approach assembler coding that provides a high probability of completing a functional program. It is not the most efficient way, but it is the safest way. Here's an approach you can consider.

A safe way to approach KEY is for you play the role of a 'compiler'. That is, code KEY in the high-level language (HLL) of your choice. Then convert each high-level language statement into assembler.

- Use each high-level language statement as a header block.
- Code the assembler instructions to implement that single HLL statement.

There is an example below. This technique provides an advantage. It allows someone to easily determine if the assembler code, does in fact, implement the high-level language statement. It is a good way to effectively use header blocks to document a program.

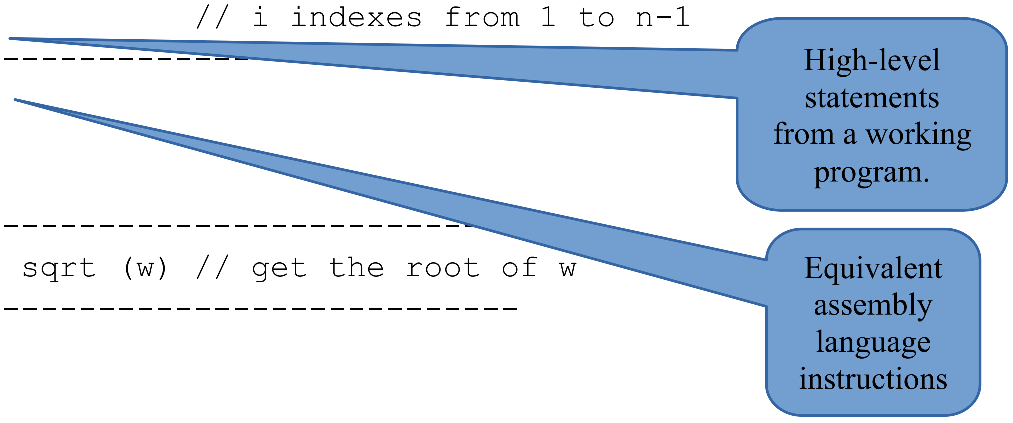
How does this help you?

1. You know your design works, because it functions in the high-level language.
2. It is possible you will need help to fix a bug in your code or to improve efficiency. That requires that the person helping you understands what you expect your code to do. The high level language statements can provide that information.

If you want another hintm read section of the notes titled *How To Design A Program* and you may be surprised at which program it uses as an example.

Here's a pictorial example of using high-level language statements as header blocks for assembler.

```
;------------------------------------------------
; for (i=1;i<=n-1;i++)          // i indexes from 1 to n-1
;------------------------------------------------
    mov       ;comment
    add       ;comment
    cmp       ;comment
;------------------------------------------------
; if (x == y) then q = sqrt (w) // get the root of w
;------------------------------------------------
    div       ;comment
    sub       ;comment
;-----------------------------------------------
; if (list[j-1]>list[j]) {      // if val(j-1) > val(j) swap
;       t=list[j-1];
;       list[j-1]=list[j];
;       list[j]=t;  }
;------------------------------------------------
    mov       ;comment
```


High-level statements from a working program.
Equivalent assembly language instructions

```
    sub        ;comment
    cmp        ;comment
    mov        ;comment
```

End of document