

# CSED101. Programming & Problem solving

## Spring, 2018

### Programming Assignment #4 (70 points)

서홍석 (hsseo@postech.ac.kr)

■ Due: 2018.05.16 23:59

■ Development Environment: Windows Visual Studio 2017

#### ■ 제출물

- C Code files (assn4.c)
  - 프로그램의 소스 코드를 이해하기 쉽도록 반드시 주석을 붙일 것. (주석이 불충분 할 시 감점)
- 보고서 파일 (.doc(x) or .hwp) 예) assn4.doc(x) 또는 assn4.hwp
  - AssnReadMe.pdf 를 참조하여 작성할 것.
  - 프로그램 실행화면 캡처하여 보고서에 포함시키고 간단히 설명 할 것!!
  - 명예서약(Honor code): 표지에 다음의 내용을 포함한다. "나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다." 보고서 표지에 명예서약이 없는 경우는 과제를 제출하지 않은 것으로 처리한다.
  - 소스코드와 보고서 파일을 LMS를 이용하여 제출한다.
- LMS에 제출할 때는 소스파일과 보고서를 압축하지 말고 각각 업로드 할 것.

#### ■ 주의사항

- 문제의 요구사항을 반드시 지킬 것.
- 에러 메시지를 포함하여 모든 문제의 출력 형식은 공백의 개수를 제외하고 아래의 예시들과 완전히 동일해야 하며, 같지 않을 시는 감점이 된다.
- 과제 작성시 전역변수는 사용할 수 없으며, 사용자 정의 함수를 적절히 작성하도록 한다. (main() 함수만 사용한 경우, 감점 대상으로 간주)
- 컴파일 & 실행이 안되면 무조건 0점 처리된다.
- 하루 late시 20%가 감점되며, 3일 이상 지나면 받지 않는다. (0점 처리)
- 부정행위에 관한 규정은 POSTECH 전자컴퓨터공학부 학부위원회의 'POSTECH 전자컴퓨터공학부 부정행위 정의'를 따른다. (LMS의 과목 공지사항의 제목 [document about cheating]의 첨부파일인 disciplinary.pdf를 참조할 것.)
- 이번 과제에서는 추가 기능 구현에 대한 추가 점수는 없습니다.

## ■ Problem: Interpreter for a Simple Matrix Language

### (목적)

이번 과제를 통하여 포인터와 동적할당, 배열의 사용법을 익힌다.

### (소개)

인터프리터는 프로그래밍 언어의 소스 코드를 바로 실행하는 컴퓨터 프로그램으로, 고급 언어로 작성된 소스 코드 명령들을 한번에 한 줄씩 실행한다. 이는, 전체 소스 코드를 읽어서 실행 가능한 기계어 프로그램으로 번역하는 컴파일러와 대비된다. (참고자료: [Interpreter in Wikipedia](#))

본 assignment에서는 간단한 행렬 연산을 수행할 수 있는 행렬 연산용 프로그래밍 언어를 정의하여 제시하고, 정의된 언어를 처리하는 인터프리터를 구현하는 것을 목표로 한다. (참고자료: [Matrix in Wikipedia](#)) **[경고: 본 문서를 꼭 끝까지 다 읽고 구현을 시작할 것]**

### (설명)

Simple Matrix Language(SML)은 사용자로 하여금 벡터와 행렬 연산을 쉽게 할 수 있게 하는 수학용 프로그래밍 언어이다. SML은 Vector와 Matrix 타입의 변수를 다룰 수 있으며, 해당 변수들의 값을 이용하여 행렬 연산을 수행할 수 있다. 다음은 SML의 간단한 예제 프로그램이다.

Example code 1 (검정색: 사용자 입력, 빨간색: 출력)	설명
<pre> USE 6 VECTOR a 4 1 2 3 4 PRINT a [  1.00,   2.00,   3.00,   4.00 ] VECTOR b 4 4 4 2 2 VECTOR c 4 1 1 1 1 ADD a b c PRINT c [  5.00,   6.00,   5.00,   6.00 ] MATRIX d 2 3 1 2 3 4 5 6 MATRIX e 3 2 7 8 9 0 1 2 MATRIX f 2 2 1 1 1 1 MATMUL d e f PRINT f [[ 28.00, 14.00 ],  [ 79.00, 44.00 ]] END </pre>	<p>프로그램에서 최대 6개의 변수 사용을 명시 4차원 벡터 a 정의 및 초기화 정의된 벡터 a 출력</p> <p>4차원 벡터 b &amp; c 정의 및 초기화</p> <p>벡터 연산 (<math>c = a + b</math>) 후 출력</p> <p>2x3 행렬 d 정의 및 초기화</p> <p>3x2 행렬 e 정의 및 초기화</p> <p>2x2 행렬 f 정의 및 초기화</p> <p>행렬 연산 (<math>f = d \times e</math> [행렬곱]) 후 출력</p> <p>프로그램 끝 선언</p>

본 assignment의 목표는 위의 example code를 (검정색 부분만) 입력 받아서 그 의미를 해석하고 의미대로 명령을 수행하여 (빨간색으로 표시된) 적절한 결과를 출력하는 것이다.

SML로 작성된 프로그램은 총 12개의 명령어의 조합으로 구성되어 있으며, 모든 프로그램은 USE 명령어로 시작하여 END 명령어로 끝난다. 다음은 각 명령어에 대한 설명과 실행 예제이다. (참고: 각각의 명령어는 잘못된 입력이 주어졌을 때 적절한 오류 메시지를 출력해야한다. 해당 내용은 뒤쪽의 오류 메시지 설명 부분을 참조할 것.)

명령어	설명 (빨간색: 출력, 파란색: 코드에 대한 설명)
USE	<p>SML 프로그램에서 사용할 변수의 수를 지정하며 모든 SML 프로그램은 USE 명령어로 시작한다. SML의 변수명은 'a', 'b', 'c'... 와 같이 alphabet 한자리로 정해져 있으며, USE 명령어를 사용하여 사용할 변수를 지정하면, 프로그램 내에서 'a'부터 n개의 연속된 변수를 정의하여 사용할 수 있다. 최대 26개의 변수를 사용할 수 있다. Variable들의 정보를 저장할 공간은 use 명령어를 통해 동적으로 할당 받아야 한다.</p> <p><b>사용 문법:</b> USE n</p> <ul style="list-style-type: none"> <li>인자 1 (n): 사용할 변수의 개수 (1~26 사이의 수)</li> </ul> <p><b>사용 예제:</b> USE 3 [변수 'a', 'b', 'c'를 matrix나 vector로 정의하여 사용할 수 있다.]</p>
END	<p>SML 프로그램을 끝내는 명령어로, 모든 동적 할당 메모리를 시스템에 반환한다.</p> <p><b>사용 문법:</b> END</p> <p><b>사용 예제:</b> END [SML 프로그램을 끝내고 사용한 모든 동적 메모리를 반환]</p>
VECTOR	<p>USE 명령어로 지정하여 사용 가능한 변수를 벡터로 정의한다. VECTOR 명령어를 사용한 경우 변수를 초기화 하기 위해 벡터 원소의 수만큼의 숫자를 입력 받는다. 벡터의 원소를 저장하기 위한 공간을 VECTOR 명령어를 처리하며 동적 할당 받아야 한다.</p> <p><b>사용 문법:</b> VECTOR var n</p> <ul style="list-style-type: none"> <li>인자 1 (var): 정의할 변수 명</li> <li>인자 2 (n): 정의할 벡터의 차원</li> </ul> <p><b>사용 예제:</b> VECTOR a 3 [변수 'a'를 3차원 벡터로 정의] 1 2 3 [변수 'a'의 값을 초기화]</p>
MATRIX	<p>USE 명령어로 지정하여 사용 가능한 변수를 행렬로 정의한다. MATRIX 명령어를 사용한 경우 변수를 초기화 하기 위해 행렬 원소의 수만큼의 숫자를 입력 받는다. 행렬의 원소를 저장하기 위한 공간을 MATRIX 명령어를 처리하며 동적 할당 받아야 한다.</p> <p><b>사용 문법:</b> MATRIX var row col</p> <ul style="list-style-type: none"> <li>인자 1 (var): 정의할 변수 명</li> </ul>

	<ul style="list-style-type: none"> <li>인자 2 (row): 정의할 행렬의 행 수</li> <li>인자 3 (col): 정의할 행렬의 열 수</li> </ul> <p><b>사용 예제:</b>  MATRIX b 2 2    [변수 'b'를 2X2 행렬로 정의]  1 2                [행렬 'b'의 값을 초기화]  3 4</p>
PRINT	<p>벡터 혹은 매트릭스로 정의된 변수를 출력한다. 벡터의 경우 열벡터(column vector)로 출력한다. (참고자료: <a href="#">column vector in Wikipedia</a>)</p> <p><b>사용 문법:</b> PRINT var</p> <ul style="list-style-type: none"> <li>인자 1 (var): 출력할 변수 명</li> </ul> <p><b>사용 예제:</b>  PRINT a            [벡터 변수 'a' 출력, 변수 'a'는 먼저 벡터로 정의 되어있어야 한다.]  [    1.00,  2.00,  3.00 ]</p> <p>PRINT b            [행렬 변수 'b'를 출력, 변수 'b'는 먼저 행렬로 정의 되어있어야 한다.]  [[    1.00,    2.00 ],  [    3.00,    4.00 ]]</p>
PRINTSIZE	<p>벡터 혹은 행렬로 정의된 변수의 크기를 출력한다. 출력 형식은 사용 예제를 참고.</p> <p><b>사용 문법:</b> PRINTSIZE var</p> <ul style="list-style-type: none"> <li>인자 1 (var): 크기를 출력할 변수 명</li> </ul> <p><b>사용 예제:</b>  PRINTSIZE a            [벡터 변수 'a'의 크기 출력]  'a' is a 3-dimensional vector.</p> <p>PRINTSIZE b            [행렬 변수 'b'의 크기 출력]  'b' is a 2 X 2 matrix.</p>
ADD	<p>두 벡터 혹은 행렬의 합을 구하여 다른 변수에 넣는다. 세 개의 벡터 혹은 행렬은 모두 같은 크기를 가져야 한다.</p> <p>(trg = src1 + src2)  (참고자료: <a href="#">vector addition in Wikipedia</a>, <a href="#">matrix addition in Wikipedia</a>)</p> <p><b>사용 문법:</b> ADD src1 src2 trg</p> <ul style="list-style-type: none"> <li>인자 1 (src1): 더해질 벡터 혹은 행렬 변수</li> <li>인자 2 (src2): 더해질 두번째 벡터 혹은 행렬 변수</li> <li>인자 3 (trg): 더해진 값을 저장할 벡터 혹은 행렬 변수</li> </ul>

	<p><b>사용 예제:</b></p> <pre> USE 3 VECTOR a 2 1 2 VECTOR b 2 4 3 VECTOR c 2 0 0 ADD a b c    [c = a + b] PRINT c [    5.00,    5.00 ] </pre>
ELEMMUL	<p>두 벡터 혹은 행렬의 원소곱(element-wise multiplication)을 구하여 다른 변수에 넣는다. 세 개의 벡터 혹은 행렬은 모두 같은 크기를 가져야 한다. (trg = src1 @ src2, 단 @은 원소곱 연산자) (참고자료: <a href="#">element-wise multiplication (also known as hadamad product) in Wikipedia</a>)</p> <p><b>사용 문법:</b> ELEMMUL src1 src2 trg</p> <ul style="list-style-type: none"> <li>인자 1 (src1): 원소별로 곱해질 벡터 혹은 행렬 변수</li> <li>인자 2 (src2): 원소별로 곱해질 두번째 벡터 혹은 행렬 변수</li> <li>인자 3 (trg): 곱해진 값을 저장할 벡터 혹은 행렬 변수</li> </ul> <p><b>사용 예제:</b></p> <pre> USE 3 MATRIX a 2 2 1 2 3 4 MATRIX b 2 2 0 1 2 3 MATRIX c 2 2 1 1 1 1 ELEMMUL a b c    [c = a @ b] PRINT c [[    0.00,    2.00 ],  [    6.00,   12.00 ]] </pre>
MATMUL	<p>두 행렬 변수의 행렬 곱 (matrix multiplication)을 구하여 다른 행렬 변수에 넣는다. 세 변수의 사이즈는 행렬 곱을 할 수 있는 사이즈여야 한다. (trg = src1 X src2, 단 X는 행렬곱 연산자)</p>

	<p>(참고자료: <a href="#">matrix multiplication in Wikipedia</a>)</p> <p><b>사용 문법:</b> MATMUL src1 src2 trg</p> <ul style="list-style-type: none"> <li>인자 1 (src1): 행렬곱의 첫번째 행렬 변수</li> <li>인자 2 (src2): 행렬곱의 두번째 행렬 변수</li> <li>인자 3 (trg): 곱해진 값을 저장할 행렬 변수</li> </ul> <p><b>사용 예제:</b></p> <pre>USE 3 MATRIX a 2 2 1 2 3 4 MATRIX b 2 2 0 1 2 3 MATRIX c 2 2 1 1 1 1 MATMUL a b c    [c = a X b] PRINT c [[  4.00,  7.00 ],  [  8.00, 15.00 ]]</pre>
VECTORIZE	<p>주어진 행렬 변수를 같은 크기의 벡터 변수로 변환한다. 벡터에는 행렬의 각 열의 값이 우선되는 정렬방식으로 행렬의 각 원소를 그대로 저장한다. 만약 인자로 벡터 변수가 주어지면 그대로 둔다.</p> <p><b>사용 문법:</b> VECTORIZE var</p> <ul style="list-style-type: none"> <li>인자 1 (var): 벡터로 변환할 벡터 혹은 행렬 변수</li> </ul> <p><b>사용 예제:</b></p> <pre>MATRIX a 2 2 1 2 3 4 PRINT a [[  1.00,  2.00 ],  [  3.00,  4.00 ]]</pre> <p>VECTORIZE a    [2x2 행렬인 'a'를 4차원 벡터로 변환]</p> <pre>PRINT a [  1.00,   2.00,   3.00,   4.00 ]</pre>

RESIZEMAT	<p>주어진 벡터 혹은 행렬 변수를 주어진 크기의 행렬 변수로 바꾼다. 단, 전체 원소의 개수는 유지되어야 하며, 행렬의 원소는 열을 우선으로 하는 순서대로 채운다.</p> <p><b>사용 문법:</b> RESIZEMAT var row col</p> <ul style="list-style-type: none"> <li>인자 1 (var): 행렬로 변환할 벡터 혹은 행렬 변수</li> <li>인자 2 (row): 변환된 행렬의 행 수</li> <li>인자 3 (col): 변환된 행렬의 열 수</li> </ul> <p><b>사용 예제:</b></p> <pre>USE 1 VECTOR a 4 1 2 3 4 RESIZEMAT a 2 2    [벡터 변수 'a'를 2x2 행렬 변수로 변환] PRINT a [[  1.00,  2.00 ],  [  3.00,  4.00 ]] RESIZEMAT a 1 4    [2x2 행렬 변수 'a'를 1x4 행렬 변수로 변환] PRINT a [[  1.00,  2.00,  3.00,  4.00 ]]</pre>
TRANSPOSE	<p>주어진 벡터 혹은 행렬 변수를 전치하여 전치행렬(transpose matrix)을 구한다. 단, 벡터의 경우 열벡터이므로 nx1 행렬로 보고 전치행렬을 구한다. (참고 자료: <a href="#">transpose in Wikipedia</a>)</p> <p><b>사용 문법:</b> TRANSPOSE var</p> <ul style="list-style-type: none"> <li>인자 1 (var): 전치할 벡터 혹은 행렬 변수</li> </ul> <p><b>사용 예제:</b></p> <pre>USE 2 VECTOR a 3 1 2 3 PRINT a [  1.00,   2.00,   3.00 ] TRANSPOSE a    [벡터 'a' 전치] PRINT a [[  1.00,  2.00,  3.00 ]]</pre> <pre>MATRIX b 2 3 1 2 3 4 5 6 PRINT b</pre>

	<pre> [[ 1.00, 2.00, 3.00 ],  [ 4.00, 5.00, 6.00 ]] TRANSPOSE b    [행렬 'b' 전치] PRINT b [[ 1.00, 4.00 ],  [ 2.00, 5.00 ],  [ 3.00, 6.00 ]] </pre>
--	--

#### (주의사항)

- 모든 SML 프로그램은 USE 명령어로 시작하여 END 명령어로 끝나고, USE 명령어와 END 명령어는 단 한번씩만 사용한다.
- 모든 SML 명령어는 한 줄에 하나씩만 주어지며, 빈 줄은 무시한다.
- 잘못된 SML 명령어 입력 등에 대한 적절한 예외 처리를 해야 한다. (아래의 요청사항 참고)
- 모든 에러 메시지는 stderr 스트림에 출력해야 한다. (stdout에 출력할 경우 감점)
- USE와 VECTOR, MATRIX 명령어로 정의되는 SML 변수는 C 프로그램 내부에서 동적으로 할당 해야한다. 모든 프로그램이 완벽히 돌아가더라도, USE를 통해 주어진 개수 만큼을 동적으로 할당 받지 않고 최대 개수인 26개를 미리 정의하여 사용하면 최소 전체 점수의 70% 감점!
- 동적으로 할당 받은 모든 변수는 적절한 때에 시스템에 반환 해야 한다. (그렇지 않을 시 큰 감점요인)
- 파일 이름은 "assn4.c"로 저장 할 것
- 보고서는 "assn4.doc" or "assn4.hwp"로 저장 할 것
- 출력은 아래의 "실행예제"와 동일하도록 작성 할 것

#### (가정)

- 파일 이름은 공백을 포함하지 않으며, 30자를 넘지 않는다.
- 실행 시, 한 줄은 100자를 넘지 않는다.
- 명령어의 인자 값은 항상 올바른 형식으로 주어진다. 다만 명령어는 잘못된 명령어가 입력 될 수 있음.

#### (요청사항)

- 사용자의 입력에 따라, SML 프로그램을 한 줄씩 keyboard로 입력 받는 것뿐만 아니라, 미리 작성한 SML 프로그램을 파일로 입력 받을 수 있어야 하며, 프로그램의 실행 결과도 파일로 출력 가능해야한다. (아래 실행 예제 참조)
- 각 SML의 명령어에는 적절한 예외처리를 해서, 사용자가 잘못된 명령어를 입력 했을 때 오류를 출력하고 프로그램을 끝내야 한다. 프로그램이 처리해야 하는 오류의 종류는



다음의 표와 같다. (명령어의 인자 값은 항상 올바른 형식으로 주어진다. 다만 명령어는 잘못된 명령어가 입력 될 수 있음.)

오류 메시지	해당 명령어
The first command should be always USE n. (USE가 아닌 다른 명령어로 프로그램이 시작된 경우)	USE
The variable to define is already defined (정의하는 변수가 이미 정의 된 경우)	VECTOR, MATRIX
The variables should be defined before use. (인자로 주어진 변수가 앞서 정의되지 않은 변수일 경우)	PRINT, PRINTSIZE, ADD, MATMUL, ELEMUL, TRANSPOSE, VECTORIZE, RESIZEMAT
The sizes of variables should match. (인자로 주어진 변수의 크기가 명령어를 수행하기에 부적절 할 때)	ADD, MATMUL, ELEMUL
The numbers of elements in the variables do not match. (인자로 주어진 변수의 원소 수가 명령어를 수행하기에 부적절 할 때)	RESIZEMAT
Incorrect command entered. (잘못된 명령어를 입력 했을 때)	N/A
Incorrect file name entered. (주어진 입력 혹은 출력 파일의 경로가 잘못된 경우)	N/A

(실행예제) 빨간 밑줄은 사용자가 입력하는 부분에 해당됨.

1. 입력 및 출력 파일 선택 (파일을 입력하지 않고 바로 enter를 친 경우, 입력 파일 대신 stdin을, 출력 파일 대신 stdout을 사용한다.)

```

C:\WINDOWS\system32\cmd.exe
Enter input file <default=stdin>:
Enter output file <default=stdout>:

```

2. 파일을 입력한 경우 (다음 프로그램은 example.sml에서 SML 프로그램을 입력받아 처리된 결과를 output.txt 파일에 저장한다.)

```

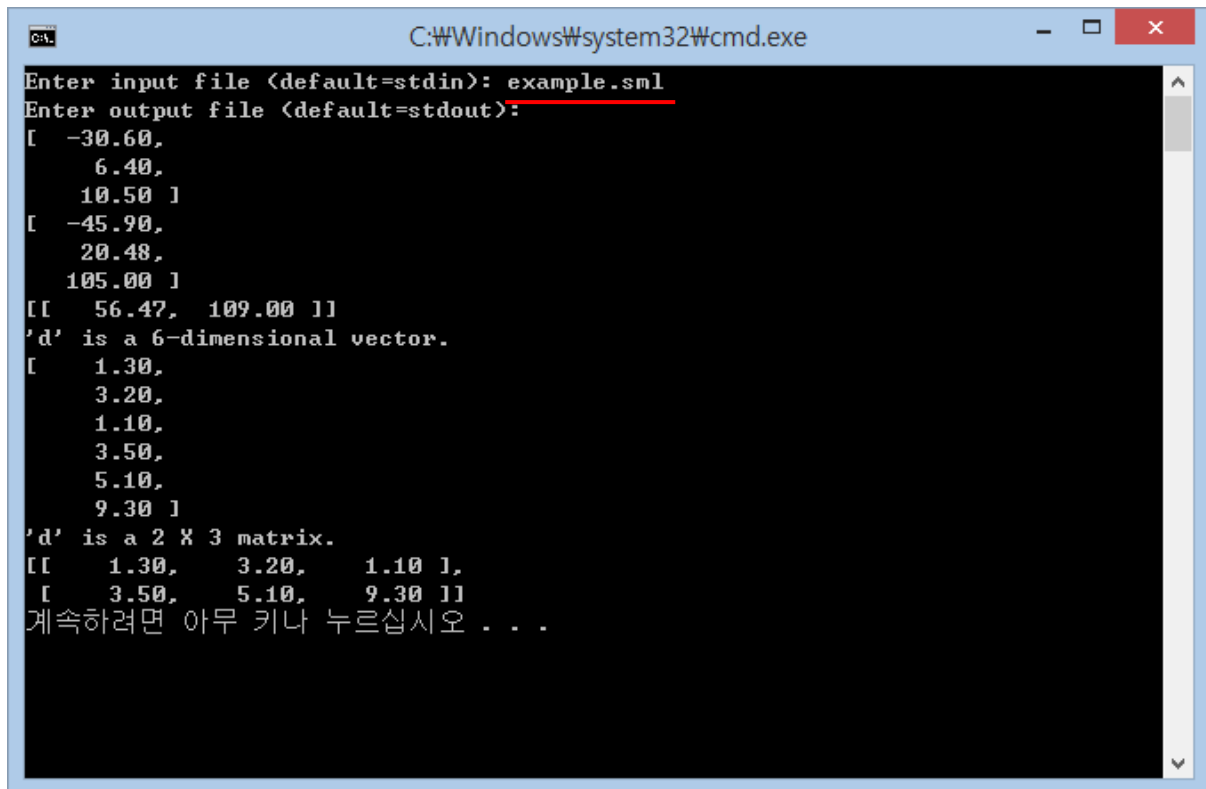
C:\WINDOWS\system32\cmd.exe
Enter input file <default=stdin>: example.sml
Enter output file <default=stdout>: output.txt
계속하려면 아무 키나 누르십시오 . . .

```

- A. 예제: example.sml 파일이 아래와 같이 작성된 경우, 아래와 같은 output.txt 파일이 생성된다.

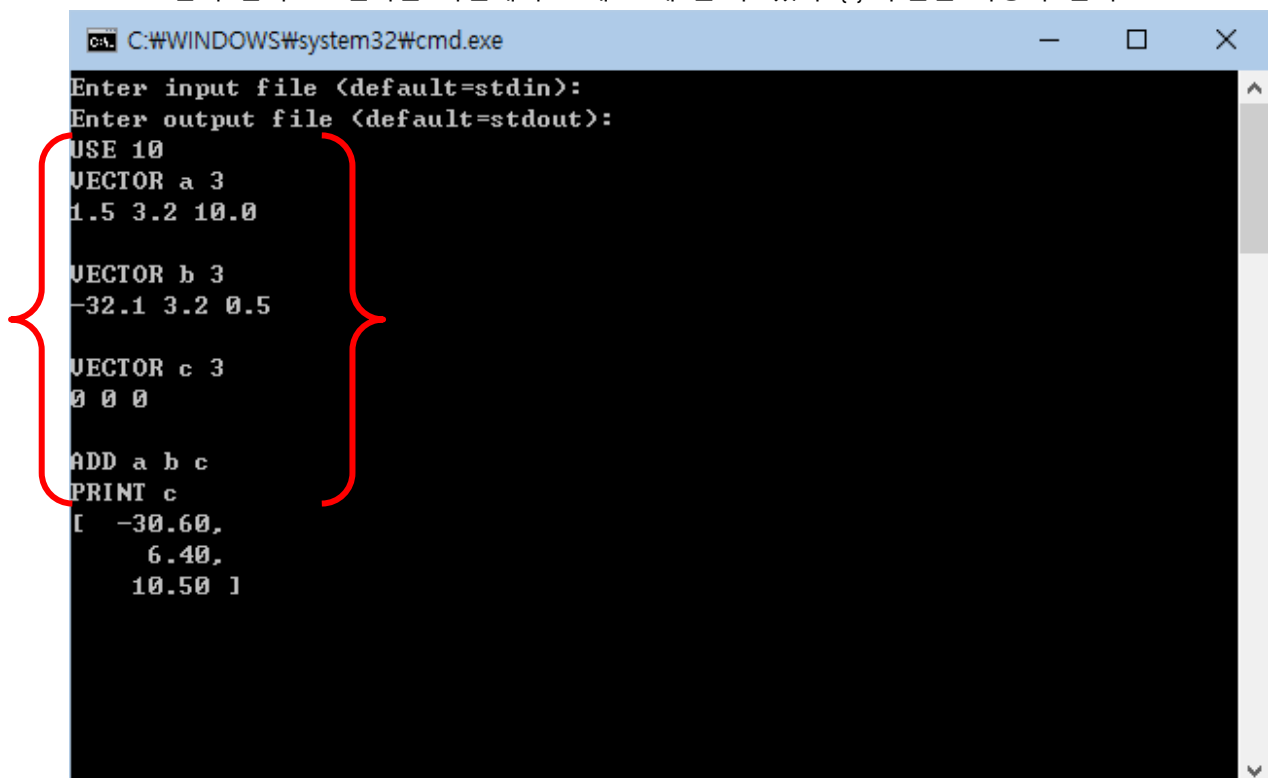
example.sml 파일 예시	output.txt 결과 파일
USE 10 VECTOR a 3 1.5 3.2 10.0  VECTOR b 3 -32.1 3.2 0.5  VECTOR c 3 0 0 0  ADD a b c PRINT c  ELEMMUL a c b PRINT b  MATRIX d 3 2 1.3 3.2 1.1 3.5 5.1 9.3  MATRIX e 1 2 0 0  TRANSPPOSE a MATMUL a d e PRINT e  VECTORIZE d PRINTSIZE d PRINT d  RESIZEMAT d 2 3 PRINTSIZE d PRINT d  END	[ -30.60, 6.40, 10.50 ]  [ -45.90, 20.48, 105.00 ]  [[ 56.47, 109.00 ]] 'd' is a 6-dimensional vector. [ 1.30, 3.20, 1.10, 3.50, 5.10, 9.30 ] 'd' is a 2 X 3 matrix. [[ 1.30, 3.20, 1.10 ], [ 3.50, 5.10, 9.30 ]]

3. 만약 출력 파일을 지정하지 않으면, 입력파일에 주어진 프로그램 실행결과를 화면에 출력한다. 만약 입력 파일을 지정하지 않으면 프로그램을 키보드로 입력 받고 그 결과를 주어진 출력 파일에 저장한다.



```
C:\Windows\system32\cmd.exe
Enter input file <default=stdin>: example.sml
Enter output file <default=stdout>:
[ -30.60,
  6.40,
  10.50 ]
[ -45.90,
  20.48,
  105.00 ]
[[ 56.47, 109.00 ]]
'd' is a 6-dimensional vector.
[ 1.30,
  3.20,
  1.10,
  3.50,
  5.10,
  9.30 ]
'd' is a 2 x 3 matrix.
[[ 1.30, 3.20, 1.10 ],
 [ 3.50, 5.10, 9.30 ]]
계속하려면 아무 키나 누르십시오 . . .
```

4. 만약 둘다 지정하지 않은 경우, 아래와 같이 Interactive mode가 되어, 키보드로 한줄 한줄 입력 받아 그 결과를 화면에서 그때 그때 볼 수 있다. {} 부분은 사용자 입력



```
C:\WINDOWS\system32\cmd.exe
Enter input file <default=stdin>:
Enter output file <default=stdout>:
USE 10
VECTOR a 3
1.5 3.2 10.0
VECTOR b 3
-32.1 3.2 0.5
VECTOR c 3
0 0 0
ADD a b c
PRINT c
[ -30.60,
  6.40,
  10.50 ]
계속하려면 아무 키나 누르십시오 . . .
```

```
C:\WINDOWS\system32\cmd.exe
Enter input file <default=stdin>:
Enter output file <default=stdout>:
USE 10
VECTOR a 3
1.5 3.2 10.0
VECTOR b 3
-32.1 3.2 0.5
VECTOR c 3
0 0 0
ADD a b c
PRINT c
[ -30.60,
  6.40,
  10.50 ]
{ ELEMUL a c b }
PRINT b
[ -45.90,
  20.48,
  105.00 ]
```

```
C:\WINDOWS\system32\cmd.exe
PRINT c
[ -30.60,
  6.40,
  10.50 ]
{ ELEMUL a c b }
PRINT b
[ -45.90,
  20.48,
  105.00 ]
{ MATRIX d 3 2
  1.3 3.2
  1.1 3.5
  5.1 9.3
  MATRIX e 1 2
  0 0
  TRANSPOSE a
  MATMUL a d e
  PRINT e
  [ 56.47, 109.00 ] }
```

```
C:\WINDOWS\system32\cmd.exe

MATRIX d 3 2
1.3 3.2
1.1 3.5
5.1 9.3

MATRIX e 1 2
0 0

TRANPOSE a
MATMUL a d e
PRINT e
[[ 56.47, 109.00 1]

{ VECTORIZED
PRINTSIZE d }
'd' is a 6-dimensional vector.
{ PRINT d }
[ 1.30,
  3.20,
  1.10,
  3.50,
  5.10,
  9.30 1
```

```
C:\WINDOWS\system32\cmd.exe

{ VECTORIZED
PRINTSIZE d }
'd' is a 6-dimensional vector.
{ PRINT d }
[ 1.30,
  3.20,
  1.10,
  3.50,
  5.10,
  9.30 1

{ RESIZEMAT d 2 3 }
PRINTSIZE d
'd' is a 2 x 3 matrix.
{ PRINT d }
[[ 1.30, 3.20, 1.10 1,
[ 3.50, 5.10, 9.30 1]
```

```
C:\WINDOWS\system32\cmd.exe

PRINT d
[ 1.30,
  3.20,
  1.10,
  3.50,
  5.10,
  9.30 ]

RESIZEMAT d 2 3
PRINTSIZE d
'd' is a 2 X 3 matrix.
PRINT d
[[ 1.30, 3.20, 1.10 ],
 [ 3.50, 5.10, 9.30 ]]

END
계속하려면 아무 키나 누르십시오 . . .
```

(힌트)

1. 파일 입출력 & standard I/O: C 언어에서 모니터로 출력하는 standard output과 키보드로 입력받는 standard input 역시 파일로 처리한다. 따라서 입출력 파일 대신 stdin 혹은 stdout을 사용하면, 입력과 출력을 키보드와 모니터로 할 수 있다.  
A. 참고자료: <https://www.mksssoftware.com/docs/man5/stdio.5.asp>
2. 오류 메시지 출력: 오류 메시지는 stdout과 비슷한 stderr(오류 출력용 파일 스트림)에 출력한다.  
A. 참고자료: 위의 참고자료와 동일
3. 인터프리터를 만들기 위해서는 문자열 (string) 처리가 중요하다. 자주 사용 되는 문자열 처리가 있기 때문에 이와 관련된 많은 library가 이미 존재한다. <string.h>를 include하여 다음과 같은 문자열 관련 함수를 사용할 수 있다.  
A. strcmp: 문자열 비교 함수  
B. strlen: 문자열 길이 함수  
C. strtok: 문자열 분리 함수  
D. 참고자료: 다음 링크에서 string.h에 있는 함수의 목록과 간단한 사용방법을 확인 할 수 있다:  
[https://www.ibm.com/support/knowledgecenter/ko/ssw\\_ibm\\_i\\_73/rtrf/stringh.htm](https://www.ibm.com/support/knowledgecenter/ko/ssw_ibm_i_73/rtrf/stringh.htm)