# CMPE 230 Systems Programming

**Homework 1** (due April 5th)

( This project should be implemented in C.
It can be done in groups of at most 2 students)

In this project, you will implement a translator for a language called MatLang that will translate MatLang code to C language code. The C language code generated can then be compiled by a C compiler to produce an executable file.

MatLang language statements will be as follows:

1. Matlang programs will have two sections: The first will be a variable definitions section, followed by the executable statements section.
2. Variable definitions section will allow scalar, one-dimensional, two dimensional number variable definitions. For example
   ```
   scalar    x
   vector    y[4]
   matrix    z[3,4]
   ```
   Assume all array dimensions are given as integer constants. In the above example **`x`** is a scalar variable, **`y`** is a vector (i.e. matrix of size 4x1) and **`z`** is a 3x4 matrix. Assume that each line contains a single definition. A definition cannot be written on more than one line.
3. Executable statements will consist of one line statements and for loop compound statements. Note that no nested for loops are allowed.
4. One line statements are either assignment statements or print statements which print the value(s) of a scalar, vector, or a matrix variable or a separator.
5. A vector or a matrix variable can be assigned values in curly brackets. Note that such assignment should fit on line. For the vector `y[4]` and `z[3,4],` the following example assignment can be made:
   ```
   y = { 1 2 3 4 }
   z  = {1 2 3 4 5 3 2 2 2 1  0 1 }
   ```
   Note that array indices start with 1.
6. There are no if statements in the language.
7. As operations in expressions, you are required to implement only multiplication, addition and subtraction: **`*,+,-`** . These are binary operand operations. Unary minus operation is *not* supported. Note that these are to be interpreted as either matrix or scalar operations depending on the context (i.e. type of operands). When a scalar expression multiplies a matrix or a vector, its meaning is multiplication of each individual component of a matrix or vector.
8. A function **`tr(expr)`** is also available which transposes a scalar, vector or matrix.
9. A function **`sqrt(expr)`** is also available which takes square root of a scalar expression.
10. A function **`choose(expr1,expr2,expr3,expr4)`** which returns **`expr2`** if **`expr1`** is equal to 0, returns **`expr3`** if **`expr1`** is positive and returns **`expr4`** if **`expr1`** is negative. Note that **`expr1`**, **`expr2`**, **`expr3`** and **`expr4`** are expressions that evaluate to a scalar.
11. On a line, everything after the **#** sign are considered as comments.
12. For loop will have the following formats:
    ```
    for (id in expr1:expr2:expr3) {
        .....
        .....
    }
    ```
    Here, **`id`** is a variable, **`expr1`** is starting value of **`id`**, **`expr2`** is the bound on the value of **`id`** during the loop iteration and **`expr3`** is the added value to **`id`** at each iteration.

For loop can also have the following syntax:

```
for (id1,id2 in expr1:expr2:expr3, expr4:expr5: expr6) {
    .....
    .....
}
```

This will be equivalent to doubly nested loops in languages like C/Java. You can assume that the values of ids `id1,id2` and expressions `expr1,expr2,expr3,expr4,expr5` and `expr6` cannot be changed inside the for loop body. You can also assume that `expr1 < expr2` and `expr4 < expr5` and that `expr3` and `expr6` evaluate to a positive value.

13. `print(id)` statement, prints the value of variable `id`.
14. `printsep()` statement, prints a separator line "`----------`"
15. Please note that the C code generated must compute the MatLang outputs. You should not generate C code that just prints MatLang program outputs.
16. Assume that default value for variables is 0.

Some example programs in the MatLang language are given below. Note that MatLang language programs have .mat extension.

| ex1.mat | Output when compiled and executed |
|---|---|
| <pre># this program computes fibonacci<br># numbers<br><br># variable definitions<br>scalar i<br>scalar n<br>vector x[2]<br>vector y[2]<br>matrix A[2,2]<br>matrix B[2,2]<br><br># statements<br>n = 10<br>x = { 1 1 }<br>A = { 1 1 1 0 }<br>B = { 1 0 0 1 }<br>print(x)<br>for(i in 1:n:1) {<br>  B = A*B<br>  y = B*x<br>  print(y[1])<br>}</pre> | <pre>1<br>1<br>2<br>3<br>5<br>8<br>13<br>21<br>34<br>55<br>89<br>144</pre> |

| ex2.mat | Output when compiled[1] |
|---|---|
| <pre># variable definitions<br>vector z[3]<br>vector y[4]<br>matrix A[2,2]<br>matrix B[2,3]<br><br>z = A*B*y</pre> | <pre>Error (Line 6)</pre> |

[1] matrix dimensions in expression do not match.

ex3.mat

```
# simple pageranking
# algorithm

matrix   A[3,3]
matrix   T[1,1]
vector   x[3]
vector   y[3]
scalar   r
scalar   i

A = { 0.5  0 0.5 0 0 0.5 0.5 1 0 }
x = { 1 1 1 }
for(i in 1:10:1) {
  y = A*x
  T = tr(y-x)*(y-x)
  r = sqrt(T[1,1])
  print(r)
  x = y
}
printsep()
print(x)
```

Output when compiled and executed

```
0.707107
0.612372
0.467707
0.385276
0.309359
0.250975
0.202824
0.164156
0.132784
0.107430
------------
1.214844
0.624023
1.161133
```

ex4.mat

```
matrix   A[4,4]
matrix   T[1,1]
vector   x[4]
vector   xy2[4]
scalar   s

A = {0 1 2 3 4 5 6 7 8 9 1 1 1 2 3 4 }
x = {1 1 1 1 }
xy2 = { 2 1 3 1 }

T = tr(x)*A*xy2
s = T[1,1]
print(s)
```

Output when compiled and executed

```
94
```

ex5.mat

```
# count how many elements are
# greater than or equal to 4
matrix   A[4,4]
scalar   count
scalar   incr
scalar   i
scalar   j

A = {0 1 2 3 4 5 6 7 8 9 1 1 1 2 3 4}
count = 0
for (i,j in 1:4:1,1:4:1) {
    incr  = choose(A[i,j]-4,1,1,0)
    count = count + incr
}
print(count)
```

Output when compiled and executed

```
7
```

**Grading**

Your project will be graded according to the following criteria:

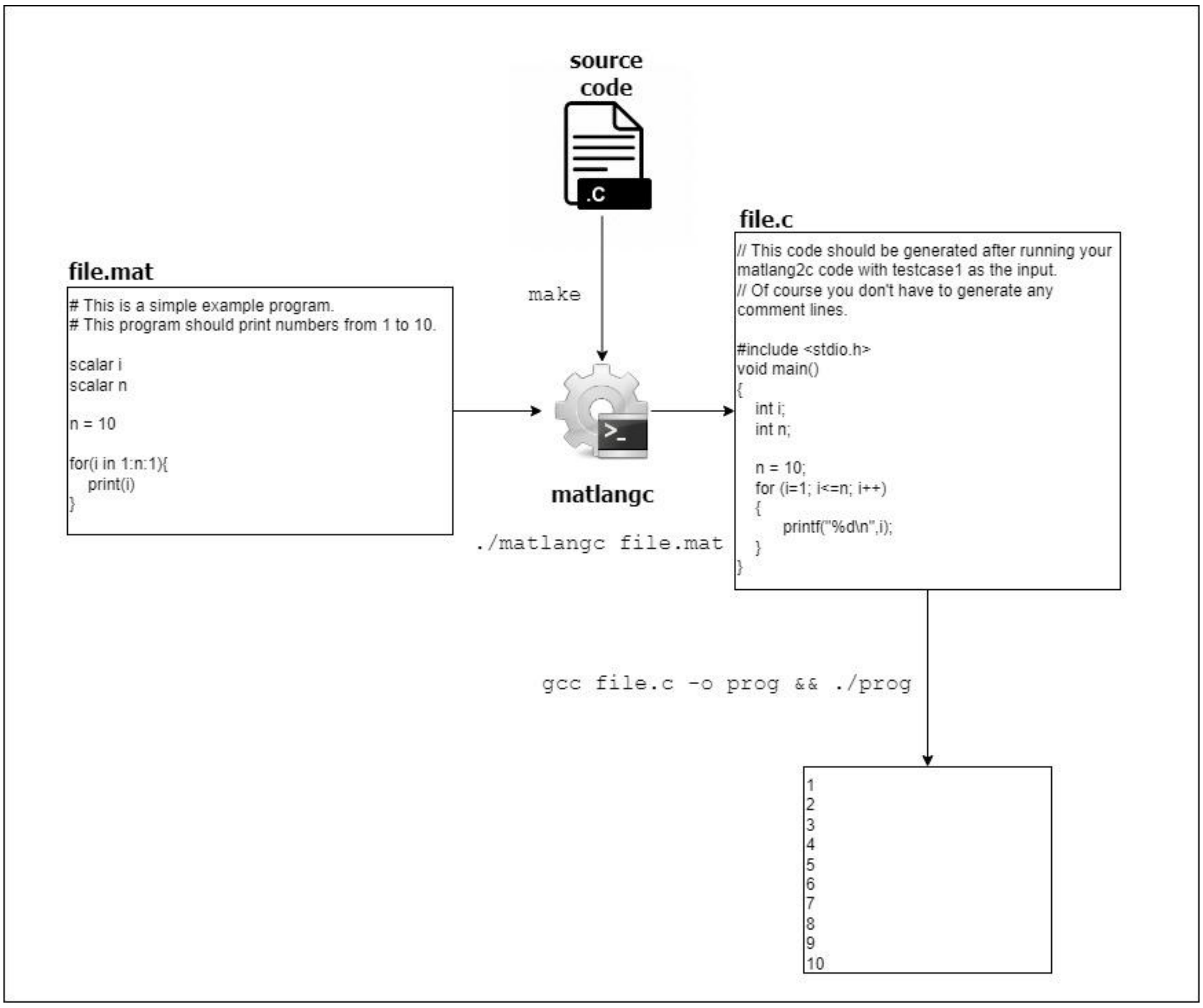
| | |
|---|---|
| Documentation (written document describing how you implemented your project) | 12% |
| Comments in your code | 8% |
| Implementation and tests | 80% |

**Testing**

Your code will be graded automatically in the virtual machine we provided for various test cases. Specifically, we will execute the following commands to check the output of your program.

| Step | Command | Explanation |
|---|---|---|
| **1** | `make` | Compiles your code and produces the executable, matlang2c |
| **2** | `./matlang2c file.mat` | Runs matlang2c on file.mat and produces C code in file.c |
| **3** | `gcc file.c -o prog && ./prog` | Compiles and executes C code |

Note that you must write your own Makefile that produces the executable named **matlang2c** when `make` command runs. The grading pipeline is depicted in the figure below with an example input.



file.mat

```
# This is a simple example program.
# This program should print numbers from 1 to 10.

scalar i
scalar n

n = 10

for(i in 1:n:1){
    print(i)
}
```

file.c

```
// This code should be generated after running your
matlang2c code with testcase1 as the input.
// Of course you don't have to generate any
comment lines.

#include <stdio.h>
void main()
{
   int i;
   int n;

   n = 10;
   for (i=1; i<=n; i++)
   {
       printf("%d\n",i);
   }
}
```

```
1
2
3
4
5
6
7
8
9
10
```

**Submissions**

The submissions will be through Moodle. Submit a zip file named with your student number(s) (e.g. 2020400XXX.zip or 2019400XXX_2020400XXX.zip). This zip should contain your source code(s), Makefile and documentation.

**Late Submission**

If the project is submitted late, the following penalties will be applied:

- 0 < hours late <= 24 : 25%
- 24 < hours late <= 48 : 50%
- hours late > 48 : 100%