

- Here is the change log for this assignment write-up. I will try to be descriptive in my log messages.
- You can also subscribe to this page and receive Emails when changes are made.

In this assignment will you will construct some core portions of a scanner and demonstrate your implementation's correctness. The name of your application must be `NFAMATCH` (case sensitive).

Input

`NFAMATCH` will accept two or more arguments:

Argument	Value
1	Path to the NFA definition file
2	Path to the optimized DFA transition table output file
3, 4, . . .	Tokens to test for matching against the NFA

`NFAMATCH` should:

1. read the NFA definition file,
2. convert the NFA to a DFA,
3. optimize the DFA (including dead state and unreachable state pruning),
4. output the optimized DFA transition table (T) to the path name of the command line second argument.
5. and finally test the remaining tokens on the command line to determine if they are an element of the DFA's **regular set**

If a token **is part of the DFA's regular set**, your `NFAMATCH` should emit an `(:M:)`, otherwise `NFAMATCH` should emit the first **character position** (`1, 2, 3, . . .`, *not an index!*) that fails the DFA. If the token is not long enough to match the DFA, emit **one more than the length of the token**. If the token is an empty string which does not match the DFA, then a `0` should be emitted. These last few "special cases" are consistent with reporting the character position of match failure.

NFA Definition File

An NFA definition file is line oriented, all fields are white space delimited. The first line contains three or more fields:

Field	Description	Format
-------	-------------	--------

1	Number of states in the NFA	Simple decimal positive integer
2	λ character	The special character representing a λ transition in file
3,4,...	NFA alphabet Σ	Whitespace separated simple printable ASCII characters in ascending ASCII order

The **order** of characters in the alphabet's definition is **important**. Your optimized DFA's transition table *must use the same ordering for its columns!*

Subsequent lines represent transitions between two states; these lines contain 3 or more fields. State **0** is implicitly the **starting state**.

Field	Description	Format
1	Normal (-) or accepting state + flag for the From field	Simple decimal non-negative integer
2	From state id	Simple decimal non-negative integer
3	To state id	Simple decimal non-negative integer
4,5,...	White space delimited transition characters from Σ or the λ character for this NFA	

There is only one transition line permitted between any two states. Empty lines in an NFA definition file should be ignored, there is no commentary symbols or syntax. All transition lines must have at least 3 fields.

Your code is **not expected** to detect flaws or inconsistencies in an NFA definition file. Your code should abort with an exit status of 1 if an NFA file cannot be opened or is devoid of data (all lines are empty lines). Otherwise you may assume the NFA file is good to go.

NFA Definition Example

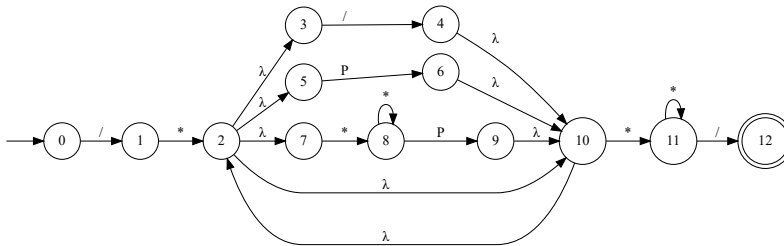
If we let P be $\Pi = \Sigma - \{ /* \}$, then this is an NFA definition file for the C/C++ block comment RE RE we've discussed in lecture:

```
13 # * / P
- 0 1 /
- 1 2 *
- 2 3 #
- 2 5 #
- 2 7 #
- 2 10 #
- 3 4 /
- 4 10 #
```

```

- 5 6 P
- 6 10 #
- 7 8 *
- 8 8 *
- 8 9 P
- 9 10 #
- 10 2 #
- 10 11 *
- 11 11 *
- 11 12 /
+ 12 12

```



Tokens

A token will be a sequence of non-whitespace containing characters from the NFA's alphabet (Σ). All the characters of a token must be matched in order for it to be considered a successful match against the DFA.

Output

You will report the following values from your application NFAMATCH on the process stdout (in this order, and according to the course submission requirements):

1. The match results (non-negative integers) for command line provided arguments 3, 4, ...

An example of a successful run of your NFAMATCH might look like this at a Linux console (the ... represents arbitrary console messages that will be ignored by the grader script):

```

$ ls
NFAMATCH  cblock.nfa
$ ./NFAMATCH cblock.nfa _ttable.dat  '/*PPPPPP*/'  '/*/**/'  ""
'/******/'  '/*P*/'  '/*PP'
...
OUTPUT :M:
...
OUTPUT 1
...
OUTPUT 0
...
OUTPUT :M:
...
OUTPUT :M:
...

```

```

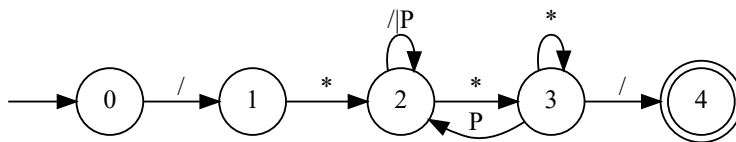
OUTPUT 5
...
$ cat _ttable.dat
- 0 E 1 E
- 1 2 E E
- 2 3 2 2
- 3 3 4 2
+ 4 E E E

```

The match results could also be written on one line:

```
OUTPUT :M: 1 0 :M: :M: 5
```


(And merely for completeness sake)



Examples

Data file tarballs of this C/C++ block comment and other examples (three underscored identifiers, floating points) that have been part of previous LGAs and we *may have* discussed in lecture are available.

grader.sh

I am providing to students the same tarball the grader will use for testing your NFAMATCH. is how to use it (a  video is posted here):

First, download this tarball to your Mines Linux account ("alamode" machines!) and unroll it in a temporary directory.

```

$ ls nfamatch-student.*
nfamatch-student.tar.bz2
$ mkdir ~/tmp
$ cd ~/tmp
$ tar xjf ../nfamatch-student.tar.bz2
:
$ ls
nfamatch/

```

Second, set the COMPGRADING environmental variable with:

```
$ source ~khellman/COMPGRADING/setup.sh ~khellman/COMPGRADING
```

Now go to the directory holding your NFAMATCH and execute the `grader.sh` script from the `nfamatch-student.tar.bz2` resource.

```
$ cd ~/compilers/nfaMatch
$ ls NFAMATCH
NFAMATCH
$ ~/tmp/nfamatch/grader.sh
:
:
:
```

You will need to read any messages from the script carefully, and hit ENTER several times throughout its course. This script checks for:

- a. missing data files
- b. truncated data files
- c. and the difference between your NFAMATCH results and expected results.

The latter test results are displayed on the terminal screen, but they whip by pretty quickly so you may need to scroll up and make sure you see them all.

When there is a discrepancy in expected results the script either:

- points you to a data file along side your NFAMATCH where more failure details are available,
- or all the details are displayed on screen.

Before the `grader.sh` terminates, it will show a summary of some specific rubric line results for the programming project.

The grader will use this very same script, along with some additional testing data to check your submitted work. If your NFAMATCH flies through without a hitch, you can likely suspect a good grade for the assignment.

In the `nfamatch-student.tar.bz2` tarball are also PDFs of initial `.nfa` files as well as their optimized DFAs, I find these much easier to "visualize" than the textual `.nfa` input files or the transition table output files. You might like them as well.

Submit Your Work

Partners

You may complete this assignment in a group of two or three other students from the course

(you may also work solo if you choose). Exams may have questions specific to course programming assignments — so be sure your group is truly working as a team and have a solid understanding of your submission's design and algorithms.

If you decide to complete this assignment as a group, you must tell your instructor one week before the assignment due date. Only one of you should submit the assignment for grading. You will all be able to see the grading result from your own logins.

Checking and Submitting Your Work


Here are some things to double check your submission against:

- i. Your archive does not raise errors when "unrolling" with the `${COMPGRADING}/explode-subm` script.

```
tmpdir@alamode $ ls
my-program.tar.bz2
tmpdir@alamode $ ${COMPGRADING}/explode-subm -P PROGRAM my-
program.tar.bz2 ./here
```

(Where PROGRAM is the required application name for this assignment.)

- ii. Your archive contains only the essential files, **don't provide unneeded files**
- iii. Don't provide files *that have been provided to you* (such as `cblock.nfa`); that would just be silly.
- iv. Make sure the `grader.sh` script for the assignment runs to completion.

When you are happy with your work, log into  the course website and submit your project archive file for grading.

Rubric

This work is worth 65 points.

Requirements	Points	Notes
Meets compilers course project requirements	10	
Executable named NFAMATCH --- case matters!	5	
Exit status of 1 when NFA definition file not found	5	
Exit status of 1 if NFA definition does not contain data	5	
DFAs pruned of dead states	5	
DFAs pruned of unreachable states	5	
DFAs optimized to minimal number of nodes	10	
Token matching	20	