Second try at Scanning & Parsing

COMP360

“Failing doesn’t make us a failure. But not trying to do better, to be better, does make us fools.”

Wes Moore

Parsing Program

- The Parsing programming assignment is due by midnight today, Friday, February 21, 2020
- Two example snowflake programs are provided as sample input
- Modify an example to create an error to see if it is properly caught

Breaking the Program into Stages

- A compiler can be a large complex program
- The program can be divided into manageable steps
- Each step does a specific task
- Computer Science theory provides a solution to many of the challenges in writing a compiler

Output of Compiler Stages

Step	Output
• Lexical Analysis (scanning)	List of tokens
• Syntactic Analysis (parsing)	Parse tree
• Semantic Analysis	Intermediate code
• Optimization	Better Intermediate code
• Code Generation	Machine language

Lexical Analysis

- Lexical Analysis or scanner reads the source code
- It removes all comments and white space
- The output of the scanner is a stream of tokens
- Tokens can be words, symbols or character strings

Tokens Only

- The scanner ONLY makes a list of tokens
- It does **NOT** check syntax, such as matching brackets

Classical Languages Recognized

Machine	Language
Deterministic Finite State Automata (DFA)	Regular expressions
Push Down Automata (PDA)	Context free grammars, $a^n b^n$
Bounded Turing Machine	$a^n b^n c^n$
Turing Machine (TM)	Anything that can be recognized

Exam Discarded

- The first COMP360 exam held on Wednesday, February 19, 2020 will not be graded

Exam 1 Second Attempt

- Another version of the first COMP360 exam has been posted on Blackboard
- This is a take-home exam
- The exam may be done individually or in teams of up to four students
- Put all names on the exam and submit it only once
- Due by midnight on Wednesday, February 26, 2020

Do you agree with this exam plan?

A. Yes
B. No
C. I have a better idea
D. Just give me a drop form

```
public class Token {         // Token for Snowflake or others
    private String     value = "";
    private int          type;
    private final int  lineNumber, column;
public Token(char symbol1, int line, int col) { // constructor
      value          = value + symbol1;
      lineNumber = line;
      column       = col;
      if (Character.isLetter(symbol1)) {
         type = VARIABLE;
      } else if (symbol1 == "\'") {
         type = CONSTANT;
         value = "";   // remove quote
      } else {
         type = PUNCTUATION;       }
  }
```

Adding a character to the end of the token

```
public void add2Token(char nextSymbol) {
    value = value + nextSymbol;
    if (type == KEYWORD) {        // add letter changes a keyword
        type = VARIABLE;
    } else if (type == VARIABLE) {     // check if this became a keyword
        for (String key : KEYWORDS) {   // search list of keywords
            if (value.equalsIgnoreCase(key)) {
                type = KEYWORD;
            }
        }
    }
}
```

Useful Token methods

It may help if your Token class has methods

- isVar() – true if this is a variable
- isVarConst() – true if this is a variable or a constant
- getValue() – returns the Token string
- toString() – use to display the tokens

"Simplicity makes me happy."

Alicia Keys

- A scanner can be implemented with a finite state automata (FSA)
- Do **NOT** attempt to create one in an ad hoc manner
- Implement an FSA
  - Supported by theory
  - Easy to program

Languages and Machines

Language	Machine
Regular Expressions	Deterministic Finite State Automata (DFA)
Context Free	Push Down Automata (PDA)
Context Sensitive	Bounded Turing Machine
Recursively Enumerable	Turing Machine

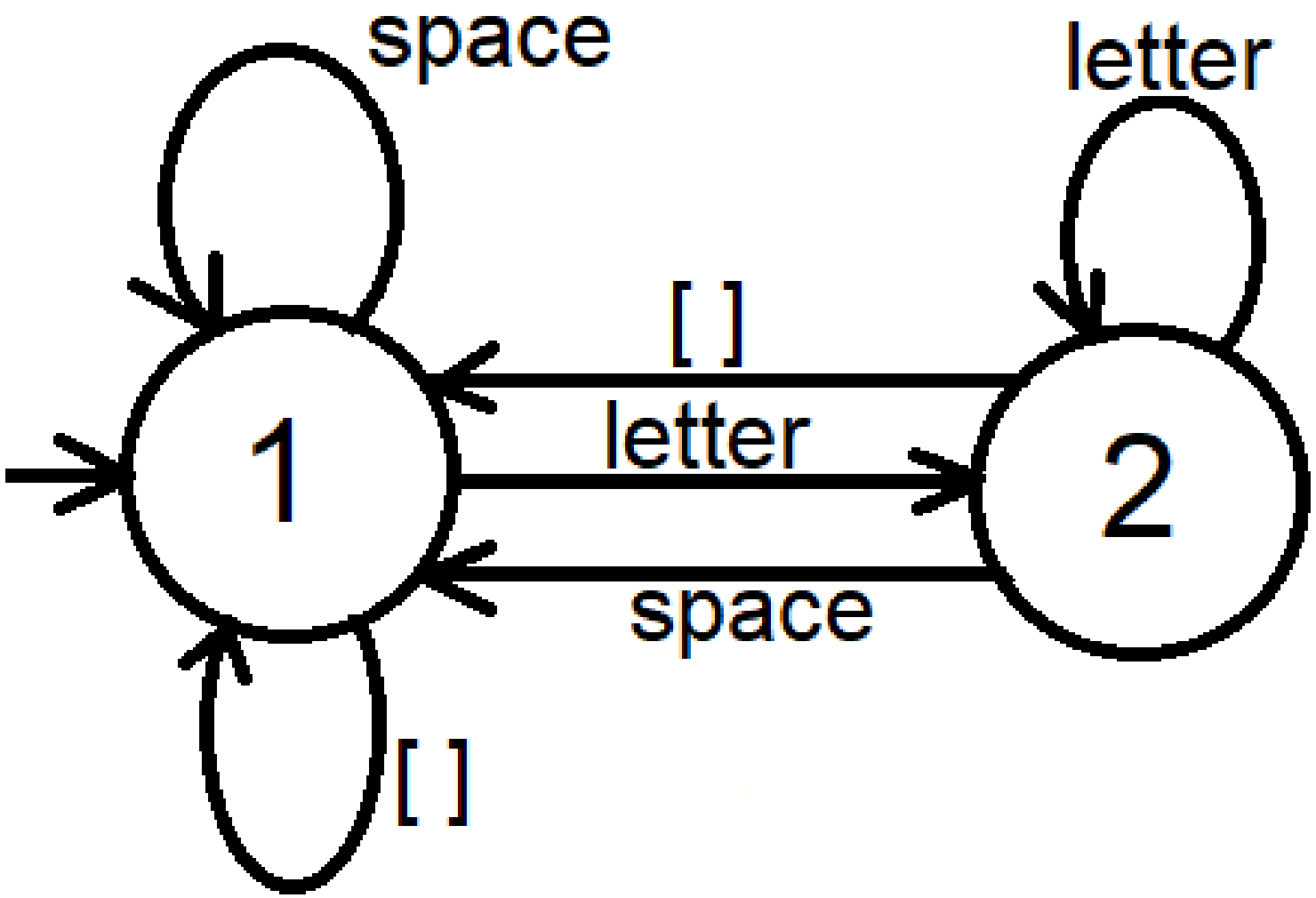
Steps in Creating a Scanner

1. Create an FSA graph with circles and arrow
2. Translate the graph into a two dimensional array
3. Add actions to be taken at each transition
4. Use the provided program to implement

Committee Language

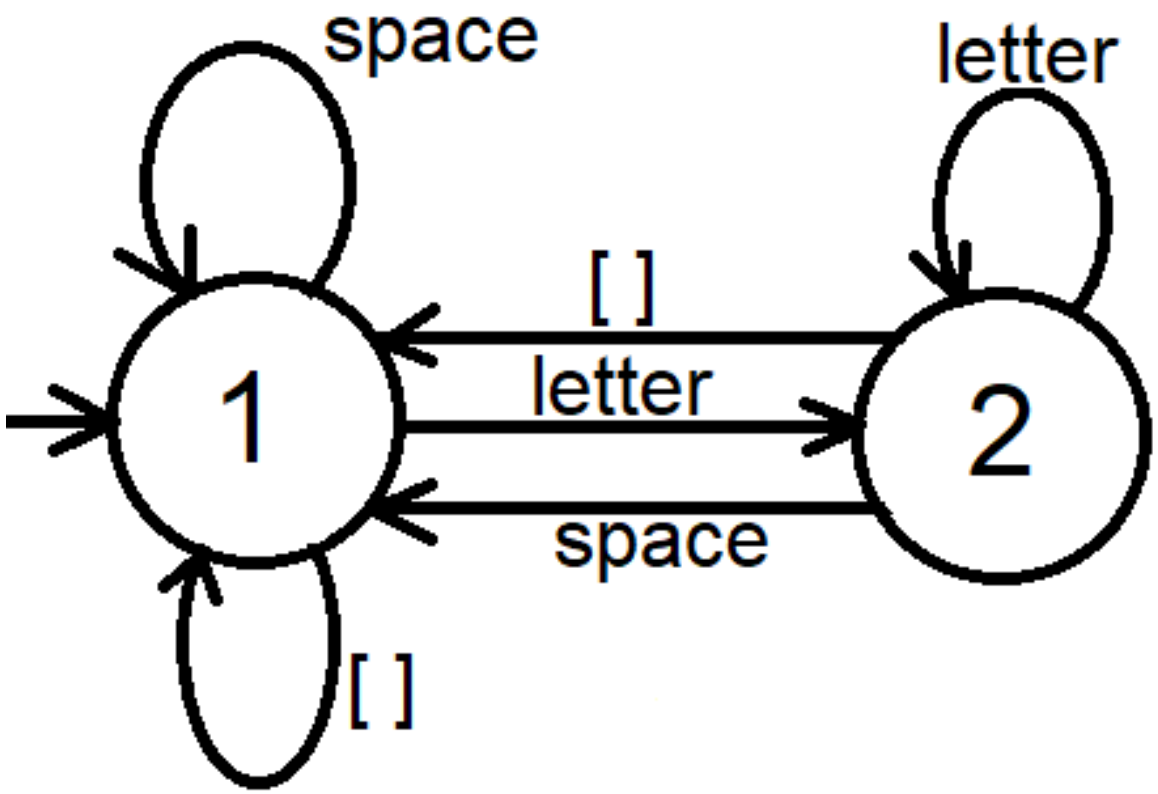
- A committee or subcommittee is enclosed in [brackets]
- After the left bracket is the name of the leader
- After the leader there may be the names of other members or subcommittees
  - [Fred]
  - [Sadie Betsy [Mark [Cindy Jake]] Joe]
  - [Dorothy Ben]

FSA for the Committee Language



Creating the State Table

input symbol	State 1	State 2
letter	2	2
[ ]	1	1
space	1	1
other	0	0



Converting the State Table to Java

```
int[][] stateTable = {
    /* state 0,  1,  2 */
            {  0,  2,  2 }, // 0 letter
            {  0,  1,  1 }, // 1 [bracket]
            {  0,  1,  1 }, // 2 space
            { 0,   0,  0 }};// 3 0other
```

Convert Input Character to Index

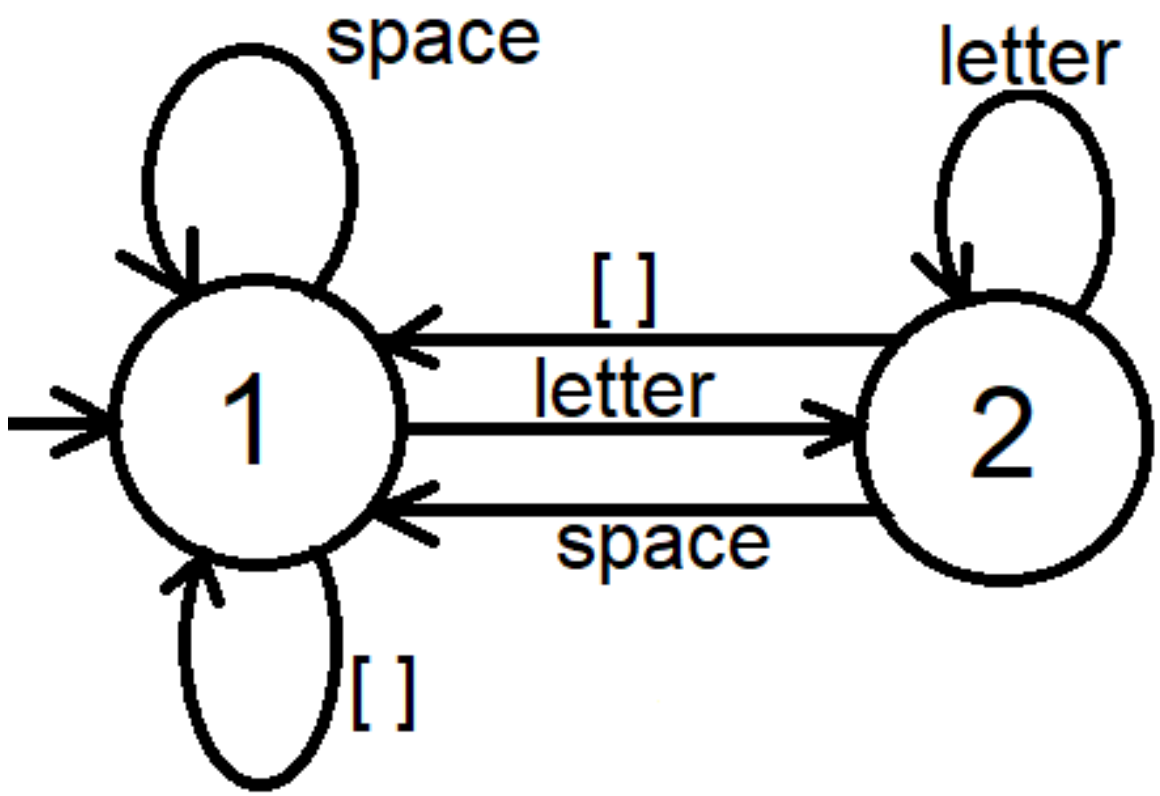
```
inChar = read();          // read a character
if (Character.isLetter(inChar)) {
    index = 0;
} else if (inChar == '[' || inChar == ']') {
    index = 1;
} else if (inChar == ' ' || inChar == '\t') {
    index = 2;
} else {
    index = 3;
}
```

Actions

- A Mealy machine takes an action on each input character or transition
- The purpose of the actions is to create tokens of the input words and punctuation

Creating the Action Table

input character	State 1	State 2
letter	1	2
[ ]	1	1
space	0	0
other	0	0



Actions

0 Do nothing

1 Create a token with the input character

2 Append input character to last token

FSA Program

```
state = 0
while not end of file
        inChar = read next input character
        index = number based on type of character
        action = actionTable[index][state ]
        state   = stateTable[index][state ]
        if state = 0 then error
        switch ( action )
                case 1: new Token(symbol); put on list
                case 2 : append symbol to last token
```

Syntactic Analysis

- Syntactic Analysis or parsing reads the stream of tokens created by the scanner
- It checks that the language syntax is correct
- The parser can be implemented by a context free grammar stack machine
- The output of the Syntactic Analyzer is a parse tree
- Our parsers just indicate if the syntax is good or the first error found

Steps in creating a Parser

1. Create a BNF for the language
2. Write a recursive descent method for each rule
3. Integrate with scanner

BNF Fundamentals

- Terminal symbols are tokens or symbols in the language. They come from the lexical scanner
- Nonterminal symbols are “variables” that represent patterns of terminals and nonterminal symbols
- A BNF consists of a set of **productions** or **rules**
- A language description is called a **grammar**

BNF Structure

- Nonterminal symbols are sometimes enclosed in brackets to differentiate them from terminal symbols
- I will use **bold font** for terminal symbols and no brackets
- Productions are of the form:

  nonterminal → nonterminals or terminals

- Multiple definitions are separated by | meaning OR

  whatever → this | that

Committee Language

- A committee or subcommittee is enclosed in [brackets]
- After the left bracket is the name of the leader
- After the leader there may be the names of other members or subcommittees
  - [Fred]
  - [Sadie Betsy [Mark [Cindy Jake]] Joe]
  - [Dorothy Ben]

BNF for Committee Language

cmte → **[name]**
| **[name** memberlist **]**

memberlist → member
| member memberlist

member → **name**
| cmte

- The terminal symbols are **name, [** and **]**

Example BNF

1. gerbil → mouse | mouse **$** gerbil
2. mouse → hamster | hamster ***** mouse
3. hamster → ***name*** | **#** gerbil

Derivation of Strings

mouse $ gerbil	rule 2
hamster $ gerbil	rule 3
trout $ gerbil	rule 5
trout $ mouse	rule 1
trout $ hamster	rule 3
trout $ cod	rule 5

1. gerbil → mouse
2. | mouse **$** gerbil
3. mouse → hamster
4. | hamster * mouse
5. hamster → ***name***
6. | **#** gerbil

What rules derive bass * shark?

mouse

hamster * mouse

bass * mouse

bass * hamster

bass * shark

1. gerbil → mouse
2. | mouse **$** gerbil
3. mouse → hamster
4. | hamster * mouse
5. hamster → ***name***
6. | **#** gerbil

Derivation of bass * shark?

mouse	rule 1
hamster * mouse	rule 4
bass * mouse	rule 5
bass * hamster	rule 3
bass * shark	rule 5

1. gerbil → mouse
2. | mouse **$** gerbil
3. mouse → hamster
4. | hamster * mouse
5. hamster → ***name***
6. | **#** gerbil

Which strings are in the language?

1. gerbil → mouse | mouse **$** gerbil
2. mouse → hamster | hamster ***** mouse
3. hamster→ ***name*** | **#** gerbil

A. bass * trout
B. shark $ cod $ trout
C. hamster $ mouse
D. # cod
E. All the above

Example BNF

- Consider a language that has a string of A's and B's
- Strings in the language may start with either an A or B
- All A's must be separated by B's
- There can be multiple B's in a row

valid strings:

A or A B A or B B A or A B B A

BNF:

$$T \rightarrow \mathbf{A} \mid \mathbf{A}\ M \mid M$$

$$M \rightarrow \mathbf{B} \mid \mathbf{B}\ T$$

Recursive Descent

- Recursive descent is a top down parsing method
- It is relatively easy algorithm to implement
- Functions or methods are written to represent each BNF production
- Recursive descent does not use parse tables

BNF to Recursive Descent

- Each production becomes a separate method
- Usually the method name matches the name of the non-terminal
- For each non-terminal on the right side, the method for the non-terminal is called
- For terminal symbols, the program checks if the next token is the proper symbol

Input tokens

- The input to the parser is a list of tokens from the lexical scanner (called **`tokens`** in the examples)
- I used an integer (called **`curtok`**) to keep track of the current location in the input that is being parsed
- When a method matches a BNF production, it increments **`curtok`** to move past that input token

Possible Snowflake BNF

1. SF → parmstmt code ret
2. parmstmt → **parm** varlist **;**
3. varlist → **var**
   | **var** varlist
4. varclist → **varconst**
   | **varconst** varclist
5. ret → **return var ;**
6. pattern → **varconst**
   | **varconst | pattern**

Possible Snowflake BNF (cont)

7. code → line
   | line code
8. line → assign
   | loop
9. assign → **var** **=** varclist **;**
   | **var** pattern **=** varclist **;**
10. loop → **while** **varconst** pattern **{** code **}**

Recursive Descent example for Snowflake

4. varclist → **varconst**
| **varconst** varclist

```
boolean varclist() {
        if ( !tokens[curtok].isVarConst() ) return false;
        curtok++;          // move past variable
        varclist();        // can be true or false
        return true;
}
```

Write the Recursive Descent parser method

8. line → assign
| loop

Recursive Descent parser method for

8. line → assign
| loop

```
boolean line() {
    if ( !assign() ) {
        if ( !loop() ) {
            return false;
        }
    }
    return true;
}
```

Parsing Program

- The Parsing programming assignment is due by midnight today, Friday, February 21, 2020
- Two example snowflake programs are provided as sample input
- Modify an example to create an error to see if it is properly caught

Exam Discarded

- The first COMP360 exam held on Wednesday, February 19, 2020 will not be graded

Exam 1 Second Attempt

- Another version of the first COMP360 exam has been posted on Blackboard
- This is a take-home exam
- The exam may be done individually or in teams of up to four students
- Put all names on the exam and submit it only once
- Due by midnight on Wednesday, February 26, 2020