Introduction to Compiler Construction

Scanning: Handcrafting a Scanner

Outline

State Diagrams

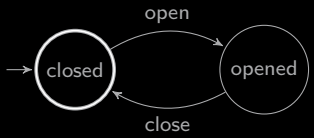
State Diagrams

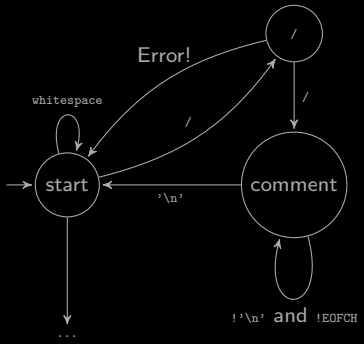
State diagrams can be used to graphically represent finite-state machines such as scanners

State Diagrams

State diagrams can be used to graphically represent finite-state machines such as scanners

Example (state diagram for a door that can be opened or closed)



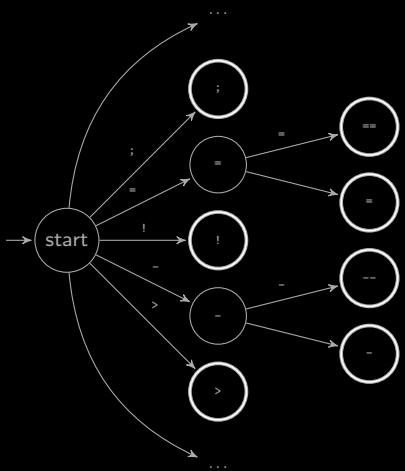


whitespace: ' ' or '\t' or '\n' or '\f'

Handcrafted Scanner for j-- · Whitespaces and Single-line Comments

</> Scanner.java

```
1     public TokenInfo getNextToken() {
2         StringBuilder buffer;
3         boolean moreWhiteSpace = true;
4         while (moreWhiteSpace) {
5             while (isWhitespace(ch)) {
6                 nextCh();
7             }
8             if (ch == '/') {
9                 nextCh();
10                if (ch == '/') {
11                    while (ch != '\n' && ch != EOFCH) {
12                        nextCh();
13                    }
14                } else {
15                    reportScannerError("operator / is not supported in j--");
16                }
17            } else {
18                moreWhiteSpace = false;
19            }
20        }
21        ...
22    }
```

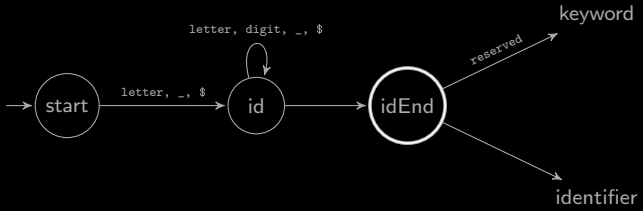

...
;
=
!
-
>
start
;
=
!
-
>
=
==
=
--
-
...

Handcrafted Scanner for j-- · Separators and Operators

Scanner.java

```
    public TokenInfo getNextToken() {
        ...
        switch (ch) {
            ...
            case ';':
                nextCh();
                return new TokenInfo(SEMI, line);
            case '=':
                nextCh();
                if (ch == '=') {
                    nextCh();
                    return new TokenInfo(EQUAL, line);
                } else {
                    return new TokenInfo(ASSIGN, line);
                }
            case '!':
                nextCh();
                return new TokenInfo(LNOT, line);
            case '-':
                nextCh();
                if (ch == '-') {
                    nextCh();
                    return new TokenInfo(DEC, line);
                } else {
                    return new TokenInfo(MINUS, line);
                }
            case '>':
                nextCh();
                return new TokenInfo(GT, line);
            ...
        }
    }
```

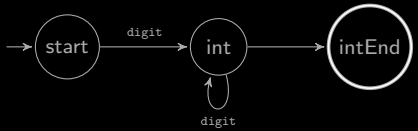
Handcrafted Scanner for j-- · Identifiers and Keywords



Handcrafted Scanner for j-- · Identifiers and Keywords

</> Scanner.java

```
1     public Scanner(String fileName) throws FileNotFoundException {
2         ...
3         reserved = new Hashtable<String, Integer>();
4         reserved.put("abstract", ABSTRACT);
5         reserved.put("boolean", BOOLEAN);
6         ...
7         reserved.put("while", WHILE);
8         ...
9     }
10
11    public TokenInfo getNextToken() {
12        ...
13        switch(ch) {
14            ...
15            default:
16                if (isIdentifierStart(ch)) {
17                    buffer = new StringBuilder();
18                    while (isIdentifierPart(ch)) {
19                        buffer.append(ch);
20                        nextCh();
21                    }
22                    String identifier = buffer.toString();
23                    if (reserved.containsKey(identifier)) {
24                        return new TokenInfo(reserved.get(identifier), line);
25                    } else {
26                        return new TokenInfo(IDENTIFIER, identifier, line);
27                    }
28                    ...
29                }
30        }
31    }
```


start
digit
int
intEnd
digit

Handcrafted Scanner for j-- · Integer Literals

</> Scanner.java

```
1     public TokenInfo getNextToken() {
2         ...
3         switch(ch) {
4             ...
5             case '0':
6             case '1':
7             ...
8             case '9':
9                 buffer = new StringBuilder();
10                while (isDigit(ch)) {
11                    buffer.append(ch);
12                    nextCh();
13                }
14                return new TokenInfo(INT_LITERAL, buffer.toString(), line);
15            ...
16        }
17    }
```