Plan for Today

How do you implement DFAs?

- For recognizing strings in a language
- For recognizing tokens in a string

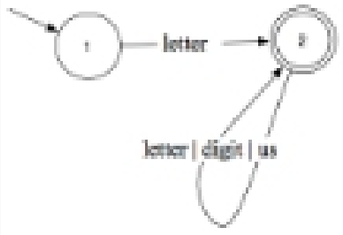
Context Free Grammars

- What is it?
- Derivations
- Parse trees
- Specifying them in SableCC

DFAs for Recognizing Language and Tokens

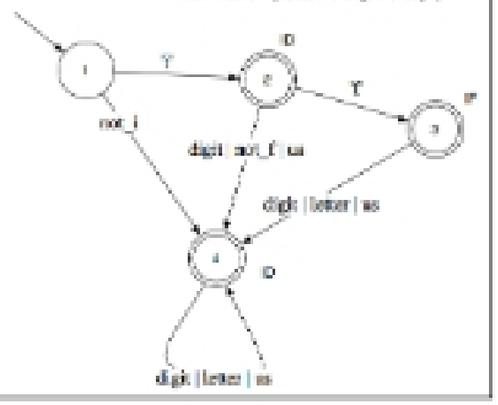
```
digit  = ['0'..'9'];
letter = ['a'..'z'] | ['A'..'Z'];
us = '_';

id = letter (letter | digit | us)*;
```




```
digit  = ['0'..'9'];
letter = ['a'..'z'] | ['A'..'Z'];
not_i  = [letter - 'i'];
not_f  = [letter - 'f'];
us = '_';

Tokens
  'if'    = 'if';
  id = letter (letter | digit | us)*;
```



Implementing DFAs

```
state = 1;

char c = read();
while ( !error && c !eof ) {

    switch ( state ) {
        case 1:
            switch ( c ) {
                case letter:
                    state = 2;
                    break;
                default:
                    error = true;
            }
        case 2:
            switch ( c ) {
                case letter:
                case digit:
                case us:
                    state = 2;
                    break;
                default:
                    error = true;
            }
    }

    char c = read();
}

if (!error && state==2) accept
else reject
```


Implementing a DFA that recognizes tokens

```
(token,string) lex() {
    char buffer[SIZEOF];

    if (firststrim) {
        state = 1;
        lastFinal = -1;
        lastState = -1;
        startState = 0;
        current = 0;
        buffer = fillIn input FILE;
        c = buffer[current];
    }

    while ( current<SIZEOF && !error && c!= eof ) {
        switch ( state ) {
            case 1:
                switch ( c ) {
                    case 'i':
                        state = 2;  lastFinal = 2;  lastState = current
                    case not_i:
                        state = 4;  lastFinal = 4;  lastState = current
                        break;
                    default:
                        error = true;
                }
            case 2:
                switch ( c ) {
                    case 'f':
                        state = 3;  lastFinal = 3;  lastState = current
                    case not_f:
                    case digit:
                    case us:
                        state = 4;  lastFinal = 4;  lastState = current
                        break;
                    default:
                        error = true;
                }
```

Token Impl cont.

```
case 3:
case 4:
    switch ( c ) {
        case letter:
        case sub_f:
        case digit:
        case us:
            state = 4;  lastFinal = 4;  lastIndex = currIndex;
            break;
        default:
            error = true;
    }
}
if (error) {
    if (lastFinal!=-1) {
        token = stateToken[lastFinal];
        string = buffer[startIndex..lastIndex];
        lastFinal = -1;
        currIndex = lastIndex;
        startIndex = currIndex+1;
        c = buff[currIndex++];
        error = false;
        return (token,string);
    } else {
        return error;
    }
}
c = buffer[currIndex++];
}
if (!error && state== (2 | 3 | 4)) {
    token = stateToken[lastFinal];
    string = buffer[startIndex..lastIndex];
    return (token,string);
} else return error;
}
```

CS453 Lecture

Parse Tree Example

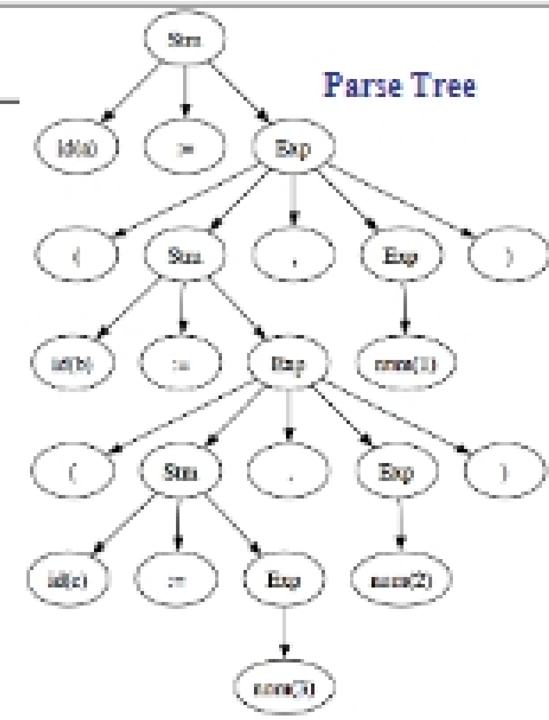
Grammar

Stm --> id := Exp
Exp --> num
Exp --> (Stm, Exp)

String

a := (b := (c := 3, 2), 1)

Parse Tree



CS453 Lecture

Specifying Grammars with SableCC

SableCC example input file:

```
Package minijava;

Helpers
  ...
Tokens
  ...
Productions
    stm =
        exp_list
      ;

    exp =
        {plus_rule} exp plus term
      | {term_rule} term
      ;
```

```
term =
        {mult_rule} term mult factor
      | {fact_rule} factor
      ;

factor =
        {id_rule} id
      ;

exp_list =
        exp exp_rest*
      ;

exp_rest =
        comma exp
      ;
```

CS453 Lecture   Intro to Parsing   3