

## 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

CS453 Lecture

Intro to Parsing

2

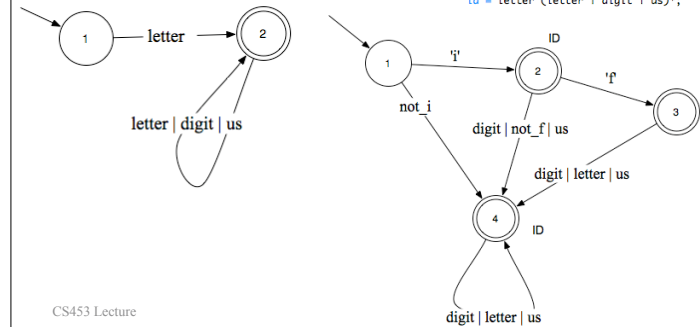
## DFAs for Recognizing Language and Tokens

```
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)*;
```



CS453 Lecture

## Implementing DFAs

```
state = 1;
char c = read();
while ( !error && c != eol ) {
    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
```

Intro to Parsing

4

## Implementing a DFA that recognizes tokens

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

    if (firsttime) {
        state = 1;
        lastfinal = -1;
        lastindex = -1;
        startindex = 0;
        currindex = 0;
        buffer = whole input file;
        c = buffer[currindex];
    }

    while ( currindex!=MAXBUF && !error && c!= eol ) {
        switch ( state ) {
            case 1:
                switch ( c ) {
                    case 'i':
                        state = 2; lastfinal = 2; lastindex = currindex;
                    case not_i:
                        state = 4; lastfinal = 4; lastindex = currindex;
                        break;
                    default:
                        error = true;
                }
            case 2:
                switch ( c ) {
                    case 'f':
                        state = 3; lastfinal = 3; lastindex = currindex;
                    case not_f:
                    case digit:
                    case us:
                        state = 4; lastfinal = 4; lastindex = currindex;
                        break;
                    default:
                        error = true;
                }
        }
        currindex++;
        c = buffer[currindex];
    }
}
```

CS453 Lecture

Intro to Parsing

5

### Token Impl cont.

```

case 3:
case 4:
  switch ( c ) {
  case letter:
  case not_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+1];
    error = false;
    return (token, string);
  } else {
    return error;
  }
}
c = buffer[currindex+1];
}
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

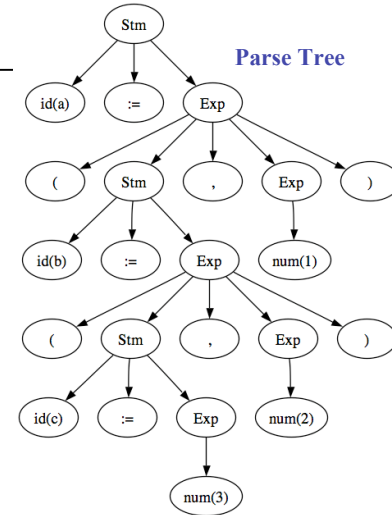
Stm --> id := Exp

Exp --> num

Exp --> ( Stm, Exp )

#### String

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



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

8