Parsing Input

Parsing Input

- Formatting output is not too difficult
  - use printf with format strings
  - formatting numbers and strings
- parsing input is much more challenging
  - must deal with erroneous input
  - must break apart strings into pieces
    - pieces are often called words lexemes
    - mapped to tokens by a program

Steps

- Decide on format of input
    - e.g., bash commands
- Describe input using regular expressions
    - ALPHA [a-zA-Z_]
    - NUMERIC [0-9]
    - WORD {ALPHA}({ALPHA}|{NUMERIC})*
    - OPERATOR [<>&|]
- Make a finite state machine to recognize the language
    - Acceptors and recognizers

Example

- ls foo<in|more>out

```
ls
foo
<
in
|
more
>
out
```

Code FSM using while and switch

```
char lexeme[BUFSIZ]; /* could put text here */
int getToken()
{
        int inch;
        while ((inch = getchar()) != EOF)
        switch (inch)
        {
                …cases on next 2 slides
        }
}
```

```
case ‘ ’:
case ‘\t’:
case ‘\n’:
        break; /* skip white space */
case ‘&’:
case ‘|’:
case ‘<’:
case ‘>’:
        return inch; /* Operators: use the character for the
token */
```

```
 /* #include <ctype.h> */
default:
        if ( isdigit(inch) )
                return parseNumber();
        else if ( isalpha(inch) )
        return parseWord();
    parse_error(“Illegal character”, inch);
```

```
int parseNumber()
{
        while (isdigit(peekc())
                getchar();
        return Number;
}
int parseWord()
{
        while (isalnum(peekc())
                getchar();
        return Word;
```

Errors go to stderr

```
void parse_error(char *msg, char ch)
{
        fprintf(stderr, “ERROR: %s: %c\n”, msg, ch);
}
```