CS434
Compiler Construction

Lecture 3

Spring 2005

Department of Computer Science
University of Alabama
Joel Jones

Outline: Parsing Without Parsing

- Announcement:
  - All future class meetings will be in 370 Bidgood
- Problem statement
- Simple text processing
- Text processing in "real" languages

Problem Statement

- When do we need to manipulate text?
  - Changing data from one format to another that another program expects
- What is the easiest way to do this transformation?
  - It depends ...

Little or Scripting Languages

- The easiest way to do text transforms in many circumstances is to use some kind of scripting language
  - Our examples in this lecture will use AWK
  - + It is easy to read and widely available
  - - Nobody uses it any more

awk

- `awk [options]'program' filenames ...`
- Like sed, but program is different:

```
pattern { action }
pattern { action }
...
```

- awk reads input in filenames one line at a time & when pattern matches, executes corresponding action
- Patterns
  - Regular expressions
  - C-like expressions

awk (cont.)

- Pattern or action is optional
  - Pattern missing—perform action on every line
  - Action missing—print every line matching pattern
- Simple action
  - print—without argument prints current line

Pair Up: Write an awk command line that does the same thing as:
`cat filenames …`

- `awk '{ print }' filenames …`

Pair Up: Write an awk command line that does the same thing as:
`grep re file`

- `awk '/re/' file`

awk (cont.)

- Variables
  - $0—entire line, $1-$NF—fields of line
    - E.g. `awk '{ print $2 }' textFile` prints 2nd field of every line of textFile
    - E.g. `who | awk '{ print $5, $1 }' | sort` prints name and login sorted by time
  - `NF` is number of fields on current line
  - `NR` is number of records (lines) read so far
- Options
  - `-Fchar`—sets `char` as the field separator

Pair Up: Write an awk command line that prints user names out of /etc/passwd, where the user name is the first field and fields are colon separated.

awk (cont.)

- `awk -F: '{ print $1 }' /etc/passwd`
  - N.B. most unix systems don't store users in /etc/passwd anymore
- Field breaking
  - Default is on space and tab and multiple contiguous white space counts as a single white space and leading separators are discarded
  - Setting separator causes leading separators to be counted

awk (cont.)

- More on patterns
  - Print user name of people who have no password

    `$2 == ""` 2nd field is empty

    `$2 ~ /^$/` 2nd field matches empty string

    `$2 !~ /./` 2nd field doesn't match any character

    `length($2) == 0` Length of 2nd field is zero

Pair Up: Write an awk command line that prints lines of input that have an odd number of fields.

- `awk 'NF % 2 != 0' files`

Pair Up: Write an awk command line that is the shortest equivalent of cat.

```
awk '/^/' files
```

What kind of languages can we parse using awk?

Pair Up: Can we parse context free grammars?

- Record Languages: languages that don't have a "parenthesized" structure
- Types of record languages:
  - key-value pairs
  - delimiter separated values
  - stanza formatted record

Key-value Pairs

- Format:
  - *key delimiter value new-line ...*
- Example
- transform

```
a b
c d
```

into

```
char *a;
char *c;
a="b";
c="d";
```

- Parsing

```
    { keyValueMap[$1] = $2; cnt++ } /* match every line */
END {
      for (i in keyValueMap) { print "char *" i ";" }
      for (i in keyValueMap) {
          print i "=" "\"" keyValueMap[i] "\";"
      }
    }
```

Delimiter Separated Values

Format:

value delimiter value delimiter ... new-line ...

Example

Parsing

transform

```
Jones, Joel
Jay, Trevor
Dunnavant, Crutcher
```

into

```
1. Joel Jones
2. Trevor Jay
3. Crutcher Dunnavant
```

```
BEGIN { cnt = 0 }
      { firstName[cnt] = $2; lastName[cnt] = $1; cnt ++ }
END   { for (i = 0; i < cnt; i++) {
            print i+1 ". " firstName[i] " " lastName[i]
        }
      }
```

awk -F,

Stanza Formatted Records

Format:

key delimiter value new-line
key delimiter value new-line
new-line
key delimiter value new-line ...

Emitting one record at a time:

```
$1 == "lastName:" { lastName = substr($0, index($0, $2)) }
$1 == "firstName:" { firstName = substr($0, index($0, $2)) }
/^$/ { emitRecord() } # empty line
function emitRecord() {
    if (lastName != "") {
        printf("%s", lastName)
        if (firstName != "") printf(", ");
    }
    printf("%s\n", firstName)
    lastName = ""; firstName = "";
}
END { emitRecord() }
```

Stanza Formatted Records

Format:

key delimiter value new-line
key delimiter value new-line
new-line
key delimiter value new-line ...

Emitting all records at the end:

```
BEGIN { id = 0 }
$1 == "lastName:" { lastName[id] = substr($0, index($0, $2))
}
$1 == "firstName:" { firstName[id] = substr($0,
index($0, $2)) }
/^$/ { id++ } # empty line
END { for (i = 0; i <= id; i++) {
          if (lastName[i] != "") {
              printf("%s", lastName[i])
              if (firstName[i] != "") printf(", ");
          }
          printf("%s\n", firstName[i])
      }
    }
```

Parsing With a Library

- See handouts, java.io.StreamTokenizer