NJIT

New Jersey's Science & Technology University

THE EDGE IN KNOWLEDGE

CS 280
Programming Language Concepts

About Assignment 2

Notes for Assignment 2

- Be sure to read and understand the assignment!
- Decide on the information that you need to keep track of to do the assignment
- What algorithm should you use?
- How should you save the data?

Assignment 2 pieces

- A header file will be given
- You will implement the lexical analyzer function in one source file
- You will implement a test main program in another source file

- Vocareum will be set up to compile everything together

- The lexical analyzer function will have the following calling signature:

```
Token getNextToken(istream *in, int *linenumber);
```

- The first argument to getNextToken is a pointer to an istream that the function should read from. The second argument to getNextToken is a pointer to an integer that contains the current line number. getNextToken will update this integer every time it reads a new line.
- getNextToken returns a Token. A Token is a class that contains a TokenType, a string for the lexeme, and the line number that the token was found on.
- A header file, tokens.h, will be provided for you. You MUST use the provided header file. You may NOT change it.

tokens.h

- Definition for all of the possible token types
- Class definition
- Some function prototypes

```
enum TokenType {
                    // keywords
        PRINT,
        IF,
        THEN,
        TRUE,
        FALSE,

                    // an identifier
        IDENT,

                    // an integer and string constant
        ICONST,
        SCONST,

                    // the operators, parens and semicolon
        PLUS,

        … // etc.

                    // any error returns this token
        ERR,

                    // when completed (EOF), return this token
        DONE
};
```

What is an enum?

- An enumerated type
- All possible values are specified in the declaration
- A variable of an enumerated type can only take on one of the specified values
- The compiler assigns the values
- The values are integers but you cannot do math on them

External definitions

```
extern ostream& operator<<(ostream& out, const Token& tok);

extern Token getNextToken(istream *in, int *linenum);
```

- “extern” tells the compiler that someone will provide functions with these signatures
- *you* are the someone

Lexical Rules

- The lexical rules represent the patterns your lexical analyzer must recognize
- You should understand the patterns and build a DFA representing all of the patterns
- Write code to implement the DFA

Pseudocode for getNextToken

- Keep track of your state
- Each state has a set of valid, expected characters in that state
- For each input character
  - Consider your state
  - Decide if the character is valid for that state
  - Process the character (changing state if necessary)

- A switch statement based on state is a reasonable way to implement this
- Create something to represent all the states you need (an enum works well for this)

A small piece of getNextToken

```
Token
getNextToken(istream *in, int *linenum)
{
            enum LexState { BEGIN, INID, INSTRING, /* others */ };
            LexState lexstate = BEGIN;
            string lexeme;
            char ch;

            while(in->get(ch)) {

                        if( ch == '\n' ) {
                                    (*linenum)++;
                        }

                        switch( lexstate ) {
                        case BEGIN:
                                    if( isspace(ch) )
                                                continue;

                                    lexeme = ch;

                                    if( isalpha(ch) ) {
                                                lexstate = INID;
                                    }
                                    else if( ch == '"' ) {
                                                lexstate = INSTRING;
                                    }
```

Peek And Putback

- Your getNextToken function might need to look at the next character from input to decide if the token is finished or not
- Method 1: use the peek() method, examine the next character, and only read it if it belongs to the token
- Method 2: if you read a character that does not belong to the token, use the putback() method to put it back, so that you get() it next time

Main test program

- Process arguments
- Decide where input comes from
- Call getNextToken until it is DONE or creates some ERR condition
- Keep statistics
- Print statistics

Reading from cin or a file

- The first argument to getNextToken is an istream*
- An istream* (a pointer to an istream) can point to cin, or it can point to an ifstream
  - cin is an istream. Therefore &cin is a pointer to an istream. SO you can pass &cin as the first argument if you want to read from standard input
  - ifstream inherits from istream, so it “is a” istream. Therefore the & some ifstream object is a pointer to an istream. SO you can pass that pointer as the first argument if you want to read from a file
- Keep it simple: create an istream* representing the input. Initialize it to either &cin or &the file you opened. Then just pass that variable to getNextToken

Main loop

```
istream *in;
int lineNumber = 0;
…

Token tok;
while( (tok = getNextToken(in, &lineNumber)) != DONE && tok != ERR )
{
        // handle verbose mode
        // keep statistics
}
```

- getNextToken routine will read things a character at a time
- the main program will process the input a token at a time
- counts and statistics are kept in main

Required statistics

- How many lines in the input?
- How many tokens in the input?
- How many strings in the input?
- How many identifiers in the input?
- What are the identifiers?

NJIT

THE EDGE IN KNOWLEDGE