

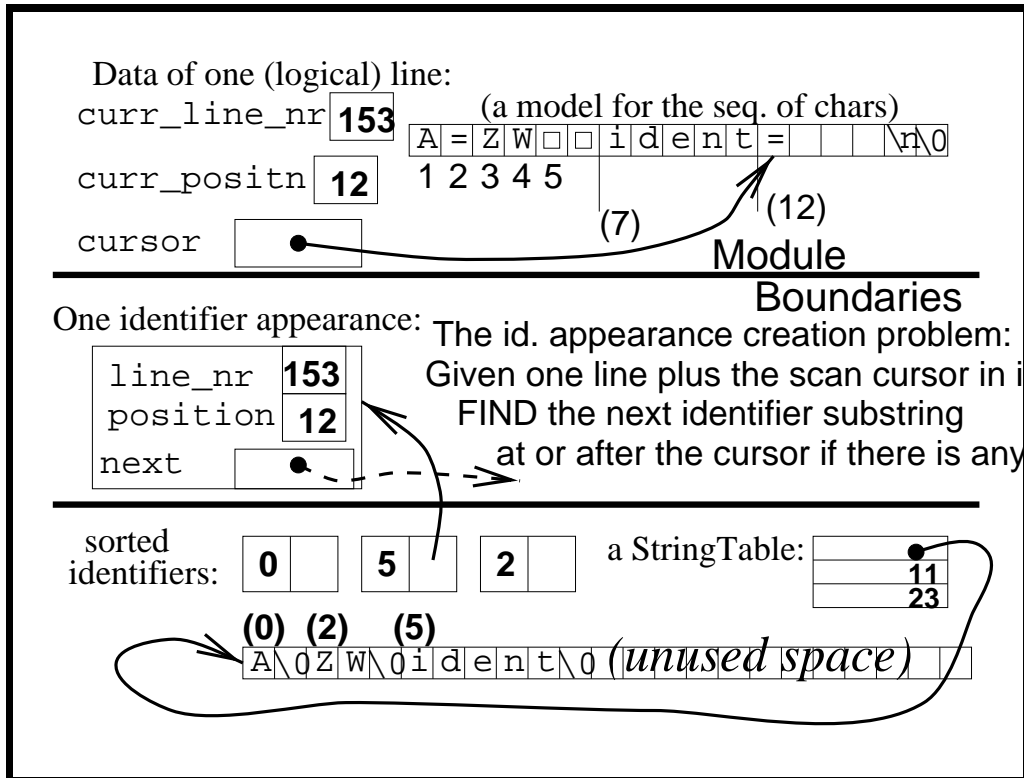
Lecture on scanning CSI333

Project 4 Announcements

1. “Lite Version” 85 points:
 - (a) No backslash line continuations.
 - (b) No multiline comments.
 - (c) No float, long, hex or unsigned constants.SO, “line” = “logical line”.
2. Early Due Date Nov. 22.
 - Thanksgiving grace period.
 - Late turnin period:Nov.27-Nov.30

Slide 1

Slide 2



Slide 3

```

line.h
#include <assert.h>
class line {
public: const int maxlen = 509;
private:
    int curr_line_nr; //for the cursor.
    int curr_positn; //for the cursor.
    char *cursor; //corresponds to above 2 vars.
    char chars[ maxlen + 1 ]; //+1 for the '\0'
public:
    int number(void){ return curr_line_nr; }
    int position(void){ return curr_positn; }
    char currCh(void){ return *cursor; }
    void moveCursor(void) { assert( *cursor != 0 );
        cursor++; curr_positn++; }
    bool readNextLine(void);
    line(void); };
    
```

Slide 4

1. Inline functions `number`, `position`, `currCh`, `moveCursor` are used here mainly for clarity.
2. They are “access functions” that control what a line’s user can access.
3. The member function `readNextLine()` is DEFINED in an implementation file.

Slide 5

```
line.cpp
#include <iostream.h>
#include <stdio.h>
#include <string.h>
#include "line.h"
bool line::readNextLine(void)
{
    int n;
    if ( fgets( chars, maxlen + 1, stdin ) ) {
        if( (strlen( chars ) == maxlen) &&
            (chars[maxlen-1] != '\n'))
            { cout << "Input line too long...exiting" << endl;
              exit(1);
            }
        curr_line_nr = curr_line_nr + 1;
        curr_positn = 1;
        cursor = chars;
        return true; }
    else return false;
}
line::line(void)
{ curr_line_nr = 0; }
```

Slide 6

```

linetester.cpp
#include <iostream.h>
#include "line.h"
line L;
int main(int argc, char *argv) {
    while(L.readNextLine()) {          //A line was read.
        while( L.currCh() != '\0' ) { //There's an unscanned char.
            cout<< L.number()<< ":" //line's number is printed.
                << L.position() << " " //unscanned char posn. printed
                << L.currCh() << endl; //char is now "scanned".
            L.moveCursor(); //Cursor points to unscanned char.
        } //End of line, no more unscanned chars.
    } //No more lines.
    cout << "Goodbye!" << endl;
}

```

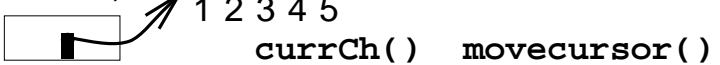
Slide 7

Algorithm: Finite State Automaton (or scanner).

Inputs: One line: **position** **number**

initial position

(1 or previous)

cursor  **currCh()** **movecursor()**

Variables:

line L; (has above data: NEXT char to process)

```
char tempident[L.maxlen]; int lineNum, posId;
```

```
char *ptempident; ptempid = tempident;
```

```
// to help copy identifier during scanning.
```

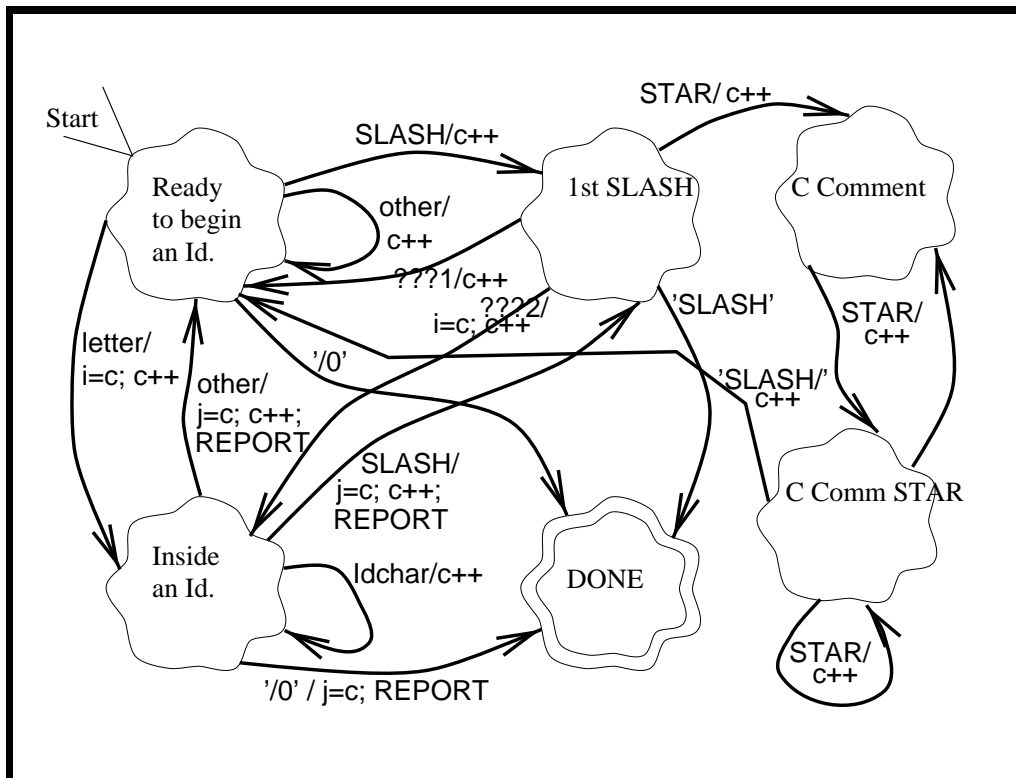
```
posid=L.position(); lineNum=L.number();
```

```
// to help report position of found id.
```

```
enum state_t state = Ready;
```

Req. Reading on enum types: HS 5.5, 5.11.2, 5.13.1

Slide 8



Slide 9

Initialize: current char. position;
and **CURRENT STATE**="Start State"

Finite state scanner loop:

1. Retrieve and classify the current character.
2. Depending on the **CURRENT STATE** and **CURRENT CHAR. CLASS**:
 - (a) Possibly take an action.
 - (b) Choose a new state (may be the same as current).
3. Exit if new state is "DONE"; otherwise make **CURRENT STATE** be the new state and make the current character be the next.

Slide 10

```
#include "classify.h"
line L;
enum state_t { Ready, Not, Inside, DONE };
state_t state; // current STATE
chclass_t chclass; // class of NEXT char
int linenum; // will record the line number
    // when the scanner finds the beginning
    // of an identifier.
int posId; // will record the position
    // when the scanner finds the beginning
    // of an identifier.
char *ptempident;
// Code in a loop that prepares for scanning
state = Ready;
ptempident = &tempident[0];
```

Slide 11

```
// classify.h Classifier for kwic333
enum chclass_t { Space, Letter, Other, Null };
extern chclass_t classify( char ch );
```

```
// classify.cpp Classifier for kwic333
#include <ctype.h>
#include "classify.h"
chclass_t classify( char ch )
{
    if( ch == ' ' )        return Space;
    if( isalpha( ch ) ) return Letter;
    if( ch == '\0' )      return Null;
    return Other;
}
```

Slide 12

```
while (state != DONE) {
    chclass = classify( L.currCh() );
    switch( state ) {
        case Ready:
            switch( chclass ) {
                case Space: state = Ready;
                    L.moveCursor(); break;
                case Other: state = Not;
                    L.moveCursor(); break;
                case Letter: state = Inside;
                    linenum = L.number();
                    posId = L.position();
                    ptempident = tempident;
                    *(ptempident++) = L.currCh();
                    L.movecursor(); break;
                case .... //You figure out!!
            }
            break; // end of state==Ready case
        case Inside: //in switch(state) block
```

Slide 13

```
case Inside: //in switch(state) block
  switch( chclass ) {
    case Letter: state = Inside;
      *(ptempident++) = L.currCh();
      L.movecursor(); break;
    // more chclass cases
  }
  break; // end of state==Inside case

case Not: //in switch(state) block
  // Handle state==Not chclass cases.
  // More state cases.
} // end of switch( state )
} // end of while
```