Agenda

- A toy program and its design
- Lexical analysis
- Lexer: interface and implementation
- Function pointers and map

A toy program and its design

PARSING COMMAND LINES

THINKING ABOUT THE MAIN BODY

A toy program

```
> exit
> quit
> bye
> foreground [red|green|blue] "Print this string"
> fc <file1> <file2> … <filek>
```

How do we design such a program?

A “Good” Design

- The skeleton should be clean, easy to read

- Easy to add codes to accommodate changes in requirements
  - Add “yellow”, “cyan” to the color list
  - Add bf, mp … to the command list

A Skeleton of a "Good" Design

- While there is more to read
  - Read an input line
  - Break the line into tokens
    > foreground green "Print this string in green"
    > bf "the pattern" inputfile.txt
  - Use an "array" indexed by strings to call functions

String	Function
"foreground"	Print_color("green", "Print this string in green")
"bf"	Brute_force_matching("the pattern", inputfile.txt)
"exit"	Quit()
"bye"	Quit()

  - If the command name is not found, say "Command not found"

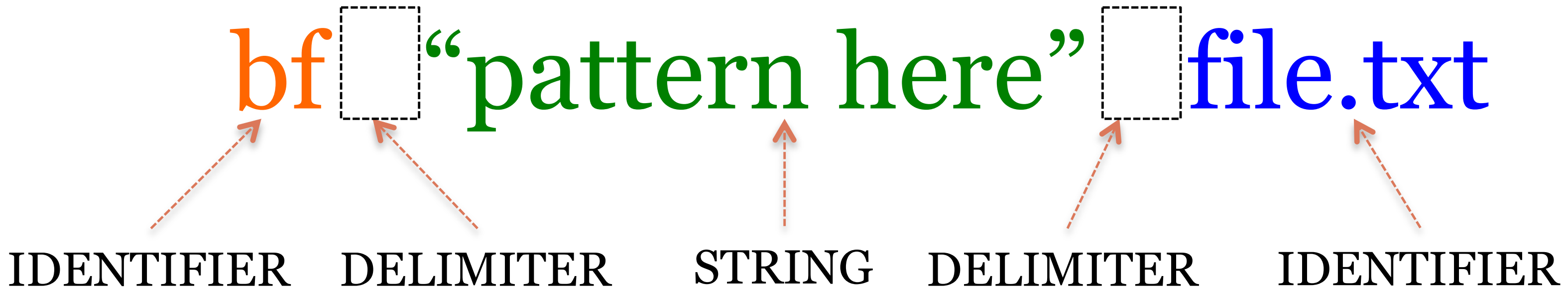
Main questions

- Is there a systematic way to break a string into tokens?
  - Lexical analysis
- How do we define/use an “array” indexed by strings?
  - map
- How do we define/use functions as data types?
  - function pointers

Lexical analysis

- TOKENS
- TOKEN SCANNING
- LEXER

Tokens


bf
"pattern here"
file.txt
IDENTIFIER
DELIMITER
STRING
DELIMITER
IDENTIFIER

Our first C++ class: Lexer

ARRAY SIZE MUST BE A CONSTANT EXPRESSION

SOME EXTENSION ALLOWS “DYNAMIC” SIZE, NOT RECOMMENDED

ARRAY NAME CAN BE USED AS A POINTER TO THE FIRST ELEMENT OF THE ARRAY

How a Lexer class might work?

```
// textest.cpp : a simple driver for the the Lexer class
#include <iostream>
#include "Lexer.h"
using namespace std;

int main() {
    Lexer lexer; Token tok; string line;
    while (cin) {
        cout << "> ";
        getline(cin, line);
        lexer.set_input(line);
        while (lexer.has_more_token()) {
            tok = lexer.next_token();
            if (tok.type == IDENT) cout  << "IDENT: " << tok.value << endl;
            if (tok.type == STRING) cout << "STR:    " << tok.value << endl;
            if (tok.type == ERRTOK) cout << "Don't drink and type" << endl;
        }
    }
    return 0;
}
```

Defining the Lexer's interface: Lexer.h

```
class Lexer {
  public:
    // constructor
    Lexer(std::string str="") : input_str(str), cur_pos(0), in_err(false),
        separators(" \t\n") { }

    // a couple of modifiers
    void set_input(std::string); // set a new input,
    void restart();              // move cursor to the beginning, restart

    Token next_token();    // returns the next token
    bool has_more_token(); // are there more token(s)?

  private:
    std::string input_str;  // the input string to be scanned
    size_t      cur_pos;    // current position in the input string
    bool        in_err;     // are we in the error state?
    std::string separators; // set of separators; *not* the best option!
};
```

Constructor

- Called when an object is created
- string str; // the “default constructor” is called
- string str(“David Blaine”); // another constructor
- Lexer lexer(“This is to be parsed”);

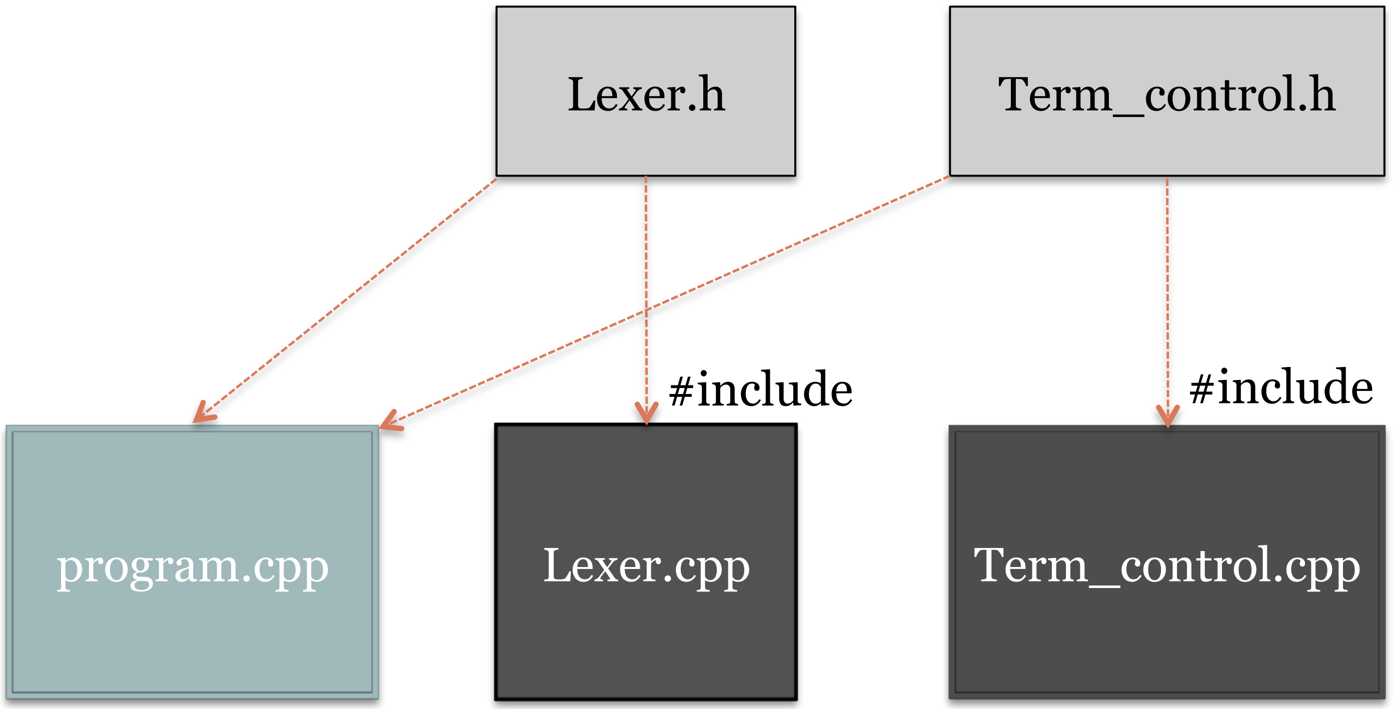
Other types in Lexer.h

```
enum token_types_t {
    IDENT,  // a sequence of alphanumeric characters and _,
    STRING, // sequence of characters between " ", no escape
    ENDTOK, // end of string/file, no more token
    ERRTOK  // unrecognized token
};

struct Token {
    token_types_t type;
    std::string value;
    // constructor for Token
    Token(token_types_t tt=ENDTOK, std::string val="") :
type(tt), value(val) {}
};
```

Struct is a class all of whose members are public by default

Organizing Codes



Function pointers and map

- DEFINING FUNCTION POINTERS
- MAP FROM STRING TO FUNCTION POINTERS

Defining function pointers

```
#include <iostream>
using std::cout;
using std::endl;

int add(int x, int y) { return x+y; }
int sub(int x, int y) { return x-y; }

int main() {
    // define fp to be a pointer to a function which takes
    // two int parameters and returns an int
    int (*fp)(int, int);

    fp = &add;
    cout << fp(6,3) << endl; // get 9
    fp = &sub;
    cout << fp(6,3) << endl; // get 3

    return 0;
}
```

Mapping strings to function pointers

```
typedef void (*cmd_handler_t)(Lexer);
void print_color(Lexer);
void bye(Lexer);
.
.
.

map<string, cmd_handler_t> cmd_map;

// simply add all commands to the map
cmd_map["foreground"] = &print_color;
cmd_map["exit"] = &bye;
cmd_map["bye"] = &bye;
cmd_map["quit"] = &bye;
```