Compilers

INF-400

Burak Arslan
ext-inf400@burakarslan.com

Galatasaray Üniversitesi

Lecture III
2023-10-19

Course website

burakarslan.com/inf400

First assignment will be out today!

Deadline: **2023-11-01 23:59**

Lexing

```
import io

func main() -> int32 {
    io.print("Hello world!\n");
    return 0;
}
```

Lexing

```
KW_IMPORT IDENTIFIER(io) OP_NEWLINE
KW_FUNC IDENTIFIER(main) OP_LPAREN OP_RPAREN OP_RETURNS
                                IDENTIFIER(int32) OP_LBRACE OP_NEWLINE
IDENTIFIER(io) OP_DOT IDENTIFIER(print) OP_LPAREN
               L_STRING("Hello world!\n") OP_RPAREN OP_SCOLON OP_NEWLINE
IDENTIFIER(return) L_INTEGER(0, 10) OP_SCOLON OP_NEWLINE
OP_RBRACE OP_NEWLINE
```

* Indented lines are continuations of the previous ones. There is actually only 5 lines in the above output

Lexing

Tokens

Language definitions start with tokens types:

Type	Examples
IDENTIFIERs	`foo bar main`
INTEGERs	123 -456 42 0xFF
FLOATs	12.34 +56.78
KEYWORDs	`if while import`
OPERATORs	, . >= ==

Lexing

Tokens

A token type specifies a **set of acceptable strings**

Lexing

Tokens

A token type specifies a **set of acceptable strings**

Type	**Examples**
INTEGERs	123 -456 42 0xFF

... but INTEGER set is infinite (?)

Lexing

Tokens

So we turn to regular languages. More specifically;

1. **Regular Expressions**: for defining the sets
2. **Deterministic Finite State Automata**: for testing set membership

Lexing

Regular Expressions

Some examples:

Type	**Regex**
IF	
±INTEGER (dec)	
+INTEGER (hex)	
IDENTIFIER	
KEYWORD	

Lexing

Regular Expressions

Some examples:

Type	**Regex**
IF	`if`
±INTEGER (dec)	
+INTEGER (hex)	
IDENTIFIER	
KEYWORD	

Lexing

Regular Expressions

Some examples:

Type	**Regex**
IF	`if`
±INTEGER (dec)	[0-9]+
+INTEGER (hex)	
IDENTIFIER	
KEYWORD	

Lexing

Regular Expressions

Some examples:

Type	**Regex**
IF	`if`
±INTEGER (dec)	`-?[0-9]+`
+INTEGER (hex)	
IDENTIFIER	
KEYWORD	

Lexing

Regular Expressions

Some examples:

Type	**Regex**
IF	`if`
±INTEGER (dec)	`(+|-)?[0-9]+`
+INTEGER (hex)	
IDENTIFIER	
KEYWORD	

Lexing

Regular Expressions

Some examples:

Type	**Regex**
IF	`if`
±INTEGER (dec)	`(+|-)?[0-9]+`
+INTEGER (hex)	`0x[0-9af-A-F]+`
IDENTIFIER	
KEYWORD	

Lexing

Regular Expressions

Some examples:

Type	**Regex**
IF	`if`
±INTEGER (dec)	`(+|-)?[0-9]+`
+INTEGER (hex)	`0x[0-9af-A-F]+`
IDENTIFIER	`[a-zA-Z][a-zA-Z0-9]*`
KEYWORD	

Lexing

Regular Expressions

Some examples:

Type	**Regex**
IF	`if`
±INTEGER (dec)	`(+|-)?[0-9]+`
+INTEGER (hex)	`0x[0-9af-A-F]+`
IDENTIFIER	`[a-zA-Z][a-zA-Z0-9]*`
KEYWORD	`(if|while|import|func)`

Lexing

Regular Expressions

Let's classify the following token: **if**

Type	**Regex**
IF	`if`
±INTEGER (dec)	`(+|-)?[0-9]+`
+INTEGER (hex)	`0x[0-9a-f-A-F]+`
IDENTIFIER	`[a-zA-Z][a-zA-Z0-9]*`
KEYWORD	`(if|while|import|func)`

Lexing

Regular Expressions

Let's classify the following token: **if** $\Rightarrow$ `IF`

Type	**Regex**
IF	`if`
±INTEGER (dec)	`(+|-)?[0-9]+`
+INTEGER (hex)	`0x[0-9a-f-A-F]+`
IDENTIFIER	`[a-zA-Z][a-zA-Z0-9]*`
KEYWORD	`(if|while|import|func)`

Lexing

Regular Expressions

Let's classify the following token: **1234**

Type	Regex
IF	`if`
±INTEGER (dec)	`(+|-)?[0-9]+`
+INTEGER (hex)	`0x[0-9a-f-A-F]+`
IDENTIFIER	`[a-zA-Z][a-zA-Z0-9]*`
KEYWORD	`(if|while|import|func)`

Lexing

Regular Expressions

Let's classify the following token: **1234** $\Rightarrow$ `INTEGER`

Type	**Regex**
IF	`if`
±INTEGER (dec)	`(+|-)?[0-9]+`
+INTEGER (hex)	`0x[0-9a-f-A-F]+`
IDENTIFIER	`[a-zA-Z][a-zA-Z0-9]*`
KEYWORD	`(if|while|import|func)`

Lexing

Regular Expressions

Let's classify the following token: **1234A**

Type	**Regex**
IF	`if`
±INTEGER (dec)	`(+|-)?[0-9]+`
+INTEGER (hex)	`0x[0-9a-f-A-F]+`
IDENTIFIER	`[a-zA-Z][a-zA-Z0-9]*`
KEYWORD	`(if|while|import|func)`

Lexing

Regular Expressions

Let's classify the following token: **1234A** $\Rightarrow$ REJECT

Type	**Regex**
IF	`if`
±INTEGER (dec)	`(+|-)?[0-9]+`
+INTEGER (hex)	`0x[0-9a-f-A-F]+`
IDENTIFIER	`[a-zA-Z][a-zA-Z0-9]*`
KEYWORD	`(if|while|import|func)`

Lexing

Regular Expressions

Let's classify the following token: **import**

Type	**Regex**
IF	`if`
±INTEGER (dec)	`(+|-)?[0-9]+`
+INTEGER (hex)	`0x[0-9a-f-A-F]+`
IDENTIFIER	`[a-zA-Z][a-zA-Z0-9]*`
KEYWORD	`(if|while|import|func)`

Lexing

Regular Expressions

Let's classify the following token: **import** $\Rightarrow$ KEYWORD

Type	**Regex**
IF	`if`
±INTEGER (dec)	`(+|-)?[0-9]+`
+INTEGER (hex)	`0x[0-9a-f-A-F]+`
IDENTIFIER	`[a-zA-Z][a-zA-Z0-9]*`
KEYWORD	`(if|while|import|func)`

Lexing

Regular Expressions

Let's classify the following token: **importer**

Type	**Regex**
IF	`if`
±INTEGER (dec)	`(+|-)?[0-9]+`
+INTEGER (hex)	`0x[0-9a-f-A-F]+`
IDENTIFIER	`[a-zA-Z][a-zA-Z0-9]*`
KEYWORD	`(if|while|import|func)`

Lexing

Regular Expressions

Let's classify the following token: **importer** $\Rightarrow$ IDENTIFIER

Type	**Regex**
IF	`if`
±INTEGER (dec)	`(+|-)?[0-9]+`
+INTEGER (hex)	`0x[0-9a-f-A-F]+`
IDENTIFIER	`[a-zA-Z][a-zA-Z0-9]*`
KEYWORD	`(if|while|import|func)`

Lexing

Regular Expressions

Let's classify the following token: **importer** ⇒ IDENTIFIER

???

Type	Regex
IF	`if`
±INTEGER (dec)	`(+|-)?[0-9]+`
+INTEGER (hex)	`0x[0-9a-f-A-F]+`
IDENTIFIER	`[a-zA-Z][a-zA-Z0-9]*`
KEYWORD	`(if|while|import|func)`

Lexing

Regular Expressions

Ambiguities in grammars are practically impossible to avoid. C++ Examples:

- `A < B > c;`
  - if A and B are types, this is a variable definition;
  - if A and B are variables, this is a (pointless) comparison;
- `cin >> var;`
  - **operator**>>
- `vector<unique_ptr<string>>;`
  - Right angle bracket

In the face of ambiguities;

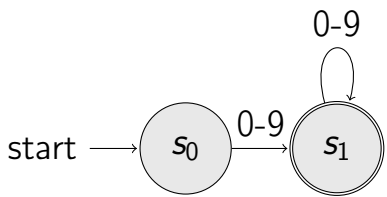
Longest match is preferred

Order of rules matter

Lexing

Deterministic Finite State Automata

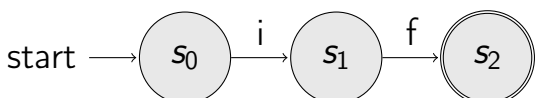
DFA that decides whether an input matches regexp: `[0-9]+`



Lexing

Deterministic Finite State Automata

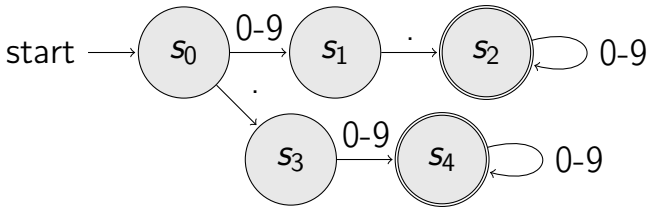
DFA that decides whether an input matches regexp: `if`



Lexing

Deterministic Finite State Automata

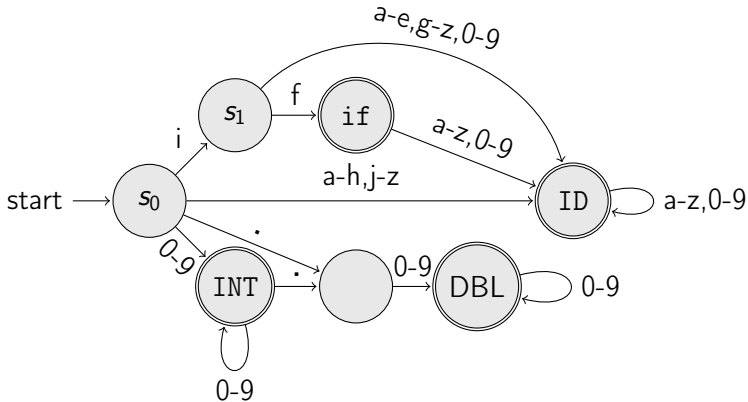
DFA ... regexp: . . . ?



Lexing

Deterministic Finite State Automata

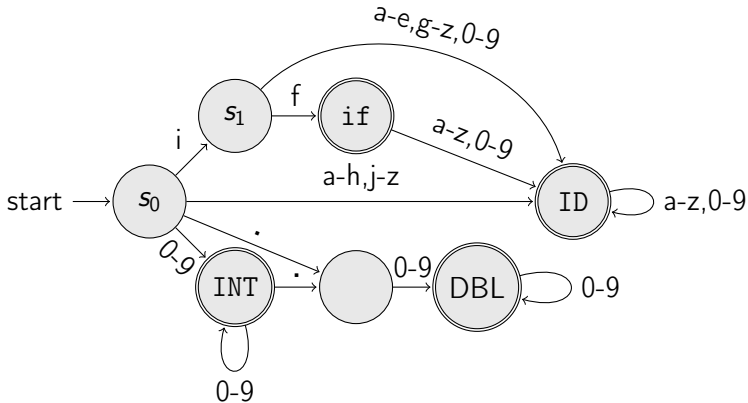
The combined DFA becomes the **lexical analyzer**



Lexing

Deterministic Finite State Automata

The **lexer generator**'s job is to combine all regexp to a coherent **DFA**



Lexing

Deterministic Finite State Automata

The lexgen is used to assign **callbacks** to accepting states