Marcello Bersani

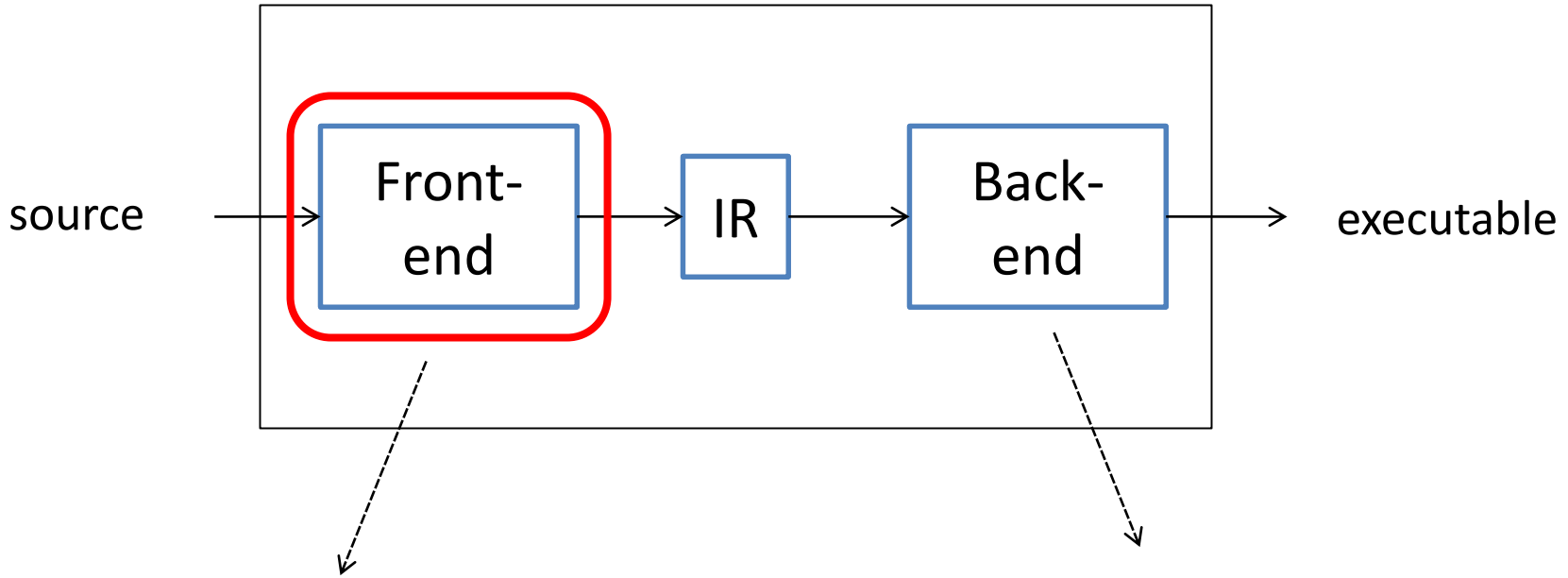
- bersani@elet.polimi.it
- http://home.dei.polimi.it/bersani/
- Ed. 22, via Golgi 42, 3 piano
- 3769

Flex , Bison and the ACSE compiler suite

Marcello M. Bersani

LFC – Politecnico di Milano

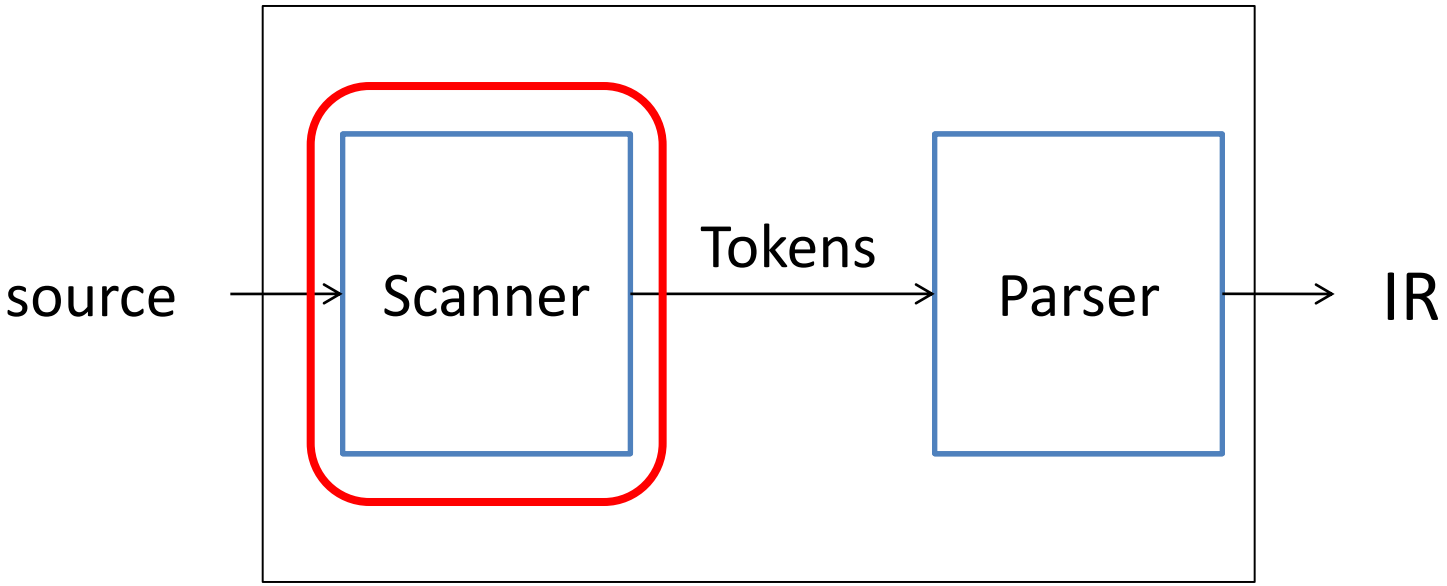
Basic compiler structure



translates a computer programming source language into an intermediate representation

works with the internal representation to produce code in a computer output language.

Front-end



Lexical analysis

- The **lexical analysis** recognizes **patterns** in a stream of characters.
- A **pattern** represents an atomic lexical elements, named “tokens”.
  - Each token can have one or more attributes

ID = (A..Z|a..z)(A..Z|a..z|0..9|’_’)*

my_variable10

Scanner

- A program that performs lexical analysis.

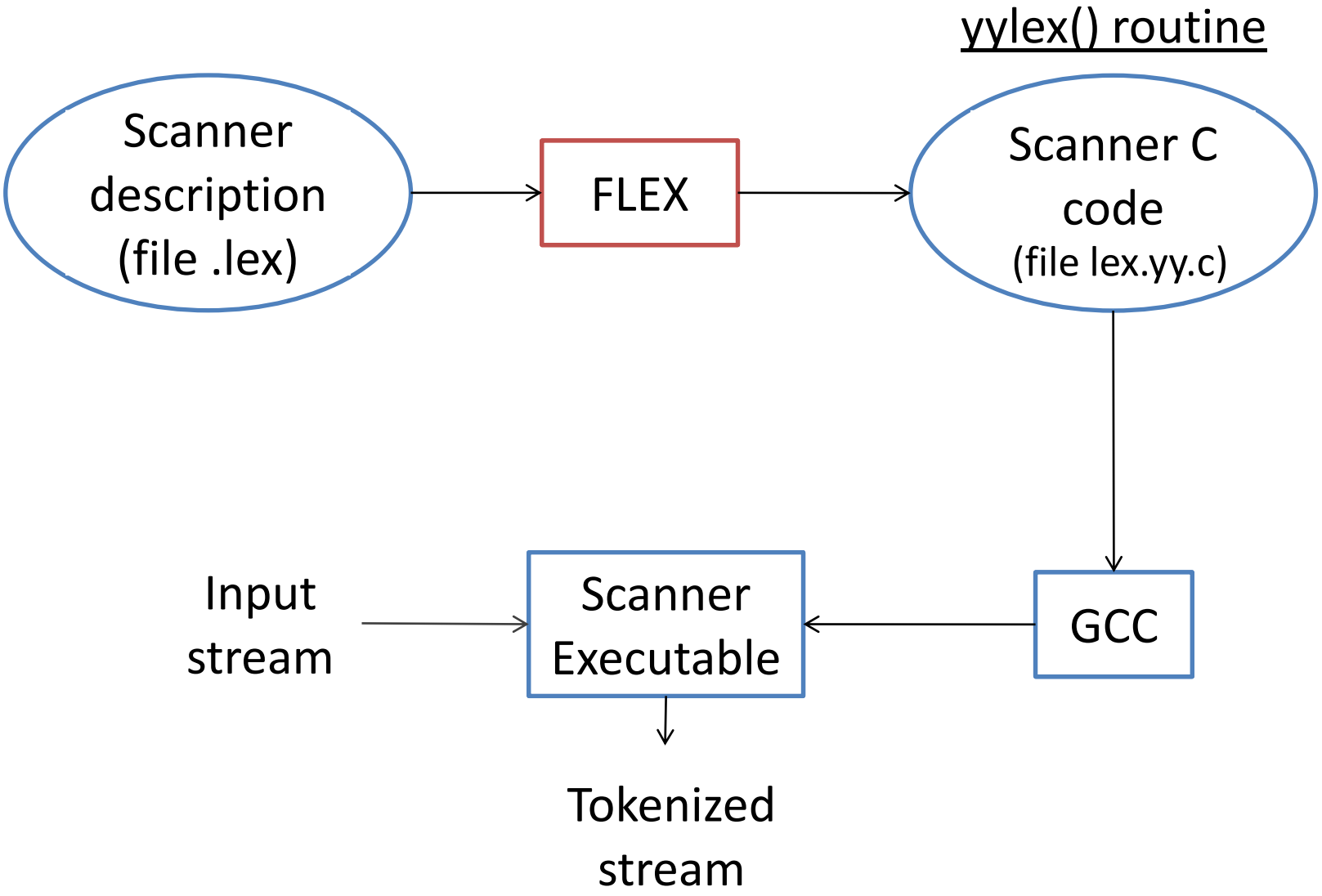
- There are programs generating scanners automatically.
  - **`flex`** is a free scanner generator, a complete rewriting of the AT&T Unix tool **`lex`**.

    www.gnu.org/software/flex/

  - A useful book:

    lex & yacc, 2nd Edition
    by John Levine, Tony Mason & Doug Brown
    O'Reilly

How flex works



Flex input

definitions

%%

rules

%%

user code

Definitions

- **name definition**

  DIGIT [0-9]

  ID [a-z][a-z0-9]*

- %{

  C code for declarations

  }%

  Remark: C code is copied verbatim into lex.yy.c

Rules

pattern **action**

[DIGIT]+	{ C code for DIGIT+}
\n	{ C code for \n}
ID	{ C code for ID}

- **Action**: C code to be executed each time a token in the input stream matches the associated pattern.
- **Pattern** condition: a regular expression

RegEx rules

x	matches the character ‘x’
.	matches any character except newline
[xyz]	any character in the given set (x | y | z)
[a-z]	any character in the given range (a | b |…| z)
[^A-Z]	any character but those in the range
{x}	expansion of x’s definition
“…”	a literal string
\x	ANSI-C interpretation of \x, if any, otherwise the literal x (to escape operators)
\0	the NUL character
<<EOF>>	the end-of-file

RegEx rules

R	the regular expression R
RS	the concatenation of R and S
R | S	either R or S
R*	zero or more R's
R+	one or more R's
R?	zero or one R's
R{m,n}	a number of R's ranging from m to n
R{n,}	n or more times R's
R{n}	exactly n R's
(R)	(parentheses to override precedence)
R/S	R, but only if followed by S
^R	R, but only at beginning of a line
R$	R, but only at end of a line
<s>R	R, but only in start condition s_1
<s_1,…,s_n>R	R, in any of start conditions s_1,…,s_n
<*>R	R, in any start condition, even an exclusive one

User code

- User C code is copied to the generated scanner source “**as is**”.

- This section may contain routines called by actions, or code which calls the scanner.

Example 1

A scanner which converts every lowercase character, in a given text, to an uppercase character and every uppercase character to a lowercase character.

```
%{
#include <unistd.h>
%}

%option noyywrap

LETTER [a-zA-Z]
DIGIT [0-9]


%%
"esci"        yyterminate();
{LETTER}+  {
               int i;
               for(i=0; i<yyleng; i=i+1)
                  if ( islower( yytext[i] ))
                    putchar( toupper( yytext[i] ));
                  else
                    putchar( tolower ( yytext[i] ));
               }
%%

main() {
    yylex();
}
```

When the scanner reads EOF it may check yywrap() function:

- true: the scanner terminates
- false: yywrap() should have set a new input stream

yywrap() is not provided

yyleng: length of current token

yytext: current token

How flex reads the input stream

1. It reads (**YY_INPUT**) the input stream (**yyin**), looking for strings matching any of its patterns.
2. Once the right match is found, the substring is available by the global **char*** variable **yytext**, and its length is available in **yyleng**.
3. Macro **YYINPUT** controls how **yyin** is read
    1. Standard one provides “block-reads” until \n, EOF

4. **yylex()** continues processing tokens (from yyin) and returns when
   - EOF is read
   - an action returns.
5. Each time **yylex()** is called it continues processing tokens from where it stopped

How the right match is determined

1. **Longest matching rule**:
   if more than one matching string is found, the rule that generates the longest one is selected.
2. **First Rule**:
   if more than one string with the same length are found, the rule listed first in the rules section is selected.
3. If no rules were found, the scanner performs the standard action: the next character in input is considered matched and it is copied to the output stream (**yyout**); then it goes on.

Example 2

A scanner which recognizes, in a given text, the words composed by only lowercase letters. It emits an message when it detects lowercase words and returns “success” if, at least one, is found.

```
%{
#include <unistd.h>
int something_good=0;
%}

%option noyywrap

LETTER [a-zA-Z]
DIGIT [0-9]
SPACE [" "]


%%
...
%%


main() {
  printf("Something is good %d\n", yylex());
}
```

```
%%
%{
printf("OK, now we try to check if some words are written in
       lowercase!\n");
%}

"esci"        {
                if (something_good)
                  return 1;
                else  yyterminate();
              }
{LETTER}+  {
                int i;
                int ok=1;
                for(i=0; i<yyleng && ok; i=i+1){
                    if ( isupper(yytext[i])) ok = 0;
                }
                if (ok){
                    printf("GOOD! %s\n", yytext);
                    something_good=1;
                }
                else
                    printf("ERROR!: upper case %c\n", yytext[i-1]);
            }
{SPACE} ;
{DIGIT} ;
%%
```

Special directives

- **ECHO**

  copies yytext to the output.
- **REJECT**

  chooses the next best matching rule;

```
%%
a       |
ab      |
abc     |
abcd    ECHO;REJECT;
.
%%
```

Input: abcd

output: abcdabcaba

Special directives

- **`yyless(n)`**
  sends back to the input stream all but the first n characters of the matched string.

```
%%
abc         ECHO; yyless(2);
[a-z]+      ECHO;
%%
```

Input: abc

Output: abcc

Special directives

- **`yymore()`**

  the next matched text is appended to yytext
  yytext is not rewritten)

```
%%
a       ECHO;yymore();
b       ECHO;
%%
```

Input: ab

Output: aab

Special directives

- **`input()`**

  consumes the next character in input.

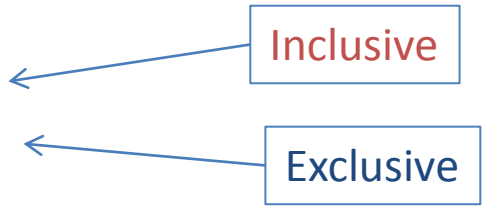
- **`unput(c)`**

  sends character c back to the input stream; c is the next character scanned

  WARNING: calls to unput(c) trash the contents of yytext; therefore contents of yytext must be copied before calling unput(c), if required.

Starting conditions

- Mechanism for conditionally activating rules




```
%s C1
%x C2
%%
<C1>R1      {do_something();}
<C2>R2      ;
<*>R3       {always_do_something();}
R4          {…}
%%
```

- Inclusive **%s** C : Rules tagged by <C> or with no conditions are active.
- Exclusive **%x** C: only rules tagged by <C> are active
- INITIAL: state where only rules with no start conditions are active
- BEGIN(C): starting condition C begins

Example

```
%x comment
%option noyywrap

%%
<INITIAL>[^/]* ECHO;
<INITIAL>"/"+[^*/]* ECHO;
<INITIAL>"/*" BEGIN(comment);

<comment>[^*]*
<comment>"*"+[^/]*
<comment>"*"+"/" BEGIN(INITIAL);
%%


int main(int argc, char* argv[]){
        argv++;  argc--;
        yyin=fopen(argv[0],"r");
        yylex();
}
```

```
ab
a b c/
ab
a b c
/*
/

/* a */
*
* a */
```

```
ab
a b c/
/*
* a */
```

Example

```
%x comment
%option noyywrap

%%
<INITIAL>[^/]* ECHO;
<INITIAL>"/"+[^*/]* ECHO;
<INITIAL>"/*" BEGIN(comment);

<comment>[^*]*
<comment>"*"+[^*/]*
<comment>"*"+"/" BEGIN(INITIAL);
%%


int main(int argc, char* argv[]){
        argv++;  argc--;
        yyin=fopen(argv[0],"r");
        yylex();
}
```

```
ab
a b c
/*
ab
a b c
 a */
```

```
ab
a b c
```

Starting conditions

- Start conditions are stored as integer values
  - The initial start condition is **INITIAL** (0)
- The current start condition is stored in **YY_START** variable
- Starting condition scopes may be nested
  - void **yy_push_state** (int n)
  - void **yy_pop_state** ()
  - int **yy_top_state** ()

%option stack required

Multiple input buffers

- The scanner requires reading from several input streams

YY_BUFFER_STATE **yy_create_buffer**(FILE* file, int size)
void **yy_switch_to_buffer**(YY_BUFFER_STATE new_b)
void **yy_delete_buffer**(YY_BUFFER_STATE buffer)
void **yypush_buffer_state**(YY_BUFFER_STATE buffer)
void **yypop_buffer_state**()

YY_CURRENT_BUFFER is the handle (YY_BUFFER_STATE) for the current buffer.

Interaction Flex/Bison