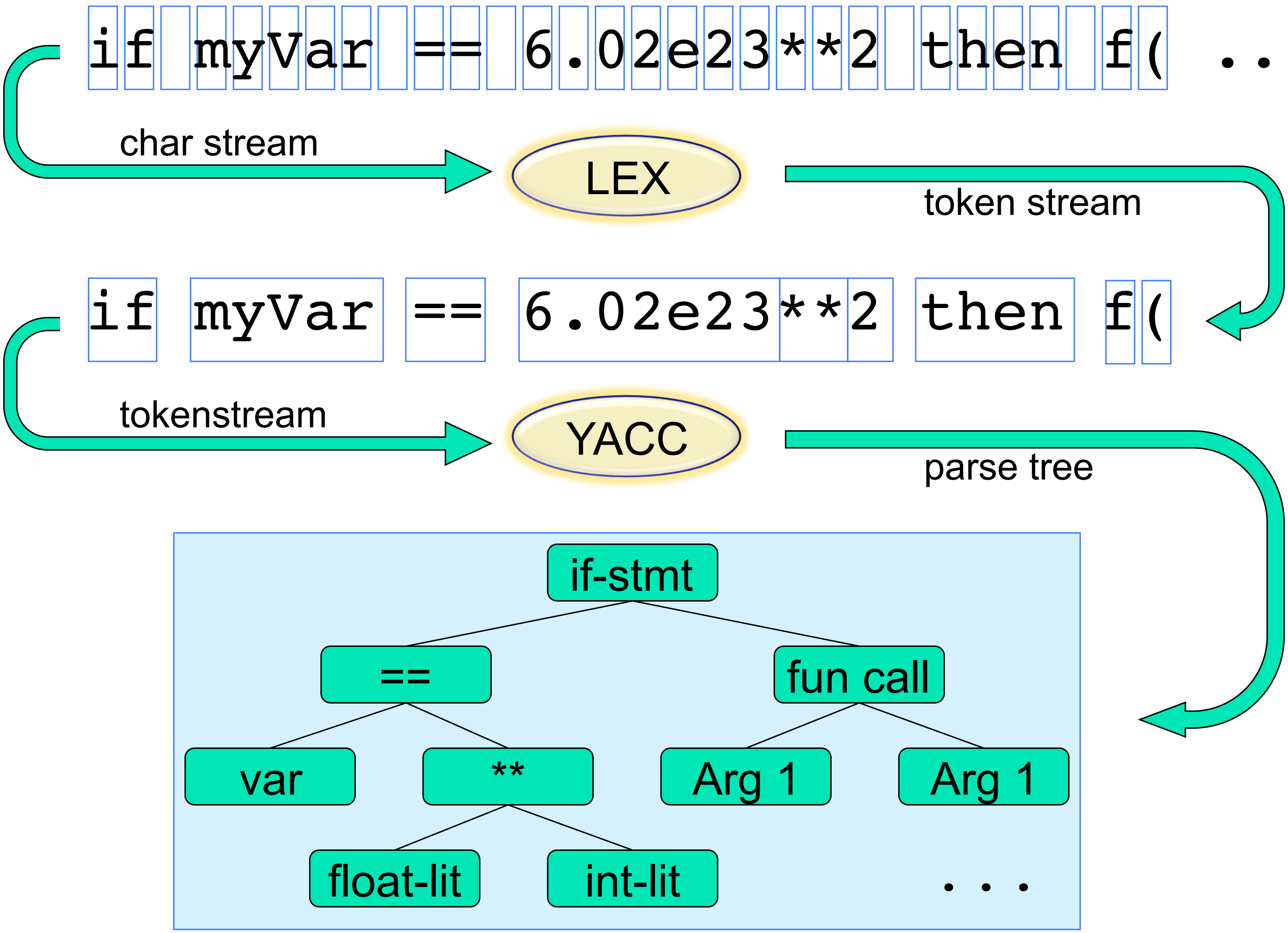
Lex and Yacc

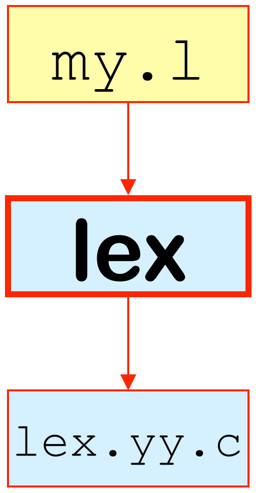
A Quick Tour



if myVar == 6.02e23**2 then f(..
char stream
LEX
token stream
if myVar == 6.02e23**2 then f(
tokenstream
YACC
parse tree
if-stmt
==
fun call
var
**
Arg 1
Arg 1
float-lit
int-lit
. . .

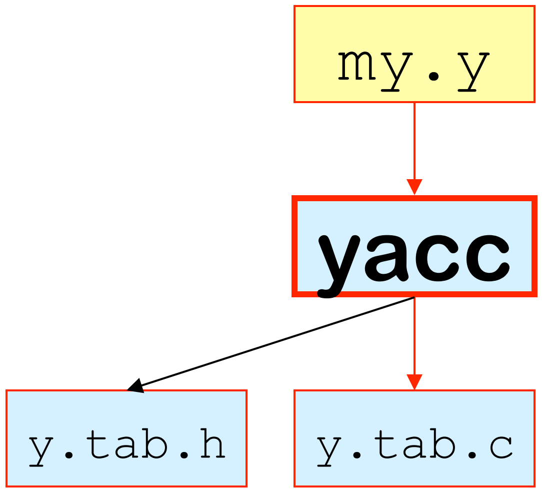
Lex (& Flex): A Lexical Analyzer Generator

- Input:
  - Regular exprs defining "tokens"
  - Fragments of C decls & code
- Output:
  - A C program "lex.yy.c"
- Use:
  - Compile & link with your main()
  - Calls to yylex() read chars & return successive tokens.



Yacc (& Bison & Byacc…): A Parser Generator

- Input:
  - A context-free grammar
  - Fragments of C declarations & code
- Output:
  - A C program & some header files
- Use:
  - Compile & link it with your main()
  - Call yyparse() to parse the entire input file
  - yyparse() calls yylex() to get successive tokens



Lex Input: "mylexer.l"

```
%{
    #include …
    int myglobal;
    …
%}
%%
[a-zA-Z]+    {handleit(); return 42; }
[ \t\n]      {; /* skip whitespace */}
…
%%
void handleit() {…}
…
```

Declarations: To front of C program

Token code

Rules and Actions

Subroutines: To end of C program

Lex Regular Expressions

Letters & numbers match themselves

Ditto \n, \t, \r

Punctuation often has special meaning

- But can be escaped: * matches "*"

Union, Concatenation and Star

- r|s, rs, r*; also r+, r?; parens for grouping

Character groups

- [ab*c] == [*cab], [a-z2648AEIOU], [^abc] S → E

E → E+n | E-n | n

Yacc Input: “expr.y”

```
C Decls     %{
               #include …                       → y.tab.c
            %}
Yacc        %token NUM VAR                     → y.tab.h
Decls       %%
            stmt: exp               { printf(”%d\n”,$1);}
              ;
Rules       exp : exp ’+’ NUM       { $$ = $1 + $3; }
and           |   exp ’-’ NUM       { $$ = $1 - $3; }
Actions       |   NUM               { $$ = $1; }
              ;
            %%
Subrs       …                                  → y.tab.c
```

Expression lexer: “expr.l”

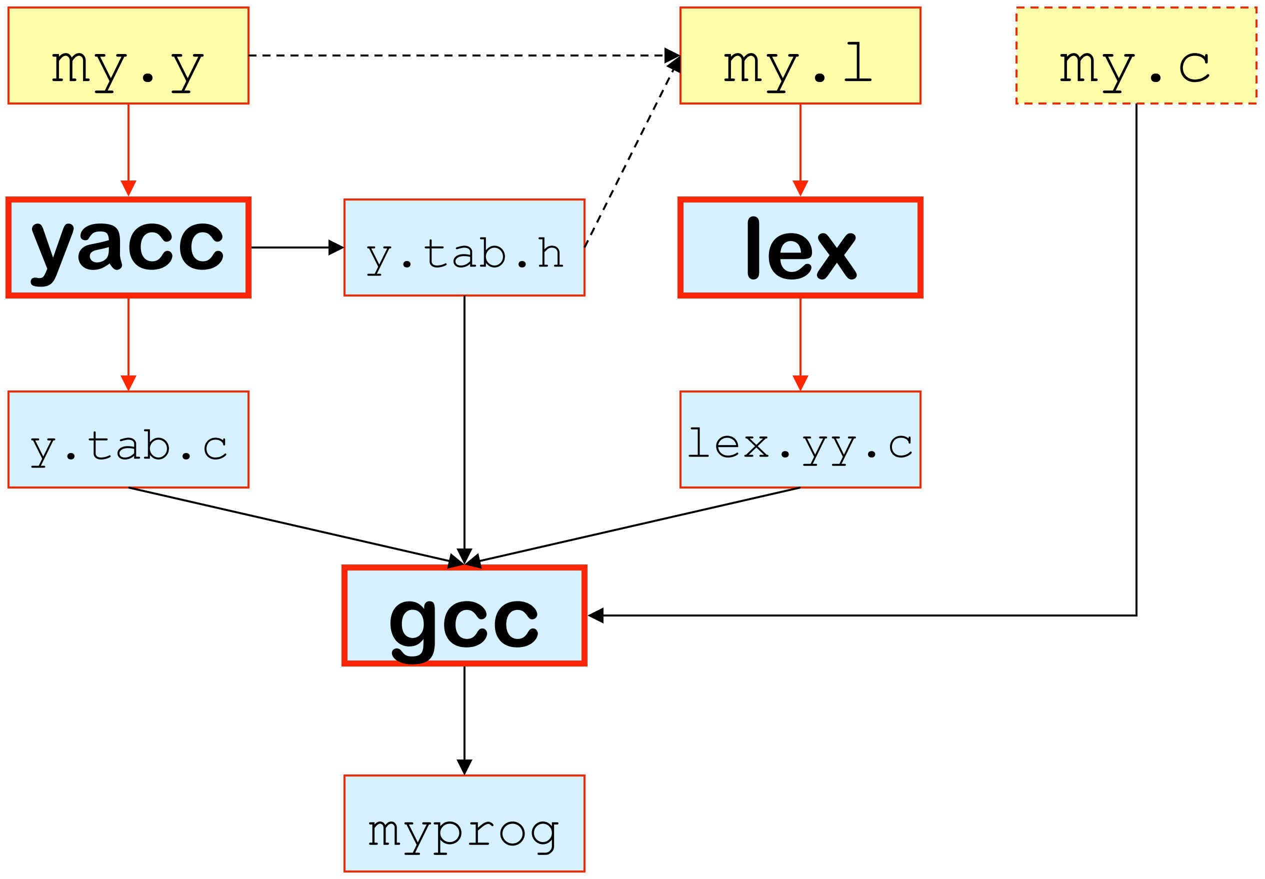
```
%{
#include "y.tab.h"

%}
%%
[0-9]+        { yylval = atoi(yytext); return NUM;}
[ \t]         {           /* ignore whitespace */ }
\n            { return 0;       /* logical EOF */ }
.             { return yytext[0]; /* +-*, etc. */ }
%%
yyerror(char *msg){printf("%s,%s\n",msg,yytext);}
int yywrap(){return 1;}
```

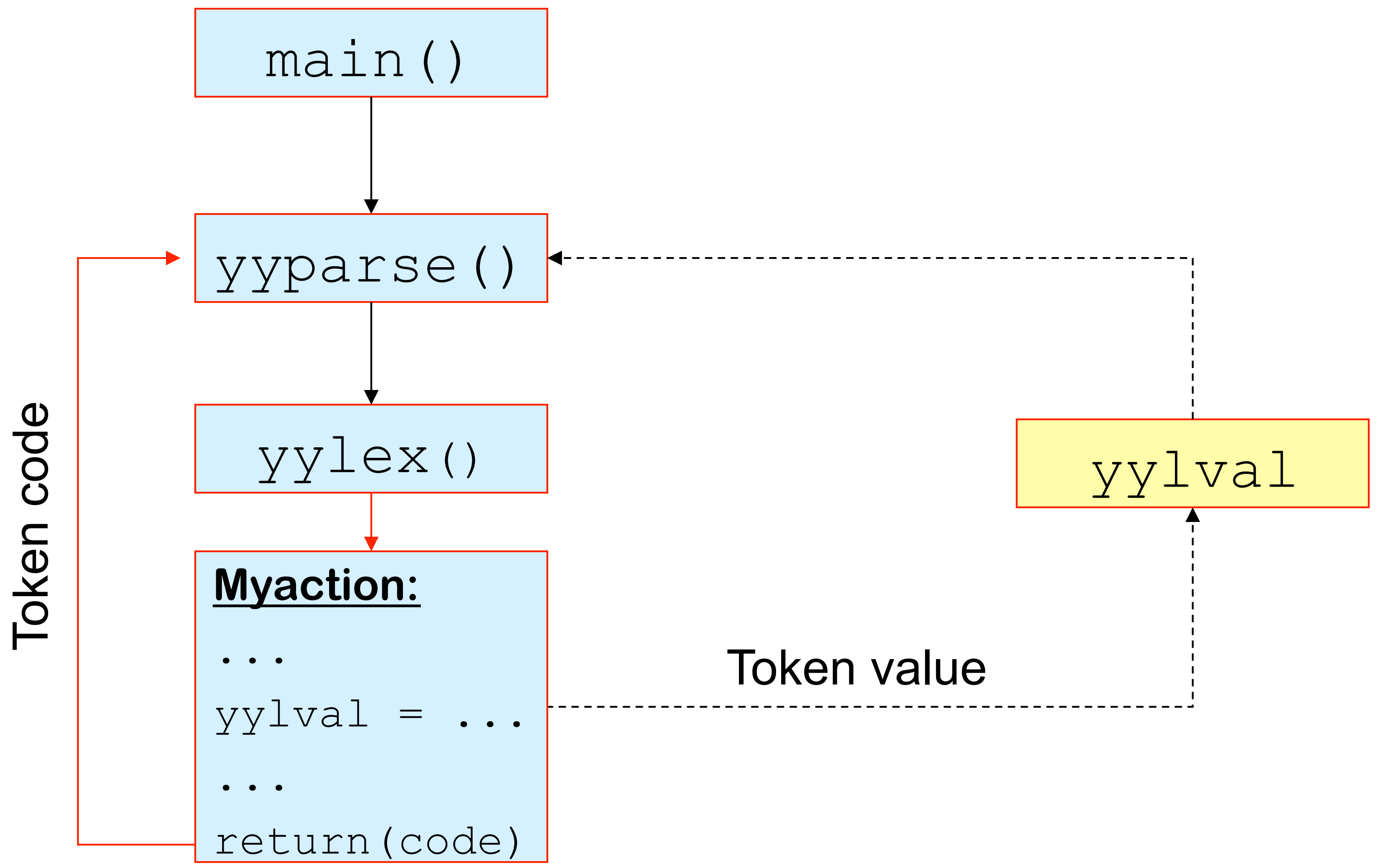
y.tab.h:

```
#define NUM     258
#define VAR     259
#define YYSTYPE int
extern  YYSTYPE yylval;
```

Lex/Yacc Interface: Compile Time



Lex/Yacc Interface: Run Time



Some C Tidbits

Enums

```
enum kind {
   title_kind,center_kind};
typedef struct node_s{
    enum kind k;
    struct node_s
      *lchild,*rchild;
    char *text;
  } node_t;
node_t root;
root.k = title_kind;
if(root.k==title_kind){…}
```

Malloc

```
root.rchild = (node_t*)
 malloc(sizeof(node_t));
```

Unions

```
typedef union {
     double d;
     int i;
 } YYSTYPE;
extern YYSTYPE yylval;
yylval.d = 3.14;
yylval.i = 3;
```