

Project 2

YACC Programming

Virtual Machine & Optimization Laboratory
Dept. of Electrical and Computer Engineering
Seoul National University





Plan

1. Lexical analysis
2. Yacc programming
3. Semantic analysis
4. Code generation



INTRODUCTION

YACC

Yet Another Compiler Compiler

➤ compiler-generator, compiler-compiler

Grammar rules의 항목들로부터 parser를 만들 수 있는 C코드를 생성
Lex와 비슷한 구조

➤ 실제로는 lex가 yacc을 본따서 만들었음

Bottom-up Parsing

여러가지 yacc중 GNU프로그램인 **bison**을 사용

➤ <http://www.gnu.org/s/bison/>

Bottom-up(Shift-Reduce) Parsing

Basic Operations

- **SHIFT**: push the next token onto the stack
- **REDUCE**: replace RHS on stack top of some production (i.e. handle) by its LHS(nonterminal)
- **ACCEPT**: reduction to the start nonterminal
 - Assume a unique S production
 - If not, augment the grammar $S' \rightarrow S$

In each step we must choose

- SHIFT or REDUCE
- If REDUCE, which production

Example of Bottom-Up Paring

$S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

SHIFT/REDUCE Parsing

a **b** b c d e

a **A** b c d e

a **A** **d** e

a A B e

S

■ : handle

Rightmost Derivation

$S \Rightarrow a A B e$

$\Rightarrow a A d e$

$\Rightarrow a A b c d e$

$\Rightarrow a b b c d e$

How Lex & YACC Works

Lex위에 yacc이 있는 구조

- yacc의 필요에 의해 lex가 호출

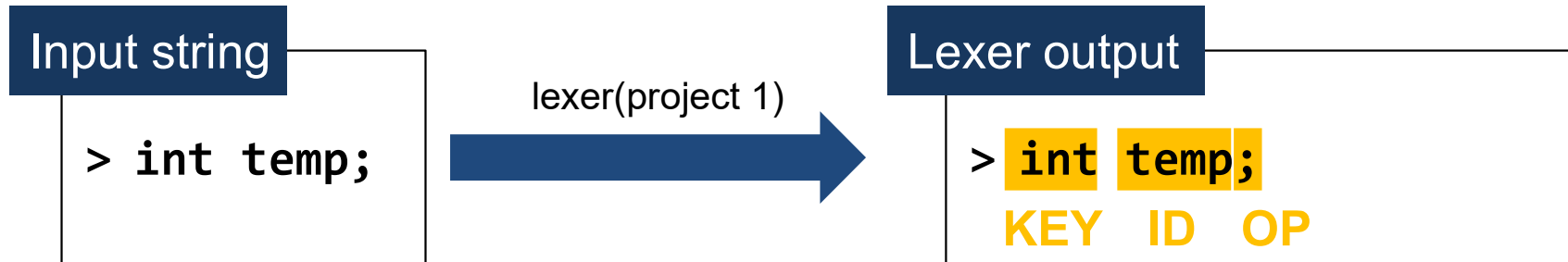
Lex

- 입력된 문자열에 대해 token을 찾아서 yacc에 넘겨준다.

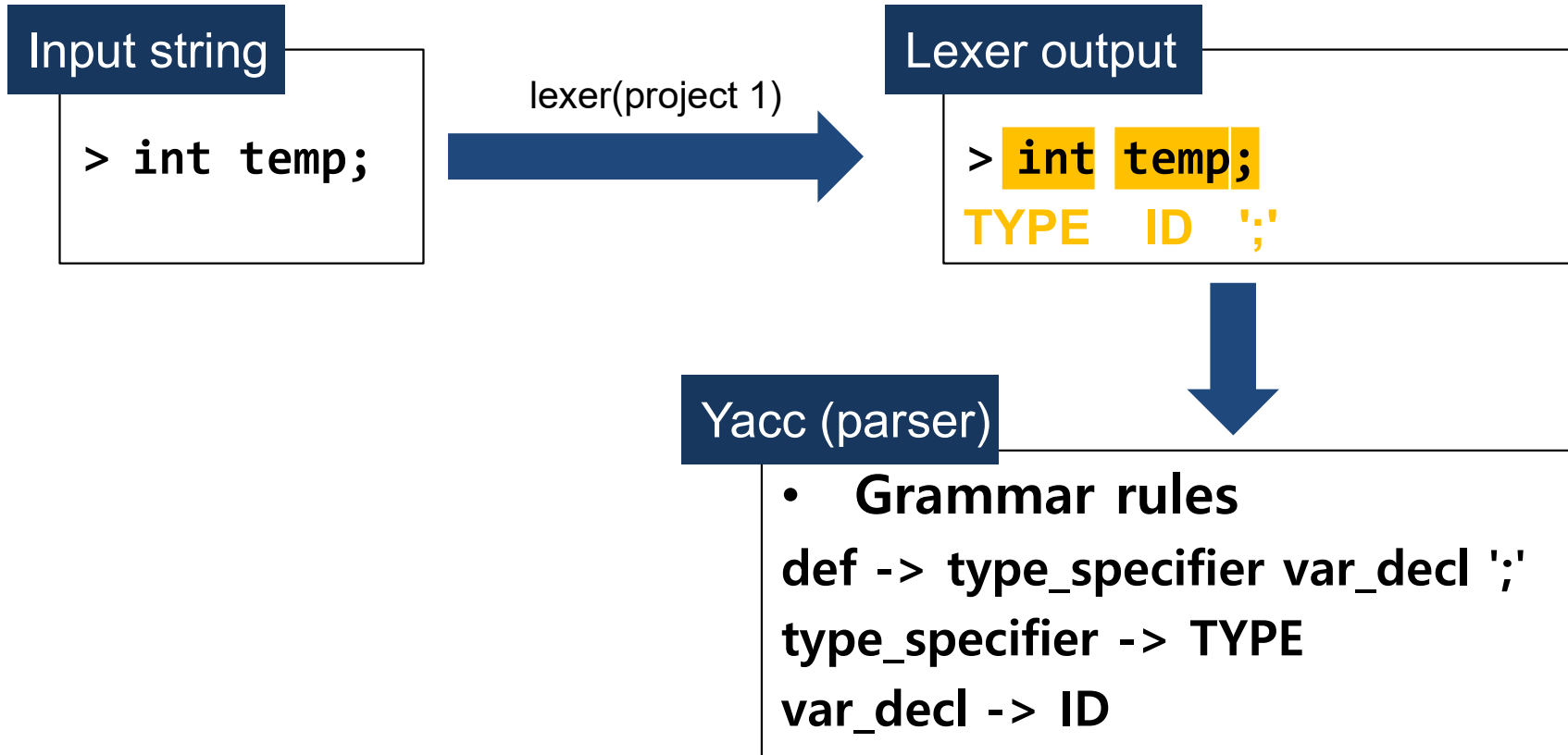
Yacc

- main함수의 yyparse()함수에 의해 시작
- token이 필요한 경우 lex의 yylex()함수를 호출해서 token 하나를 받는다.
- 입력받은 token에 대해서 문법 체크를 하고 조건에 맞는 실행을 한다.

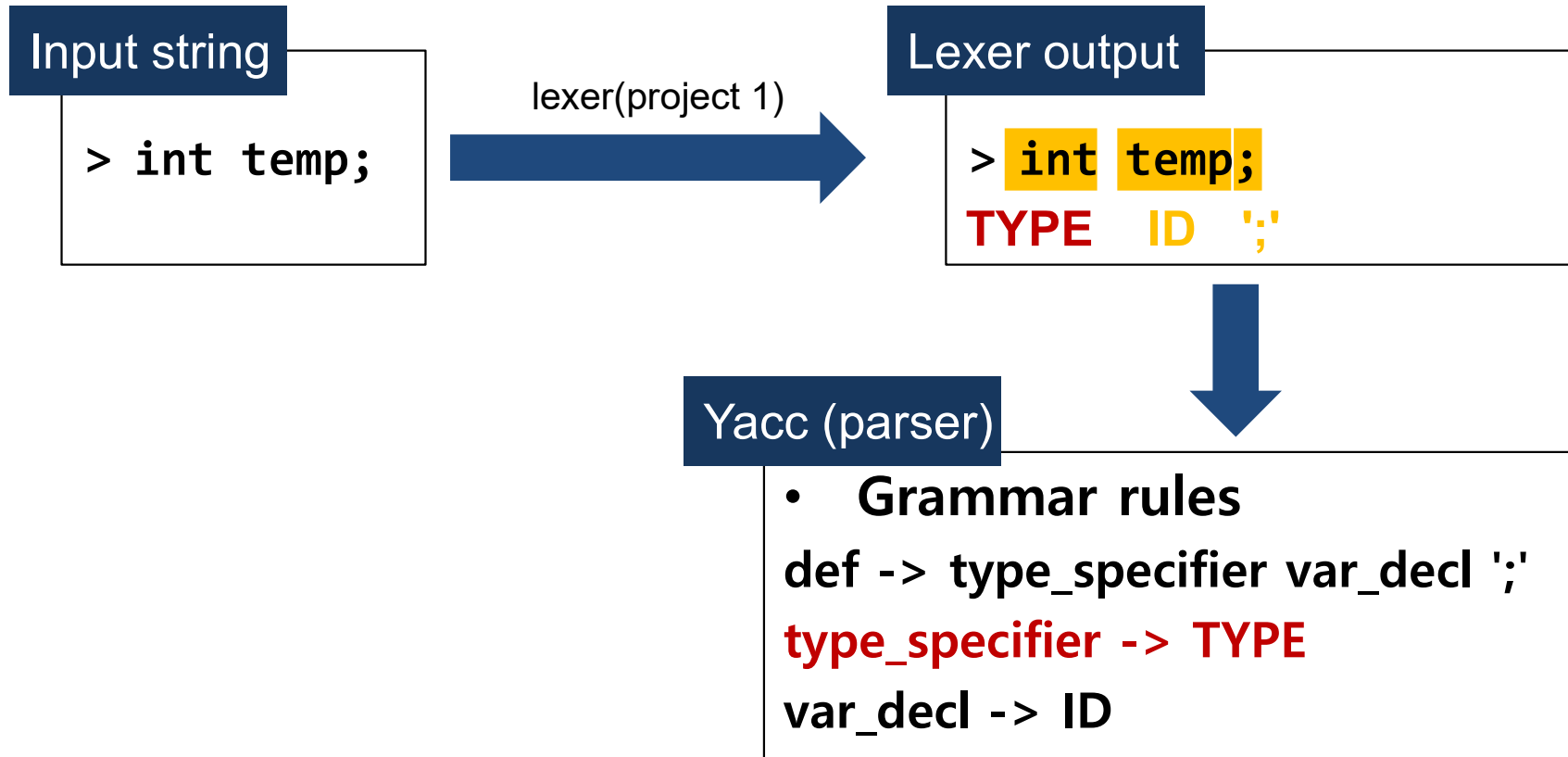
How Lex & YACC Works (Example)



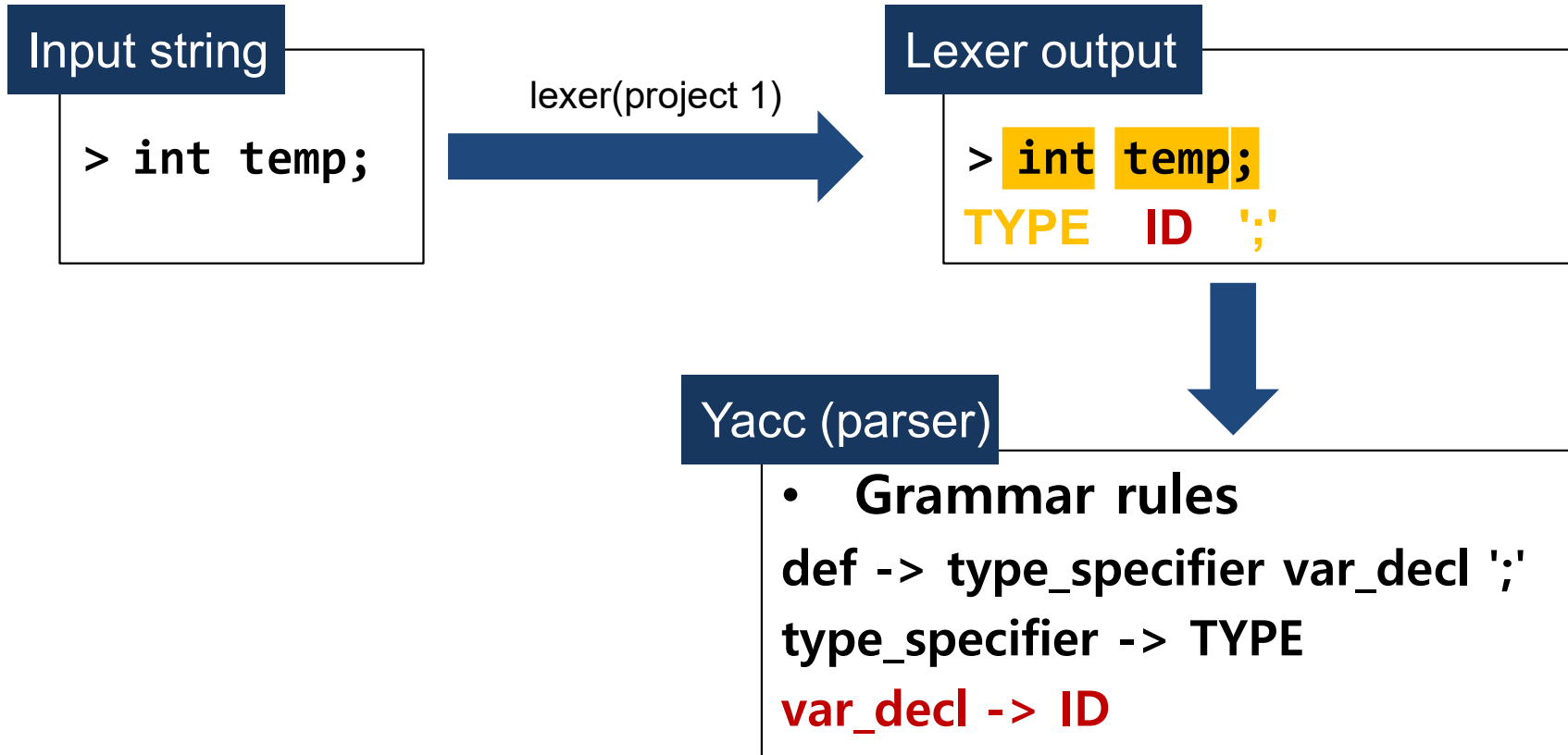
How Lex & YACC Works (Example)



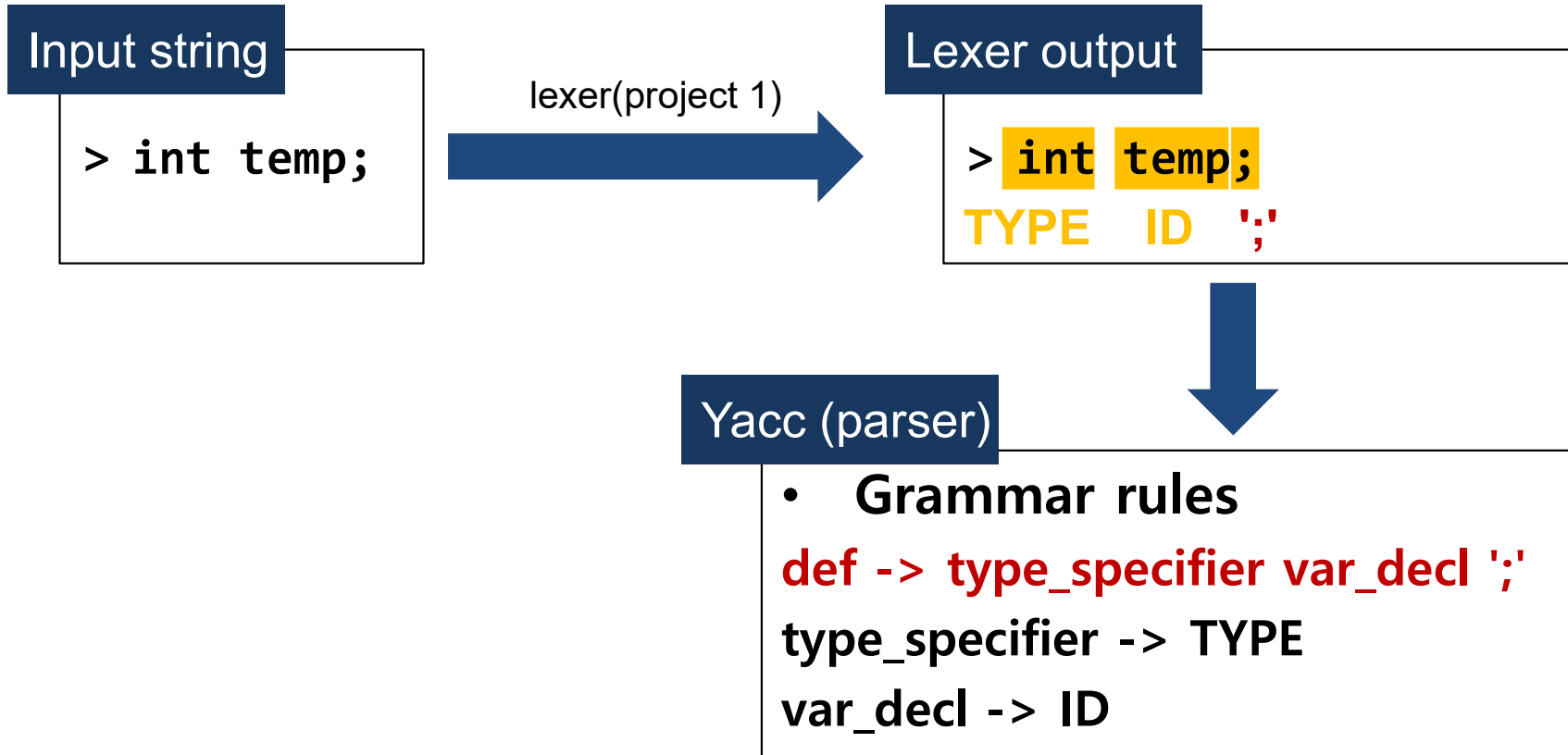
How Lex & YACC Works (Example)



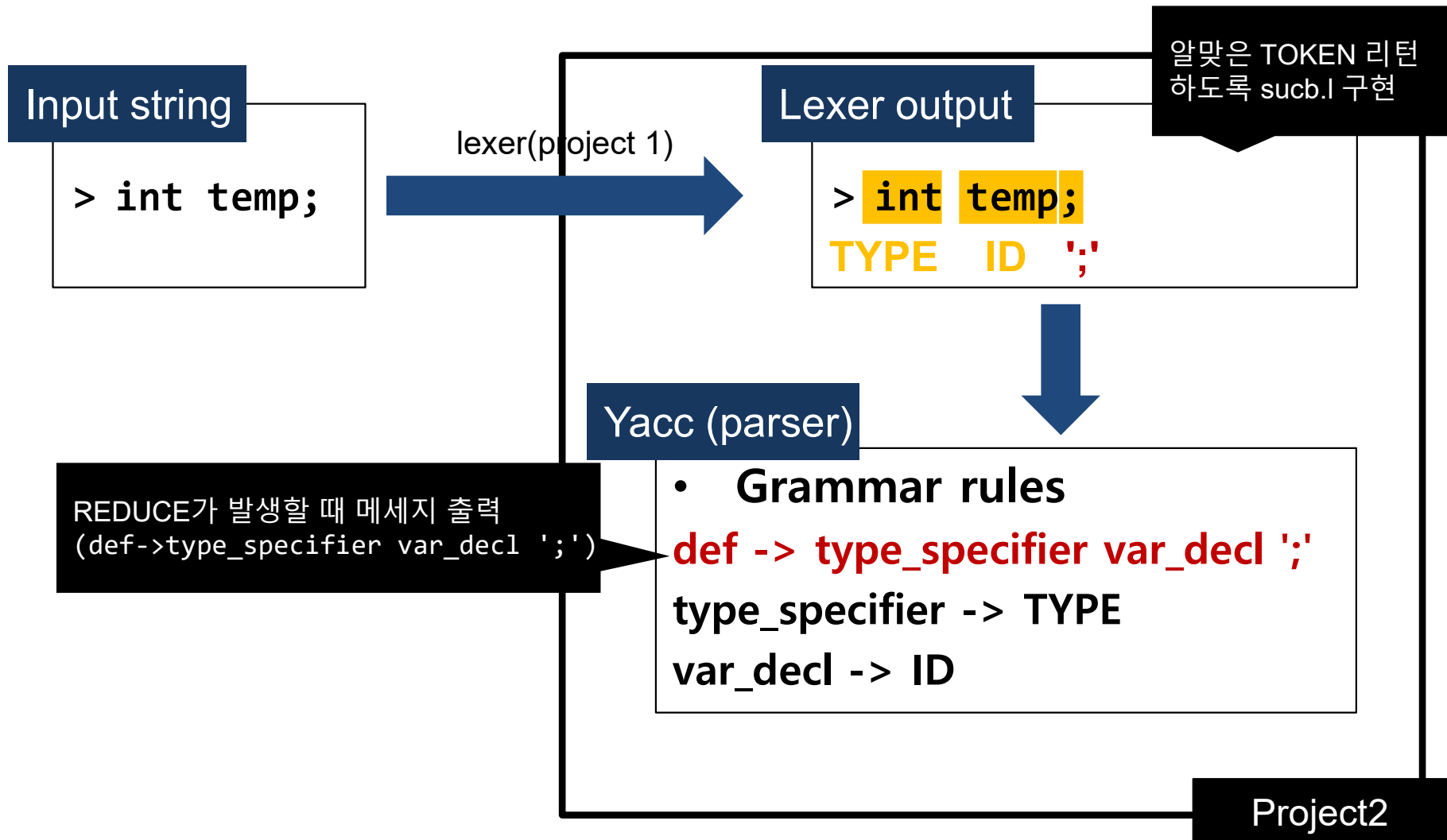
How Lex & YACC Works (Example)



How Lex & YACC Works (Example)



How Lex & YACC Works (Example)



How Lex & YACC Works

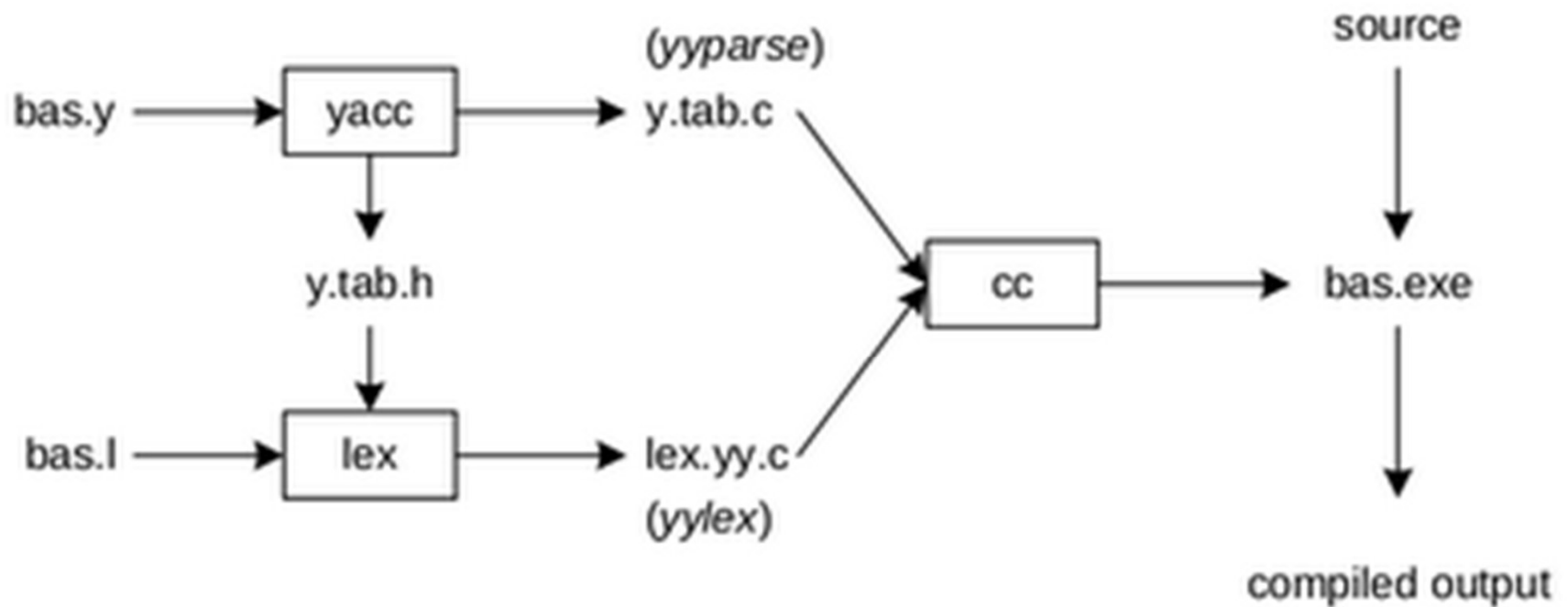


Figure 2: Building a Compiler with Lex/Yacc

Structure of Bison

Prologue

- #include, function declaration, typedef
- Literally copied into the generated parser.

Bison declarations

- Terminal (tokens) and nonterminal symbols
- Associativity & Precedence
- union 선언

Grammar rules

- Parser에서 사용할 grammar를 선언

Epilogue

- Auxiliary C codes
- Literally copied into the generated parser.

```
%{  
  
/* Prologue */  
  
%}  
  
/* Bison declarations */  
  
%%  
  
/* Grammar rules */  
  
%%  
  
/* Epilogue */
```



BISON SYMBOLS

Token

Lexer 는 bison 에게 token (terminal symbol) 을 전달

- Token 은 lex action 의 리턴 값이다.
- 내부적으로는 정수 값으로 인코딩됨.
- 예시의 lex action 은 token "INTEGER_CONST" 를 bison 에게 전달한다.

```
<normal>{integer_const} {  
    yylval.intVal = atoi(yytext);  
    return INTEGER_CONST;  
}
```

Value of a Token

yylval

- 각 token 은 yylval 이라는 변수로 참조 가능한 값을 가질 수 있음.
- 예시의 lex action 은 실제로 읽은 정수의 값을 token "INTEGER_CONST" 의 값으로 지정하여 bison 에게 전달한다.

```
<normal>{integer_const} {  
    yylval.intVal = atoi(ytext);  
    return INTEGER_CONST;  
}
```

Union

- Yylval 을 union type 으로 선언하면 다양한 타입의 값을 가질 수 있음.
- Union 은 bison declarations 에 선언함.

```
%union {  
    int      intVal;  
    char     *stringVal;  
}
```

Symbols

Token(terminal) 과 nonterminal 은 각자의 값을 가진다.

- 둘을 합쳐서 symbol 이라고 칭함.
- Terminal : Lexer 가 pattern matching 시 yylval 에 넣은 값을 가짐.
- nonterminal : Bison 의 reduce 과정에서 nonterminal 에 넣은 값을 가짐.

Defining the symbols

- Bison declarations section 에 정의
- Terminal symbols 는 **%token** modifier 로 정의

```
%token<intVal> INTEGER_CONST      /* Tokens with int type */
%token          TYPE              /* Tokens without a type */
```

- nonterminal symbols 는 **%type** modifier 로 정의

```
%type<stringVal> expr /* nonterminal symbols with char* type */
```



BISON GRAMMAR

Grammar Rules

Example1

```
statement : NAME '=' expression
expression: NUMBER '+' NUMBER
           | NUMBER '-' NUMBER
```

Example2 - recursive grammar

```
expression: NUMBER
           | expression '+' NUMBER
           | expression '-' NUMBER
```

주어진 입력 문자열 parsing이 실패할 경우 yyerror() 함수가 호출

자세한 정보는 아래 링크 참조:

https://www.gnu.org/software/bison/manual/html_node/Rules-Syntax.html

Actions

주어진 **grammar rule** 을 만족해서 **reduce**가 일어날 때 수행하는 **C코드**

- Grammar rule 의 action 에서 nonterminal 의 값을 정할 수 있음
- Value of LHS: \$\$
- Value of RHS: terminal, nonterminal의 순서에 따라 \$1, \$2, \$3 ...

Example

```
➤ date: month '/' date '/' year  
    { printf("date found"); };
```

```
➤ date: month '/' date '/' year  
    { $$ = $1 + $3 + $5;  
      printf("date %d-%d-%d found", $1, $3, $5); };
```

이번 프로젝트에서는 어떤 **grammar rule**에서 **reduce**가 발생했는지를 화면에 출력하는 코드 삽입

Midrule Actions

Grammar rule의 중간에 C코드를 삽입할 수 있음

- 중간에 nonterminal symbol 을 하나 삽입하는 효과를 냄.

Example

- `thing: A { printf("seen an A"); } B;`
- `thing: A fake_name B;`
`fake_name: %empty { printf("seen an A"); };`

Symbol value 값을 사용할 수 있음

- `thing: A { $$ = 17; } B C { printf("%d", $2); };`

자세한 정보는 아래 링크 참조:

https://www.gnu.org/software/bison/manual/html_node/Midrule-Actions.html



PRECEDENCE AND ASSOCIATIVITY

Associativity

각 operator 및 token에 대해 associativity를 설정

- Left or right associative: %left, %right
- Non-associative : %nonassoc, %precedence

```
%token NUMBER
%left '+'
%%
expr : expr '+' expr
      | NUMBER
      ;
```

```
%token NUMBER
%right '+'
%%
expr : expr '+' expr
      | NUMBER
      ;
```

expr + expr + expr



(expr + expr) + expr

Left associative



expr + (expr + expr)

Right associative

Precedence

Bison declaration 에서 아래에 쓰여진 것일수록 우선순위가 증가

- %left, %right, %nonassoc, %precedence 만 해당
 - %nonassoc : run-time error (error if an operator is used in an associative way)
 - %precedence: compile-time error (associativity-related conflict is prohibited)

Contextual precedence

- Grammar 에 %prec modifier 를 활용해서 우선순위 조작 가능.

```
%token NUMBER
%left '+' '-'
%left '*'
%%
expr : expr '+' expr %prec '*'
    | expr '-' expr
    | NUMBER
    ;
```



CONFLICTS IN GRAMMAR

Conflicts

Shift-Reduce conflicts

➤ grammar

```
e: 'X'  
    | e '+' e ;
```

➤ input

X + X + X

Reduce-Reduce conflicts

➤ grammar

```
prog:  proga | progb ;  
proga: 'X' ;  
progb: 'X' ;
```

➤ input

X

Ambiguity of grammar

```
expression      : expression '+' expression
                  | expression '-' expression
                  | expression '*' expression
                  | expression '/' expression
                  | '-' expression
                  | '(' expression ')'
                  | NUMBER
                  ;
```

Parsing Issues

➤ $1 + 2 * 3$

- $(1 + 2) * 3$ or $1 + (2 * 3)$

Ambiguity of grammar - Solution1

여러 개의 production 으로 나눠서 reduce 의 순서를 지정하기

```
expression      : expression '+' mulexp
                  | expression '-' mulexp
                  | mulexp
                  ;
```

```
mulexp          : mulexp '*' primary
                  | mulexp '/' primary
                  | primary
                  ;
```

```
primary         : '(' expression ')'
                  | '-' primary
                  | NUMBER
                  ;
```

Ambiguity of grammar - Solution2

Precedence와 associativity를 활용

➤ Bison declaration

```
%token NUMBER
%left '-' '+'
%left '*' '/'
%nonassoc UMINUS
```

➤ Grammar rules

```
expression : expression '+' expression
           | expression '-' expression
           | expression '*' expression
           | expression '/' expression
           | '-' expression %prec UMINUS
           | '(' expression ')'
           | NUMBER
           ;
```



PROJECT GOAL

Lexer & Parser

Lexer

- subC 의 토큰을 읽고 parser 에게 적절한 정보를 넘길 수 있는 lexer 구현
 - Token을 리턴하도록 변경
 - yylval 에 적당한 값을 전달하기
- 이전 프로젝트에 이어서 구현해도 되고, 새로 주어진 스켈레톤을 써도 됨
- 기존에 구현한 해시 테이블은 필요에 따라 직접 판단하여 사용

Parser

- Bison declarations section
 - Union type 정의하기
 - Symbol 및 operator precedence & associativity 정의하기
- Grammar rules section
 - subC 의 grammar 에 맞는 rule 작성하기
 - Reduce 발생 시 사용된 production 을 stdout 에 출력하도록 action 정의

Output Example

- 주어진 **예제 테스트 코드**(test.c , test_result) 와 같은 형식으로 출력
- 테스트 방법 예시) ./subc test.c > result

```
int main(){
    int a;
    char b;

    a = 10;
    b = 5;

    if (a==10 || b==5) {
        return 1;
    } else {
        return 0;
    }
}
```



```
$ ./subc ../examples/ex_format/test.c
ext_def_list->epsilon
type_specifier->TYPE
pointers->epsilon
func_decl->type_specifier pointers ID '(' ')'
def_list->epsilon
type_specifier->TYPE
pointers->epsilon
def->type_specifier pointers ID ';'
def_list->def_list def
type_specifier->TYPE
pointers->epsilon
def->type_specifier pointers ID ';'
def_list->def_list def
stmt_list->epsilon
unary->ID
unary->INTEGER_CONST
binary->unary
expr->binary
```

Input (./examples/test.c)

Output (./examples/test_result)

Report

subC grammar rule 은 conflict 가 발생하도록 설계됨

- 발생한 conflict를 해결하고, conflict의 종류 및 해결 방법을 **보고서**로 작성
- 분량은 2장 이내로, 간단하게 작성

Hint

- bison으로 subc.y 파일을 컴파일하면 subc.output 파일이 생성됨
 - `bison -vd subc.y`
 - subc.output 파일을 열면 conflict에 대한 정보를 얻을 수 있음
- Bison 의 `-Wcounterexamples` 옵션 활용

Tips

yytext

- yytext 는 현재 읽고 있는 파일의 포인터이므로 그대로 주소를 복사해서 사용한다면 Token 이후의 string도 가져오게 된다.
- yytext 를 복사할 때 malloc 을 사용해 메모리를 할당 받은 후에 strcpy 함수를 사용하여 필요한 길이 만큼을 복사하면 더 안전함.

```
{letter}* {  
    printf("String %s\n",yytext);  
}  
{num}* {  
    printf("Number %d\n",atoi(y  
    ytext));  
}
```

[Input]
Compiler12

[Output]
String Compiler12
Number 12

Submission

제출 기한

- Oct 30, 2024

제출 방법

- etl.snu.ac.kr을 통해서 제출

제출 파일

- 'src' directory 안의 파일들과 'report.pdf' 를 제출
- report.pdf 를 project2 아래에 복사한 후 submit.sh 로 압축
 - project2 의 subdirectory 도 인식할 수 있으니 아무 곳에도 넣어도 됨.
 - Project container 안에서 ./submit.sh xxxx-xxxxx 실행
- Archive 의 파일 이름 확인
 - project2_학번.zip (학번 format은 20xx-xxxxx)

Notice

수업 게시판 확인

- 수정 또는 추가되는 사항은 항상 게시판을 통하여 공지
- 제출 마지막날까지 공지된 사항을 반영해서 제출

소스코드에 자세히 주석 달기

제출 형식 지키기 (파일 이름, 출력)

- 지키지 않을 시 감점

Cheating 금지 (F처리, 모든 코드 철저히 검사)

TA

- 이상현
- E-mail : sanghyeon@snu.ac.kr