


Parser_YACC 실습

강의동영상

- 5/3 실습1 (1)
- 5/6 실습2 (2)
- 과제3 (3)
- 5/10 실습3 (3)
- 5/13 실습4 , 학기말시험 (4)



5/3 실습1 (팀 실습)

- 제출일: 5/3 24:00
- 실습결과 제출로 출석 확인
- 결과가 틀리면 실습점수 감점.

실습1: YACC 익히기

- 교재 14장 문제 14.4 cal.l cal.y를 작성하여 flex와 bison으로 실행한다. 단 main.c 와 reporterror.c 파일로 분리한다. (참고: 교재에 오타 있음)

```

* cal.l:
%{
/* Lex source for calculator program */
%}
%%
[a-z]*      {return NAME;}
[0-9]*      {yyval = atoi(yytext); return NUMBER;}
[ ' \t ]    :
"\n"       : return 0;
            : return yytext[0];
%%
int yywrap()
{
    return 1;
}

```

cal.l

Include 추가
 #include <stdio.h>
 #include <stdlib.h>
 #include "tn.h"

```

* cal.y:
%{
/* YACC source for calculator program */
%}
%token NAME NUMBER
%left '-' '+'
%left '*' '/'
%nonassoc UMINUS

%%
statement: NAME '=' expression
        { expression { printf("%d\n", $1); } }

expression: expression '+' expression { $$ = $1 + $3; }
          | expression '-' expression { $$ = $1 - $3; }
          | expression '*' expression { $$ = $1 * $3; }
          | expression '/' expression
            { if ($3 == 0) yyerror("divide by zero");
              else $$ = $1 / $3; }
          | '-' expression %prec UMINUS { $$ = -$2; }
          | '.' expression '/' { $$ = $1; }
          | NUMBER

```

cal.y

Include 추가
 #include <stdio.h>
 #include <ctype.h>
 #include <malloc.h>

오타교정: \$1 / \$3

#include <stdio.h>

```

#include <stdio.h>
void yyerror(char *s)
{
    printf("%s\n", s);
}

```

reporterror.c

```

int main()
{
    return yyparse();
}

```

main.c

#include <stdio.h>
 extern int yyparse();

삭제

실습1: YACC 익히기

- 교재 p608 문제 14.4 cal.l cal.y를 작성하여 flex와 bison으로 실행한다. 단 main.c 와 reporterror.c 파일로 분리한다. (참고: 교재에 오타 있음)
 - 1) flex cal.l
 - 2) bison cal.y *~ 열거보기*
 - 3) 생성된 cal_tab.c 파일 검토

이름	수정한 날짜	유형	크기
flex	2002-09-24 오후 3:44	응용 프로그램	177KB
bison	2002-11-06 오후 1:21	응용 프로그램	192KB
bison.hairy	2002-11-06 오후 1:22	HAIRY 파일	7KB
bison.simple	2002-11-06 오후 1:22	SIMPLE 파일	18KB
main	2002-11-06 오후 5:35	C 파일	1KB
ReportError	2007-11-10 오전 9:58	C 파일	1KB
cal	2016-05-02 오후 2:10	L 파일	1KB
cal	2016-05-13 오전 10:04	Y 파일	1KB

text 파일.
확장자 변경

C:\Users\user\Desktop\실습1>flex cal.l

flex 실행

C:\Users\user\Desktop\실습1>bison cal.y

bison 실행

C:\Users\user\Desktop\실습1>dir

C 드라이브의 볼륨: System
볼륨 일련 번호: 34DB-181B

C:\Users\user\Desktop\실습1 디렉터리

```

2020-05-03 오전 11:25 <DIR> .
2020-05-03 오전 11:25 <DIR> ..
2002-11-06 오후 01:21 196,096 bison.exe
2002-11-06 오후 01:22 6,811 bison.hairy
2002-11-06 오후 01:22 17,775 bison.simple
2016-05-02 오후 02:10 302 cal.l
2016-05-13 오전 10:04 674 cal.y
2020-05-03 오전 11:25 23,126 cal_tab.c
2002-09-24 오후 03:44 181,248 flex.exe
2020-05-03 오전 11:24 37,005 lex.yy.c
2002-11-06 오후 05:35 76 main.c
2007-11-10 오전 09:58 69 ReportError.c
10개 파일 463,182 바이트
  
```

생성됨

} → 총 4개의 C파일 → vs code로 컴파일

실습1 : YACC 익히기

- 교재 p608 문제 14.4 cal.l cal.y를 작성하여 flex와 bison으로 실행한다. 단 main.c 와 reporterror.c 파일로 분리한다. (참고: 교재에 오타 있음)

- 1) flex cal.l
- 2) bison cal.y
- 3) 생성된 cal_tab.c 파일 검토
- 4) tn.h 작성

- 참고

- tn.h 생성법

C: \> bison -d cal.y 자동으로

cal_tab.c, cal_tab.h 가 생성됨

C: \> copy cal_tab.h tn.h 파일명 변경

❖ 한번 tn.h 가 만들어 졌으면, 그 다음에는 더 이상 -d 를 할 필요 없다

cal.l

lex.yy.c

```

* cal.l:
%{
/* Lex source for calculator program */
%}
%%
[a-z]*      {return NAME;}
[0-9]*      {yyival = atoi(yytext); return NUMBER;}
[ \t]*      :
"\n"       : return 0;
            : return yytext[0];
%%
int yywrap()
{
    return 1;
}

```

Include 추가

#include <stdio.h>

#include <stdlib.h>

#include "tn.h"

작성해야함!

cal.y

cal.tab.c

```

* cal.y:
%{
/* YACC source for calculator program */
%}
%token NAME NUMBER
%left '-' '+'
%left '*' '/'
%nonassoc UMINUS

%%
statement: NAME '=' expression
        { expression { printf("%d\n", $1); } }

expression: expression '+' expression { $$ = $1 + $3; }
          | expression '-' expression { $$ = $1 - $3; }
          | expression '*' expression { $$ = $1 * $3; }
          | expression '/' expression
            { if ($3 == 0) yyerror("divide by zero");
              else $$ = $1 / $3; }
          | '-' expression %prec UMINUS { $$ = -$2; }
          | '.' expression '/' { $$ = $1; }
          | NUMBER

```

Include 추가

#include <stdio.h>

#include <ctype.h>

#include <malloc.h>

오타교정: \$1 / \$3

#include <stdio.h>

삭제

reporterror.c

```

void yyerror(char *s)
{
    printf("%s\n", s);
}

```

#include <stdio.h>
extern int yyparse();

main.c

```

int main()
{
    return yyparse();
}

```


C:\Users\User\Desktop\실습1>bison -d cal.y

bison -d cal.y 실행

C:\Users\User\Desktop\실습1>dir

C 드라이브의 볼륨: System
볼륨 일련 번호: 34DB-181B

C:\Users\User\Desktop\실습1 디렉터리

```
2020-05-03 오후 12:05 <DIR> .
2020-05-03 오후 12:05 <DIR> ..
2022-11-06 오후 01:21      196,096 bison.exe
2022-11-06 오후 01:22       6,811 bison.hairy
2022-11-06 오후 01:22      17,775 bison.simple
2016-05-02 오후 02:10       302 cal.l
2016-05-13 오전 10:04       674 cal.y
2020-05-03 오후 12:05     23,126 cal_tab.c
2020-05-03 오후 12:05      132 cal_tab.h
2022-09-24 오후 03:44     181,248 flex.exe
2020-05-03 오전 11:24     37,005 lex.yy.c
2022-11-06 오후 05:35        76 main.c
2020-11-10 오전 09:58         69 ReportError.c
11개 파일              463,314 바이트
2개 디렉터리    197,947,744,256 바이트 남음
```

한번 tn.h 을 확보했으면,
다음 또 bison을 실행할
때에는 -d 를 할 필요없다.
즉, 그냥 bison cal.y 하면 됨.

tn.h 로 파일명 변경

이름	수정한 날짜	유형	크기
flex	2002-09-24 오후 3:44	응용 프로그램	177KB
bison	2002-11-06 오후 1:21	응용 프로그램	192KB
bison.hairy	2002-11-06 오후 1:22	HAIRY 파일	7KB
bison.simple	2002-11-06 오후 1:22	SIMPLE 파일	18KB
main	2002-11-06 오후 5:35	C 파일	1KB
ReportError	2007-11-10 오전 9:58	C 파일	1KB
cal	2016-05-02 오후 2:10	L 파일	1KB
cal	2016-05-13 오전 10:04	Y 파일	1KB
lex.yy	2020-05-03 오전 11:24	C 파일	37KB
cal_tab	2020-05-03 오후 12:05	C 파일	23KB
tn.h	2020-05-03 오후 12:05	H 파일	1KB

실습1 : YACC 익히기

- 교재 p608 문제 14.4 cal.l cal.y를 작성하여 flex와 bison으로 실행한다. 단 main.c 와 reporterror.c 파일로 분리한다. (참고: 교재에 오타 있음)
 - 1) flex cal.l
 - 2) bison cal.y
 - 3) 생성된 cal_tab.c 파일 검토
 - 4) tn.h 작성
 - 5) lex.yy.c cal_tab.c main.c reporterror.c 파일을 “MS Visual Studio”로 컴파일, 실행한 후 결과화면을 캡처하여 제출. (주의:학번 이름도 출력되도록 하시오!)

- 참고

- tn.h 생성법

- C: \> bison -d cal.y

- cal_tab.c, cal_tab.h 가 생성됨

- C: \> copy cal_tab.h tn.h

- ❖ 한번 tn.h 가 만들어 졌으면, 그 다음에는 더 이상 -d 를 할 필요 없다

실행 결과 예시

```
C:\WINDOWS\system32\cmd.exe
2-5
= -3
계속하려면 아무 키나 누르십시오 . . .
```

결과 출력 시 본인의 학번, 이름도
함께 출력되도록 수정할 것

또다른 실행 결과 예시들

결과가 어떻게 해서 나왔는지 cal.l, cal.y 를 공부

```
C:\WINDOWS\system32\cmd.exe
5/0
divide by zero
= 5
계속하려면 아무 키나 누르십시오 . . .
```

```
C:\WINDOWS\system32\cmd.exe
15/1+2
= 17
계속하려면 아무 키나 누르십시오 . . .
```

```
C:\WINDOWS\system32\cmd.exe
15/(1+2)
= 5
계속하려면 아무 키나 누르십시오 . . .
```

```
C:\WINDOWS\system32\cmd.exe
15/1+
parse error
계속하려면 아무 키나 누르십시오 . . .
```

5/6 실습2 (팀 실습)

- 제출일: 5/6 24:00
- 제출물: parser.y, 결과화면캡처 (팀원, 학번도 출력되게)
- 실습결과 제출로 출석 확인
(참여하지 않은 팀원이 있을 경우, 조교에게 보고)
- 결과가 틀리면 실습점수 감점.

실습2: 교과서의 miniC

- 교과서 14장의 Mini C <parser.y> 을 bison으로 실행 한다. (사캠에 교재및 parser.y)
사캠에 있음
 - 교과서 4장의 scanner 프로그램 <MiniC.l> 을 flex로 실행한다. (사캠에 교재및 MiniC.l)
- 1) bison parser.y *→ ambiguous*
- 만일 Shift/Reduce Error가 나온다면, 이를 없앨 수 있도록 의도한 문법은 그대로 둔 채 yacc 표현방식을 바꾸어 고쳐야 함.
- (방법: *** 중요!*)
- 실제 교재의 MiniC 문법에는 1 shift/reduce error가 있다.
 - `bison -v parser.y` → <parser.out> 파일에서 원인 파악

이름	수정한 날짜	유형	크기
bison	2002-11-06 오후 1:21	응용 프로그램	192KB
bison.hairy	2002-11-06 오후 1:22	HAIRY 파일	7KB
bison.simple	2002-11-06 오후 1:22	SIMPLE 파일	18KB
flex	2002-09-24 오후 3:44	응용 프로그램	177KB
parser	2020-05-03 오후 1:20	Y 파일	6KB

```
C:\Users\User\Desktop\실습2>bison parser.y
```

```
parser.y contains 1 shift/reduce conflict.
```

1 shift/reduce 발생

```
C:\Users\User\Desktop\실습2>bison -v parser.y
```

```
parser.y contains 1 shift/reduce conflict.
```

이러보기

bison -v parser.y 실행

```
C:\Users\User\Desktop\실습2>dir
```

```
C 드라이브의 볼륨: System
```

```
볼륨 일련 번호: 34DB-181B
```

```
C:\Users\User\Desktop\실습2 디렉터리
```

```

2020-05-03 오후 01:24 <DIR>
2020-05-03 오후 01:24 <DIR>
2002-11-06 오후 01:21 196,096 bison.exe
2002-11-06 오후 01:22 6,811 bison.hairy
2002-11-06 오후 01:22 17,775 bison.simple
2002-09-24 오후 03:44 181,248 flex.exe
2020-05-03 오후 01:24 56,566 parser.output
2020-05-03 오후 01:20 5,378 parser.y
2020-05-03 오후 01:24 38,506 parser_tab.c
7개 파일 502,380 바이트
2개 디렉터리 198,010,089,472 바이트 남음

```

이 파일 보고
디버깅

State 146 contains 1 shift/reduce conflict.

<parser.out>

Grammar

rule 1 mini_c -> translation_unit

rule 2 translation_unit -> external_dcl

rule 3 translation_unit -> translation_unit external_dcl

rule 4 external_dcl -> function_def

rule 5 external_dcl -> state 145

rule 6 function_def -> postfix_exp -> postfix_exp '[' expression ']' . (rule 86)

rule 7 function_def -> \$default reduce using rule 86 (postfix_exp)

rule 8 dcl_spec -> \$default reduce using rule 86 (postfix_exp)

rule 9 dcl_spec -> \$default reduce using rule 86 (postfix_exp)

rule 10 dcl_spec -> \$default reduce using rule 86 (postfix_exp)

rule 11 dcl_spec -> \$default reduce using rule 86 (postfix_exp)

rule 12 dcl_spec -> \$default reduce using rule 86 (postfix_exp)

rule 13 type_spec -> \$default reduce using rule 86 (postfix_exp)

rule 14 type_spec -> \$default reduce using rule 86 (postfix_exp)

rule 15 type_spec -> \$default reduce using rule 86 (postfix_exp)

rule 16 function_def -> \$default reduce using rule 86 (postfix_exp)

rule 17 formal_param_list -> \$default reduce using rule 86 (postfix_exp)

rule 18 opt_formal_param_list -> \$default reduce using rule 86 (postfix_exp)

rule 19 opt_formal_param_list -> \$default reduce using rule 86 (postfix_exp)

rule 20 formal_param_list -> \$default reduce using rule 86 (postfix_exp)

rule 21 formal_param_list -> \$default reduce using rule 86 (postfix_exp)

rule 22 param_list -> \$default reduce using rule 86 (postfix_exp)

rule 23 compound_statement -> \$default reduce using rule 86 (postfix_exp)

rule 24 opt_dcl -> \$default reduce using rule 86 (postfix_exp)

state 146

if_st -> tif '(' expression ')' statement . (rule 49)

if_st -> tif '(' expression ')' statement telse statement (rule 50)

telse shift, and go to state 149

telse [reduce using rule 49 (if_st)]

\$default reduce using rule 49 (if_st)

이것이
ambiguous
↓
1

state 147

while_st -> twwhile '(' expression ')' statement . (rule 51)

rule 48 opt_expression -> /* empty */

rule 49 if_st -> tif '(' expression ')' statement

rule 50 if_st -> tif '(' expression ')' statement telse statement

rule 51 while_st -> twwhile '(' expression ')' statement

rule 52 return_st -> treturn opt_expression ';' ;

rule 53 expression -> assignment_exp

← parser.y 파일

실습2: 교과서의 miniC

- 교과서 p579의 Mini C <parser.y> 을 bison으로 실행 한다. (사캠에 교재및 parser.y)
- 교과서 pp162의 scanner 프로그램 <MiniC.l> 을 flex로 실행한다.(사캠에 교재및 MiniC.l)

1) bison parser.y

- 만일 Shift/Reduce Error가 나온다면, 이를 없앨 수 있도록 의도한 문법은 그대로 둔 채 yacc 표현방식을 바꾸어 고쳐야 함.

(방법: 가상터미널로 우선순위교정(실습2), 혹은 keyword 추가하여 문법 교정)

- 실제 P579의 MiniC 문법에는 1 shift/reduce error가 있다.
- bison -v parser.y → <parser.out> 파일에서 원인 파악

2) parser_tab.c main.c reporterror.c, lex.yy.c, tn.h 등 파일을 컴파일한다.

```
bison -d parser.y
      parser_tab.c, parser_tab_h
copy parser_tab.h tn.h
```


실습2: 교과서의 miniC

- 교과서 p579의 Mini C <parser.y> 을 bison으로 실행 한다. (사캠에 교재 및 parser.y)
- 교과서 pp162의 scanner 프로그램 <MiniC.l> 을 flex로 실행한다.(사캠에 교재 및 MiniC.l)

1) bison parser.y

- 만일 Shift/Reduce Error가 나온다면, 이를 없앨 수 있도록 의도한 문법은 그대로 둔 채 yacc 표현방식을 바꾸어 고쳐야 함.

(방법: 가상터미널로 우선순위교정(실습2), 혹은 keyword 추가하여 문법 교정)

→ 추천! 이론에 있던거. 최대한 이점

- 실제 P579의 MiniC 문법에는 1 shift/reduce error가 있다.
- bison -v parser.y → <parser.out> 파일에서 원인 파악

→ 알뜰 들여서 나오는 거

2) parser_tab.c main.c reporterror.c, lex.yy.c, tn.h 등 파일을 컴파일한다.

bison -d parser.y

parser_tab.c, parser_tab.h

copy parser_tab.h tn.h

vs code로 실행

- ### 3) 입력 데이터인 MiniC 프로그램 <prime.mc> 로 실행하여 결과를 확인후, 교정한 parser.y, 결과화면 캡처 (팀원, 학번도 출력되게) 제출

실행 결과

(이어서)

```
C:\WINDOWS\system32\cmd.exe
reduced rule number = 13
reduced rule number = 11
reduced rule number = 9
reduced rule number = 14
reduced rule number = 12
reduced rule number = 10
reduced rule number = 8
reduced rule number = 33
reduced rule number = 32
reduced rule number = 29
reduced rule number = 28
reduced rule number = 5
reduced rule number = 2
reduced rule number = 15
reduced rule number = 12
reduced rule number = 9
reduced rule number = 8
reduced rule number = 16
reduced rule number = 19
reduced rule number = 17
reduced rule number = 7
reduced rule number = 14
reduced rule number = 12
reduced rule number = 9
reduced rule number = 8
reduced rule number = 33
reduced rule number = 31
reduced rule number = 29
reduced rule number = 33
reduced rule number = 31
reduced rule number = 30
reduced rule number = 33
reduced rule number = 31
reduced rule number = 30
reduced rule number = 28
reduced rule number = 26
reduced rule number = 14
reduced rule number = 12
reduced rule number = 9
reduced rule number = 8
reduced rule number = 33
reduced rule number = 31
reduced rule number = 29
reduced rule number = 33
reduced rule number = 31
reduced rule number = 30
reduced rule number = 28
reduced rule number = 27
reduced rule number = 24
reduced rule number = 95
reduced rule number = 85
reduced rule number = 80
```

```
C:\WINDOWS\system32\cmd.exe
reduced rule number = 80
reduced rule number = 76
reduced rule number = 73
reduced rule number = 68
reduced rule number = 65
reduced rule number = 63
reduced rule number = 61
reduced rule number = 54
reduced rule number = 53
reduced rule number = 47
reduced rule number = 46
reduced rule number = 42
reduced rule number = 49
reduced rule number = 43
reduced rule number = 40
reduced rule number = 48
reduced rule number = 46
reduced rule number = 42
reduced rule number = 40
reduced rule number = 95
reduced rule number = 85
reduced rule number = 80
reduced rule number = 83
reduced rule number = 76
reduced rule number = 73
reduced rule number = 68
reduced rule number = 65
reduced rule number = 63
reduced rule number = 61
reduced rule number = 54
reduced rule number = 53
reduced rule number = 47
reduced rule number = 46
reduced rule number = 42
reduced rule number = 40
reduced rule number = 37
reduced rule number = 23
reduced rule number = 41
reduced rule number = 51
reduced rule number = 44
reduced rule number = 40
reduced rule number = 37
reduced rule number = 23
reduced rule number = 6
reduced rule number = 4
reduced rule number = 3
reduced rule number = 1
계속하려면 아무 키나 누르십시오 . . .
```

왜 이렇게
나왔는지
이해

결과 출력 시 팀원의
학번,이름도 함께
출력되도록 수정할 것

CHAPTER 14 컴파일러 자동화 도구

```

actual_param_list : assignment_exp          { semantic(93); }
                  | actual_param_list ',' assignment_exp { semantic(94); };
primary_exp      : tidint                    { semantic(95); }
                  | tnumber                  { semantic(96); }
                  | '(' expression ')'       { semantic(97); };
%%

```

```

#include "lex.yy.c" /* #include "lex.yy.c" */

```

삭제

```

void yyerror(char *s)
{
    printf("%s\n", s);
}

```

```

void semantic(int n)
{
    printf("reduced rule number = %d\n", n);
}

```

```

void main()
{
    printf("start of parser\n");
    yyparse();
    printf("end of parser\n");
}

```

reporterror.c

주의: File include해야함.

} 이거 우리가

main.c

주의: File include해야함.

여기서 사용한 Mini C 문법은 [부록 A]에 수록된 형태이다. 그리고 함수 semantic()에서 단순히 생성 규칙 번호만 프린트하여 우파스를 출력하였는데 필요한 코드 생성 작업을 수행하여 실질적인 전단부로 사용할 수 있다. 보통 여기에서 생성 규칙에 맞는 추상 구문 트리(AST)를 만드는 작업을 행한다.

2.2 스탠포드 파서 제작 시스템

스탠포드(Stanford) PGS는 스탠포드 대학의 John Henessy에 의해 개발된 파서 생성기로 문법 규칙에 추상-구문 트리(AST: Abstract Syntax Tree)를 자동으로 만들 수 있는 정보를 결합시킬 수 있다. PGS는 context-free 문법 형태로 된 입력을

-
-
-
-
-
-
-
-
-
-

과제3

컴파일리구성(2021,1학기)

Project 3 (Modified Mini-C Parser) Due 5/26일(수) 24시

(Modified) Mini C를 위한 Parser를 과제2에서 작성한 scanner프로그램을 이용하여 scanning한 token을 bison (YACC)을 사용하여 구문 분석하라.

[제출 내용]

- 1) 전체 소스코드를 Zip 하여 제출 → 조교가 직접 팀 별로 실행 평가 시 사용

Source codes: main.c, parser.y, scanner.l, symtable.c, reporter.c, tab, globb 등.

(주의사항: lex.yy.c 와 parser.tab.c 도 제출)

- 2) 입출력 내용: (test1.mc, 출력스캔), (test2.mc, 출력스캔 2), ... ↓

문법 오류 없는 것(4개 이상), 문법 오류 있는 것(4개 이상) 입출력을 Zip 하여 제출 ↓
 <*** Output 의 경우 과제물에 명확히 정의되지 않은 부분에 있어 본인에 정의한 것이 있을 경우 그 내용을 기술하고 해당 output 부분에 표시. ***>

[과제 내용]

- 1) scanner와 parser에서 발생한 모든 error에 대하여 가능한 한 의미 있는 error 메시지로써 출력하라.

(!!점수 반영됨!!)

예) line 3, 10 에서 error 발생했다고 가정했을 때,

```
***MiniC parsing begins
3          line 3 에서 발생한 error type 에 따른 message 출력
10         line 3 에서 발생한 error type 에 따른 message 출력
Parsing ends.
2 error(s) detected
```

만일 error 가 하나도 없이 구문 분석이 끝났을 경우는, 다음이 출력된다.

```
***MiniC parsing begins
Parsing ends.
0 error(s) detected
```

- 2) 변수(즉, Identifier)가 선언되었을 때, 선언된 위치의 라인 넘버를 HStab 에 저장하라. 구문분석이 끝나면, Hstab 에 저장된 모든 identifier 에 대한 속성(예, identifier character string, its type, scalar/array 변수인지 함수인지 함수 파라미터인지, return type(함수명일 경우), line number)을 과제 1 에서처럼 출력하라.

- 출력 양식은 자유.
- Identifier 의 정의는 과제 2 와 동일함.

- scanner.l 에서 ID 인 경우 character string 을 HStab 에 저장(즉, ST 의 인덱스를 저장)하고, Parser.y 에서 ID type 을 좀더 세밀히 classify 하여 HStab 의 해당 ID 의 속성을 update 하라.
 (예, ID 인 경우 => integer scalar 변수, float scalar 변수, integer array 변수, float array 변수, void 함수명, integer 함수명, float 함수명, ...함수의 type 의 파라미터 등)

출력 예) HASH TABLE 출력 양식은 자유이다.

```
[[ HASH TABLE ]]
```

```
Hash Code 10: (abc: integer scalar variable, line3)
```

```
(bcd: integer array variable, line6)
```

```
Hash Code 20: (f: function name, return type = void, line2)
```

[과제3의 채점을 위한 주요 평가기준]

- 1) 일반적 사항 (→ 학생이 제출한 프린트물로 채점) (10점)

- 프로그램 source file이 분리되어 있는가? (th, globb, symtable.c, main.c, reporter.c, scanner.l, parser.y ... 파일이름은 약간 달라도 됨.)
- 입출력 file 빠짐없이 제출하였는가?
 - MiniC 문법 오류 없는 경우 4개 MiniC 프로그램과 그에 대한 출력
 - MiniC 문법 오류 있는 경우 4개 MiniC 프로그램과 그에 대한 출력
- 코딩 스타일이 적절한가? (각 파일의 앞부분에 파일이 담고 있는 프로그램에 대한 주석, 각 서브 function에 대한 주석 등, 적절히 blank line, Indentation을 넣었는가?)
- MiniC 문법 오류 없는 경우 4개 MiniC 프로그램과 그에 대한 출력이 맞는가?
- MiniC 문법 오류 있는 경우 4개 MiniC 프로그램과 그에 대한 출력이 맞는가?

- 2) 내용 구문분석 결과가 정확한 가? (→학생이 제출한 source code를 직접 실행한 결과로써 채점)(10점)

- MiniC 문법 오류가 없는 프로그램 (예, prime.mc)
 - Error 0로 표시
 - HS symbol Table 출력 내용
- MiniC 문법 오류가 있는 프로그램 (예, prime.mc에 적절한 error를 넣을 것임)
 - 적어도 하나의 Error를 detect하고 메시지를 source line과 함께 출력하는가?
 - 정상적으로 프로그램 종료가 되는가? (즉, core dump등 비정상적으로 종료되면 안됨)
 - HS Symbol Table 내용은 check하지 않음

3) 주의사항

- 본 과제는 Modified MiniC 임. (즉, 변경된 내용에 맞추어 문법을 디자인해야 함. 예, float 자료 형에 해당하는 keyword 추가 필요 등)
- 본 코딩 모두 대문자로
- 참고: 과제 1 의 경우 token type 을 enumerated type 으로 define 하였으나, 과제 2 의 경우는 parser.y 에서 자동적으로 생성되는 y.tab.h 를 이용하여야 한다.

<UNIX: YACC>

```
$ yacc -d parser.y
```

```
y.tab.c y.tab.h → cp y.tab.h tn.h
```

<BISON>

```
C.W> bison -d parser.y
```

```
parser.tab.c, parser.tab.h → copy parser.tab.h tn.h
```

과제 개요

- **(Modified) Mini_C의 parser를 과제2에서 작성한 scanner프로그램을 이용하여 scanning한 token을 bison (YACC)을 사용하여 구문 분석하라.**
→ 과제 2 scanner. 2 사용 (과제 1도 당연히 사용)
 - 1) scanner와 parser에서 발생한 모든 문법 오류 error에 대하여 가능한 한 의미 있는 error 메시지로써 출력하라.

예) line 3, 10에서 error 발생 가정

*** MiniC parsing begins

line
number

3

line 3에서 발생한 error type에 따른 message 출력

10

line 3에서 발생한 error type에 따른 message 출력

Parsing ends. ***

2 error(s) detected

과제2
scanner. 2
←
+ 구문분석
후이행인
메시지에세리

→ 강의노트..
<calc.y> 의미있는
 에러메세지
 어떻게?

```
%{
/* YACC source for calculator program */
# include <stdio.h>
%}

%token NUMBER DIV MOD SQR
%left '+' '-'
%left '*' DIV MOD
%left SQR
%%

comm : comm '\n'
      | lambda
      | comm expr '\n' {printf("%d\n", $2);}
      | comm error '\n' {yyerrok; printf(" Try again \n");}
      ;
      ↳ 이거 활용.
      해당 파트의 에러가 있으면 이것에 매칭됨
      ↳ 에러메세지
      수정 (정리하게)

expr : '(' expr ')'      {$$ = $2;}
      | expr '+' expr    {$$ = $1 + $3;}
      | expr '-' expr    {$$ = $1 - $3;}
      | expr '*' expr    {$$ = $1 * $3;}
```

If an error is detected in the parse, the parser skips to a newline character, the error status is reset(**yyerrok**) and an appropriate message is output.

- 만일 error가 하나도 없이 구문분석이 끝났을 경우는, 다음이 출력된다

예)

```
*** MiniC parsing begins
```

```
Parsing ends. ***
```

```
0 error(s) detected
```


- 2) 변수(즉, Identifier)가 선언, 혹은 함수가 정의되었을 때, 선언(정의)된 위치의 라인넘버를 Hstab에 저장하라. 구문분석이 끝나면, Hstab에 저장된 모든 identifier에 대한 속성, 즉
- ① - identifier character string
 - ② - its type ↑ 배열이 아닐 것
scalar 혹은 array 변수인지, 함수 파라미터인지, 함수명인지, 함수명일 경우 return type
 - ③ - line number

①, ③은 과제 2에서 함

을 과제1에서처럼 출력하라. 출력 양식은 자유.

예)

[[HASH TABLE]]

Hash Code 10: (abc: integer scalar variable, line3)

(bca: integer array variable, line6)

Hash Code 20: (f: function name, return type = void, line2)

추가적으로

- 과제2의 경우 token type을 enumerated type으로 define하였으나, 과제2의 경우는 parser.y에서 자동적으로 생성되는 y.tab.h를 이용하여야 한다.

<UNIX: YACC>

\$ yacc -d parser.y

y.tab.c, y.tab.h

cp y.tab.h tn.h

<BISON>

C:\> bison -d parser.y

parser_tab.c, parser_tab.h

→ copy parser_tab.h tn.h

- 한번 tn.h가 만들어 졌으면, 그 다음에는 더 이상 -d 를 할 필요 없다.

- 주의:**

- parser.y 토큰명을 대문자로 할 것 (→ 교재와 다른 점) scanner.l 도 같은 대문자 토큰명으로 할 것.
- 교재의 parser.y 에 있는 terminal 들 (예 ';' 등) 도 모두 토큰명을 정의할 것

↳ 따옴표로 하지 않기

```

actual_param_list : assignment_exp          { semantic(93);}
                  | actual_param_list ',' assignment_exp { semantic(94);};
primary_exp      : tidet                     { semantic(95);}
                  | number                   { semantic(96);}
                  | '(' expression ')'       { semantic(97);};
%%
#include "lexyy.c"      /* #include "lex.yy.c" */
void yyerror(char *s)
{
    printf("%s\n", s);
}
void semantic(int n)
{
    printf("reduced rule number = %d\n", n);
}
void main()
{
    printf("start of parser\n");
    yyparse();
    printf("end of parser\n");
}

```

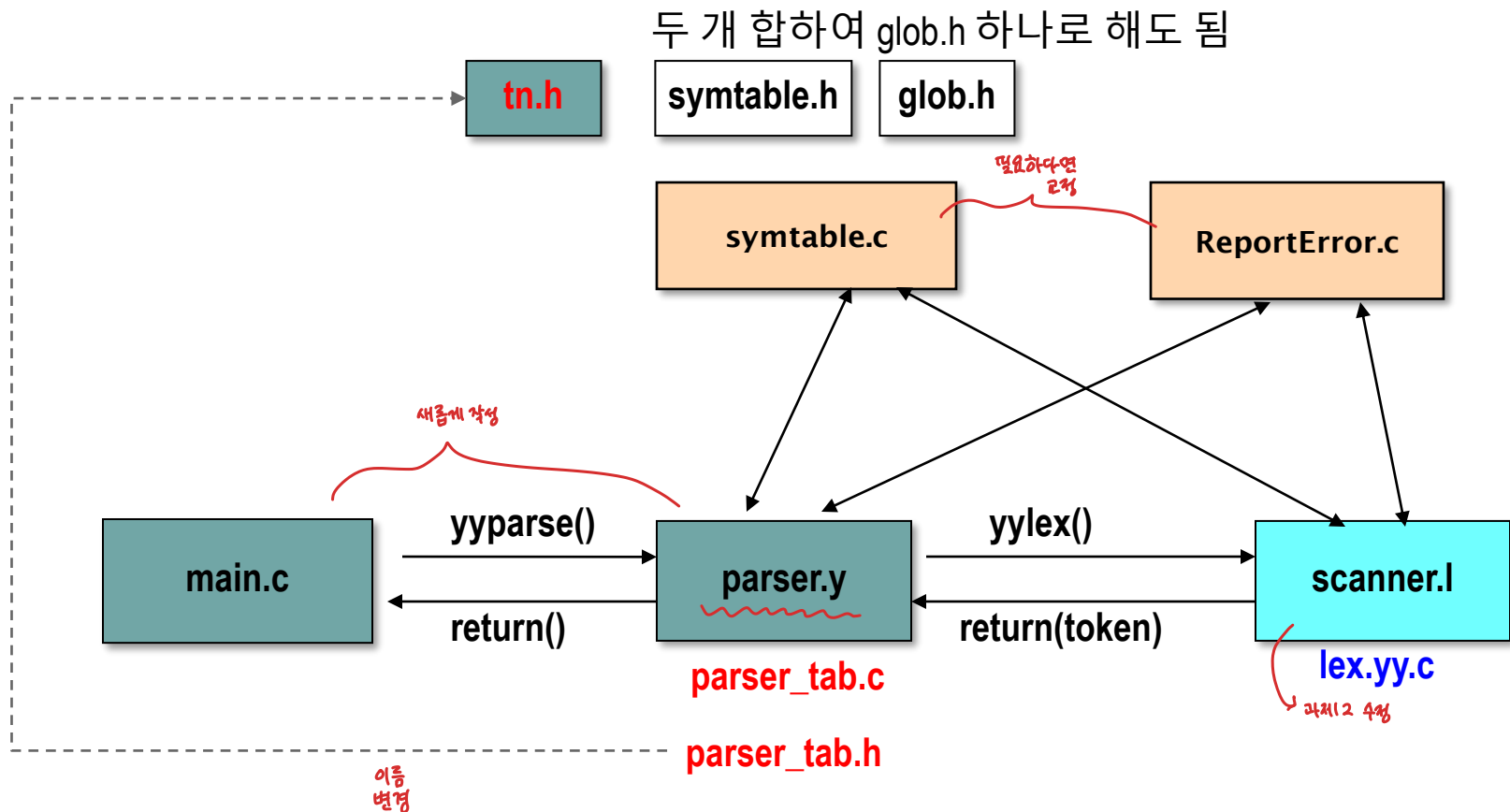
이전게
문법영부여하기!

여기서 사용한 Mini C 문법은 [부록 A]에 수록된 형태이다. 그리고 함수 `semantic()`에서 단순히 생성 규칙 번호만 프린트하여 우파스를 출력하였는데 필요한 코드 생성 작업을 수행하여 실질적인 전단부로 사용할 수 있다. 보통 여기에서 생성 규칙에 맞는 추상 구문 트리(AST)를 만드는 작업을 행한다.

2.2 스탠포드 파서 제작 시스템

스탠포드(Stanford) PGS는 스탠포드 대학의 John Henessy에 의해 개발된 파서 생성기로 문법 규칙에 추상-구문 트리(AST: Abstract Syntax Tree)를 자동으로 만들 수 있는 정보를 결합시킬 수 있다. PGS는 context-free 문법 형태로 된 입력을

Framework



/* tn.h */

Token 에 관한 정의: #define 으로 정의

C: \> bison -d parser.y

parser_tab.c, parser_tab.h 가 생성됨

C: \> copy parser_tab.h tn.h

/* glob.h */

Symbol table 및 그외 global 변수들

```
%{  
/* scanner.l  
(description.. )  
*/  
# include <stdio.h>  
# include "tn.h" /* token name definition */  
# include "glob.h" /* global variable */  
(그 외 필요한 global variable)  
%}  
  
%%  
  
%%
```

Call 하는 Subroutines

```
%{
    /* parser.y */
    #include <stdio.h>
    #include "tn.h"
    #include "glob.h"
```

```
%}
%token INTNUM ...
```

```
...
%left ...
```

```
...
%%
```

mini_c

: prog_head block DOT

필요한 경우에 symbol table management를 해야 한다.

```
...
%%
```

```
/* reporterror.c
```

```
*/
```

```
#include <stdio.h>
```

```
#include "tn.h"
```

```
#include "global.h"
```

그 외 필요한 global 변수들

```
yyerror(char *s) {
```

```
...
```

```
    printf("%s\n",s);
```

```
...
```

```
}
```

그외 Error Report에 필요한 subroutines..


```
/* main.c
```

```
*/
```

```
#include <stdio.h>
```

```
#include "tn.h"
```

```
#include "global.h"
```

그 외 필요한 global 변수들

```
main( )
```

```
{
```

```
...
```

```
    yyparse();
```

```
...
```

```
    printHT(); ...
```

```
}
```

↳ 1점 symboltable의
하수를 extern으로
사제다 써야겠다.

Submit

1) 전체 소스코드를 Zip 하여 제출 → 조교가 직접 실행하여 평가

- Source codes: main.c, parser.y, scanner.l, symtable.c, reporterror.c, tn.h, glob.h 등.

(주의사항: lex.yy.c 와 parser_tab.c 도 제출)

작업 실행할 예정

2) 입출력 8쌍 이상: (test1.mc, 출력스캔), (test2.mc, 출력스캔), Zip 하여 제출

- Mini C 문법 오류 없는 것(4개 이상), Mini C 문법 오류 있는 것(4개 이상)

<*** Output의 경우 과제물에 명확히 정의되지 않은 부분이 있어 본인이 정의한 것이 있을 경우 그 내용을 기술하고, 해당 output 부분에 표시. ***>

• 주의사항: 본 과제는 Modified MiniC 임

- ★ • 즉, 변경된 내용에 맞추어 문법을 디자인해야 함. 예, float 자료 형에 해당하는 keyword 추가 필요 등 (keyword: float)
- ★ • 교재의 parser.y 에 있는 Terminal 들 (예 ';' 등) 도 모두 토큰명을 정의할 것

5/10 실습3 (팀 실습)

- 제출일: 5/10 24:00
- 제출물: 실습지: <L10_YACC_실습_회의록.xlsx>
- 실습지에 실습 수행하였다는 증빙 있어야함
→ 출석 인정 및 실습 점수 만점

실습3: 과제3

- 과제3 를 위하여 실습2 결과를 토대로 수행할 내용

- ^{실습2} MiniC.I를 과제3의 scanner.I 로 대체한다.
 - 토큰명을 **모두 대문자로** 하고, scanner.I과 parser.y의 토큰명이 일치해야 한다.
 - 문법의 **모든 터미널은 토큰명으로 정의**한다. (혼용하지 말 것. 즉 교재의 parser.y 에 있는 Terminal 들 (예 ';' 등) 도 모두 토큰명을 정의할 것)
 - **Keyword: float 등 추가 필요**
- parser.y에서 semantic 함수와 이를 call하는 코드 (**예, semantic(1);**) **삭제**. 과제3의 내용에 따라 parser.y 보완 추가적인 속성? symbol-table.c
- 해쉬심볼테이블의 **Identifier 속성** 보완 + scanner.h 도 보완
parser.y
- 과제 결과 출력에 관한 내용 (slide 22,24,25 참고) 구현.
- reporterror.c는 과제2에의 내용뿐만 아니라 과제3에서 새롭게 필요한 error 메시지 출력에 관한 것을 추가한다.
- main.c, glob.h 보완
- **입력 데이터인 MiniC 프로그램으로써 실행하여 결과를 확인한다.**
 - 교재 MiniC 프로그램 파일 (단, **Modified Mini C에 맞추어 작성** 해야 함). 예) prime.mc
 - error가 있는 입력 MiniC 프로그램 파일

5/13 실습4: 과제 수행 (팀 실습)

- 제출일: 5/13 24:00
- 제출물: 실습지: <L10_YACC_실습_회의록.xlsx>
- 실습지에 실습 수행하였다는 증빙 있어야함
→ 출석 인정 및 실습 점수 만점

실습

- 팀 별 실습

- 팀별로 과제3를 수행합니다. <L10_YACC_실습_회의록.xlsx>의 '실습지'에 **실습 시 팀에서 진행한 내용 일부**를 증빙 자료로 넣어서 **5/13(목) 24시**까지 사이버캠퍼스에 팀 대표가 제출 → **출석체크 및 실습 점수** (참고: 실습 수행하였다는 증빙 있어야 함. → 출석 인정 및 실습 점수 만점)

- 사이버 캠퍼스 팀게시판 활용

↳ 자료 공유..?

학기말 시험 및 강의 일정

• 학기말시험 날짜 변경

- 일시: 6월 7일(월) → 6월 5일(토) 10시 ~ 22시 (최대 12시간, 추후 확정 공지)
- 방법: 온라인시험 ?도요일..
- 내용: lex와 yacc을 활용한 스캐너 및 구문분석 프로그래밍

• 강의 일정

이/중	5/17 월	구문분석(LL)	
	5/20 목	구문분석(LL&LR)	
	5/24 월	중간언어	
	5/27 목	코드최적화	5/26(수) 24:00 과제3 제출
2월..	5/31 월	창립기념일 → PI 5회중 최소 2회 참석으로 대체	
	6/3 목	시험공부	(과제3 채점: 팀별 결과물 조교가 실행확인, 필요시 해당 팀 연락)
	6/5 토	학기말시험 10시 ~ 22시 (최대 12시간, 단 문제에 따라 축소될 수 있음, 추후확정공지)	
	6/11 목	총정리 및 학기말 시험 결과	

• 과목평가기준

- 학기말시험 40%, 과제50%(과제1 10%, 과제2 20%, 과제3 20%), 출석5%, 실습 5%