<pascal_like compiler>

1. Lexical #1

```
$ cd
$ mkdir pascal-like
$ cd pascal-like

$ flex pascal-like.l
$ gcc -o pascal-like lex.yy.c -lfl
$ gcc -o -DPRINT pascal-like lex.yy.c -lfl
$ ./pascal-like
:=
begin
end
if
then
else
while
do
^Z
```

2. Lexical #2

- BNF -> bison

```
$ flex pascal-like.l
$ bison pascal-like.y
```

3. Conflict/Shift error debug

```
$ bison pascal-like.y
$ bison -v pascal-like.y
$ ls -l
$ vi pascal-like.output
: set number
```

```
$ vi pascal-like.y
expr                    : expr binaryOp expr
->
expr                    : value binaryOp expr
```

```
$ bison -v pascal-like.y
$ vi pascal-like.output
:/State 32
```

```
$ vi pascal-like.y
```

```
expr                    : expr binaryOp expr
->
```

```
expr                    : value binaryOp expr
```

```
$ bison -v pascal-like.y
$ vi pascal-like.output
:/State 32
```

```
if_statement       : if condition then statement
                    | if condition then statement else statement
                    ;
->
if_statement       : if condition then statement
                    ;
```

```
$ bison -v pascal-like.y
```

4. flex bison together

```
$ vi pascal-like.y
G > a
#include "lex.yy.c"
```

```
$ flex pascal-like.l
$ bison pascal-like.y
$ gcc -o parser pascal-like.tab.c -lfl
$ vi pascal-like.l
```

```
TOKEN(begin) -> TOKEN(begin_T)

TOKEN(end) -> TOKEN(end_T)

TOKEN(if) -> TOKEN(if_T)

TOKEN(then) -> TOKEN(then_T)

TOKEN(else) -> TOKEN(else_T)

TOKEN(while) -> TOKEN(while_T)

TOKEN(do) -> TOKEN(do_T)


$ vi pascal-like.y

begin -> begin_T

end -< end_T

.

.

.


$ flex pascal-like.l

$ bison pascal-like.y

$ gcc -o parser pascal-like.tab.c -lfl

$ cp arith.c pascal-like.c

$ gcc -o parser pascal-like.tab.c pascal-like.c -lfl


$ vi pascal-like.y


int main()

{

#if YYDEBUG == 1
extern int yydebug;

yydebug = 1;

#endif

        return(yyparse());

}
```

```
$ gcc -o parser pascal-like.tab.c pascal-like.c -lfl -DYYDEBUG

$ ./parser < test_program.pas

$
```