

Professor Chung Yung

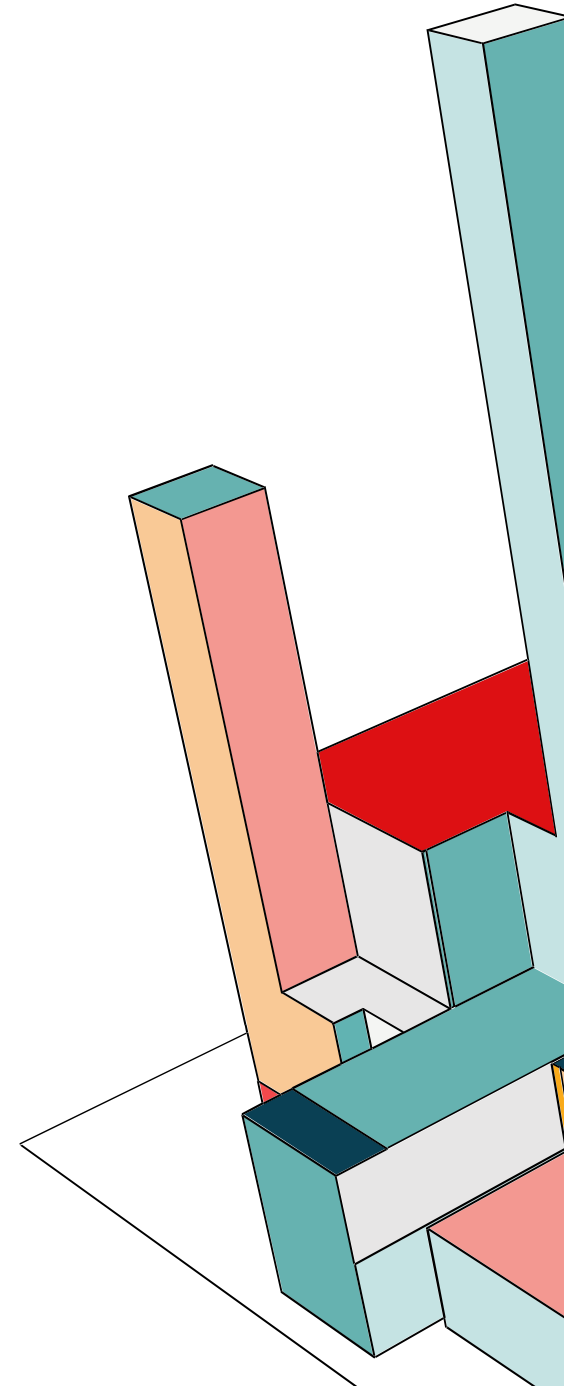
PROGRAMMING LANGUAGES PRINCIPLES AND PARADIGMS

CSIE Dept., NDHU, Fall 2024

Programming Notes #1

PROGRAMMING NOTES #1

- Suggested Steps
- Programming Environment
- Basics of Flex and Bison
- A Practical Example

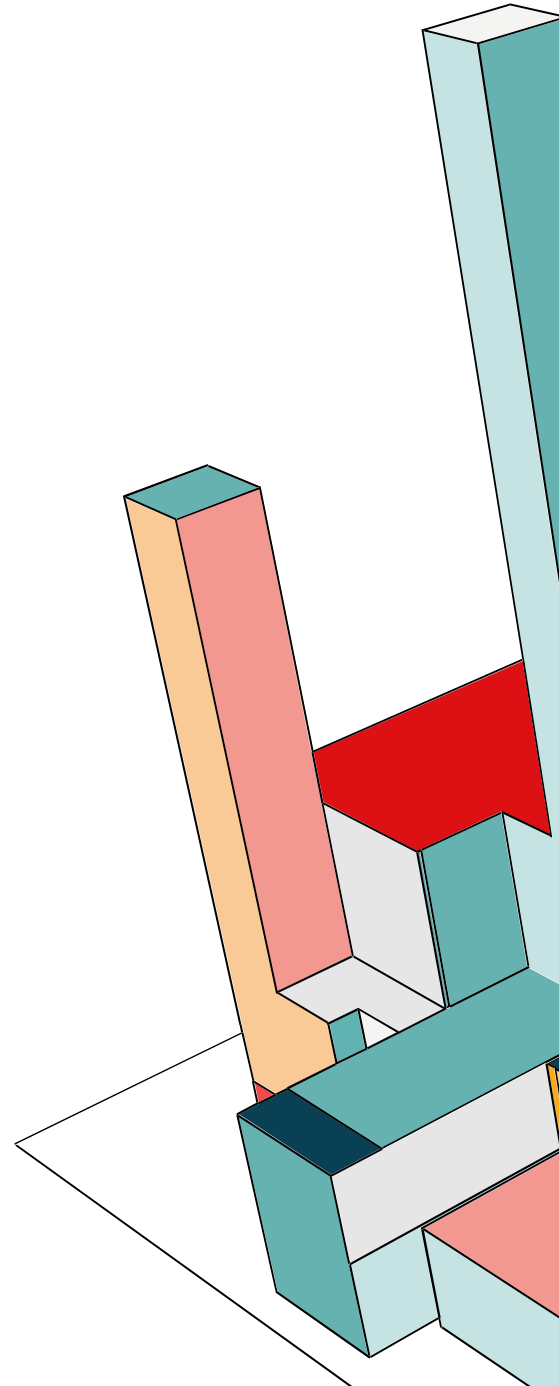


SUGGESTED STEPS



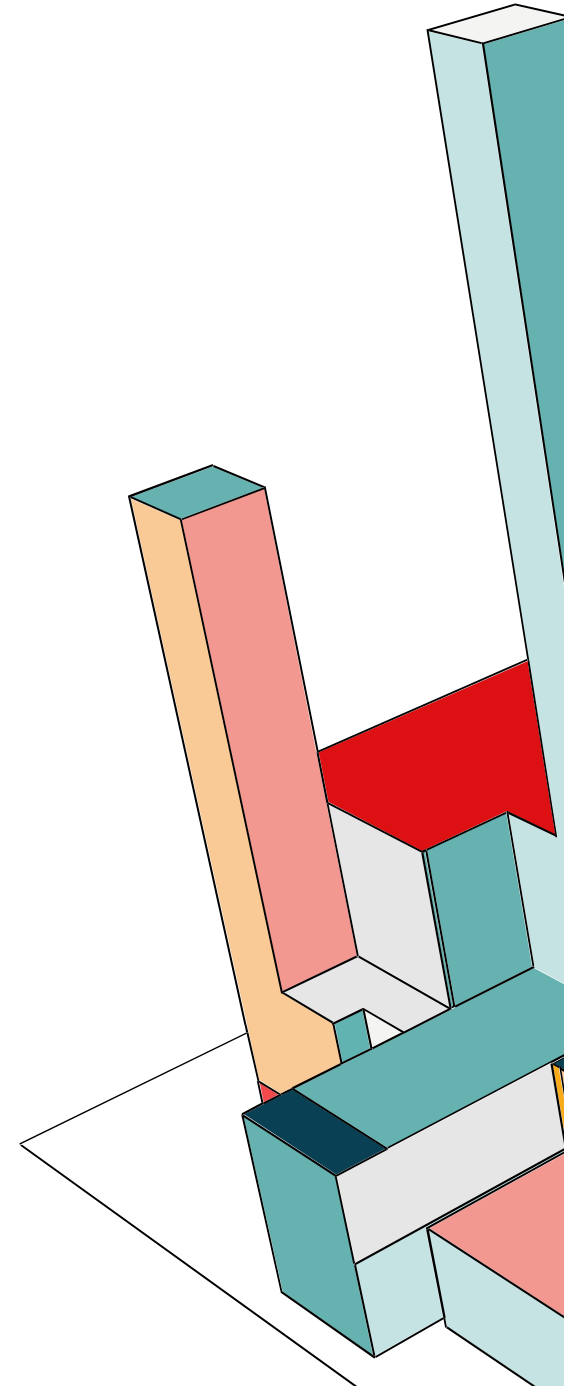
WHAT TO DO (T)

- Write a lex specification `t_lex.l`.
- Write a yacc specification `t_parse.y`.
- Write a main function `t2c.c`.
- Write a header file `t2c.h`.

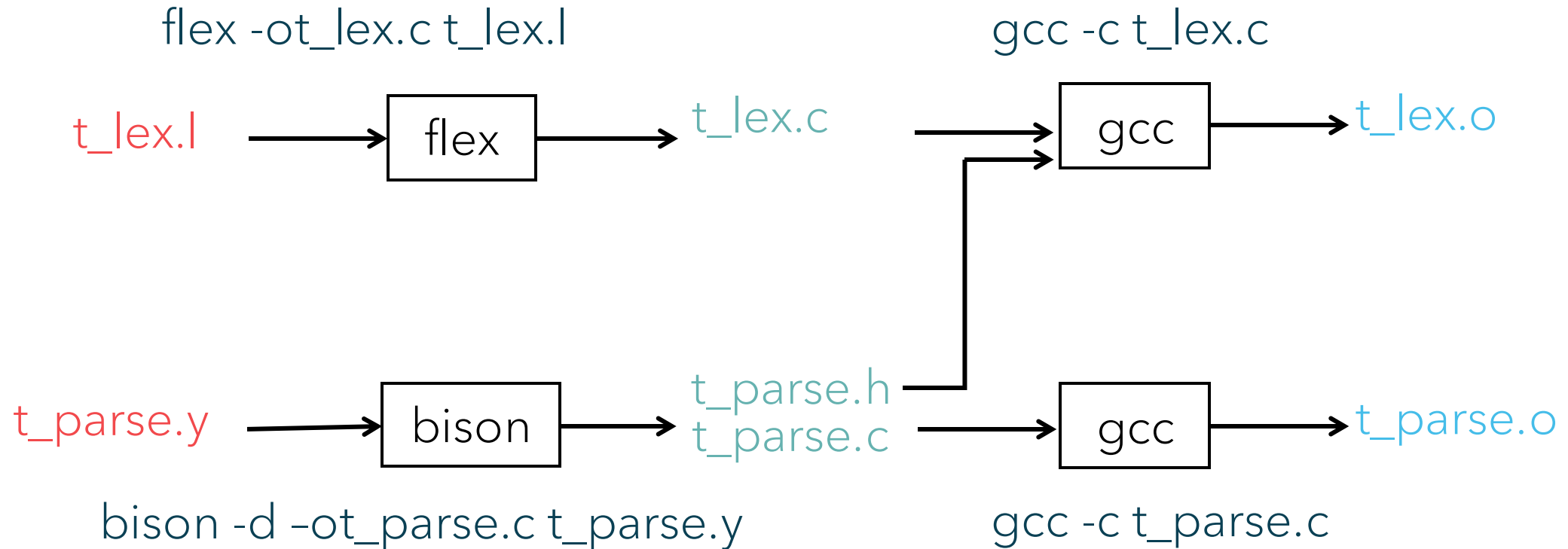


SUGGESTED STEPS

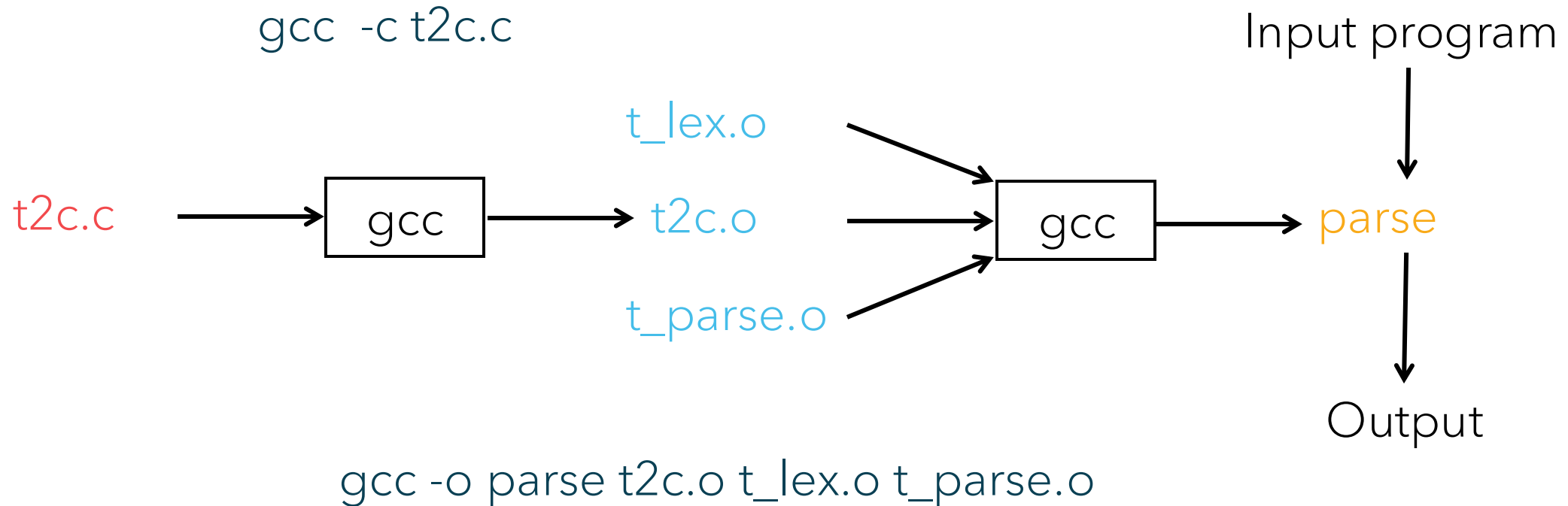
- 1) Use bison to compile `t_parse.y` into `t_parse.c`.
The `-d` switch produces a header file `t_parse.h`.
- 2) Use flex to compile `t_lex.l` into `t_lex.c`.
- 3) Use gcc to compile `t_lex.c` into `t_lex.o`,
`t_parse.c` into `t_parse.o`, and `t2c.c` into `t2c.o`.
- 4) Use gcc to link `t2c.o`, `t_lex.o`, and `t_parse.o` into
`parse`.



GENERATING C CODE



BUILDING EXECUTABLE





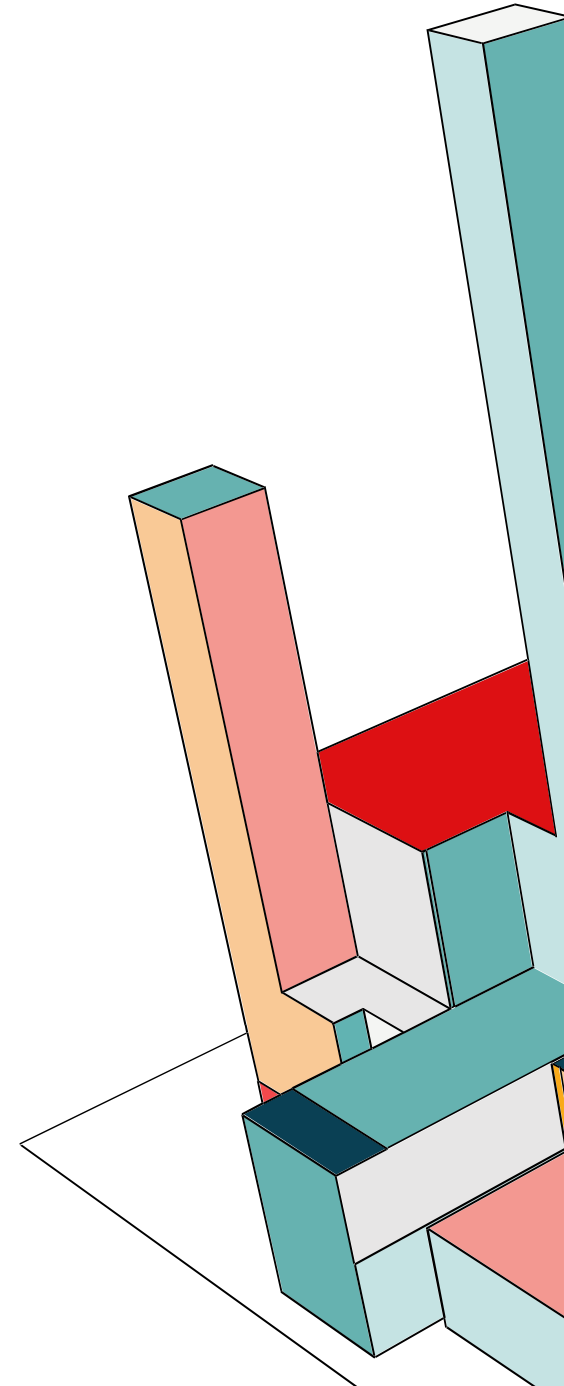
PROGRAMMING ENVIRONMENT

SOFTWARE

- 1) Flex 2.5.4
- 2) Bison 2.4.1
- 3) MinGW 4.6.2

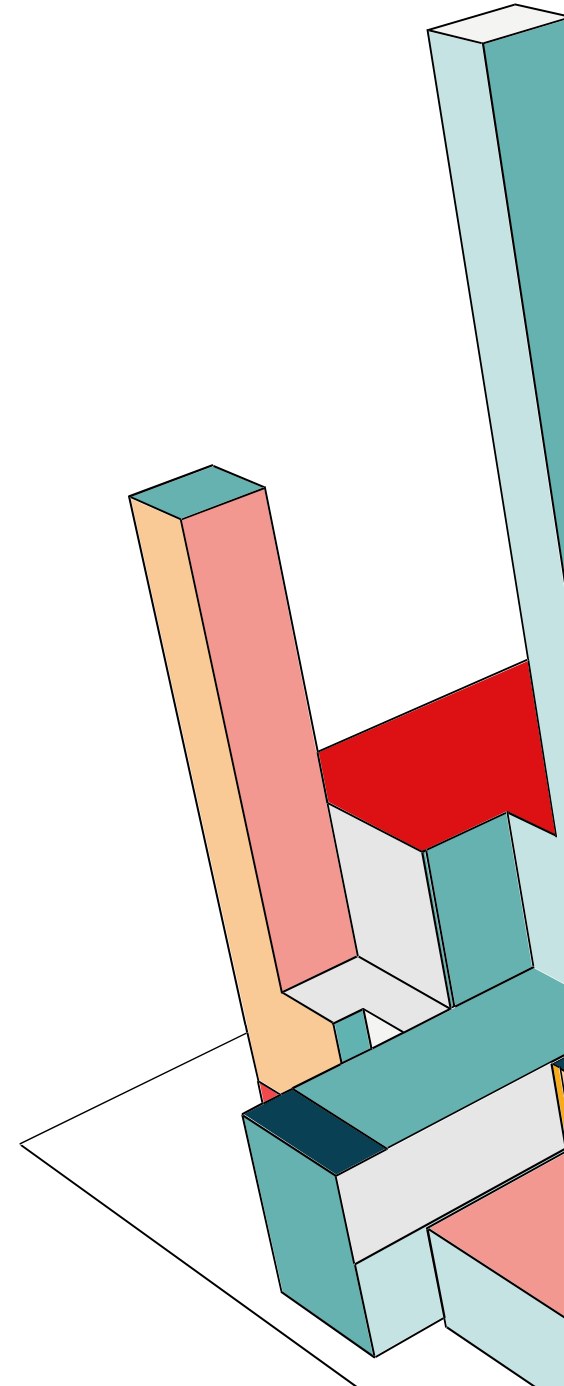
Notes

- DO NOT install both MinGW and DevC.
- Check and set up environment variables.
- Rebooting after installation is suggested.

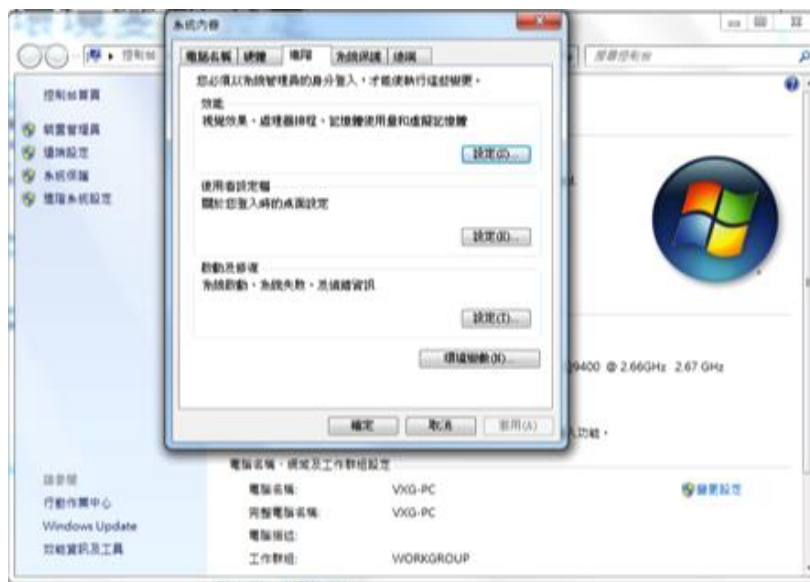


ENVIRONMENT VARIABLES

- Check whether `C:\MinGW\bin` is in `PATH`. If not, add it into `PATH`, using `;` as separators.
- In case of using `DevC`, check for `C:\Dev-Cpp\bin` instead.



ON WINDOWS



TEST INSTALLATION

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [版本 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\fdas>gcc --version
gcc (GCC) 4.6.2
Copyright (C) 2011 Free Software Foundation, Inc.
本程式是自由軟體；請參看來源程式碼的版權宣告。本軟體沒有任何擔保；
包括沒有適銷性和某一專用目的下的適用性擔保。

C:\Documents and Settings\fdas>flex -U
flex version 2.5.4

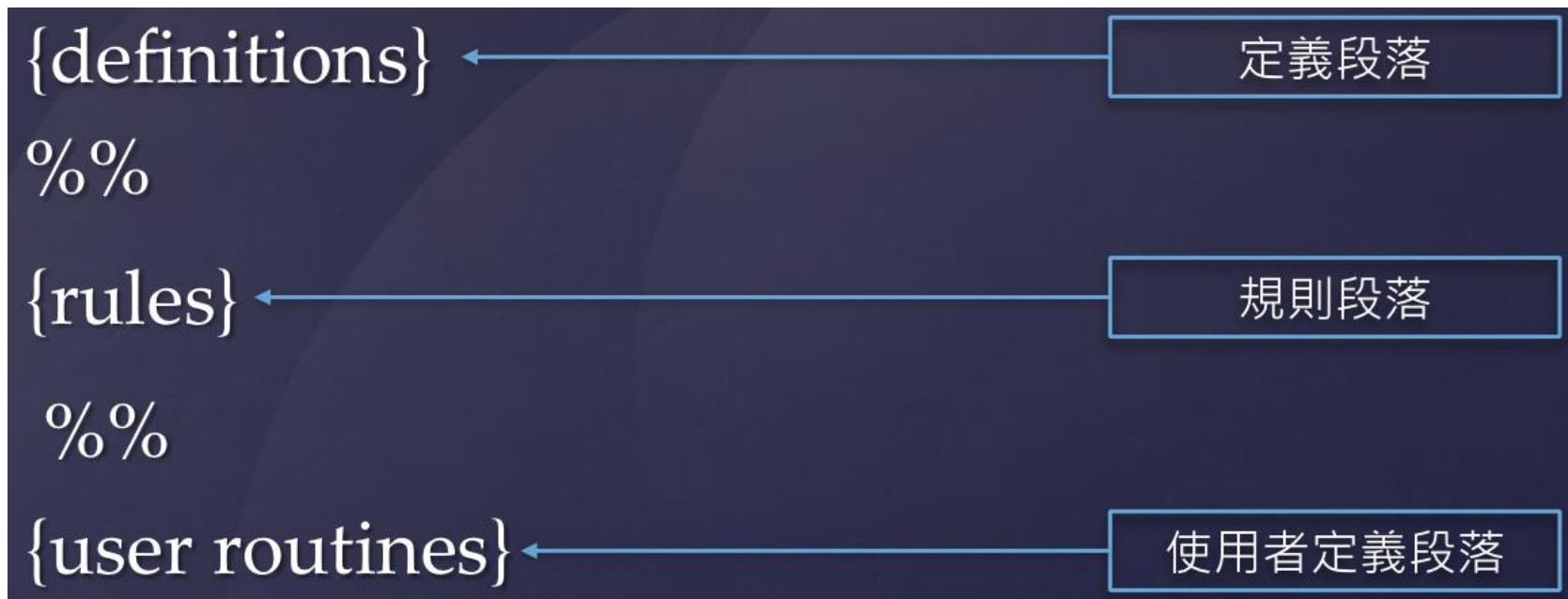
C:\Documents and Settings\fdas>_
```

FLEX AND BISON

Basics



FLEX SPECIFICATION

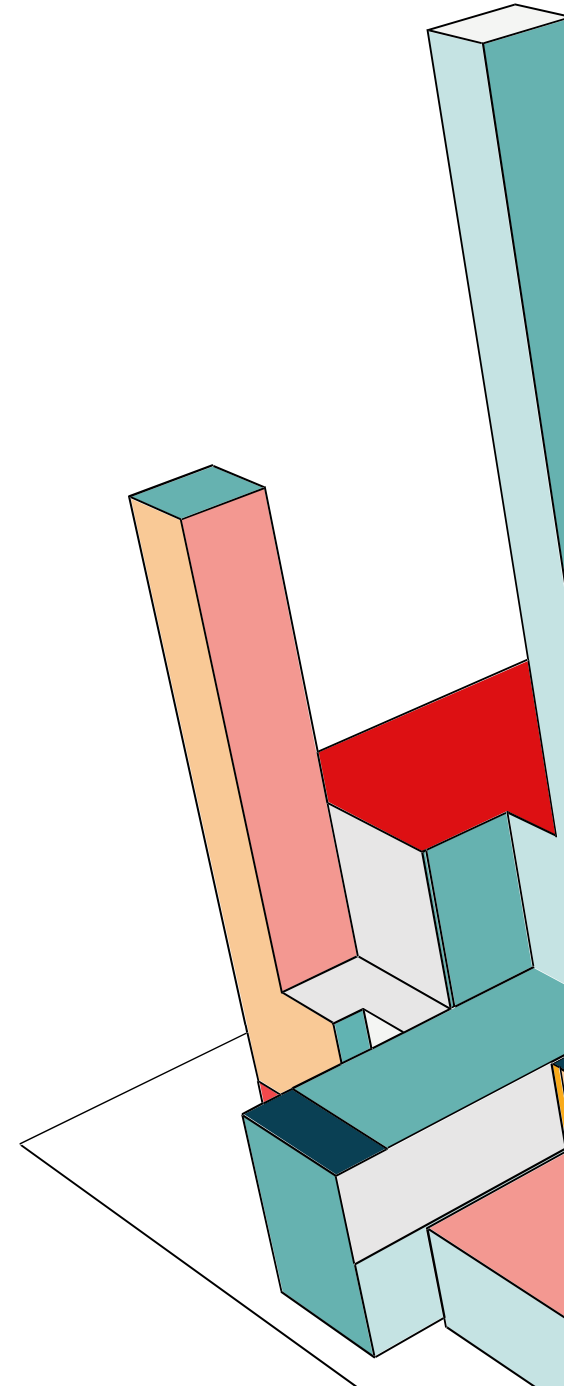


FLEX SPECIAL CHARACTERS

Char	Meaning
.	Any character except for '\n'
-	The range of characters, e.g., a-z
*	Repeat for 0 or more times
+	Repeat for 1 or more times
?	Appear either 0 or 1 time
^	Negation
A B	A or B
"sth."	Exactly the characters appeared in parentheses (sth.)

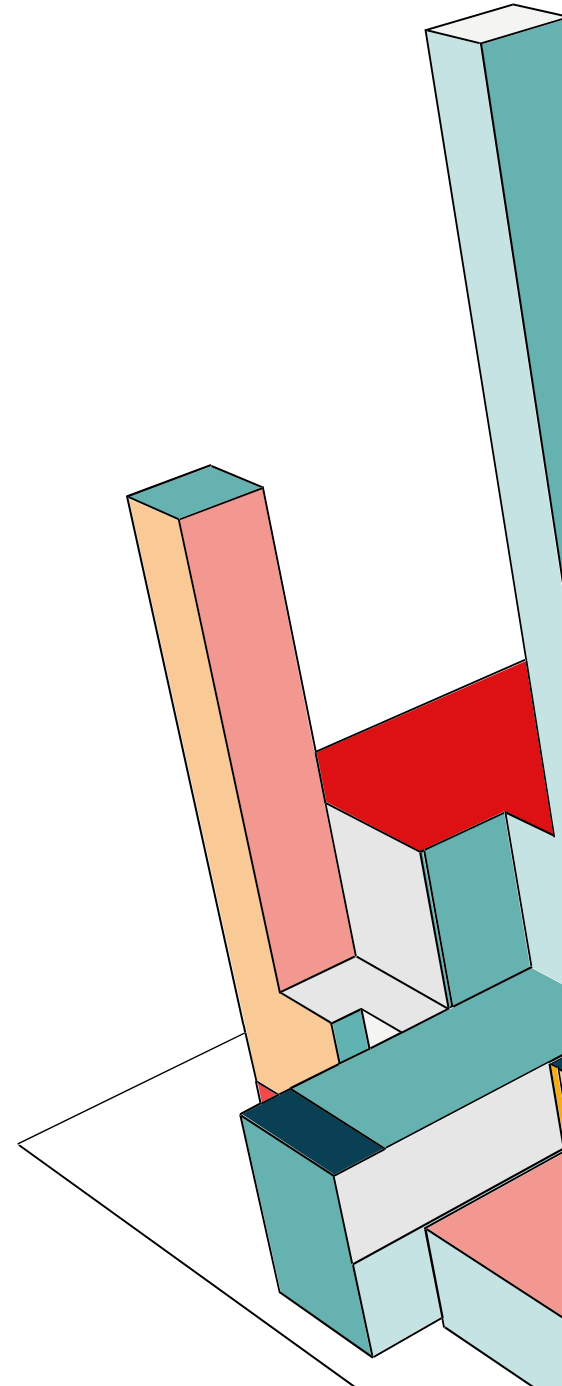
FLEX COMPILATION

- `flex -ot_lex.c t_lex.l`
- Translate `t_lex.l` into `t_lex.c`.
(Specify the filename of your own.)
- (With `old` versions)
There is **NO space** between `-o` and `t_lex.c`.

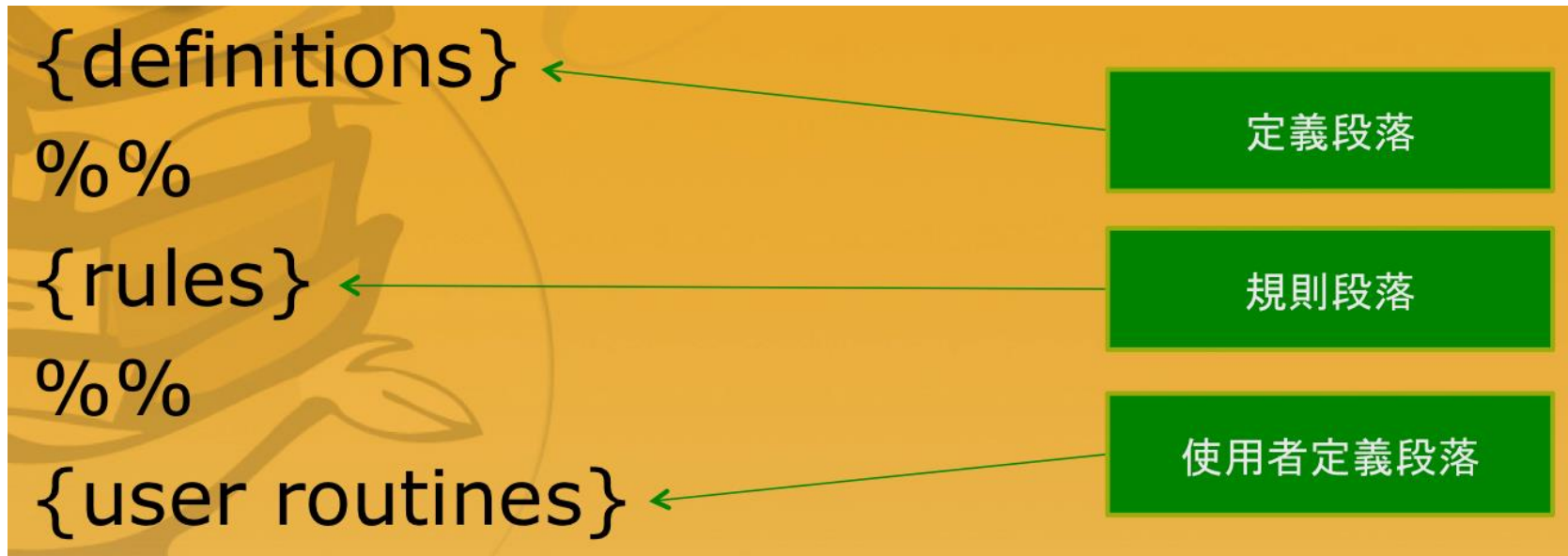


FOR THIS ASSIGNMENT

- Write your own `t_lex.l` according to the description of T lexicons, and compile into `t_lex.c`.
- `flex -ot_lex.c t_lex.l`



A BISON SPECIFICATION



DEFINITION (P EXAMPLE)

```
1  %{
2      #include <stdio.h>
3      #include <stdlib.h>
4      #include <string.h>
5      #include "pascal.h"
6  %}
7  %token PROG PROC BG END INT BOOL TRUE
8  %token LP RP LSP RSP FALSE CC
9  %token DOT SEMI VAR ARRAY OF DOTDOT
10 %token IF THEN READ WRITE WHILE DO
11 %token ELSE ASSIGN COMMA COLON ID NUM
12 %left OR AND
13 %left NOT
14 %left EQ NE LT GT LE GE
15 %left ADD MINUS
16 %left DIV TIMES
17 %expect 1
18
19 %%
```

定義段落

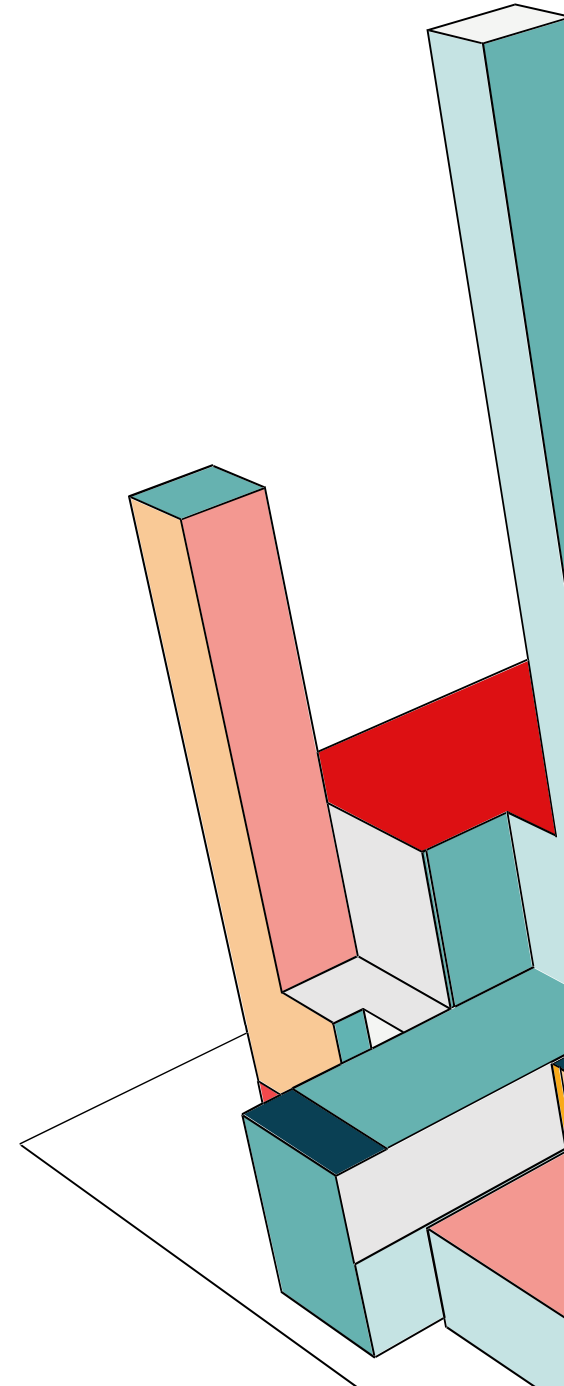
RULES (P EXAMPLE)

```
19  %%
20  prog  :  PROG ID SEMI block DOT {printf("prog => PROG ID SEMI block DOT \n*****
21         |  {printf("*****Parsing failed! \n");}
22         ;
23
24  block  :  vardecs prodecs stmts {printf("block=>vardecs prodecs stmts \n");}
25         ;
26
27  vardecs :  VAR vardec SEMI morevd {printf("vardecs => VAR vardec SEMI morevd \n");}
28         |  {printf("vardecs => Null \n");}
29         ;
30
31  morevd  :  vardec SEMI morevd {printf("morevd => vardec SEMI morevd \n");}
32         |  {printf("morevd => Null \n");}
33         ;
34
35  vardec  :  ID moreid COLON type {printf("vardec => ID moreid COLON type\n");}
36         ;
37
38  moreid  :  COMMA ID moreid {printf("moreid => COMMA ID moreid \n");}
39         |  {printf("moreid => Null \n");}
40         ;
41
```

規則段落

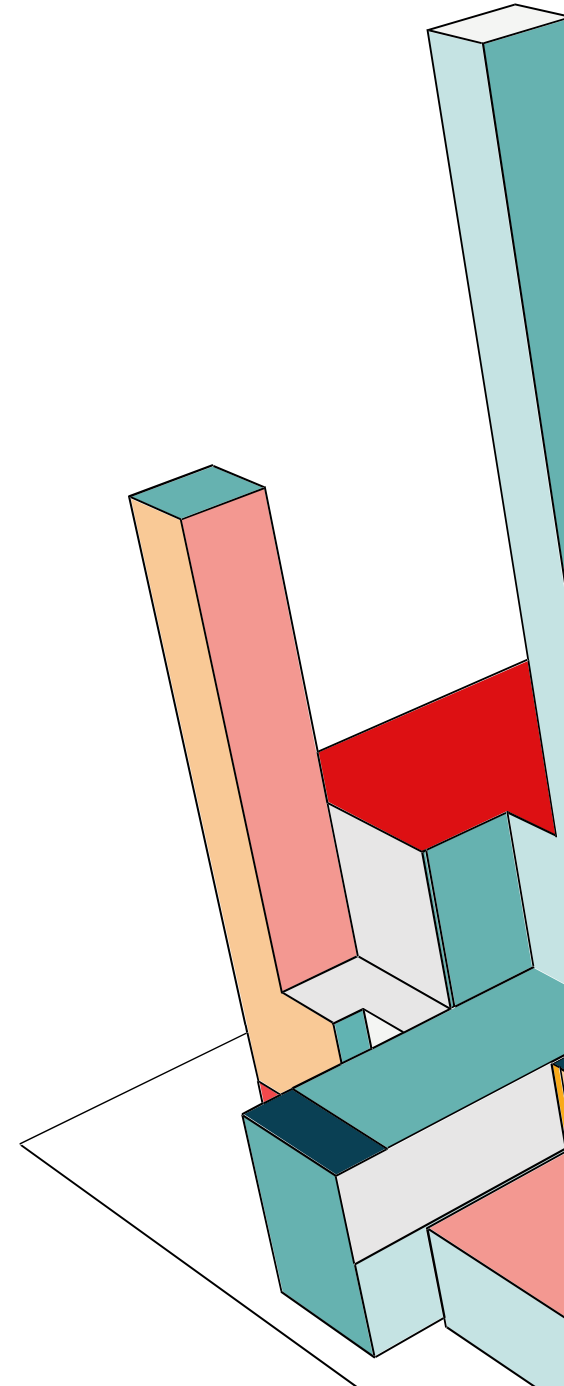
FOR THIS ASSIGNMENT

- Write your own `t_parse.y` according to the description of T grammars, and compile into `t_parse.c`.
- `bison -d -o t_parse.c t_parse.y`
- To run your parser:
`./parse test.t`



COMMANDS USED

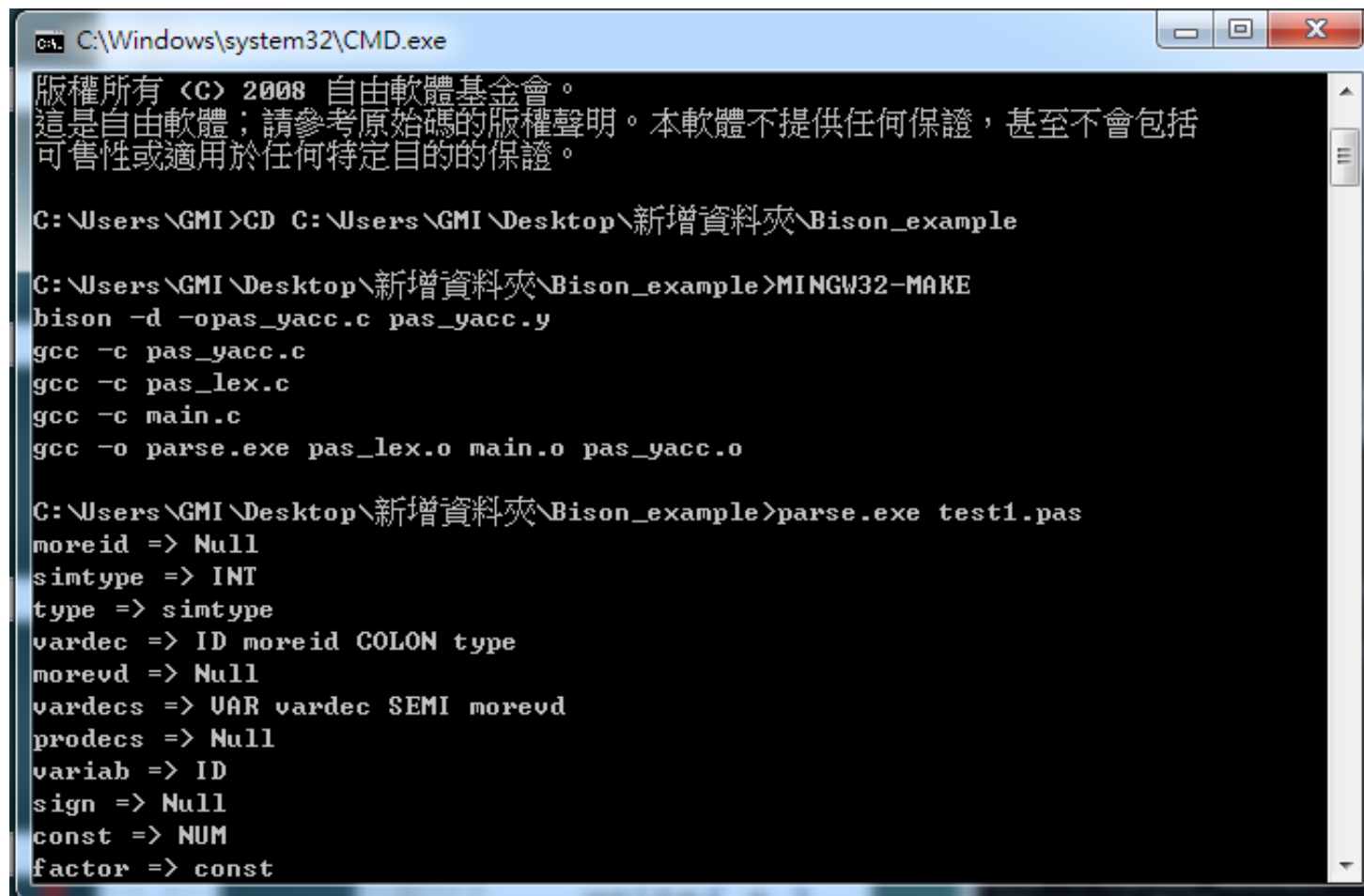
- **Bison**
`bison -d -o t_parse.c t_parse.y`
- **Flex**
`flex -ot_lex.c t_lex.l`
- **GCC**
`gcc -c t_lex.c`
`gcc -c t_parse.c`
`gcc -c t2c.c`
`gcc -o parse t_lex.o t2c.o t_parse.o`





A PRACTICAL EXAMPLE

BUILD EXECUTABLE (P)



```
C:\Windows\system32\CMD.exe
版權所有 (C) 2008 自由軟體基金會。
這是自由軟體；請參考原始碼的版權聲明。本軟體不提供任何保證，甚至不會包括
可售性或適用於任何特定目的的保證。

C:\Users\GMI>CD C:\Users\GMI\Desktop\新增資料夾\Bison_example

C:\Users\GMI\Desktop\新增資料夾\Bison_example>MINGW32-MAKE
bison -d -opas_yacc.c pas_yacc.y
gcc -c pas_yacc.c
gcc -c pas_lex.c
gcc -c main.c
gcc -o parse.exe pas_lex.o main.o pas_yacc.o

C:\Users\GMI\Desktop\新增資料夾\Bison_example>parse.exe test1.pas
moreid => Null
simtype => INT
type => simtype
vardec => ID moreid COLON type
morevd => Null
vardec => VAR vardec SEMI morevd
prodec => Null
variab => ID
sign => Null
const => NUM
factor => const
```

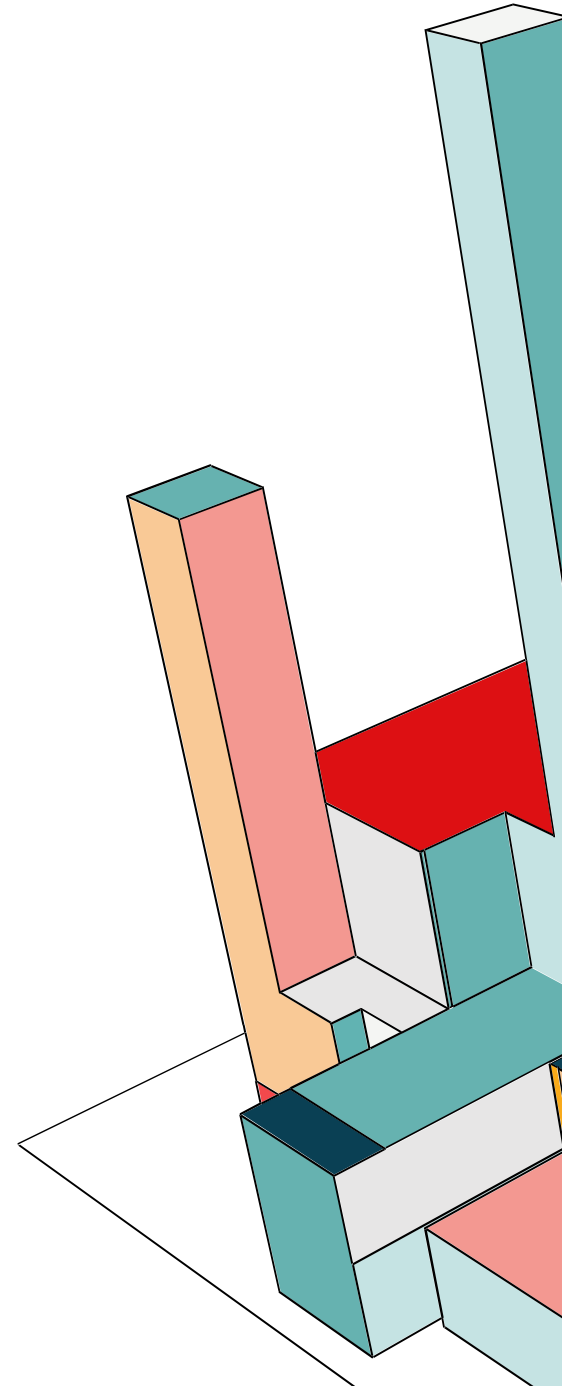

EXECUTE YOUR PROGRAM (P)

```
cmd 系統管理員: C:\Windows\system32\cmd.exe
factor => const
mulfact => Null
term => factor mulfact
addterm => Null
simexpr => sign term addterm
expr => simexpr
assstmt => variab ASSIGN expr
simstmt => assstmt
stmt => simstmt
sign => Null
variab => ID
factor => variab
mulfact => Null
term => factor mulfact
addterm => Null
simexpr => sign term addterm
expr => simexpr
outval => expr
moreout => Null
wristmt => WRITE LP outval moreout RP
simstmt => wristmt
stmt => simstmt
morestm => Null
morestm => SEMI stmt morestm
constnt => BG stmt morestm      END
stmts => constnt
block=>vardec prodec stmts
prog => PROG ID SEMI block DOT
*****Parsed OK!*****
```

```
1 program test1;
2 var a: integer;
3 begin
4     a := 3;
5     write( a )
6 end.
```

REFERENCES

- <http://www.mingw.org/>
- <http://flex.sourceforge.net/>
- <http://www.gnu.org/software/bison/>
- <http://sourceforge.net/projects/gnuwin32>





WHAT TO DO?

Use *lex* (or *flex*) and *yacc* (or *bison*) to implement a front end (including a lexical analyzer and a syntax recognizer) of the compiler for the *T* language, showing the grammar rules applied while parsing.

- See an attached file for the lexical rules in details.
- You are requested to separate the *C* code, the *Lex* specification, the *Yacc* specification into distinct files.

WHAT EXACTLY TO DO

By design, the program is splitted into 4 files:

- `t_lex.l`
- `t_parse.y`
- `t2c.c`
- `t2c.h`

▼ 04月 6日 - 04月 12日 Week8



檔案

Programming Assignment #1



檔案

Programming Notes #1



檔案

Assignment #1 Package



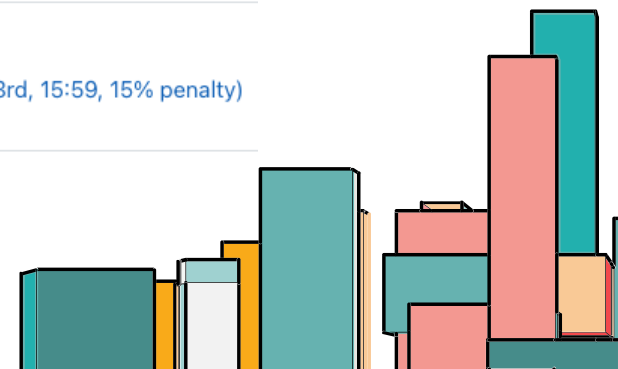
作業

Report for Assignment #1 (before May 27th, 15:59)



作業

Late Report for Assignment #1 (before June 3rd, 15:59, 15% penalty)



HW01 PACKAGE

名稱	修改日期	大小
circle.t	2017年3月23日 下午12:58	167 byte
Makefile	今天 上午9:20	556 byte
t_lex.l	今天 上午9:25	2 KB
t_parse.y	今天 上午9:25	1 KB
t2c.c	今天 上午9:20	224 byte
t2c.h	2016年11月3日 下午11:55	718 byte
test.t	2017年3月23日 下午12:58	284 byte
test4.t	2017年3月23日 下午12:58	232 byte

No changes needed!

Test Input files!

T_LEX.L

```
%{
#include "t2c.h"
#include "t_parse.h"
%}

%x C_COMMENT

ID  [A-Za-z][A-Za-z0-9]*
DIG [0-9][0-9]*
RNUM {DIG}'.'{DIG}
NQUO [^"]

%%

WRITE      {return \WRITE;}
READ       {return \READ;}
IF          {return \IF;}
ELSE       {return \ELSE;}
RETURN     {return \RETURN;}
BEGIN      {return \BEGIN;}
END         {return \END;}
MAIN       {return \MAIN;}
INT        {return \INT;}
REAL       {return \REAL;}
";"        {return \SEMI;}
","        {return \COMMA;}
"("        {return \LP;}
")"        {return \RP;}
"+"        {return \ADD;}
"_"        {return \MINUS;}
"*"        {return \TIMES;}
"/"        {return \DIVIDE;}
">"        {return \GT;}
"<"        {return \LT;}
":="       {return \ASSIGN;}
"=="       {return \EQU;}
"!="       {return \NEQ;}
">="       {return \GE;}
"<="       {return \LE;}

// Identifiers, Integer numbers, Real numbers

\"{NQUO}*\" {sscanf(yytext,"%s", qstr); return \QSTR;}
"/*"        { BEGIN(C_COMMENT); }
<C_COMMENT>"*/" { BEGIN(INITIAL); }
<C_COMMENT>\n { }
<C_COMMENT>.\n { }
[ \t\n]      { }
.%%
```

Understand the rules
and write them here!

The T Lexicons

Keywords (All keywords are reserved. Each keyword can be a terminal.):

WRITE READ IF ELSE RETURN BEGIN END MAIN INT REAL

Single-character separators (Each operator can be a terminal.):

; , ()

Single-character operators (Each operator can be a terminal.):

+ - * / > <

Multi-character operators (Each operator can be a terminal.):

:= == != >= <=

Identifiers:

An *identifier* consists of a letter followed by any number of letters or digits.

Integer numbers:

An *integer number* is a sequence of digits, where a *digit* has the following definition:

Digit -> '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

Real numbers:

A *real number* is a sequence of digits followed by a dot, and followed by digits.

Comments:

A *comment* is a string between /* and */. Comments can be longer than one line.

QStrings:

A *QString* is any sequence of characters except double quote itself, enclosed in double quotes.

T_PARSE.Y (1/3)

```
%{  
    #include <stdio.h>  
    #include <stdlib.h>  
    #include "t2c.h"  
    #include "t_parse.h"  
}%
```

```
%token \WRITE \READ \IF \ASSIGN  
%token \RETURN \BEGIN \END  
%left  \EQ \NEQ \GT \LT \GE \LE  
%left  \ADD \MINUS  
%left  \TIMES \DIVIDE  
%token \LP \RP  
%token \INT \REAL \STRING  
%token \ELSE  
%token \MAIN  
%token \SEMI \COMMA  
%token \ID \INUM \RNUM \QSTR
```

```
%expect 1
```

```
%%
```

Declare your tokens
and delete others

No need to change!

T_PARSE.Y (2/3)

```
prog : mthdcls
      { printf("Program -> MethodDecls\n");
        printf("Parsed OK!\n"); }
      |
      { printf("***** Parsing failed!\n"); }
      ;
```

```
mthdcls : mthdcl mthdcls
          { printf("MethodDecls -> MethodDecl MethodDecls\n"); }
          |
          mthdcl
          { printf("MethodDecls -> MethodDecl\n"); }
          ;
```

```
type : lINT
        { printf("Type -> INT\n"); }
        |
        lREAL
        { printf("Type -> REAL\n"); }
        ;
```

```
mthdcl : type lMAIN lID lLP formals lRP block
         { printf("MethodDecl -> Type MAIN ID LP Formals RP Block\n"); }
         |
         type lID lLP formals lRP block
         { printf("MethodDecl -> Type ID LP Formals RP Block\n"); }
         ;
```

```
formals : formal oformal
          { printf("Formals -> Formal OtherFormals\n"); }
          |
          { printf("Formals -> \n"); }
          ;
```

```
formal : type lID
          { printf("Formal -> Type ID\n"); }
          ;
```

```
oformal : lCOMMA formal oformal
           { printf("OtherFormals -> COMMA Formal OtherFormals\n"); }
           |
           { printf("OtherFormals -> \n"); }
           ;
```

High-level program structures:

Program -> MethodDecl MethodDecl*

Type -> INT | REAL

MethodDecl -> Type [MAIN] Id '(' (FormalParams ')' Block

FormalParams -> [FormalParam (',' FormalParam)*]

FormalParam -> Type Id

T_PARSE.Y (3/3)

Understand the rules and write them here!

// Statements and Expressions

%%

```
int yyerror(char *s)
{
    printf("%s\n",s);
    return 1;
}
```

No need to change!

Statements:

Block -> BEGIN Statement+ End

Statement -> Block
| LocalVarDecl
| AssignStmt
| ReturnStmt
| IfStmt
| WriteStmt
| ReadStmt

LocalVarDecl -> Type Id ';' | Type AssignStmt

AssignStmt -> Id := Expression ';' ;

ReturnStmt -> RETURN Expression ';' ;

IfStmt -> IF '(' BoolExpression ')' Statement
| IF '(' BoolExpression ')' Statement ELSE Statement

WriteStmt -> WRITE '(' Expression ',' QString ')' ';' ;

ReadStmt -> READ '(' Id ',' QString ')' ';' ;

Expressions:

Expression -> MultiplicativeExpr ('+' | '-') MultiplicativeExpr *

MultiplicativeExpr -> PrimaryExpr ('*' | '/') PrimaryExpr *

PrimaryExpr -> Num // Integer or Real numbers
| Id
| '(' Expression ')'
| Id '(' ActualParams ')'

BoolExpr -> Expression '==' Expression
| Expression '!=' Expression
| Expression '>' Expression
| Expression '>=' Expression
| Expression '<' Expression
| Expression '<=' Expression

ActualParams -> [Expression (',' Expression)*]

THAT'S ALL FOR THE LECTURE!

By Professor [Chung Yung](#).

