งานชิ้นที่ 2 : Elementary Compiler

จัดทำโดย

นายกษิดิส รัตนวิจิตร 59010056

นายณัฐปัตย์ พิมพ์ทอง 59010444

นายบัณฑิต สีดาว 59010759

นายปฐวี สุทธิโฉม 59010780

เสนอ

รศ. ดร.เกียรติกูล เจียรนัยธนะกิจ

ผศ.อัครเดช วัชระภูพงษ์

รายงานนี้เป็นส่วนหนึ่งของวิชา Compiler Construction (01076262)

ภาคการเรียนที่ 2 ปีการศึกษา 2561

ภาควิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

สารบัญ

การออกแบบภาษา Hunnaeee

Token	Regular Expression	ตัวอย่าง
ค่าคงที่		
จำนวนเต็มฐาน 10	[-]?[0-9]+	-1 , -500 , 0 , 10 , 1000
จำนวนเต็มฐาน 16	0[xX][0-9a-fA-F]+	0x0000 , 0xAAA , 0XE4
ตัวแปรพื้นฐาน		
จำนวนเต็มคิดเครื่องหมายขนาด 64 บิต	[a-z]{2}	ab , bz , er
นิพจน์คำนวณจำนวนเต็ม		
ติดลบ	“-”	-aa , -5 , -0
คูณ	“*”	aa * bb , 0 * 5 , 10 * qq
หารเอาส่วน	“/”	aa / bb , 11 / 5 , 10 / qq
หารเอาเศษ	“%”	aa % bb , 11 % 5 , 10 % qq
บวก	“+”	aa + bb , 11 + 5 , 10 + qq
ลบ	“-”	aa - bb , 11 - 5 , 10 - qq
วงเล็บ	“(” , “)”	(5*8)+(3-1)
ประโยคคำสั่งที่ทำตามลำดับ		
ให้ค่าแก่ตัวแปร (assignment)	“=”	aa = 5
แสดงค่าตัวแปรโดดแบบฐาน 10	“print”	print 5
แสดงค่าตัวแปรโดดแบบฐาน 16	“print”	print hex(78)
แสดงสายอักษรที่กำหนด (ยาวไม่เกิน 255 ตัว)	“print”	print “...”
ประโยคคาสั่งตัดสินใจ		
เท่ากัน	“==”	0 == 0
มากกว่าหรือเท่ากับ	“>=”	5 >= 5 , 5 >= 0
น้อยกว่าหรือเท่ากับ	“<=”	1 <= 10 , 10 <= 10
มากกว่า	“>”	5 > 0
น้อยกว่า	“<”	1 < 10

ประโยคเงื่อนไข แบบที่ 1	“if”	if pm == 0 {
		pr[id] = fm
		id = id + 1
		}

ประโยคเงื่อนไข แบบที่ 2	“else”	else fm % st == 0 {
		pm = 1
		}

ประโยคคาสั่งวนซ้ำ		

For loop	“for”	for fm : to {
		pm = 0; st = 2
		}

แนวคิด และรายละเอียดวิธีการดำเนินงานสร้างเชิงเทคนิค

แนวคิดคือสร้าง Compiler ที่คงความเป็นภาษา C ไว้ แต่ได้ตัดไวยาการ์บางสิ่งออกเพื่อสร้างความสะดวกในการใช้งาน เช่น สามารถจบคำสั่งได้โดยการขึ้นบรรทัดใหม่ ไม่ต้องใช้ “;” เพื่อจบบรรทัด แต่ก็ยังสามารถใช้ “;” ได้ในกรณีที่ต้องการให้ใน 1 บรรทัดมีหลายคำสั่ง หรือไม่ว่าจะเป็นการ ประกาศตัวแปรที่คล้ายคลึงกับภาษา java script หรือว่าปริ้นภาพออกหน้าจอที่มีความคล้ายคลึงกับภาษา python เป็นต้น

เทคนิคที่ใช้คือ ใช้ flex ในการตัด input ต่าง ๆ ออกเป็น token เพื่อส่งให้ bison มาวิเคราะห์ไวยากรณ์ว่าถูกต้องหรือไม่ ครบทุกกรณีหรือไม่ แล้วจึงทำการสร้าง abstract syntax tree หลังจากนั้นจึงทำการท่องไปตาม tree เพื่อทำการสร้างภา assembly

ไวยากรณ์และตัวอย่างที่ทำให้เข้าใจภาษานั้นง่าย

ไวยากรณ์

```
program:
| program stmt      {
                      asmGen($2);
                      freeNode($2);
                    }
| program error ';' { errorflag = 1; yyerrok; }
;

stmt:
  exp
|  exp ';'
| if
| T_FOR exp ':' exp '{' block '}'      { $$ = newFor($2, $4, $6); }
;

if:
  T_IF exp T_EQ exp '{' block '}'      { $$ = newIf($2, $4, $6, NULL, 'e'); }
| T_IF exp T_NE exp '{' block '}'      { $$ = newIf($2, $4, $6, NULL, 'n'); }
| T_IF exp T_GE exp '{' block '}'      { $$ = newIf($2, $4, $6, NULL, 'h'); }
| T_IF exp T_LE exp '{' block '}'      { $$ = newIf($2, $4, $6, NULL, 'm'); }
| T_IF exp T_GT exp '{' block '}'      { $$ = newIf($2, $4, $6, NULL, 'g'); }
| T_IF exp T_LT exp '{' block '}'      { $$ = newIf($2, $4, $6, NULL, 'l'); }
```

```
| T_IF exp T_EQ exp '{' block '}' T_ELSE '{' block '}'  { $$ = newIf($2, $4, $6,
$10, 'e'); }
| T_IF exp T_NE exp '{' block '}' T_ELSE '{' block '}'  { $$ = newIf($2, $4, $6,
$10, 'n'); }
| T_IF exp T_GE exp '{' block '}' T_ELSE '{' block '}'  { $$ = newIf($2, $4, $6,
$10, 'h'); }
| T_IF exp T_LE exp '{' block '}' T_ELSE '{' block '}'  { $$ = newIf($2, $4, $6,
$10, 'm'); }
| T_IF exp T_GT exp '{' block '}' T_ELSE '{' block '}'  { $$ = newIf($2, $4, $6,
$10, 'g'); }
| T_IF exp T_LT exp '{' block '}' T_ELSE '{' block '}'  { $$ = newIf($2, $4, $6,
$10, 'e'); }

exp:
  term
| T_CONST VAR T_ASSIGN exp         { $$ = newDeclar($2, $4, 1); }
| T_VAR VAR T_ASSIGN exp         { $$ = newDeclar($2, $4, 0); }
| T_VAR VAR                { $$ = newDeclar($2, NULL, 0); }
| T_VAR VAR '[' NUM ']'    { $$ = newArray($2, $4); }
| VAR T_ASSIGN exp         { $$ = newAssign($1, $3); }
| VAR '[' exp ']' T_ASSIGN exp     { $$ = newArrayAssign($1, $6, $3); }
| exp '+' exp               { $$ = newNode($1, $3, '+'); }
| exp '-' exp               { $$ = newNode($1, $3, '-'); }
| exp '*' exp               { $$ = newNode($1, $3, '*'); }
| exp '/' exp               { $$ = newNode($1, $3, '/'); }
| exp '%' exp               { $$ = newNode($1, $3, '%'); }
| '^' exp %prec NEG         { $$ = newNode($2, NULL, '^'); }
| '(' exp ')'               { $$ = $2; }
| T_PRINT TEXT               { $$ = newPrint(NULL, $2, 'S'); }
| T_PRINT exp                { $$ = newPrint($2, NULL, 'D'); }
| T_PRINT T_HEX '(' exp ')'                { $$ = newPrint($4, NULL, 'H'); }
;

term:
  NUM                      { $$ = newNum($1); }
| VAR                      { $$ = newVar($1); }
| VAR '[' exp ']'          { $$ = newVarArray($1, $3); }
;

block:
                           { $$ = NULL; }
| stmt block               {
```

```
                                    if ($2 == NULL) {
                                      $$ = $1;
                                    } else {
                                      $$ = newNode($1, $2, 'B');
                                    }
                                  }
        ;
```

ตัวอย่าง

```
var fm = 2                          //ประกาศตัวแปรแบบ variable
var to = 0x64                       //ประกาศค่าตัวแปรเป็นเลขฐาน16
const mi = 9223372036854775807      //ประกาศตัวแปรแบบ constant
print "MAX INT: "                   //แสดงผลข้อความ
print mi                            //แสดงค่าของตัวแปร
print "\n"                          //เว้นบรรทัด
print hex(in)                       //แสดงค่าเลขฐาน16
var pr[25]                          //ประกาศตัวแปรแบบอาเรย์
for fm : to {                       //สร้าง for loop
    pm = 0; st = 2
        for st : fm {               //สร้าง loop ซ้อน loop
                if fm % st == 0 {   //สร้างฟังก์ชันเงื่อนไข
                        pm = 1
                }
        }
        if pm == 0 {                //สร้างฟังก์ชันเงื่อนไข
         pr[id] = fm                //กำหนดค่าให้อาเรย์
         id = id + 1
        }
}
```

คำอธิบายโค้ดในไฟล์ flex และ bison

Flex

```
%option noyywrap nodefault yylineno
%{
  #include "asmgen.h"
  #include "node.h"
  #include "parser.tab.h"
  #include <stdio.h>
  #include <stdlib.h>
  #include <math.h>
  #include <ctype.h>
  #include <string.h>

  extern int errorflag;
%}

%%
[ \t\v\f]           {}
"//"[^\n]*          {}
"/*"                {comment();}

"(" |
")" |
"[" |
"]" |
"{" |
"}" |
"+" |
"-" |
"*" |
"/" |
"%" |
"^" |
":" |
";" |
","               { return (yytext[0]); }

"=="          { return (T_EQ); }
"!="          { return (T_NE); }
">="          { return (T_GE); }
```

```
"<="          { return (T_LE); }
">"          { return (T_GT); }
"<"          { return (T_LT); }
"="             { return (T_ASSIGN); }
"print"          { return (T_PRINT); }
"hex"          { return (T_HEX); }
"const"          { return (T_CONST); }
"var"          { return (T_VAR); }
"if"             { return (T_IF); }
"else"             { return (T_ELSE); }
"for"             { return (T_FOR); }

[-]?[0-9]+           { yylval.num = (int64_t)atol(yytext); return (NUM); }
0[xX][0-9a-fA-F]+  { yylval.num = (int64_t)strtol(yytext, NULL, 0); return (NUM);}
[a-z]{2}        { yylval.sym = lookup(yytext, 0, 0); return (VAR); }
["].*["]          { yylval.str = strdup(yytext); return (TEXT); }

\n                {  }
.                 { errorflag = 1; yyerror("Mystery character %c\n", *yytext); }
%%

void comment() {
       char c, c1;

loop:
       while ((c = input()) != '*' && c != 0) {}

       if ((c1 = input()) != '/' && c != 0) {
              unput(c1);
              goto loop;
       }
}
```

Flex จะทำหน้าที่ตัดสตริงแบ่งเป็น token ซึ่งสตริงที่ส่งเข้ามานั้นจะต้องสามารถตัดเป็น token ได้ หรือภาษาของเรานั้นจะต้องตสามารถตัดเป็น token ได้ โดยใน flex นั้นจะมีลำดับความสำคัญของ token อยู่อ ซึ่งการคืนค่า token กับการเก็บค่านั้น จะมีความเกี่ยวข้องกับ bison ซึ่งจะอธิบายในส่วนต่อไป

Bison

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "asmgen.h"
#include "node.h"
extern int errorflag;
%}

%union {
    struct ast* node;
    struct symbol* sym;
    int64_t num;
    char* str;
}

%token<str> TEXT
%token<num> NUM
%token<sym> VAR
%token T_ASSIGN T_CONST T_VAR
%token T_NEWLINE
%token T_IF T_ELSE T_FOR
%token T_PRINT T_HEX

%right T_ASSIGN
%left '+' '-'
%left '*' '/' '%'
%left T_EQ T_NE T_GE T_LE T_GT T_LT
%nonassoc NEG

%type <node> stmt exp if term block

%start program

%%

program:
| program stmt      {
                      asmGen($2);
                      freeNode($2);
                    }
```

```
| program error ';' { errorflag = 1; yyerrok; }
;

stmt:
  exp
|  exp ';'
| if
| T_FOR exp ':' exp '{' block '}'      { $$ = newFor($2, $4, $6); }
;

if:
  T_IF exp T_EQ exp '{' block '}'      { $$ = newIf($2, $4, $6, NULL, 'e'); }
| T_IF exp T_NE exp '{' block '}'      { $$ = newIf($2, $4, $6, NULL, 'n'); }
| T_IF exp T_GE exp '{' block '}'      { $$ = newIf($2, $4, $6, NULL, 'h'); }
| T_IF exp T_LE exp '{' block '}'      { $$ = newIf($2, $4, $6, NULL, 'm'); }
| T_IF exp T_GT exp '{' block '}'      { $$ = newIf($2, $4, $6, NULL, 'g'); }
| T_IF exp T_LT exp '{' block '}'      { $$ = newIf($2, $4, $6, NULL, 'l'); }

| T_IF exp T_EQ exp '{' block '}' T_ELSE '{' block '}'  { $$ = newIf($2, $4, $6,
$10, 'e'); }
| T_IF exp T_NE exp '{' block '}' T_ELSE '{' block '}'  { $$ = newIf($2, $4, $6,
$10, 'n'); }
| T_IF exp T_GE exp '{' block '}' T_ELSE '{' block '}'  { $$ = newIf($2, $4, $6,
$10, 'h'); }
| T_IF exp T_LE exp '{' block '}' T_ELSE '{' block '}'  { $$ = newIf($2, $4, $6,
$10, 'm'); }
| T_IF exp T_GT exp '{' block '}' T_ELSE '{' block '}'  { $$ = newIf($2, $4, $6,
$10, 'g'); }
| T_IF exp T_LT exp '{' block '}' T_ELSE '{' block '}'  { $$ = newIf($2, $4, $6,
$10, 'e'); }

exp:
  term
| T_CONST VAR T_ASSIGN exp         { $$ = newDeclar($2, $4, 1); }
| T_VAR VAR T_ASSIGN exp         { $$ = newDeclar($2, $4, 0); }
| T_VAR VAR                { $$ = newDeclar($2, NULL, 0); }
| T_VAR VAR '[' NUM ']'    { $$ = newArray($2, $4); }
| VAR T_ASSIGN exp         { $$ = newAssign($1, $3); }
| VAR '[' exp ']' T_ASSIGN exp     { $$ = newArrayAssign($1, $6, $3); }
| exp '+' exp               { $$ = newNode($1, $3, '+'); }
| exp '-' exp               { $$ = newNode($1, $3, '-'); }
| exp '*' exp               { $$ = newNode($1, $3, '*'); }
| exp '/' exp               { $$ = newNode($1, $3, '/'); }
```

```
        | exp '%' exp               { $$ = newNode($1, $3, '%'); }
        | '^' exp %prec NEG         { $$ = newNode($2, NULL, '^'); }
        | '(' exp ')'               { $$ = $2; }
        | T_PRINT TEXT               { $$ = newPrint(NULL, $2, 'S'); }
        | T_PRINT exp                { $$ = newPrint($2, NULL, 'D'); }
        | T_PRINT T_HEX '(' exp ')'               { $$ = newPrint($4, NULL, 'H'); }
        ;

        term:
          NUM                       { $$ = newNum($1); }
        | VAR                       { $$ = newVar($1); }
        | VAR '[' exp ']'           { $$ = newVarArray($1, $3); }
        ;

        block:
                                    { $$ = NULL; }
        | stmt block                {
                                      if ($2 == NULL) {
                                        $$ = $1;
                                      } else {
                                        $$ = newNode($1, $2, 'B');
                                      }
                                    }
        ;
```

Bison จะรับ token ที่ได้จาก flex มาทำการตรวจสอบกับ grammar ว่าจะเข้ากันได้หรือไม่ โดย ลำดับของ grammar นั้นมีความสำคัญเหมือนกันโดย grammar ที่อยู่อันดับบนนั้นจะมี node ที่สูงกว่า grammar ที่อันดับต่ำกว่า ซึ่ง token ของเรานั้นจะต้องเข้ากับ grammar ทั้งหมด ซึ่งในแต่ละ grammar นั้นจะทำการสร้าง node ขึ้นมาในของแต่ละชนิด โดย node ที่ต่ำกว่าจะถูกชี้ด้วย node ที่สูงกว่า เช่น node if โดยค่าในของแต่ละ node จะถูกเก็บด้วย struct

ผลการรันกับตัวอย่าง

รูปแบบที่ถูกต้อง

```
min-arrayvalues.simple                                                buffers
 1 var ay[20]
 2
 3
 4 print "print loop mn\n"
 5
 6 var id = 0
 7 for id : 20 {
 8     print "ay["
 9     print id
10     print "]: "
11
12     print ay[id]
13     print "\n"
14 }
15
16
17 var mn = 0x7fffffffffffffff
18 id = 0
19 for id : 20 {
20     if mn > ay[id] {
21         mn = ay[id]
22     }
23 }
24
25 print "mn: "
26 print mn
```

```
j@bdintu build → git:(master) ./simple example/min-arrayvalues.simple
j@bdintu build → git:(master) gcc -no-pie example/min-arrayvalues.s
j@bdintu build → git:(master) ./a.out
print loop mn
ay[0]: 0
ay[1]: 0
ay[2]: 0
ay[3]: 0
ay[4]: 0
ay[5]: 0
ay[6]: 11
ay[7]: 15775231
ay[8]: 194
ay[9]: 140733578642422
ay[10]: 1
ay[11]: 140664765638677
ay[12]: 0
ay[13]: 4199373
ay[14]: 140664767142000
ay[15]: 0
ay[16]: 4199296
ay[17]: 4198464
ay[18]: 140733578642688
ay[19]: 0
mn: 0
j@bdintu build → git:(master)
```

max-threevalues.simple

```
var xx = 9
var yy = 2
var zz = 5

print "xx: "
print xx
print "\n"

print "yy: "
print yy
print "\n"

print "zz: "
print zz
print "\n"

var mx = xx

if yy > mx {
    mx = yy
}

if zz > mx {
    mx = zz
}

print "mx: "
print mx
```

```
j@bdintu build → git:(master) ./simple example/max-threevalues.simple
j@bdintu build → git:(master) gcc -no-pie example/max-threevalues.s
j@bdintu build → git:(master) ./a.out
xx: 9
yy: 2
zz: 5
mx: 9%
j@bdintu build → git:(master)
```

for.simple

```
for 0 : 5 {
    print "x"
}
```

```
j@bdintu build → git:(master) ./simple example/for.simple
j@bdintu build → git:(master) gcc -no-pie example/for.s
j@bdintu build → git:(master) ./a.out
xxxxx%
j@bdintu build → git:(master)
```

for-with-variable.simple

```
1 var ii = 0
2 for ii : 5 {
3     print ii
4 }
```

```
j@bdintu build → git:(master) ./simple example/for-with-variable.simple
j@bdintu build → git:(master) gcc -no-pie example/for-with-variable.s
j@bdintu build → git:(master) ./a.out
01234%
j@bdintu build → git:(master)
```

if.simple

```
1 if 1 == 1 {
2     print 1
3 }
```

```
j@bdintu build → git:(master) ./simple example/if.simple
j@bdintu build → git:(master) gcc -no-pie example/if.s
j@bdintu build → git:(master) ./a.out
1%
j@bdintu build → git:(master)
```

if-else.simple

```
1 if 1 != 1 {
2     print 1
3 } else {
4     print 0
5 }
```

```
j@bdintu build → git:(master) ./simple example/if-else.simple
j@bdintu build → git:(master) gcc -no-pie example/if-else.s
j@bdintu build → git:(master) ./a.out
0%
j@bdintu build → git:(master)
```

if-nested.simple

```
1 if 1 == 1 {
2     if 2 != 3 {
3         if 3 > 1 {
4             print "Hunnaeee"
5         }
6     } else {
7         print "Hahaha"
8     }
9 } else {
10     print "Oh nooo"
11 }
```

```
j@bdintu build → git:(master) ./simple example/if-nested.simple
j@bdintu build → git:(master) gcc -no-pie example/if-nested.s
j@bdintu build → git:(master) ./a.out
Hunnaeee
j@bdintu build → git:(master)
```

declaration.simple

```
1 const cc = 0x4a
2 var xx = -64
3
4 print cc
5 print "\n"
6
7 print xx
8 print "\n"
```

```
j@bdintu build → git:(master) ✗ ./simple example/declaration.simple
j@bdintu build → git:(master) ✗ gcc -no-pie example/declaration.s
j@bdintu build → git:(master) ✗ ./a.out
74
-64
j@bdintu build → git:(master) ✗
```

print.simple

```
print "Hello, world\n"

print "1+2 = "
print 1+2
print "\n"

print "64 in hex is "
print hex(64)
print "\n"
```

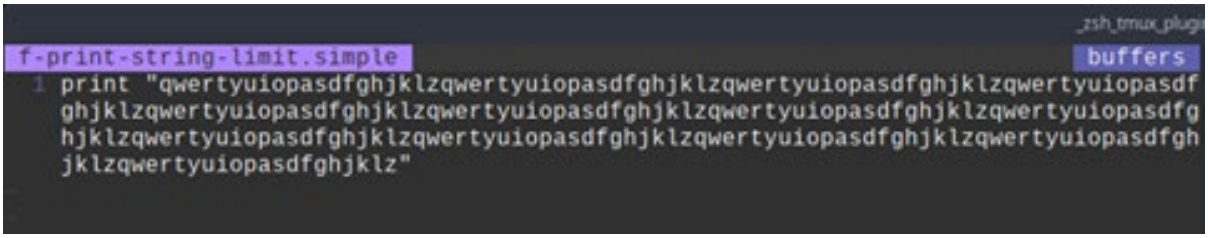
```
j@bdintu build → git:(master) ✗ ./simple example/print.simple
j@bdintu build → git:(master) ✗ gcc -no-pie example/print.s
j@bdintu build → git:(master) ✗ ./a.out
Hello, world
1+2 = 3
64 in hex is 0x40
j@bdintu build → git:(master) ✗
```

array.simple

```
var ay[10]

var ii = 0
for ii : 10 {
    ay[ii] = 10 - ii
}

ii = 0
for ii : 10 {
    print "ay["
    print ii
    print "]: "

    print ay[ii]
    print "\n"
}
```

```
j@bdintu build → git:(master) ✗ ./simple example/array.simple
j@bdintu build → git:(master) ✗ gcc -no-pie example/array.s
j@bdintu build → git:(master) ✗ ./a.out
ay[0]: 10
ay[1]: 9
ay[2]: 8
ay[3]: 7
ay[4]: 6
ay[5]: 5
ay[6]: 4
ay[7]: 3
ay[8]: 2
ay[9]: 1
j@bdintu build → git:(master) ✗
```

รูปแบบไม่ถูกต้อง



f-notdefined.simple

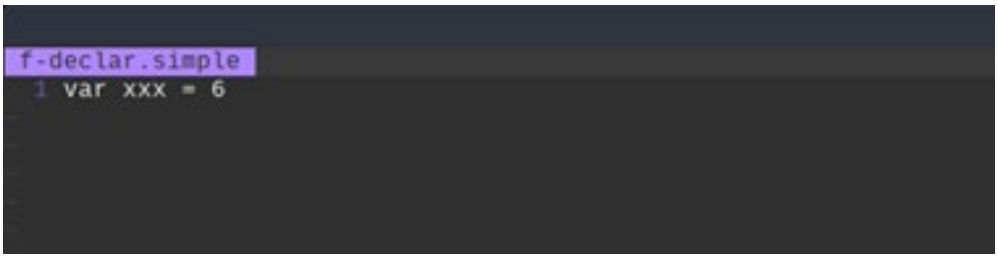
```
1 var aa = 5
2 print bb
```

```
j,run attach -tj
j@bdintu build → git:(master) ✗ ./simple test/f-notdefined.simple
line: 3, error: variable bb not defined
j@bdintu build → git:(master) ✗
```

f-print-unknow-error.simple

```
1 print Hello, world!\n
```

```
j@bdintu build → git:(master) ✗ ./simple test/f-print-unknow-error.simple
line: 1, error: unknow character H

line: 1, error: variable el not defined
j@bdintu build → git:(master) ✗
```


f-declar.simple
1 var xxx = 6


j run attach -t j
j@bdintu build ➜ git:(master) ✗ ./simple test/f-declar.simple
line: 1, error: unknow character x
Parsing Error
j@bdintu build ➜ git:(master) ✗

Source Code