**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 1](#_Toc7819917)

[แนวคิด และรายละเอียดวิธีการดำเนินงานสร้างเชิงเทคนิค 3](#_Toc7819918)

[ไวยากรณ์และตัวอย่างที่ทำให้เข้าใจภาษานั้นง่าย 3](#_Toc7819919)

[ไวยากรณ์ 3](#_Toc7819920)

[ตัวอย่าง 5](#_Toc7819921)

[คำอธิบายโค้ดในไฟล์ flex และ bison 6](#_Toc7819922)

[Flex 6](#_Toc7819923)

[Bison 8](#_Toc7819924)

[ผลการรันกับตัวอย่าง 11](#_Toc7819925)

[รูปแบบที่ถูกต้อง 11](#_Toc7819926)

[รูปแบบไม่ถูกต้อง 16](#_Toc7819927)

[Source Code 19](#_Toc7819928)

# **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

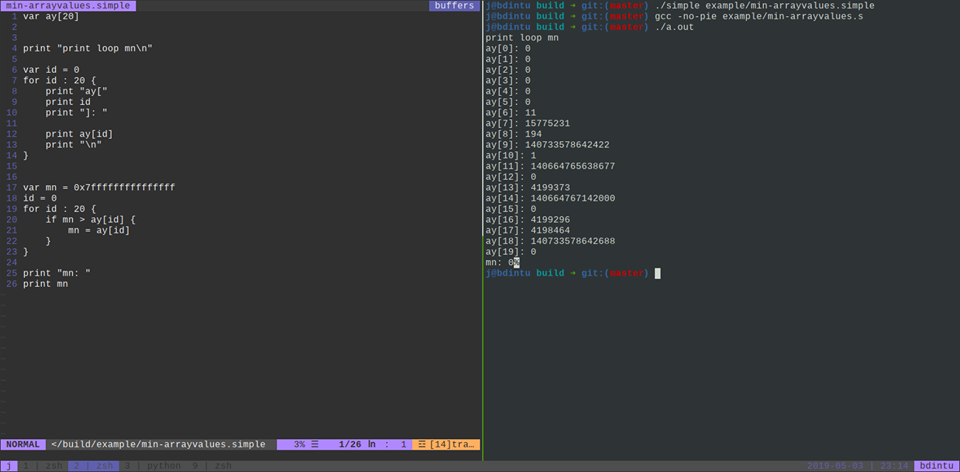
## **Bison**

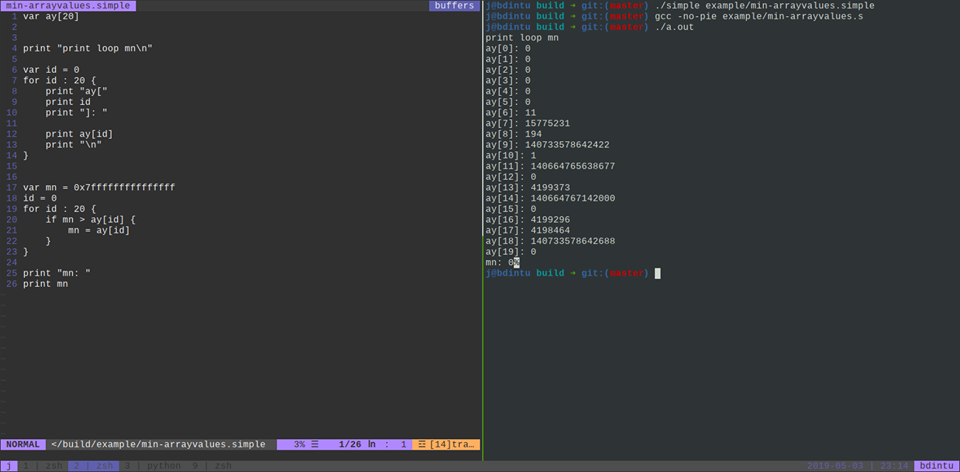
|  |
| --- |
|  |
|  | %{  #include <stdio.h> |
|  | #include <stdlib.h> |
|  | #include <stdint.h> |
|  | #include "asmgen.h" |
|  | #include "node.h" |
|  |  |
|  | extern int errorflag; |
|  |  |
|  | %} |
|  |  |
|  | %union { |
|  | struct ast\* node; |
|  | struct symbol\* sym; |
|  | int64\_t 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'); |
|  | } |
|  | } |
|  | ; |

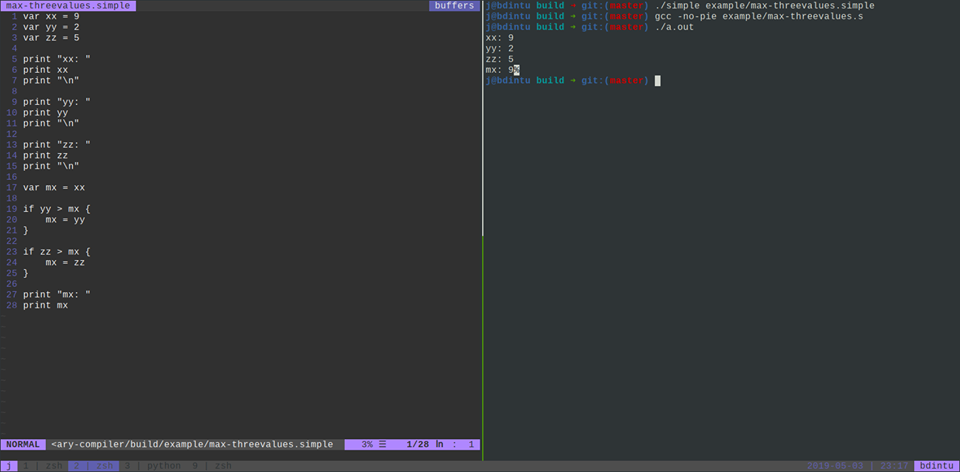
Bison จะรับ token ที่ได้จาก flex มาทำการตรวจสอบกับ grammar ว่าจะเข้ากันได้หรือไม่ โดย ลำดับของ grammar นั้นมีความสำคัญเหมือนกันโดย grammar ที่อยู่อันดับบนนั้นจะมี node ที่สูงกว่า grammar ที่อันดับต่ำกว่า ซึ่ง token ของเรานั้นจะต้องเข้ากับ grammar ทั้งหมด ซึ่งในแต่ละ grammar นั้นจะทำการสร้าง node ขึ้นมาในของแต่ละชนิด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

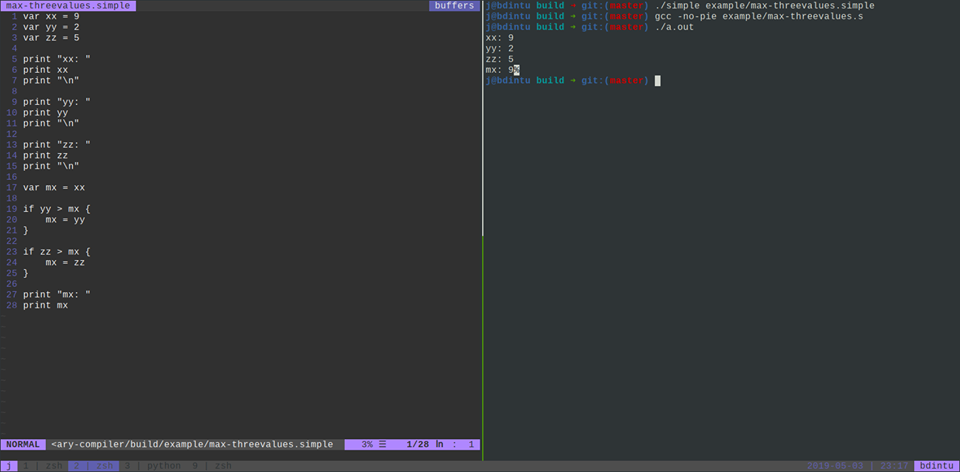
# **ผลการรันกับตัวอย่าง**

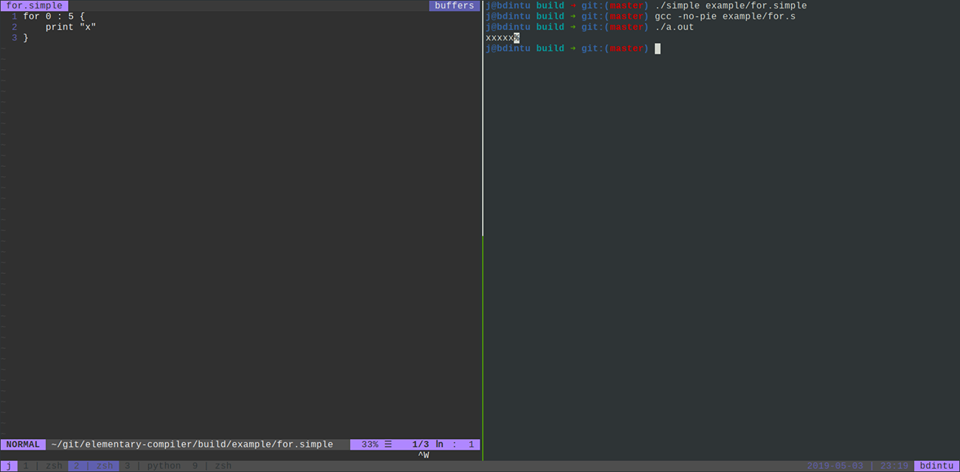
## **รูปแบบที่ถูกต้อง**

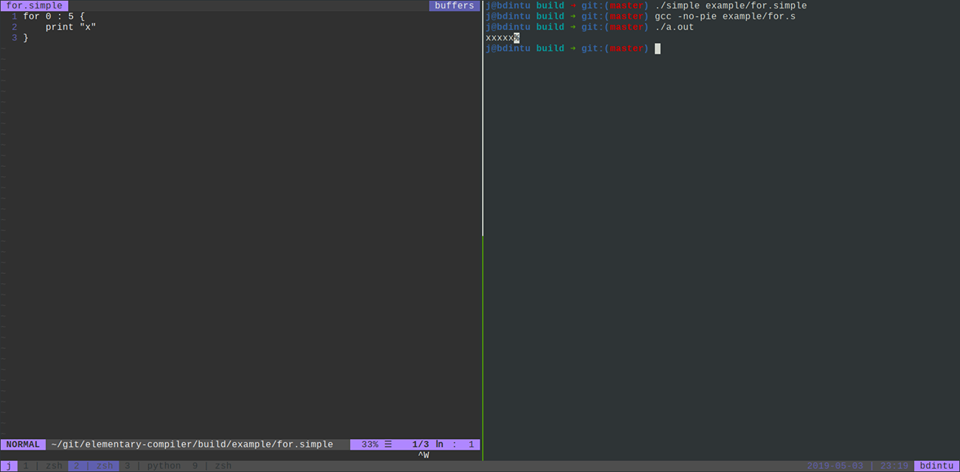


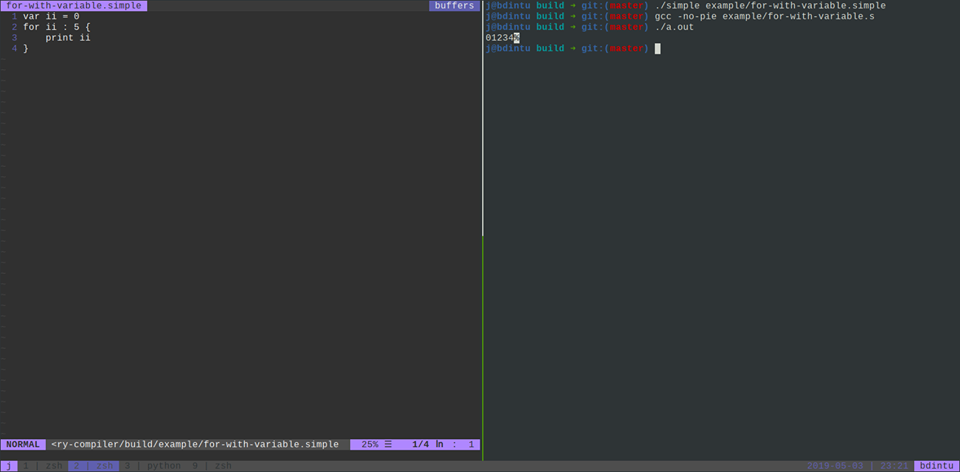


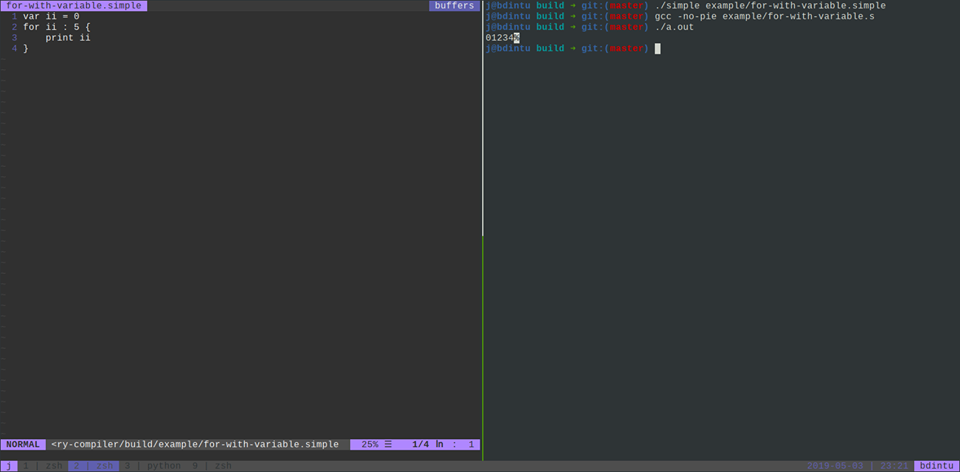


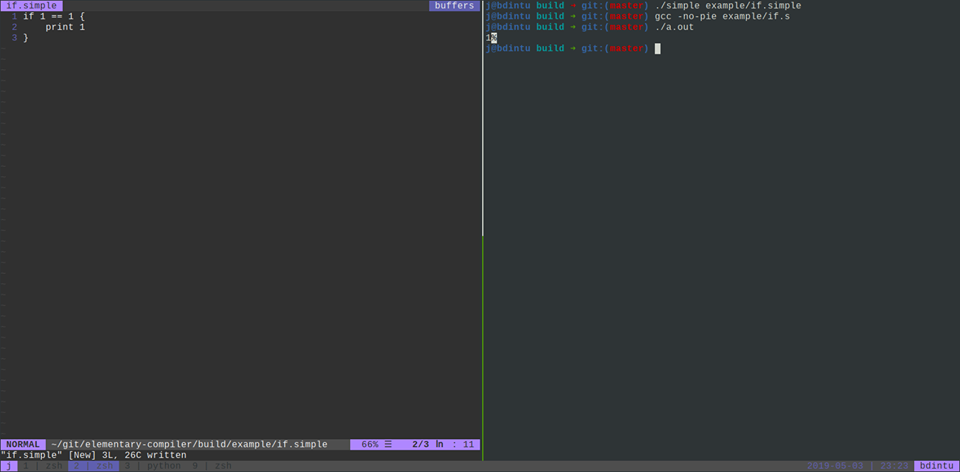


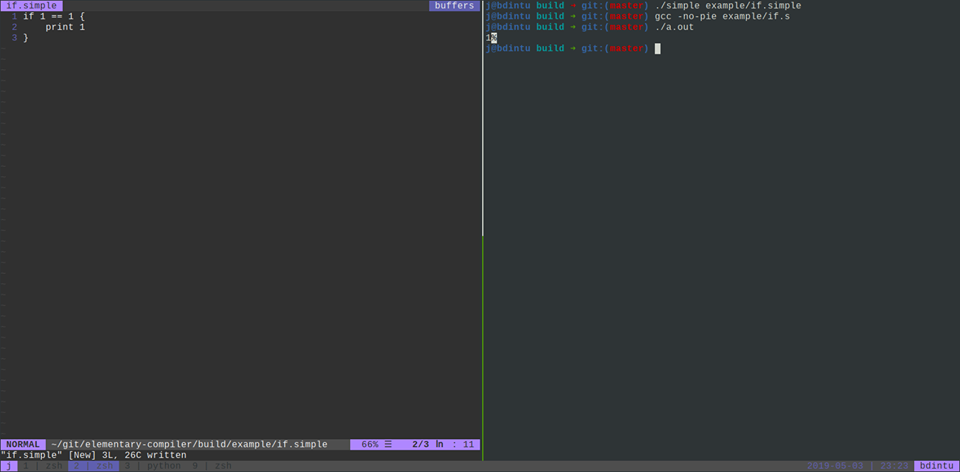


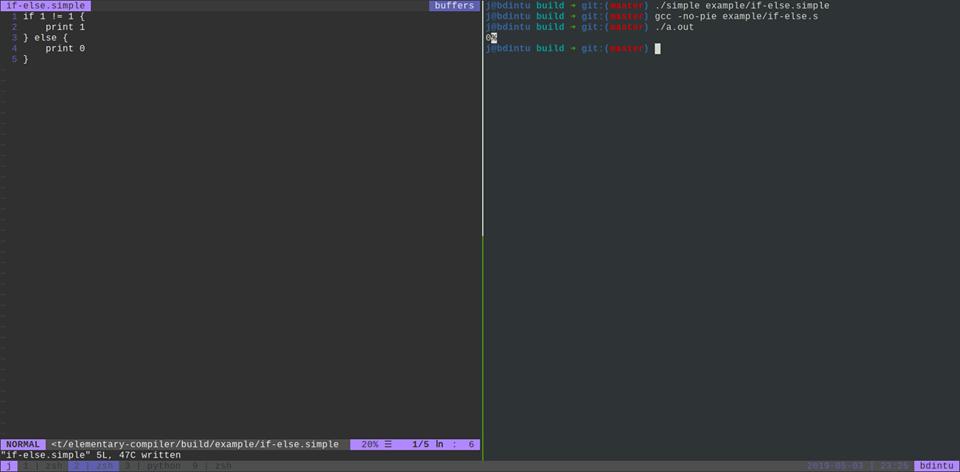


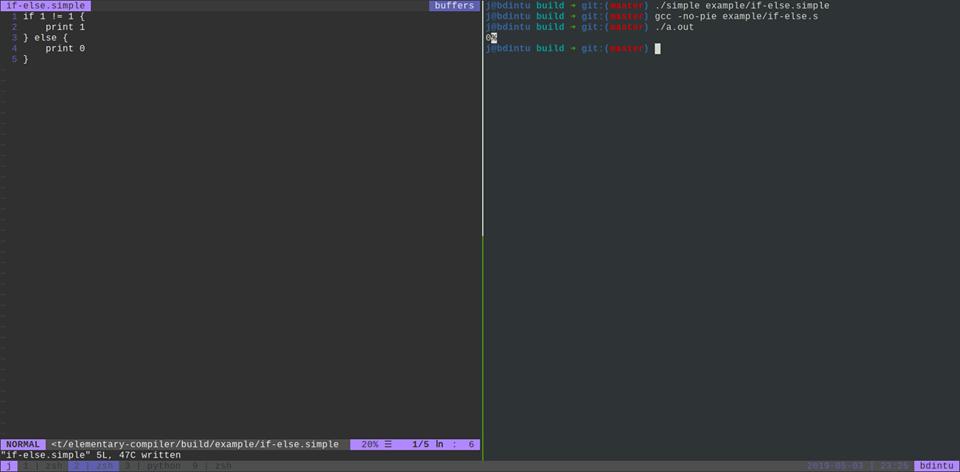


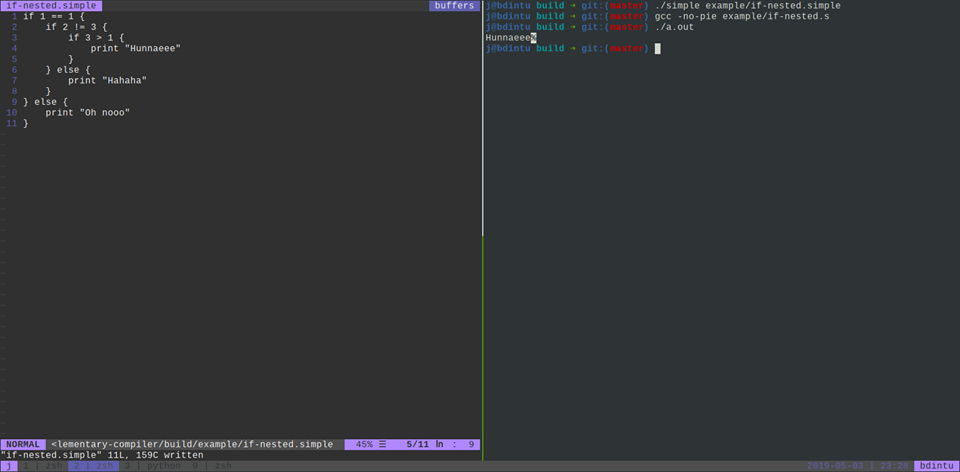


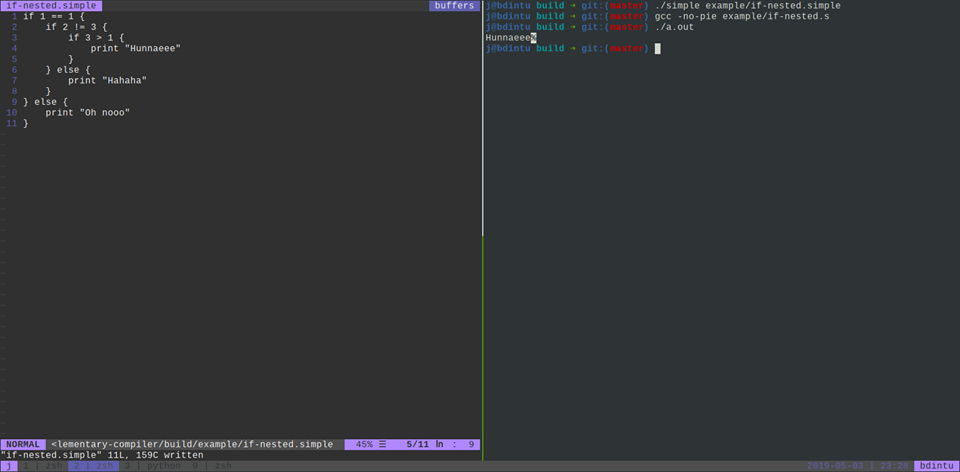


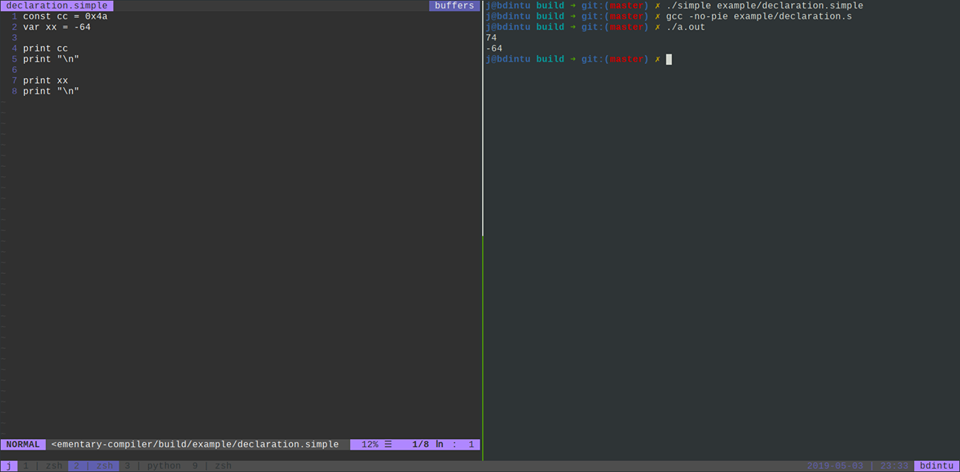


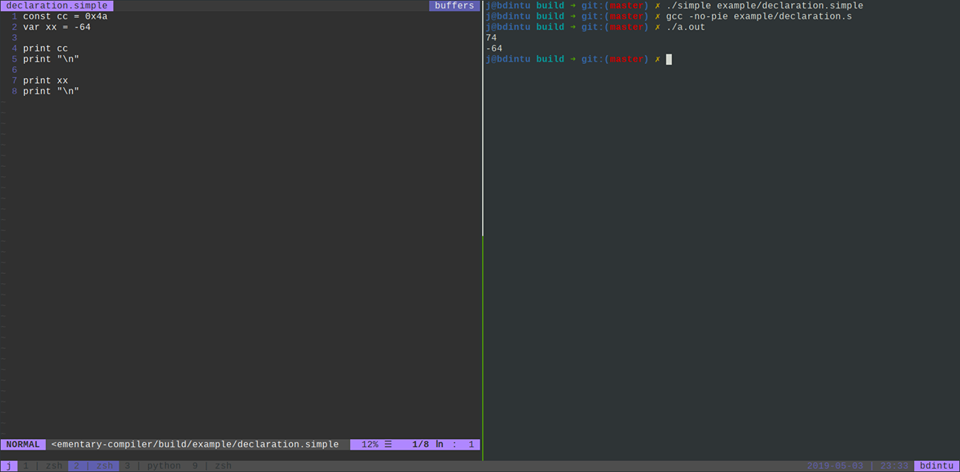


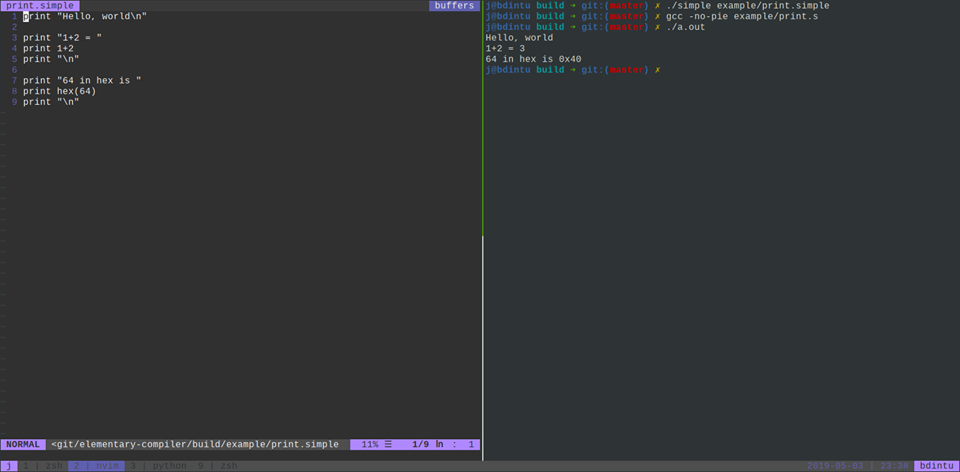


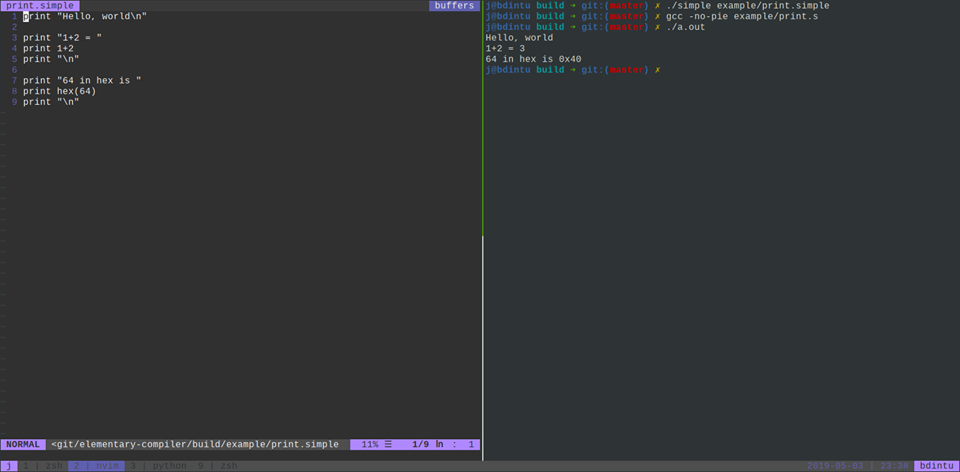


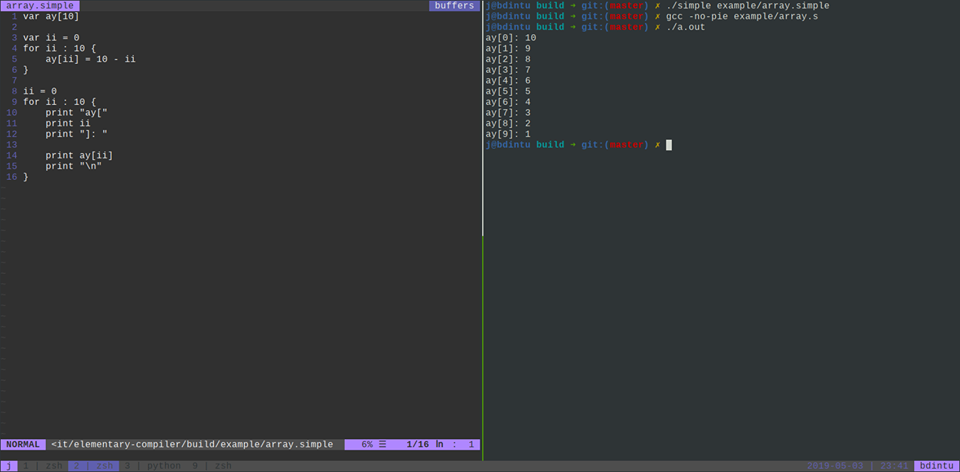


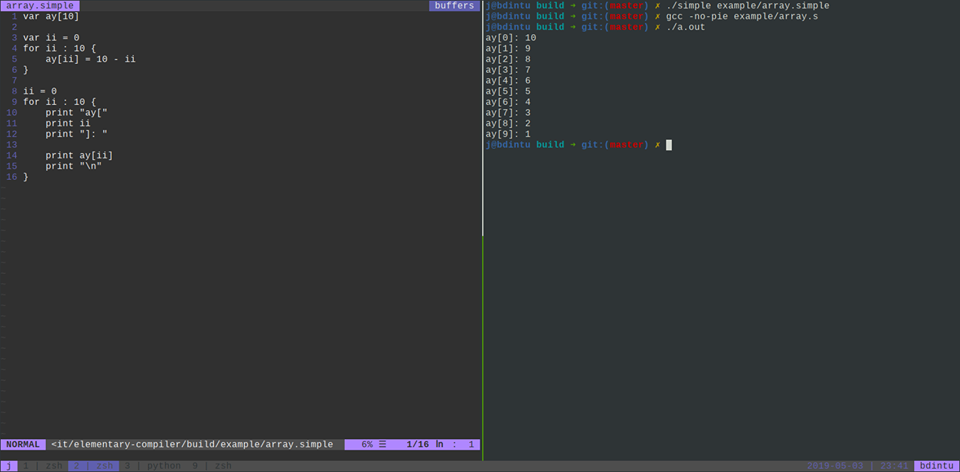




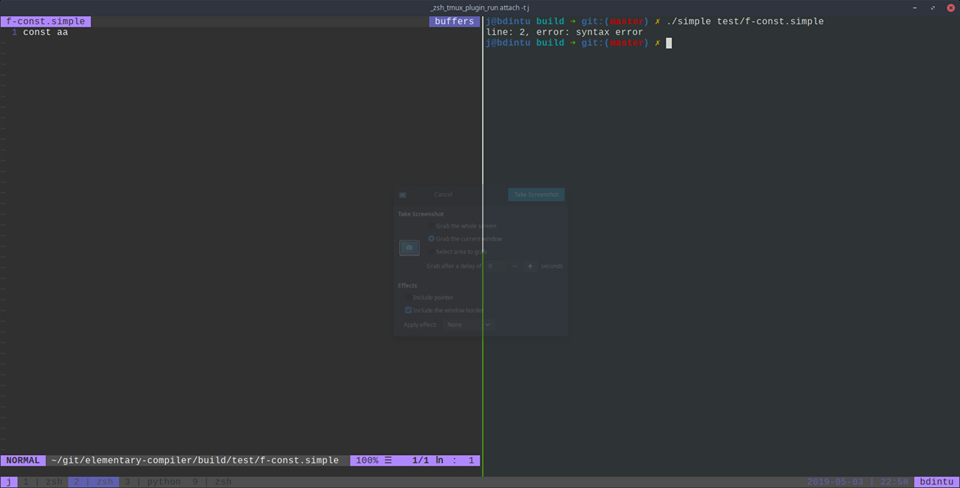


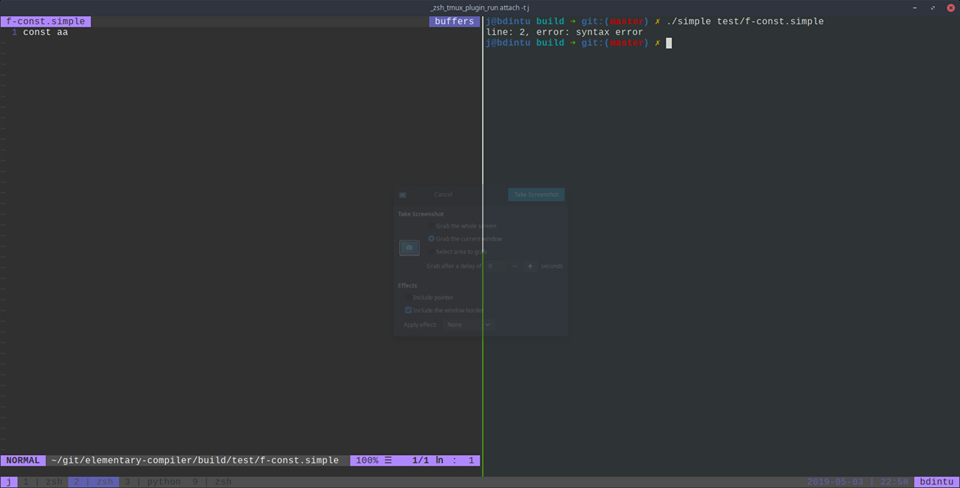


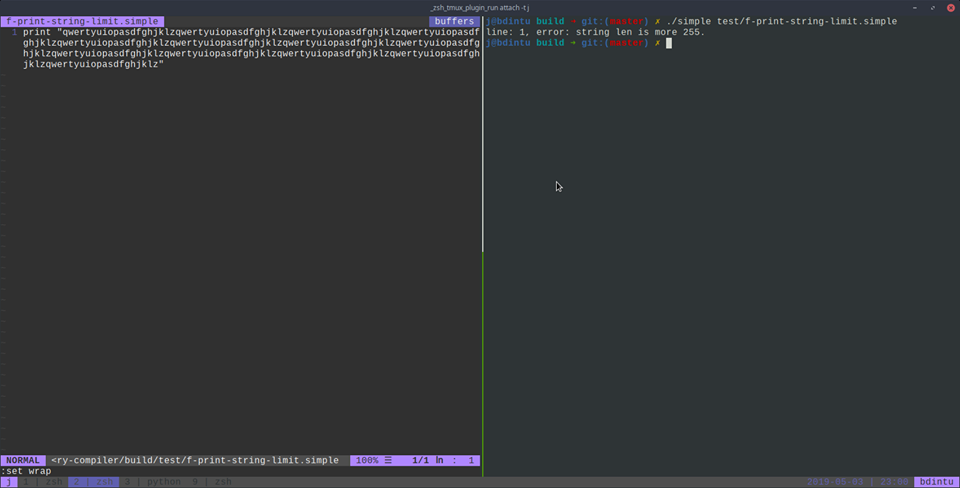


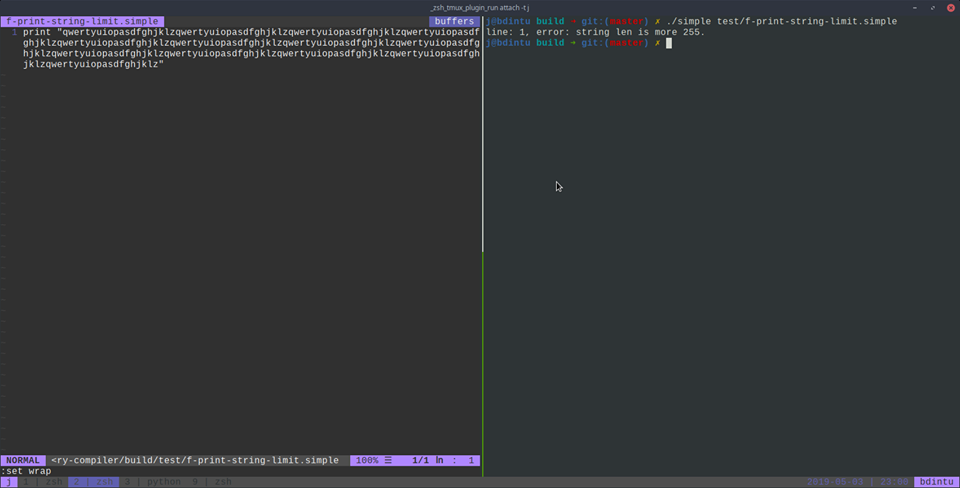


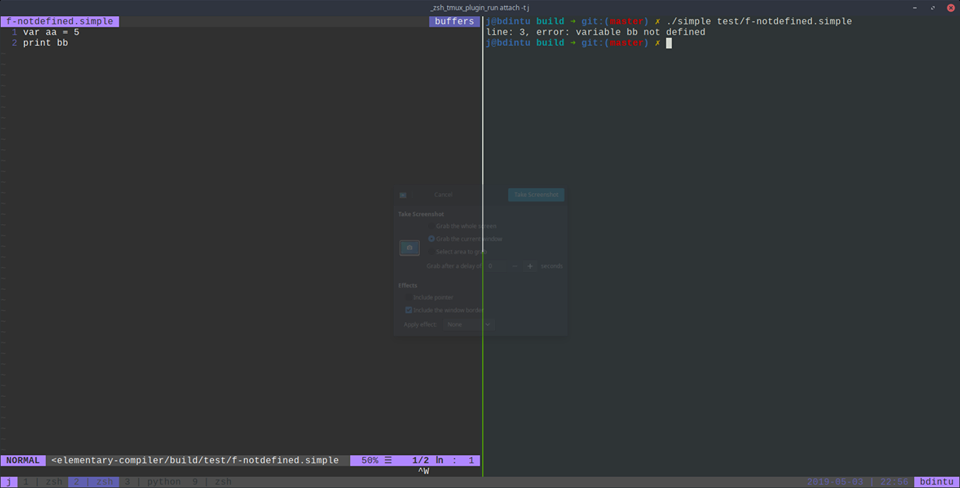
## **รูปแบบไม่ถูกต้อง**

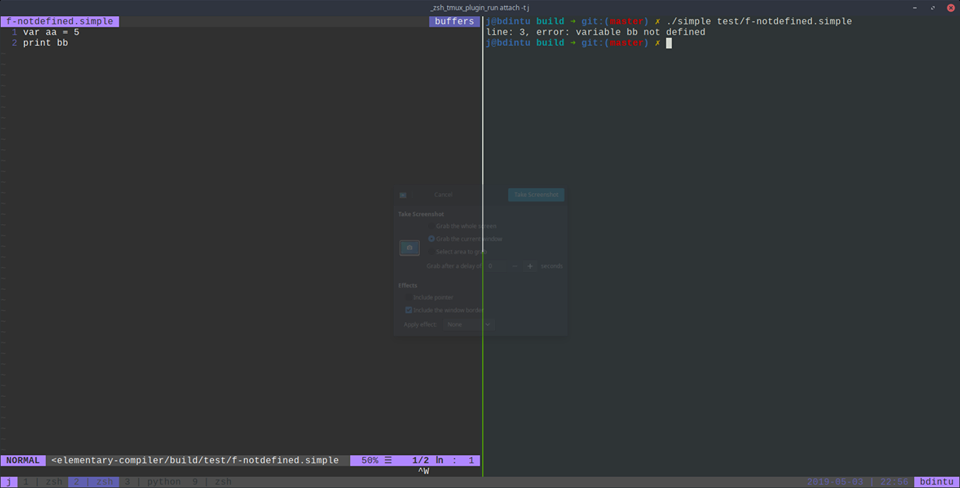


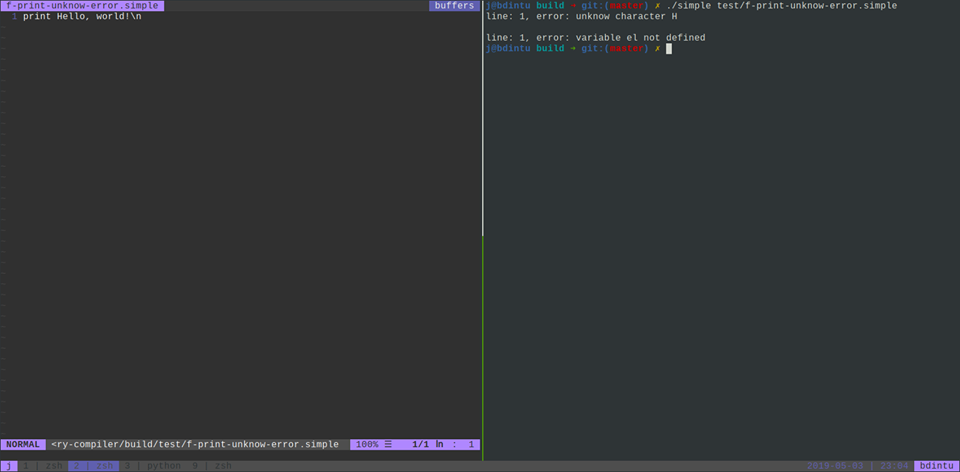


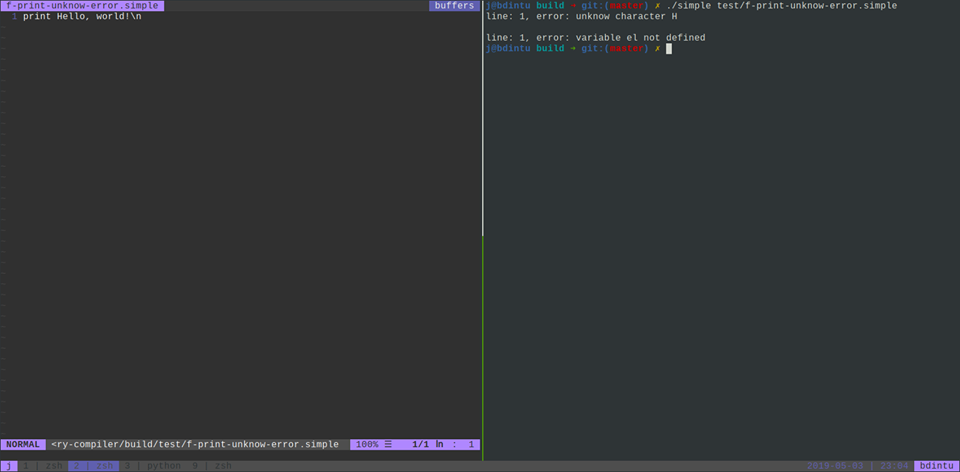


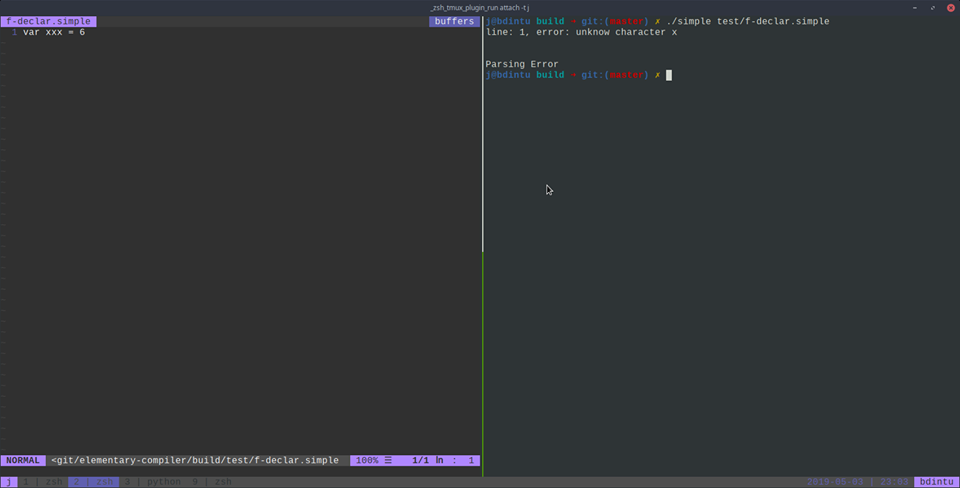


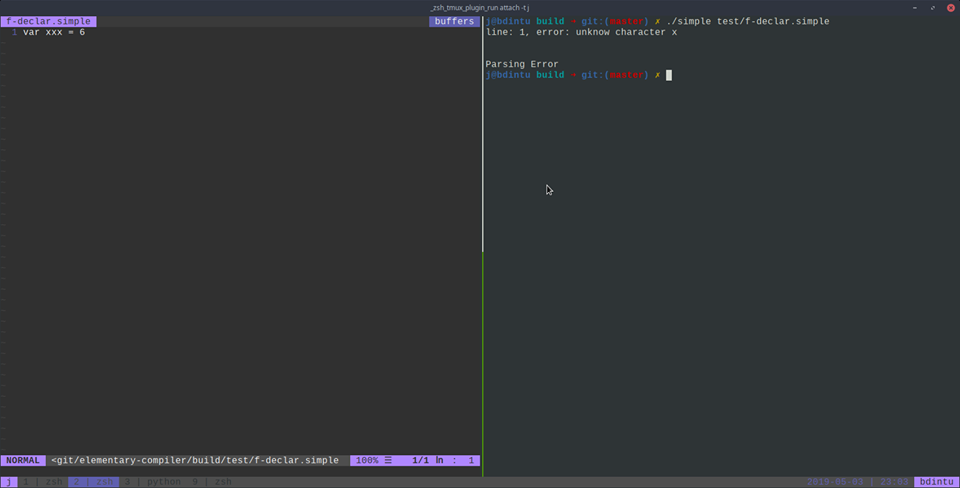












# **Source Code**

[](https://github.com/bdintu/elementary-compiler)