งานชิ้นที่ 3 วิชา 01076262 Compiler Construction

“Expression Evaluator”

จัดทำโดย

นางสาววรนิษฐา	ไกรสิทธิพงศ์	56011055
นายวิณัฐ	จิรฤกษ์มงคล	56011127
นายสรัล	รักษ์วิจิตรศิลป์	56011278

เสนอ

อ. อัครเดช วัชระภูพงษ์

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

Bison

GNU bison, commonly known as Bison, is a parser generator that is part of the GNU Project. Bison reads a specification of a context-free language, warns about any parsing ambiguities, and generates a parser
From: Wikipedia.org

Compiler generator is a programming tool that creates a parser, interpreter, or compiler from some form of formal description of a language and machine. The earliest and still most common form of compiler-compiler is a **parser generator**, whose input is a grammar
From: Wikipedia.org

- Bison นั้นคือ compiler generator หรือก็คือโปรแกรมที่ให้ output ออกมาเป็น source code (ภาษา C) ของ compiler ตาม input grammar ที่ป้อนเข้าไป
- Bison จะทำการอ่าน input grammar และ generate parser's source code เป็น output
- เมื่อได้ source code (ภาษา C) จึงนำมา compile, และ build สร้าง binary เพื่อทำการรัน
- โดย bison อนุญาตให้ผู้ใช้กำหนดการทำงานเมื่อ accepted input ใดๆ ได้อีกด้วย
- Parser (output ที่ได้จาก bison) นั้นจะทำการอ่านค่า input และพยายามสร้าง parse tree เพื่อตรวจสอบว่า accept หรือ reject input นั้น
- Input ที่ parser อ่านนั้น จะอยู่ในรูป tokens
- Tokens ที่นำมาให้ bison พิจารณา (bison's input) มาจาก output ของ lexical analyzer

Flex

Flex (fast lexical analyzer generator) is a free and open-source software alternative to lex. It is a computer program that generates lexical analyzers (also known as "scanners" or "lexers")
From: Wikipedia.org

Lexical analysis is the process of converting a sequence of characters (such as in a computer program or web page) into a sequence of tokens (strings with an identified "meaning")
From: Wikipedia.org

- Flex นั้นคือ lexical analysis หรือก็คือโปรแกรมที่ให้ output ออกมาเป็น source code (ภาษา C) ของ lexical analyzer ที่เราต้องการสร้าง ตาม input ที่กำหนด
- Output ของ lexical analyzer นั้นคือ tokens
- Tokens ที่ได้ขึ้นอยู่กับ string ที่อ่านเข้ามาเป็น input
- ส่งต่อ tokens แบบเรียงลำดับ เพื่อให้ compiler generator สร้าง parse tree ได้

เริ่มต้นการสร้างด้วยการนำ infix calculator เป็นตัวอย่าง ศึกษาและดัดแปลงจนกระทั่งได้สามารถบวก, ลบ และทำการคำนวณเบื้องต้นได้

ออกแบบ grammar เพื่อรองรับการคำนวณ

หลังจากนั้นทำการเพิ่ม grammar ส่วนที่รองรับ register, stack, และ error handling ตามลำดับ ได้ออกมาเป็น Mhee.y ใช้สำหรับสร้าง compiler และทำงาน ด้วย bison

```
exp :  CONSTANT                 { $$ = $1; }
        | exp OR exp            { $$ = $1 | $3; }
        | exp AND exp           { $$ = $1 & $3; }
        | NOT exp               { $$ = ~$2; }
        | exp '+' exp           { $$ = $1 + $3; }
        | exp '-' exp           { $$ = $1 - $3; }
        | exp '*' exp           { $$ = $1 * $3; }
        | exp '/' exp           { if($3 == 0){
                                        printf("Can't divide by 0.");
                                        yyerror();
                                  }
                                  else{ $$ = $1 / $3; }
                                }
        | exp '\\' exp          { if($3 == 0){
                                        printf("Can't mod by 0.");
                                        yyerror();
                                  }
                                  else{ $$ = $1 % $3; }
                                }
        | '-' exp               { $$ = -$2; }
        | exp '^' exp           { $$ = pow($1,$3); }
        | '(' exp ')'           { $$ = $2; }
```

Mhee.y

```
%{
        #include <stdio.h>
        #include <math.h>
        #include <stdlib.h>
        #define YYSTYPE int

        // Stack node struct //
        typedef struct node node;
        struct node{
                int data;
                node* next;
        };

        // Stack pointers //
        node* head = NULL;
        node* tail = NULL;

        // Register //
        int r[15];

        // Error flag //
        int er=0;

        // Functions declarations //
        void push(int data);
        void count();
        void setTopAndSize();
        int pop();
        int isEmpty();
%}
```

สร้าง node ในการ implement stack

สร้าง stack pointers

สร้าง array สำหรับ register

สร้าง error flag

```
// tokens //
%token R0 14 R1 1 R2 2 R3 3 R4 4 R5 5 R6 6 R7 7 R8 8 R9 9
%token ACC 11 TOP 12 SIZE 13
%token LOAD
%token SHOW
%token POP
%token PUSH
%token CONSTANT
%token ERR
%left OR
%left AND
%left '-' '+'
%left '*' '/' '\\'
%right NEG NOT
%right '^'
%left '(' ')'
%%
```

ประกาศ และกำหนดค่า tokens ต่างๆที่ใช้สร้าง parse tree

ประกาศ priority จากซ้ายไปขวา หรือขวาไปซ้าย

```
input:  | input line
line: '\n'
        | exp '\n'                { if(er==0) printf("%d\n",$1);
                                    else er=0;
                                  }
        | rexp '\n'               { if(er==0) printf("%d\n",$1);
                                    else er=0;
                                  }
        | SHOW reg '\n'           { printf("%d\n",r[$2]); }
        | LOAD reg '>' reg '\n'   { if( $4 == TOP || $4 == SIZE || $4 == ACC){
                                        printf("Can't assign register to $top or $size or $acc.\n");
                                        yyerror();
                                    }
                                    else r[$4] = r[$2];
                                  }
        | PUSH reg                { push(r[$2]);
                                    setTopAndSize();
                                  }
        | POP reg                 { if($2 != ACC && $2 != TOP && $2 != SIZE){
                                        if(!isEmpty()){
                                            r[$2] = pop();
                                            setTopAndSize();
                                        }
                                        else{
                                            printf("Stack is Empty.\n");
                                            yyerror();
                                        }
                                    }
                                    else{
                                        printf("Can't assign number to $acc or $top or $size\n");
                                        yyerror();
                                    }
                                  }
        | SHOW ERR                { printf("SHOW only follow by register");
                                    yyerror();
                                  }
        | SHOW error              { printf("SHOW only follow by register");
                                    yyerror();
                                  }
```

Accept rule

แสดงค่าตัวเลขที่รับเข้ามา

แสดงค่าใน register

แสดงค่าใน register

ตรวจสอบเงื่อนไขต่างๆ (เป็น token TOP หรือ SIZE หรือ ACC หรือไม่) และคัดลอกค่าใน register $4 ไปยัง register $2

Push ค่าใน register $2 ลง stack
เรียก function เพื่อเปลี่ยนค่า top และ size ของ stack

ตรวจสอบเงื่อนไขต่างๆ (เป็น token TOP หรือ SIZE หรือ ACC หรือไม่) stack ว่างหรือไม่ และทำการ pop ค่าออกมาจาก stack

Error handling (ERR มาจาก lexer พบว่ามี input ผิดปกติ)

Error handling เนื่องจาก input token ไม่ตรงกับ rules ใดๆ

```
| LOAD ERR          { printf("ERROR!");
                      yyerror();
                    }
| LOAD error        { printf("");
                      yyerror();
                    }
| PUSH ERR          { printf("Can't PUSH this input");
                      yyerror();
                    }
| PUSH error        { printf("Can't PUSH this input");
                      yyerror();
                    }
| POP ERR           { printf("Can't POP to this input");
                      yyerror();
                    }
| POP error         { printf("Can't POP to this input"
                      yyerror();
                    }
| exp ERR           { printf("ERROR!");
                      yyerror();
                    }
| exp error         { printf("ERROR!");
                      yyerror();
                    }
| ERR               { printf("ERROR!");
                      yyerror();
                    }
| error             { yyerror(); }
```

Error handling (ERR มาจาก lexer พบว่ามี input ผิดปกติ)

Error handling เนื่องจาก input token ไม่ตรงกับ rules ใดๆ

Error handling (ERR มาจาก lexer พบว่ามี input ผิดปกติ)

Error handling เนื่องจาก input token ไม่ตรงกับ rules ใดๆ

Error handling (ERR มาจาก lexer พบว่ามี input ผิดปกติ)

Error handling เนื่องจาก input token ไม่ตรงกับ rules ใดๆ

Error handling (ERR มาจาก lexer พบว่ามี input ผิดปกติ)

Error handling เนื่องจาก input token ไม่ตรงกับ rules ใดๆ

Error handling (ERR มาจาก lexer พบว่ามี input ผิดปกติ)

```
exp : CONSTANT          { $$ = $1;
                          r[ACC] = $1;
                        }
    | exp OR exp        { $$ = $1 | $3;
                          r[ACC] = $1 | $3;
                        }
    | exp AND exp       { $$ = $1 & $3;
                          r[ACC] = $1 & $3;
                        }
    | NOT exp           { $$ = ~$2;
                          r[ACC] = ~$2;
                        }
    | exp '+' exp       { $$ = $1 + $3;
                          r[ACC] = $1 + $3;
                        }
    | exp '-' exp       { $$ = $1 - $3;
                          r[ACC] = $1 - $3;
                        }
    | exp '*' exp       { $$ = $1 * $3;
                          r[ACC] = $1 * $3;
                        }
    | exp '/' exp       { if($3 == 0){
                              printf("Can't divide by 0.");
                              er=1;
                              yyerror();
                          }
                          else{
                              $$ = $1 / $3;
                              r[ACC] = $1 / $3;
                          }
                        }
    | exp '\\' exp      { if($3 == 0){
                              printf("Can't mod by 0.");
                              er=1;
                              yyerror();
                          }
                          else{
                              $$ = $1 % $3;
                              r[ACC] = $1 % $3;
                          }
                        }
    | '-' exp           { $$ = -$2;
                          r[ACC] = -$2;
                        }
    | exp '^' exp       { $$ = pow($1,$3);
                          r[ACC] = pow($1,$3);
                        }
```

exp -> ตัวเลขค่าคงที่
return ตัวเลข $1 และเก็บค่าไว้ใน $acc

return exp OR exp และเก็บค่าไว้ใน $acc

return exp AND exp และเก็บค่าไว้ใน $acc

return NOT exp และเก็บค่าไว้ใน $acc

return exp + exp และเก็บค่าไว้ใน $acc

return exp - exp และเก็บค่าไว้ใน $acc

return exp * exp และเก็บค่าไว้ใน $acc

ตรวจสอบว่าตัวหารเป็น 0 หรือไม่ ถ้าเป็นให้แสดง ข้อความทางหน้าจอ และตั้ง er flag พร้อมเรียกฟังก์ชัน yyerror() ถ้าตัวหารไม่เป็น 0 return ค่า exp หาร exp และเก็บค่าไว้ใน $acc

ตรวจสอบว่าตัวหารเป็น 0 หรือไม่ ถ้าเป็นให้แสดง ข้อความทางหน้าจอ และตั้ง er flag พร้อมเรียกฟังก์ชัน yyerror() ถ้าตัวหารไม่เป็น 0 return ค่า exp หาร เอาเศษ exp และเก็บค่าไว้ใน $acc

return –exp และเก็บค่าไว้ใน $acc

return exp ยกกำลัง exp และเก็บค่าไว้ใน $acc

```
| '(' exp ')'          { $$ = $2;
                         r[ACC] = $2;
                       }
```
return exp ด้านใน ( ) ซึ่งมีความสำคัญมากที่สุด และเก็บค่าไว้ใน $acc

```
rexp: reg              { $$ = r[$1];
                         r[ACC] = r[$1];
                       }
```
return ค่าใน register ตัวที่ $1 และเก็บค่าไว้ใน $acc

```
| reg OR reg           { $$ = r[$1] | r[$3];
                         r[ACC] = r[$1] | r[$3];
                       }
```
return register OR register และเก็บค่าไว้ใน $acc

```
| reg AND reg          { $$ = r[$1] & r[$3];
                         r[ACC] = r[$1] & r[$3];
                       }
```
return register AND register และเก็บค่าไว้ใน $acc

```
| NOT reg              { $$ = ~r[$2];
                         r[ACC] = ~r[$2];
                       }
```
return NOT register และเก็บค่าไว้ใน

```
| reg '+' reg          { $$ = r[$1] + r[$3];
                         r[ACC] = r[$1] + r[$3];
                       }
```
return register + register และเก็บค่าไว้ใน $acc

```
| reg '-' reg          { $$ = r[$1] - r[$3];
                         r[ACC] = r[$1] - r[$3];
                       }
```
return register - register และเก็บค่าไว้ใน $acc

```
| reg '*' reg          { $$ = r[$1] * r[$3];
                         r[ACC] = r[$1] * r[$3];
                       }
```
return register * register และเก็บค่าไว้ใน $acc

```
| reg '/' reg          { f($3 == 0){
                                printf("Can't divide by 0.");
                                er=1;
                                yyerror();
                         }
                         else{
                                $$ = r[$1] / r[$3];
                                r[ACC] = r[$1] / r[$3];
                         }
                       }
```
ตรวจสอบว่าตัวหารเป็น 0 หรือไม่ ถ้าเป็นให้แสดง ข้อความทางหน้าจอ และตั้ง er flag พร้อมเรียกฟังก์ชัน yyerror() ถ้าตัวหารไม่เป็น 0 return ค่า register หาร register และเก็บค่าไว้ใน $acc

```
| reg '\\' reg          { if($3 == 0){
                                printf("Can't mod by 0.");
                                er=1;
                                yyerror();
                         }
                         else{
                                $$ = r[$1] % r[$3];
                                r[ACC] = r[$1] % r[$3];
                         }
                       }
```
ตรวจสอบว่าตัวหารเป็น 0 หรือไม่ ถ้าเป็นให้แสดง ข้อความทางหน้าจอ และตั้ง er flag พร้อมเรียกฟังก์ชัน yyerror() ถ้าตัวหารไม่เป็น 0 return ค่า register หารเอาเศษ register และเก็บค่าไว้ใน $acc

```
| '-' reg                 { $$ = -r[$2];
                            r[ACC] = -r[$2];
                          }

| reg '^' reg             { $$ = pow(r[$1], r[$3]);
                            r[ACC] = pow(r[$1], r[$3]);
                          }

| '(' reg ')'             { $$ = r[$2];
                            r[ACC] = r[$2];
                          }

reg: R0                                   { $$ = R0; }
        |R1                               { $$ = R1; }
        |R2                               { $$ = R2; }
        |R3                               { $$ = R3; }
        |R4                               { $$ = R4; }
        |R5                               { $$ = R5; }
        |R6                               { $$ = R6; }
        |R7                               { $$ = R7; }
        |R8                               { $$ = R8; }
        |R9                               { $$ = R9; }
        |ACC                              { $$ = ACC; }
        |TOP                              { $$ = TOP; }
        |SIZE                             { $$ = SIZE; }
%%

void yyerror() {


}


node* init(int data, node *s) {
        node* temp = (node*)malloc(sizeof(node));
        (*temp).data = data;
        (*temp).next = s;
        return temp;
}
```

return – register และเก็บค่าไว้ใน $acc

return register ยกกำลัง register และเก็บค่าไว้ใน $acc

return register ด้านใน ( ) ซึ่งมีความสำคัญมากที่สุด และเก็บค่าไว้ใน $acc

ตั้งค่า value ของ register เพื่อไปใช้ร่วมกับการทำงานใน stack

ฟังก์ชันสำหรับสร้าง node ใหม่ขึ้นมาและเพิ่มลงใน Stack

```
void push(int data) {
        if(head != NULL) {
                node* temp = init(data, head);
                head = temp;
        }
        else {
                node* temp = init(data, NULL);
                head = temp;
                tail = head;
        }
}
```

ฟังก์ชันสำหรับ push data ลงไปใน stack

```
int pop() {
        if(head != NULL) {
                node *temp = head;
                head = (*head).next;
                return (*temp).data;
        }
        else {
                return -1;
        }
}
```

ฟังก์ชันสำหรับ pop data ออกจาก stack

```
int isEmpty() {
        if(head == NULL) {
                return 1;
        }
        else {
                return 0;
        }
}
```

ฟังก์ชันสำหรับ ตรวจสอบว่า stack ว่างหรือไม่

```
void count() {
        node* temp = head;
        int count = 0;
        while(temp != NULL) {
                count++;
                temp = (*temp).next;
        }
        r[SIZE] = count;
}

void setTopAndSize() {
        count();
        if(head != NULL) {
                r[TOP] = (*head).data;
        }
        else {
                r[TOP] = 0;
        }
}

int main() {
        yyparse();
        return 0;
}
```

ฟังก์ชันสำหรับ ตรวจสอบขนาดของ stack ปัจจุบัน

ฟังก์ชันสำหรับ ตรวจสอบขนาดของ stack ปัจจุบัน และตั้งค่า top of stack

Flex2.l

```
D                       [0-9]
L                       [a-zA-Z_]
H                       [a-fA-F0-9]
B                       [0-1]
E                       [Ee][+-]?{D}+
FS                      (f|F|l|L)
IS                      (u|U|l|L)*

%option noyywrap

%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "Mhee.tab.h"

int fromHexa();
int fromBinary();
void maximumMunch();
%}

%%
```

ประกาศกำหนดเซ็ตของ input

```
"OR"        { return OR; }
"AND"       { return AND; }
"NEG"       { return NEG; }
"NOT"       { return NOT; }
"\n"        { return '\n'; }
"+"         { return '+'; }
"-"         { return '-'; }
"*"         { return '*'; }
"^"         { return '^'; }
"/"         { return '/'; }
"\\"        { return '\\'; }
"("         { return '('; }
")"         { return ')'; }
"$r0"       { return R0; }
"$r1"       { return R1; }
"$r2"       { return R2; }
"$r3"       { return R3; }
"$r4"       { return R4; }
"$r5"       { return R5; }
"$r6"       { return R6; }
"$r7"       { return R7; }
"$r8"       { return R8; }
"$r9"       { return R9; }
"$acc"      { return ACC; }
"$top"      { return TOP; }
"$size"     { return SIZE; }
```

กำหนดการทำงานของ lexer โดยเมื่ออ่าน input ทางด้านซ้ายแล้วจะ return token ทางด้านขวา

```
"$r0"{D}+               { return ERR; }
"$r1"{D}+               { return ERR; }
"$r2"{D}+               { return ERR; }
"$r3"{D}+               { return ERR; }
"$r4"{D}+               { return ERR; }
"$r5"{D}+               { return ERR; }
"$r6"{D}+               { return ERR; }
"$r7"{D}+               { return ERR; }
"$r8"{D}+               { return ERR; }
"$r9"{D}+               { return ERR; }

"LOAD"                  { return LOAD; }
"SHOW"                  { return SHOW; }
"POP"                   { return POP; }
"PUSH"                  { return PUSH; }

">"                     { return '>';}
{B}+"b"                 { yylval = fromBinary(); return(CONSTANT); }
{H}+"h"                 { yylval = fromHexa(); return(CONSTANT); }
{D}+                    { yylval = atoi(yytext); return(CONSTANT); }

[ \t\v\f]               {  }
.                       { /* ignore bad characters */ maximumMunch(); return ERR;}
%%
void maximumMunch(){
        while(input() != '\n');
}
int fromBinary()
{
        int i, j, result = 0;
        for(i = strlen(yytext) - 2;i >= 0;i--) {
                result += (yytext[i] - '0') * pow(2, strlen(yytext) - 2 - i);
        }
        return result;
}
```

กำหนดการทำงานของ lexer โดยเมื่ออ่าน input ทางด้านซ้ายแล้วจะ return token ทางด้านขวา

เมื่อพบ input ตัวเลข 0-1 หรือ 0-F มี b หรือมี h ตามหลัง จะมองเลขชุดนั้นเป็นฐาน 2 และแปลงเป็นฐาน 10 ส่งต่อไปผ่าน yylval

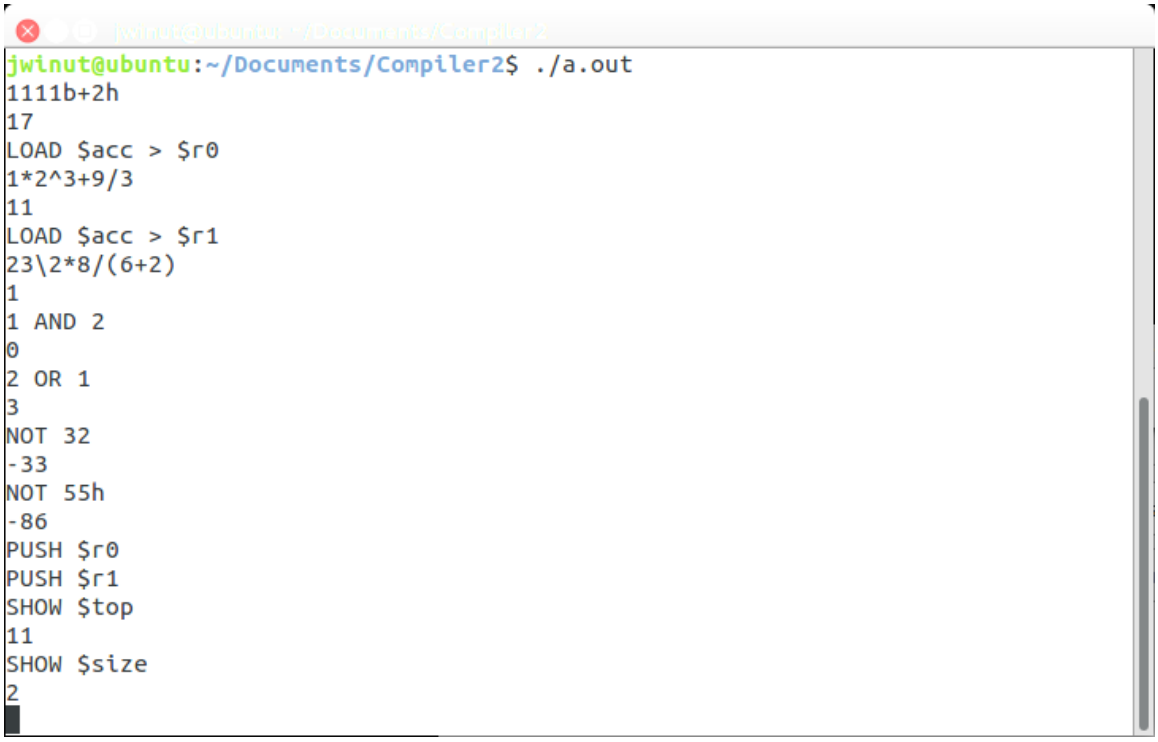
อ่าน input ต่อไปไม่หยุดจนกว่าจะสิ้นสุด input string ที่ลงท้ายด้วย \n (เคาะ Enter)

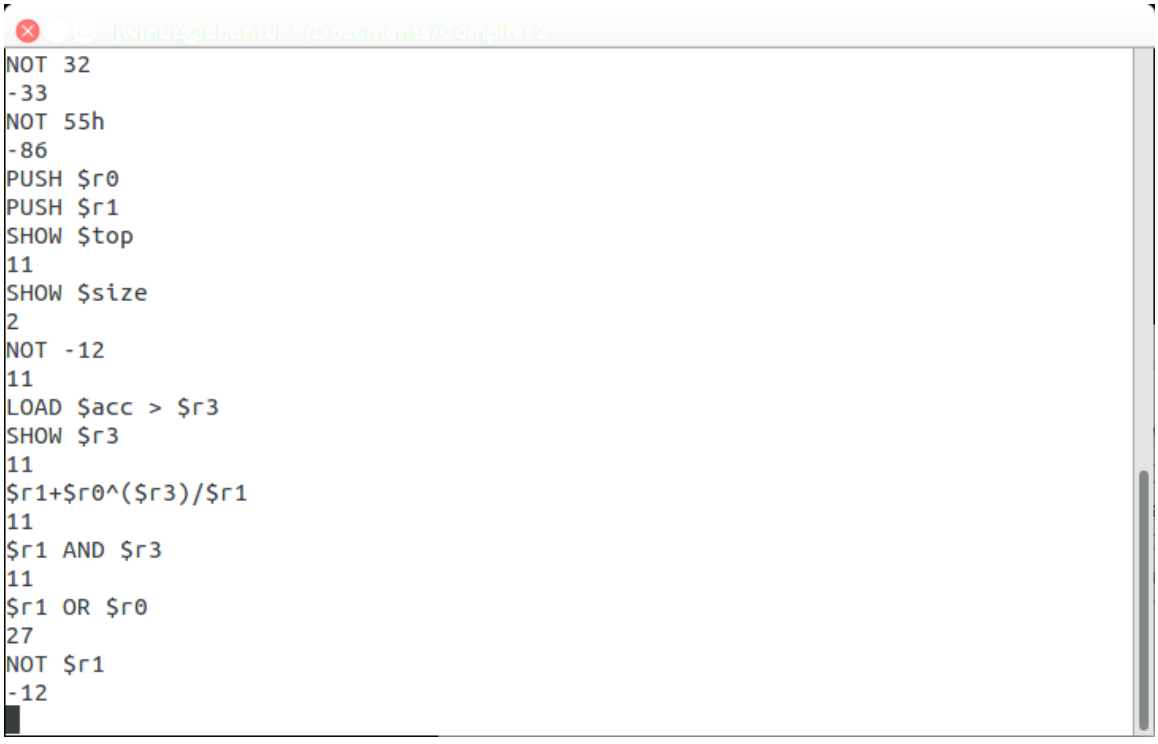
ฟังก์ชันแปลงเลขฐาน 2 เป็น ฐาน 10

ฟังก์ชันแปลงเลขฐาน 16 เป็น ฐาน 10

```
int fromHexa()
{
	int i, j, result = 0;
	for(i = strlen(yytext) - 2;i >= 0;i--) {
		int temp = 0;
		if(yytext[i] == 'A' || yytext[i] == 'a') {
			temp = 10;
		}
		else if(yytext[i] == 'B' || yytext[i] == 'b') {
			temp = 11;
		}
		else if(yytext[i] == 'C' || yytext[i] == 'c') {
			temp = 12;
		}
		else if(yytext[i] == 'D' || yytext[i] == 'd') {
			temp = 13;
		}
		else if(yytext[i] == 'E' || yytext[i] == 'e') {
			temp = 14;
		}
		else if(yytext[i] == 'F' || yytext[i] == 'f') {
			temp = 15;
		}
		else {
			temp = yytext[i] - '0';
		}
		result += temp * pow(16, strlen(yytext) - 2 - i);
	}
	return result;
}
```

ภาพตัวอย่างการทำงาน





```
jwinut@ubuntu:~/Documents/Compiler2$ ./a.out
1/0
Can't divide by 0
78\0
Can't mod by 0
SHOW afs
SHOW only follow by register
LOAD t > h
ERROR!
PUSH yyyd
Can't PUSH this input
POP $r1
Stack is Empty.
POP ff
Can't POP to this input
1 AND sss
ERROR!ERROR!
ss OR 2
ERROR!
NOT sssj
ERROR!
1--
ERROR!
```

```
NOT sssj
ERROR!
1--
ERROR!
10000h----swesd
ERROR!ERROR!
sas4gere23r4xt4365terf
ERROR!
yyyy^s*62
ERROR!
12+3
15
LOAD $acc > $top
Can't assign register to $top or $size or $acc.
LOAD $acc > $size
Can't assign register to $top or $size or $acc.
PUSH $acc
POP $top
Can't assign number to $acc or $top or $size
1 OR ss AND ssss
ERROR!ERROR!
POP $size
Can't assign number to $acc or $top or $size
```