**“EA0015: 컴파일러” 프로젝트**

(1) 컴파일러 (2) 프로젝트 (3) 임현 (4) 201511054

**① 파일 parser.y를 indentation을 잘 하여 출력하고 그 옆에 프로그램을 자세하게 설명하기**

/\*\*

\* @File : parser.y

\* @Author : 임현 (201511054@sangmyung.kr)

\* @Since : 2018 - 05 - 31

\*/

/\* ====================================================================== \*/

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include "symbol.c"

/\* ====================================================================== \*/

#define MAX\_NO\_CODES 1024 /\* maximum number of codes generated \*/

#define MAX\_LINE 256 /\* max code line length \*/

#define NO\_TMP\_REGIS 27 /\* r0 ~ r26 for temporaly registers \*/

#define REGI\_RETURN 27 /\* r27 is for return value \*/

/\* ====================================================================== \*/

extern FILE \*yyin; /\* FILE \* for input file \*/

extern char \*yytext; /\* current lexeme is stored here \*/

extern char \*lex; /\* lexeme of ID and NUM from scanner \*/

extern int source\_line\_no; /\* souce line number \*/

/\* ====================================================================== \*/

void regi\_init(void);

void regi\_print(void);

void regi\_free(int i);

int regi\_new(void);

// int generate(char \*fmt, int i1, int i2, int i3, int i4);

void backpatch(unsigned int ip1, unsigned int ip2);

void backpatch\_funcalls(void);

void print\_code(void);

int yyerror(char \*message);

/\* ====================================================================== \*/

char \*prog\_name; /\* program file name from argv[] \*/

char option\_symbol; /\* -s option means print symbol table \*/

int position; /\* current symbol's kind \*/

int global\_offset; /\* global variable offset \*/

int local\_offset; /\* local variable offset \*/

int \*current\_offset; /\* current offset pointer \*/

int farg\_count; /\* no of formal args in function declaration \*/

int aarg\_count; /\* no of actual args in function call \*/

char \*current\_fun\_name = 0; /\* current function name \*/

unsigned int ip = 0; /\* instruction pointer \*/

char \*code[MAX\_NO\_CODES]; /\* generated codes \*/

char regi\_used[NO\_TMP\_REGIS]; /\* 1 if register is used \*/

/\* ====================================================================== \*/

typedef struct l\_type\_struct { /\* lex attribute for var and num \*/

char \*lex;

} l\_type;

typedef struct t\_type\_struct { /\* type attribute for type\_specifier \*/

unsigned char type;

} t\_type;

typedef struct r\_type\_struct {

unsigned char regi;

} r\_type;

typedef struct p\_type\_struct {

unsigned int ip;

} p\_type;

%}

/\* ====================================================================== \*/

%start program

%union {

l\_type lval;

t\_type tval;

r\_type rval;

p\_type pval;

}

%token VOID INT

%token IF ELSE WHILE RETURN

%token INPUT OUTPUT

%token PLUS MINUS MULTIPLY DIVIDE

%token LT LE GT GE EQ NE

%token ASSIGN

%token SEMICOLON COMMA

%token LPAR RPAR LBRACKET RBRACKET LBRACE RBRACE

%token ID NUM

%token UNDEFINED

%type <lval> var num

%type <tval> type\_specifier

%type <rval> simple\_expression additive\_expression term factor call

%type <pval> if\_rpar if\_else while\_lpar while\_rpar

%%

/\* ====================================================================== \*/

/\* Syntax Rule (문맥 자유 언어의 규칙 {C 코드}로 구성) \*/

program

:

{

struct symbol \*symbolp;

position=GLOBAL;

current\_table=global\_table=create\_table("\_global");

current\_offset=&global\_offset;

\*current\_offset = 0;

generate("%d: ld gp, 0(0)", ip++);

generate("%d: st 0, 0(0)", ip++);

generate("%d: lda fp, -%%d(gp)", ip++);

generate("%d: lda sp, -%%d(gp)", ip++);

generate("%d: push fp", ip++);

generate("%d: lda 0, 2(pc)", ip++);

generate("%d: push 0", ip++);

symbolp = add\_symbol(global\_table, "main", FUNCTIONI, VOID\_TYPE, 0, 0);

generate("%d: ldc pc, %%d", ip++);

symbolp->ip[0] = ip - 1;

symbolp->ipc = 1;

generate("%d: halt", ip++);

}

var\_declaration\_list fun\_declaration\_list

{

backpatch(2, global\_offset);

backpatch(3, global\_offset);

backpatch\_funcalls();

// print\_table(global\_table);

}

;

var\_declaration\_list

: var\_declaration\_list var\_declaration

| empty

;

fun\_declaration\_list

: fun\_declaration\_list fun\_declaration

| fun\_declaration

;

var\_declaration

: type\_specifier var SEMICOLON

{

if ($<tval>1.type == 3)

error("error 00: %s %s \"%s\"", "wrong void", "variable", $<lval>2.lex);

if (find\_symbol(current\_table, $<lval>2.lex) != NULL)

error("error 10: %s %s \"%s\"", "redefined", "variable", $<lval>2.lex);

add\_symbol(current\_table, $<lval>2.lex, position, $<tval>1.type, 1, \*current\_offset);

\*current\_offset = \*current\_offset + 1;

}

| type\_specifier var LBRACKET num RBRACKET SEMICOLON

{

int n = atoi($<lval>4.lex);

if ($<tval>1.type == 3)

error("error 01: %s %s \"%s\"", "wrong void", "array", $<lval>2.lex);

if (find\_symbol(current\_table, $<lval>2.lex) != NULL)

error("error 11: %s %s \"%s\"", "redefined", "array", $<lval>2.lex);

add\_symbol(current\_table, $<lval>2.lex, position, INT\_ARRAY\_TYPE, n, \*current\_offset + n - 1);

\*current\_offset = \*current\_offset + n;

}

;

type\_specifier

: INT

{

$<tval>$.type = INT\_TYPE;

}

| VOID

{

$<tval>$.type = VOID\_TYPE;

}

;

var

: ID

{

$<lval>$.lex = lex;

}

;

num

: NUM

{

$<lval>$.lex = lex;

}

;

fun\_declaration

: type\_specifier var

{

position = ARGUMENT;

current\_table = local\_table = create\_table($<lval>2.lex);

/\* if (find\_symbol(global\_table, $<lval>2.lex) != NULL)

error("error 12: %s %s \"%s\"", "redefined", "function", $<lval>2.lex); \*/

current\_offset = &local\_offset;

\*current\_offset = 0;

farg\_count = 0;

}

LPAR params RPAR

{

struct symbol \*symbolp;

symbolp = find\_symbol(global\_table, $<lval>2.lex);

if (symbolp == NULL) {

add\_symbol(global\_table, $<lval>2.lex, FUNCTION, $<tval>1.type, farg\_count, ip);

} else {

if (symbolp->kind == FUNCTIONI) {

symbolp->kind = FUNCTION;

symbolp->type = $<tval>1.type;

symbolp->offset = ip;

if (symbolp->size != farg\_count)

error("error 40: wrong no argument function \"%s\"", $<lval>2.lex);

} else {

error("error 12: redefined function \"%s\"", $<lval>2.lex);

}

}

position = LOCAL;

}

LBRACE local\_declarations

{

generate("%d: lda sp, -%d(sp)", ip++, \*current\_offset - farg\_count);

}

statement\_list RBRACE

{

generate("%d: ldc %d, 0", ip++, REGI\_RETURN);

generate("%d: lda sp, 0(fp)", ip++);

generate("%d: ld fp, 0(fp)", ip++);

generate("%d: ld pc, -1(sp)", ip++);

// print\_table(current\_table);

// fprintf(stdout, "---------- ---------- ---------- ---------- ---------- ----------\n");

}

;

params

: param\_list

| VOID

;

param\_list

: param\_list COMMA param

{

farg\_count++;

}

| param

{

farg\_count = 1;

}

;

param

: type\_specifier var

{

if ($<tval>1.type == 3)

error("error 02: %s %s \"%s\"", "wrong void", "argument", $<lval>2.lex);

if (find\_symbol(current\_table, $<lval>2.lex) != NULL)

error("error 13: %s %s \"%s\"", "redefined", "argument", $<lval>2.lex);

add\_symbol(current\_table, $<lval>2.lex, position, $<tval>1.type, 1, \*current\_offset);

\*current\_offset = \*current\_offset + 1;

}

| type\_specifier var LBRACKET RBRACKET

{

if ($<tval>1.type == 3)

error("error 03: %s %s \"%s\"", "wrong void", "array argument", $<lval>2.lex);

if (find\_symbol(current\_table, $<lval>2.lex) != NULL)

error("error 14: %s %s \"%s\"", "redefined", "array argument", $<lval>2.lex);

add\_symbol(current\_table, $<lval>2.lex, position, INT\_P\_TYPE, 1, \*current\_offset);

\*current\_offset = \*current\_offset + 1;

}

;

local\_declarations

: local\_declarations var\_declaration

| empty

;

statement\_list

: statement\_list statement

| empty

;

statement

: compound\_stmt

| expression\_stmt

| selection\_stmt

| iteration\_stmt

| funcall\_stmt

| return\_stmt

| input\_stmt

| output\_stmt

;

compound\_stmt

: LBRACE statement\_list RBRACE

;

expression\_stmt

: expression SEMICOLON

{

regi\_free($<rval>1.regi);

}

| SEMICOLON

;

expression

: var ASSIGN expression

{

char \*var = $<lval>1.lex;

struct symbol \*symbolp;

symbolp = lookup\_symbol(var);

if (symbolp == NULL)

error("error 20: %s %s \"%s\"", "undefined", "variable", $<lval>1.lex);

if (symbolp->kind == 3 || symbolp->kind == 4)

error("error 30: %s %s \"%s\"", "type error", "variable", $<lval>1.lex);

if (symbolp->kind == GLOBAL) {

int offset = symbolp->offset;

generate("%d: st %d, -%d(gp)", ip++, $<rval>3.regi, offset);

$<rval>$.regi = $<rval>3.regi;

} else {

int offset = symbolp->offset;

generate("%d: st %d, -%d(fp)", ip++, $<rval>3.regi, offset + 2);

$<rval>$.regi = $<rval>3.regi;

}

}

| var LBRACKET expression RBRACKET ASSIGN expression

{

char \*var = $<lval>1.lex;

struct symbol \*symbolp;

symbolp = lookup\_symbol(var);

if (symbolp == NULL)

error("error 21: %s %s \"%s\"", "undefined", "array", $<lval>1.lex);

if (symbolp->kind == 3 || symbolp->kind == 4)

error("error 31: %s %s \"%s\"", "type error", "array", $<lval>1.lex);

if (symbolp->kind == GLOBAL) {

int regi1 = regi\_new();

int regi2 = regi\_new();

int offset = symbolp->offset;

generate("%d: add %d, gp, %d", ip++, regi1, $<rval>3.regi);

generate("%d: st %d, -%d(%d)", ip++, $<rval>6.regi, offset, regi1);

regi\_free($<rval>3.regi);

regi\_free($<rval>6.regi);

regi\_free(regi1);

regi\_free(regi2);

} else {

int regi1 = regi\_new();

int regi2 = regi\_new();

int offset = symbolp->offset;

if (symbolp->type == INT\_P\_TYPE) {

generate("%d: ld %d, -%d(fp)", ip++, regi1, offset + 2);

generate("%d: add %d, %d, %d", ip++, regi2, regi1, $<rval>3.regi);

generate("%d: st %d, -%d(%d)", ip++, $<rval>6.regi, offset, regi2);

} else {

generate("%d: add %d, fp, %d", ip++, regi1, $<rval>3.regi);

generate("%d: st %d, -%d(%d)", ip++, $<rval>6.regi, offset + 2, regi1);

}

regi\_free($<rval>3.regi);

regi\_free($<rval>6.regi);

regi\_free(regi1);

regi\_free(regi2);

}

}

| simple\_expression

{

$<rval>$.regi = $<rval>1.regi;

}

;

simple\_expression

: additive\_expression LT additive\_expression

{

int regi = regi\_new();

generate("%d: lt %d, %d, %d", ip++, regi, $<rval>1.regi, $<rval>3.regi);

regi\_free($<rval>1.regi);

regi\_free($<rval>3.regi);

$<rval>$.regi = regi;

}

| additive\_expression LE additive\_expression

{

int regi = regi\_new();

generate("%d: le %d, %d, %d", ip++, regi, $<rval>1.regi, $<rval>3.regi);

regi\_free($<rval>1.regi);

regi\_free($<rval>3.regi);

$<rval>$.regi = regi;

}

| additive\_expression GT additive\_expression

{

int regi = regi\_new();

generate("%d: gt %d, %d, %d", ip++, regi, $<rval>1.regi, $<rval>3.regi);

regi\_free($<rval>1.regi);

regi\_free($<rval>3.regi);

$<rval>$.regi = regi;

}

| additive\_expression GE additive\_expression

{

int regi = regi\_new();

generate("%d: ge %d, %d, %d", ip++, regi, $<rval>1.regi, $<rval>3.regi);

regi\_free($<rval>1.regi);

regi\_free($<rval>3.regi);

$<rval>$.regi = regi;

}

| additive\_expression EQ additive\_expression

{

int regi = regi\_new();

generate("%d: eq %d, %d, %d", ip++, regi, $<rval>1.regi, $<rval>3.regi);

regi\_free($<rval>1.regi);

regi\_free($<rval>3.regi);

$<rval>$.regi = regi;

}

| additive\_expression NE additive\_expression

{

int regi = regi\_new();

generate("%d: ne %d, %d, %d", ip++, regi, $<rval>1.regi, $<rval>3.regi);

regi\_free($<rval>1.regi);

regi\_free($<rval>3.regi);

$<rval>$.regi = regi;

}

| additive\_expression

{

$<rval>$.regi = $<rval>1.regi;

}

;

/\* relop

: LT

| LE

| GT

| GE

| EQ

| NE

; \*/

additive\_expression

: additive\_expression PLUS term

{

int regi = regi\_new();

generate("%d: add %d, %d, %d", ip++, regi, $<rval>1.regi, $<rval>3.regi);

regi\_free($<rval>1.regi);

regi\_free($<rval>3.regi);

$<rval>$.regi = regi;

}

| additive\_expression MINUS term

{

int regi = regi\_new();

generate("%d: sub %d, %d, %d", ip++, regi, $<rval>1.regi, $<rval>3.regi);

regi\_free($<rval>1.regi);

regi\_free($<rval>3.regi);

$<rval>$.regi = regi;

}

| term

{

$<rval>$.regi = $<rval>1.regi;

}

;

/\* addop

: PLUS

| MINUS

; \*/

term

: term MULTIPLY factor

{

int regi = regi\_new();

generate("%d: mul %d, %d, %d", ip++, regi, $<rval>1.regi, $<rval>3.regi);

regi\_free($<rval>1.regi);

regi\_free($<rval>3.regi);

$<rval>$.regi = regi;

}

| term DIVIDE factor

{

int regi = regi\_new();

generate("%d: div %d, %d, %d", ip++, regi, $<rval>1.regi, $<rval>3.regi);

regi\_free($<rval>1.regi);

regi\_free($<rval>3.regi);

$<rval>$.regi = regi;

}

| factor

{

$<rval>$.regi = $<rval>1.regi;

}

;

/\* mulop

: MULTIPLY

| DIVIDE

; \*/

factor

: LPAR expression RPAR

{

$<rval>$.regi = $<rval>2.regi;

}

| var

{

char \*var = $<lval>1.lex;

struct symbol \*symbolp;

symbolp = lookup\_symbol(var);

if (symbolp == NULL)

error("error 20: %s %s \"%s\"", "undefined", "variable", $<lval>1.lex);

if (symbolp->kind == GLOBAL) {

int regi = regi\_new();

int offset = symbolp->offset;

if (symbolp->type == INT\_ARRAY\_TYPE)

generate("%d: lda %d, -%d(gp)", ip++, regi, offset);

else

generate("%d: ld %d, -%d(gp)", ip++, regi, offset);

$<rval>$.regi = regi;

} else {

int regi = regi\_new();

int offset = symbolp->offset;

if (symbolp->type == INT\_ARRAY\_TYPE)

generate("%d: lda %d, -%d(fp)", ip++, regi, offset + 2);

else

generate("%d: ld %d, -%d(fp)", ip++, regi, offset + 2);

$<rval>$.regi = regi;

}

}

| var LBRACKET expression RBRACKET

{

char \*var = $<lval>1.lex;

struct symbol \* symbolp;

symbolp = lookup\_symbol(var);

if (symbolp == NULL)

error("error 21: %s %s \"%s\"", "undefined", "array", $<lval>1.lex);

if (symbolp->kind == GLOBAL) {

int regi1 = regi\_new();

int regi2 = regi\_new();

int offset = symbolp->offset;

generate("%d: add %d, gp, %d", ip++, regi1, $<rval>3.regi);

generate("%d: ld %d, -%d(%d)", ip++, regi2, offset, regi1);

regi\_free(regi1);

regi\_free($<rval>3.regi);

$<rval>$.regi = regi2;

} else {

int regi1 = regi\_new();

int regi2 = regi\_new();

int offset = symbolp->offset;

if (symbolp->type == INT\_P\_TYPE) {

generate("%d: ld %d, -%d(fp)", ip++, regi2, offset + 2);

generate("%d: add %d, %d, %d", ip++, regi1, regi2, $<rval>3.regi);

generate("%d: ld %d, -%d(%d)", ip++, regi2, offset, regi1);

} else {

generate("%d: add %d, fp, %d", ip++, regi1, $<rval>3.regi);

generate("%d: ld %d, -%d(%d)", ip++, regi2, offset + 2, regi1);

}

regi\_free(regi1);

regi\_free($<rval>3.regi);

$<rval>$.regi = regi2;

}

}

| num

{

int regi = regi\_new();

int num = atoi(lex);

generate("%d: ldc %d, %d", ip++, regi, num);

$<rval>$.regi = regi;

}

| PLUS num

{

int regi = regi\_new();

int num = atoi(lex);

generate("%d: ldc %d, %d", ip++, regi, num);

$<rval>$.regi = regi;

}

| MINUS num

{

int regi = regi\_new();

int num = atoi(lex);

generate("%d: ldc %d, %d", ip++, regi, num);

$<rval>$.regi = regi;

}

;

selection\_stmt

: IF LPAR expression

{

generate("%d: jle %d, %%d(pc)", ip++, $<rval>3.regi);

regi\_free($<rval>3.regi);

}

if\_rpar statement

{

generate("%d: ldc pc, %%d", ip++);

backpatch($<pval>5.ip - 1, ip - $<pval>5.ip);

}

if\_else statement

{

backpatch($<pval>8.ip - 1, ip);

}

;

if\_rpar

: RPAR

{

$<pval>$.ip = ip;

}

;

if\_else

: ELSE

{

$<pval>$.ip = ip;

}

;

iteration\_stmt

: WHILE while\_lpar expression

{

generate("%d: jle %d, %%d(pc)", ip++, $<rval>3.regi);

regi\_free($<rval>3.regi);

}

while\_rpar statement

{

generate("%d: ldc pc, %d", ip++, $<pval>2.ip);

backpatch($<pval>5.ip - 1, ip - $<pval>5.ip);

}

;

while\_lpar

: LPAR

{

$<pval>$.ip = ip;

}

;

while\_rpar

: RPAR

{

$<pval>$.ip = ip;

}

;

funcall\_stmt

: var ASSIGN call

{

char \*var = $<lval>1.lex;

struct symbol \*symbolp;

if (symbolp->kind == GLOBAL) {

int offset = symbolp->offset;

generate("%d: st %d, -%d(gp)", ip++, $<rval>3.regi, offset);

regi\_free($<rval>3.regi);

} else {

int offset = symbolp->offset;

generate("%d: st %d, -%d(fp)", ip++, $<rval>3.regi, offset + 2);

regi\_free($<rval>3.regi);

}

}

| var LBRACKET expression RBRACKET ASSIGN call

{

char \*var = $<lval>2.lex;

struct symbol \*symbolp;

symbolp = lookup\_symbol(var);

if (symbolp->kind == GLOBAL) {

int regi = regi\_new();

int offset = symbolp->offset;

generate("%d: add %d, gp, %d", ip++, regi, $<rval>3.regi);

generate("%d: st %d, -%d(%d)", ip++, $<rval>3.regi, offset, regi);

regi\_free(regi);

regi\_free($<rval>6.regi);

} else {

int regi = regi\_new();

int offset = symbolp->offset;

generate("%d: add %d, fp, %d", ip++, regi, $<rval>3.regi);

generate("%d: st %d, -%d(%d)", ip++, $<rval>3.regi, offset + 2, regi);

regi\_free(regi);

regi\_free($<rval>6.regi);

}

}

| call

{

regi\_free($<rval>1.regi);

}

;

call

: var

{

struct symbol \*symbolp;

symbolp = lookup\_symbol($<lval>1.lex);

/\* if (symbolp == NULL)

error("error 22: %s %s \"%s\"", "undefined", "function call", $<lval>1.lex);

if (symbolp->kind == 3 || symbolp->kind == 4)

error("error 40: %s %s \"%s\"", "type error", "function", $<lval>1.lex); \*/

aarg\_count = 0;

generate("%d: lda sp, -2(sp)", ip++);

}

LPAR args RPAR

{

int regi;

struct symbol \*symbolp;

symbolp = lookup\_symbol($<lval>1.lex);

if (symbolp->size != aarg\_count)

error("error 32: %s %s \"%s\"", "wrong no argument", "function", $<lval>1.lex);

generate("%d: st fp, -%d(fp)", ip++, \*current\_offset + 2);

generate("%d: lda fp, -%d(fp)", ip++, \*current\_offset + 2);

regi = regi\_new();

generate("%d: lda %d, 2(pc)", ip++, regi);

generate("%d: st %d, -1(fp)", ip++, regi);

regi\_free(regi);

if (symbolp->offset != 0) {

generate("%d: ldc pc, %d", ip++, symbolp->offset);

} else {

generate("%d: halt", ip++);

}

regi = regi\_new();

generate("%d: lda %d, 0(%d)", ip++, regi, REGI\_RETURN);

$<rval>$.regi = regi;

}

;

args

: arg\_list

| empty

;

arg\_list

: arg\_list COMMA expression

{

aarg\_count++;

generate("%d: push %d", ip++, $<rval>3.regi);

regi\_free($<rval>3.regi);

}

| expression

{

aarg\_count++;

generate("%d: push %d", ip++, $<rval>1.regi);

regi\_free($<rval>1.regi);

}

;

return\_stmt

: RETURN SEMICOLON

{

generate("%d: ldc %d, 0", ip++, REGI\_RETURN);

generate("%d: lda sp, 0(fp)", ip++);

generate("%d: ld fp, 0(fp)", ip++);

generate("%d: ld pc, -1(sp)", ip++);

}

| RETURN expression SEMICOLON

{

generate("%d: lda %d, 0(%d)", ip++, REGI\_RETURN, $<rval>2.regi);

generate("%d: lda sp, 0(fp)", ip++);

generate("%d: ld fp, 0(fp)", ip++);

generate("%d: ld pc, -1(sp)", ip++);

regi\_free($<rval>2.regi);

}

;

input\_stmt

: INPUT var SEMICOLON

{

char \*var = $<lval>2.lex;

struct symbol \*symbolp;

symbolp = lookup\_symbol(var);

if (symbolp->kind == GLOBAL) {

int regi = regi\_new();

int offset = symbolp->offset;

generate("%d: in %d", ip++, regi);

generate("%d: st %d, -%d(gp)", ip++, regi, offset);

regi\_free(regi);

} else {

int regi = regi\_new();

int offset = symbolp->offset;

generate("%d: in %d", ip++, regi);

generate("%d: st %d, -%d(fp)", ip++, regi, offset + 2);

regi\_free(regi);

}

}

| INPUT var LBRACKET expression RBRACKET SEMICOLON

{

char \*var = $<lval>2.lex;

struct symbol \*symbolp;

symbolp = lookup\_symbol(var);

if (symbolp->kind == GLOBAL) {

int regi1 = regi\_new();

int regi2 = regi\_new();

int offset = symbolp->offset;

generate("%d: add %d, gp, %d", ip++, regi1, $<rval>4.regi);

generate("%d: in %d", ip++, regi2);

generate("%d: st %d, -%d(%d)", ip++, regi2, offset, regi1);

regi\_free(regi1);

regi\_free(regi2);

} else {

int regi1 = regi\_new();

int regi2 = regi\_new();

int offset = symbolp->offset;

generate("%d: add %d, fp, %d", ip++, regi1, $<rval>4.regi);

generate("%d: in %d", ip++, regi2);

generate("%d: st %d, -%d(%d)", ip++, regi2, offset + 2, regi1);

regi\_free(regi1);

regi\_free(regi2);

}

}

;

output\_stmt

: OUTPUT expression SEMICOLON

{

generate("%d: out %d", ip++, $<rval>2.regi);

regi\_free($<rval>2.regi);

}

;

empty

:

;

%%

/\* ====================================================================== \*/

void regi\_init(void)

{

int i;

for (i = 0; i < NO\_TMP\_REGIS; i++)

regi\_used[i] = 0;

}

/\* ====================================================================== \*/

void regi\_free(int i)

{

regi\_used[i] = 0;

}

/\* ====================================================================== \*/

int regi\_new(void)

{

int i;

for (i = 0; i < NO\_TMP\_REGIS; i++) {

if (regi\_used[i] == 0) {

regi\_used[i] = 1;

return i;

}

}

error("error 50: all registers are used!");

}

/\* ====================================================================== \*/

int generate(char \*fmt, int i1, int i2, int i3, int i4)

{

char tmp[MAX\_LINE];

char \*p;

sprintf(tmp, fmt, i1, i2, i3, i4);

p = (char \*) malloc(strlen(tmp) + 1);

strcpy(p, tmp);

code[ip - 1] = p;

}

/\* ====================================================================== \*/

void backpatch(unsigned int ip1, unsigned int ip2)

{

char tmp[MAX\_LINE];

char \*p;

sprintf(tmp, code[ip1], ip2);

p = (char \*) malloc(strlen(tmp) + 1);

strcpy(p, tmp);

free(code[ip1]);

code[ip1] = p;

}

/\* ====================================================================== \*/

void backpatch\_funcalls(void)

{

int i, j;

struct symbol \*symbolp;

for (i = 0; i < HASH\_SIZE; i++)

for (symbolp = global\_table->hash[i]; symbolp != NULL;

symbolp = symbolp->next)

for (j = 0; j < symbolp->ipc; j++)

backpatch(symbolp->ip[j], symbolp->offset);

}

/\* ====================================================================== \*/

void print\_code(void)

{

int i;

char file[MAX\_LINE];

FILE \*fp;

i = strlen(prog\_name);

if ((i > 2) && (prog\_name[i - 2] == '.') && (prog\_name[i - 1] == 'c'))

prog\_name[i - 2] = '\0';

else if ((i > 2) && (prog\_name[i - 2] == '.') && (prog\_name[i - 1] == 'C'))

prog\_name[i - 2] = '\0';

sprintf(file, "%s.tm", prog\_name);

if ((fp = fopen(file, "w")) == NULL) {

fprintf(stderr, "%s: %s\n", file, strerror(errno));

exit(1);

}

fprintf(fp, "// ====================\n");

fprintf(fp, "// c startup\n");

fprintf(fp, "// ====================\n");

for (i = 0; i < ip; i++)

fprintf(fp, "%s\n", code[i]);

fprintf(fp, "// ====================\n");

fclose(fp);

}

/\* ====================================================================== \*/

int yyerror(char \*message)

{

if (option\_symbol == 1) {

print\_table(current\_table);

fprintf(stdout,

"---------- ---------- ---------- ---------- ---------- ----------\n");

print\_table(global\_table);

fprintf(stdout,

"---------- ---------- ---------- ---------- ---------- ----------\n");

}

fprintf(stderr, "line %d: %s at \"%s\"\n", source\_line\_no, message,

yytext);

}

/\* ====================================================================== \*/

int error(char \*fmt, char \*s1, char \*s2, char \*s3, char \*s4)

{

if (option\_symbol == 1) {

print\_table(current\_table);

fprintf(stdout,

"---------- ---------- ---------- ---------- ---------- ----------\n");

print\_table(global\_table);

fprintf(stdout,

"---------- ---------- ---------- ---------- ---------- ----------\n");

}

fprintf(stdout, "line %d: ", source\_line\_no);

fprintf(stdout, fmt, s1, s2, s3, s4);

fprintf(stdout, "\n");

fflush(stdout);

exit(-1);

}

/\* ====================================================================== \*/

int main(int argc, char \*argv[])

{

if (argc == 2) {

option\_symbol = 0;

prog\_name = argv[1];

} else if (argc == 3) {

if (strcmp(argv[1], "-s") != 0) {

fprintf(stderr, "usage: cm [-s] file\n");

exit(1);

}

option\_symbol = 1;

prog\_name = argv[2];

} else {

fprintf(stderr, "usage: cm [-s] file\n");

exit(1);

}

yyin = fopen(prog\_name, "r");

if (yyin == NULL) {

fprintf(stderr, "%s: %s\n", prog\_name, strerror(errno));

exit(1);

}

yyparse();

print\_code();

return 0;

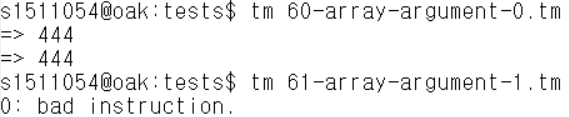
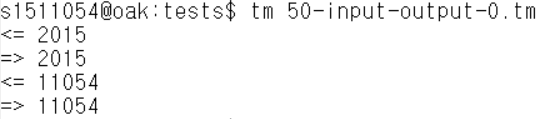
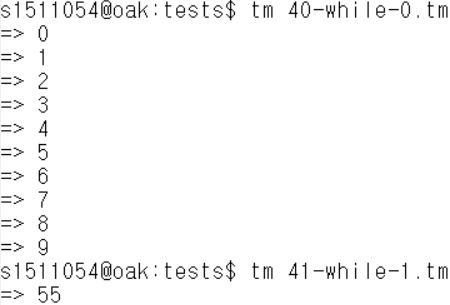
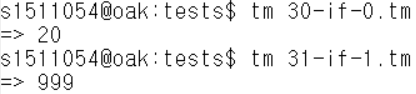
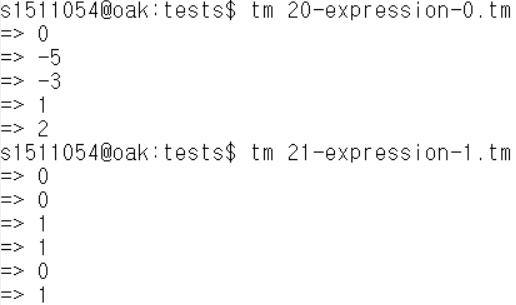
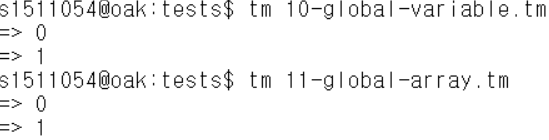
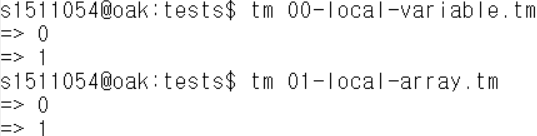
}

/\* ====================================================================== \*/

**② 프로그래밍 서버 oak.smu.ac.kr의 /home/compiler/project/tests 아래 18개 test 프로그램 중 바르게 컴파일된 후 TM이 바르게 수행하는 프로그램의 개수 적기**

11개

**③ 각 test 프로그램에 대하여 C-Minus 프로그램 출력, compile한 TM 파일 출력 및 TM 파일 수행 결과 출력을 제출하기**



00-local-variable.tm

// ====================

// c startup

// ====================

0: ld gp, 0(0)

1: st 0, 0(0)

2: lda fp, -0(gp)

3: lda sp, -0(gp)

4: push fp

5: lda 0, 2(pc)

6: push 0

7: ldc pc, 9

8: halt

9: lda sp, -2(sp)

10: ldc 0, 0

11: st 0, -2(fp)

12: ldc 0, 1

13: st 0, -3(fp)

14: ld 0, -2(fp)

15: out 0

16: ld 0, -3(fp)

17: out 0

18: ldc 27, 0

19: lda sp, 0(fp)

20: ld fp, 0(fp)

21: ld pc, -1(sp)

// ====================

01-local-array.tm

// ====================

// c startup

// ====================

0: ld gp, 0(0)

1: st 0, 0(0)

2: lda fp, -0(gp)

3: lda sp, -0(gp)

4: push fp

5: lda 0, 2(pc)

6: push 0

7: ldc pc, 9

9: lda sp, -11(sp)

10: ldc 0, 0

11: st 0, -2(fp)

12: ldc 0, 4

13: ldc 1, 1

14: add 2, fp, 0

15: st 1, -12(2)

16: ld 0, -2(fp)

17: out 0

18: ldc 0, 4

19: add 1, fp, 0

20: ld 2, -12(1)

21: out 2

22: ldc 27, 0

23: lda sp, 0(fp)

24: ld fp, 0(fp)

25: ld pc, -1(sp)

// ====================

10-global-variable.tm

// ====================

// c startup

// ====================

0: ld gp, 0(0)

1: st 0, 0(0)

2: lda fp, -2(gp)

3: lda sp, -2(gp)

4: push fp

5: lda 0, 2(pc)

6: push 0

7: ldc pc, 9

8: halt

9: lda sp, -0(sp)

10: ldc 0, 0

11: st 0, -0(gp)

12: ldc 0, 1

13: st 0, -1(gp)

14: ld 0, -0(gp)

15: out 0

16: ld 0, -1(gp)

17: out 0

18: ldc 27, 0

19: lda sp, 0(fp)

20: ld fp, 0(fp)

21: ld pc, -1(sp)

// ====================

11-global-array.tm

// ====================

// c startup

// ====================

0: ld gp, 0(0)

1: st 0, 0(0)

2: lda fp, -11(gp)

3: lda sp, -11(gp)

4: push fp

5: lda 0, 2(pc)

6: push 0

7: ldc pc, 9

9: lda sp, -0(sp)

10: ldc 0, 0

11: st 0, -0(gp)

12: ldc 0, 4

13: ldc 1, 1

14: add 2, gp, 0

15: st 1, -10(2)

16: ld 0, -0(gp)

17: out 0

18: ldc 0, 4

19: add 1, gp, 0

20: ld 2, -10(1)

21: out 2

22: ldc 27, 0

23: lda sp, 0(fp)

24: ld fp, 0(fp)

25: ld pc, -1(sp)

// ====================

20-expression-0.tm

// ====================

// c startup

// ====================

0: ld gp, 0(0)

1: st 0, 0(0)

2: lda fp, -1(gp)

3: lda sp, -1(gp)

4: push fp

5: lda 0, 2(pc)

6: push 0

7: ldc pc, 9

8: halt

9: lda sp, -0(sp)

10: ldc 0, 1

11: ldc 1, 2

12: add 2, 0, 1

13: ldc 0, 3

14: sub 1, 2, 0

15: st 1, -0(gp)

16: ld 0, -0(gp)

17: out 0

18: ldc 0, 1

19: ldc 1, 2

20: ldc 2, 3

21: mul 3, 1, 2

22: sub 1, 0, 3

23: st 1, -0(gp)

24: ld 0, -0(gp)

25: out 0

26: ldc 0, 1

27: ldc 1, 2

28: sub 2, 0, 1

29: ldc 0, 3

30: mul 1, 2, 0

31: st 1, -0(gp)

32: ld 0, -0(gp)

33: out 0

34: ldc 0, 1

35: ldc 1, 2

36: ldc 2, 3

37: div 3, 1, 2

38: add 1, 0, 3

39: st 1, -0(gp)

40: ld 0, -0(gp)

41: out 0

42: ldc 0, 4

43: ldc 1, 2

44: add 2, 0, 1

45: ldc 0, 3

46: div 1, 2, 0

47: st 1, -0(gp)

48: ld 0, -0(gp)

49: out 0

50: ldc 27, 0

51: lda sp, 0(fp)

52: ld fp, 0(fp)

53: ld pc, -1(sp)

// ====================

21-expression-1.tm

// ====================

// c startup

// ====================

0: ld gp, 0(0)

1: st 0, 0(0)

2: lda fp, -1(gp)

3: lda sp, -1(gp)

4: push fp

5: lda 0, 2(pc)

6: push 0

7: ldc pc, 9

8: halt

9: lda sp, -0(sp)

10: ldc 0, 1

11: ldc 1, 2

12: gt 2, 0, 1

13: st 2, -0(gp)

14: ld 0, -0(gp)

15: out 0

16: ldc 0, 1

17: ldc 1, 2

18: ge 2, 0, 1

19: st 2, -0(gp)

20: ld 0, -0(gp)

21: out 0

22: ldc 0, 1

23: ldc 1, 2

24: lt 2, 0, 1

25: st 2, -0(gp)

26: ld 0, -0(gp)

27: out 0

28: ldc 0, 2

29: ldc 1, 2

30: le 2, 0, 1

31: st 2, -0(gp)

32: ld 0, -0(gp)

33: out 0

34: ldc 0, 1

35: ldc 1, 2

36: eq 2, 0, 1

37: st 2, -0(gp)

38: ld 0, -0(gp)

39: out 0

40: ldc 0, 1

41: ldc 1, 2

42: ne 2, 0, 1

43: st 2, -0(gp)

44: ld 0, -0(gp)

45: out 0

46: ldc 27, 0

47: lda sp, 0(fp)

48: ld fp, 0(fp)

49: ld pc, -1(sp)

// ====================

30-if-0.tm

// ====================

// c startup

// ====================

0: ld gp, 0(0)

1: st 0, 0(0)

2: lda fp, -0(gp)

3: lda sp, -0(gp)

4: push fp

5: lda 0, 2(pc)

6: push 0

7: ldc pc, 9

8: halt

9: lda sp, -2(sp)

10: ldc 0, 10

11: st 0, -2(fp)

12: ldc 0, 20

13: st 0, -3(fp)

14: ld 0, -2(fp)

15: ld 1, -3(fp)

16: lt 2, 0, 1

17: jle 2, 3(pc)

18: ld 0, -3(fp)

19: out 0

20: ldc pc, 23

21: ld 0, -2(fp)

22: out 0

23: ldc 27, 0

24: lda sp, 0(fp)

25: ld fp, 0(fp)

26: ld pc, -1(sp)

// ====================

31-if-1.tm

// ====================

// c startup

// ====================

0: ld gp, 0(0)

1: st 0, 0(0)

2: lda fp, -0(gp)

3: lda sp, -0(gp)

4: push fp

5: lda 0, 2(pc)

6: push 0

7: ldc pc, 9

8: halt

9: lda sp, -3(sp)

10: ldc 0, 10

11: st 0, -2(fp)

12: ldc 0, 999

13: st 0, -3(fp)

14: ldc 0, 30

15: st 0, -4(fp)

16: ld 0, -2(fp)

17: ld 1, -3(fp)

18: lt 2, 0, 1

19: jle 2, 10(pc)

20: ld 0, -3(fp)

21: ld 1, -4(fp)

22: lt 2, 0, 1

23: jle 2, 3(pc)

24: ld 0, -4(fp)

25: out 0

26: ldc pc, 29

27: ld 0, -3(fp)

28: out 0

29: ldc pc, 39

30: ld 0, -2(fp)

31: ld 1, -4(fp)

32: lt 2, 0, 1

33: jle 2, 3(pc)

34: ld 0, -4(fp)

35: out 0

36: ldc pc, 39

37: ld 0, -2(fp)

38: out 0

39: ldc 27, 0

40: lda sp, 0(fp)

41: ld fp, 0(fp)

42: ld pc, -1(sp)

// ====================

40-while-0.tm

// ====================

// c startup

// ====================

0: ld gp, 0(0)

1: st 0, 0(0)

2: lda fp, -0(gp)

3: lda sp, -0(gp)

4: push fp

5: lda 0, 2(pc)

6: push 0

7: ldc pc, 9

8: halt

9: lda sp, -1(sp)

10: ldc 0, 0

11: st 0, -2(fp)

12: ld 0, -2(fp)

13: ldc 1, 10

14: lt 2, 0, 1

15: jle 2, 7(pc)

16: ld 0, -2(fp)

17: out 0

18: ld 0, -2(fp)

19: ldc 1, 1

20: add 2, 0, 1

21: st 2, -2(fp)

22: ldc pc, 12

23: ldc 27, 0

24: lda sp, 0(fp)

25: ld fp, 0(fp)

26: ld pc, -1(sp)

// ====================

41-while-1.tm

// ====================

// c startup

// ====================

0: ld gp, 0(0)

1: st 0, 0(0)

2: lda fp, -0(gp)

3: lda sp, -0(gp)

4: push fp

5: lda 0, 2(pc)

6: push 0

7: ldc pc, 9

8: halt

9: lda sp, -2(sp)

10: ldc 0, 1

11: st 0, -2(fp)

12: ldc 0, 0

13: st 0, -3(fp)

14: ld 0, -2(fp)

15: ldc 1, 10

16: le 2, 0, 1

17: jle 2, 9(pc)

18: ld 0, -3(fp)

19: ld 1, -2(fp)

20: add 2, 0, 1

21: st 2, -3(fp)

22: ld 0, -2(fp)

23: ldc 1, 1

24: add 2, 0, 1

25: st 2, -2(fp)

26: ldc pc, 14

27: ld 0, -3(fp)

28: out 0

29: ldc 27, 0

30: lda sp, 0(fp)

31: ld fp, 0(fp)

32: ld pc, -1(sp)

// ====================

50-input-output-0.tm

// ====================

// c startup

// ====================

0: ld gp, 0(0)

1: st 0, 0(0)

2: lda fp, -0(gp)

3: lda sp, -0(gp)

4: push fp

5: lda 0, 2(pc)

6: push 0

7: ldc pc, 9

8: halt

9: lda sp, -11(sp)

10: in 0

11: st 0, -2(fp)

12: ld 0, -2(fp)

13: out 0

14: ldc 0, 8

15: add 1, fp, 0

16: in 2

17: st 2, -12(1)

18: ldc 1, 8

19: add 2, fp, 1

20: ld 3, -12(2)

21: out 3

22: ldc 27, 0

23: lda sp, 0(fp)

24: ld fp, 0(fp)

25: ld pc, -1(sp)

// ====================

60-array-argument-0.tm

// ====================

// c startup

// ====================

0: ld gp, 0(0)

1: st 0, 0(0)

2: lda fp, -10(gp)

3: lda sp, -10(gp)

4: push fp

5: lda 0, 2(pc)

6: push 0

7: ldc pc, 19

8: halt

9: lda sp, -0(sp)

10: ldc 0, 4

11: ld 2, -2(fp)

12: add 1, 2, 0

13: ld 2, -0(1)

14: out 2

15: ldc 27, 0

16: lda sp, 0(fp)

17: ld fp, 0(fp)

18: ld pc, -1(sp)

19: lda sp, -0(sp)

20: ldc 0, 4

21: ldc 1, 444

22: add 2, gp, 0

23: st 1, -9(2)

24: ldc 0, 4

25: add 1, gp, 0

26: ld 2, -9(1)

27: out 2

28: lda sp, -2(sp)

29: lda 0, -9(gp)

30: push 0

31: st fp, -2(fp)

32: lda fp, -2(fp)

33: lda 0, 2(pc)

34: st 0, -1(fp)

35: ldc pc, 9

36: lda 0, 0(27)

37: ldc 27, 0

38: lda sp, 0(fp)

39: ld fp, 0(fp)

40: ld pc, -1(sp)

// ====================

61-array-argument-1.tm

// ====================

// c startup

// ====================

1: st 0, 0(0)

2: lda fp, -10(gp)

3: lda sp, -10(gp)

4: push fp

5: lda 0, 2(pc)

6: push 0

7: ldc pc, 19

8: halt

9: lda sp, -0(sp)

10: ldc 0, 5

11: ldc 1, 555

12: ld 2, -2(fp)

13: add 3, 2, 0

14: st 1, -0(3)

15: ldc 27, 0

16: lda sp, 0(fp)

17: ld fp, 0(fp)

18: ld pc, -1(sp)

19: lda sp, -0(sp)

20: ldc 0, 4

21: ldc 1, 444

22: add 2, gp, 0

23: st 1, -9(2)

24: ldc 0, 4

25: add 1, gp, 0

26: ld 2, -9(1)

27: out 2

28: ldc 0, 5

29: add 1, gp, 0

30: ld 2, -9(1)

31: out 2

32: lda sp, -2(sp)

33: lda 0, -9(gp)

34: push 0

35: st fp, -2(fp)

36: lda fp, -2(fp)

37: lda 0, 2(pc)

38: st 0, -1(fp)

39: ldc pc, 9

40: lda 0, 0(27)

41: ldc 0, 4

42: add 1, gp, 0

43: ld 2, -9(1)

44: out 2

45: ldc 0, 5

46: add 1, gp, 0

47: ld 2, -9(1)

48: out 2

49: ldc 27, 0

50: lda sp, 0(fp)

51: ld fp, 0(fp)

52: ld pc, -1(sp)

// ====================

70-add.tm

// ====================

// c startup

// ====================

0: ld gp, 0(0)

1: st 0, 0(0)

2: lda fp, -1(gp)

3: lda sp, -1(gp)

4: push fp

5: lda 0, 2(pc)

6: push 0

7: ldc pc, 18

8: halt

9: lda sp, -0(sp)

10: ld 0, -2(fp)

11: ld 1, -3(fp)

12: add 2, 0, 1

13: st 2, -0(gp)

14: ldc 27, 0

15: lda sp, 0(fp)

16: ld fp, 0(fp)

17: ld pc, -1(sp)

18: lda sp, -0(sp)

19: lda sp, -2(sp)

20: ldc 0, 111

21: push 0

22: ldc 0, 222

23: push 0

24: st fp, -2(fp)

25: lda fp, -2(fp)

26: lda 0, 2(pc)

27: st 0, -1(fp)

28: ldc pc, 9

29: lda 0, 0(27)

30: ld 0, -0(gp)

31: out 0

32: ldc 27, 0

33: lda sp, 0(fp)

34: ld fp, 0(fp)

35: ld pc, -1(sp)

// ====================