

컴파일러 과제-7

20192800 권대현

1. 과제 내용

- 이번 컴파일러 과제는 과제-6에서 구현한 C언어 시멘틱 분석기에 더불어 9장에서 설명한 코드 생성기 프로그램을 완성하여 실험하는 것이다.
- 입력으로 다양한 선언문과 명령문을 포함하는 프로그램들이 주어진다.
- 수식이 잘못된 경우 line 번호와 함께 syntax 오류가 어디서 일어났는지를 출력한다.
- 수식이 올바른 경우 syntax tree를 출력하여 신택스 분석이 된 과정을 보여준다. 그 다음으로 semantic tree를 출력하여 시멘틱 분석이 된 과정을 보여준다.
- 수식이 잘못된 경우 에러 번호를 출력하여 어떤 에러가 발생했는지 알려준다.
- 수식이 올바른 경우 어셈블리어로 주어진 코드를 번역하고 a.asm 파일을 생성하고 코드를 저장한다.
- 생성된 a.asm 파일을 주어진 어셈블러 인터프리터인 interp.exe에 넘겨준다.
- 수식이 잘못된 경우 line 번호와 함께 syntax 오류가 어디서 일어났는지를 출력한다.
- 수식이 올바른 경우 추출된 symbol과 번호를 출력하고, 코드를 번호 순으로 출력하여 최종적으로 코드를 계산한 결과값을 출력한다.

2. 문제 및 해결 방법

- gen.c 구현 문제
 - 이번 과제에서 추가로 구현해야 할 사항은 분석이 끝난 소스 코드를 어셈블리어로 번역하는 함수들을 만드는 것이다. 이에 따라 컴파일러 pdf 9장, 10-11장과 9장 화상 강의를 참고하여 gen.c를 생성하였고, 내부 함수를 구현하는 도중에 몇몇 문제가 발생했다.
 - gen_literal_table() 구현 문제
 - ◆ 처음 해당 함수를 만들 때 i=1부터 semantic.c에 선언된 literal_no까지 for 문으로 반복하여 .literal 주소와 상수를 저장하도록 코드를 작성했다. 그러나, 상수를 저장할 때 쓰이는 value값이 union 타입인 줄 모르고, value값만 사용했더니 fprintf로 파일에 저장하는 함수에 오류가 생겼다.
 - ◆ 이에 literal_table[] 배열을 참조하여 해당 멤버의 type값에 따라 value.i, value.f 등 다르게 저장할 수 있도록 타입에 따른 파일 저장을 구현하여 해결했다.
 - Typedef enum op OP CODE 사용 문제
 - ◆ 새로 받은 type.h에 어셈블리어 명령어를 포함하는 enum형 OP CODE가 선언되어 있다. 이를 참조하기 위해 extern enum op OP CODE나 typedef enum op OP CODE 등 여러 시도를 했지만 제대로 참조되지 못하는 문제가 발생했다.

- ◆ 답은 간단했다. 새로 interp/type.h로 경로를 재설정하여 헤더파일을 참조하기만 하면 됐다. 오히려 따로 extern 키워드를 통해 불러올 필요도 없었다.

- gen_declaration_list() 구현 문제

- ◆ pdf 상으로 gen_declaration() 함수는 예시 코드가 있지만, gen_declaration_list()는 존재하지 않아서 구현에 애를 먹게 됐다.
- ◆ gen_declaration_list라는 함수 이름에서 알 수 있듯이, gen_declaration()을 list로 받아서 한꺼번에 실행해주는 함수라는 것을 알고, while문으로 인자로 넘겨 받은 id를 넣어서 계속해서 다음 link로 gen_declaration()을 수행할 수 있게끔 코드를 작성하여 해결했다.

3. 테스트

- 올바른 C언어 코드

- Hello World 출력

```

~/compiler/07
start code generation
neosk@neoskyclad-GRAM ~/compiler/07
$ ./interp/interp.exe a.asm
===== symbol =====
1: printf -1
2: malloc -2
3: scanf -3
4: main 3
===== code =====
0: INT 0,28
1: SUP 0,3
2: RET 0,0
3: INT 0,12
4: INT 0,12
5: LDA 0,12
6: POP 0,4
7: ADDR 0,-1
8: CAL 0,0
9: LDA 1,-4
10: LITI 0,0
11: STO 0,1
12: RET 0,0
13: RET 0,0
start execution
Hello World
end execution
neosk@neoskyclad-GRAM ~/compiler/07
$

```

- ◆ Main 함수에서 printf("Hello World!\n")를 수행했다.
- ◆ a.exe를 통해 Hello World를 출력하는 C코드를 어셈블리어로 번역하고 이를 interp.exe에 넣음으로써 어셈블리/인터프리터를 실행한다.

■ int *fun() 함수

```

~/compiler/07
neosk@neoskyclad-GRAM ~/compiler/07
$ ./interp/interp.exe a.asm
===== symbol =====
1: printf -1
2: malloc -2
3: scanf -3
4: main 15
5: fun 3
===== code =====
0: INT 0,12
1: SUP 0,15
2: RET 0,0
3: INT 0,20
4: LDA 1,16
5: LOD 1,12
6: LOD 1,12
7: MULI 0,0
8: STX 0,1
9: POP 0,1
10: LDA 1,-4
11: LOD 1,16
12: STO 0,1
13: RET 0,0
14: RET 0,0
15: INT 0,16
16: LDA 1,-4
17: LITI 0,0
18: STO 0,1
19: RET 0,0
20: RET 0,0
start execution
end execution
neosk@neoskyclad-GRAM ~/compiler/07

```

◆ 파라미터로 int a를 넘겨받고 지역 변수로 int x를 선언하는 int *fun() 함수를 선언한다.

■ Enum 선언 테스트

```

~/compiler/07
start syntax analysis
start semantic analysis
start code generation
neosk@neoskyclad-GRAM ~/compiler/07
$ ./interp/interp.exe a.asm
===== symbol =====
1: printf -1
2: malloc -2
3: scanf -3
4: main 3
===== code =====
0: INT 0,48
1: SUP 0,3
2: RET 0,0
3: INT 0,12
4: INT 0,12
5: LDA 0,24
6: LITI 0,10
7: POP 0,5
8: ADDR 0,-1
9: CAL 0,0
10: LDA 1,-4
11: LITI 0,0
12: STO 0,1
13: RET 0,0
14: RET 0,0
start execution
enum color red = 10
end execution
neosk@neoskyclad-GRAM ~/compiler/07
$ |

```

- ◆ Enum 형으로 color을 정의하고, color 내부 멤버를 초기화하는 코드를 선언했다.
- ◆ Enum color c1, c2; 를 통해 사전 정의된 color 형을 사용하는 변수 선언을 테스트했다.
- ◆ 마지막으로 printf로 enum red 값이 정수로 얼마인지를 출력했다.
- Printf, scanf 테스트

```

~/compiler/07
neosk@neoskyclad-GRAM ~/compiler/07
$ ./interp/interp.exe a.asm
===== symbol =====
1: printf -1
2: malloc -2
3: scanf -3
4: main 3
===== code =====
0: INT 0,28
1: SUP 0,3
2: RET 0,0
3: INT 0,16
4: INT 0,12
5: LDA 0,12
6: LDA 1,12
7: POP 0,5
8: ADDR 0,-3
9: CAL 0,0
10: INT 0,12
11: LDA 0,20
12: LOD 1,12
13: POP 0,5
14: ADDR 0,-1
15: CAL 0,0
16: LDA 1,-4
17: LITI 0,0
18: STO 0,1
19: RET 0,0
20: RET 0,0
start execution
1
1
end execution

```

- ◆ 간단하게 scanf를 통해 사용자 입력을 받고, 이를 변수 i에 저장했다가 출력하는 코드를 테스트해봤다.
- ◆ 정상적으로 1을 입력받고 1을 다시 출력했다.
- 함수 선언 및 곱셈 테스트

```
~/compiler/07
neosk@neoskyclad-GRAM ~/compiler/07
$ ./interp/interp.exe a.asm
===== symbol =====
1: printf -1
2: malloc -2
3: scanf -3
4: main 15
5: mult 3
===== code =====
0: INT 0,28
1: SUP 0,15
2: RET 0,0
3: INT 0,24
4: LDA 1,20
5: LOD 1,12
6: LOD 1,16
7: MULI 0,0
8: STX 0,1
9: POP 0,1
10: LDA 1,-4
11: LOD 1,20
12: STO 0,1
13: RET 0,0
14: RET 0,0
15: INT 0,16
16: LDA 1,12
17: INT 0,16
18: LITI 0,5
19: LITI 0,6
20: POP 0,5
21: ADDR 0,3
22: CAL 0,0
23: STX 0,1
24: POP 0,1
25: INT 0,12
26: LDA 0,12
27: LOD 1,12
28: POP 0,5
29: ADDR 0,-1
30: CAL 0,0
31: LDA 1,-4
32: LITI 0,0
33: STO 0,1
34: RET 0,0
35: RET 0,0
start execution
result = 30
end execution
```

- ◆ 인자로 a와 b를 입력 받아서 이를 곱한 뒤 값을 리턴하는 함수 mult()를 선언했다.
- ◆ Main 함수에서 int i를 선언하고 mult를 호출하여 a와 b 각각 5, 6을 넣어서 성공적으로 30이라는 값을 리턴 받고 이를 출력했다.

■ 더 복잡한 함수 및 int형 배열 테스트

```

~/compiler/07
212: POP      0,1
213: LOD      1,12
214: LITI     0,10
215: LSSI     0,0
216: JPC      0,235
217: INT      0,12
218: LDA      0,52
219: LDA      0,12
220: LOD      1,12
221: LITI     0,4
222: MULI     0,0
223: OFFSET   0,0
224: LDI      0,1
225: POP      0,5
226: ADDR     0,-1
227: CAL      0,0
228: LOD      1,12
229: LDA      1,12
230: LDX      0,1
231: INCI     0,0
232: STO      0,1
233: POP      0,1
234: JMP      0,213
235: INT      0,12
236: LDA      0,60
237: POP      0,4
238: ADDR     0,-1
239: CAL      0,0
240: INT      0,12
241: LITI     0,0
242: LITI     0,9
243: POP      0,5
244: ADDR     0,3
245: CAL      0,0
246: LDA      1,12
247: LITI     0,0
248: STX      0,1
249: POP      0,1
250: LOD      1,12
251: LITI     0,10
252: LSSI     0,0
253: JPC      0,272
254: INT      0,12
255: LDA      0,68
256: LDA      0,12
257: LOD      1,12
258: LITI     0,4
259: MULI     0,0
260: OFFSET   0,0
261: LDI      0,1
262: POP      0,5
263: ADDR     0,-1
264: CAL      0,0
265: LOD      1,12
266: LDA      1,12
267: LDX      0,1
268: INCI     0,0
269: STO      0,1
270: POP      0,1
271: JMP      0,250
272: INT      0,12
273: LDA      0,76
274: POP      0,4
275: ADDR     0,-1
276: CAL      0,0
277: RET      0,0

start execution
0 1 3 5 7 9 2 4 6 8
0 1 2 3 4 5 6 7 8 9
end execution

neosk@neoskyclad-GRAM ~/compiler/07
$ |

```

- ◆ int형으로 크기가 10인 배열을 선언하여 무작위로 숫자를 부여하고, 이를 sort함수에서 값을 오름차순으로 정렬하여 출력하는 코드를 생성했다.
- ◆ 초기값을 for문을 통해 출력하고 sort를 통해 정렬한 뒤 성공적으로 정렬된 값을 배열을 참조하여 출력하고 있다.

- 잘못된 C언어 코드

■ Struct 테스트

```
neosk@neoskyclad-GRAM ~/compiler/07
$ ./a.exe test.c

start syntax analysis

start semantic analysis
start code generation
*** error at line 11: not implemented N_EXP_STRUCT for code generation
*** error at line 11: not implemented T_STRUCT for code generation
```

- ◆ Struct 타입으로 int i와 char c를 갖는 변수 s를 선언했다. 이후 main 함수에서 s.i = 10을 통해 구조체의 값을 설정하고 printf로 이를 출력하는 테스트 코드를 작성했다.
- ◆ Syntax와 semantic 분석기에서는 struct에 대해 처리를 해줬기 때문에 무사히 통과했으나, 어셈블리어 번역기에서 struct에 대해 코드를 구현하지 않았기 때문에 error가 발생했다.

4. 원시프로그램

- gen.c

```
#include <stdio.h>
#include <string.h>
#include "type.h"
#include "interp/type.h"

char *opcode_name[] = {"OP_NULL", "LOD", "LDX", "LDXB", "LDA",
    "LITI",
    "STO", "STOB", "STX", "STXB", "SUBI", "SUBF",
    "DIVI", "DIVF", "ADDI", "ADDF",
    "OFFSET", "MULI", "MULF", "MOD", "LSSI",
    "LSSF", "GTRI", "GTRF", "LEQI", "LEQF",
    "GEQI", "GEQF", "NEQI", "NEQF", "EQLI",
    "EQLF", "NOT", "OR", "AND",
    "CVTI", "CVTF", "JPC", "JPCR", "JMP", "JPT",
    "JPTR",
    "INT", "INCI", "INCF", "DECI", "DECF", "SUP",
    "CAL", "ADDR",
    "RET", "MINUSI", "MINUSF", "CHK", "LDI",
    "LDIB", "SWITCH", "SWVALUE",
    "SWDEFAULT", "SWLABEL", "SWEND", "POP",
    "POPB"};

typedef enum
{
    SW_VALUE,
    SW_DEFAULT
} SW_KIND;
```

```

- typedef struct sw
- {
-     SW_KIND kind;
-     int val;
-     int label;
- } A_SWITCH;
-
- void code_generation(A_NODE *);
-
- void gen_literal_table();
- void gen_program(A_NODE *);
- void gen_expression(A_NODE *);
- void gen_expression_left(A_NODE *);
- void gen_arg_expression(A_NODE *);
- void gen_statement(A_NODE *, int, int, A_SWITCH[], int *);
- void gen_statement_list(A_NODE *, int, int, A_SWITCH[], int *);
- void gen_initializer_global(A_NODE *, A_TYPE *, int);
- void gen_initializer_local(A_NODE *, A_TYPE *, int);
- void gen_declaration_list(A_ID *);
- void gen_declaration(A_ID *);
- void gen_code_i(OPCODE, int, int);
- void gen_code_f(OPCODE, int, float);
- void gen_code_s(OPCODE, int, char *);
- void gen_code_l(OPCODE, int, int);
- void gen_label_number(int);
- void gen_label_name(char *);
- void gen_error();
-
- int get_label();
-
- int label_no = 0;
- int gen_err = 0;
-
- extern FILE *fout;
- extern A_TYPE *int_type, *float_type, *char_type, *void_type,
- *string_type;
- extern A_LITERAL literal_table[];
- extern int literal_no;
-
- void code_generation(A_NODE *node)
- {
-     gen_program(node);

```



```

-     gen_literal_table();
- }
-
- void gen_literal_table()
- {
-     int i;
-     for (i = 1; i <= literal_no; i++)
-     {
-         fprintf(fout, ".literal %5d ", literal_table[i].addr);
-         if (literal_table[i].type == int_type)
-             fprintf(fout, "%d\n", literal_table[i].value.i);
-         else if (literal_table[i].type == float_type)
-             fprintf(fout, "%f\n", literal_table[i].value.f);
-         else if (literal_table[i].type == char_type)
-             fprintf(fout, "%d\n", literal_table[i].value.c);
-         else if (literal_table[i].type == string_type)
-             fprintf(fout, "%s\n", literal_table[i].value.s);
-     }
- }
-
- void gen_program(A_NODE *node)
- {
-     switch (node->name)
-     {
-         case N_PROGRAM:
-             gen_code_i(INT, 0, node->value);
-             gen_code_s(SUP, 0, "main");
-             gen_code_i(RET, 0, 0);
-             gen_declaration_list(node->clink);
-             break;
-     }
- }
-
- void gen_expression(A_NODE *node)
- {
-     A_ID *id;
-     A_TYPE *t;
-     int i, ll;
-     switch (node->name)
-     {
-         case N_EXP_IDENT:
-             id = node->clink;

```

```

-     t = id->type;
-     switch (id->kind)
-     {
-     case ID_VAR:
-     case ID_PARM:
-         switch (t->kind)
-         {
-         case T_ENUM:
-         case T_POINTER:
-             gen_code_i(LOD, id->level, id->address);
-             break;
-         case T_ARRAY:
-             if (id->kind == ID_VAR)
-                 gen_code_i(LDA, id->level, id->address);
-             else
-                 gen_code_i(LOD, id->level, id->address);
-             break;
-         case T_STRUCT:
-             gen_error(24, node->line, "T_STRUCT");
-         case T_UNION:
-             gen_code_i(LDA, id->level, id->address);
-             i = id->type->size;
-             gen_code_i(LDI, 0, i % 4 ? i / 4 + 1 : i / 4);
-             break;
-         default:
-             gen_error(11, id->line);
-             break;
-         }
-         break;
-     case ID_ENUM_LITERAL:
-         gen_code_i(LITI, 0, id->init);
-         break;
-     default:
-         gen_error(11, node->line);
-         break;
-     }
-     break;
- case N_EXP_INT_CONST:
-     gen_code_i(LITI, 0, node->clink);
-     break;
- case N_EXP_FLOAT_CONST:
-     i = node->clink;

```

```

-         gen_code_i(LOD, 0, literal_table[i].addr);
-         break;
-     case N_EXP_CHAR_CONST:
-         gen_code_i(LITI, 0, node->clink);
-         break;
-     case N_EXP_STRING_LITERAL:
-         i = node->clink;
-         gen_code_i(LDA, 0, literal_table[i].addr);
-         break;
-     case N_EXP_ARRAY:
-         gen_expression(node->llink);
-         gen_expression(node->rlink);
-         // gen_code_i(CHK,0,node->llink->type->expr);
-         if (node->type->size > 1)
-         {
-             gen_code_i(LITI, 0, node->type->size);
-             gen_code_i(MULI, 0, 0);
-         }
-         gen_code_i(OFFSET, 0, 0);
-         if (!isArrayType(node->type))
-         {
-             i = node->type->size;
-             if (i == 1)
-                 gen_code_i(LDIB, 0, 0);
-             else
-                 gen_code_i(LDI, 0, i % 4 ? i / 4 + 1 : i / 4);
-         }
-         break;
-     case N_EXP_FUNCTION_CALL:
-         t = node->llink->type;
-         i = t->element_type->element_type->size;
-         if (i % 4)
-             i = i / 4 * 4 + 4;
-         if (node->rlink)
-         {
-             gen_code_i(INT, 0, 12 + i);
-             gen_arg_expression(node->rlink);
-             gen_code_i(POP, 0, node->rlink->value / 4 + 3);
-         }
-         else
-             gen_code_i(INT, 0, i);
-         gen_expression(node->llink);

```

```

-     gen_code_i(CAL, 0, 0);
-     break;
- case N_EXP_STRUCT:
-     gen_error(24, node->line, "N_EXP_STRUCT");
- case N_EXP_ARROW:
-     gen_expression(node->llink);
-     id = node->rlink;
-     if (id->address > 0)
-     {
-         gen_code_i(LITI, 0, id->address);
-         gen_code_i(OFFSET, 0, 0);
-     }
-     if (!isArrayType(node->type))
-     {
-         i = node->type->size;
-         if (i == 1)
-             gen_code_i(LDIB, 0, 0);
-         else
-             gen_code_i(LDI, 0, i % 4 ? i / 4 + 1 : i / 4);
-     }
-     break;
- case N_EXP_POST_INC:
-     gen_expression(node->clink);
-     gen_expression_left(node->clink);
-     t = node->type;
-     if (node->type->size == 1)
-         gen_code_i(LDXB, 0, 0);
-     else
-         gen_code_i(LDX, 0, 1);
-     if (isPointerOrArrayType(node->type))
-     {
-         gen_code_i(LITI, 0, node->type->element_type->size);
-         gen_code_i(ADDI, 0, 0);
-     }
-     else if (isFloatType(node->type))
-         gen_code_i(INCF, 0, 0);
-     else
-         gen_code_i(INCI, 0, 0);
-     if (node->type->size == 1)
-         gen_code_i(STOB, 0, 0);
-     else
-         gen_code_i(STO, 0, 1);

```

```

-         break;
-     case N_EXP_POST_DEC:
-         gen_expression(node->clink);
-         gen_expression_left(node->clink);
-         t = node->type;
-         if (node->type->size == 1)
-             gen_code_i(LDXB, 0, 0);
-         else
-             gen_code_i(LDX, 0, 1);
-         if (isPointerOrArrayType(node->type))
-         {
-             gen_code_i(LITI, 0, node->type->element_type->size);
-             gen_code_i(SUBI, 0, 0);
-         }
-         else if (isFloatType(node->type))
-             gen_code_i(DECF, 0, 0);
-         else
-             gen_code_i(DECI, 0, 0);
-         if (node->type->size == 1)
-             gen_code_i(STOB, 0, 0);
-         else
-             gen_code_i(STO, 0, 1);
-         break;
-     case N_EXP_PRE_INC:
-         gen_expression_left(node->clink);
-         t = node->type;
-         if (node->type->size == 1)
-             gen_code_i(LDXB, 0, 0);
-         else
-             gen_code_i(LDX, 0, 1);
-         if (isPointerOrArrayType(node->type))
-         {
-             gen_code_i(LITI, 0, node->type->element_type->size);
-             gen_code_i(ADDI, 0, 0);
-         }
-         else if (isFloatType(node->type))
-             gen_code_i(INCF, 0, 0);
-         else
-             gen_code_i(INCI, 0, 0);
-         if (node->type->size == 1)
-             gen_code_i(STXB, 0, 0);
-         else

```

```

-         gen_code_i(STX, 0, 1);
-     break;
- case N_EXP_PRE_DEC:
-     gen_expression_left(node->clink);
-     t = node->type;
-     if (node->type->size == 1)
-         gen_code_i(LDXB, 0, 0);
-     else
-         gen_code_i(LDX, 0, 1);
-     if (isPointerOrArrayType(node->type))
-     {
-         gen_code_i(LITI, 0, node->type->element_type->size);
-         gen_code_i(SUBI, 0, 0);
-     }
-     else if (isFloatType(node->type))
-         gen_code_i(DECF, 0, 0);
-     else
-         gen_code_i(DECI, 0, 0);
-     if (node->type->size == 1)
-         gen_code_i(STXB, 0, 0);
-     else
-         gen_code_i(STX, 0, 1);
-     break;
- case N_EXP_NOT:
-     gen_expression(node->clink);
-     gen_code_i(NOT, 0, 0);
-     break;
- case N_EXP_PLUS:
-     gen_expression(node->clink);
-     break;
- case N_EXP_MINUS:
-     gen_expression(node->clink);
-     if (isFloatType(node->type))
-         gen_code_i(MINUSF, 0, 0);
-     else
-         gen_code_i(MINUSI, 0, 0);
-     break;
- case N_EXP_AMP:
-     gen_expression_left(node->clink);
-     break;
- case N_EXP_STAR:
-     gen_expression(node->clink);

```

```

-         i = node->type->size;
-         if (i == 1)
-             gen_code_i(LDIB, 0, 0);
-         else
-             gen_code_i(LDI, 0, i % 4 ? i / 4 + 1 : i / 4);
-         break;
-     case N_EXP_SIZE_EXP:
-         gen_code_i(LITI, 0, node->clink);
-         break;
-     case N_EXP_SIZE_TYPE:
-         gen_code_i(LITI, 0, node->clink);
-         break;
-     case N_EXP_CAST:
-         gen_expression(node->rlink);
-         if (node->type != node->rlink->type)
-             if (isFloatType(node->type))
-                 gen_code_i(CVTF, 0, 0);
-             else if (isFloatType(node->rlink->type))
-                 gen_code_i(CVTI, 0, 0);
-         break;
-     case N_EXP_MUL:
-         gen_expression(node->llink);
-         gen_expression(node->rlink);
-         if (isFloatType(node->type))
-             gen_code_i(MULF, 0, 0);
-         else
-             gen_code_i(MULI, 0, 0);
-         break;
-     case N_EXP_DIV:
-         gen_expression(node->llink);
-         gen_expression(node->rlink);
-         if (isFloatType(node->type))
-             gen_code_i(DIVF, 0, 0);
-         else
-             gen_code_i(DIVI, 0, 0);
-         break;
-     case N_EXP_MOD:
-         gen_expression(node->llink);
-         gen_expression(node->rlink);
-         gen_code_i(MOD, 0, 0);
-         break;
-     case N_EXP_ADD:

```

```

-         gen_expression(node->llink);
-         if (isPointerOrArrayType(node->rlink->type))
-         {
-             gen_code_i(LITI, 0, node->rlink->type->element_type-
>size);
-             gen_code_i(MULI, 0, 0);
-         }
-         gen_expression(node->rlink);
-         if (isPointerOrArrayType(node->llink->type))
-         {
-             gen_code_i(LITI, 0, node->llink->type->element_type-
>size);
-             gen_code_i(MULI, 0, 0);
-         }
-         if (isFloatType(node->type))
-             gen_code_i(ADDF, 0, 0);
-         else
-             gen_code_i(ADDI, 0, 0);
-         break;
-     case N_EXP_SUB:
-         gen_expression(node->llink);
-         gen_expression(node->rlink);
-         if (isPointerOrArrayType(node->llink->type)
&& !isPointerOrArrayType(node->rlink->type))
-         {
-             gen_code_i(LITI, 0, node->llink->type->element_type-
>size);
-             gen_code_i(MULI, 0, 0);
-         }
-         if (isFloatType(node->type))
-             gen_code_i(SUBF, 0, 0);
-         else
-             gen_code_i(SUBI, 0, 0);
-         break;
-     case N_EXP_LSS:
-         gen_expression(node->llink);
-         gen_expression(node->rlink);
-         if (isFloatType(node->llink->type))
-             gen_code_i(LSSF, 0, 0);
-         else
-             gen_code_i(LSSI, 0, 0);
-         break;

```



```

- case N_EXP_GTR:
-     gen_expression(node->llink);
-     gen_expression(node->rlink);
-     if (isFloatType(node->llink->type))
-         gen_code_i(GTRF, 0, 0);
-     else
-         gen_code_i(GTRI, 0, 0);
-     break;
- case N_EXP_LEQ:
-     gen_expression(node->llink);
-     gen_expression(node->rlink);
-     if (isFloatType(node->llink->type))
-         gen_code_i(LEQF, 0, 0);
-     else
-         gen_code_i(LEQI, 0, 0);
-     break;
- case N_EXP_GEQ:
-     gen_expression(node->llink);
-     gen_expression(node->rlink);
-     if (isFloatType(node->llink->type))
-         gen_code_i(GEQF, 0, 0);
-     else
-         gen_code_i(GEQI, 0, 0);
-     break;
- case N_EXP_NEQ:
-     gen_expression(node->llink);
-     gen_expression(node->rlink);
-     if (isFloatType(node->llink->type))
-         gen_code_i(NEQF, 0, 0);
-     else
-         gen_code_i(NEQI, 0, 0);
-     break;
- case N_EXP_EQL:
-     gen_expression(node->llink);
-     gen_expression(node->rlink);
-     if (isFloatType(node->llink->type))
-         gen_code_i(EQLF, 0, 0);
-     else
-         gen_code_i(EQLI, 0, 0);
-     break;
- case N_EXP_AND:
-     gen_expression(node->llink);

```

```

-     gen_code_l(JPCR, 0, i = get_label());
-     gen_expression(node->rlink);
-     gen_label_number(i);
-     break;
- case N_EXP_OR:
-     gen_expression(node->llink);
-     gen_code_l(JPTR, 0, i = get_label());
-     gen_expression(node->rlink);
-     gen_label_number(i);
-     break;
- case N_EXP_ASSIGN:
-     gen_expression_left(node->llink);
-     gen_expression(node->rlink);
-     i = node->type->size;
-     if (i == 1)
-         gen_code_i(STXB, 0, 0);
-     else
-         gen_code_i(STX, 0, i % 4 ? i / 4 + 1 : i / 4);
-     break;
- default:
-     gen_error(100, node->line);
-     break;
- }
- }

- void gen_expression_left(A_NODE *node)
- {
-     A_ID *id;
-     A_TYPE *t;
-     int result;
-     switch (node->name)
-     {
-
-     case N_EXP_IDENT:
-         id = node->clink;
-         t = id->type;
-         switch (id->kind)
-         {
-         case ID_VAR:
-         case ID_PARM:
-             switch (t->kind)
-             {

```

```

-         case T_ENUM:
-         case T_POINTER:
-         case T_STRUCT:
-         case T_UNION:
-             gen_code_i(LDA, id->level, id->address);
-             break;
-         case T_ARRAY:
-             if (id->kind == ID_VAR)
-                 gen_code_i(LDA, id->level, id->address);
-             else
-                 gen_code_i(LOD, id->level, id->address);
-             break;
-         default:
-             gen_error(13, node->line, id->name);
-             break;
-     }
-     break;
- case ID_FUNC:
-     gen_code_s(ADDR, 0, id->name);
-     break;
- default:
-     gen_error(13, node->line, id->name);
-     break;
- }
- break;
- case N_EXP_ARRAY:
-     gen_expression(node->llink);
-     gen_expression(node->rlink);
-     // gen_code_i(CHK, 0, node->llink->type->expr);
-     if (node->type->size > 1)
-     {
-         gen_code_i(LITI, 0, node->type->size);
-         gen_code_i(MULI, 0, 0);
-     }
-     gen_code_i(OFFSET, 0, 0);
-     break;
- case N_EXP_STRUCT:
-     gen_expression_left(node->llink);
-     id = node->rlink;
-     if (id->address > 0)
-     {
-         gen_code_i(LITI, 0, id->address);

```

```

-         gen_code_i(OFFSET, 0, 0);
-     }
-     break;
- case N_EXP_ARROW:
-     gen_expression(node->llink);
-     id = node->rlink;
-     if (id->address > 0)
-     {
-         gen_code_i(LITI, 0, id->address);
-         gen_code_i(OFFSET, 0, 0);
-     }
-     break;
- case N_EXP_STAR:
-     gen_expression(node->clink);
-     break;
- case N_EXP_INT_CONST:
- case N_EXP_FLOAT_CONST:
- case N_EXP_CHAR_CONST:
- case N_EXP_STRING_LITERAL:
- case N_EXP_FUNCTION_CALL:
- case N_EXP_POST_INC:
- case N_EXP_POST_DEC:
- case N_EXP_PRE_INC:
- case N_EXP_PRE_DEC:
- case N_EXP_NOT:
- case N_EXP_MINUS:
- case N_EXP_SIZE_EXP:
- case N_EXP_SIZE_TYPE:
- case N_EXP_CAST:
- case N_EXP_MUL:
- case N_EXP_DIV:
- case N_EXP_MOD:
- case N_EXP_ADD:
- case N_EXP_SUB:
- case N_EXP_LSS:
- case N_EXP_GTR:
- case N_EXP_LEQ:
- case N_EXP_GEQ:
- case N_EXP_NEQ:
- case N_EXP_EQL:
- case N_EXP_AMP:
- case N_EXP_AND:

```

```

-     case N_EXP_OR:
-     case N_EXP_ASSIGN:
-         gen_error(12, node->line);
-         break;
-     default:
-         gen_error(100, node->line);
-         break;
-     }
- }

- void gen_arg_expression(A_NODE *node)
- {
-     A_NODE *n;
-     switch (node->name)
-     {
-     case N_ARG_LIST:
-         gen_expression(node->llink);
-         gen_arg_expression(node->rlink);
-         break;
-     case N_ARG_LIST_NIL:
-         break;
-     default:
-         gen_error(100, node->line);
-         break;
-     }
- }

- int get_label()
- {
-     label_no++;
-     return (label_no);
- }

- void gen_statement(A_NODE *node, int cont_label, int break_label,
- A_SWITCH sw[], int *sn)
- {
-     A_NODE *n;
-     int i, l1, l2, l3;
-     switch (node->name)
-     {
-     case N_STMT_LABEL_CASE:
-     case N_STMT_LABEL_DEFAULT:

```

```

-         break;
-     case N_STMT_COMPOUND:
-         if (node->llink)
-             gen_declaration_list(node->llink);
-         gen_statement_list(node->rlink, cont_label, break_label, sw,
sn);
-         break;
-     case N_STMT_EMPTY:
-         break;
-     case N_STMT_EXPRESSION:
-         n = node->clink;
-         gen_expression(n);
-         i = n->type->size;
-         if (i)
-             gen_code_i(POP, 0, i % 4 ? i / 4 + 1 : i / 4);
-         break;
-     case N_STMT_IF:
-         gen_expression(node->llink);
-         gen_code_l(JPC, 0, l1 = get_label());
-         gen_statement(node->rlink, cont_label, break_label, 0, 0);
-         gen_label_number(l1);
-         break;
-     case N_STMT_IF_ELSE:
-         gen_expression(node->llink);
-         gen_code_l(JPC, 0, l1 = get_label());
-         gen_statement(node->clink, cont_label, break_label, 0, 0);
-         gen_code_l(JMP, 0, l2 = get_label());
-         gen_label_number(l1);
-         gen_statement(node->rlink, cont_label, break_label, 0, 0);
-         gen_label_number(l2);
-         break;
-     case N_STMT_SWITCH:
-         break;
-     case N_STMT_WHILE:
-         l3 = get_label();
-         gen_label_number(l1 = get_label());
-         gen_expression(node->llink);
-         gen_code_l(JPC, 0, l2 = get_label());
-         gen_statement(node->rlink, l3, l2, 0, 0);
-         gen_label_number(l3);
-         gen_code_l(JMP, 0, l1);
-         gen_label_number(l2);

```

```

-         break;
-     case N_STMT_D0:
-         l3 = get_label();
-         l2 = get_label();
-         gen_label_number(l1 = get_label());
-         gen_statement(node->llink, l2, l3, 0, 0);
-         gen_label_number(l2);
-         gen_expression(node->rlink);
-         gen_code_l(JPT, 0, l1);
-         gen_label_number(l3);
-         break;
-     case N_STMT_FOR:
-         n = node->llink;
-         l3 = get_label();
-         if (n->llink)
-         {
-             gen_expression(n->llink);
-             i = n->llink->type->size;
-             if (i)
-                 gen_code_i(POP, 0, i % 4 ? i / 4 + 1 : i / 4);
-         }
-         gen_label_number(l1 = get_label());
-         l2 = get_label();
-         if (n->clink)
-         {
-             gen_expression(n->clink);
-             gen_code_l(JPC, 0, l2);
-         }
-         gen_statement(node->rlink, l3, l2, 0, 0);
-         gen_label_number(l3);
-         if (n->rlink)
-         {
-             gen_expression(n->rlink);
-             i = n->rlink->type->size;
-             if (i)
-                 gen_code_i(POP, 0, i % 4 ? i / 4 + 1 : i / 4);
-         }
-         gen_code_l(JMP, 0, l1);
-         gen_label_number(l2);
-         break;
-     case N_STMT_CONTINUE:
-         if (cont_label)

```

```

-         gen_code_1(JMP, 0, cont_label);
-     else
-         gen_error(22, node->line);
-     break;
- case N_STMT_BREAK:
-     if (break_label)
-         gen_code_1(JMP, 0, break_label);
-     else
-         gen_error(23, node->line);
-     break;
- case N_STMT_RETURN:
-     n = node->clink;
-     if (n)
-     {
-         i = n->type->size;
-         if (i % 4)
-             i = i / 4 * 4 + 4;
-         gen_code_i(LDA, 1, -i);
-         gen_expression(n);
-         gen_code_i(STO, 0, i / 4);
-     }
-     gen_code_i(RET, 0, 0);
-     break;
- default:
-     gen_error(100, node->line);
-     break;
- }
- }

- void gen_statement_list(A_NODE *node, int cont_label, int
- break_label, A_SWITCH sw[], int *sn)
- {
-     switch (node->name)
-     {
-     case N_STMT_LIST:
-         gen_statement(node->llink, cont_label, break_label, sw, sn);
-         gen_statement_list(node->rlink, cont_label, break_label, sw,
- sn);
-         break;
-     case N_STMT_LIST_NIL:
-         break;
-     default:

```



```

-         gen_error(100, node->line);
-         break;
-     }
- }
-
- void gen_initializer_global(A_NODE *node, A_TYPE *t, int addr)
- {
-     //구현 x
- }
- void gen_initializer_local(A_NODE *node, A_TYPE *t, int addr)
- {
-     //구현 x
- }
-
- void gen_declaration_list(A_ID *id)
- {
-     while (id)
-     {
-         gen_declaration(id);
-         id = id->link;
-     }
- }
-
- void gen_declaration(A_ID *id)
- {
-     int i;
-     A_NODE *node;
-     switch (id->kind)
-     {
-     case ID_VAR:
-         if (id->init)
-             if (id->level == 0)
-                 gen_initializer_global(id->init, id->type, id-
- >address);
-             else
-                 gen_initializer_local(id->init, id->type, id-
- >address);
-         break;
-     case ID_FUNC:
-         if (id->type->expr)
-         {
-             gen_label_name(id->name);

```

```

-         gen_code_i(INT, 0, id->type->local_var_size);
-         gen_statement(id->type->expr, 0, 0, 0, 0);
-         gen_code_i(RET, 0, 0);
-     }
-     break;
- case ID_PARM:
- case ID_TYPE:
- case ID_ENUM:
- case ID_STRUCT:
- case ID_FIELD:
- case ID_ENUM_LITERAL:
- case ID_NULL:
-     break;
- default:
-     gen_error(100, id->line);
-     break;
- }
- }

- void gen_error(int i, int ll, char *s)
- {
-     gen_err++;
-     printf("*** error at line %d: ", ll);

-     switch (i)
-     {
-     case 11:
-         printf("illegal identifier in expression \n");
-         break;
-     case 12:
-         printf("illegal l-value expression \n");
-         break;
-     case 13:
-         printf("identifier %s not l-value expression \n", s);
-         break;
-     case 20:
-         printf("illegal default label in switch statement \n");
-         break;
-     case 21:
-         printf("illegal case label in switch statement \n");
-         break;
-     case 22:

```

```

-         printf("no destination for continue statement \n");
-         break;
-     case 23:
-         printf("no destination for break statement \n");
-         break;
-     case 24:
-         printf("not implemented %s for code generation\n", s);
-         break;
-     case 100:
-         printf("fatal compiler error during code generation\n");
-         break;
-     default:
-         printf("unknown \n");
-         break;
- }
- }

- void gen_code_i(OPCODE op, int l, int a)
- {
-     fprintf(fout, "\t%9s    %d, %d\n", opcode_name[op], l, a);
- }

- void gen_code_f(OPCODE op, int l, float a)
- {
-     fprintf(fout, "\t%9s    %d, %f\n", opcode_name[op], l, a);
- }

- void gen_code_s(OPCODE op, int l, char *a)
- {
-     fprintf(fout, "\t%9s    %d, %s\n", opcode_name[op], l, a);
- }

- void gen_code_l(OPCODE op, int l, int a)
- {
-     fprintf(fout, "\t%9s    %d, L%d\n", opcode_name[op], l, a);
- }

- void gen_label_number(int i)
- {
-     fprintf(fout, "L%d:\n", i);
- }

```

```
- void gen_label_name(char *s)
- {
-     fprintf(fout, "%s:\n", s);
- }
-
```