과제 #7. Code Generation

(2023-1 Compiler\_Final)

소프트웨어학부 20180325 임성현

1. 과제 내용

* 이전 과제를 토대로 어셈블러 코드를 만든 뒤, 교수님이 제공한 인터프리터를 이용하여 코드 생성이 제대로 되었는지 확인한다.

2. 문제 해결 방법

* 수업 강의 자료와 교재 부록을 활용하여 generation 소스코드를 추가 및 수정하였다.
* gcc 컴파일 결과를 참고하였다.
* makefile을 활용하였다.
* **struct, union, switch, initializer는 교수님 지침에 따라 구현하지 않았다.**

3. 테스트 케이스

- C 소스 코드

텍스트, 스크린샷, 번호, 메뉴이(가) 표시된 사진

자동 생성된 설명

* 어셈블러 코드

텍스트, 스크린샷, 번호, 소프트웨어이(가) 표시된 사진

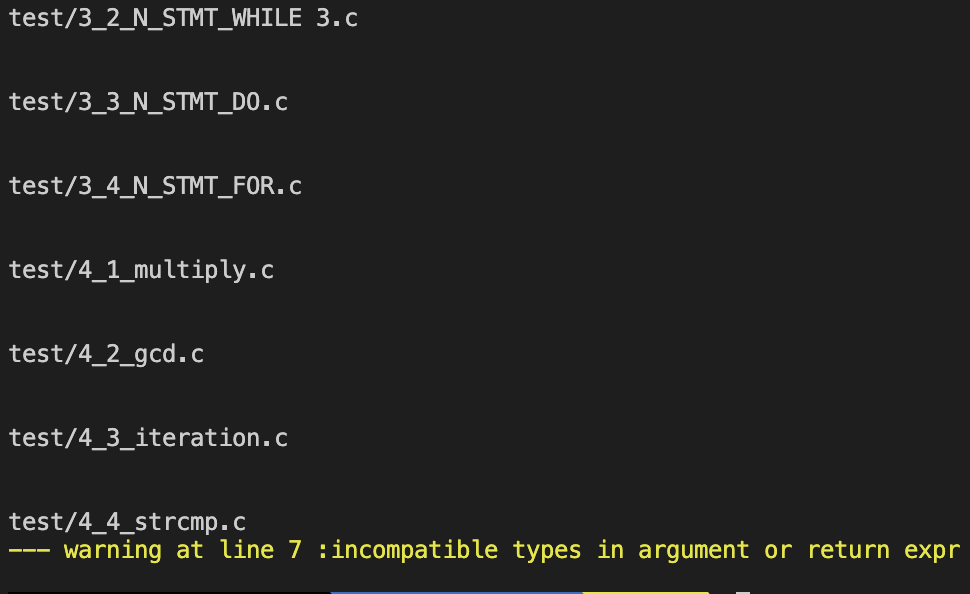
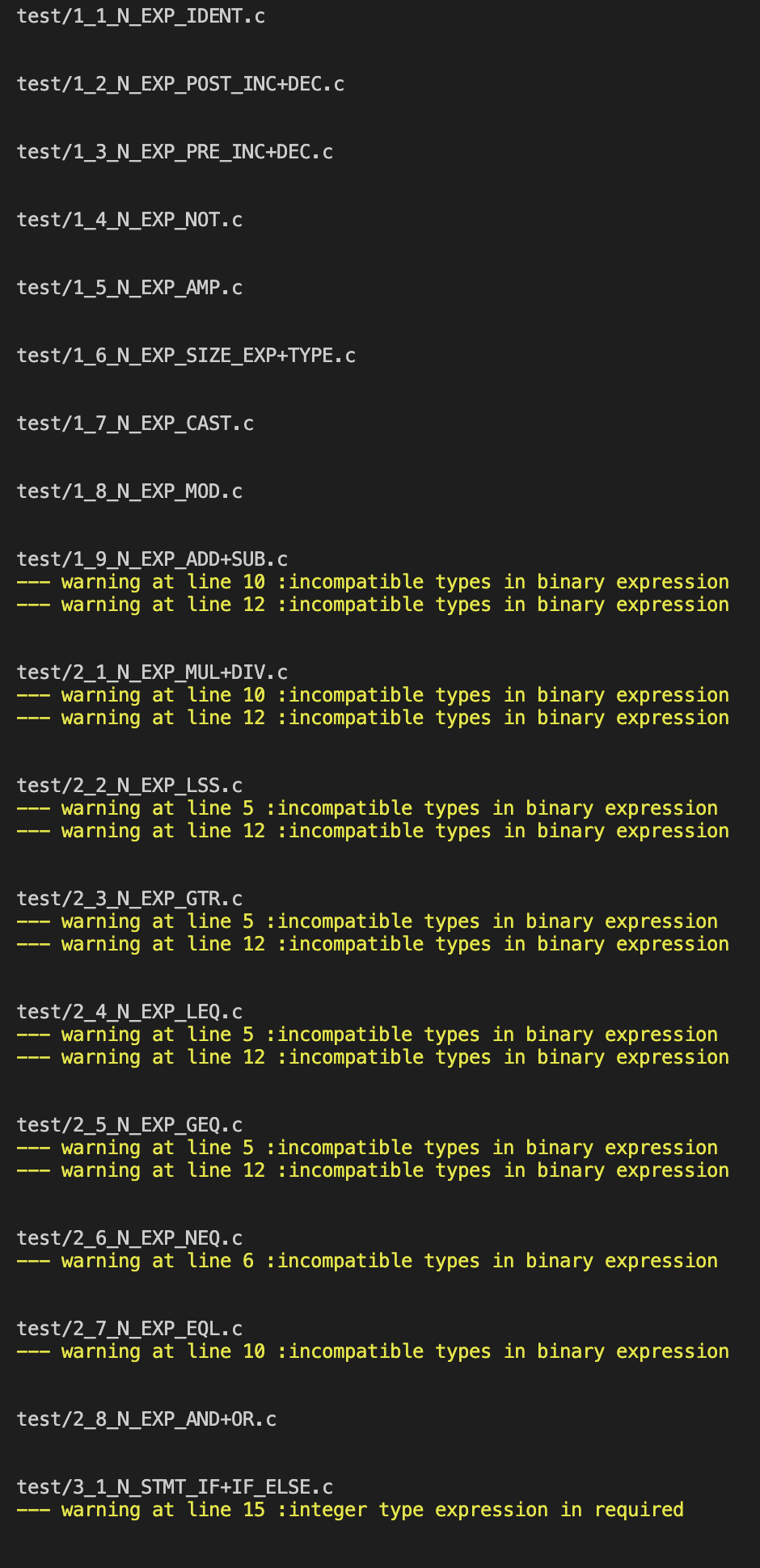
자동 생성된 설명

4. 실행 결과 [요약]

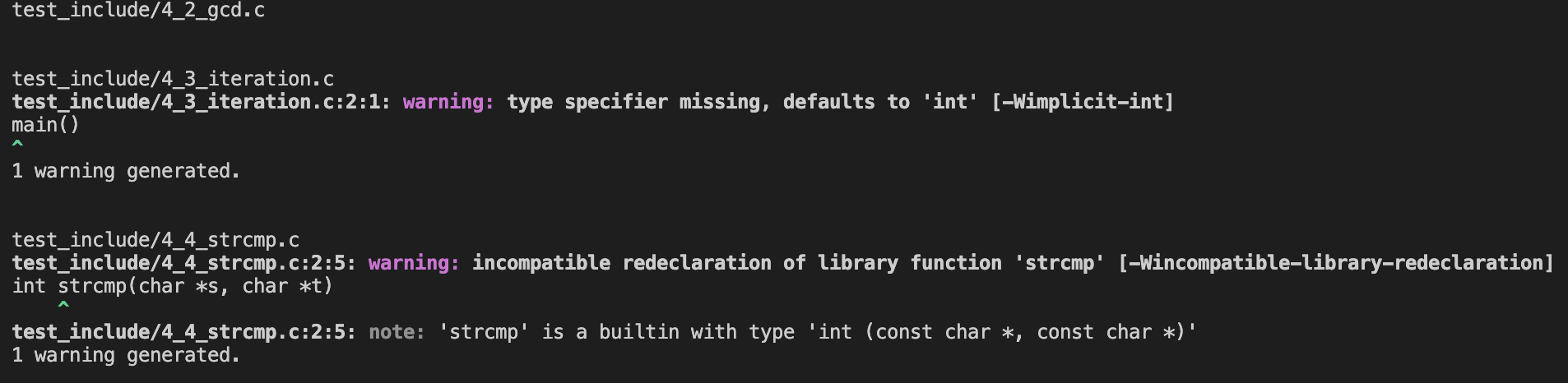
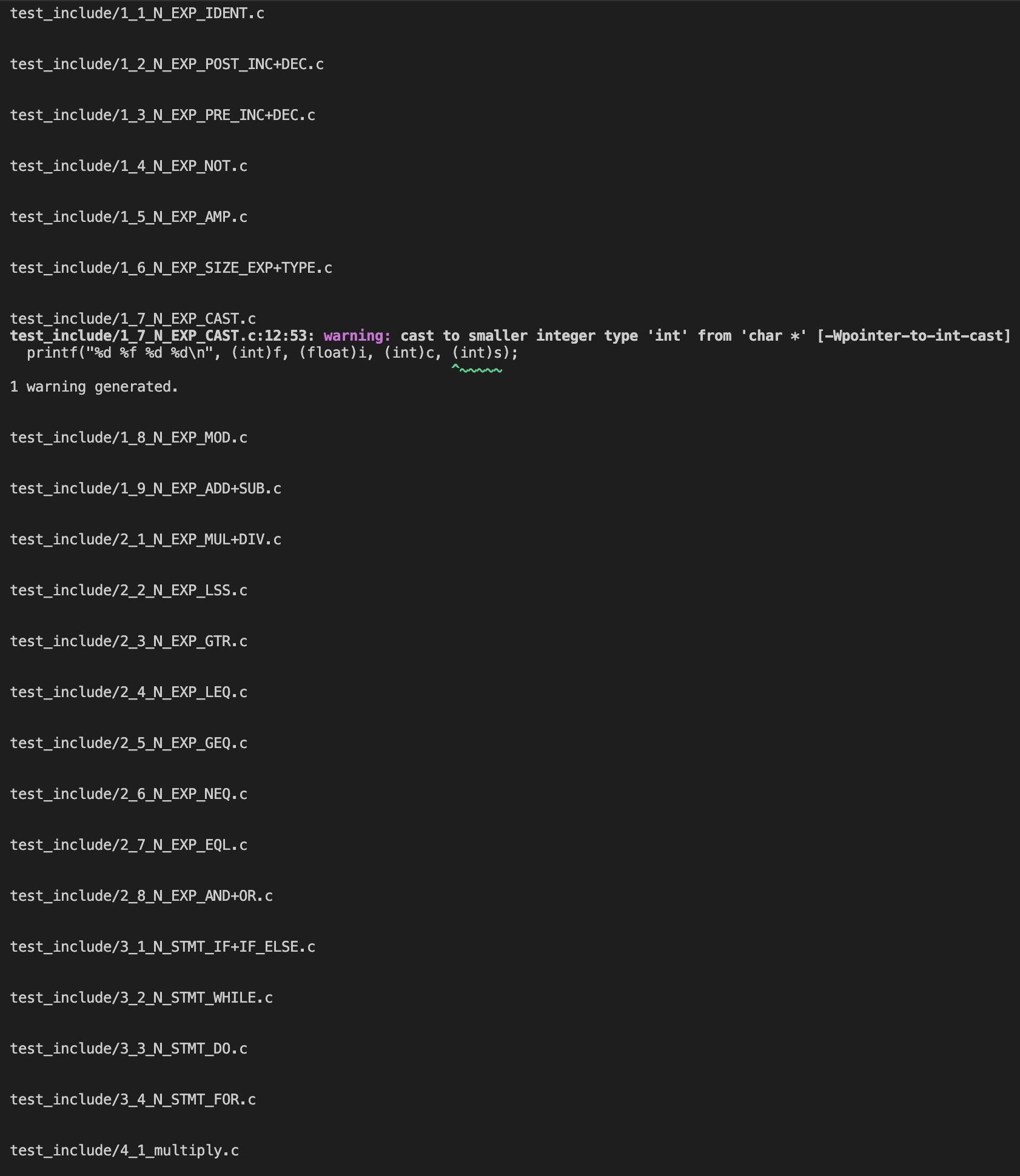
* 제작한 컴파일러와 gcc 컴파일러와 비교
  + compile warning 또는 compile error이 발생하지 않은 경우 공백으로 두었음.
  + 편의상, gcc 컴파일시 #include<stdio.h> 헤더 파일을 추가하고, main() 함수의 return type을 int로 하였음. (test\_include라는 별도 폴더를 둠)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | 파일명 | 컴파일러 | gcc | 설명 |
| 1 | 1\_1\_N\_EXP\_IDENT.c |  |  |  |
| 2 | 1\_2\_N\_EXP\_POST\_INC+DEC.c |  |  |  |
| 3 | 1\_3\_N\_EXP\_PRE\_INC+DEC.c |  |  |  |
| 4 | 1\_4\_N\_EXP\_NOT.c |  |  |  |
| 5 | 1\_5\_N\_EXP\_AMP.c |  |  |  |
| 6 | 1\_6\_N\_EXP\_SIZE\_EXP+TYPE.c |  |  |  |
| 7 | 1\_7\_N\_EXP\_CAST.c |  | warning | cast to smaller integer type 'int' from 'char \*' [-Wpointer-to-int-cast] |
| 8 | 1\_8\_N\_EXP\_MOD.c |  |  |  |
| 9 | 1\_9\_N\_EXP\_ADD+SUB.c | warning |  | incompatible types in binary expression |
| 10 | 2\_1\_N\_EXP\_MUL+DIV.c | warning |  | incompatible types in binary expression |
| 11 | 2\_2\_N\_EXP\_LSS.c | warning |  | incompatible types in binary expression |
| 12 | 2\_3\_N\_EXP\_GTR.c | warning |  | incompatible types in binary expression |
| 13 | 2\_4\_N\_EXP\_LEQ.c | warning |  | incompatible types in binary expression |
| 14 | 2\_5\_N\_EXP\_GEQ.c | warning |  | incompatible types in binary expression |
| 15 | 2\_6\_N\_EXP\_NEQ.c | warning |  | incompatible types in binary expression |
| 16 | 2\_7\_N\_EXP\_EQL.c | warning |  | incompatible types in binary expression |
| 17 | 2\_8\_N\_EXP\_AND+OR.c |  |  | incompatible types in binary expression |
| 18 | 3\_1\_N\_STMT\_IF+IF\_ELSE.c | warning |  | incompatible types in binary expression |
| 19 | 3\_2\_N\_STMT\_WHILE 3.c |  |  |  |
| 20 | 3\_3\_N\_STMT\_DO.c |  |  |  |
| 21 | 3\_4\_N\_STMT\_FOR.c |  |  |  |
| 22 | 4\_1\_multiply.c |  |  |  |
| 23 | 4\_2\_gcd.c |  |  |  |
| 24 | 4\_3\_iteration.c |  | warning | type specifier missing, defaults to 'int' [-Wimplicit-int]  main() |
| 25 | 4\_4\_strcmp.c | warning | warning | **제작한 컴파일러 :**  incompatible types in argument or return expr  **gcc 컴파일러 :**  incompatible redeclaration of library function 'strcmp' [-Wincompatible-library-redeclaration] |

* 참고 자료
  + 제작한 컴파일러를 통한 컴파일 결과



* + gcc 컴파일러를 통한 컴파일 결과



* **1**
  + 1\_1\_N\_EXP\_IDENT.c

enum e

{

// N\_EXP\_IDENT - ID\_ENUM\_LITERAL

A = 10,

B = 20

};

// N\_EXP\_IDENT - ID\_PARM

void func(int i, float f, char c, char \*s, int \*p, int arr[])

{

printf("%d %f %c %s %d %d %d %d\n", i, f, c, s, \*p, arr[0], arr[1], arr[2]);

}

void main()

{

// compound\_statment : {declaration\_list statement\_list} 이므로

// 미리 선언하고 시작한다.

// N\_EXP\_IDENT - ID\_VAR

int i;

float f;

char c;

char \*s;

int \*p;

int arr[3];

i = 10 + A; // N\_EXP\_INT\_CONST

f = 3.14; // N\_EXP\_FLOAT\_CONST

c = 'a'; // N\_EXP\_CHAR\_CONST

s = "string"; // N\_EXP\_STRING\_LITERAL

\*p = i + B;

// N\_EXP\_ARRAY

arr[0] = 1;

arr[1] = 2;

arr[2] = 3;

// N\_EXP\_FUNCTION\_CALL

printf("%d %f %c %s %d %d %d %d\n", i, f, c, s, \*p, arr[0], arr[1], arr[2]);

func(i, f, c, s, p, arr);

}

* + 1\_1\_N\_EXP\_IDENT.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 34

5: func 3

======== code =========

0: INT 0,88

1: SUP 0,34

2: RET 0,0

3: INT 0,36

4: INT 0,12

5: LDA 0,16

6: LOD 1,12

7: LOD 1,16

8: LOD 1,20

9: LOD 1,24

10: LOD 1,28

11: LDI 0,1

12: LOD 1,32

13: LITI 0,0

14: LITI 0,4

15: MULI 0,0

16: OFFSET 0,0

17: LDI 0,1

18: LOD 1,32

19: LITI 0,1

20: LITI 0,4

21: MULI 0,0

22: OFFSET 0,0

23: LDI 0,1

24: LOD 1,32

25: LITI 0,2

26: LITI 0,4

27: MULI 0,0

28: OFFSET 0,0

29: LDI 0,1

30: POP 0,12

31: ADDR 0,-1

32: CAL 0,0

33: RET 0,0

34: INT 0,44

35: LDA 1,12

36: LITI 0,10

37: LITI 0,10

38: ADDI 0,0

39: STX 0,1

40: POP 0,1

41: LDA 1,16

42: LOD 0,44

43: STX 0,1

44: POP 0,1

45: LDA 1,20

46: LITI 0,97

47: STXB 0,0

48: POP 0,1

49: LDA 1,24

50: LDA 0,48

51: STX 0,1

52: POP 0,1

53: LOD 1,28

54: LOD 1,12

55: LITI 0,20

56: ADDI 0,0

57: STX 0,1

58: POP 0,1

59: LDA 1,32

60: LITI 0,0

61: LITI 0,4

62: MULI 0,0

63: OFFSET 0,0

64: LITI 0,1

65: STX 0,1

66: POP 0,1

67: LDA 1,32

68: LITI 0,1

69: LITI 0,4

70: MULI 0,0

71: OFFSET 0,0

72: LITI 0,2

73: STX 0,1

74: POP 0,1

75: LDA 1,32

76: LITI 0,2

77: LITI 0,4

78: MULI 0,0

79: OFFSET 0,0

80: LITI 0,3

81: STX 0,1

82: POP 0,1

83: INT 0,12

84: LDA 0,60

85: LOD 1,12

86: LOD 1,16

87: LOD 1,20

88: LOD 1,24

89: LOD 1,28

90: LDI 0,1

91: LDA 1,32

92: LITI 0,0

93: LITI 0,4

94: MULI 0,0

95: OFFSET 0,0

96: LDI 0,1

97: LDA 1,32

98: LITI 0,1

99: LITI 0,4

100: MULI 0,0

101: OFFSET 0,0

102: LDI 0,1

103: LDA 1,32

104: LITI 0,2

105: LITI 0,4

106: MULI 0,0

107: OFFSET 0,0

108: LDI 0,1

109: POP 0,12

110: ADDR 0,-1

111: CAL 0,0

112: INT 0,12

113: LOD 1,12

114: LOD 1,16

115: LOD 1,20

116: LOD 1,24

117: LOD 1,28

118: LDA 1,32

119: POP 0,9

120: ADDR 0,3

121: CAL 0,0

122: RET 0,0

======== start execution =========

20 3.140000 a string 40 1 2 3

20 3.140000 a string 40 1 2 3

======== end execution =========

* **2.** 
  + 1\_2\_N\_EXP\_POST\_INC+DEC.c

void main() {

int i;

i = 0;

++i;

printf("%d\n", i);

++i;

printf("%d\n", i);

--i;

printf("%d\n", i);

}

* + 1\_2\_N\_EXP\_POST\_INC+DEC.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 3

======== code =========

0: INT 0,36

1: SUP 0,3

2: RET 0,0

3: INT 0,16

4: LDA 1,12

5: LITI 0,0

6: STX 0,1

7: POP 0,1

8: LDA 1,12

9: LDX 0,1

10: INCI 0,0

11: STX 0,1

12: POP 0,1

13: INT 0,12

14: LDA 0,12

15: LOD 1,12

16: POP 0,5

17: ADDR 0,-1

18: CAL 0,0

19: LDA 1,12

20: LDX 0,1

21: INCI 0,0

22: STX 0,1

23: POP 0,1

24: INT 0,12

25: LDA 0,20

26: LOD 1,12

27: POP 0,5

28: ADDR 0,-1

29: CAL 0,0

30: LDA 1,12

31: LDX 0,1

32: DECI 0,0

33: STX 0,1

34: POP 0,1

35: INT 0,12

36: LDA 0,28

37: LOD 1,12

38: POP 0,5

39: ADDR 0,-1

40: CAL 0,0

41: RET 0,0

======== start execution =========

1

2

1

======== end execution =========

* **3.**
  + 1\_3\_N\_EXP\_PRE\_INC+DEC.c

void main() {

int i;

i = 0;

i++;

printf("%d\n", i);

i++;

printf("%d\n", i);

i--;

printf("%d\n", i);

}

* + 1\_3\_N\_EXP\_PRE\_INC+DEC.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 3

======== code =========

0: INT 0,36

1: SUP 0,3

2: RET 0,0

3: INT 0,16

4: LDA 1,12

5: LITI 0,0

6: STX 0,1

7: POP 0,1

8: LOD 1,12

9: LDA 1,12

10: LDX 0,1

11: INCI 0,0

12: STO 0,1

13: POP 0,1

14: INT 0,12

15: LDA 0,12

16: LOD 1,12

17: POP 0,5

18: ADDR 0,-1

19: CAL 0,0

20: LOD 1,12

21: LDA 1,12

22: LDX 0,1

23: INCI 0,0

24: STO 0,1

25: POP 0,1

26: INT 0,12

27: LDA 0,20

28: LOD 1,12

29: POP 0,5

30: ADDR 0,-1

31: CAL 0,0

32: LOD 1,12

33: LDA 1,12

34: LDX 0,1

35: DECI 0,0

36: STO 0,1

37: POP 0,1

38: INT 0,12

39: LDA 0,28

40: LOD 1,12

41: POP 0,5

42: ADDR 0,-1

43: CAL 0,0

44: RET 0,0

======== start execution =========

1

2

1

======== end execution =========

* **4.**
  + 1\_4\_N\_EXP\_NOT.c

void main() {

int i;

float f;

char c;

i = 10;

f = 10.1;

c = 'a'; // unary conversion으로 인해 char가 int로 변환된다.

printf("%d %d %d\n", !i, !f, !c);

}

* + 1\_4\_N\_EXP\_NOT.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 3

======== code =========

0: INT 0,32

1: SUP 0,3

2: RET 0,0

3: INT 0,24

4: LDA 1,12

5: LITI 0,10

6: STX 0,1

7: POP 0,1

8: LDA 1,16

9: LOD 0,12

10: STX 0,1

11: POP 0,1

12: LDA 1,20

13: LITI 0,97

14: STXB 0,0

15: POP 0,1

16: INT 0,12

17: LDA 0,16

18: LOD 1,12

19: NOT 0,0

20: LOD 1,16

21: NOT 0,0

22: LOD 1,20

23: NOT 0,0

24: POP 0,7

25: ADDR 0,-1

26: CAL 0,0

27: RET 0,0

======== start execution =========

0 0 0

======== end execution =========

* **5.**
  + 1\_5\_N\_EXP\_AMP.c

void main()

{

int i;

scanf("%d", &i);

printf("%d\n", i);

}

* + 1\_5\_N\_EXP\_AMP.c.asm 인터프리터 결과

======== 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: RET 0,0

======== start execution =========

10

10

======== end execution =========

* **6.**
  + 1\_6\_N\_EXP\_SIZE\_EXP+TYPE.c

void main() {

int i\_size, f\_size, c\_size, s\_size;

// N\_EXP\_SIZE\_EXP

i\_size = sizeof(10);

f\_size = sizeof(10.1);

c\_size = sizeof('a');

s\_size = sizeof("size");

printf("%d %d %d %d\n", i\_size, f\_size, c\_size, s\_size);

// N\_EXP\_SIZE\_TYPE

i\_size = sizeof(int);

f\_size = sizeof(float);

c\_size = sizeof(char);

s\_size = sizeof(char \*);

printf("%d %d %d %d\n", i\_size, f\_size, c\_size, s\_size);

}

* + 1\_6\_N\_EXP\_SIZE\_EXP+TYPE.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 3

======== code =========

0: INT 0,56

1: SUP 0,3

2: RET 0,0

3: INT 0,28

4: LDA 1,12

5: LITI 0,4

6: STX 0,1

7: POP 0,1

8: LDA 1,16

9: LITI 0,4

10: STX 0,1

11: POP 0,1

12: LDA 1,20

13: LITI 0,1

14: STX 0,1

15: POP 0,1

16: LDA 1,24

17: LITI 0,4

18: STX 0,1

19: POP 0,1

20: INT 0,12

21: LDA 0,24

22: LOD 1,12

23: LOD 1,16

24: LOD 1,20

25: LOD 1,24

26: POP 0,8

27: ADDR 0,-1

28: CAL 0,0

29: LDA 1,12

30: LITI 0,4

31: STX 0,1

32: POP 0,1

33: LDA 1,16

34: LITI 0,4

35: STX 0,1

36: POP 0,1

37: LDA 1,20

38: LITI 0,1

39: STX 0,1

40: POP 0,1

41: LDA 1,24

42: LITI 0,4

43: STX 0,1

44: POP 0,1

45: INT 0,12

46: LDA 0,40

47: LOD 1,12

48: LOD 1,16

49: LOD 1,20

50: LOD 1,24

51: POP 0,8

52: ADDR 0,-1

53: CAL 0,0

54: RET 0,0

======== start execution =========

4 4 1 4

4 4 1 4

======== end execution =========

* + cf. gcc 컴파일러 실행 결과

텍스트, 폰트, 스크린샷, 블랙이(가) 표시된 사진

자동 생성된 설명

🡪 pointer의 경우, 4 Byte가 아닌 8 Byte로 취급되었다.

🡪 string의 경우, ‘\0’문자가 마지막에 존재하기 때문에 4 Byte가 아닌 5 Byte로 취급 되었다.

* **7.**
  + 1\_7\_N\_EXP\_CAST.c

void main() {

int i;

float f;

char c, \*s;

i = 10;

f = 22.2;

c = 'c';

s = "cast";

printf("%d %f %d %d\n", (int)f, (float)i, (int)c, (int)s);

}

* + 1\_7\_N\_EXP\_CAST.c.asm 인터프리터 실행 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 3

======== code =========

0: INT 0,40

1: SUP 0,3

2: RET 0,0

3: INT 0,28

4: LDA 1,12

5: LITI 0,10

6: STX 0,1

7: POP 0,1

8: LDA 1,16

9: LOD 0,12

10: STX 0,1

11: POP 0,1

12: LDA 1,20

13: LITI 0,99

14: STXB 0,0

15: POP 0,1

16: LDA 1,24

17: LDA 0,16

18: STX 0,1

19: POP 0,1

20: INT 0,12

21: LDA 0,24

22: LOD 1,16

23: CVTI 0,0

24: LOD 1,12

25: CVTF 0,0

26: LOD 1,20

27: LOD 1,24

28: POP 0,8

29: ADDR 0,-1

30: CAL 0,0

31: RET 0,0

======== start execution =========

22 10.000000 99 16

======== end execution =========

* 8.
  + 1\_8\_N\_EXP\_MOD.c

void main() { printf("%d %d\n", 10 % 3, 3 % 10); }

* + 1\_8\_N\_EXP\_MOD.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 3

======== code =========

0: INT 0,24

1: SUP 0,3

2: RET 0,0

3: INT 0,12

4: INT 0,12

5: LDA 0,12

6: LITI 0,10

7: LITI 0,3

8: MOD 0,0

9: LITI 0,3

10: LITI 0,10

11: MOD 0,0

12: POP 0,6

13: ADDR 0,-1

14: CAL 0,0

15: RET 0,0

======== start execution =========

1 3

======== end execution =========

* 9.
  + 1\_9\_N\_EXP\_ADD+SUB.c

void main() {

int i, j;

float f, d;

i = 10;

j = 20;

f = 11.1;

d = 22.2;

// N\_EXP\_ADD

printf("%d %f %f\n", i + j, i + f, f + d);

// N\_EXP\_SUB

printf("%d %f %f\n", i - j, i - f, f - d);

}

* + 1\_9\_N\_EXP\_ADD+SUB.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 3

======== code =========

0: INT 0,52

1: SUP 0,3

2: RET 0,0

3: INT 0,28

4: LDA 1,12

5: LITI 0,10

6: STX 0,1

7: POP 0,1

8: LDA 1,16

9: LITI 0,20

10: STX 0,1

11: POP 0,1

12: LDA 1,20

13: LOD 0,12

14: STX 0,1

15: POP 0,1

16: LDA 1,24

17: LOD 0,16

18: STX 0,1

19: POP 0,1

20: INT 0,12

21: LDA 0,20

22: LOD 1,12

23: LOD 1,16

24: ADDI 0,0

25: LOD 1,12

26: CVTF 0,0

27: LOD 1,20

28: ADDF 0,0

29: LOD 1,20

30: LOD 1,24

31: ADDF 0,0

32: POP 0,7

33: ADDR 0,-1

34: CAL 0,0

35: INT 0,12

36: LDA 0,36

37: LOD 1,12

38: LOD 1,16

39: SUBI 0,0

40: LOD 1,12

41: CVTF 0,0

42: LOD 1,20

43: SUBF 0,0

44: LOD 1,20

45: LOD 1,24

46: SUBF 0,0

47: POP 0,7

48: ADDR 0,-1

49: CAL 0,0

50: RET 0,0

======== start execution =========

30 21.100000 33.300003

-10 -1.100000 -11.100000

======== end execution =========

* 10.
  + 2\_1\_N\_EXP\_MUL+DIV.c

void main() {

int i, j;

float f, d;

i = 10;

j = 20;

f = 11.1;

d = 22.2;

// N\_EXP\_MUL

printf("%d %f %f\n", i \* j, i \* f, f \* d);

// N\_EXP\_DIV

printf("%d %f %f\n", i / j, i / f, f / d);

}

* + 2\_1\_N\_EXP\_MUL+DIV.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 3

======== code =========

0: INT 0,52

1: SUP 0,3

2: RET 0,0

3: INT 0,28

4: LDA 1,12

5: LITI 0,10

6: STX 0,1

7: POP 0,1

8: LDA 1,16

9: LITI 0,20

10: STX 0,1

11: POP 0,1

12: LDA 1,20

13: LOD 0,12

14: STX 0,1

15: POP 0,1

16: LDA 1,24

17: LOD 0,16

18: STX 0,1

19: POP 0,1

20: INT 0,12

21: LDA 0,20

22: LOD 1,12

23: LOD 1,16

24: MULI 0,0

25: LOD 1,12

26: CVTF 0,0

27: LOD 1,20

28: MULF 0,0

29: LOD 1,20

30: LOD 1,24

31: MULF 0,0

32: POP 0,7

33: ADDR 0,-1

34: CAL 0,0

35: INT 0,12

36: LDA 0,36

37: LOD 1,12

38: LOD 1,16

39: DIVI 0,0

40: LOD 1,12

41: CVTF 0,0

42: LOD 1,20

43: DIVF 0,0

44: LOD 1,20

45: LOD 1,24

46: DIVF 0,0

47: POP 0,7

48: ADDR 0,-1

49: CAL 0,0

50: RET 0,0

======== start execution =========

200 111.000000 246.420013

0 0.900901 0.500000

======== end execution =========

* 11.
  + 2\_2\_N\_EXP\_LSS.c

void main() {

if (10 < 20)

printf("LSS : INT\n");

if (10.1 < 20)

printf("LSS : INT FLOAT\n");

if (10.1 < 20.2)

printf("LSS : FLOAT\n");

if (20 < 10)

printf("LSS : NOT INT\n");

if (20.2 < 10)

printf("LSS : NOT INT FLOAT\n");

if (20.2 < 10.1)

printf("LSS : NOT FLOAT\n");

}

* + 2\_2\_N\_EXP\_LSS.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 3

5: L1 13

6: L2 23

7: L3 32

8: L4 41

9: L5 51

10: L6 60

======== code =========

0: INT 0,152

1: SUP 0,3

2: RET 0,0

3: INT 0,12

4: LITI 0,10

5: LITI 0,20

6: LSSI 0,0

7: JPC 0,13

8: INT 0,12

9: LDA 0,12

10: POP 0,4

11: ADDR 0,-1

12: CAL 0,0

13: LOD 0,28

14: LITI 0,20

15: CVTF 0,0

16: LSSF 0,0

17: JPC 0,23

18: INT 0,12

19: LDA 0,32

20: POP 0,4

21: ADDR 0,-1

22: CAL 0,0

23: LOD 0,52

24: LOD 0,56

25: LSSF 0,0

26: JPC 0,32

27: INT 0,12

28: LDA 0,60

29: POP 0,4

30: ADDR 0,-1

31: CAL 0,0

32: LITI 0,20

33: LITI 0,10

34: LSSI 0,0

35: JPC 0,41

36: INT 0,12

37: LDA 0,76

38: POP 0,4

39: ADDR 0,-1

40: CAL 0,0

41: LOD 0,96

42: LITI 0,10

43: CVTF 0,0

44: LSSF 0,0

45: JPC 0,51

46: INT 0,12

47: LDA 0,100

48: POP 0,4

49: ADDR 0,-1

50: CAL 0,0

51: LOD 0,124

52: LOD 0,128

53: LSSF 0,0

54: JPC 0,60

55: INT 0,12

56: LDA 0,132

57: POP 0,4

58: ADDR 0,-1

59: CAL 0,0

60: RET 0,0

======== start execution =========

LSS : INT

LSS : INT FLOAT

LSS : FLOAT

======== end execution =========

* 12.
  + 2\_3\_N\_EXP\_GTR.c

void main() {

if (20 > 10)

printf("GTR : INT\n");

if (20.2 > 10)

printf("GTR : INT FLOAT\n");

if (20.2 > 10.1)

printf("GTR : FLOAT\n");

if (10 > 20)

printf("GTR : NOT INT\n");

if (10.1 > 20)

printf("GTR : NOT INT FLOAT\n");

if (10.1 > 20.2)

printf("GTR : NOT FLOAT\n");

}

* + 2\_3\_N\_EXP\_GTR.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 3

5: L1 13

6: L2 23

7: L3 32

8: L4 41

9: L5 51

10: L6 60

======== code =========

0: INT 0,152

1: SUP 0,3

2: RET 0,0

3: INT 0,12

4: LITI 0,20

5: LITI 0,10

6: GTRI 0,0

7: JPC 0,13

8: INT 0,12

9: LDA 0,12

10: POP 0,4

11: ADDR 0,-1

12: CAL 0,0

13: LOD 0,28

14: LITI 0,10

15: CVTF 0,0

16: GTRF 0,0

17: JPC 0,23

18: INT 0,12

19: LDA 0,32

20: POP 0,4

21: ADDR 0,-1

22: CAL 0,0

23: LOD 0,52

24: LOD 0,56

25: GTRF 0,0

26: JPC 0,32

27: INT 0,12

28: LDA 0,60

29: POP 0,4

30: ADDR 0,-1

31: CAL 0,0

32: LITI 0,10

33: LITI 0,20

34: GTRI 0,0

35: JPC 0,41

36: INT 0,12

37: LDA 0,76

38: POP 0,4

39: ADDR 0,-1

40: CAL 0,0

41: LOD 0,96

42: LITI 0,20

43: CVTF 0,0

44: GTRF 0,0

45: JPC 0,51

46: INT 0,12

47: LDA 0,100

48: POP 0,4

49: ADDR 0,-1

50: CAL 0,0

51: LOD 0,124

52: LOD 0,128

53: GTRF 0,0

54: JPC 0,60

55: INT 0,12

56: LDA 0,132

57: POP 0,4

58: ADDR 0,-1

59: CAL 0,0

60: RET 0,0

======== start execution =========

GTR : INT

GTR : INT FLOAT

GTR : FLOAT

======== end execution =========

* 13.
  + 2\_4\_N\_EXP\_LEQ.c

void main() {

if (10 <= 20)

printf("LEQ : INT\n");

if (10.1 <= 20)

printf("LEQ : INT FLOAT\n");

if (10.1 <= 20.2)

printf("LEQ : FLOAT\n");

if (20 <= 10)

printf("LEQ : NOT INT\n");

if (20.2 <= 10)

printf("LEQ : NOT INT FLOAT\n");

if (20.2 <= 10.1)

printf("LEQ : NOT FLOAT\n");

}

* + 2\_4\_N\_EXP\_LEQ.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 3

5: L1 13

6: L2 23

7: L3 32

8: L4 41

9: L5 51

10: L6 60

======== code =========

0: INT 0,152

1: SUP 0,3

2: RET 0,0

3: INT 0,12

4: LITI 0,10

5: LITI 0,20

6: LEQI 0,0

7: JPC 0,13

8: INT 0,12

9: LDA 0,12

10: POP 0,4

11: ADDR 0,-1

12: CAL 0,0

13: LOD 0,28

14: LITI 0,20

15: CVTF 0,0

16: LEQF 0,0

17: JPC 0,23

18: INT 0,12

19: LDA 0,32

20: POP 0,4

21: ADDR 0,-1

22: CAL 0,0

23: LOD 0,52

24: LOD 0,56

25: LEQF 0,0

26: JPC 0,32

27: INT 0,12

28: LDA 0,60

29: POP 0,4

30: ADDR 0,-1

31: CAL 0,0

32: LITI 0,20

33: LITI 0,10

34: LEQI 0,0

35: JPC 0,41

36: INT 0,12

37: LDA 0,76

38: POP 0,4

39: ADDR 0,-1

40: CAL 0,0

41: LOD 0,96

42: LITI 0,10

43: CVTF 0,0

44: LEQF 0,0

45: JPC 0,51

46: INT 0,12

47: LDA 0,100

48: POP 0,4

49: ADDR 0,-1

50: CAL 0,0

51: LOD 0,124

52: LOD 0,128

53: LEQF 0,0

54: JPC 0,60

55: INT 0,12

56: LDA 0,132

57: POP 0,4

58: ADDR 0,-1

59: CAL 0,0

60: RET 0,0

======== start execution =========

LEQ : INT

LEQ : INT FLOAT

LEQ : FLOAT

======== end execution =========

* 14.
  + 2\_5\_N\_EXP\_GEQ.c

void main() {

if (20 >= 10)

printf("GEQ : INT\n");

if (20.2 >= 10)

printf("GEQ : INT FLOAT\n");

if (20.2 >= 10.1)

printf("GEQ : FLOAT\n");

if (10 >= 20)

printf("GEQ : NOT INT\n");

if (10.1 >= 20)

printf("GEQ : NOT INT FLOAT\n");

if (10.1 >= 20.2)

printf("GEQ : NOT FLOAT\n");

}

* + 2\_5\_N\_EXP\_GEQ.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 3

5: L1 13

6: L2 23

7: L3 32

8: L4 41

9: L5 51

10: L6 60

======== code =========

0: INT 0,152

1: SUP 0,3

2: RET 0,0

3: INT 0,12

4: LITI 0,20

5: LITI 0,10

6: GEQI 0,0

7: JPC 0,13

8: INT 0,12

9: LDA 0,12

10: POP 0,4

11: ADDR 0,-1

12: CAL 0,0

13: LOD 0,28

14: LITI 0,10

15: CVTF 0,0

16: GEQF 0,0

17: JPC 0,23

18: INT 0,12

19: LDA 0,32

20: POP 0,4

21: ADDR 0,-1

22: CAL 0,0

23: LOD 0,52

24: LOD 0,56

25: GEQF 0,0

26: JPC 0,32

27: INT 0,12

28: LDA 0,60

29: POP 0,4

30: ADDR 0,-1

31: CAL 0,0

32: LITI 0,10

33: LITI 0,20

34: GEQI 0,0

35: JPC 0,41

36: INT 0,12

37: LDA 0,76

38: POP 0,4

39: ADDR 0,-1

40: CAL 0,0

41: LOD 0,96

42: LITI 0,20

43: CVTF 0,0

44: GEQF 0,0

45: JPC 0,51

46: INT 0,12

47: LDA 0,100

48: POP 0,4

49: ADDR 0,-1

50: CAL 0,0

51: LOD 0,124

52: LOD 0,128

53: GEQF 0,0

54: JPC 0,60

55: INT 0,12

56: LDA 0,132

57: POP 0,4

58: ADDR 0,-1

59: CAL 0,0

60: RET 0,0

======== start execution =========

GEQ : INT

GEQ : INT FLOAT

GEQ : FLOAT

======== end execution =========

* 15.
  + 2\_6\_N\_EXP\_NEQ.c

void main()

{

if (10 != 20)

printf("NEQ : INT\n");

if (10.1 != 20)

printf("NEQ : INT FLOAT\n");

if (10.1 != 20.2)

printf("NEQ : FLOAT\n");

if (10 != 10)

printf("NEQ : NOT INT\n");

if (10.1 != 10.1)

printf("NEQ : NOT FLOAT\n");

}

* + 2\_6\_N\_EXP\_NEQ.c.asm 인터프리터 결과

1: printf -1

2: malloc -2

3: scanf -3

4: main 3

5: L1 13

6: L2 23

7: L3 32

8: L4 41

9: L5 50

======== code =========

0: INT 0,124

1: SUP 0,3

2: RET 0,0

3: INT 0,12

4: LITI 0,10

5: LITI 0,20

6: NEQI 0,0

7: JPC 0,13

8: INT 0,12

9: LDA 0,12

10: POP 0,4

11: ADDR 0,-1

12: CAL 0,0

13: LOD 0,28

14: LITI 0,20

15: CVTF 0,0

16: NEQF 0,0

17: JPC 0,23

18: INT 0,12

19: LDA 0,32

20: POP 0,4

21: ADDR 0,-1

22: CAL 0,0

23: LOD 0,52

24: LOD 0,56

25: NEQF 0,0

26: JPC 0,32

27: INT 0,12

28: LDA 0,60

29: POP 0,4

30: ADDR 0,-1

31: CAL 0,0

32: LITI 0,10

33: LITI 0,10

34: NEQI 0,0

35: JPC 0,41

36: INT 0,12

37: LDA 0,76

38: POP 0,4

39: ADDR 0,-1

40: CAL 0,0

41: LOD 0,96

42: LOD 0,100

43: NEQF 0,0

44: JPC 0,50

45: INT 0,12

46: LDA 0,104

47: POP 0,4

48: ADDR 0,-1

49: CAL 0,0

50: RET 0,0

======== start execution =========

NEQ : INT

NEQ : INT FLOAT

NEQ : FLOAT

======== end execution =========

* 16.
  + 2\_7\_N\_EXP\_EQL.c

void main() {

if (10 == 10)

printf("EQL : INT\n");

if (10.1 == 10.1)

printf("EQL : FLOAT\n");

if (20 == 10)

printf("EQL : NOT INT\n");

if (20.2 == 20)

printf("EQL : NOT INT FLOAT\n");

if (20.2 == 20.1)

printf("EQL : NOT FLOAT\n");

}

* + 2\_7\_N\_EXP\_EQL.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 3

5: L1 13

6: L2 22

7: L3 31

8: L4 41

9: L5 50

======== code =========

0: INT 0,128

1: SUP 0,3

2: RET 0,0

3: INT 0,12

4: LITI 0,10

5: LITI 0,10

6: EQLI 0,0

7: JPC 0,13

8: INT 0,12

9: LDA 0,12

10: POP 0,4

11: ADDR 0,-1

12: CAL 0,0

13: LOD 0,28

14: LOD 0,32

15: EQLF 0,0

16: JPC 0,22

17: INT 0,12

18: LDA 0,36

19: POP 0,4

20: ADDR 0,-1

21: CAL 0,0

22: LITI 0,20

23: LITI 0,10

24: EQLI 0,0

25: JPC 0,31

26: INT 0,12

27: LDA 0,52

28: POP 0,4

29: ADDR 0,-1

30: CAL 0,0

31: LOD 0,72

32: LITI 0,20

33: CVTF 0,0

34: EQLF 0,0

35: JPC 0,41

36: INT 0,12

37: LDA 0,76

38: POP 0,4

39: ADDR 0,-1

40: CAL 0,0

41: LOD 0,100

42: LOD 0,104

43: EQLF 0,0

44: JPC 0,50

45: INT 0,12

46: LDA 0,108

47: POP 0,4

48: ADDR 0,-1

49: CAL 0,0

50: RET 0,0

======== start execution =========

EQL : INT

EQL : FLOAT

======== end execution =========

* 17.
  + 2\_8\_N\_EXP\_AND+OR.c

void main() {

int and, or ;

and = 10 && 0;

or = 10 || 0;

printf("%d %d\n", and, or);

}

* + 2\_8\_N\_EXP\_AND+OR.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 3

5: L1 8

6: L2 14

======== code =========

0: INT 0,24

1: SUP 0,3

2: RET 0,0

3: INT 0,20

4: LDA 1,12

5: LITI 0,10

6: JPCR 0,8

7: LITI 0,0

8: STX 0,1

9: POP 0,1

10: LDA 1,16

11: LITI 0,10

12: JPTR 0,14

13: LITI 0,0

14: STX 0,1

15: POP 0,1

16: INT 0,12

17: LDA 0,12

18: LOD 1,12

19: LOD 1,16

20: POP 0,6

21: ADDR 0,-1

22: CAL 0,0

23: RET 0,0

======== start execution =========

0 10

======== end execution =========

* 18.
  + 3\_1\_N\_STMT\_IF+IF\_ELSE.c

void main()

{

int i;

float f;

i = 0;

f = 1.1;

// N\_EXP\_IF

if (i)

printf("IF\n");

// N\_EXP\_IF\_ELSE

if (i)

printf("IF\n");

else if (f)

printf("ELSE\_IF\n");

}

* + 3\_1\_N\_STMT\_IF+IF\_ELSE.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 3

5: L1 19

6: L2 27

7: L3 35

8: L4 35

======== code =========

0: INT 0,44

1: SUP 0,3

2: RET 0,0

3: INT 0,20

4: LDA 1,12

5: LITI 0,0

6: STX 0,1

7: POP 0,1

8: LDA 1,16

9: LOD 0,12

10: STX 0,1

11: POP 0,1

12: LOD 1,12

13: JPC 0,19

14: INT 0,12

15: LDA 0,16

16: POP 0,4

17: ADDR 0,-1

18: CAL 0,0

19: LOD 1,12

20: JPC 0,27

21: INT 0,12

22: LDA 0,24

23: POP 0,4

24: ADDR 0,-1

25: CAL 0,0

26: JMP 0,35

27: LOD 1,16

28: CVTI 0,0

29: JPC 0,35

30: INT 0,12

31: LDA 0,32

32: POP 0,4

33: ADDR 0,-1

34: CAL 0,0

35: RET 0,0

======== start execution =========

ELSE\_IF

======== end execution =========

* 19.
  + 3\_2\_N\_STMT\_WHILE.c

void main() {

int i;

i = 0;

while (i < 10) {

printf("%d ", i++);

}

printf("\n");

}

* + 3\_2\_N\_STMT\_WHILE.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 3

5: L2 8

6: L3 23

7: L1 22

======== code =========

0: INT 0,28

1: SUP 0,3

2: RET 0,0

3: INT 0,16

4: LDA 1,12

5: LITI 0,0

6: STX 0,1

7: POP 0,1

8: LOD 1,12

9: LITI 0,10

10: LSSI 0,0

11: JPC 0,23

12: INT 0,12

13: LDA 0,12

14: LOD 1,12

15: LDA 1,12

16: LDX 0,1

17: INCI 0,0

18: STO 0,1

19: POP 0,5

20: ADDR 0,-1

21: CAL 0,0

22: JMP 0,8

23: INT 0,12

24: LDA 0,20

25: POP 0,4

26: ADDR 0,-1

27: CAL 0,0

28: RET 0,0

======== start execution =========

0 1 2 3 4 5 6 7 8 9

======== end execution =========

* 20.
  + 3\_3\_N\_STMT\_DO.c

void main() {

int i;

i = 0;

do {

printf("%d ", i++);

} while (i < 10);

printf("\n");

do {

printf("%d ", i++);

} while (i < 10);

printf("\n");

}

* + 3\_3\_N\_STMT\_DO.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 3

5: L3 8

6: L2 18

7: L1 22

8: L6 27

9: L5 37

10: L4 41

======== code =========

0: INT 0,44

1: SUP 0,3

2: RET 0,0

3: INT 0,16

4: LDA 1,12

5: LITI 0,0

6: STX 0,1

7: POP 0,1

8: INT 0,12

9: LDA 0,12

10: LOD 1,12

11: LDA 1,12

12: LDX 0,1

13: INCI 0,0

14: STO 0,1

15: POP 0,5

16: ADDR 0,-1

17: CAL 0,0

18: LOD 1,12

19: LITI 0,10

20: LSSI 0,0

21: JPT 0,8

22: INT 0,12

23: LDA 0,20

24: POP 0,4

25: ADDR 0,-1

26: CAL 0,0

27: INT 0,12

28: LDA 0,28

29: LOD 1,12

30: LDA 1,12

31: LDX 0,1

32: INCI 0,0

33: STO 0,1

34: POP 0,5

35: ADDR 0,-1

36: CAL 0,0

37: LOD 1,12

38: LITI 0,10

39: LSSI 0,0

40: JPT 0,27

41: INT 0,12

42: LDA 0,36

43: POP 0,4

44: ADDR 0,-1

45: CAL 0,0

46: RET 0,0

======== start execution =========

0 1 2 3 4 5 6 7 8 9

10

======== end execution =========

* 21.
  + 3\_4\_N\_STMT\_FOR.c

void main()

{

int i;

for (i = 0; i < 100; i++)

{

if (i % 10 == 0 && i != 0)

{

printf("\n");

continue; // N\_STMT\_CONTINUE

}

if (i == 77)

{

printf("\n");

break; // N\_STMT\_BREAK

}

printf("%d ", i);

}

}

* + 3\_4\_N\_STMT\_FOR.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 3

5: L2 8

6: L3 51

7: L4 21

8: L5 28

9: L1 44

10: L6 38

======== code =========

0: INT 0,36

1: SUP 0,3

2: RET 0,0

3: INT 0,16

4: LDA 1,12

5: LITI 0,0

6: STX 0,1

7: POP 0,1

8: LOD 1,12

9: LITI 0,100

10: LSSI 0,0

11: JPC 0,51

12: LOD 1,12

13: LITI 0,10

14: MOD 0,0

15: LITI 0,0

16: EQLI 0,0

17: JPCR 0,21

18: LOD 1,12

19: LITI 0,0

20: NEQI 0,0

21: JPC 0,28

22: INT 0,12

23: LDA 0,12

24: POP 0,4

25: ADDR 0,-1

26: CAL 0,0

27: JMP 0,44

28: LOD 1,12

29: LITI 0,77

30: EQLI 0,0

31: JPC 0,38

32: INT 0,12

33: LDA 0,20

34: POP 0,4

35: ADDR 0,-1

36: CAL 0,0

37: JMP 0,51

38: INT 0,12

39: LDA 0,28

40: LOD 1,12

41: POP 0,5

42: ADDR 0,-1

43: CAL 0,0

44: LOD 1,12

45: LDA 1,12

46: LDX 0,1

47: INCI 0,0

48: STO 0,1

49: POP 0,1

50: JMP 0,8

51: RET 0,0

======== start execution =========

0 1 2 3 4 5 6 7 8 9

11 12 13 14 15 16 17 18 19

21 22 23 24 25 26 27 28 29

31 32 33 34 35 36 37 38 39

41 42 43 44 45 46 47 48 49

51 52 53 54 55 56 57 58 59

61 62 63 64 65 66 67 68 69

71 72 73 74 75 76

======== end execution =========

* 22.
  + 4\_1\_multiply.c

int muitiply(int a, int b) {

int result;

result = 0;

while (a) {

if (a % 2)

result = result + b;

a = a / 2;

b = b \* 2;

}

return result;

}

void main() {

int res;

res = muitiply(10, 22);

printf("%d\n", res);

}

* + 4\_1\_multiply.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 37

5: muitiply 3

6: L2 8

7: L3 33

8: L4 20

9: L1 32

======== code =========

0: INT 0,20

1: SUP 0,37

2: RET 0,0

3: INT 0,24

4: LDA 1,20

5: LITI 0,0

6: STX 0,1

7: POP 0,1

8: LOD 1,12

9: JPC 0,33

10: LOD 1,12

11: LITI 0,2

12: MOD 0,0

13: JPC 0,20

14: LDA 1,20

15: LOD 1,20

16: LOD 1,16

17: ADDI 0,0

18: STX 0,1

19: POP 0,1

20: LDA 1,12

21: LOD 1,12

22: LITI 0,2

23: DIVI 0,0

24: STX 0,1

25: POP 0,1

26: LDA 1,16

27: LOD 1,16

28: LITI 0,2

29: MULI 0,0

30: STX 0,1

31: POP 0,1

32: JMP 0,8

33: LDA 1,-4

34: LOD 1,20

35: STO 0,1

36: RET 0,0

37: INT 0,16

38: LDA 1,12

39: INT 0,16

40: LITI 0,10

41: LITI 0,22

42: POP 0,5

43: ADDR 0,3

44: CAL 0,0

45: STX 0,1

46: POP 0,1

47: INT 0,12

48: LDA 0,12

49: LOD 1,12

50: POP 0,5

51: ADDR 0,-1

52: CAL 0,0

53: RET 0,0

======== start execution =========

220

======== end execution =========

* 23.
  + 4\_2\_gcd.c

int gcd(int x, int y) {

int a, b;

a = x;

b = y;

while (a != b) {

if (a < b) {

b = b - a;

}

if (a > b) {

a = a - b;

}

}

return a;

}

void main() {

int i;

i = gcd(84, 36);

printf("result=%d\n", i);

}

* + 4\_2\_gcd.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 41

5: gcd 3

6: L2 12

7: L3 37

8: L4 26

9: L5 36

10: L1 36

======== code =========

0: INT 0,28

1: SUP 0,41

2: RET 0,0

3: INT 0,28

4: LDA 1,20

5: LOD 1,12

6: STX 0,1

7: POP 0,1

8: LDA 1,24

9: LOD 1,16

10: STX 0,1

11: POP 0,1

12: LOD 1,20

13: LOD 1,24

14: NEQI 0,0

15: JPC 0,37

16: LOD 1,20

17: LOD 1,24

18: LSSI 0,0

19: JPC 0,26

20: LDA 1,24

21: LOD 1,24

22: LOD 1,20

23: SUBI 0,0

24: STX 0,1

25: POP 0,1

26: LOD 1,20

27: LOD 1,24

28: GTRI 0,0

29: JPC 0,36

30: LDA 1,20

31: LOD 1,20

32: LOD 1,24

33: SUBI 0,0

34: STX 0,1

35: POP 0,1

36: JMP 0,12

37: LDA 1,-4

38: LOD 1,20

39: STO 0,1

40: RET 0,0

41: INT 0,16

42: LDA 1,12

43: INT 0,16

44: LITI 0,84

45: LITI 0,36

46: POP 0,5

47: ADDR 0,3

48: CAL 0,0

49: STX 0,1

50: POP 0,1

51: INT 0,12

52: LDA 0,12

53: LOD 1,12

54: POP 0,5

55: ADDR 0,-1

56: CAL 0,0

57: RET 0,0

======== start execution =========

result=12

======== end execution =========

* 24.
  + 4\_3\_iteration.c

main() {

int n, x, i, k, lim, prim;

int p[300];

p[1] = 2;

printf("%d ", p[1]);

n = 100;

x = 1;

lim = 1;

for (i = 2; i <= n; i++) {

do {

x = x + 1;

x++;

if (p[lim] \* p[lim] <= x) {

lim++;

}

k = 2;

prim = 1;

while (prim && (k < lim)) {

prim = x % p[k];

k++;

}

} while (!prim);

p[i] = x;

printf("%d ", x);

if (i % 10 == 0) {

printf("\n");

}

}

}

* + 4\_3\_iteration.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 3

5: L2 39

6: L3 144

7: L6 43

8: L7 77

9: L9 85

10: L10 90

11: L11 109

12: L8 108

13: L5 109

14: L4 112

15: L12 137

16: L1 137

======== code =========

0: INT 0,36

1: SUP 0,3

2: RET 0,0

3: INT 0,1236

4: LDA 1,36

5: LITI 0,1

6: LITI 0,4

7: MULI 0,0

8: OFFSET 0,0

9: LITI 0,2

10: STX 0,1

11: POP 0,1

12: INT 0,12

13: LDA 0,12

14: LDA 1,36

15: LITI 0,1

16: LITI 0,4

17: MULI 0,0

18: OFFSET 0,0

19: LDI 0,1

20: POP 0,5

21: ADDR 0,-1

22: CAL 0,0

23: LDA 1,12

24: LITI 0,100

25: STX 0,1

26: POP 0,1

27: LDA 1,16

28: LITI 0,1

29: STX 0,1

30: POP 0,1

31: LDA 1,28

32: LITI 0,1

33: STX 0,1

34: POP 0,1

35: LDA 1,20

36: LITI 0,2

37: STX 0,1

38: POP 0,1

39: LOD 1,20

40: LOD 1,12

41: LEQI 0,0

42: JPC 0,144

43: LDA 1,16

44: LOD 1,16

45: LITI 0,1

46: ADDI 0,0

47: STX 0,1

48: POP 0,1

49: LOD 1,16

50: LDA 1,16

51: LDX 0,1

52: INCI 0,0

53: STO 0,1

54: POP 0,1

55: LDA 1,36

56: LOD 1,28

57: LITI 0,4

58: MULI 0,0

59: OFFSET 0,0

60: LDI 0,1

61: LDA 1,36

62: LOD 1,28

63: LITI 0,4

64: MULI 0,0

65: OFFSET 0,0

66: LDI 0,1

67: MULI 0,0

68: LOD 1,16

69: LEQI 0,0

70: JPC 0,77

71: LOD 1,28

72: LDA 1,28

73: LDX 0,1

74: INCI 0,0

75: STO 0,1

76: POP 0,1

77: LDA 1,24

78: LITI 0,2

79: STX 0,1

80: POP 0,1

81: LDA 1,32

82: LITI 0,1

83: STX 0,1

84: POP 0,1

85: LOD 1,32

86: JPCR 0,90

87: LOD 1,24

88: LOD 1,28

89: LSSI 0,0

90: JPC 0,109

91: LDA 1,32

92: LOD 1,16

93: LDA 1,36

94: LOD 1,24

95: LITI 0,4

96: MULI 0,0

97: OFFSET 0,0

98: LDI 0,1

99: MOD 0,0

100: STX 0,1

101: POP 0,1

102: LOD 1,24

103: LDA 1,24

104: LDX 0,1

105: INCI 0,0

106: STO 0,1

107: POP 0,1

108: JMP 0,85

109: LOD 1,32

110: NOT 0,0

111: JPT 0,43

112: LDA 1,36

113: LOD 1,20

114: LITI 0,4

115: MULI 0,0

116: OFFSET 0,0

117: LOD 1,16

118: STX 0,1

119: POP 0,1

120: INT 0,12

121: LDA 0,20

122: LOD 1,16

123: POP 0,5

124: ADDR 0,-1

125: CAL 0,0

126: LOD 1,20

127: LITI 0,10

128: MOD 0,0

129: LITI 0,0

130: EQLI 0,0

131: JPC 0,137

132: INT 0,12

133: LDA 0,28

134: POP 0,4

135: ADDR 0,-1

136: CAL 0,0

137: LOD 1,20

138: LDA 1,20

139: LDX 0,1

140: INCI 0,0

141: STO 0,1

142: POP 0,1

143: JMP 0,39

144: RET 0,0

======== start execution =========

2 3 5 7 11 13 17 19 23 29

31 37 41 43 47 53 59 61 67 71

73 79 83 89 97 101 103 107 109 113

127 131 137 139 149 151 157 163 167 173

179 181 191 193 197 199 211 223 227 229

233 239 241 251 257 263 269 271 277 281

283 293 307 311 313 317 331 337 347 349

353 359 367 373 379 383 389 397 401 409

419 421 431 433 439 443 449 457 461 463

467 479 487 491 499 503 509 521 523 541

======== end execution =========

* 25.
  + 4\_4\_strcmp.c

int strcmp(char \*s, char \*t) {

for (; \*s == \*t; s++) {

if (\*s == 0)

return 0;

t++;

}

return \*s - \*t;

}

void main() {

int t, f;

t = strcmp("hello", "hello");

f = strcmp("hello", "hi");

printf("%d %d\n", t, f);

}

* + 4\_4\_strcmp.c.asm 인터프리터 결과

======== symbol =========

1: printf -1

2: malloc -2

3: scanf -3

4: main 42

5: strcmp 3

6: L2 4

7: L3 34

8: L4 19

9: L1 26

======== code =========

0: INT 0,56

1: SUP 0,42

2: RET 0,0

3: INT 0,20

4: LOD 1,12

5: LDIB 0,0

6: LOD 1,16

7: LDIB 0,0

8: EQLI 0,0

9: JPC 0,34

10: LOD 1,12

11: LDIB 0,0

12: LITI 0,0

13: EQLI 0,0

14: JPC 0,19

15: LDA 1,-4

16: LITI 0,0

17: STO 0,1

18: RET 0,0

19: LOD 1,16

20: LDA 1,16

21: LDX 0,1

22: LITI 0,1

23: ADDI 0,0

24: STO 0,1

25: POP 0,1

26: LOD 1,12

27: LDA 1,12

28: LDX 0,1

29: LITI 0,1

30: ADDI 0,0

31: STO 0,1

32: POP 0,1

33: JMP 0,4

34: LDA 1,-4

35: LOD 1,12

36: LDIB 0,0

37: LOD 1,16

38: LDIB 0,0

39: SUBI 0,0

40: STO 0,1

41: RET 0,0

42: INT 0,20

43: LDA 1,12

44: INT 0,16

45: LDA 0,12

46: LDA 0,20

47: POP 0,5

48: ADDR 0,3

49: CAL 0,0

50: STX 0,1

51: POP 0,1

52: LDA 1,16

53: INT 0,16

54: LDA 0,28

55: LDA 0,36

56: POP 0,5

57: ADDR 0,3

58: CAL 0,0

59: STX 0,1

60: POP 0,1

61: INT 0,12

62: LDA 0,44

63: LOD 1,12

64: LOD 1,16

65: POP 0,6

66: ADDR 0,-1

67: CAL 0,0

68: RET 0,0

======== start execution =========

0 -4

======== end execution =========

5. 결론

* 지난 과제를 토대로 어셈블리어 코드가 생성되는 것을 확인하고 이를 인터프리터로 실행하였다.

6. 문제점

* gcc 컴파일러에 비해, compile warning 메시지를 더 구체적으로 수정할 필요가 있다.

7. 수정사항

**main.c**

* FILE \*fout; // 파일 포인터 추가
* 그 외 부록을 활용하여 code generation 코드를 추가하였음

**gen.c**

* int is\_return = 0; 을 추가함

// 함수에서 return을 명시하면 .asm파일에서 RET이 두번 되므로 한번만 RET되게 사용하는 변수.

* **struct s{struct s\* ptr;}a; a.ptr; 같은 경우 허용되어야 한다.**

**if(!isPointerOrArrayType(id->type))**

**gen\_error(24,node->line,"N\_EXP\_STRUCT");**

* macOS에서 발생하는 implict function declaration 컴파일 에러 방지위해 추가

int isArrayType();

int isPointerOrArrayType();

int isFloatType();

**Semantic.c**

* float atof() 🡪 **double** atof() // literal table값이 제대로 출력되지 않아 변경함
* **N\_EXP\_STRUCT에서 i.e. struct s{struct s b;}a; a.b;시 에러가 나게끔 변경함**

**if (isStructOrUnionType(result)) // 추가된 부분**

**semantic\_error(37, node->line)**

**interp.l**

* extern YYSTYPE yylval; 을 추가함. // 선언되어 있지 않아 생기는 에러 방지

**interp.y**

* double atof(); //float에서 double로 바꿔야 함수 충돌 안남

8. 기타

* Ubuntu Linux 22.04.2 LTS, macOS Ventura 13.4에서 동작가능하다. (23/06/17)

9. 원시 프로그램 (소스코드)

main.c

#include "type.h"

#include "y.tab.h"

#include <stdio.h>

#include <stdlib.h>

// 터미널에서 printf()문을 색 구별하여 출력하게 끔 하기 위함

#define C\_NRML "\033[0m"

#define C\_BLCK "\033[30m"

#define C\_RED "\033[31m"

#define C\_GREN "\033[32m"

#define C\_AQUA "\033[36m"

extern int syntax\_err;

extern A\_NODE \*root;

void initialize();

void print\_ast();

int yyparse();

// 부록9를 참고하여 semantic 코드 추가

extern int semantic\_err;

void print\_sem\_ast();

void semantic\_analysis();

// 부록9를 참고하여 generation 코드 추가

extern int gen\_err; // gen.c

void code\_generation();

FILE \*fout;

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

initialize();

// printf("%s신택스 분석 시작\n", C\_AQUA);

// syntax analysis

yyparse(); // from y.tab.c

if (syntax\_err) {

printf("%s신택스 에러로 인해 종료%s", C\_RED, C\_NRML);

exit(1);

} else {

// printf("%s신택스 분석 완료. 트리가 출력됩니다.\n%s", C\_GREN, C\_BLCK);

// print\_ast(root);

}

// 부록9를 참고하여 semantic 코드 추가

// semantic analysis

// printf("%s시맨틱 분석 시작\n", C\_AQUA);

semantic\_analysis(root);

if (semantic\_err) {

printf("%s시맨틱 에러로 인해 종료%s", C\_RED, C\_NRML);

exit(1);

} else {

// printf("%s시맨틱 분석 완료. 트리가 출력됩니다.\n%s", C\_GREN, C\_BLCK);

// print\_sem\_ast(root);

}

printf("%s", C\_NRML);

// generation 코드 추가

if ((fout = fopen("a.asm", "w+")) == NULL) {

printf("can not open output file: a.asm\n");

exit(1);

}

// printf("%s코드 생성 시작\n", C\_AQUA);

code\_generation(root);

if (gen\_err) {

printf("%s코드 생성 에러로 인해 종료%s", C\_RED, C\_NRML);

exit(1);

} else {

// printf("%s코드 생성 완료\n%s", C\_GREN, C\_BLCK);

}

printf("%s", C\_NRML);

}

gen.c

#include "type.h"

#include <stdio.h>

#include <string.h>

typedef enum op {

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

} OPCODE;

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();

// macOS에서 발생하는 implict func declaration 컴파일 에러 방지

int isArrayType();

int isPointerOrArrayType();

int isFloatType();

int get\_label();

int label\_no = 0;

int gen\_err = 0;

// 함수에서 return을 명시하면 .asm에서 RET이 두번 되므로 한번만 RET되게 사용하는

// 변수.

int is\_return = 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;

default:

gen\_error(100, node->line);

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"); // 구현 범위 제외

break;

case T\_UNION:

gen\_error(24, node->line, "T\_UNION"); // 구현 범위 제외

// gen\_error(24, node->line, "T\_STRUCT");

// 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");

// STRUCT은 구현 범위 제외이므로 error 처리 하였다.

// 단, struct s{struct s\* ptr;}a; a.ptr; 같은 경우,

// if(!isPointerOrArrayType(id->type))

// gen\_error(24,node->line,"N\_EXP\_STRUCT");

// 처럼 코드를 작성하여 코드 생성을 유도할 수 있다.

// 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);

// }

// 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\_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:

gen\_code\_i(LDA, id->level, id->address);

break;

case T\_STRUCT:

gen\_error(24, node->line, "T\_STRUCT"); // 구현 범위 제외

break;

case T\_UNION:

gen\_error(24, node->line, "T\_UNION"); // 구현 범위 제외

// 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\_error(24, node->line, "N\_EXP\_STRUCT");

// 구현 범위 제외

// 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\_SWITCH switch\_table[100];

int switch\_no = 0;

A\_NODE \*n;

int i, l1, l2, l3;

switch (node->name) {

case N\_STMT\_LABEL\_CASE:

gen\_error(24, node->line, "N\_STMT\_LABEL\_CASE"); // 구현 범위 제외

// if (sw) {

// \*sn=\*sn+1;

// sw[\*sn].kind=SW\_VALUE;

// sw[\*sn].val=node->llink;

// sw[\*sn].label=l1=get\_label();

// gen\_label\_number(l1);

// }

// else

// gen\_error(21,node->line);

// gen\_statement(node->rlink,cont\_label,break\_label,sw,sn);

break;

case N\_STMT\_LABEL\_DEFAULT:

gen\_error(24, node->line, "N\_STMT\_LABEL\_DEFAULT"); // 구현 범위 제외

// if (sw) {

// \*sn=\*sn+1;

// sw[\*sn].kind=SW\_DEFAULT;

// sw[\*sn].label=l1=get\_label();

// gen\_label\_number(l1);

// }

// else

// gen\_error(20,node->line);

// gen\_statement(node->clink,cont\_label,break\_label,sw,sn);

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:

gen\_error(24, node->line, "N\_STMT\_SWITCH"); // 구현 범위 제외

// gen\_expression(node->llink);

// gen\_code\_l(SWITCH, 0,l1=get\_label());

// gen\_code\_l(JMP,0,l2=get\_label());

// gen\_statement(node->rlink,cont\_label,l2,switch\_table,&switch\_no);

// gen\_label\_number(l1);

// for (i=1;i<=switch\_no;i++) {

// if (switch\_table[i].kind==SW\_VALUE)

// gen\_code\_i(SWVALUE,0,switch\_table[i].val);

// else

// gen\_code\_i(SWDEFAULT,0,0);

// gen\_code\_l(SWLABEL,0,switch\_table[i].label);

// }

// gen\_code\_i(SWEND,0,0);

// gen\_label\_number(l2);

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\_DO:

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\_l(JMP, 0, cont\_label);

else

gen\_error(22, node->line);

break;

case N\_STMT\_BREAK:

if (break\_label)

gen\_code\_l(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);

is\_return = 1; // 리턴 여부 추가

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) {}

void gen\_initializer\_local(A\_NODE \*node, A\_TYPE \*t, int addr) {}

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);

// 이미 return을 명시한 상태면 RET이 두번 되기 때문에 이를 조정한다.

if (!is\_return)

gen\_code\_i(RET, 0, 0);

else

is\_return = 0;

}

break;

case ID\_PARM:

case ID\_TYPE:

case ID\_ENUM:

break;

case ID\_STRUCT:

gen\_error(24, node->line, "ID\_STRUCT"); // 구현 범위 제외

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); }

semantic.c

// 부록6을 참고하였음.

#include "type.h"

#include <stdio.h> // printf()

#include <string.h> // strlen(), strcmp()

// 터미널에서 printf()문을 색 구별하여 출력하게 끔 하기 위함

#define C\_YLLW "\033[33m"

#define C\_RED "\033[31m"

double atof(); // literal table에서 float 값이 제대로 출력되지 않아 double로 변경

// void \*sem\_initializer(A\_NODE \*node);

void semantic\_analysis(A\_NODE \*);

void set\_literal\_address(A\_NODE \*);

int put\_literal(A\_LITERAL, int);

void sem\_program(A\_NODE \*);

A\_TYPE \*sem\_expression(A\_NODE \*);

int sem\_statement(A\_NODE \*, int, A\_TYPE \*, BOOLEAN, BOOLEAN, BOOLEAN);

int sem\_statement\_list(A\_NODE \*, int, A\_TYPE \*, BOOLEAN, BOOLEAN, BOOLEAN);

void sem\_for\_expression(A\_NODE \*);

int sem\_A\_TYPE(A\_TYPE \*);

int sem\_declaration\_list(A\_ID \*id, int addr);

int sem\_declaration(A\_ID \*, int);

void sem\_arg\_expr\_list(A\_NODE \*, A\_ID \*);

A\_ID \*getStructFieldIdentifier(A\_TYPE \*, char \*);

A\_ID \*getPointerFieldIdentifier(A\_TYPE \*, char \*);

A\_NODE \*convertScalarToInteger(A\_NODE \*);

A\_NODE \*convertUsualAssignmentConversion(A\_TYPE \*, A\_NODE \*);

A\_NODE \*convertUsualUnaryConversion(A\_NODE \*);

A\_TYPE \*convertUsualBinaryConversion(A\_NODE \*);

A\_NODE \*convertCastingConversion(A\_NODE \*, A\_TYPE \*);

BOOLEAN isAllowableAssignmentConversion(A\_TYPE \*, A\_TYPE \*, A\_NODE \*);

BOOLEAN isAllowableCastingConversion(A\_TYPE \*, A\_TYPE \*);

BOOLEAN isModifiableLvalue(A\_NODE \*);

BOOLEAN isConstantZeroExp(A\_NODE \*);

BOOLEAN isSameParameterType(A\_ID \*, A\_ID \*);

BOOLEAN isNotSameType(A\_TYPE \*, A\_TYPE \*);

BOOLEAN isCompatibleType(A\_TYPE \*, A\_TYPE \*);

BOOLEAN isCompatiblePointerType(A\_TYPE \*, A\_TYPE \*);

BOOLEAN isIntType(A\_TYPE \*);

BOOLEAN isFloatType(A\_TYPE \*);

BOOLEAN isArithmeticType(A\_TYPE \*);

BOOLEAN isAnyIntegerType(A\_TYPE \*);

BOOLEAN isIntegralType(A\_TYPE \*);

BOOLEAN isStructOrUnionType(A\_TYPE \*);

BOOLEAN isFunctionType(A\_TYPE \*);

BOOLEAN isScalarType(A\_TYPE \*);

BOOLEAN isPointerType(A\_TYPE \*);

BOOLEAN isPointerOrArrayType\_sem(A\_TYPE \*);

// func.h에서 사용하는 함수와 이름이 중복되어 link가 안되므로 접미사로 \_sem을

// 붙여 구분함

BOOLEAN isArrayType(A\_TYPE \*);

BOOLEAN isStringType(A\_TYPE \*);

BOOLEAN isVoidType(A\_TYPE \*);

A\_LITERAL checkTypeAndConvertLiteral(A\_LITERAL, A\_TYPE \*, int);

A\_LITERAL getTypeAndValueOfExpression(A\_NODE \*);

A\_TYPE \*setTypeElementType(A\_TYPE \*, A\_TYPE \*);

A\_TYPE \*makeType(T\_KIND);

void setTypeSize(A\_TYPE \*, int);

void semantic\_warning(int, int);

void semantic\_error();

A\_NODE \*makeNode(NODE\_NAME, A\_NODE \*, A\_NODE \*, A\_NODE \*);

extern A\_TYPE \*int\_type, \*float\_type, \*char\_type, \*string\_type, \*void\_type;

int global\_address = 12;

int semantic\_err = 0;

#define LIT\_MAX 100

A\_LITERAL literal\_table[LIT\_MAX];

int literal\_no = 0;

int literal\_size = 0;

void semantic\_analysis(A\_NODE \*node)

{

sem\_program(node);

set\_literal\_address(node);

}

void set\_literal\_address(A\_NODE \*node)

{

int i;

for (i = 1; i <= literal\_no; i++)

literal\_table[i].addr += node->value;

node->value += literal\_size;

}

void sem\_program(A\_NODE \*node)

{

int i;

switch (node->name)

{

case N\_PROGRAM:

i = sem\_declaration\_list(node->clink, 12); // first parm addr = 12

node->value = global\_address;

break;

default:

semantic\_error(90, node->line);

break;

}

}

int put\_literal(A\_LITERAL lit, int ll)

{

float ff;

if (literal\_no >= LIT\_MAX)

semantic\_error(93, ll);

else

literal\_no++;

literal\_table[literal\_no] = lit;

literal\_table[literal\_no].addr = literal\_size;

if (lit.type->kind == T\_ENUM)

literal\_size += 4;

else if (isStringType(lit.type))

literal\_size += strlen(lit.value.s) + 1;

if (literal\_size % 4)

literal\_size = literal\_size / 4 \* 4 + 4;

return (literal\_no);

}

A\_TYPE \*sem\_expression(A\_NODE \*node)

{

A\_TYPE \*result = NIL, \*t, \*t1, \*t2;

A\_ID \*id;

A\_LITERAL lit;

int i;

BOOLEAN lvalue = FALSE;

switch (node->name)

{

case N\_EXP\_IDENT:

id = node->clink;

switch (id->kind)

{

case ID\_VAR:

case ID\_PARM:

result = id->type;

if (!isArrayType(result))

lvalue = TRUE;

break;

case ID\_FUNC:

result = id->type;

break;

case ID\_ENUM\_LITERAL:

result = int\_type;

break;

default:

semantic\_error(38, node->line, id->name);

break;

}

break;

case N\_EXP\_INT\_CONST:

result = int\_type;

break;

case N\_EXP\_FLOAT\_CONST:

lit.type = float\_type;

lit.value.f = atof(node->clink);

node->clink = put\_literal(lit, node->line); // index of literal table

result = float\_type;

break;

case N\_EXP\_CHAR\_CONST:

result = char\_type;

break;

case N\_EXP\_STRING\_LITERAL:

lit.type = string\_type;

lit.value.s = node->clink;

node->clink = put\_literal(lit, node->line); // index of literal table

result = string\_type;

break;

case N\_EXP\_ARRAY:

t1 = sem\_expression(node->llink);

t2 = sem\_expression(node->rlink);

// usual unary conversion

t = convertUsualBinaryConversion(node);

t1 = node->llink->type;

t2 = node->rlink->type;

if (isPointerOrArrayType\_sem(t1))

result = t1->element\_type;

else

semantic\_error(32, node->line);

if (!isIntegralType(t2))

semantic\_error(29, node->line);

if (!isArrayType(result))

lvalue = TRUE;

break;

case N\_EXP\_STRUCT:

t = sem\_expression(node->llink);

id = getStructFieldIdentifier(t, node->rlink);

if (id)

{

result = id->type;

printf(result);

// i.e. struct s{struct s b;}a; a.b;시 에러가 나게 된다.

if (isStructOrUnionType(result)) // 추가된 부분

semantic\_error(37, node->line);

if (node->llink->value && !isArrayType(result))

lvalue = TRUE;

}

else

semantic\_error(37, node->line);

node->rlink = id;

break;

case N\_EXP\_ARROW:

t = sem\_expression(node->llink);

id = getPointerFieldIdentifier(t, node->rlink);

if (id)

{

result = id->type;

if (!isArrayType(result))

lvalue = TRUE;

}

else

semantic\_error(37, node->line);

node->rlink = id;

break;

case N\_EXP\_FUNCTION\_CALL:

t = sem\_expression(node->llink);

// usual unary conversion

node->llink = convertUsualUnaryConversion(node->llink);

t = node->llink->type;

if (isPointerType(t) && isFunctionType(t->element\_type))

{

sem\_arg\_expr\_list(node->rlink, t->element\_type->field);

result = t->element\_type->element\_type;

}

else

semantic\_error(21, node->line);

break;

case N\_EXP\_POST\_INC:

case N\_EXP\_POST\_DEC:

result = sem\_expression(node->clink);

// usual binary conversion between the expression and 1

if (!isScalarType(result))

semantic\_error(27, node->line);

// check if modifiable lvalue

if (!isModifiableLvalue(node->clink))

semantic\_error(60, node->line); //

break;

case N\_EXP\_CAST:

result = node->llink;

i = sem\_A\_TYPE(result);

t = sem\_expression(node->rlink);

// check allowable casting conversion

if (!isAllowableCastingConversion(result, t))

semantic\_error(58, node->line);

break;

case N\_EXP\_SIZE\_TYPE:

t = node->clink;

i = sem\_A\_TYPE(t);

// check if incomplete array, function, void

if (isArrayType(t) && t->size == 0 || isFunctionType(t) || isVoidType(t))

semantic\_error(39, node->line);

else

node->clink = i;

result = int\_type;

break;

case N\_EXP\_SIZE\_EXP:

t = sem\_expression(node->clink);

// check if incomplete array, function

if ((node->clink->name != N\_EXP\_IDENT ||

((A\_ID \*)node->clink->clink)->kind != ID\_PARM) &&

(isArrayType(t) && t->size == 0 || isFunctionType(t)))

semantic\_error(39, node->line);

else

node->clink = t->size;

result = int\_type;

break;

case N\_EXP\_PLUS:

case N\_EXP\_MINUS:

t = sem\_expression(node->clink);

if (isArithmeticType(t))

{

node->clink = convertUsualUnaryConversion(node->clink);

result = node->clink->type;

}

else

semantic\_error(13, node->line);

break;

case N\_EXP\_NOT:

t = sem\_expression(node->clink);

if (isScalarType(t))

{

node->clink = convertUsualUnaryConversion(node->clink);

// 부록6 : result = node->clink->type;

// 7강 슬라이드

result = int\_type;

}

else

semantic\_error(27, node->line);

break;

case N\_EXP\_AMP:

t = sem\_expression(node->clink);

if (node->clink->value == TRUE || isFunctionType(t))

{

result = setTypeElementType(makeType(T\_POINTER), t);

result->size = 4;

}

else

semantic\_error(60, node->line);

break;

case N\_EXP\_STAR:

t = sem\_expression(node->clink);

node->clink = convertUsualUnaryConversion(node->clink);

if (isPointerType(t))

{

result = t->element\_type;

// lvalue if points to an object

if (isStructOrUnionType(result) || isScalarType(result))

lvalue = TRUE;

}

else

semantic\_error(31, node->line);

break;

case N\_EXP\_PRE\_INC:

case N\_EXP\_PRE\_DEC:

result = sem\_expression(node->clink);

// usual binary conversion between the expression and 1

if (!isScalarType(result))

semantic\_error(27, node->line);

// check if modifiable lvalue

if (!isModifiableLvalue(node->clink))

semantic\_error(60, node->line);

break;

case N\_EXP\_MUL:

case N\_EXP\_DIV:

t1 = sem\_expression(node->llink);

t2 = sem\_expression(node->rlink);

if (isArithmeticType(t1) && isArithmeticType(t2))

result = convertUsualBinaryConversion(node);

else

semantic\_error(28, node->line);

break;

case N\_EXP\_MOD:

t1 = sem\_expression(node->llink);

t2 = sem\_expression(node->rlink);

if (isIntegralType(t1) && isIntegralType(t2))

result = convertUsualBinaryConversion(node);

else

semantic\_error(29, node->line);

result = int\_type;

break;

case N\_EXP\_ADD:

t1 = sem\_expression(node->llink);

t2 = sem\_expression(node->rlink);

if (isArithmeticType(t1) && isArithmeticType(t2))

result = convertUsualBinaryConversion(node);

else if (isPointerType(t1) && isIntegralType(t2))

result = t1;

else if (isIntegralType(t1) && isPointerType(t2))

result = t2;

else

semantic\_error(24, node->line);

break;

case N\_EXP\_SUB:

t1 = sem\_expression(node->llink);

t2 = sem\_expression(node->rlink);

if (isArithmeticType(t1) && isArithmeticType(t2))

result = convertUsualBinaryConversion(node);

else if (isPointerType(t1) && isIntegralType(t2))

result = t1;

else if (isCompatiblePointerType(t1, t2))

result = t1;

else

semantic\_error(24, node->line);

break;

case N\_EXP\_LSS:

case N\_EXP\_GTR:

case N\_EXP\_LEQ:

case N\_EXP\_GEQ:

t1 = sem\_expression(node->llink);

t2 = sem\_expression(node->rlink);

if (isArithmeticType(t1) && isArithmeticType(t2))

result = convertUsualBinaryConversion(node);

else if (!isCompatiblePointerType(t1, t2))

semantic\_error(40, node->line);

result = int\_type;

break;

case N\_EXP\_NEQ:

case N\_EXP\_EQL:

t1 = sem\_expression(node->llink);

t2 = sem\_expression(node->rlink);

if (isArithmeticType(t1) && isArithmeticType(t2))

result = convertUsualBinaryConversion(node);

else if (!isCompatiblePointerType(t1, t2) &&

(!isPointerType(t1) || isConstantZeroExp(node->rlink)) &&

(!isPointerType(t2) || isConstantZeroExp(node->rlink)))

semantic\_error(40, node->line);

result = int\_type;

break;

case N\_EXP\_AND:

case N\_EXP\_OR:

t = sem\_expression(node->llink);

if (isScalarType(t))

node->llink = convertUsualUnaryConversion(node->llink);

else

semantic\_error(27, node->line);

t = sem\_expression(node->rlink);

if (isScalarType(t))

node->rlink = convertUsualUnaryConversion(node->rlink);

else

semantic\_error(27, node->line);

result = int\_type;

break;

case N\_EXP\_ASSIGN:

result = sem\_expression(node->llink);

// check modifiable lvalue

if (!isModifiableLvalue(node->llink))

semantic\_error(60, node->line);

t = sem\_expression(node->rlink);

// 부록6

// if (isAllowableAssignmentConversion(result, t, node->rlink))

// 7강 슬라이드에는 node->rlink가 아닌 node로 되어있음

if (isAllowableAssignmentConversion(result, t, node))

{

if (isArithmeticType(result) && isArithmeticType(t))

node->rlink = convertUsualAssignmentConversion(result, node->rlink);

}

else

semantic\_error(58, node->line);

break;

default:

semantic\_error(90, node->line);

break;

}

node->type = result;

node->value = lvalue;

return (result);

}

// check argument-expression-list in function call expression

void sem\_arg\_expr\_list(A\_NODE \*node, A\_ID \*id)

{

A\_TYPE \*t;

A\_ID \*a;

int arg\_size = 0;

switch (node->name)

{

case N\_ARG\_LIST:

if (id == 0)

semantic\_error(34, node->line);

else

{

if (id->type)

{

t = sem\_expression(node->llink);

node->llink = convertUsualUnaryConversion(node->llink);

if (isAllowableCastingConversion(id->type, node->llink->type))

node->llink = convertCastingConversion(node->llink, id->type);

else

semantic\_error(59, node->line);

sem\_arg\_expr\_list(node->rlink, id->link);

}

else

{ // DOTDOT parameter : no conversion

t = sem\_expression(node->llink);

sem\_arg\_expr\_list(node->rlink, id);

}

arg\_size = node->llink->type->size + node->rlink->value;

}

break;

case N\_ARG\_LIST\_NIL:

if (id && id->type) // check if '...' argument

semantic\_error(35, node->line);

break;

default:

semantic\_error(90, node->line);

break;

}

if (arg\_size % 4)

arg\_size = arg\_size / 4 \* 4 + 4;

node->value = arg\_size;

}

BOOLEAN isModifiableLvalue(A\_NODE \*node)

{

if (node->value == FALSE || isFunctionType(node->type))

return (FALSE);

else

return (TRUE);

}

// check statement and return local variable size

int sem\_statement(A\_NODE \*node, int addr, A\_TYPE \*ret, BOOLEAN sw, BOOLEAN brk,

BOOLEAN cnt)

{

int local\_size = 0, i;

A\_LITERAL lit;

A\_TYPE \*t;

switch (node->name)

{

case N\_STMT\_LABEL\_CASE:

if (sw == FALSE)

semantic\_error(71, node->line);

lit = getTypeAndValueOfExpression(node->llink);

if (isIntegralType(lit.type))

node->llink = lit.value.i;

else

semantic\_error(51, node->line);

local\_size = sem\_statement(node->rlink, addr, ret, sw, brk, cnt);

break;

case N\_STMT\_LABEL\_DEFAULT:

if (sw == FALSE)

semantic\_error(72, node->line);

local\_size = sem\_statement(node->clink, addr, ret, sw, brk, cnt);

break;

case N\_STMT\_COMPOUND:

if (node->llink)

local\_size = sem\_declaration\_list(node->llink, addr);

local\_size +=

sem\_statement\_list(node->rlink, local\_size + addr, ret, sw, brk, cnt);

break;

case N\_STMT\_EMPTY:

break;

case N\_STMT\_EXPRESSION:

t = sem\_expression(node->clink);

break;

case N\_STMT\_IF:

t = sem\_expression(node->llink);

if (isScalarType(t))

node->llink = convertScalarToInteger(node->llink);

else

semantic\_error(50, node->line);

local\_size = sem\_statement(node->rlink, addr, ret, FALSE, brk, cnt);

break;

case N\_STMT\_IF\_ELSE:

t = sem\_expression(node->llink);

if (isScalarType(t))

node->llink = convertScalarToInteger(node->llink);

else

semantic\_error(50, node->line);

local\_size = sem\_statement(node->clink, addr, ret, FALSE, brk, cnt);

i = sem\_statement(node->rlink, addr, ret, FALSE, brk, cnt);

if (local\_size < i)

local\_size = i;

break;

case N\_STMT\_SWITCH:

t = sem\_expression(node->llink);

if (!isIntegralType(t))

semantic\_error(50, node->line);

local\_size = sem\_statement(node->rlink, addr, ret, TRUE, TRUE, cnt);

case N\_STMT\_WHILE:

t = sem\_expression(node->llink);

if (isScalarType(t))

node->llink = convertScalarToInteger(node->llink);

else

semantic\_error(50, node->line);

local\_size = sem\_statement(node->rlink, addr, ret, FALSE, TRUE, TRUE);

break;

case N\_STMT\_DO:

local\_size = sem\_statement(node->llink, addr, ret, FALSE, TRUE, TRUE);

t = sem\_expression(node->rlink);

if (isScalarType(t))

node->rlink = convertScalarToInteger(node->rlink);

else

semantic\_error(50, node->line);

break;

case N\_STMT\_FOR:

sem\_for\_expression(node->llink);

local\_size = sem\_statement(node->rlink, addr, ret, FALSE, TRUE, TRUE);

break;

case N\_STMT\_CONTINUE:

if (cnt == FALSE)

semantic\_error(74, node->line);

break;

case N\_STMT\_BREAK:

if (brk == FALSE)

semantic\_error(73, node->line);

break;

case N\_STMT\_RETURN:

if (node->clink)

{

t = sem\_expression(node->clink);

if (isAllowableCastingConversion(ret, t))

node->clink = convertCastingConversion(node->clink, ret);

else

semantic\_error(57, node->line);

}

break;

default:

semantic\_error(90, node->line);

break;

}

node->value = local\_size;

return (local\_size);

}

void sem\_for\_expression(A\_NODE \*node)

{

A\_TYPE \*t;

switch (node->name)

{

case N\_FOR\_EXP:

if (node->llink)

t = sem\_expression(node->llink);

if (node->clink)

{

t = sem\_expression(node->clink);

if (isScalarType(t))

node->clink = convertScalarToInteger(node->clink);

else

semantic\_error(49, node->line);

}

if (node->rlink)

t = sem\_expression(node->rlink);

break;

default:

semantic\_error(90, node->line);

break;

}

}

// check statement-list and return local variable size

int sem\_statement\_list(A\_NODE \*node, int addr, A\_TYPE \*ret, BOOLEAN sw,

BOOLEAN brk, BOOLEAN cnt)

{

int size, i;

switch (node->name)

{

case N\_STMT\_LIST:

size = sem\_statement(node->llink, addr, ret, sw, brk, cnt);

i = sem\_statement\_list(node->rlink, addr, ret, sw, brk, cnt);

if (size < i)

size = i;

break;

case N\_STMT\_LIST\_NIL:

size = 0;

break;

default:

semantic\_error(90, node->line);

break;

}

node->value = size;

return (size);

}

// check type and return its size (size of incomplete type is 0)

int sem\_A\_TYPE(A\_TYPE \*t)

{

A\_ID \*id;

A\_TYPE \*tt;

A\_LITERAL lit;

int result = 0, i;

if (t->check)

return (t->size);

t->check = 1;

switch (t->kind)

{

case T\_NULL:

semantic\_error(80, t->line);

break;

case T\_ENUM:

i = 0;

id = t->field;

while (id)

{ // enumerators

if (id->init)

{

lit = getTypeAndValueOfExpression(id->init);

if (!isIntType(lit.type))

semantic\_error(81, id->line);

i = lit.value.i;

}

id->init = i++;

id = id->link;

}

result = 4;

break;

case T\_ARRAY:

if (t->expr)

{

lit = getTypeAndValueOfExpression(t->expr);

if (!isIntType(lit.type) || lit.value.i <= 0)

{

semantic\_error(82, t->line);

t->expr = 0;

}

else

t->expr = lit.value.i;

}

// 부록6에는 i에 대입하나 7강 강의 슬라이드에는 result에 대입함

result = sem\_A\_TYPE(t->element\_type) \* (int)t->expr;

if (isVoidType(t->element\_type) || isFunctionType(t->element\_type))

semantic\_error(83, t->line);

// 7강 슬라이드에서 생략됨

// else

// result = i;

break;

case T\_STRUCT:

result = 0; // 7강 슬라이드에 존재

id = t->field;

while (id)

{

result += sem\_declaration(id, result);

id = id->link;

}

break;

case T\_UNION:

result = 0; // 7강 슬라이드에 존재

id = t->field;

while (id)

{

i = sem\_declaration(id, 0);

if (i > result)

result = i;

id = id->link;

}

break;

case T\_FUNC:

tt = t->element\_type;

i = sem\_A\_TYPE(tt);

if (isArrayType(tt) || isFunctionType(tt)) // check return type

semantic\_error(85, t->line);

i = sem\_declaration\_list(t->field, 12) + 12; // parameter type and size

if (t->expr)

{ // skip prototype declaration

i = i + sem\_statement(t->expr, i, t->element\_type, FALSE, FALSE, FALSE);

t->local\_var\_size = i;

break;

}

t->local\_var\_size = i;

break;

case T\_POINTER:

i = sem\_A\_TYPE(t->element\_type);

result = 4;

break;

case T\_VOID:

break;

default:

semantic\_error(90, t->line);

break;

}

t->size = result;

return (result);

}

// set variable address in declaration-list, and return its total variable size

int sem\_declaration\_list(A\_ID \*id, int addr)

{

int i = addr;

while (id)

{

addr += sem\_declaration(id, addr);

id = id->link;

}

return (addr - i);

}

// check declaration (identifier), set address, and return its size

int sem\_declaration(A\_ID \*id, int addr)

{

A\_TYPE \*t;

int size = 0, i;

A\_LITERAL lit;

switch (id->kind)

{

case ID\_VAR:

i = sem\_A\_TYPE(id->type);

// check empty array

if (isArrayType(id->type) && id->type->expr == NIL)

semantic\_error(86, id->line);

if (i % 4)

i = i / 4 \* 4 + 4;

if (id->specifier == S\_STATIC)

id->level = 0;

if (id->level == 0)

{

id->address = global\_address;

global\_address += i;

}

else

{

id->address = addr;

size = i;

}

// if (id->init) // initializer

// sem\_initializer(id->init);

break;

case ID\_FIELD:

i = sem\_A\_TYPE(id->type);

if (isFunctionType(id->type) || isVoidType(id->type))

semantic\_error(84, id->line);

if (i % 4)

i = i / 4 \* 4 + 4;

id->address = addr;

size = i;

break;

case ID\_FUNC:

i = sem\_A\_TYPE(id->type);

break;

case ID\_PARM:

if (id->type)

{

size = sem\_A\_TYPE(id->type);

// usual unary conversion of parm type

if (id->type == char\_type)

id->type = int\_type;

else if (isArrayType(id->type))

{

id->type->kind = T\_POINTER;

id->type->size = 4;

}

else if (isFunctionType(id->type))

{

t = makeType(T\_POINTER);

t->element\_type = id->type;

t->size = 4;

id->type = t;

}

size = id->type->size;

if (size % 4)

size = size / 4 \* 4 + 4;

id->address = addr;

}

break;

case ID\_TYPE:

i = sem\_A\_TYPE(id->type);

break;

default:

semantic\_error(89, id->line, id->name);

break;

}

return (size);

}

A\_ID \*getStructFieldIdentifier(A\_TYPE \*t, char \*s)

{

A\_ID \*id = NIL;

if (isStructOrUnionType(t))

{

id = t->field;

while (id)

{

if (strcmp(id->name, s) == 0)

break;

id = id->link;

}

return (id);

}

}

A\_ID \*getPointerFieldIdentifier(A\_TYPE \*t, char \*s)

{

A\_ID \*id = NIL;

if (t && t->kind == T\_POINTER)

{

t = t->element\_type;

if (isStructOrUnionType(t))

{

id = t->field;

while (id)

{

if (strcmp(id->name, s) == 0)

break;

id = id->link;

}

}

}

return (id);

}

BOOLEAN isSameParameterType(A\_ID \*a, A\_ID \*b)

{

while (a)

{

if (b == NIL || isNotSameType(a->type, b->type))

return (FALSE);

a = a->link;

b = b->link;

}

if (b)

return (FALSE);

else

return (TRUE);

}

BOOLEAN isCompatibleType(A\_TYPE \*t1, A\_TYPE \*t2)

{

if (isArrayType(t1) && isArrayType(t2))

if (t1->size == 0 || t2->size == 0 || t1->size == t2->size)

return (isCompatibleType(t1->element\_type, t2->element\_type));

else

return (FALSE);

else if (isFunctionType(t1) && isFunctionType(t2))

if (isSameParameterType(t1->field, t2->field))

return (isCompatibleType(t1->element\_type, t2->element\_type));

else

return (FALSE);

else if (isPointerType(t1) && isPointerType(t2))

return (isCompatibleType(t1->element\_type, t2->element\_type));

else

return (t1 == t2);

}

BOOLEAN isConstantZeroExp(A\_NODE \*node)

{

if (node->name == N\_EXP\_INT\_CONST && node->clink == 0)

return (TRUE);

else

return (FALSE);

}

BOOLEAN isCompatiblePointerType(A\_TYPE \*t1, A\_TYPE \*t2)

{

if (isPointerType(t1) && isPointerType(t2))

return (isCompatibleType(t1->element\_type, t2->element\_type));

else

return (FALSE);

}

A\_NODE \*convertScalarToInteger(A\_NODE \*node)

{

if (isFloatType(node->type))

{

semantic\_warning(16, node->line);

node = makeNode(N\_EXP\_CAST, int\_type, NIL, node);

}

node->type = int\_type;

return (node);

}

A\_NODE \*convertUsualAssignmentConversion(A\_TYPE \*t1, A\_NODE \*node)

{

A\_TYPE \*t2;

t2 = node->type;

if (!isCompatibleType(t1, t2))

{

semantic\_warning(11, node->line);

node = makeNode(N\_EXP\_CAST, t1, NIL, node);

node->type = t1;

}

return (node);

}

A\_NODE \*convertUsualUnaryConversion(A\_NODE \*node)

{

A\_TYPE \*t;

t = node->type;

if (t == char\_type)

{

t = int\_type;

node = makeNode(N\_EXP\_CAST, t, NIL, node);

node->type = t;

}

else if (isArrayType(t))

{

t = setTypeElementType(makeType(T\_POINTER), t->element\_type);

t->size = 4;

node = makeNode(N\_EXP\_CAST, t, NIL, node);

node->type = t;

}

else if (isFunctionType(t))

{

t = setTypeElementType(makeType(T\_POINTER), t);

t->size = 4;

node = makeNode(N\_EXP\_AMP, NIL, node, NIL);

node->type = t;

}

return (node);

}

A\_TYPE \*convertUsualBinaryConversion(A\_NODE \*node)

{

A\_TYPE \*t1, \*t2, \*result = NIL;

t1 = node->llink->type;

t2 = node->rlink->type;

if (isFloatType(t1) && !isFloatType(t2))

{

semantic\_warning(14, node->line);

node->rlink = makeNode(N\_EXP\_CAST, t1, NIL, node->rlink);

node->rlink->type = t1;

result = t1;

}

else if (!isFloatType(t1) && isFloatType(t2))

{

semantic\_warning(14, node->line);

node->llink = makeNode(N\_EXP\_CAST, t2, NIL, node->llink);

node->llink->type = t2;

result = t2;

}

else if (t1 == t2)

result = t1;

else

result = int\_type;

return (result);

}

A\_NODE \*convertCastingConversion(A\_NODE \*node, A\_TYPE \*t1)

{

A\_TYPE \*t2;

t2 = node->type;

if (!isCompatibleType(t1, t2))

{

semantic\_warning(12, node->line);

node = makeNode(N\_EXP\_CAST, t1, NIL, node);

node->type = t1;

}

return (node);

}

BOOLEAN isAllowableAssignmentConversion(A\_TYPE \*t1, A\_TYPE \*t2,

A\_NODE \*node) // t1 <--- t2

{

if (isArithmeticType(t1) && isArithmeticType(t2))

return (TRUE);

else if (isStructOrUnionType(t1) && isCompatibleType(t1, t2))

return (TRUE);

// 두 포인터 타입이 같은 경우. i.e. int \* <- int \*나 float \* <- float \*

else if (isPointerType(t1) &&

(isConstantZeroExp(node) || isCompatiblePointerType(t1, t2)))

return (TRUE);

// 두 포인터 타입이 다른 경우. i.e. int \* <- float \*나 char \* <- int \*

else if (isPointerType(t1) && isPointerType(t2))

semantic\_warning(11, node->line);

// integral\_type <- pointer\_type

else if (isIntegralType(t1) && isPointerType(t2))

semantic\_warning(11, node->line);

// pointer\_type <- integral\_type

else if (isPointerType(t1) && isIntegralType(t2))

semantic\_warning(11, node->line);

else

return (FALSE);

}

BOOLEAN isAllowableCastingConversion(A\_TYPE \*t1, A\_TYPE \*t2)

{

// t1 <--- t2

if (isAnyIntegerType(t1) &&

(isAnyIntegerType(t2) || isFloatType(t2) || isPointerType(t2)))

return (TRUE);

else if (isFloatType(t1) && isArithmeticType(t2))

return (TRUE);

else if (isPointerType(t1) && (isAnyIntegerType(t2) || isPointerType(t2)))

return (TRUE);

else if (isVoidType(t1))

return (TRUE);

// i.e. int\_type <- enum\_type이 가능하도록 함.

else if (isAnyIntegerType(t1) && (isScalarType(t2)))

return (TRUE);

// i.e. enum\_type <- scalar\_type이 가능하도록 함.

else if (isIntegralType(t1) && (isScalarType(t2)))

return (TRUE);

// i.e. pointer\_type <- enum\_type이 가능하도록 함.

else if (isPointerType(t1) && (isScalarType(t2)))

return (TRUE);

else

return (FALSE);

}

BOOLEAN isFloatType(A\_TYPE \*t)

{

if (t == float\_type)

return (TRUE);

else

return (FALSE);

}

BOOLEAN isArithmeticType(A\_TYPE \*t)

{

if (t && t->kind == T\_ENUM)

return (TRUE);

else

return (FALSE);

}

BOOLEAN isScalarType(A\_TYPE \*t)

{

if (t && ((t->kind == T\_ENUM) || (t->kind == T\_POINTER)))

return (TRUE);

else

return (FALSE);

}

BOOLEAN isAnyIntegerType(A\_TYPE \*t)

{

if (t && (t == int\_type || t == char\_type))

return (TRUE);

else

return (FALSE);

}

BOOLEAN isIntegralType(A\_TYPE \*t)

{

if (t && t->kind == T\_ENUM && t != float\_type)

return (TRUE);

else

return (FALSE);

}

BOOLEAN isFunctionType(A\_TYPE \*t)

{

if (t && t->kind == T\_FUNC)

return (TRUE);

else

return (FALSE);

}

BOOLEAN isStructOrUnionType(A\_TYPE \*t)

{

if (t && (t->kind == T\_STRUCT || t->kind == T\_UNION))

return (TRUE);

else

return (FALSE);

}

BOOLEAN isPointerType(A\_TYPE \*t)

{

if (t && t->kind == T\_POINTER)

return (TRUE);

else

return (FALSE);

}

BOOLEAN isPointerOrArrayType\_sem(A\_TYPE \*t)

{

if (t && (t->kind == T\_POINTER || t->kind == T\_ARRAY))

return (TRUE);

else

return (FALSE);

}

BOOLEAN isIntType(A\_TYPE \*t)

{

if (t && t == int\_type)

return (TRUE);

else

return (FALSE);

}

BOOLEAN isVoidType(A\_TYPE \*t)

{

if (t && t == void\_type)

return (TRUE);

else

return (FALSE);

}

BOOLEAN isArrayType(A\_TYPE \*t)

{

if (t && t->kind == T\_ARRAY)

return (TRUE);

else

return (FALSE);

}

BOOLEAN isStringType(A\_TYPE \*t)

{

if (t && (t->kind == T\_POINTER || t->kind == T\_ARRAY) &&

t->element\_type == char\_type)

return (TRUE);

else

return (FALSE);

}

// convert literal type

A\_LITERAL checkTypeAndConvertLiteral(A\_LITERAL result, A\_TYPE \*t, int ll)

{

if (result.type == int\_type && t == int\_type ||

result.type == char\_type && t == char\_type ||

result.type == float\_type && t == float\_type)

;

else if (result.type == int\_type && t == float\_type)

{

result.type = float\_type;

result.value.f = result.value.i;

}

else if (result.type == int\_type && t == char\_type)

{

result.type = char\_type;

result.value.c = result.value.i;

}

else if (result.type == float\_type && t == int\_type)

{

result.type = int\_type;

result.value.i = result.value.f;

}

else if (result.type == char\_type && t == int\_type)

{

result.type = int\_type;

result.value.i = result.value.c;

}

else

semantic\_error(41, ll);

return (result);

}

A\_LITERAL getTypeAndValueOfExpression(A\_NODE \*node)

{

A\_TYPE \*t;

A\_ID \*id;

A\_LITERAL result, r;

result.type = NIL;

switch (node->name)

{

case N\_EXP\_IDENT:

id = node->clink;

if (id->kind != ID\_ENUM\_LITERAL)

semantic\_error(19, node->line, id->name);

else

{

result.type = int\_type;

result.value.i = id->value;

// 부록6

// result.value.i = id->init;

}

break;

case N\_EXP\_INT\_CONST:

result.type = int\_type;

result.value.i = (int)node->clink;

break;

case N\_EXP\_CHAR\_CONST:

result.type = char\_type;

result.value.c = (char)node->clink;

break;

case N\_EXP\_FLOAT\_CONST:

result.type = float\_type;

result.value.f = atof(node->clink);

break;

case N\_EXP\_STRING\_LITERAL:

case N\_EXP\_ARRAY:

case N\_EXP\_FUNCTION\_CALL:

case N\_EXP\_STRUCT:

case N\_EXP\_ARROW:

case N\_EXP\_POST\_INC:

case N\_EXP\_PRE\_INC:

case N\_EXP\_POST\_DEC:

case N\_EXP\_PRE\_DEC:

case N\_EXP\_AMP:

case N\_EXP\_STAR:

case N\_EXP\_NOT:

semantic\_error(18, node->line);

break;

case N\_EXP\_MINUS:

result = getTypeAndValueOfExpression(node->clink);

if (result.type == int\_type)

result.value.i = -result.value.i;

else if (result.type == float\_type)

result.value.f = -result.value.f;

else

semantic\_error(18, node->line);

break;

case N\_EXP\_SIZE\_EXP:

t = sem\_expression(node->clink);

result.type = int\_type;

result.value.i = t->size;

break;

case N\_EXP\_SIZE\_TYPE:

result.type = int\_type;

result.value.i = sem\_A\_TYPE(node->clink);

break;

case N\_EXP\_CAST:

result = getTypeAndValueOfExpression(node->rlink);

result =

checkTypeAndConvertLiteral(result, (A\_TYPE \*)node->llink, node->line);

break;

case N\_EXP\_MUL:

result = getTypeAndValueOfExpression(node->llink);

r = getTypeAndValueOfExpression(node->rlink);

if (result.type == int\_type && r.type == int\_type)

{

result.type = int\_type;

result.value.i = result.value.i \* r.value.i;

}

else if (result.type == int\_type && r.type == float\_type)

{

result.type = float\_type;

result.value.f = result.value.i \* r.value.f;

}

else if (result.type == float\_type && r.type == int\_type)

{

result.type = float\_type;

result.value.f = result.value.f \* r.value.i;

}

else if (result.type == float\_type && r.type == float\_type)

{

result.type = float\_type;

result.value.f = result.value.f \* r.value.f;

}

else

semantic\_error(18, node->line);

break;

case N\_EXP\_DIV:

result = getTypeAndValueOfExpression(node->llink);

r = getTypeAndValueOfExpression(node->rlink);

if (result.type == int\_type && r.type == int\_type)

{

result.type = int\_type;

result.value.i = result.value.i / r.value.i;

}

else if (result.type == int\_type && r.type == float\_type)

{

result.type = float\_type;

result.value.f = result.value.i / r.value.f;

}

else if (result.type == float\_type && r.type == int\_type)

{

result.type = float\_type;

result.value.f = result.value.f / r.value.i;

}

else if (result.type == float\_type && r.type == float\_type)

{

result.type = float\_type;

result.value.f = result.value.f / r.value.f;

}

else

semantic\_error(18, node->line);

break;

case N\_EXP\_MOD:

result = getTypeAndValueOfExpression(node->llink);

r = getTypeAndValueOfExpression(node->rlink);

if (result.type == int\_type && r.type == int\_type)

result.value.i = result.value.i % r.value.i;

else

semantic\_error(18, node->line);

break;

case N\_EXP\_ADD:

result = getTypeAndValueOfExpression(node->llink);

r = getTypeAndValueOfExpression(node->rlink);

if (result.type == int\_type && r.type == int\_type)

{

result.type = int\_type;

result.value.i = result.value.i + r.value.i;

}

else if (result.type == int\_type && r.type == float\_type)

{

result.type = float\_type;

result.value.f = result.value.i + r.value.f;

}

else if (result.type == float\_type && r.type == int\_type)

{

result.type = float\_type;

result.value.f = result.value.f + r.value.i;

}

else if (result.type == float\_type && r.type == float\_type)

{

result.type = float\_type;

result.value.f = result.value.f + r.value.f;

}

else

semantic\_error(18, node->line);

break;

case N\_EXP\_SUB:

result = getTypeAndValueOfExpression(node->llink);

r = getTypeAndValueOfExpression(node->rlink);

if (result.type == int\_type && r.type == int\_type)

{

result.type = int\_type;

result.value.i = result.value.i - r.value.i;

}

else if (result.type == int\_type && r.type == float\_type)

{

result.type = float\_type;

result.value.f = result.value.i - r.value.f;

}

else if (result.type == float\_type && r.type == int\_type)

{

result.type = float\_type;

result.value.f = result.value.f - r.value.i;

}

else if (result.type == float\_type && r.type == float\_type)

{

result.type = float\_type;

result.value.f = result.value.f - r.value.f;

}

else

semantic\_error(18, node->line);

break;

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\_AND:

case N\_EXP\_OR:

case N\_EXP\_ASSIGN:

semantic\_error(18, node->line);

break;

default:

semantic\_error(90, node->line);

break;

} // close switch statement

return (result);

}

void semantic\_error(int i, int ll, char \*s)

{

semantic\_err++;

printf("%s\*\*\* semantic error at line %d : ", C\_RED, ll);

switch (i)

{

// errors in expression

case 13:

printf("artih type expr required in unary operation\n");

break;

case 18:

printf("illegal constant expression\n");

break;

case 19:

printf("illegal identifier %s in constant expression\n", s);

break;

case 21:

printf("illegal type in function call expression\n");

break;

case 24:

printf("incompatible type in additive expression\n");

break;

case 27:

printf("scalar type expr required in expression\n");

break;

case 28:

printf("arith type expression required in binary operation\n");

break;

case 29:

printf("integral type expression required in expression\n");

break;

case 31:

printf("pointer type expr required in pointer operation\n");

break;

case 32:

printf("array type required in array expression\n");

break;

case 34:

printf("too many arguments in function call\n");

break;

case 35:

printf("too few arguments in function call\n");

break;

case 37:

printf("illegal struct field identifier in struct reference expr\n");

break;

case 38:

printf("illegal kind of identifier %s in expression\n",

s); // 부록6에 s가 빠져있는 오타가 있음.

break;

case 39:

printf("illegal type size in sizeof operation\n");

break;

case 40:

printf("illegal expression type in relational operation\n");

break;

case 41:

printf("incompatible type in literal\n");

// 부록6에는 해당 에러코드가 주석 처리 되어 있음.

break;

// errors in statement

case 49:

printf("scalar type expr required in middle of for-expr\n");

break;

case 50:

printf("intergral type expression required in statement\n");

break;

case 51:

printf("illegal expression type in case label\n");

break;

case 57:

printf("not permitted type conversion in return expression\n");

break;

case 58:

printf("not permitted type casting in expression\n");

break;

case 59:

printf("not permitted type conversion in argument\n");

break;

case 60:

printf("expression is not an lvalue\n");

break;

case 71:

printf("case label not within a switch statement\n");

break;

case 72:

printf("default label not within a switch statement\n");

break;

case 73:

printf("break statement not within loop or switch stmt\n");

break;

case 74:

printf("continue statement not within a loop\n");

break;

// errors in type & declarator

case 80:

printf("undefined type\n");

break;

case 81:

printf("integer type expression required in enumerator\n");

break;

case 82:

printf("illegal array size or type\n");

break;

case 83:

printf("illegal element type of array declarator\n");

break;

case 84:

printf("illegal type in struct or union field\n");

break;

case 85:

printf("invalid function return type\n");

break;

case 86:

printf("illegal array size or empty array\n");

break;

case 89:

printf("unknown identifer kind : %s\n", s);

break;

// misc errors

case 90:

printf("fatal compiler error in parse result\n");

break;

case 93:

printf("too many literals in source program\n");

break;

default:

printf("unknown \n");

break;

}

}

void semantic\_warning(int i, int ll)

{

printf("%s--- warning at line %d :", C\_YLLW, ll); // 노란색으로 warning 표시

switch (i)

{

case 11:

printf("incompatible types in assignment expression\n");

break;

case 12:

printf("incompatible types in argument or return expr\n");

break;

case 14:

printf("incompatible types in binary expression\n");

break;

case 16:

printf("integer type expression in required\n");

break;

default:

printf("unknown\n");

break;

}

}

// void \*sem\_initializer(A\_NODE \*node)

// {

// A\_TYPE \*t;

// switch (node->name)

// {

// case N\_INIT\_LIST:

// sem\_initializer(node->llink);

// sem\_initializer(node->rlink);

// break;

// case N\_INIT\_LIST\_ONE:

// sem\_expression(node->clink);

// break;

// case N\_INIT\_LIST\_NIL:

// break;

// }

// }