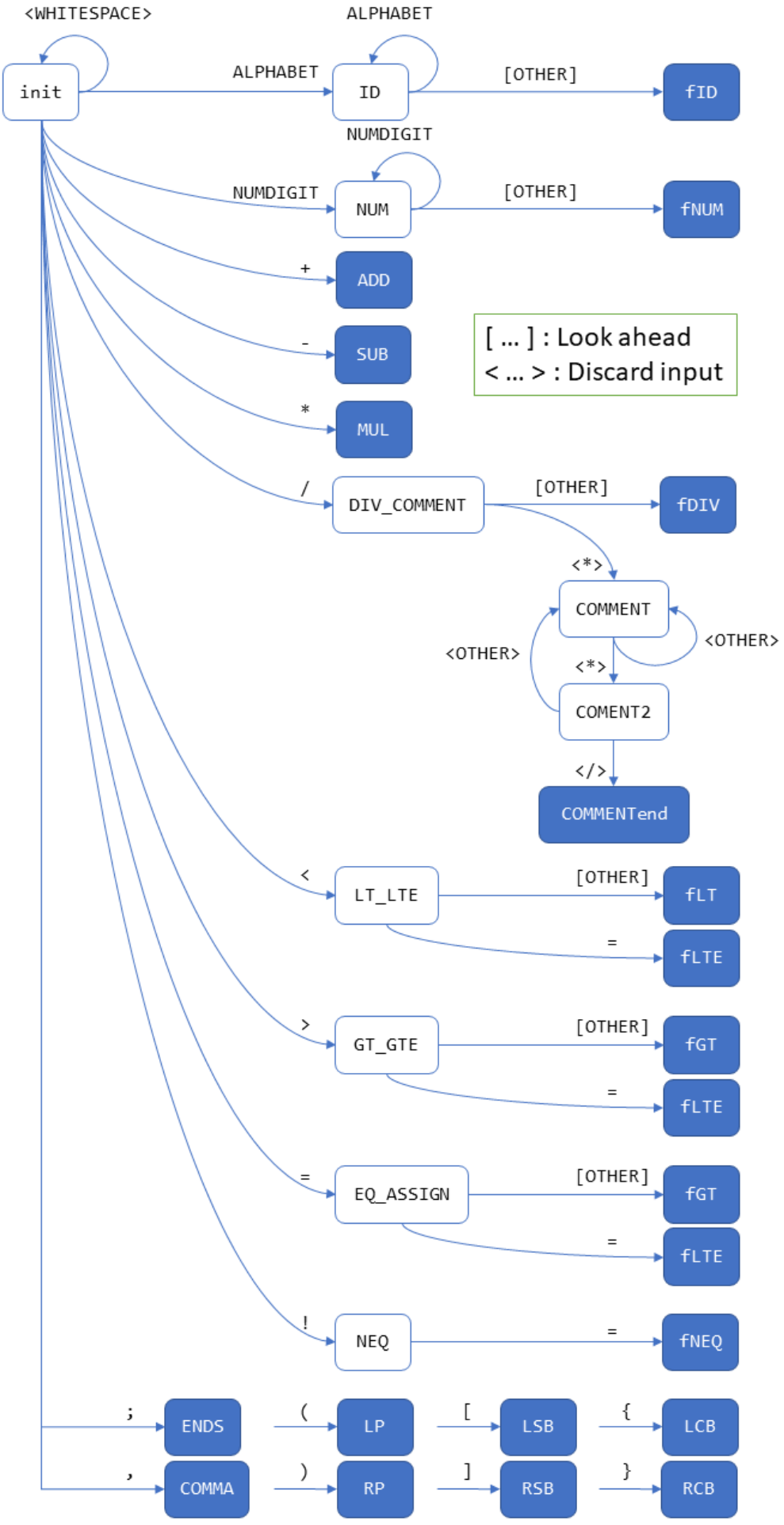
Compiler COMP032001

Scanner Project / 2016112905 김민섭

순서

1. DFA


<WHITESPACE>
ALPHABET
init
ALPHABET
ID
[OTHER]
fID
NUMDIGIT
NUMDIGIT
NUM
[OTHER]
fNUM
+
ADD
-
SUB
*
MUL
/
DIV_COMMENT
[OTHER]
fDIV
[...] : Look ahead
< ... > : Discard input
<*>
COMMENT
<OTHER>
<OTHER>
<*>
COMENT2
</>
COMMENTend
<
LT_LTE
[OTHER]
fLT
=
fLTE
>
GT_GTE
[OTHER]
fGT
=
fLTE
=
EQ_ASSIGN
[OTHER]
fGT
=
fLTE
!
NEQ
=
fNEQ
;
ENDS
(
LP
[
LSB
{
LCB
,
COMMA
)
RP
]
RSB
}
RCB

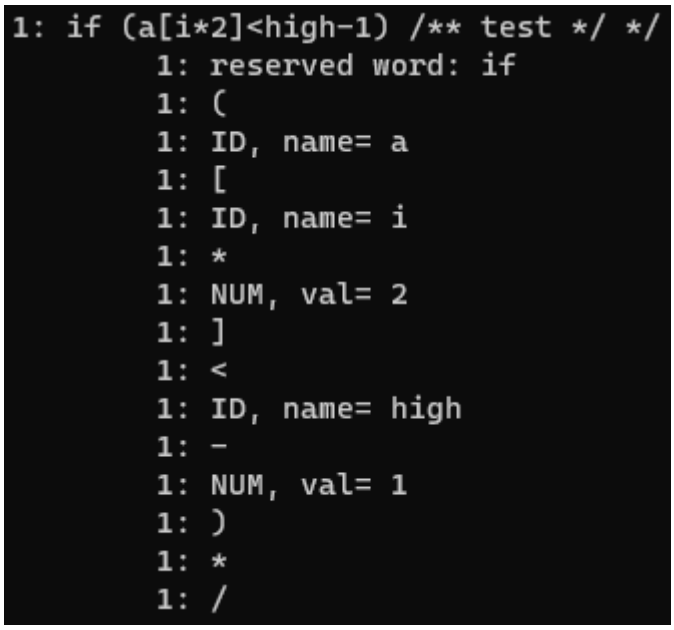
2. Scanning Process

2.1. 정상 예제 문장: if (a[i*2]<high-1) /** test */ */

Input character	Current state	Next state	Token output	설명
	init			시작 state는 init으로 초기화됩니다.
i	init	ID		
f	ID	ID		
<SPACE>	ID	fID	IF	SPACE는 discard 처리됩니다. (토큰 버퍼에 수집되지 않고 버려집니다.) 프로그램에 의해 final state에서 init으로 초기화됩니다.
<SPACE>	init	init		
(	init	LP	(	
a	init	ID		
[[]	ID	fID	ID, name= a	[는 lookahead 처리됩니다.
[	init	LSB	[	
i	init	ID		
[*]	ID	fID	ID, name= i	
*	init	MUL	*	
2	init	NUM		
[]]	NUM	fNUM	NUM, val= 2	
]	init	RSB	]	
<	init	LT_LTE		
[h]	LT_LTE	fLT	<	
h	init	ID		
i	ID	ID		
g	ID	ID		
h	ID	ID		
[-]	ID	fID	ID, name= high	
-	init	SUB	-	
1	init	NUM		
[)]	NUM	fNUM	NUM, val= 1	
)	init	RP	)	
<SPACE>	init	init		
/	init	DIV_COMMENT		
<*>	DIV_COMMENT	COMMENT		주석의 시작이므로 입력은 discard 처리됩니다.
<*>	COMMENT	COMMENT2		
<SPACE>	COMMENT2	COMMENT		
<t>	COMMENT	COMMENT		
<e>	COMMENT	COMMENT		
<s>	COMMENT	COMMENT		
<t>	COMMENT	COMMENT		

<SPACE>	COMMENT	COMMENT		
<*>	COMMENT	COMMENT2		
</>	COMMENT2	COMMENTend		주석이 끝났으므로 프로그램에 의해 final state에서 init으로 초기화됩니다.
<SPACE>	init	init		
*	init	MUL	*	
/	init	DIV_COMMENT		
<EOF>	DIV_COMMENT	fDIV	/	나누기 연산자인지 주석의 시작부분인지를 알려주는 delimiter가 필요할 때는 EOF도 delimiter로서 동작합니다. 파일 디스크립터는 한번 EOF에 도달하면 read할 때마다 EOF를 반환하기 때문에 lookahead와 discard의 차이가 없습니다. 이 프로그램에서는 코드를 줄이기 위해 다른 whitespace 문자와 함께 discard 처리했습니다.
EOF	init			EOF임을 알려주는 값을 반환합니다.

- 스캐닝 결과 콘솔화면:



2.2. 에러 예제 문장: &₩n!@₩n/*

Input character	Current state	Next state	Token output	설명
	init			
&	init		eINVALIDINPUT	init에서 transition이 없는 입력일 경우.
<\n>	init	init		
!	init	NEQ		
[@]	NEQ		eINVALIDRULE	이전 입력에 의해 init이 아닌 다른 state로 이동한 상태에서 transition이 없는 입력일 경우. 프로그램에 의해 state는 init으로 초기화되고 입력은 lookahead처리합니다.
@	init		eINVALIDINPUT	init에서 transition이 없는 입력일 경우.
<\n>	init	init		
/	init	DIV_COMMENT		
<*>	DIV_COMMENT	COMMENT		
EOF	COMMENT		eNOCOMMENTEND	COMMENT나 COMMENT2 state에서 EOF가 입력되면 EOF임을 알려주는 대신 주석이 끝나지 않았다는 에러값을 반환합니다.

- 스캐닝 결과 콘솔화면:

```
1: &
        1: error: invalid input: &
2: !@
        2: error: incomplete token: !
        2: error: invalid input: @
3: /*
        3: error: missing comment end '*/'
```

- 처리할 수 있는 에러 타입

에러 타입	설명
eNOMATCHINGTTYPE	DFA로 구현되었으나 해당 토큰 타입을 지정하지 않은 경우. 기능 확장 시 프로그래밍 실수에 대비한 에러 타입입니다.
eINVALIDINPUT	DFA initial state에서 input에 대해 transition이 만들어지지 않은 경우. ex) #, %
eINVALIDRULE	DFA initial 이 아닌 state에 대해 transition이 만들어지지 않은 경우. ex) '!' 입력 후 '='입력일 기대하지만 다른 문자가 입력된 경우

3. 구현 방법

Table driven 방식으로 하였으나 Matrix 형태가 아닌 List로서 DFA state를 하나씩 추가하는 방식으로 구현했습니다. 따라서 클래스 멤버와 함수를 자세하게 기술한 뒤 List를 구축하는 소스코드를 첨부했습니다.

3.1. 클래스

3.1.1. class SingleState

실제 Finite state automata의 state처럼 동작하는 클래스입니다. Scanner 클래스에서 DFA를 구축할 때 사용됩니다.

class SingleState		
private	int data	final인지 nonfinal인지 구분할 수 있는 변수.
	std::map<char, std::pair<SingleState*, int>> transition	입력 문자에 대응하는 transition 리스트를 저장하는 변수. transition 결과 state 포인터와 옵션(lookahead, discard를 구분하는 정보)을 저장합니다.
public	void addMap(char in, SingleState* next, int opt)	입력 문자 in에 대해 transition 결과 state 포인터 next와 옵션 opt를 저장하는 함수.
	std::pair<SingleState*, int> pushChar(char in)	transition에서 일치하는 입력 문자를 찾고 해당 값(next, opt)을 반환하는 함수. 입력 문자가 없을 경우 (nullptr, -1)를 반환합니다.
	int getData()	data를 반환합니다.

3.1.2. class FileHeader

파일 읽기를 담당하는 클래스입니다. 입력 파일 스트림으로부터 문자열을 한 줄 씩 가져와서 버퍼에 저장했다가, Scanner 클래스의 요청에 의해 문자를 하나씩 반환합니다.

class FileHeader		
private	std::ifstream* fin	입력 파일 스트림 변수.
	std::string buffer	입력 스트림에서 읽은 문자열 한 줄을 저장하는 변수.
	size_t length	버퍼의 길이를 저장하는 함수.
	int cursor	버퍼에서 다음 순서로 반환할 문자 인덱스를 저장하는 변수.
	char putBackChar	lookahead 처리된 입력을 저장하는 변수.
	bool putBackFlag	lookahead 처리된 입력이 있는지 나타내는 변수.
public	char getChar()	lookahead를 우선적으로 반환합니다. 없을 경우 버퍼에서 다음 순서의 문자를 반환합니다. 만약 버퍼의 문자를 끝까지 다 읽었을 경우 입력 스트림에서 새로운 문자열을 읽어오고 newline('\n') 문자를 반환합니다. 만약 EOF에 도달할 경우 EOF를 반환합니다.
	void put Back(char c)	문자 c를 lookahead 처리합니다.

3.1.3. class Scanner

이 클래스는 SingleState 클래스를 이용해 DFA를 구축하고 processChar() 함수 호출을 기다립니다.

class Scanner		
private	FileHeader fileheader	
	map<string, SingleState*> states	전체 DFA state 리스트 변수. DFA를 설계할 때 state에 이름을 붙이듯이 이 프로그램에서도 state를 문자열로 검색할 수 있도록 했습니다.
	SingleState* currentState	DFA에서 현재 state를 가리키는 포인터 변수.
	string tokenBuffer	처리된 토큰 문자열을 저장하는 변수.
	TokErrType errType	processChar() 실행 중 발생한 에러 종류를 저장하는 변수.
	bool newline	processChar() 실행 중 입력 스트림으로부터 newline 문자를 읽었는지를 기억하는 변수.
public	TokenType processChar()	입력 스트림으로부터 문자 한 개를 가져와서 DFA에 입력한 뒤 결과를 반환합니다. 아래에서 더 자세하게 기술합니다.
	string getToken()	실제 처리된 토큰 문자열을 반환합니다.
	TokErrType getErrorType()	에러 종류에 대한 정보를 반환합니다.
	bool isNewLine()	최근에 입력 스트림으로부터 newline 문자를 읽었는지를 알 수 있습니다.

3.2. DFA 구축 소스코드

3.2.1. addState(), mapState() 함수와 Scanner 클래스 생성자

```
class Scanner {
private:
...
    void addState(string name, StateData data = StateData::nonf) {
        // 새로운 state 를 검색 이름과 함께 생성합니다.
        // StateData 는 state 종류를 나타냅니다. 인자 기본값은 non-final 입니다.
        states[name] = new SingleState(static_cast<int>(data));
    }

    void mapState(string from, string to,
        string instring,
        TransitionOption opt = TransitionOption::optNORMAL) {
        // 두 state 사이에 transition 을 만듭니다.
        // Lookahead, Discard 등의 입력 문자 처리 방법은 opt 인자로 설정합니다.
        // 입력 문자는 코딩 편의성을 위해 스트링으로 받아서 일괄처리합니다.
        SingleState* s1 = states[from];
        SingleState* s2 = states[to];
        for (char in : instring) {
            s1->addMap(in, s2, static_cast<int>(opt));
        }
    }
...
public:
    Scanner(ifstream* f) : fileHeader(FileHeader(f)),
                           tokenBuffer(""),
                           flushFlag(false),
                           errorType(TokErrType::eNOERROR) {
        EOFSTR.push_back(EOF);

        // DFA 를 구축하는 각 단계는 void 함수로 구분했습니다.
        buildState();
        buildInit();
        buildID();
        buildNUM();
        buildDIV_COMMENT();
        buildLT_LTE();
        buildGT_GTE();
        buildEQ_ASSIGN();
        buildNEQ();

        currentState = states["init"];
    }
...
};
```

3.2.2. buildState()

```
void buildState() {
    // DFA 의 init state 와 각 REX 의 시작이 되는 state 들을 추가하는 과정입니다.
    // 명시하지 않은 StateData 의 기본값은 StateData::nonf (non-final)입니다.
    addState("init");
    addState("ID");
    addState("NUM");
    addState("ADD", StateData::fADD); // 여기는 final state 이므로 init 으로부터
    addState("SUB", StateData::fSUB); // transition 를 잇는 것 외에 추가할 것이
    addState("MUL", StateData::fMUL); // 없습니다.
    addState("DIV_COMMENT");
    addState("LT_LTE");
    addState("GT_GTE");
    addState("EQ_ASSIGN");
    addState("NEQ");
    addState("ENDS", StateData::fENDS);
    addState("COMMA", StateData::fCOMMA);
    addState("LP", StateData::fLP);
    addState("RP", StateData::fRP);
    addState("LSB", StateData::fLSB);
    addState("RSB", StateData::fRSB);
    addState("LCB", StateData::fLCB);
    addState("RCB", StateData::fRCB);
}
```

3.2.3. buildInit()

```
void buildInit() {
    // init 과 각 REX 의 시작 state 들 사이에 transition 을 잇습니다.
    mapState("init", "init", WHITESPACE, TransitionOption::optDISCARD);
    mapState("init", "ID", ALPHABET);
    mapState("init", "NUM", NUMDIGIT);
    mapState("init", "ADD", "+");
    mapState("init", "SUB", "-");
    mapState("init", "MUL", "*");
    mapState("init", "DIV_COMMENT", "/");
    mapState("init", "LT_LTE", "<");
    mapState("init", "GT_GTE", ">");
    mapState("init", "EQ_ASSIGN", "=");
    mapState("init", "NEQ", "!");
    mapState("init", "ENDS", ";");
    mapState("init", "COMMA", ",");
    mapState("init", "LP", "(");
    mapState("init", "RP", ")");
    mapState("init", "LSB", "[");
    mapState("init", "RSB", "]");
    mapState("init", "LCB", "{");
    mapState("init", "RCB", "}");
}
```

3.2.4. buildID()

```
void buildID() {
    // Identifier REX 를 처리하는 state 입니다.
    mapState("ID", "ID", ALPHABET);

    addState("fID", StateData::fID);
    mapState("ID", "fID",
        NUMDIGIT + OTHERCHAR,
        TransitionOption::optLOOKAHEAD);
    mapState("ID", "fID",
        WHITESPACE + EOFSTR,
        TransitionOption::optDISCARD); // 의미 없는 문자는 버립니다.
}
```

3.2.5. buildNUM()

```
void buildNUM() {
    // Number REX 를 처리하는 state 입니다.
    mapState("NUM", "NUM", NUMDIGIT);

    addState("fNUM", StateData::fNUM);
    mapState("NUM", "fNUM",
        ALPHABET + OTHERCHAR,
        TransitionOption::optLOOKAHEAD);
    mapState("NUM", "fNUM",
        WHITESPACE + EOFSTR,
        TransitionOption::optDISCARD);
}
```

3.2.6. buildDIV_COMMENT()

```
void buildDIV_COMMENT() {
    // 나누기 연산자(/)와 주석(/* */) REX 를 처리하는 state 입니다.
    addState("fDIV", StateData::fDIV);
    mapState("DIV_COMMENT", "fDIV",
        ALPHABET + NUMDIGIT + dropChars(OTHERCHAR, "*"),
        TransitionOption::optLOOKAHEAD);
    mapState("DIV_COMMENT", "fDIV",
        WHITESPACE + EOFSTR,
        TransitionOption::optDISCARD);

    addState("COMMENT");
    mapState("DIV_COMMENT", "COMMENT", "*");
    mapState("COMMENT", "COMMENT",
        ALPHABET + NUMDIGIT + dropChars(OTHERCHAR, "*")
        + WHITESPACE,
        TransitionOption::optDISCARD);  // 주석 내용은 모두 버립니다.

    addState("COMMENT2");
    mapState("COMMENT", "COMMENT2", "*");
    mapState("COMMENT2", "COMMENT",
```

```
        ALPHABET + NUMDIGIT + dropChars(OTHERCHAR, "/")
        + WHITESPACE,
        TransitionOption::optDISCARD);

    addState("COMMENTend", StateData::fCOMMENT);
    mapState("COMMENT2", "COMMENTend", "/",
        TransitionOption::optDISCARD);
}
```

3.2.7. buildLT_LTE()

```
void buildLT_LTE() {
    // 비교 연산자 <, <= REX 를 처리하는 state 입니다.
    addState("fLT", StateData::fLT);
    mapState("LT_LTE", "fLT",
        ALPHABET + NUMDIGIT + dropChars(OTHERCHAR, "="),
        TransitionOption::optLOOKAHEAD);
    mapState("LT_LTE", "fLT",
        WHITESPACE + EOFSTR,
        TransitionOption::optDISCARD);

    addState("fLTE", StateData::fLTE);
    mapState("LT_LTE", "fLTE", "=");
}
```

3.2.8. buildGT_GTE()

```
void buildGT_GTE() {
    // 비교 연산자 >, >= REX 를 처리하는 state 입니다.
    addState("fGT", StateData::fGT);
    mapState("GT_GTE", "fGT",
        ALPHABET + NUMDIGIT + dropChars(OTHERCHAR, "="),
        TransitionOption::optLOOKAHEAD);
    mapState("GT_GTE", "fGT",
        WHITESPACE + EOFSTR,
        TransitionOption::optDISCARD);

    addState("fGTE", StateData::fGTE);
    mapState("GT_GTE", "fGTE", "=");
}
```

3.2.9. buildEQ_ASSIGN()

```
void buildEQ_ASSIGN() {
    // 비교 연산자 ==와 할당 연산자 REX 를 처리하는 state 입니다.
    addState("fEQ", StateData::fEQ);
    mapState("EQ_ASSIGN", "fEQ", "=");

    addState("fASSIGN", StateData::fASSIGN);
    mapState("EQ_ASSIGN", "fASSIGN",
        ALPHABET + NUMDIGIT + dropChars(OTHERCHAR, "="),
```

```
        TransitionOption::optLOOKAHEAD);
    mapState("EQ_ASSIGN", "fASSIGN",
        WHITESPACE + EOFSTR,
        TransitionOption::optDISCARD);
}
```

3.2.10. buildNEQ()

```
void buildNEQ() {
    // 비교 연산자 != REX 를 처리하는 state 입니다.
    addState("fNEQ", StateData::fNEQ);
    mapState("NEQ", "fNEQ", "=");
}
```

3.3. processChar() in Pseudo-Code

```
TokenType processChar () {
    // 소스코드에서 문자 하나를 처리하는 과정을 Pseudo-Code 로 나타냈습니다.

    in ← getChar ()              // Lookahead 도 처리할 수 있습니다.
    detectNewLine (&newline, in) // '\n'일 경우 True, 그렇지 않으면 False
    flushBufferIf (flushFlag)    // flushFlag 가 True 일 경우 토큰 버퍼를 비우고 False

    nextState, opt ← pushCharToDFA (in)
    if nextState==nullptr then   // DFA 로부터 아무 결과를 얻지 못했으므로 에러입니다.
        errorType, flushFlag ← examineError (currentState, in)
        return ERROR

    processTransitionOption (opt, in) // Lookahead, Discard 를 처리합니다.

    stateData ← nextState.getData ()
    tokenType, currentState, flushFlag ← examineNextState (stateData)
    return tokenType // 토큰이 아직 없는 경우도 포함됩니다.
}
```

4. 2.c 실행 결과

```
1: /* A program to perform selection sort on a 10
2:    element array. */
3:
4: int x[10];
        4: reserved word: int
        4: ID, name= x
        4: [
        4: NUM, val= 10
        4: ]
        4: ;
5:
6: int minloc ( int a[], int low, int high )
        6: reserved word: int
        6: ID, name= minloc
        6: (
        6: reserved word: int
        6: ID, name= a
        6: [
        6: ]
        6: ,
        6: reserved word: int
        6: ID, name= low
        6: ,
        6: reserved word: int
        6: ID, name= high
        6: )
7: {    int i; int x; int k;
        7: {
        7: reserved word: int
        7: ID, name= i
        7: ;
        7: reserved word: int
        7: ID, name= x
        7: ;
        7: reserved word: int
        7: ID, name= k
        7: ;
8:      k = low;
        8: ID, name= k
        8: =
        8: ID, name= low
        8: ;
9:      x = a[low];
        9: ID, name= x
        9: =
        9: ID, name= a
        9: [
        9: ID, name= low
        9: ]
        9: ;
10:     i = low + 1;
        10: ID, name= i
        10: =
        10: ID, name= low
        10: +
        10: NUM, val= 1
        10: ;
```

```
11:     while (i < high)
        11: reserved word: while
        11: (
        11: ID, name= i
        11: <
        11: ID, name= high
        11: )
12:     {       if (a[i] < x)
        12: {
        12: reserved word: if
        12: (
        12: ID, name= a
        12: [
        12: ID, name= i
        12: ]
        12: <
        12: ID, name= x
        12: )
13:                     { x = a[i];
        13: {
        13: ID, name= x
        13: =
        13: ID, name= a
        13: [
        13: ID, name= i
        13: ]
        13: ;
14:                       k = i;  }
        14: ID, name= k
        14: =
        14: ID, name= i
        14: ;
        14: }
15:             i = i + 1;
        15: ID, name= i
        15: =
        15: ID, name= i
        15: +
        15: NUM, val= 1
        15: ;
16:     }
        16: }
17:     return k;
        17: reserved word: return
        17: ID, name= k
        17: ;
18: }
        18: }
19:
20: void sort( int a[], int low, int high)
        20: reserved word: void
        20: ID, name= sort
        20: (
        20: reserved word: int
        20: ID, name= a
        20: [
        20: ]
        20: ,
        20: reserved word: int
```

```
        20: ID, name= low
        20: ,
        20: reserved word: int
        20: ID, name= high
        20: )
21: {   int i; int k;
        21: {
        21: reserved word: int
        21: ID, name= i
        21: ;
        21: reserved word: int
        21: ID, name= k
        21: ;
22:     i = low;
        22: ID, name= i
        22: =
        22: ID, name= low
        22: ;
23:
24:     while (i < high-1)
        24: reserved word: while
        24: (
        24: ID, name= i
        24: <
        24: ID, name= high
        24: -
        24: NUM, val= 1
        24: )
25:     {       int t;
        25: {
        25: reserved word: int
        25: ID, name= t
        25: ;
26:             k = minloc(a,i,high,i);
        26: ID, name= k
        26: =
        26: ID, name= minloc
        26: (
        26: ID, name= a
        26: ,
        26: ID, name= i
        26: ,
        26: ID, name= high
        26: ,
        26: ID, name= i
        26: )
        26: ;
27:             t = a[k];
        27: ID, name= t
        27: =
        27: ID, name= a
        27: [
        27: ID, name= k
        27: ]
        27: ;
28:             a[k] = a[i];
        28: ID, name= a
        28: [
        28: ID, name= k
```

```
        28: ]
        28: =
        28: ID, name= a
        28: [
        28: ID, name= i
        28: ]
        28: ;
29:             a[i] = t;
        29: ID, name= a
        29: [
        29: ID, name= i
        29: ]
        29: =
        29: ID, name= t
        29: ;
30:             i = i + 1;
        30: ID, name= i
        30: =
        30: ID, name= i
        30: +
        30: NUM, val= 1
        30: ;
31:     }
        31: }
32: }
        32: }
33:
34: void main(void)
        34: reserved word: void
        34: ID, name= main
        34: (
        34: reserved word: void
        34: )
35: {   int i;
        35: {
        35: reserved word: int
        35: ID, name= i
        35: ;
36:     i = 0;
        36: ID, name= i
        36: =
        36: NUM, val= 0
        36: ;
37:     while (i < 10)
        37: reserved word: while
        37: (
        37: ID, name= i
        37: <
        37: NUM, val= 10
        37: )
38:     {       x[i] = input();
        38: {
        38: ID, name= x
        38: [
        38: ID, name= i
        38: ]
        38: =
        38: ID, name= input
        38: (
```

```
        38: )
        38: ;
39:             i = i + 1; }
        39: ID, name= i
        39: =
        39: ID, name= i
        39: +
        39: NUM, val= 1
        39: ;
        39: }
40:     sort(x,0,10);
        40: ID, name= sort
        40: (
        40: ID, name= x
        40: ,
        40: NUM, val= 0
        40: ,
        40: NUM, val= 10
        40: )
        40: ;
41:     i = 0;
        41: ID, name= i
        41: =
        41: NUM, val= 0
        41: ;
42:     while (i < 10)
        42: reserved word: while
        42: (
        42: ID, name= i
        42: <
        42: NUM, val= 10
        42: )
43:     {       output(x[i]);
        43: {
        43: ID, name= output
        43: (
        43: ID, name= x
        43: [
        43: ID, name= i
        43: ]
        43: )
        43: ;
44:             i = i + 1; }
        44: ID, name= i
        44: =
        44: ID, name= i
        44: +
        44: NUM, val= 1
        44: ;
        44: }
45: }
        45: }
```