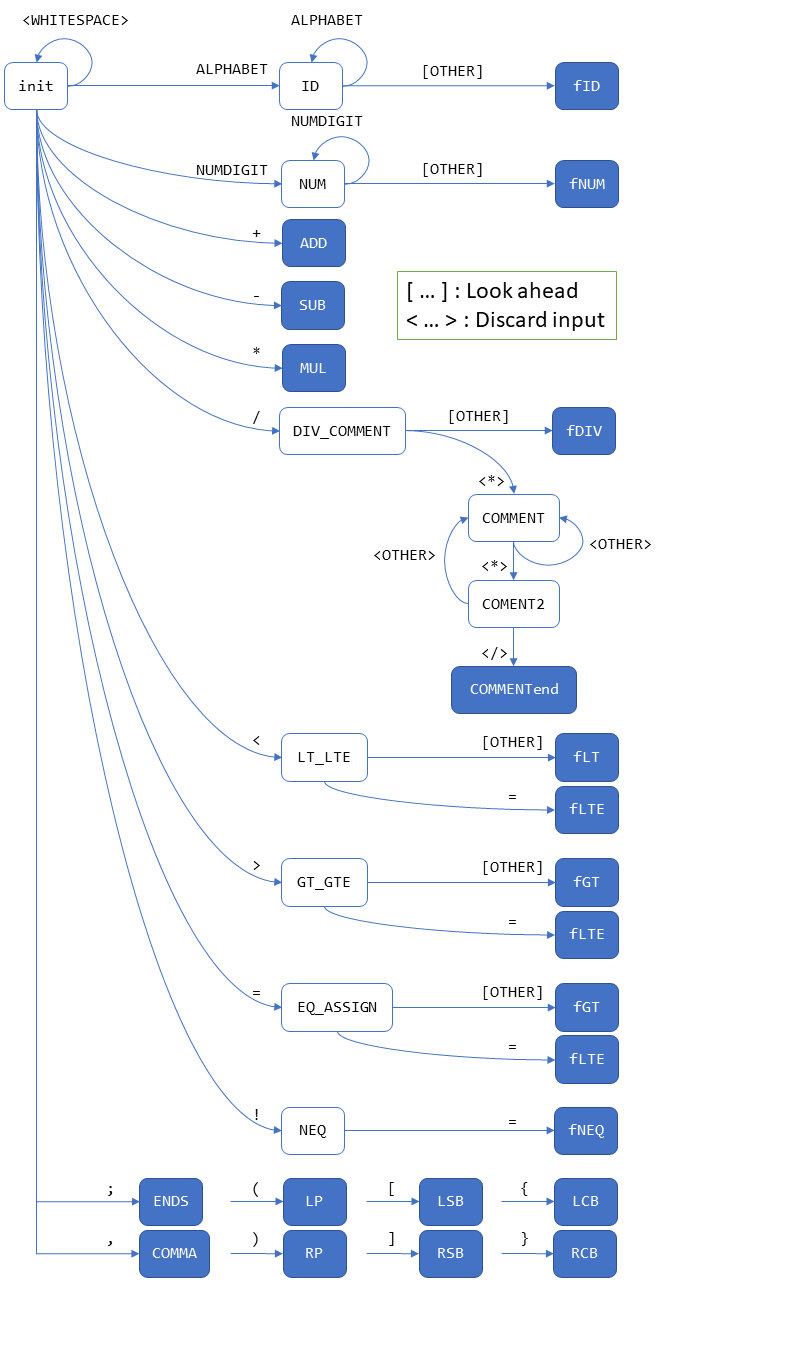
Compiler COMP032001  
Scanner / 2016112905 김민섭

# **1. DFA**

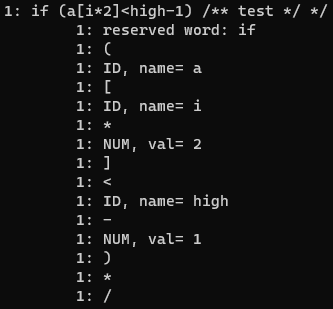


# **2. Scanning Process**

## **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. |

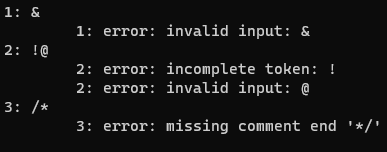
* 스캐닝 결과 콘솔화면:



## **2.1. 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:



# **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 {  ...      void addState(string name, StateData data = StateData::nonf) {  *// 새로운 state를 검색 이름과 함께 생성합니다.*          states[name] = new SingleState(static\_cast<int>(data));      }      void mapState(string from, string to,          string instring,          TransitionOption opt = TransitionOption::optNORMAL) {  *// state 사이에 transition을 만듭니다.*  *// 입력 문자를 코딩 편의성을 위해 스트링으로 한번에 처리합니다.*          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 구축*          buildState();          buildInit();          buildID();          buildNUM();          buildDIV\_COMMENT();          buildLT\_LTE();          buildGT\_GTE();          buildEQ\_ASSIGN();          buildNEQ();          currentState = states["init"];      }  ...  }; |

### **3.2.2. buildState()**

|  |
| --- |
| void buildState() {  *// 오토마타의 init과 각 REX의 시작이 되는 state들을 추가하는 과정입니다.*  *// StateData의 기본값은 StateData::nonf 입니다.*      addState("init");      addState("ID");      addState("NUM");      addState("ADD", StateData::fADD);      addState("SUB", StateData::fSUB);      addState("MUL", StateData::fMUL);      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() {  *// init과 각 REX의 시작 state들 사이에 transition을 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() {      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() {      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()**

|  |
| --- |
| *// processing: /, /\*\*/*  void buildDIV\_COMMENT() {      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); *// ignore comments*      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()**

|  |
| --- |
| *// processing: <, <=*  void buildLT\_LTE() {      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()**

|  |
| --- |
| *// processing: >, >=*  void buildGT\_GTE() {      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()**

|  |
| --- |
| *// processing: ==, =*  void buildEQ\_ASSIGN() {      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()**

|  |
| --- |
| *// processing: !=*  void buildNEQ() {      addState("fNEQ", StateData::fNEQ);      mapState("NEQ", "fNEQ", "=");  } |

## **3.3. processChar() in Pseudo-Code**

|  |
| --- |
| TokenType processChar () {      in ← getChar ()      detectNewLine (&newline, in)      flushBufferIf (flushFlag)      nextState, opt ← pushCharToDFA (in)      if nextState==nullptr then      errorType, flushFlag ← examineError (currentState, in)      return ERROR      processTransitionOption (opt, in)      stateData ← nextState.getData ()      tokenType, currentState, flushFlag ← examineNextState (stateData)      return tokenType  } |

# **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: } |