[제 10 주]

# Hash Table로 구현하는 Dictionary

강 지 훈
jhkang@cnu.ac.kr

충남대학교 컴퓨터공학과

# 개요

# 목표

- Dictionary의 개념
- Hash Table로 Dictionary를 구현
- Lexical Scanner 사용법

# 문제

- 주어진 파일에서 identifier 들을 추출하여 Dictionary 에 관리한다.
  - 처음 나오는 것은 삽입
  - 두 번째 이후는 사용 빈도를 +1.
  - 파일을 모두 처리 후, 각 identifier 의 빈도를 출력한다.
    - Loading factor와 평균 synonym 리스트 길이를 출력한다.

- 문제 해결 방식
  - Dictionary는 Hash table로 구현
    - Hashing: Division 방식을 사용
      - 문자를 정수로 보고 모두 더해서 테이블의 크기로 나눈 나머지를 취한다.
    - Overflow 처리: Open Hashing 방식으로
      - bucket다 마다 synonym들의 연결리스트를 만든다.

# Input & Output

- Input:
  - 파일을 입력받는다.
  - Scanner를 사용하여 ID들을 차례로 추출한다.
    - ID의 정의
      - 첫 문자는 알파벳 또는 '_'로 시작한다.
      - 두 번째 이후 문자는 알파벳 또는 숫자 또는 '_'로 구성된다.
      - 최대 길이는 254이다.

- Output:
  - Hash table의 loading factor를 출력한다.
    - (테이블에 저장된 심볼의 수 / 해쉬 테이블의 크기)
  - Hash table의 버킷의 synonym 연결리스트의 평균 길이
    - NULL이 아닌 버킷의 연결 리스트들의 평균 길이
  - 파일에 사용된 각 id들의 빈도를 출력한다.

# main

# main()

```
void  main()
{
    this._appView = new AppView();
    _appView.outputMsg(_appView.MSG_StartProgram);
    while (doesUserWantScanning()) {
        inputFileNameFromKBD();
        if (startScanning()) {
            _appView.outputMsg(_appView.MSG_StartScanning);
            runScanning();
            showResults();
            endScanning();
            _appView.outputMsg(_appView.MSG_EndScanning);
        } else {
            _appView.outputMsg(_appView.MSG_ErrorFileNotFound);
        }
    }
    _appView.outputMsg(_appView.MSG_EndProgram);
}
```

# Class "AppControllor"

# Public Functions

- public AppControllor()
- public boolean doesUserWantScanning()
  - 새로운 파일을 스캔할 것인지를 사용자에게 묻는다.
  - 사용자가 계속하겠다는 의사를 표현하면 TRUE, 아니면 FALSE를 얻는다.
- public void inputFileNameFromKBD()
  - 키보드로부터 사용자가 스캔할 파일의 이름을 입력하게 한다.
  - 입력된 파일이름은 class 내부에 보관한다.
- public boolean startScanning()
  - 파일 스캔 준비를 한다.
  - 파일이 존재하면 TRUE, 아니면 FALSE를 얻는다.
    - TRUE인 경우에는, scan 종료시 endScanning()으로 마무리 지어야 한다.
- public void runScanning()
  - 파일을 스캔한다.
- public void showResults()
  - 스캔 결과를 보여준다.
- public void endScanning()
  - 스캔을 마무리 짓는다

# Class "AppControllor"의 구현

- Class App에서 사용하는 상수
  - private static final int MaxFileNameLength = 254;
- Class App의 attribute
  - Public attributes: 없음
    - public attribute는 사용하지 않는다.
  - Private attributes:
    - private AppView _appView;
    - private String _fileName; // 스캔할 파일의 이름
    - private BufferedInputStream _file; // 스캔할 파일을 open하여 저장
    - private Dictionary _dictionary; // ID들을 저장할 딕셔너리 (symbol table)
    - Scanner _scanner;

# Public Function의 구현: 생성자

- public AppControllor()
  - 파일 이름과 파일 포인터 초기화
    - _fileName = null;
    - _file = null
  - 딕셔너리를 NULL로 초기화.
    - _dictionary = null;
  - 스캐너를 NULL로 초기화
    - _scanner = null ;

# 입력 처리

- public boolean doesUserWantScanning()
  - 파일을 스캔할 것인지를 사용자에게 (Y/N)으로 물어본다.
  - 'Y' 또는 'y' 또는 'N' 또는 'n'이 아니면 다시 물어본다.
  - 'Y' 또는 'y'이면 TRUE, 'N' 또는 'n'이면 FALSE를 return 한다.

- public void inputFileNameFromKBD()
  - 스캔할 파일의 이름을 입력하라는 메시지를 내보낸다.
  - 파일 이름을 입력받아, _fileName에 저장한다

# 스캔의 시작과 끝

- public boolean startScanning()
  - 스캔할 파일을 open 한다.
    - 정상적으로 open 되면, 스캐너와 딕셔너리를 생성
    - _scanner = new Scanner(_file) ;
      - 스캐너 생성시, 스캔 파일 즉 읽어 들일 파일의 파일 포인터를 넘겨준다.
    - _dictionary = new Dictionary() ;
  - 정상적으로 open 되면 TRUE, 그렇지 않으면 FALSE를 return 한다.
    - 파일이 존재하지 않으면 FALSE가 될 것이다.

- public void endScanning()
  - 스캔 파일을 close 한다.

# 결과 보기

```
public void showResults()
{
    String printMsg = "File Name: " + _fileName;
    _appView.outputMsg(printMsg);

    float      loadingFactor = _dictionary.loadingFactor() ;
     printMsg = "Loading Factor: " + loadingFactor;
    _appView.outputMsg(printMsg);

    floataverageSynonymListLength = _dictionary.averageSynonymListLength() ;
     printMsg = "Average Synonym List Length:" + averageSynonymListLength;
    _appView.outputMsg(printMsg);

    showAllKeysAndItems () ;
}
```

# 스캔

```
public void runScanning()
{
    try {
        int  newItemValue ;
        Key key = null ; // 모든 객체 포인터는 선언과 동시에  초기화
        Item     item = null ; // 모든 객체 포인터는 선언과 동시에  초기화
        Token token = _scanner.nextToken(); // 첫번째 token을 얻어옴
        while ( token != null ) {
            if (token.tokenType() == token.tokenTypeID()) {
                key = keyFromToken(token) ;
                if ( _dictionary.doesKeyExist(key) ) {
                    item = this._dictionary.itemForKey(key);
                    newItemValue = item.count() + 1 ;
                    _dictionary.replaceItemValueForKey(newItemValue, key);
                } else {
                    item = new item(1); // 생성
                    _dictionary.insertKeyAndItem(key, item) ; // 저장
                }
            }
            token = _scanner.nextToken() ; // 새로운  next token 을 얻어옴
        }
    } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
    }
}
```

# Private Functions

- private Key keyFromToken (Token token)
- private void showToken(Token token)
- private void showAllKeysAndItems ()

# Private Functions

- private Key keyFromToken (Token token)

```
// token으로부터 key를 생성하여 돌려준다
{
        String  tokenValue = token.tokenValue() ; // 문자열을 복사해 얻어옴
        Key key = new Key();
        key.setValue(tokenValue);
        return  key ;
}
```

- private void  showToken(Token token)

```
{
        String  tokenValue = token.tokenValue() ;
        String printMsg = "Token: ₩" " + tokenValue +"₩" of Type( " + token.typeNameString() + ")" ;
        _appView.outputMsg(printMsg);
}
```

- private void  showAllKeysAndItems ()

```
{
        Item item = null;
        String keyValue = "";

        Dictionary.DictionaryKeyIterator dicKeyIterator = _dictionary.dictionaryKeyIterator();
        dicKeyIterator.begin();
        int  itemCount = 0 ; // key와 item가 출력되는 순서를 보여주기 위해 사용
        Key key = dicKeyIterator.nextKey(); // key  객체의  reference 만 얻어옴
        while (key != null) {
                keyValue = key.value();
                item = _dictionary.itemForKey(key);
                System.out.println(.....); //출력 할 내용을 넣으세요.
                itemCount++ ;
                key = dicKeyIterator.nextKey();
        }
}
```

# Class "AppView"

# Message 관리

## private attribute

```
private Scanner _scanner;
public String MSG_StartProgram = "<<파일 스캔 프로그램을 시작합니다>>";
public String MSG_EndProgram = "<<프로그램을 종료합니다>>";
public String MSG_StartScanning = "<파일 스캔을 시작합니다>";
public String MSG_EndScanning = "<주어진 파일의 스캔을 종료합니다>";
public String MSG_ErrorFileNotFound = "Error: 파일이 존재하지 않습니다";
```

# Public Functions

- public AppView()

- public void showErrorMsg()
- public void outputMsg(String aString)

- public String inputString()
- public int inputInt()

# Class "Dictionary"

# Public Functions

- public Dictionary()
  - 생성자
- public boolean doesKeyExist (Key aKey)
  - 주어진 key가 딕셔너리에 존재하면 TRUE, 아니면 FALSE를 얻는다.
- public Item itemForKey (Key aKey)
  - 주어진 key를 갖는 객체를 얻는다.
  - 객체가 존재하지 않으면 NULL을 얻는다.
- public void insertKeyAndItem(Key akey, Item anItem)
  - 주어진 키가 이미 존재하면 FALSE를 얻는다
  - 주어진 키 aKey와 객체 anItem를 딕셔너리에 넣는다. TRUE를 넣는다.
- public boolean replaceItemValueForKey(int newItemValue, Key aKey)
  - 주어진 키가 존재하지 않으면, FALSE를 얻는다.
  - 키가 존재하면, 주어진 aKey를 갖는 객체의 값을 주어진 newItemValue로 대체한다.
  - 주어진 객체 item의 key를 갖는 딕셔너리에 이미 있는 객체를 주어진 item로 대체한다.
    - 즉, item의 속성 값을 수정하는 일을 하게 된다.
    - (이 응용에서는 count의 값이 1 증가하게 된다.)
- public float loadingFactor()
  - loading factor를 얻는다.
- public float averageSynonymListLength()
  - 평균 synonym 길이를 얻는다.

# Class "Dictionary"의 구현

## - Hash Table 로 -

- Class Private Constants
  - private static final int HashTableSize = 193 // Prime number
  - private static final int MaxIDLength = 255

- Class Dictionary의 attribute (Instance Variables)
  - Public attributes: 없음
    - public attribute는 사용하지 않는다.
  - Private attributes:

    private Node [] _bucket ; // hash table
    private int _hashTableSize; // HashTableSize로 초기화
    private int _numOfItems ; // hash table에 들어 있는 심볼의 개수
    private int _numOfNonEmptyBuckets ; // 비어 있지 않은 버킷의 개수

# Public Function의 구현: 생성자와 소멸자

- public Dictionary()

```
{
    this._hashTableSize = HashTableSize;
    this._numOfItems = 0;
    this._numOfNonEmptyBuckets = 0;
    this._bucket = new Node[this._hashTableSize];
}
```

# Public Function의 구현

```
public boolean doesKeyExist (Key aKey)
{
        int hashIndex = hash(aKey.value()) ;
        Node node = _bucket[hashIndex] ;
        while ( node != null) {
                ..... // 여기를 채우시오 (이 while loop 안에서 찾게 될 것임)
        }
        return  false ; // 못 찾은 경우
}
```

```
public Item itemForKey (Key aKey)
{
        //  aKey 값을 갖는 item를 copy 하여 돌려준다.
         ..... // 여기를 채우시오
}
```

```
public void insertKeyAndItem(Key key, Item item)
{
        ..... // 키와 Item을 삽입한다.
}
```

```
public void  replaceItem (Item anItem, Key aKey)
{
        // 원래 존재하던 객체는 소멸시키고, 새로운 객체로 aKey의 객체로 설정한다.
        ..... // 여기를 채우시오
}
```

# Hash table

## 합산변환

| identifier | Additive Transformation | x | Hash |
|---|---|---|---|
| For | 102+111+114 | 327 | 2 |
| Do | 100+111 | 211 | 3 |
| While | 119+104+105+108+101 | 537 | 4 |
| If | 105+102 | 207 | 12 |
| Else | 101+108+115+101 | 4256 | 9 |
| function | 102+117+110+99+116+105+111+110 | 870 | 12 |

# Hash table

## 선형 조사법

| | |
|---|---|
| [0] | Function |
| [1] | |
| [2] | For |
| [3] | Do |
| [4] | While |
| [5] | |
| [6] | |
| [7] | |
| [8] | |
| [9] | Else |
| [10] | |
| [11] | |
| [12] | If |

# Private Function의 구현

- private int hash (String idString)

```
{
    int sum = 0 ;
    int   index = 0;
    char [] id = idString.toCharArray();
    while ( id[index] != '₩0' ) {
        sum = sum + (int) id[index] ;
        index++;
    }
    return  (sum % _hashTableSize);
}
```

# Public Function의 구현: 통계와 출력

- public float loadingFactor()
  {
  ..... // 여기를 채우시오
  }

- public float averageSynonymListLength()
  {
  ..... // 여기를 채우시오
  }

# Class "DictionaryKeyIterator"

# Public & private Functions

- 딕셔너리의 모든 키를 차례로 얻는다
- public DictionaryKeyIterator dictionaryKeyIterator()

- private DictionaryKeyIterator()
  - 생성자

- public void begin()
  - Iterator를 초기화 한다.

- public boolean nextNonEmptyBucket()
  - 비어있지 않은 다음 bucket을 찾는다.

- public Key nextKey()
  - 다음 key를 얻는다
  - 새로운 객체가 생성되는 것은 아니고, 단지  reference 만 얻는다.
  - 다 소진되어 더 이상 존재하지 않으면 NULL을 얻는다.

# Class "DictionaryKeyIterator"의 구현

# DictionaryKeyIterator의 구현

- Private member variables :

```
private int _currentBucketIndex  ;
private Node _availableNode;
// currentBucketIndex 가 가리키는  bucket의 노드 리스트 중에서
// 다음  key 를 얻기 위해 사용할 노드를 가리킨다.
// 이 값이 NULL 이면, 리스트의 맨 끝이므로 다음 bucket을 찾아가야 한다.
// 반복을 시작하기 위해, 0 번째 bucket 의 첫번째 노드로 초기화 된다.
```

- 생성자와 소멸자

```
private DictionaryKeyIterator()
{
    _currentBucketIndex = 0 ;
    _availableNode = _bucket[0];
}
```

# DictionaryKeyIterator의 구현

- begin과 nextKey

```
public void begin()
{
    _currentBucketIndex = 0 ;
    _availableNode = _bucket[0] ;
}

public Key nextKey()
{
    if (_availableNode == null) {
        if (! nextNonEmptyBucket()) {
            return  null; // No more NonEmpty bucket, so no more key.
        }
        _availableNode = _bucket[_currentBucketIndex] ;
    }
    Key nextKey = _availableNode.key() ;
    _availableNode = _availableNode.next();
    return  nextKey ;
}
```

# Class "Node"

# Node class

```
// Node for synonym lists in a Hash Table
private Key _key;
private Node _next;
private Item _item;

public Node()
public Node(Key givenKey, Node givenNode, Item givenObject)
public Key key()
public void setKey(Key aKey)
public Node next()
public void setNext(Node aNode)
public Item item()
public void setItem(Item anItem)
```

# Class "Key"

# Class "Key"의 선언

- Public Functions
  - public Key()
  - public Key(String givenKey)
  - public void setValue(String aValue)
  - public String value()
  - public int compareTo(Object arg0)
    - _this와 otherKey의 key value 값을 비교
    - 같으면 0, _this의 key value가 작으면 -1, _this의 key value가 크면 +1을 얻는다

# Class "Key"의 구현

- Instance Variables
  - private String _value;

- 공개함수의 구현

```
public Key()
{
    _value = null;
}
```

# value의 getter/setter 구현

- Setter 구현

```
public void setValue(String aValue)
{
      this._value = aValue;
}

public String value()
{
      return this._value;
}

public int compareTo(Object arg0)
{
     Key aKey = (Key)arg0;
     int index = 0;
     while(_value.charAt(index) == aKey.value().charAt(index)){
          if(_value.charAt(index) == '₩0')
               return 0;
          index++;
     }
     return (_value.charAt(index) - aKey.value().charAt(index));
}
```

# Class "Item"

# Class "Item"의 선언

- Public Functions
  - public Item()
  - public Item(int givenCount)
  - public int count()
  - public void setCount(int aCount)
  - public void incrementCount()

# Class "Object"의 구현

- Instance Variables
  - private int _count;

- 공개함수의 구현
  - 선언에 맞도록 구현

# Class "Scanner"
# &
# Class Token

# Scanner와 Finite State Automata

- 입력은?
  - 문자들:
    - (예) 파일에 저장된 순서대로
- 출력은?
  - 토큰들: 인식되는 순서대로
- 토큰이란?
  - 의미 있다고 정의되는 문자열
    - (예) Id, 정수, 실수, 연산자, 주석문, ……

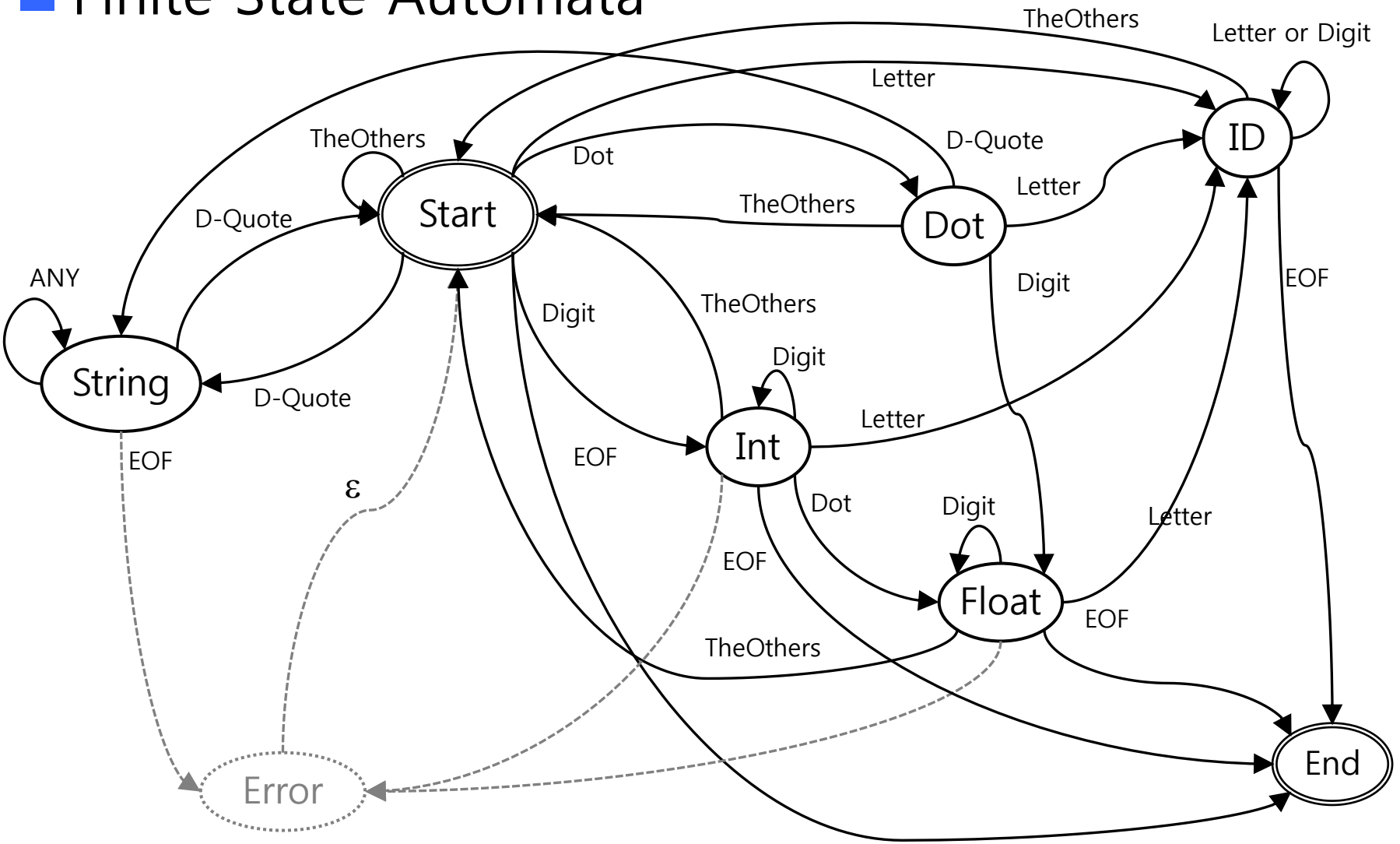
# Class "Scanner"

# Class "Scanner"

- Public Functions
  - public Scanner(BufferedInputStream givenDataInputStream)
    - 생성자
    - 주어진 파일 포인터로 파일 스캔 작업을 하기 위한 객체를 생성하고 초기화 한다.
  - public Token nextToken() throws IOException
    - Token 하나를 얻는다.
    - 파일 끝을 만나 더이상 토큰을 얻을 수 없으면, NULL 을 얻는다.

# Class "Scanner"의 구현

## Finite State Automata



# State-Transition Table

| State \ Input | Letter | Digit | Dot | Sign | Double Quote | Return | The Others | EOF |
|---|---|---|---|---|---|---|---|---|
| Start | ID | Int | Dot | Start | String | Start | Start | End |
| ID | ID | ID | Start | Start | String | Start | Start | End |
| Int | ID | Int | Float | Start | String | Start | Start | End |
| Float | ID* | Float | Start | Start | String | Start | Start | End |
| String | String | String | String | String | Start | Start | String | End |
| Dot | ID | Float | Dot | Start | String | Start | Start | End |
| End | | | | | | | | |

- 문자 타입
  - Letter : 'a'~'z', 'A'~'Z', '_'
  - Digit : '0'~'9'
  - Dot : '.'
  - Sign : '+', '-'
  - DoubleQuote : '"'
  - TheOthers : 그 외 나머지 모든 문자
  - EOF: End Of File을 만났을 때를 나타내는 문자 타입

- ID* : Float는 지수가 없는 형태만 인식
  (예) 3.5E-10 은 인식 불가능
  여기서는, "3.5", "E", "-", "10" 이렇게 나뉘어 인식됨
- Single Quote는 인식 불가능
  (예) 's'는 "'", "s", "'"로 나뉘어 인식됨
- Dot과 Sign만 연산자로 인식하되, 단순히 Delimiter의 역할로 한정
- 그 밖의 연산자 역시 단순히 Delimiter의 역할로 한정
- 주석문(comment) 인식 불가능: "//", "/* ... */"

# Action Table

| State \ Input | Letter | Digit | Dot | Sign | Double Quote | Return | The Others | EOF |
|---|---|---|---|---|---|---|---|---|
| Start | IDBegin | IntBegin | FloatBegin | None | StringBegin | None | None | None |
| ID | AddToToken | AddToToken | TokenEnd | TokenEnd | TokenEnd, StringBegin | TokenEnd | TokenEnd | TokenEnd |
| Int | TokenEnd, IDBegin | AddToToken | AddToToken, IntToFloat | TokenEnd | TokenEnd, StringBegin | TokenEnd | TokenEnd | TokenEnd |
| Float | TokenEnd, IDBegin | AddToToken | TokenEnd | TokenEnd | TokenEnd, StringBegin | TokenEnd | TokenEnd | TokenEnd |
| String | AddToToken | AddToToken | AddToToken | AddToToken | TokenEnd | Error (TokenEnd) | AddToToken | Error (TokenEnd) |
| Dot | IDBegin | FloatBegin | None | None | StringBegin | None | None | None |
| End | | | | | | | | |

- Action
  - AddToToken: 현재 상태의 토큰 문자열 끝에 입력 문자를 붙인다.
  - None: 어떠한 action도 취할 필요가 없다.

# Token의 type을 더 일반화 한다면?

- Single Character:
  - 'a', '\n', '\0'
- 지수 표현을 갖는 Floating Point Number는?
  - 3.05+2, 2.421E-13, 0.025E10
- 문자열 안의 '\'
  - "문자열: \"abc\" ."
- 주석문은?
  - //
  - /* ... */

# Class "Scanner"의 구현

```
public class Scanner {
        private static final int NumOfStates = 7;
        private static final int NumOfCharTypes = 8;

        enum CharType {
                CHAR_Letter, CHAR_Digit, CHAR_Dot, CHAR_Sign, CHAR_DoubleQuote, CHAR_Return, CHAR_TheOthers, CHAR_EOF
        };

        // typedef enum {TOKEN_ID, TOKEN_Int, TOKEN_Float, TOKEN_String} TokenType ;
        // // Token 에서 선언
        enum State {
                STATE_Start, STATE_ID, STATE_Int, STATE_Float, STATE_String, STATE_Dot, STATE_End
        };

        enum Action {
                ACTION_IDBegin, ACTION_IntBegin, ACTION_FloatBegin, ACTION_StringBegin, ACTION_TokenEnd, ACTION_TokenEndIDBegin,
ACTION_TokenEndStringBegin, ACTION_AddToToken, ACTION_AddToTokenIntToFloat, ACTION_None
        };

        State Scanner_stateTransitionTable[][] = {
                        { State.STATE_ID, State.STATE_Int, State.STATE_Dot,
                                State.STATE_Start, State.STATE_String, State.STATE_Start,
                                State.STATE_Start, State.STATE_End },
                        { State.STATE_ID, State.STATE_ID, State.STATE_Start,
                                State.STATE_Start, State.STATE_String, State.STATE_Start,
                                State.STATE_Start, State.STATE_End },
                        { State.STATE_ID, State.STATE_Int, State.STATE_Float,
                                State.STATE_Start, State.STATE_String, State.STATE_Start,
                                State.STATE_Start, State.STATE_End },
                        { State.STATE_ID, State.STATE_Float, State.STATE_Start,
                                State.STATE_Start, State.STATE_String, State.STATE_Start,
                                State.STATE_Start, State.STATE_End },
                        { State.STATE_String, State.STATE_String, State.STATE_String,
                                State.STATE_String, State.STATE_Start, State.STATE_String,
                                State.STATE_Start, State.STATE_End },
                        { State.STATE_ID, State.STATE_Float, State.STATE_Dot,
                                State.STATE_Start, State.STATE_String, State.STATE_Start,
                                State.STATE_Start, State.STATE_End } };
```

# Class "Scanner"의 구현

```
Action Scanner_actionTable[][] = {
        { Action.ACTION_IDBegin, Action.ACTION_IntBegin,
                Action.ACTION_FloatBegin, Action.ACTION_None,
                Action.ACTION_StringBegin, Action.ACTION_None,
                Action.ACTION_None, Action.ACTION_None },
        { Action.ACTION_AddToToken, Action.ACTION_AddToToken,
                Action.ACTION_TokenEnd, Action.ACTION_TokenEnd,
                Action.ACTION_TokenEndStringBegin, Action.ACTION_TokenEnd,
                Action.ACTION_TokenEnd, Action.ACTION_TokenEnd },
        { Action.ACTION_TokenEndIDBegin, Action.ACTION_AddToToken,
                Action.ACTION_AddToTokenIntToFloat, Action.ACTION_TokenEnd,
                Action.ACTION_TokenEndStringBegin, Action.ACTION_TokenEnd,
                Action.ACTION_TokenEnd, Action.ACTION_TokenEnd },
        { Action.ACTION_TokenEndIDBegin, Action.ACTION_AddToToken,
                Action.ACTION_TokenEnd, Action.ACTION_TokenEnd,
                Action.ACTION_TokenEndStringBegin, Action.ACTION_TokenEnd,
                Action.ACTION_TokenEnd, Action.ACTION_TokenEnd },
        { Action.ACTION_AddToToken, Action.ACTION_AddToToken,
                Action.ACTION_AddToToken, Action.ACTION_AddToToken,
                Action.ACTION_TokenEnd, Action.ACTION_AddToToken,
                Action.ACTION_TokenEnd, Action.ACTION_TokenEnd },
        { Action.ACTION_IDBegin, Action.ACTION_FloatBegin,
                Action.ACTION_None, Action.ACTION_None,
                Action.ACTION_StringBegin, Action.ACTION_None,
                Action.ACTION_None, Action.ACTION_None }, };
```

# Class "Scanner"의 구현

- Instance Variables

```
private BufferedInputStream filename;
private State currentState;
private boolean tokenIsAvailable;
private Token availableToken;
private Token token;
```

# Class "Scanner"의 구현: 생성자와 소멸자

- public Scanner(BufferedInputStream givenDataInputStream)

```
{
    // 생성자
    this.filename = givenDataInputStream;
    this.currentState = State.STATE_Start;
    this.tokenIsAvailable = false;
    this.availableToken = null;
    this.token = null;
}
```

- public Token nextToken() throws IOException

```
{
    // 파일을 스캔하여, 얻은 Token 을 return 한다.
    // 파일에 더 이상 Token이 없으면 NULL을 return 한다.
    while (!(this.currentState == State.STATE_End || this.tokenIsAvailable))
        transitToNextStateAndTakeAction();
    if (this.tokenIsAvailable) {
        // Token이 생산 완료된 시점에 TRUE가 되고,
        // 소비가 되는 시점인 여기에서 FALSE로 바뀐다.
        this.tokenIsAvailable = false;
        return this.availableToken;
    } else
        return null;
}
```

# Class "Scanner"의 Private Functions

- private void transitToNextStateAndTakeAction()
  - 파일로부터 문자 하나를 입력 받아, 상태를 전이(변경)시키고, 필요한 일을 한다.
- private String readFileChar()
  - 현재 가지고 있는 파일로부터 문자 하나를 읽어서 얻는다:
  - 파일 끝을 만나면, EOF 값을 얻는다.
- private CharType  charType(String inputString)
  - 주어진 문자의 CharType을 얻는다.

```
private void transitToNextStateAndTakeAction() throws IOException
{
    String inputChar = readFileChar();
    CharType inputCharType = charType(inputChar);
    State nextState = Scanner_stateTransitionTable[currentState.hashCode()][inputCharType
            .hashCode()];
    Action actionToBeTaken = Scanner_actionTable[currentState.hashCode()][inputCharType
            .hashCode()];
    switch (actionToBeTaken) {
        case ACTION_IDBegin:
            …
            break ;
        case ACTION_IntBegin:
            …
            break ;
        …

        case ACTION_None:
            break ;
    }
     currentState = nextState;
}
```

```
 case ACTION_IDBegin:
        this.token.clearValue();
        this.token.setType(this.token.tokenTypeID()) ;
        this.token.addChar(inputChar);
        break;

 case ACTION_IntBegin:
        this.token.clearValue();
        this.token.setType(this.token.tokenTypeInt()) ;
        this.token.addChar(inputChar);
        break;

case ACTION_FloatBegin :
        this.token.clearValue();
        this.token.setType(this.token.tokenTypeFloat()) ;
        this.token.addChar(inputChar);
        break ;

case ACTION_StringBegin :
        this.token.clearValue();
        this.token.setType(this.token.tokenTypeString()) ;
        // inputChar는 문자열 시작을 알리는 Double Quote를 가지고 있으므로,
        // inputChar는 addChar 할 필요가 없음
        break ;
```

```
case ACTION_TokenEnd:
      availableToken = this.token.copy();
      tokenIsAvailable = true;
      break;

case ACTION_TokenEndIDBegin:
      availableToken = this.token.copy();
      tokenIsAvailable = true;
      this.token.clearValue();
      this.token.setType(this.token.tokenTypeID()) ;
      this.token.addChar(inputChar);
      break ;

case ACTION_TokenEndStringBegin:
      availableToken = this.token.copy();
      tokenIsAvailable = true;
      this.token.clearValue();
      this.token.setType(this.token.tokenTypeString()) ;
      // inputChar는 문자열 시작을 알리는 Double Quote를 가지고 있으므로,
      // inputChar는 addChar 할 필요가 없음
      break ;
```

```
case ACTION_AddToToken:
    this.token.addChar(inputChar);
    break ;

case ACTION_AddToTokenIntToFloat:
    this.token.addChar(inputChar);
    this.token.setType(this.token. tokenTypeFloat()) ;
    break ;

case ACTION_None :
    break ;
```

```
private String readFileChar() throws IOException
{
    int buf;
    buf = this.filename.read();
    return Integer.toString((char)buf);
}

private CharType  charType(String inputString)
{
    //주어진 문자의 CharType을 얻는다.
    char inputChar = inputString.charAt(0);
    if ((inputChar >= 'a' && inputChar <= 'z') || (inputChar >= 'A' && inputChar <= 'Z') ||
    (inputChar == '_'))
    {
        return CharType.CHAR_Letter ;
    } else if ((inputChar >= '0' && inputChar <= '9')) {
        return CharType.CHAR_Digit ;
    } else if (inputChar == '.') {
        return CharType.CHAR_Dot ;
    } else if (inputChar == '+' || inputChar == '-') {
        return CharType.CHAR_Sign ;
    } else if (inputChar == '"') {
        return CharType.CHAR_DoubleQuote ;
    } else if (inputChar == '₩n') {
        return CharType.CHAR_Return ;
    } else if (inputString.endsWith("-1")) {
        return CharType.CHAR_EOF ;
    } else {
        return CharType.CHAR_TheOthers ;
    }
}
```

# Class "Token"

# Class "Token"의 선언

- 필요한 type 선언
  - public enum TokenType {TOKEN_NULL, TOKEN_ID, TOKEN_Int, TOKEN_Float, TOKEN_String};
  - private static final int MAX_TOKENVALUE_LENGTH = 255 ;
  - public HashMap<TokenType, String> TokenTypeMap = new HashMap<TokenType, String>();

# Class "Token"의 선언

- Public Functions
  - 생성자
    - public Token()
    - public Token(TokenType givenTokenType, String givenTokenValue)
    - public void MakeTokenMap()
  - 객체 복사
    - public Token copy()
  - Getter/Setter for Token Type
    - public TokenType  tokenType ()
    - public void setTokenType(TokenType aType)
  - Getter/Setter for Token Value
    - public String tokenValue()
      - value를 반환한다.
    - public void clearValue()
      - Null 문자열로 설정한다.
  - 문자 추가
    - public void addChar(String aChar)
      - 현재 문자열 끝에 주어진 문자를 더한다.
  - 기타
    - public String typeNameString ()
      - 토큰의 type을 출력용으로 사용할 수 있는 type name 문자열을 얻는다.

# Class "Token"의 구현

- Class public Constants
  - public TokenType tokenTypeNULL()
  - public TokenType tokenTypeID()
  - public TokenType tokenTypeInt()
  - public TokenType tokenTypeFloat()
  - public TokenType tokenTypeString()
  - public void MakeTokenMap()

- Instance Variables

```
private TokenType    _tokenType ;
private String _tokenValue;
```

# Class "Token"의 구현

- 공개함수의 구현

```
public Token()
{
    this._tokenType = TokenType.TOKEN_NULL;
    this._tokenValue = null;
    this.MakeTokenMap();
}

public Token(TokenType givenTokenType, String givenTokenValue)
{
    this._tokenType = givenTokenType;
    this._tokenValue = givenTokenValue;
    this.MakeTokenMap();
}

public Token copy()
{
    return new Token(this._tokenType, _tokenValue);
}
```

# Class "Token"의 구현

- 공개함수의 구현

```
public void MakeTokenMap()
{
    TokenTypeMap.put(TokenType.TOKEN_NULL, "NULL");
    TokenTypeMap.put(TokenType.TOKEN_ID, "ID");
    TokenTypeMap.put(TokenType.TOKEN_Int,"Int");
    TokenTypeMap.put(TokenType.TOKEN_Float, "Float");
    TokenTypeMap.put(TokenType.TOKEN_String, "String");
}
public TokenType tokenTypeNULL() { … }
public TokenType tokenTypeID() { … }
public TokenType tokenTypeInt() { … }
public TokenType tokenTypeFloat() { … }
public TokenType tokenTypeString()
{
    return TokenType.TOKEN_NULL;
}
```

# Class "Token"의 구현

```
public String tokenValue()
{
      return this._tokenValue;
}

public void clearValue()
{
      this._tokenValue = null;
}

public void addChar(String aChar)
{
      if (this._tokenValue.length() < MAX_TOKENVALUE_LENGTH-1)
          this._tokenValue = this._tokenValue + aChar;
}
```

# Class "Token"의 구현

```
public TokenType  tokenType ()
{
      return this._tokenType;
}

public void setType(TokenType aType)
{
      this._tokenType = aType;
}

public String typeNameString ()
{
      return TokenTypeMap.get(_tokenType);
}
```

# 요 약

# 생각해 볼 점

- 삽입, 삭제, 대치의 시간 복잡도는?
- showAllKeysAndObjects()의 시간 복잡도는?
  - 효율적으로 처리될 수 있도록 할 수 있는 방안은?

# [문제 10] 요약

- Hash의 이해
  - Hash로 예상 가능한 입력 파일을 작성하여 해당 파일을 기준으로 HashTable을 그려서 제출하세요.

# 과제 제출

# 과제 제출

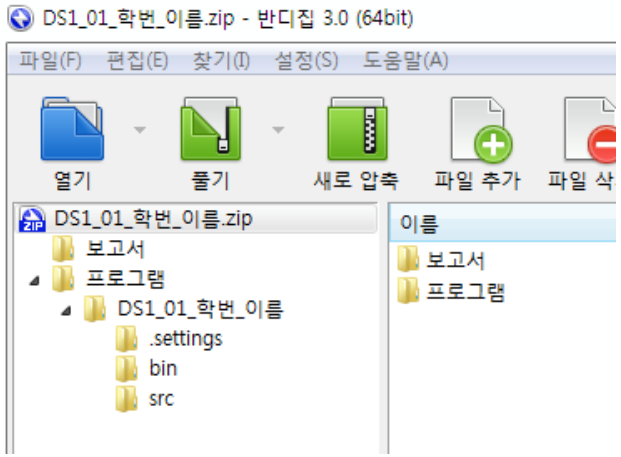
- pineai@cnu.ac.kr
  - 메일 제목 : [0X]DS2_10_학번_이름
    - 양식에 맞지 않는 메일 제목은 미제출로 간주됨
    - 앞의 0X는 분반명 ( 오전10시 : 00반 / 오후4시 : 01반 )

- 제출 기한
  - 11월 12일(화) 23시59분까지
  - 시간 내 제출 엄수

  - 제출을 하지 않을 경우 0점 처리하고, 숙제를 50% 이상 제출하지 않으면 F 학점 처리하며, 2번 이상 제출하지 않으면 A 학점을 받을 수 없다.

# 과제 제출

- 파일 이름 작명 방법
  - DS2_10_학번_이름.zip
  - 폴더의 구성
    - DS2_10_학번_이름
      - 프로그램
        - 프로젝트 폴더 / 소스
        - 메인 클래스 이름 : DS2_10_학번_이름.java
      - 보고서
        - 이곳에 보고서 문서 파일을 저장한다.
        - 입력과 실행 결과는 화면 image로 문서에 포함시킨다.
        - 문서는 pdf 파일로 만들어 제출한다.



# 보고서 작성 방법

- 겉장
  - 제목: 자료구조 실습 보고서
  - [제xx주] 숙제명
  - 제출일
  - 학번/이름
- 내용
  1. 프로그램 설명서
     1. 주요 알고리즘 /자료구조 /기타
     2. 함수 설명서
     3. 종합 설명서 : 프로그램 사용방법 등을 기술
  2. 구현 후 느낀 점 : 요약의 내용을 포함하여 작성한다.
  3. 실행 결과 분석
     1. 입력과 출력 (화면 capture : 실습예시와 다른 예제로 할 것)
     2. 결과 분석

     -------- 표지 제외 3장 이내 작성 --------
  4. 소스코드 : 화면 capture가 아닌 소스를 붙여넣을 것

     소스는 장수 제한이 없음.

# [제 10 주] 끝

삽입, 검색, 대체가 모두 O(1) !