

# Report

제 목: Text Miner

## <과제물 제출 전 체크리스트>

1	이 과제물은 내가/우리가 직접 연구하고 작성한 것이다.	○
2	인용한 모든 자료(책, 논문, 인터넷자료 등)의 인용표시를 바르게 하였다.	○
3	인용한 자료의 표현이나 내용을 왜곡하지 않았다.	○
4	정확한 출처 제시 없이 다른 사람의 글이나 아이디어를 가져오지 않았다.	○
5	정확한 출처 제시 없이 여러 사람의 글이나 아이디어를 짜깁기하지 않았다.	○
6	과제물 작성 중 도표나 데이터를 조작(위조 또는 변조)하지 않았다.	○
7	과제물은 다른 사람으로부터 받거나 구매하여 제출하지 않았다.	○
8	이 과제물에 실질적으로 참여하지 않은 사람을 공동 제출자로 명기하지 않았다.	○
9	이 과제물과 동일한 내용을 다른 교과목의 과제물로 제출한 적이 없다.	○

과목명	IMEN281
담당교수	김병인
학 과	산업경영공학과
학 번	20130220
성 명	이다진
제출일	2017년 10월 5일

## 목차

<b>I. 목적 .....</b>	<b>1</b>
<b>II. 방법과 알고리즘 .....</b>	<b>2</b>
1. Flow Chart.....	2
2. 클래스 구성 및 설명 .....	3
3. 구현 코드 설명.....	5
<b>III. 수행 결과에 대한 토의 .....</b>	<b>9</b>
1. 토론 .....	9
2. 결론 .....	10
<b>IV. 실행화면 스크린샷 .....</b>	<b>11</b>

## I. 목적

### 1. C++의 문법 연습

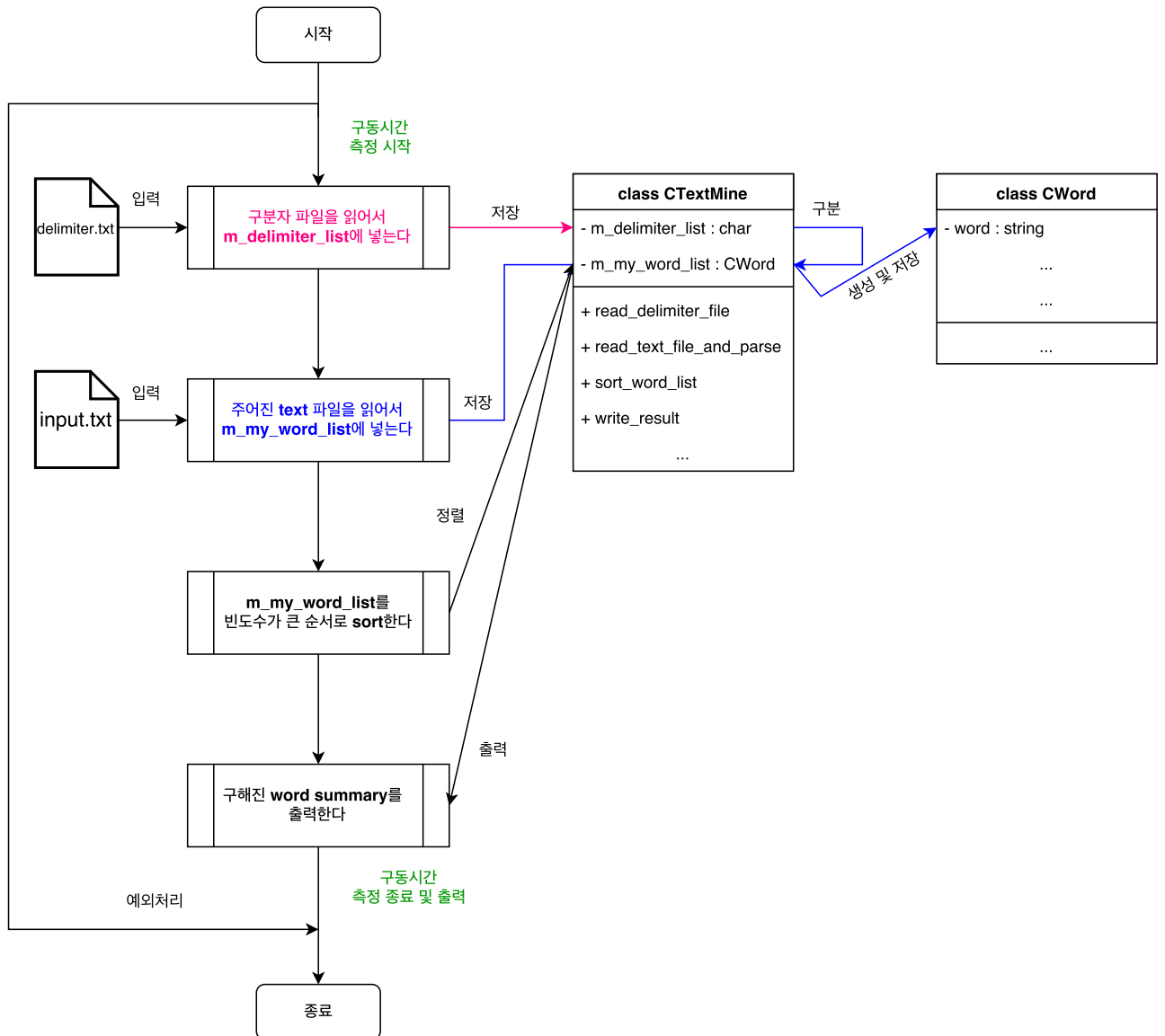
- (1) 여러 가지 클래스와 메소드를 구현함으로써 선언(Declaration)과 정의(Definition)를 구별하는 방법을 익힌다.
- (2) Vector를 사용하는 방법을 익힌다.
- (3) File I/O를 구현하는 방법을 익힌다.
- (4) string 헤더파일에 있는 메소드를 사용하는 방법을 익힌다.

### 2. 기능 구현 연습

- (1) Class를 디자인하는 방법을 익힌다.
- (2) 정렬(sorting)을 구현하는 방법을 익힌다.
- (3) 문장을 구분자(delimiter)에 의해 나누는 방법을 익힌다.
- (4) Main 함수의 인자를 이용하여 프로그램을 실행하는 방법을 익힌다.
- (5) 프로그래밍을 통한 문제 해결 능력을 키운다.

## II. 방법과 알고리즘

### 1. Flow Chart



## 2. 클래스 구성 및 설명

(1) class CWord

```
class CWord {  
    string word;  
    int count;  
public:  
    string getWord() { return word; }  
    int getCount() { return count; }  
    void setWord(string _word) { word = _word; }  
    void setCount(int _count) { count = _count; }  
  
    CWord(string _word, int _count = 1) {  
        word = _word;  
        count = _count;  
    }  
};
```

(설명) CWord는 구분자에 의해 추출된 단어 클래스이다. 생성된 인스턴스는 하나의 단어를 의미한다. 단어가 반복되면 기존 인스턴스의 count를 1 만큼 증가된다. getWord, getCount, setWord, setCount는 private한 word와 count에 접근하기 위한 메소드이다.

(2) class CTextMine

```
class CTextMine {
    vector <char> m_delimiter_list;
    vector <CWord> m_my_word_list;
    long m_total_word_count;
public:
    long getTotalWordCount() { return m_total_word_count; }
    long getWordCount() { return (long)m_my_word_list.size(); }

    void read_delimiter_file(const char *filename);
    void read_text_file_and_parse(const char *filename);
    void sort_word_list();
    void write_result(const char *filename);
    void bubbleSort(vector<int> &indexList);
};
```

(설명) CTextMine은 텍스트 파일의 입력스트림을 처리한다.

- 1) **m\_delimiter\_list**: delimiter.txt 파일에서 추출한 구분자를 저장하는 vector 컨테이너이다.
- 2) **m\_my\_word\_list**: 인풋 텍스트 파일에서 구분자로 인해 추출된 단어의 클래스 CWord의 인스턴트를 저장하는 vector 컨테이너이다.
- 3) **read\_delimiter\_file**: delimiter.txt 파일에서 구분자를 추출하여 저장하기 위한 메소드이다.
- 4) **read\_text\_file\_and\_parse**: 인풋 텍스트 파일에서 구분자로 단어 인스턴스를 생성한다. 반복되면 인스턴스를 생성하지 않고 기존 인스턴스의 count를 1 만큼 증가시킨다.
- 5) **sort\_word\_list**: 단어 인스턴스를 정렬한다. 우선순위는 단어의 빈도수(count 내림차순), 문자열의 값(사전 상 더 앞쪽에 위치하는 순서) 순이다.
- 6) **write\_result**: 인풋 텍스트 파일에서 추출한 단어의 숫자(m\_total\_word\_count)와 정렬된 단어 인스턴스들의 총 개수(m\_my\_list.size())를 화면에 출력한다. 아웃풋 텍스트 파일을 생성하여 단어들의 정보(CWord의 word, count)를 출력한다.
- 7) **bubbleSort**: 인풋 텍스트 파일에서 찾은 구분자의 인덱스를 정렬하기 위한 버블 소트.

### 3. 구현 코드 설명

#### (1) read\_delimiter\_file

```
void CTextMine::read_delimiter_file(const char *filename){
    ifstream fin(filename);
    if ( !fin.is_open() )
        throw -1;

    string _delimiter;
    while ( !fin.eof() ) {
        getline(fin, _delimiter);
        m_delimiter_list.push_back(_delimiter[0]);
    }
    m_delimiter_list.push_back('\n');
}
```

(설명) delimiter.txt 파일을 열고 실패 시 main으로 -1을 throw한다. 파일의 끝까지 getline으로 한 줄을 \_delimiter에 저장하고 개행문자를 제외한 구분자 \_delimiter[0]만 m\_delimiter\_list에 저장한다. delimiter.txt에는 없지만 인풋 텍스트 파일 입력 시 처리하기 곤란한 '\n'도 구분자로 활용하기 위해 리스트에 추가하였다.

#### (2) read\_text\_file\_and\_parse

```
void CTextMine::read_text_file_and_parse(const char *filename){
    ifstream fin(filename);
    if ( !fin.is_open() )
        throw -2;

    string tempLine;
    string _word;
    while ( !fin.eof() ) {
        getline(fin, tempLine, '\r');
        if ( !(tempLine == "" || tempLine == "\r" || tempLine == "\n") ) {
            vector<int> delimiterIndexList;
            delimiterIndexList.push_back(-1);
            for (int i = 0; i < (int)m_delimiter_list.size(); i++) {
                int tempIndex = -1;
                while (true) {
                    tempIndex = (int)tempLine.find(m_delimiter_list[i], tempIndex + 1);
                    if ( tempIndex == -1 )
                        break;
                    delimiterIndexList.push_back(tempIndex);
                }
            }
            bubbleSort(delimiterIndexList);
        }
    }
}
```

- 1) 인풋 텍스트 파일을 오픈하고 실패 시 main으로 -2를 throw한다.
- 2) getline함수로 'Wr' 문자<sup>1</sup> 이전까지의 문자열을 tempLine에 저장한다. 'Wr' 문자가 연속으로 있는 경우(tempLine == "")와 개행문자로 인한 예상치 못한 버그(tempLine == "Wr" || tempLine == "Wn")를 방지하기 위해 if 조건문을 작성하였다.
- 3) tempLine에서 구분자를 찾고 인덱스를 tempIndex에 저장한다. 구분자를 찾을 때마다 tempIndex를 vector 컨테이너 delimiterIndexList에 저장해둔다.
- 4) 2번, 3번의 과정을 파일의 끝을 만날 때까지 반복한다.
- 5) delimiterIndexList에 저장된 구분자의 인덱스를 오름차순으로 정렬한다.

```

bubbleSort(delimiterIndexList);
for (int i = 0; i < (int)delimiterIndexList.size(); i++) {
    _word = tempLine.substr(delimiterIndexList[i]+1, delimiterIndexList[i+1]-delimiterIndexList[i]-1);
    if ( !(_word == "" || _word == "\r" || _word == "\n") ) {
        m_total_word_count++;
        bool isRepeated = false;
        for (int i = 0; i < (int)m_my_word_list.size(); i++) {
            if ( _word.compare(m_my_word_list[i].getWord()) == 0 ) {
                int count = m_my_word_list[i].getCount();
                m_my_word_list[i].setCount(count + 1);
                isRepeated = true;
                break;
            }
        }
        if ( !isRepeated ) {
            m_my_word_list.push_back(CWord(_word));
        }
    }
}
}
}
}
fin.close();
}

```

- 6) delimiterIndexList를 이용하여 구분자 인덱스 사이의 단어를 추출한다.
- 7) 추출한 단어를 m\_my\_word\_list에 있는 기존의 단어와 비교한다.
- 8-1) 기존의 단어가 반복되는 경우, 그 인스턴스의 count를 1만큼 증가시킨다.
- 8-2) 새로운 단어인 경우, CWord에 단어를 저장하며 인스턴스화하고 m\_my\_word\_list에 추가한다.
- 9) 입력 스트림을 클로즈한다.

---

<sup>1</sup> MacOS에서는 'Wr'이 개행문자이다.



### (3) sort\_word\_list

```
void CTextMine::sort_word_list(){
    for (int j = (int)m_my_word_list.size() - 1; j > 0; j--) {
        for (int i = 0; i < j; i++) {
            if (m_my_word_list[i].getCount() < m_my_word_list[i+1].getCount()) {
                CWord temp = m_my_word_list[i];
                m_my_word_list[i] = m_my_word_list[i+1];
                m_my_word_list[i+1] = temp;
            }
            else if (m_my_word_list[i].getCount() == m_my_word_list[i+1].getCount()){
                if (m_my_word_list[i].getWord().compare(m_my_word_list[i+1].getWord()) > 0) {
                    CWord temp = m_my_word_list[i];
                    m_my_word_list[i] = m_my_word_list[i+1];
                    m_my_word_list[i+1] = temp;
                }
            }
        }
    }
}
```

(설명) CWord 객체의 count가 큰 순서로 정렬한다. count가 같은 경우 CWord의 word가 작은 순서로 정렬한다.

### (4) write\_result

```
void CTextMine::write_result(const char *filename){
    ofstream fout(filename);
    if ( !fout.is_open() )
        throw -3;
    fout << "전체 word 수 (중복) = " << getTotalWordCount() << endl;
    fout << "전체 word 수 = " << getWordCount() << endl << endl;
    fout << "Word, count" << endl;
    for (int i = 0; i < getWordCount(); i++)
        fout << m_my_word_list[i].getWord() << ", " << m_my_word_list[i].getCount() << endl;
    fout.close();
}
```

- 1) 아웃풋 텍스트 파일을 열고 실패 시 main으로 -3을 throw한다.
- 2) CTextMine 인스턴스의 멤버 변수 m\_total\_word\_count를 getTotalWordCount 메소드를 이용해 출력한다.
- 3) CTextMine 인스턴스의 멤버 변수 m\_my\_word\_list의 사이즈를 getWordCount 메소드를 이용해 출력한다.
- 4) 모든 CWord 인스턴스의 멤버 변수 word와 count를 메소드를 getWord, getCount 메소드를 이용해 각각 아웃풋 텍스트 파일에 출력한다.
- 5) 출력 스트림을 클로즈한다.

#### (4) main

```
int main(int argc, const char *argv[])
{
    try {
        clock_t    clockA;
        double elapsedTime;
        clockA = clock();

        if (argc != 4) {
            cout << "프로그램 구동 방법이 잘못되었습니다. *.exe input_text.txt delimiter.txt output.txt" << endl;
            throw - 1;
        }

        CTextMine my_text_miner;
        my_text_miner.read_delimiter_file(argv[2]);
        my_text_miner.read_text_file_and_parse(argv[1]);
        my_text_miner.sort_word_list();
        my_text_miner.write_result(argv[3]);

        cout << "전체 word 수 (중복) = " << my_text_miner.getTotalWordCount() << endl;
        cout << "전체 word 수 = " << my_text_miner.getWordCount() << endl;
        elapsedTime = (double)(double)(clock() - clockA) / CLOCKS_PER_SEC / 1000.0;
        cout << "구동시간 = " <<fixed<< elapsedTime << " milli seconds" << endl;
    }
    catch (int exception) {
        if (exception == -1)
            cout << "input 오류입니다" << endl;
        else if (exception == -2)
            cout << "delimiter.txt 파일을 열 수 없습니다" << endl;
        else if (exception == -3)
            cout << "input_text.txt 파일을 열 수 없습니다" << endl;
        else if (exception == -4)
            cout << "output.txt 파일을 열 수 없습니다" << endl;
    }
    catch (...) {
        cout << "구분되지 않은 예외입니다" << endl;
    }
    return 0;
}
```

- 1) 프로그램 실행 시작 시각을 clockA에 저장한다.
- 2) Terminal(Window의 경우 cmd)에서 실행 시 입력이 잘못 된 경우를 예외처리한다.
- 3) CTextMine를 인스턴스화한다.
- 4) Flow chart에 따라 메소드를 호출한다.
- 5) 메소드 호출이 모두 끝난 시각과 1번의 clockA를 이용하여 구동 시간을 계산, 출력한다.
- 6) 프로그램을 종료한다.

### Ⅲ. 수행 결과에 대한 토의

#### 1. 토론

##### (1) 프로그램 개발 과정 중 발생한 문제와 해결 방법

###### 1) 개행문자 처리

텍스트 파일을 읽어 들일 때 개행문자 처리가 곤란했다. delimiter.txt 파일에서 구분자 문자를 받기 위해 문자 단위 입력 함수를 사용하려고 했지만 입력 버퍼에 자꾸 'wr' 문자가 남아있어서 getline 함수를 이용하여 'wr'까지 문자열로 입력받았다. 문자열의 첫번째 문자만 구분자로 따로 추출하였다(read\_delimiter\_file).

하지만 인풋 텍스트 파일에는 'wr' 뿐만 아니라 'wn', 'wrwn' 문자도 존재했다. 'wr'은 getline으로 1차적으로 거르고, 남아있는 빈 문자열, 'wn'은 메소드 내의 if 문으로 2차적으로 처리하였다(read\_input\_file). 3차적으로는 'wn'이 연속으로 입력되었을 경우를 방지하여 'wn'를 프로그램 내에서 구분자 목록에 추가하였다(read\_delimiter\_file).

###### 2) 구분자를 이용한 단어 추출

인풋 텍스트 파일을 읽어들이는 단계에서 바로 구분자와 비교해서 단어를 추출하여 인스턴스화 하고자 했다. 구분자는 여러개이기 때문에 비교는 반복적으로 이루어진다. 그런데 스트림은 한번 쓰여지면 다음 문자로 커서가 이동했기 때문에 결국 비교 대상을 임시로 저장하고, 저장한 것과 구분자를 비교했다. 비교 대상은 개행문자 'wr'이 나타나기 까지의 문자열을 tempLine에 임시 저장하기로 했다(read\_input\_file).

###### 3) calculateWordCount 메소드 미사용

과제에서 주어진 메소드를 이용하지 않았다. 단어를 인스턴스화 할 때 중복 여부를 검사하고 바로 WordCount를 계산했다. 따로 메소드를 사용하는 것 보다 이 방법이 더 효율적일 것이라 판단했다. 하지만 calculateWordCount가 어떻게 정의된 메소드인지 모르기 때문에 효율성 측면에서 미해결 문제로 남아있다.

## 2. 결론

책에서 읽어보거나 간단한 예제로만 다루어 본 vector, file I/O, main 함수 인자를 프로그램의 목적에 맞게 응용하고 구현하는 과정에서 개념을 더욱 자세히 알게 되었다. 특히 file I/O는 다루기 어려웠는데 이번 과제를 통해 예전보다 수월해졌다. 앞으로도 직접 개념을 코드화하고 프로그램 기능을 구현해봄으로써 C++의 모든 파트에서 능숙해지길 기대한다.

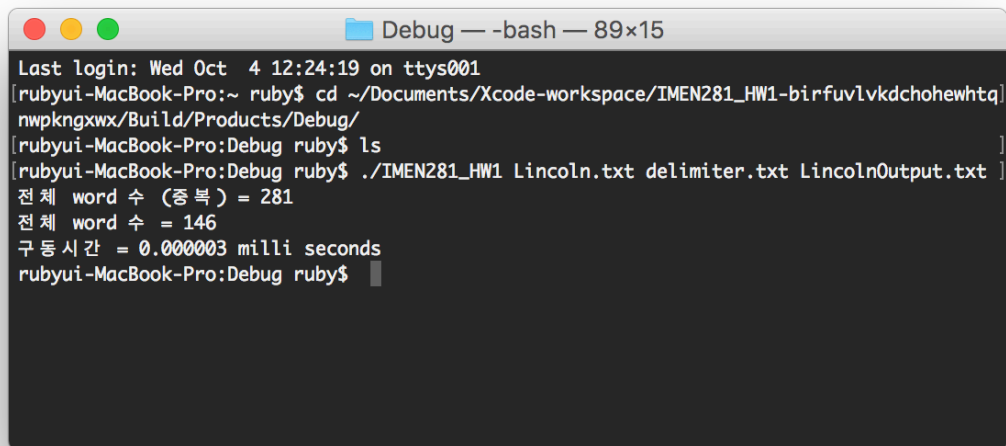
**과제에 소요된 시간**

소스 코드 작성: 5 시간

보고서 작성 및 주석 수정: 2 시간

**총 7 시간 소요**

#### IV. 실행화면 스크린샷



```
Debug — -bash — 89x15
Last login: Wed Oct 4 12:24:19 on ttys001
[rubyui-MacBook-Pro:~ ruby$ cd ~/Documents/Xcode-workspace/IMEN281_HW1-birfuvlvkdchohewhtq]
nwpkngxwx/Build/Products/Debug/
[rubyui-MacBook-Pro:Debug ruby$ ls
[rubyui-MacBook-Pro:Debug ruby$ ./IMEN281_HW1 Lincoln.txt delimiter.txt LincolnOutput.txt ]
전체 word 수 (중복) = 281
전체 word 수 = 146
구동 시간 = 0.000003 milli seconds
rubyui-MacBook-Pro:Debug ruby$
```



```
LincolnOutput.txt
전체 word 수 (중복) = 281
전체 word 수 = 146

Word, count
that, 13
the, 9
here, 8
to, 8
we, 8
--, 7
a, 7
and, 6
can, 5
for, 5
have, 5
nation, 5
not, 5
of, 5
dedicated, 4
in, 4
this, 4
It, 3
are, 3
dead, 3
great, 3
is, 3
people, 3
shall, 3
so, 3
they, 3
```