O-O Programming & Data Structure Lab. 12

12. 예측구문 (predictive text) 탐색을 위한 Trie 자료구조 구현

12.1 class MyVoca

```
/** MyVoca.h */

#ifndef MY_VOCA_H
#define MY_VOCA_H

#include <iostream>
#include <string>
#include <list>
using namespace std;

enum Word_Type {NOUN, VERB, ADJ, ADV, PREPOS}; // noun, verb, adjective, adverbs, preposition
typedef list<string> List_Str;
typedef list<string>::iterator Lst_Str_Itr;

class MyVoca
{
        friend ostream& operator<<(ostream& fout, MyVoca& mv)
        { . . . . .  }
public:
        MyVoca(string kw, Word_Type wt, List_Str thes, List_Str ex_usg)
                :keyWord(kw), type(wt), thesaurus(thes), usages(ex_usg)
        {}
        MyVoca() {} // default constructor
        string getKeyWord() { return keyWord; }

private:
        string keyWord; // entry word (also key)
        Word_Type type;
        List_Str thesaurus; // thesarus of the entry word in the type
        List_Str usages;
};

#endif
```

12.2 MyVocaList.h

```
/* MyVocaList.h */
#ifndef MY_VOCA_LIST_H
#define MY_VOCA_LIST_H

int   NUM_MY_TOEIC_VOCA = 100;
MyVoca myToeicVocaList[]; // defined in MyVocaList.cpp

#endif
```

12.3 class TrieNode

```
#define VALUE_INTERNAL_NODE NULL
using namespace std;
typedef list<string> STL_list;

template <typename E>
class TrieNode
{
public:
        TrieNode() {} // default constructor
        TrieNode(char k, E v) : key(k), value(v) { prev = next = parent = child = NULL; }
        void setKey(char k) { key = k; }
        void setValue(E v) { value = v; }
        void setNext(TrieNode<E> *nxt) { next = nxt; }
        void setPrev(TrieNode<E> *pv) { prev = pv; }
        void setParent(TrieNode<E> *pr) { parent = pr; }
        void setChild(TrieNode<E> *chld) { child = chld; }
        char getKey() { return key; }
```

```
        E getValue() { return value; }
        TrieNode<E> *getPrev() { return prev; }
        TrieNode<E> *getNext() { return next; }
        TrieNode<E> *getParent() { return parent; }
        TrieNode<E> *getChild() { return child; }
        void _fprint(ostream& fout, TrieNode<E> *pTN, int indent);

private:
        char key;
        E value;
        TrieNode<E> *prev;
        TrieNode<E> *next;
        TrieNode<E> *parent;
        TrieNode<E> *child;
};
```

12.4 class Trie

```
#include <iostream>
#include <string>
#include "TrieNode.h"
#define MAX_STR_LEN 50

//#define DEBUG
using namespace std;

typedef list<MyVoca *> List_pVoca;
typedef list<MyVoca *>::iterator List_pVoca_Iter;
enum SearchMode {FULL_MATCH, PREFIX_MATCH};
template <typename E>
class Trie
{
public:
        Trie(string name); // constructor
        int size() { return num_keys; }
        void insert(string keyStr, E value);
        void insertExternalTN(TrieNode<E> *pTN, string keyStr, E value);
        TrieNode<E> *find(string keyStr);
        void findPrefixMatch(string prefix, List_String& predictWords);
        void deleteKeyStr(string keyStr);
        void eraseTrie();
        void fprintTrie(ostream& fout);
protected:
        TrieNode<E> *_find(string keyStr, SearchMode sm=FULL_MATCH);
        void _traverse(TrieNode<E> *pTN, List_pVoca& list_keywords);
private:
        TrieNode<E> *_root; // _root trie node
        int num_keys;
        string trie_name;
};
```

12.5 main() function

```
/* main_trie.cpp */
#include <iostream>
#include <fstream>
#include <list>
#include <string>
#include "MyVoca.h"
#include "MyVocaList.h"
#include "Trie.h"
#include "TrieNode.h"

using namespace std;
#define MAX_WORD_LENGTH 100
```

```
#define NUM_TEST_VOCAS 7
//#define TEST_SIMPLE_TRIE
void test_simple_trie(ostream& fout);
void test_trie_myVoca(ostream& fout);

void main()
{
	ofstream fout;
	fout.open("output.txt");
	if (fout.fail())
	{
		printf("Error in opening output file !\n");
		exit;
	}

	Trie<MyVoca *> trie_myVoca("Trie_MyVoca");
	TrieNode<MyVoca *> *pTN;
	MyVoca *pVoca;
	int word_count;
	string keyStr;
	char keyWord[MAX_WORD_LENGTH];
	List_pVoca predictVocas;
	List_pVoca_Iter itr;

	/* Testing Basic Operation in trie */
	MyVoca testVocas[NUM_TEST_VOCAS] =
	{
		MyVoca("xyz", NOUN, { "" }, { "" }),
		MyVoca("ABCD", NOUN, { "" }, { "" }),
		MyVoca("ABC", NOUN, { "" }, { "" }),
		MyVoca("AB", NOUN, { "" }, { "" }),
		MyVoca("A", NOUN, { "" }, { "" }),
		MyVoca("xy", NOUN, { "" }, { "" }),
		MyVoca("x", NOUN, { "" }, { "" }),
	};
	fout << "Testing basic operations of trie inserting ..... " << endl;
	for (int i = 0; i < NUM_TEST_VOCAS; i++)
	{
		trie_myVoca.insert(testVocas[i].getKeyWord(), &testVocas[i]);
	}
	trie_myVoca.fprintTrie(fout);

	/*Destroy the trie*/
	fout << "\nTesting TrieDestroy...\n";
	trie_myVoca.eraseTrie();
	trie_myVoca.fprintTrie(fout);

	/* inserting keyWords into trie */
	fout << "Inserting My Vocabularies to myVocaDict . . . " << endl;
	word_count = 0;
	pVoca = &myToeicVocaList[0];
	while (pVoca->getKeyWord() != "-1")
	{
		keyStr = pVoca->getKeyWord();
		trie_myVoca.insert(keyStr, pVoca);
		pVoca++;
	}
	fout << "Total " << trie_myVoca.size() << " words in trie_myVoca .." << endl;
	trie_myVoca.fprintTrie(fout);

	/* testing keyWord search in trie */
	while (1)
	{
		cout << "\nInput any prefix to search in trie (. to finish) : ";
		cin >> keyStr;
		if (keyStr == string("."))
```

```
                        break;
                predictVocas.clear();
                trie_myVoca.findPrefixMatch(keyStr, predictVocas);
                cout << "list of predictive wors with prefix (" << keyStr << ") :" << endl;
                itr = predictVocas.begin();
                for (int i = 0; i < predictVocas.size(); i++)
                {
                        pVoca = *itr;
                        cout << *pVoca << endl;
                        ++itr;
                }
        }
        cout << "\nErasing trie_myVoca ...." << endl;
        fout << "\nErasing trie_myVoca ...." << endl;
        trie_myVoca.eraseTrie();

        fout.close();
}
```

12.6 Example output (output.txt)

```
Testing basic operations of trie inserting .....
trie ( Trie_MyVoca) with 7 trie_nodes

A
 B
  C
   D
x
 y
  z

Testing TrieDestroy...
trie ( Trie_MyVoca) with 0 trie_nodes
Empty trie !
```

12.7 Example output (output.txt)

```
Inserting My Vocabularies to myVocaDict . . .
Total 230 words in trie_myVoca ..
trie ( Trie_MyVoca) with 230 trie_nodes

abstract
 ccelerometer
   ident
   umulator
  hievement
  id
  oustics
  tuator
 dequate
.....

ultra-sonic
versatile
 iolate
  rtualization
  scous
 olatile
 ulnerable

Erasing trie_myVoca ....
```

12.7 Example output (console)

```
Input any prefix to search in trie (. to finish) : ac
list of predictive wors with prefix (ac) :
accelerometer(n):
  - thesaurus(, )
  - example usage( )
accident(n):
  - thesaurus(, )
  - example usage( )
accumulator(n):
  - thesaurus(, )
  - example usage( )
achievement(n):
  - thesaurus(, )
  - example usage( )
acid(n):
  - thesaurus(, )
  - example usage( )
acoustics(n):
  - thesaurus(, )
  - example usage( )
actuator(n):
  - thesaurus(, )
  - example usage( )

Input any prefix to search in trie (. to finish) :
```

```
Input any prefix to search in trie (. to finish) : v
list of predictive wors with prefix (v) :
versatile(n):
  - thesaurus(, )
  - example usage( )
violate(n):
  - thesaurus(, )
  - example usage( )
virtualization(n):
  - thesaurus(, )
  - example usage( )
viscous(n):
  - thesaurus(, )
  - example usage( )
volatile(n):
  - thesaurus(, )
  - example usage( )
vulnerable(n):
  - thesaurus(, )
  - example usage( )

Input any prefix to search in trie (. to finish) :
```

<Oral Test 12>

(1) 문자열 (string) 자료형의 키워드에 대한 예측구문 (predictive text) 응용 분야와 이를 구현하기 위한 trie 자료구조에 대하여 그림과 함께 상세하게 설명하라.

<Key Points>

(1) 문장열 (string) 자료형의 키워드에 대한 예측구문 (predictive text) 응용 분야

(2) trie 자료구조에 대한 설명

(2) trie 자료구조를 구현하기 위한 class TrieNode 에 대하여 그림과 pseudo code 를 사용하여 설명하라.

<Key Points>

(1) class TrieNode의 데이터 멤버

(2) class TrieNode의 멤버함수들

(3) class TrieNode의 _fprint() 멤버함수

(3) trie 자료구조에서 주어진 키 문자열을 접두어 (prefix)로 구성될 수 있는 예측 구문을 탐색 (find)하는 절차에 대하여 그림과 pseudo code 를 사용하여 설명하라.

<Key Points>

(1) _find() 멤버함수

(2) _traverse() 멤버함수

(3) findPrefixMatch() 멤버함수

(4) trie 자료구조에서 주어진 키 문자열을 삽입 (insert)하는 절차에 대하여 그림과 pseudo code 를 사용하여 설명하라.

<Key Points>

(1) 키 문자열 삽입을 위한 _find() 멤버 함수 실행

(2) 이미 포함된 키 문자열들 보다 앞선 순서의 새로운 문자열 삽입

(3) 이미 포함된 키 문자열들 보다 뒤 순서의 새로운 문자열 삽입

(4) 이미 포함된 키 문자열들 중간에 새로운 문자열 삽입

(5) 기존에 포함된 키 문자열의 일부가 새로운 키 문자열로 삽입되는 경우