Object-Oriented Programming and Data Structure

Lab. 12 (보충설명) 예측구문 탐색을 위한 Trie 자료구조

정보통신공학과

교수 김 영 탁

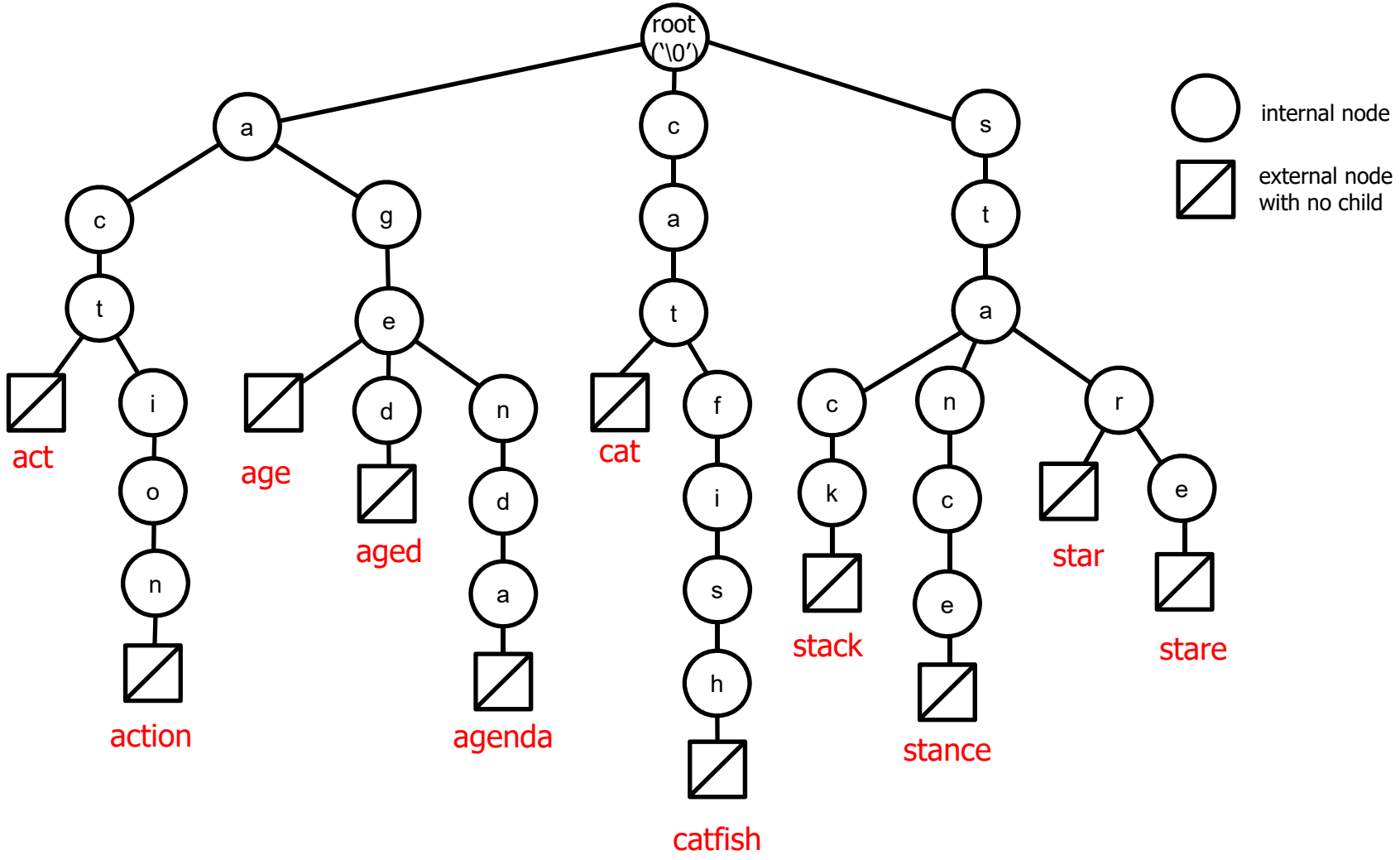
(Tel : +82-53-810-2497; E-mail : ytkim@yu.ac.kr)

Outline

- **trie 자료구조**
- **trie 자료구조의 C++ 구현**
  - class TrieNode
  - class Trie_String
  - insert()
  - find()
  - findPrefixMatch()
  - deleteKeyStr()
  - eraseTrie()

trie 자료구조

◆ trie 구성 예



trie 자료구조의 주요 응용 분야

◆ 예측 구문 (predictive text)

- 스마트 기기의 입력에서 적은 수의 타이핑으로도 예측되는 구문 (predictive text)을 안내하여 신속하게 구문을 완성할 수 있게 함
- 사전 (dictionary)을 활용한 단어 검색에서 자동 완성 (auto complete) 기능으로 예상되는 단어를 열거하여 주고, 이 단어 들 중에서 고르게 함

◆ Longest prefix matching

- 인터넷 패킷 교환기 (router)의 packet forwarding table 구성에서 목적지 주소 (destination address)와 가장 많이 일치하는 항목을 선정하여 전달 할 수 있도록 라우팅 테이블을 구성

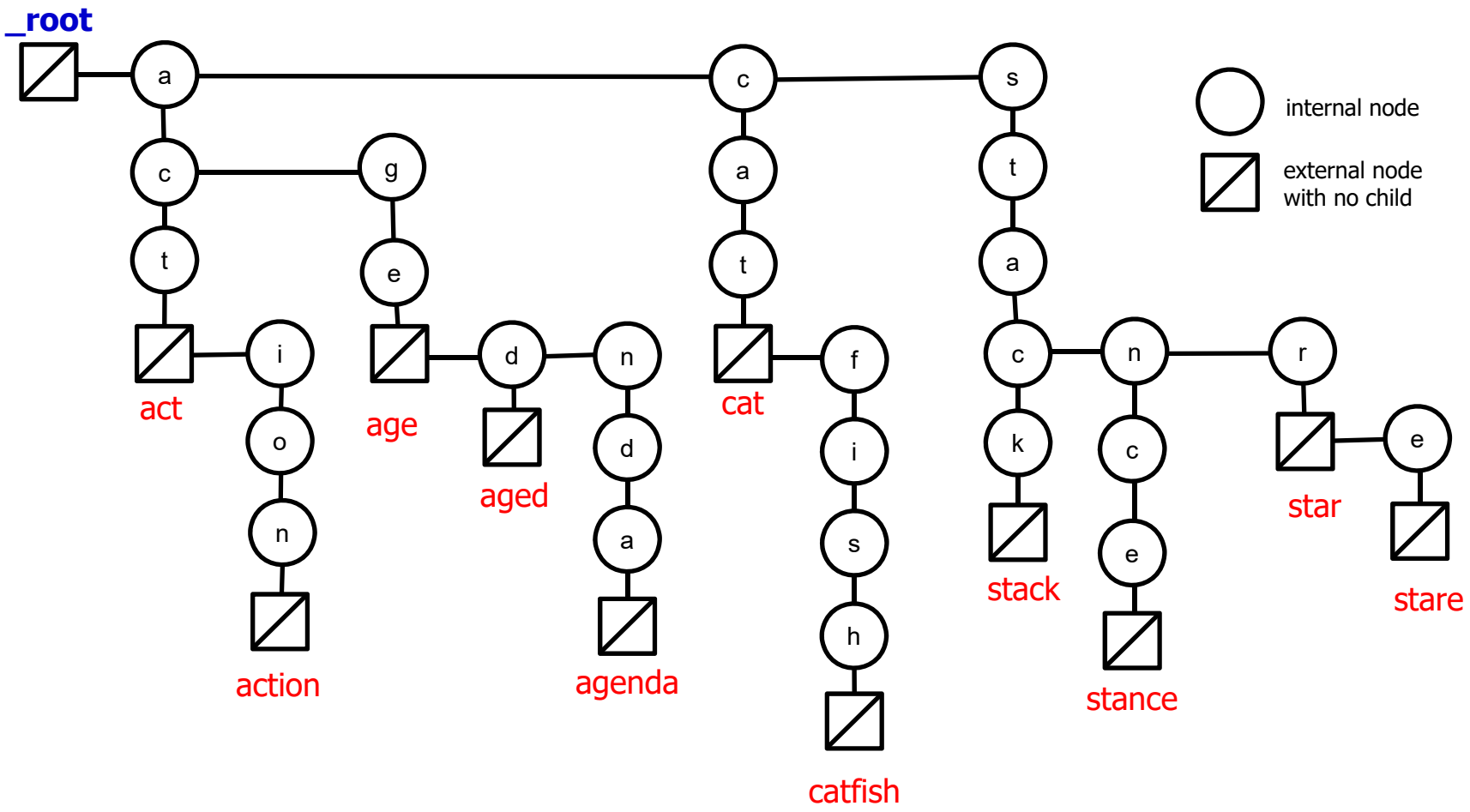
trie 자료구조의 구현

◆ trie 자료 구조의 구현에서의 고려사항

- 이진 탐색 트리와 달리 하나의 트리 노드에 접속되는 자식 노드의 수가 3개 이상 포함될 수 있음
- 동일한 substring을 가지는 다수의 key string이 존재할 수 있으며, 어떤 key string의 prefix가 다른 key string이 될 수 도 있음
  - 예) key string "age", "aged", "agenda"
- root node로 부터 trie tree 탐색에서 longest matching이 가능하여 predictive text가 제공 될 수 있도록 구성하여야 함

trie 자료구조의 구현

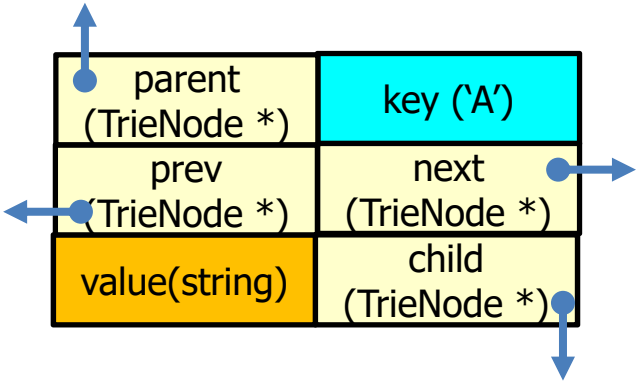
◆ trie 구현 예



trie의 C++ 프로그램 모듈 구현 (1) – Class TrieNode

◆ Class TrieNode

- private data members
  - char key; // key character of this trie_node
  - E value; // value assigned to the key string that ends at this trie_node
  - 4 pointers to previous, next, parent, and child trie_nodes



class MyVoca

```
/** MyVoca.h (1) */

#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)
        {
                string wd_ty[] = { "n", "v", "adj", "adv", "prepos" };
                list<string>::iterator itr;
                fout << mv.keyWord << "(" << wd_ty[mv.type] << "): ₩n";
                fout << "  - thesaurus(";
```

```
/** MyVoca.h (2) */

            for (itr = mv.thesaurus.begin(); itr != mv.thesaurus.end(); ++itr)
            {
                  fout << *itr << ", ";
            }
            fout << ")" << endl;
            fout << "  - example usage(";
            for (itr = mv.usages.begin(); itr != mv.usages.end(); ++itr)
            {
                  fout << *itr << " ";
            }
            fout << ")";
            return fout;
      }
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
```

MyVocaList.h, MyVocaList.cpp

```
/* MyVocaList.h */
#ifndef MY_VOCA_LIST_H
#define MY_VOCA_LIST_H
#include "MyVoca.h"

int NUM_MY_TOEIC_VOCA = 13;
MyVoca myToeicVocaList[]; // defined in MyVocaList.cpp

#endif
```

```
/* MyVocaList.cpp */
#include "MyVoca.h"
#define TEST_FULL_SET

MyVoca myToeicVocaList[] =
{
        MyVoca("mean", NOUN, { "average", "norm", "median", "middle", "midpoint", "(ant) extremity" }, { "the mean error", "the golden
                mean", "the arithmetical mean", "the geometric mean" }),
        MyVoca("offer", NOUN, { "proposal" }, { "He accepted out offer to write the business plan." }),
        MyVoca("compromise", NOUN, { "give-and-take", "bargaining", "accommodation" }, { "The couple made a compromise and ordered
                food to take out." }),
        MyVoca("delegate", NOUN, { "representative", "agent", "substitute" }, { "" }),

        . . . . .

        MyVoca("-1", NOUN, { "" }, { "" }) // end sentinel
};
```

class TrieNode

```
/* TrieNode.h  (1) */

#ifndef TRIE_NODE_H
#define TRIE_NODE_H
#include <iostream>
#include <string>
#include <list>
#define VALUE_INTERNAL_NODE  NULL
using namespace std;

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; }
```

```
/* TrieNode.h  (2) */

    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;
};
```

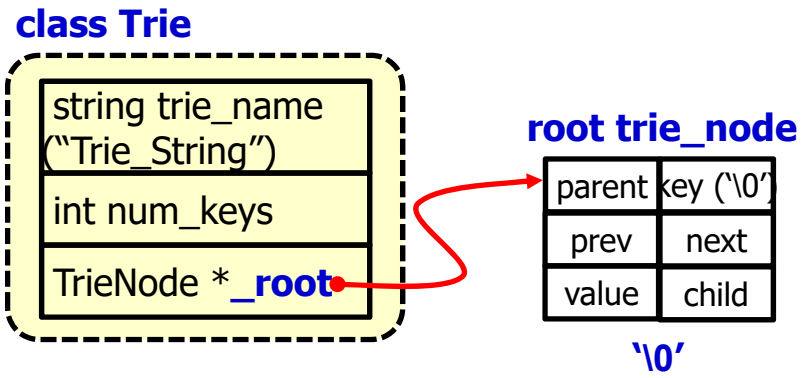
```
/* TrieNode.h  (3) */

template<typename E>
void TrieNode<E>::_fprint(ostream& fout, TrieNode<E> *pTN, int indent)
{
    if (pTN == NULL)
    {
        fout << endl;
        return;
    }
    else
    {
        fout << pTN->key;
        _fprint(fout, pTN->child, indent + 1);
        if (pTN->next == NULL)
        {
            return;
        }
        for (int i = 0; i < indent; i++)
        {
            fout << " ";
        }
        _fprint(fout, pTN->next, indent);
    }
}


#endif
```

```
act         // act
   ion      // action
 ge         //age
   d        //aged
   nda      //agenda
cat         //cat
   fish     //catfish
stack       //stack
   nce      //stance
   r        //star
     e      //stare
```

Class Trie

Class Trie

- data members
  - TrieNode<E> *_root;
  - int num_keys;
  - string trie_name;
- member functions
  - Trie(string name); // constructor
  - void insert(string keyWord, E value);
  - void insertExternalTN(TrieNode<E> *pTN, string keyWord, E value);
  - TrieNode<E> *find(string keyWord);
  - void findPrefixMatch(string keyWord, List_pVoca& predictWords);
  - void deletekeyWord(string keyWord);
  - void eraseTrie();
  - void fprintTrie(ostream& fout);
  - TrieNode<E> *_find(string keyWord, SearchMode sm=FULL_MATCH);
  - void _traverse(TrieNode<E> *pTN, List_pVoca& predictWords);

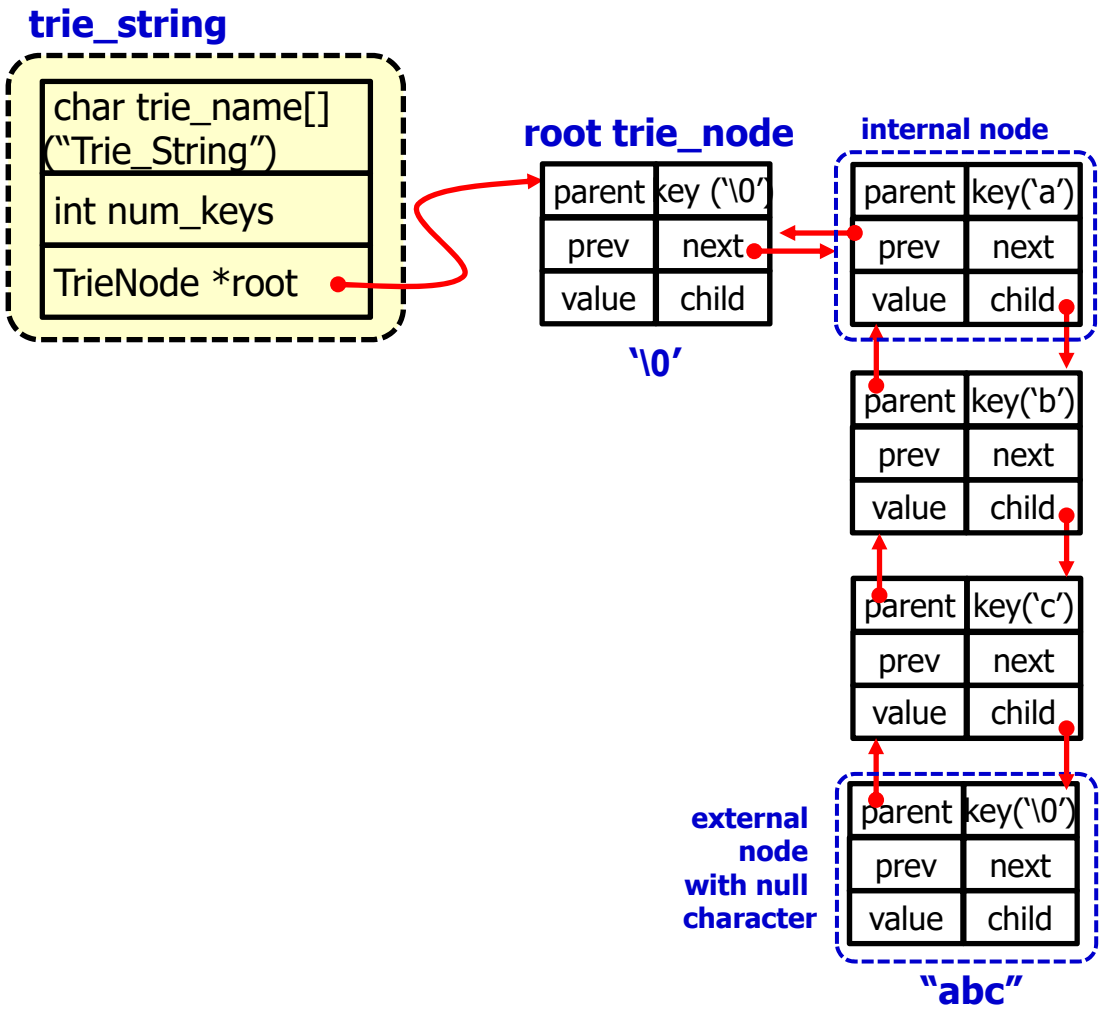


```
/* Trie.h  (1) */
#ifndef Trie_H
#define Trie_H
#include <iostream>
#include <string>
#include "TrieNode.h"
#define MAX_STR_LEN 50

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; }
    string getName() { return trie_name; }
    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_pVocas& predictVocas);
    void deleteKeyStr(string keyStr);
    void eraseTrie();
    void fprintTrie(ostream& fout);
```

```
/* Trie.h  (2) */
protected:
    TrieNode<E> *_find(string keyStr, SearchMode
        sm=FULL_MATCH);
    void _traverse(TrieNode<E> *pTN, List_pVoca&
        list_pVocas);
private:
    TrieNode<E> *_root; // _root trie node
    int num_keys;
    string trie_name;
};

template<typename E>
Trie<E>::Trie(string name)
{
    trie_name = name;
    _root = new TrieNode<E>('₩0', NULL);
    _root->setKey('₩0');
    _root->setPrev(NULL);
    _root->setNext(NULL);
    _root->setParent(NULL);
    _root->setChild(NULL);
    num_keys = 0;
}
```

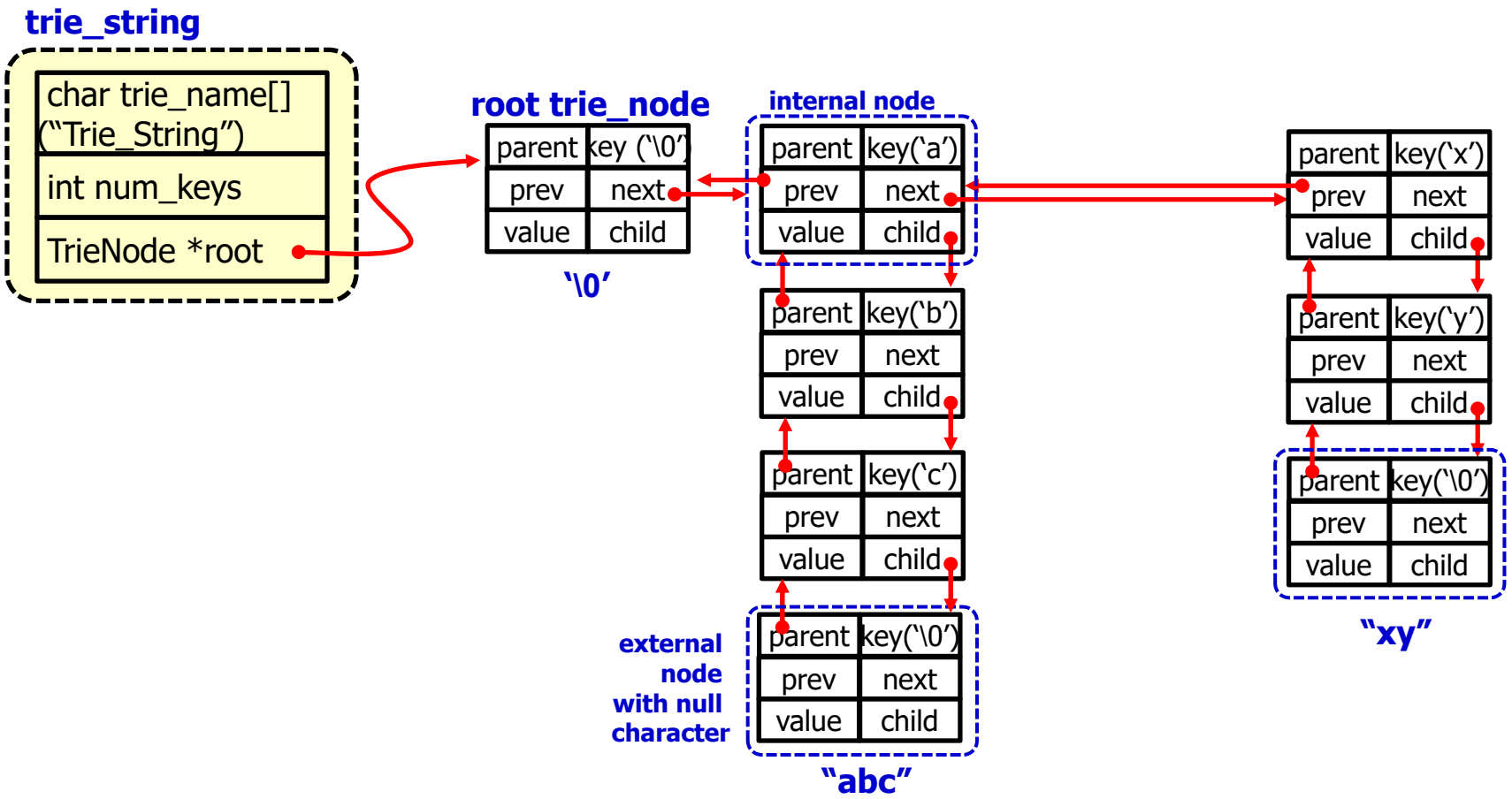
insert operation

◆ insert ("abc")



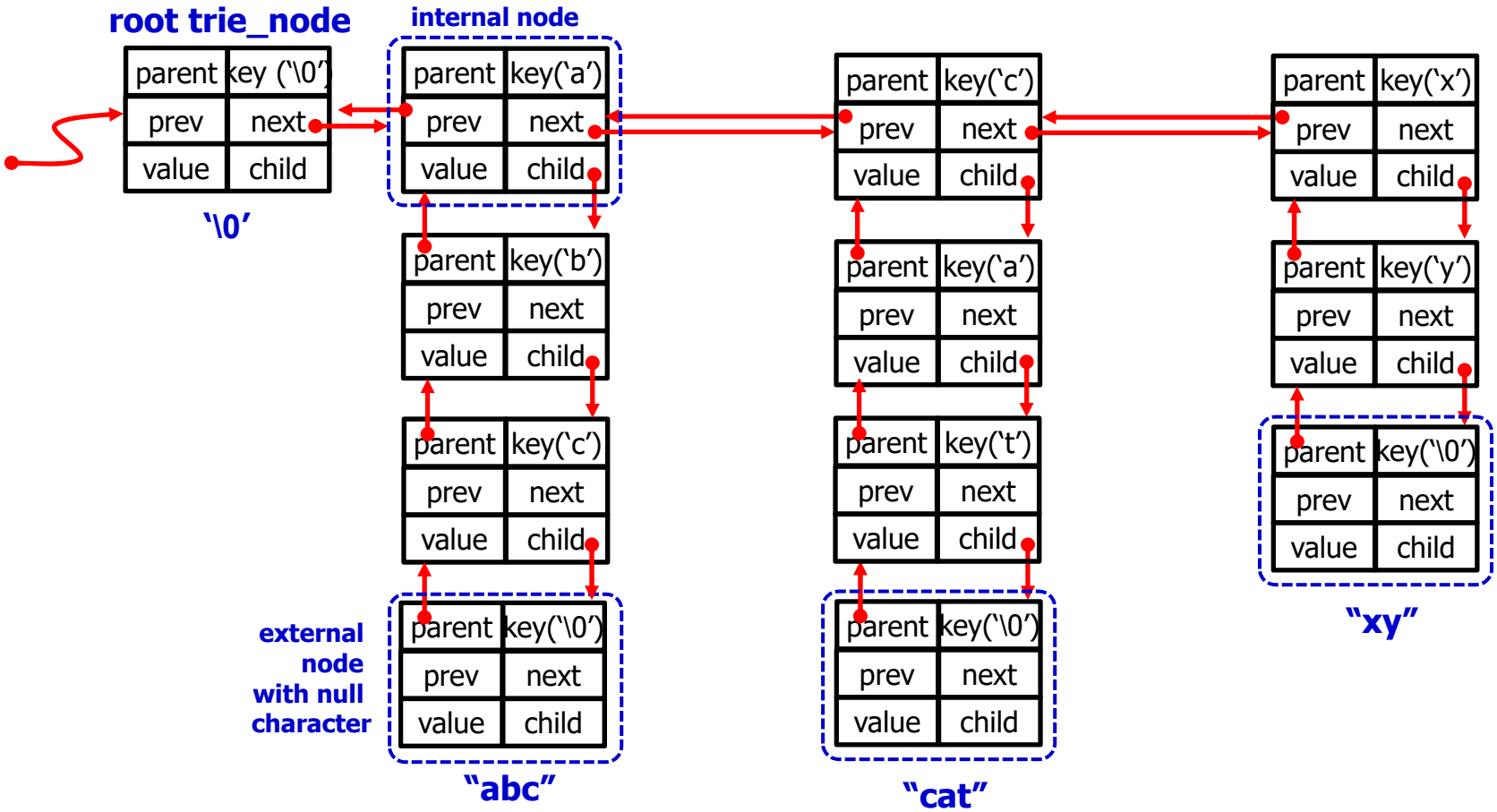
insert operation

- insert ("xy") while "abc" is already inserted before



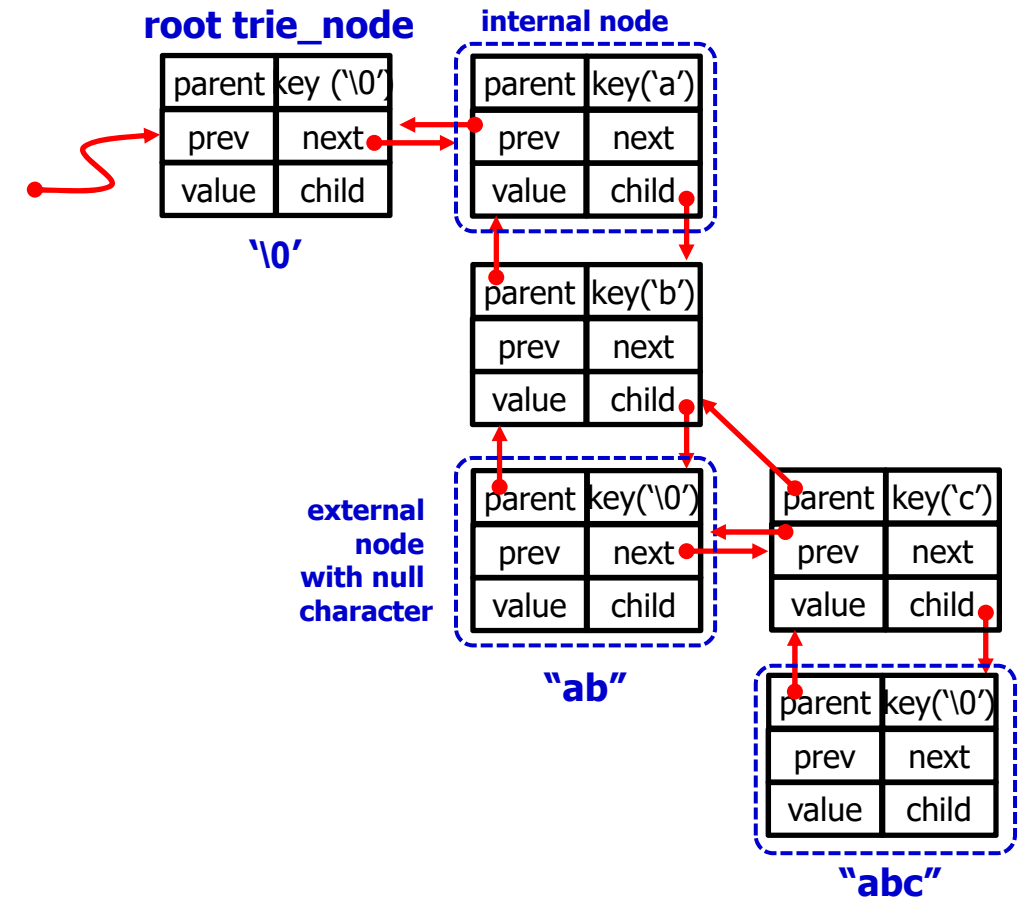
insert operation

- insert ("cat") while "abc" and "xy" are already inserted before



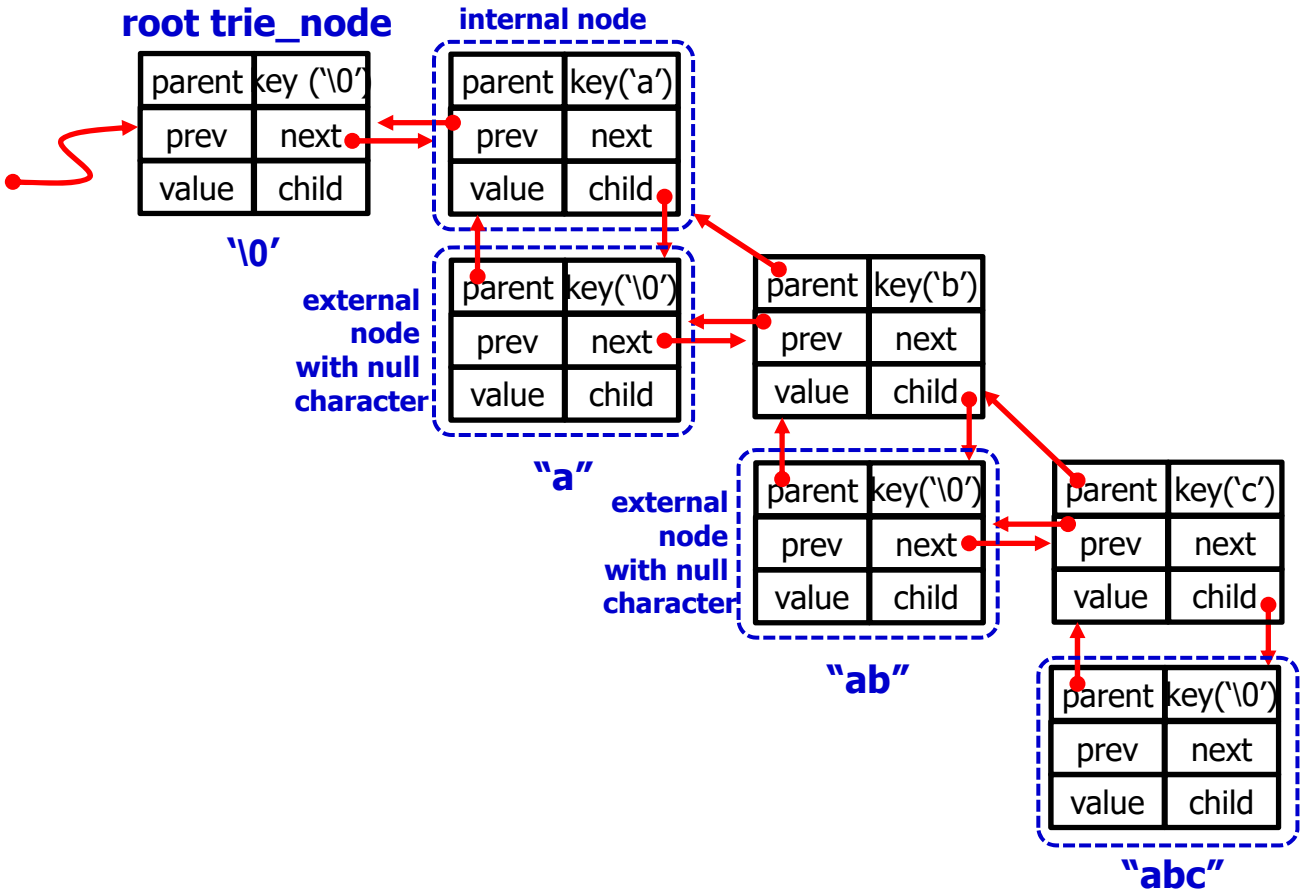
insert operation

- insert ("ab") while "abc" was already inserted before



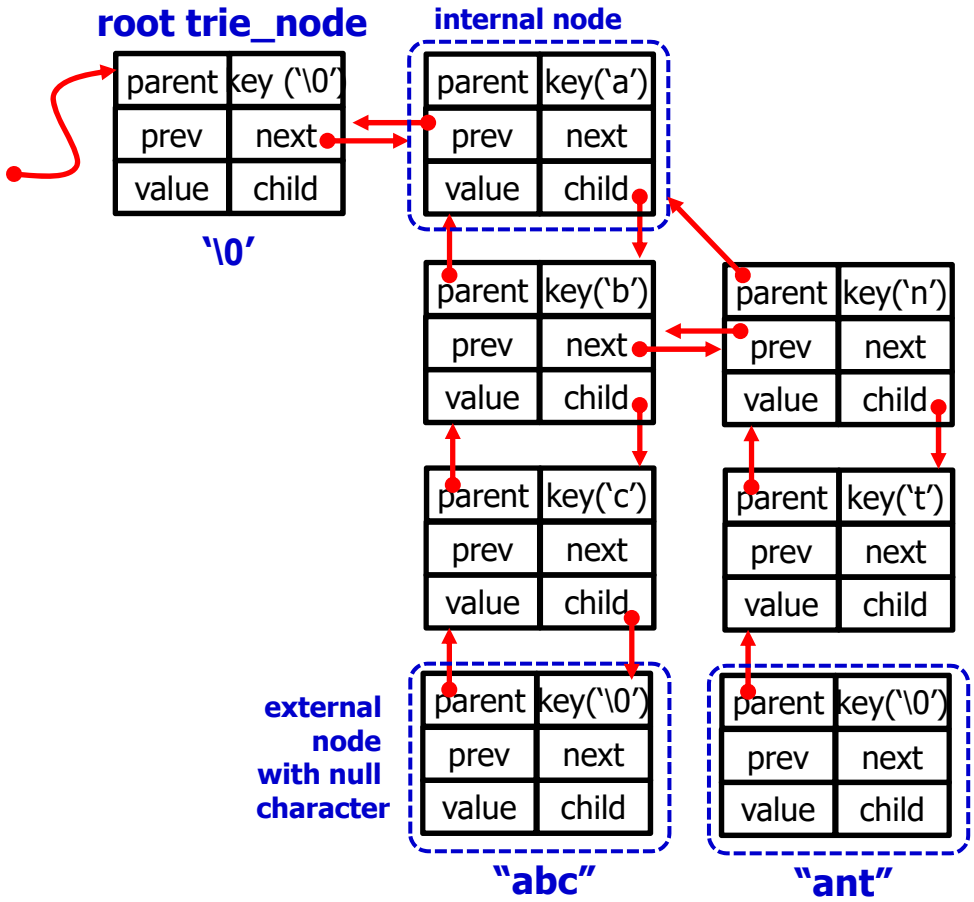
insert operation

- insert ("a") while "abc" and "ab" were already inserted before



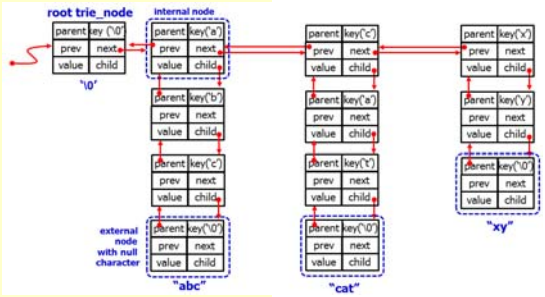
insert operation

- insert "ant" while "abc" is already existing

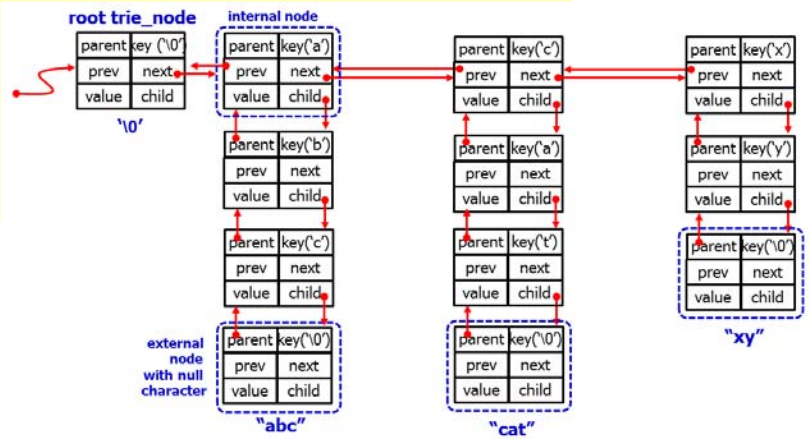


```
/* Trie.h  (3) */

template<typename E>
void Trie<E>::insertExternalTN(TrieNode<E> *pTN, string keyStr, E value)
{
    TrieNode<E> *pTN_New = NULL;

    pTN_New = new TrieNode<E>('₩0', value);
    pTN->setChild(pTN_New);
    (pTN->getChild())->setParent(pTN);
    pTN_New->setValue(value);
    //cout << "key (" << keyStr << ") is inserted ₩n";
}

template<typename E>
void Trie<E>::insert(string keyStr, E value)
{
    TrieNode<E> *pTN = NULL, *pTN_New = NULL;
    char *keyPtr = (char *)keyStr.c_str();

    if (keyPtr == NULL)
        return;

    /* Firstly, check any possible duplicated key insertion  */
    if (_find(keyStr, FULL_MATCH) != NULL)
    {
        cout << "The given key string (" << keyStr << ") is already existing; just return !!" << endl;
        return;
    }
```

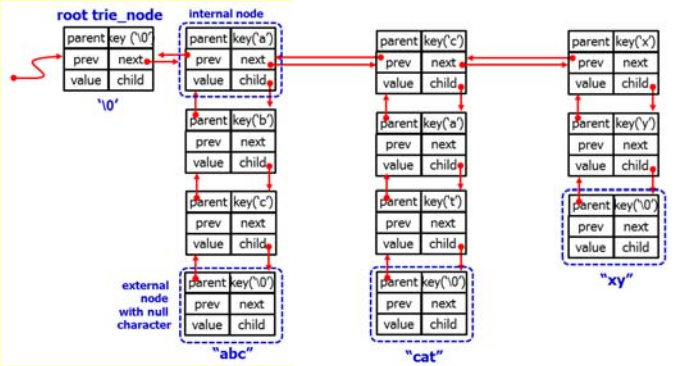
```
/* Trie.h  (4) */

    pTN = this->_root;
    while ((pTN != NULL) && (*keyPtr != '\0'))
    {
        if ((pTN->getKey() < *keyPtr) && (pTN->getNext() == NULL) && (*keyPtr != '\0'))
            break;
        while ((pTN->getKey() < *keyPtr) && (pTN->getNext() != NULL))
            pTN = pTN->getNext();
        while ((pTN != NULL) && (pTN->getKey() == *keyPtr) && (*keyPtr != '\0'))
        {
            pTN = pTN->getChild();
            keyPtr++;
        }
        if ((pTN->getKey() > *keyPtr) && (*keyPtr != '\0'))
            break;
    } // end while for positioning

    /* Secondly, the given key string is a sub-string of an existing key */
    /* e.g.) trying to insert "abc" while "abcde" is already exisiting. */
    if ((pTN->getKey() != '\0') && (*keyPtr == '\0'))
    {
        /* there was a longer key string already !! */
        /* break the longer key string, and connected to the separated key strings */
        pTN_New = new TrieNode<E>('\0', value);
        pTN_New->setParent(pTN->getParent());
        (pTN->getParent())->setChild(pTN_New);
        pTN_New->setNext(pTN);
        pTN->setPrev(pTN_New);
        //cout << "key (" << keyWord << ") is inserted" << endl;
        this->num_keys++;
        return;
    }
```



```
/* Trie.h  (5) */

    else if ((pTN->getKey() < *keyPtr) && (pTN->getNext() == NULL) && (*keyPtr != '\0'))
    {
        /* at this level, a new substring is inserted as the last nodes */
        pTN_New = new TrieNode<E>(*keyPtr, VALUE_INTERNAL_NODE);
        pTN_New->setParent(pTN->getParent());
        pTN_New->setPrev(pTN);
        pTN->setNext(pTN_New);
        pTN = pTN_New;
        keyPtr++;
        while (*keyPtr != '\0')
        {
            pTN_New = new TrieNode<E>(*keyPtr, VALUE_INTERNAL_NODE);
            pTN->setChild(pTN_New);
            (pTN->getChild())->setParent(pTN);
            pTN = pTN->getChild();
            keyPtr++;
        }
        if (*keyPtr == '\0')
        {
            insertExternalTN(pTN, keyWord, value);
            this->num_keys++;
            return;
        }
    }
```

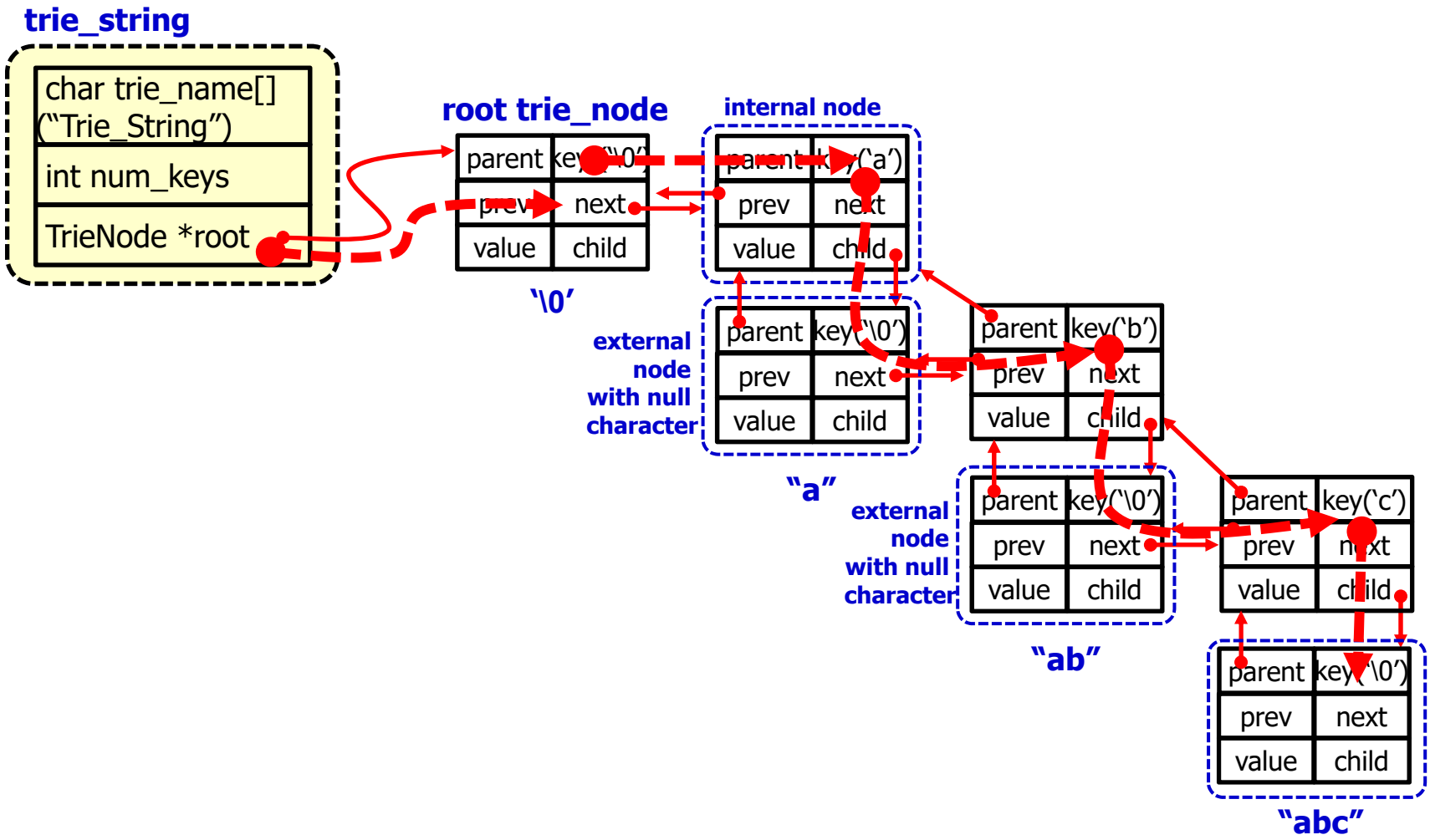


```
/* Trie.h  (6) */

    else if ((pTN->getKey() > *keyPtr) && (*keyPtr != '\0'))
    {
        /* insert between two existing trie nodes */
        pTN_New = new TrieNode<E>(*keyPtr, VALUE_INTERNAL_NODE);
        pTN_New->setNext(pTN);
        pTN_New->setParent(pTN->getParent());
        if (pTN->getPrev() == NULL)
        {   /* this pTN_new becomes the new first in this level */
             if (pTN->getParent() != NULL)
                  (pTN->getParent())->setChild(pTN_New);
        } else {
             (pTN->getPrev())->setNext(pTN_New);
        }
        pTN_New->setPrev(pTN->getPrev());
        pTN->setPrev(pTN_New);

        pTN = pTN_New;
        keyPtr++;
        while (*keyPtr != '\0')
        {
             pTN_New = new TrieNode<E>(*keyPtr, VALUE_INTERNAL_NODE);
             pTN->setChild(pTN_New);
             (pTN->getChild())->setParent(pTN);
             pTN = pTN->getChild();
             keyPtr++;
        }
        if (*keyPtr == '\0')
        {
             insertExternalTN(pTN, keyWord, value);
             this->num_keys++;
             return;
        }
    }
}
```

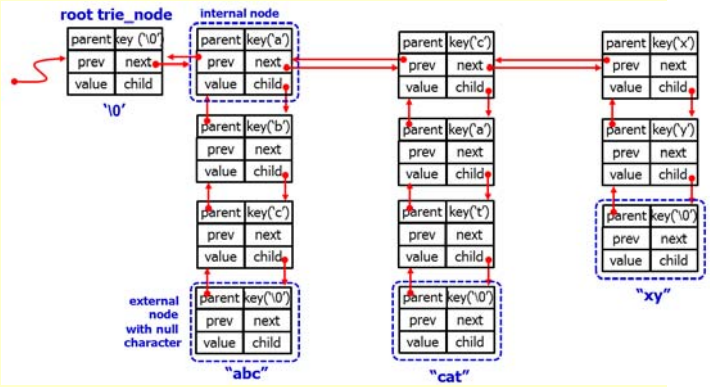


find in trie

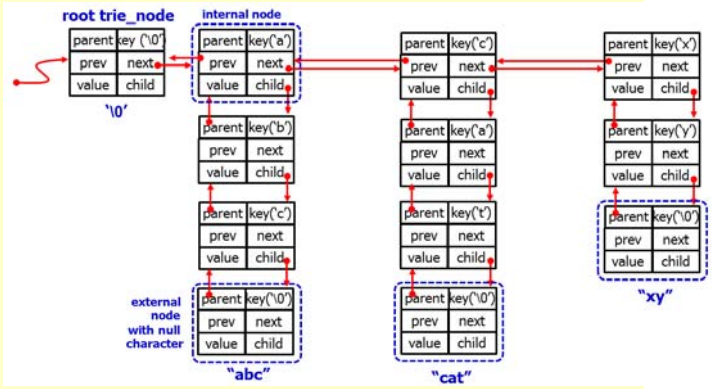
◆ find ("abc")



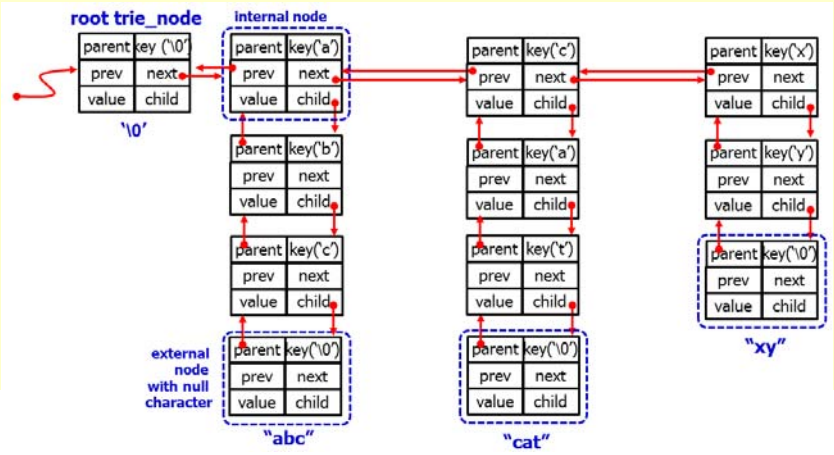
```
/* Trie.h  (7) */

template<typename E>
TrieNode<E> *Trie<E>::find(const char *keyWord)
{
        TrieNode<E> *pTN = NULL;

        pTN = _find(keyWord, FULL_MATCH);
        return pTN;
}

template<typename E>
TrieNode<E> * Trie<E>::_find(const char * keyStr, SearchMode sm = FULL_MATCH)
{
      const char *keyPtr;
      TrieNode<E> *pTN = NULL;
      TrieNode<E> *found = NULL;

      if (keyStr == NULL)
      return NULL;

      keyPtr = keyStr;
      pTN = this->_root;
      while ((pTN != NULL) && (*keyPtr != '\0'))
      {
            while ((pTN != NULL) && (pTN->getKey() < *keyPtr))
            {
                  if (pTN->getNext() == NULL)
                        return NULL;
                  pTN = pTN->getNext();
            }
```



```
/* Trie.h  (8) */

        if ((pTN != NULL) && (pTN->getKey() > *keyPtr))
        {
            // key not found
            return NULL;
        }
        else if ((pTN == NULL) && (*keyPtr != '\0'))
        {
            // key not found
            return NULL;
        }
        else if ((pTN->getKey() == *keyPtr) && (*keyPtr != '\0'))
        {
            pTN = pTN->getChild();
            keyPtr++;
            if (*keyPtr == '\0')
            {
                /* key or prefix found */
                if (sm == FULL_MATCH)
                {
                    if (pTN->getKey() == '\0')
                    {
                        /* found the key string as a full-match */
                        return pTN;
                    }
                    else // (pTN->getKey() != '\0')
                    {
                        /* found the key string as a substring of a longer existing string */
                        return NULL;
                    }
                }
```



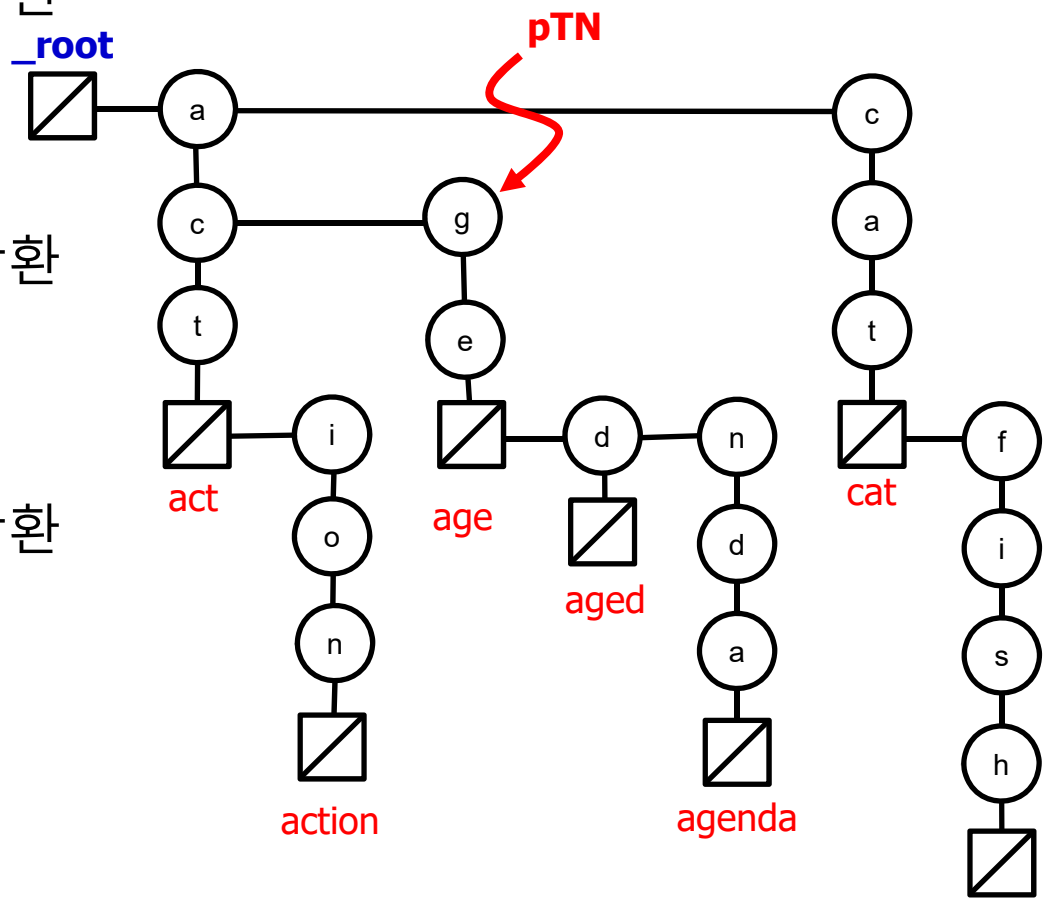
```
/* Trie.h  (9) */

                        else if (sm == PREFIX_MATCH)
                        {
                                /* found the key string as a full-match or as a substring of a longer existing
                                string */
                                return pTN;
                        }
                }
                else if ((pTN->getKey() == '\0') && (*keyPtr != '\0'))
                {
                        if (pTN->getNext() != NULL)
                        {
                                pTN = pTN->getNext();
                                continue;
                        }
                        else
                                return NULL;
                }
                else
                {
                        continue;
                }

        }
    } // end while
}
```



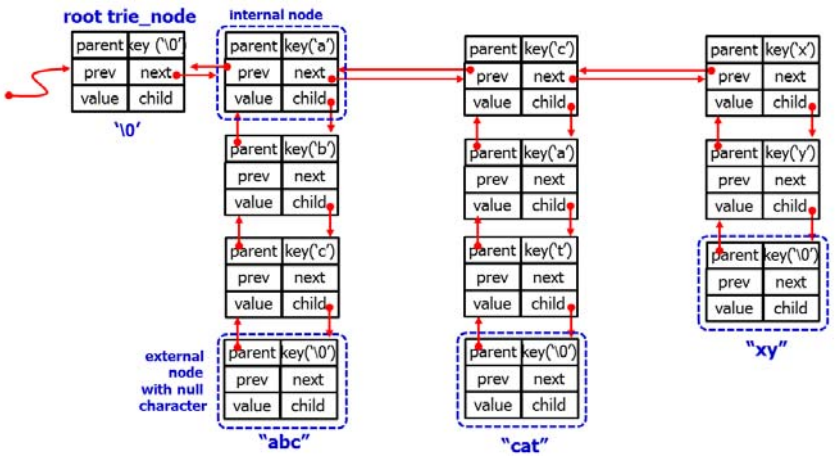
findPrefixMatch() in trie

◆ traverse(TrieNode<E> *pTN, STL_list& list_keywords)

- pTN이 가리키는 현재 위치의 prefix를 가지는 모든 단어들을 list_keywords에 담아 반환
- 만약 pTN이 "ac" prefix를 가리키는 경우, act, action을 list_keywords에 담아 반환
- 만약 pTN이 "ag" prefix를 가리키는 경우, age, aged, agenda를 list_keywords에 담아 반환



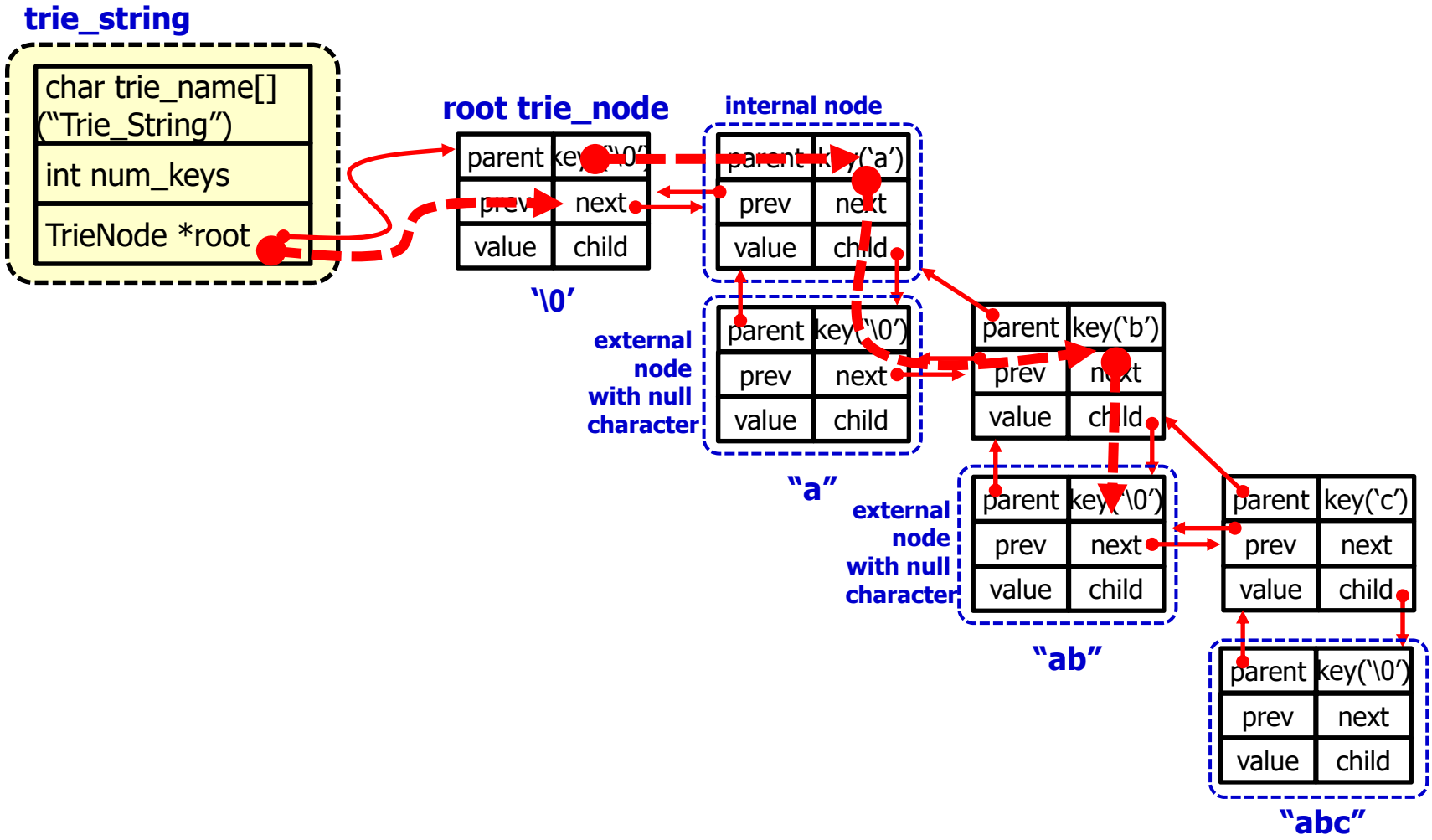
```
/* Trie.h  (9) */


template<typename E>
void Trie<E>::_traverse(TrieNode<E> *pTN, STL_list& list_keywords)
{
        if (pTN == NULL)
                return;
        if (pTN->getChild() == NULL)
        {
                list_keywords.push_back(pTN->getValue());
        }
        else
        {
                _traverse(pTN->getChild(), list_keywords);
        }

        if (pTN->getNext() != NULL)
        {
                _traverse(pTN->getNext(), list_keywords);
        }
}
```



```
/* Trie.h  (9) */

template<typename E>
void Trie<E>::findPrefixMatch(const char * keyStr, List_String& predictWords)
{
        TrieNode<E> *pPtr = NULL;
        const char *keyPtr;
        TrieNode<E> *pTN = NULL;

        TrieNode<E> *found = NULL;
        keyPtr = keyStr;

        if (keyStr == NULL)
                return;

        pTN = this->_root;
        pTN = _find(keyStr, PREFIX_MATCH);

        _traverse(pTN, predictWords);

        //printf("Error in TrieSearch (key: %s) !!\n", keyWord);

}
```

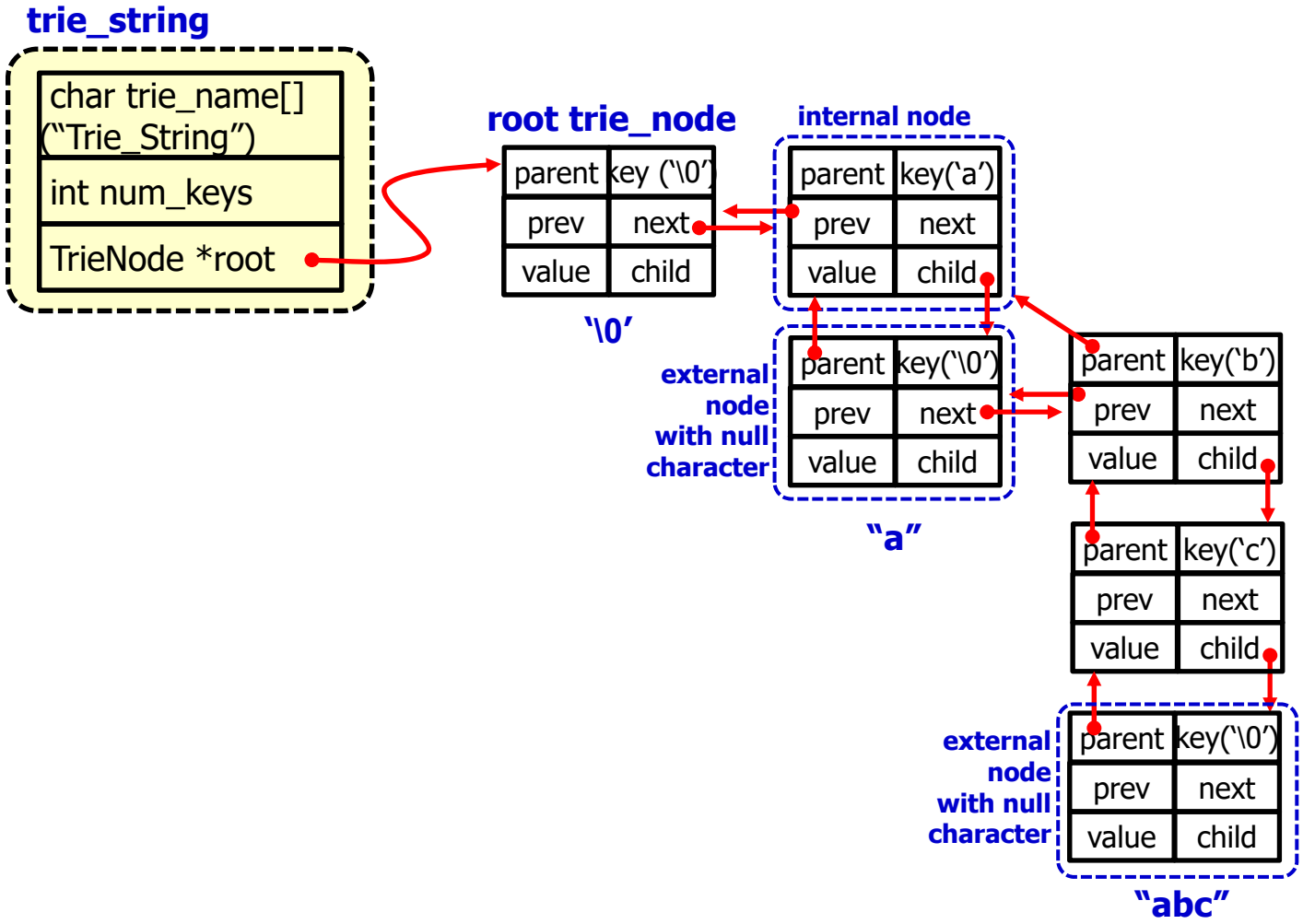
deleteKeyWord() in trie (1)

◆ find ("ab") for deleteKeyWord("ab")



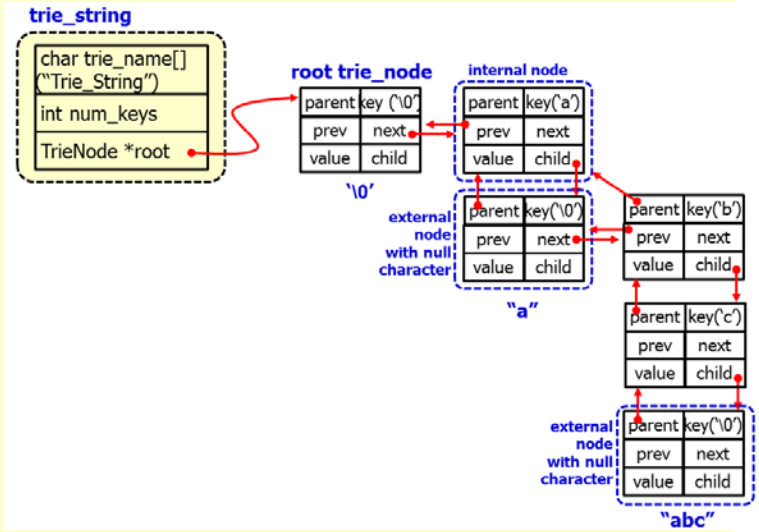
deleteKeyWord() in trie (2)

◆ delete the portion of "ab"

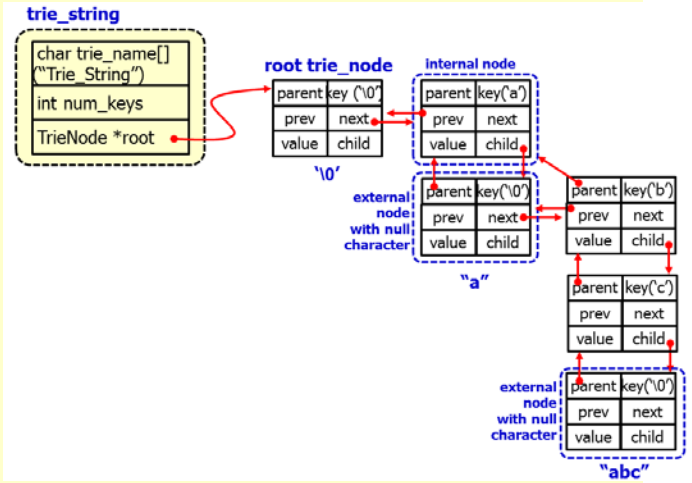


/* Trie.h (9) */

```
template<typename E>
void Trie<E>::deletekeyWord(const char * keyWord)
{
        TrieNode<E> *pTN = NULL, *_root;
        TrieNode<E> *tmp = NULL;
        int trie_val;

        _root = this->_root;
        if (NULL == _root || NULL == keyWord)
                return;

        pTN = _find(keyWord, FULL_MATCH);

        if (pTN == NULL)
        {
                cout << "Key [" << keyWord << "] not found in trie" << endl;
                return;
        }

        while (1)
        {
                if (pTN == NULL)
                        break;
                if (pTN->getPrev() && pTN->getNext())
                {
                        tmp = pTN;
                        (pTN->getNext())->setPrev(pTN->getPrev());
                        (pTN->getPrev())->setNext(pTN->getNext());
                        free(tmp);
                        break;
                }
```



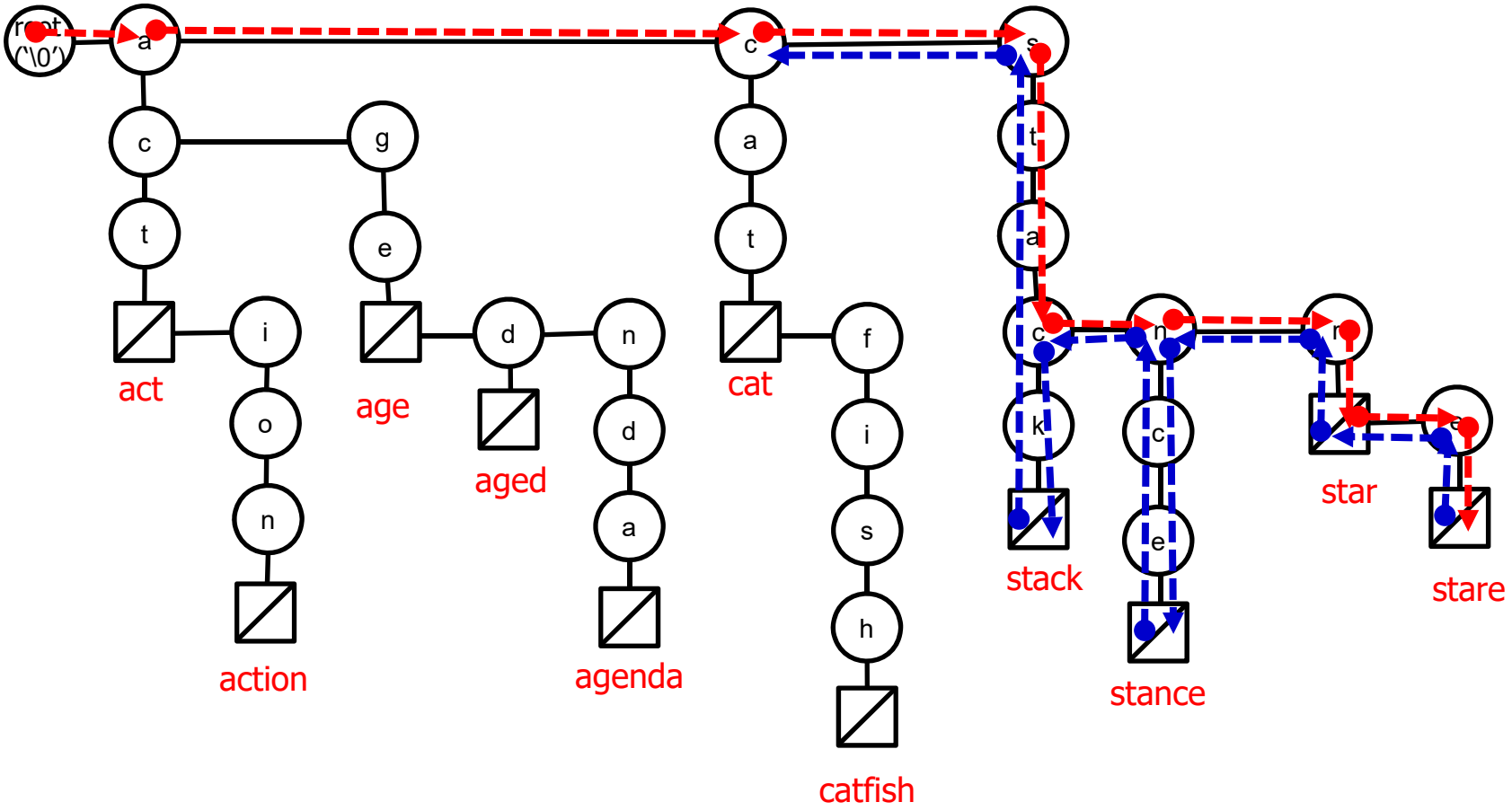
```
/* Trie.h  (10) */

                else if (pTN->getPrev() && !(pTN->getNext()))
                {
                        tmp = pTN;
                        (pTN->getPrev())->setNext(NULL);
                        free(tmp);
                        break;
                }
                else if (!(pTN->getPrev()) && pTN->getNext())
                {
                        tmp = pTN;
                        (pTN->getParent())->setChild(pTN->getNext());
                        pTN = pTN->getNext();
                        pTN->setPrev(NULL);
                        free(tmp);
                        break;
                }
                else
                {
                        tmp = pTN;
                        pTN = pTN->getParent();
                        if (pTN != NULL)
                        pTN->setChild(NULL);
                        free(tmp);
                        if ((pTN == _root) && (pTN->getNext() == NULL) && (pTN->getPrev() == NULL))
                        {
                                cout << "Now, the trie is empty !!" << endl;
                                break;
                        }
                }
        }
        this->num_keys--;
}
```



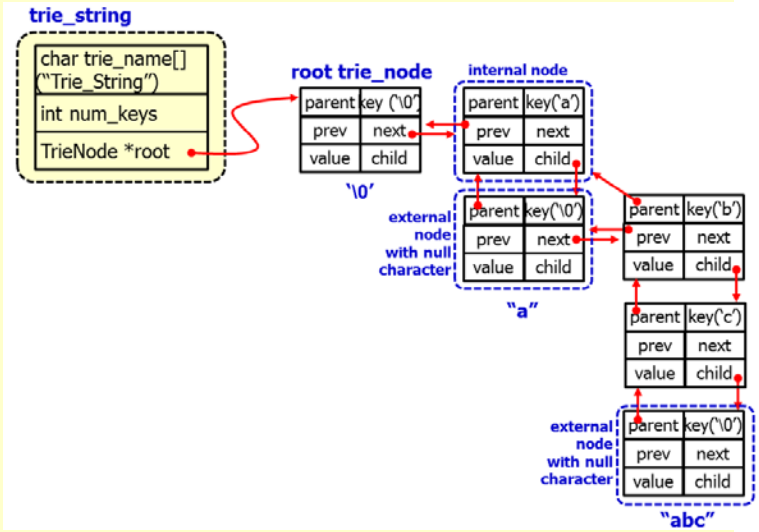
erase trie

◆ trie에 포함된 internal/external trie-node 삭제

- 가장 마지막 key string의 마지막 character 부터 역순으로 삭제

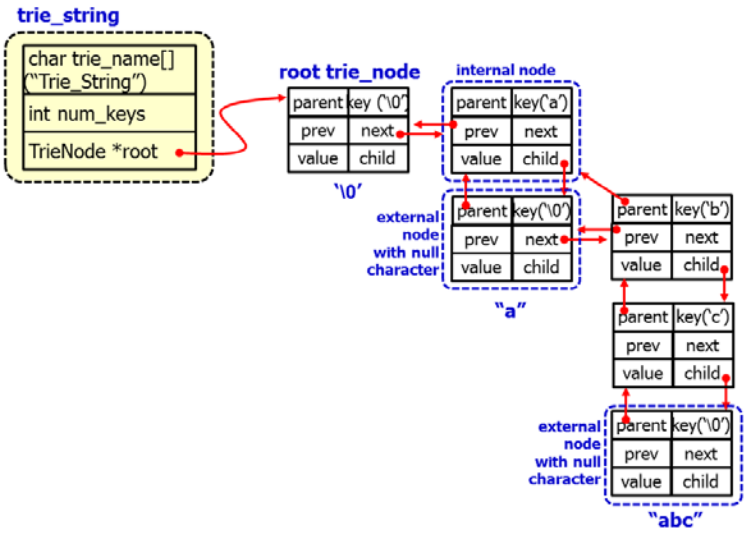


```
/* Trie.h  (11) */

template<typename E>
void Trie<E>::eraseTrie()
{
        TrieNode<E> *pTN;
        TrieNode<E> *pTN_to_be_deleted = NULL;

        if (this->_root == NULL)
                return;
        pTN = this->_root;

        /* delete the last key word first */

        while (pTN != NULL)
        {
                while ((pTN != NULL) &&(pTN->getNext()))
                        pTN = pTN->getNext();
                while (pTN->getChild())
                {
                        if (pTN->getNext())
                                break;
                        pTN = pTN->getChild();
                }
                if (pTN->getNext())
                        continue;

                if (pTN->getPrev() && pTN->getNext())
                {
                        pTN_to_be_deleted = pTN;
                        (pTN->getNext())->setPrev(pTN->getPrev());
                        (pTN->getPrev())->setNext(pTN->getNext());
                        pTN = pTN->getNext();
                        free(pTN_to_be_deleted);
                }
```



```
/* Trie.h  (11) */

            else if (pTN->getPrev() && !(pTN->getNext()))
            {
                    pTN_to_be_deleted = pTN;
                    (pTN->getPrev())->setNext(NULL);
                    pTN = pTN->getPrev();
                    free(pTN_to_be_deleted);
            }
            else if (!(pTN->getPrev()) && pTN->getNext())
            {
                    pTN_to_be_deleted = pTN;
                    (pTN->getParent())->setChild(pTN->getNext());
                    (pTN->getNext())->setPrev(NULL);
                    pTN = pTN->getNext();
                    free(pTN_to_be_deleted);
            }
            else
            {
                    pTN_to_be_deleted = pTN;
                    if (pTN == this->_root)
                    {
                            /* _root */
                            this->num_keys = 0;

                            return;
                    }
                    if (pTN->getParent() != NULL)
                    {
                            pTN = pTN->getParent();
                            pTN->setChild(NULL);
                    }
```

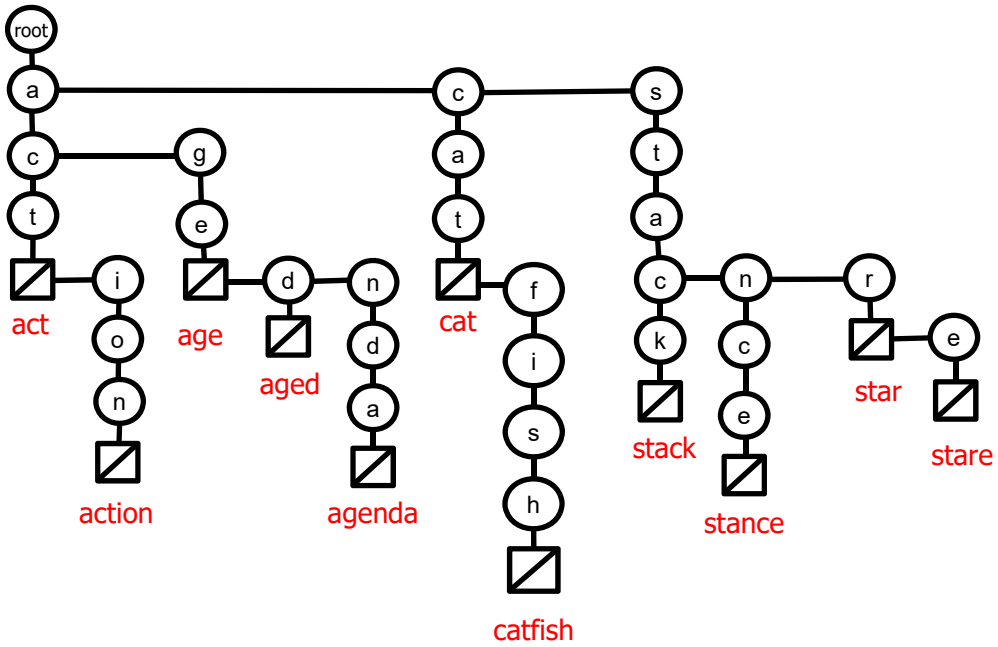
```
/* Trie.h  (11) */

                    else
                    {
                            pTN = pTN-
                            >getPrev();
                    }

                    free(pTN_to_be_deleted);
            } // end if - else
        }  // end while
}
```



fprint trie

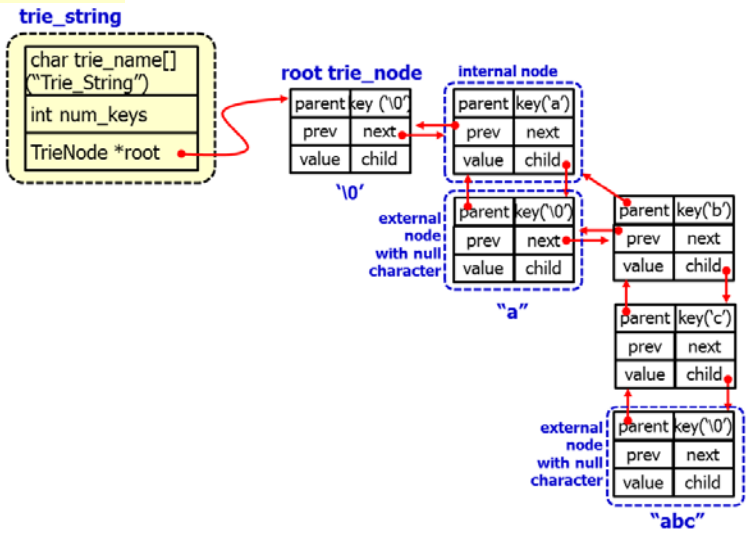
◆ trie에 포함된 모든 key string을 차례로 출력

- 들여쓰기 (indentation)를 사용하여 substring 관계를 표시

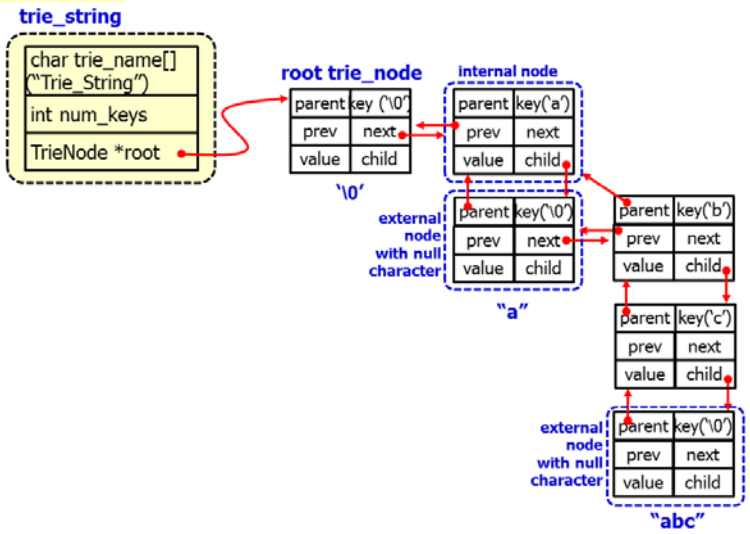



```
act          // act
   ion       // action
 ge          //age
   d         //aged
   nda       //agenda
cat          //cat
   fish      //catfish
stack        //stack
   nce       //stance
   r         //star
     e       //stare
```

```
/* Trie.h  (11) */

template<typename E>
void Trie<E>::fprintTrie(ostream& fout)
{
    TrieNode<E> *pTN;
    int line = 1, indent = 0;

    fout << "trie ( " << this->trie_name << ") with "
         << this->num_keys << " trie_nodes\n";
    if (this->num_keys == 0)
    {
        fout << "Empty trie !" << endl;
        return;
    }
    pTN = this->_root;
    pTN->_fprint(fout, pTN, indent);
}
#endif
```



```
/* Trie.h  (11) */

template<typename E>
void Trie<E>::fprintTrie(ostream& fout)
{
    TrieNode<E> *pTN;
    int line = 1, indent = 0;

    fout << "trie ( " << this->trie_name << ") with "
         << this->num_keys << " trie_nodes\n";
    if (this->num_keys == 0)
    {
        fout << "Empty trie !" << endl;
        return;
    }
    pTN = this->_root;
    pTN->_fprint(fout, pTN, indent);
}
#endif
```



trie의 C++ 프로그램 모듈 구현 (3) – main()

```
/* main_trie.cpp (1) */
#include <iostream>
#include <fstream>
#include <list>
#include "Trie.h"
#include "TrieNode.h"

using namespace std;

const char *test_strings_A[] =
{
    "a", "ab", "abc", "abcdefg", "abnormal", "abridge", "abreast", "abroad", "absence", "absolute",
    "andrew",
    "zealot", "yacht", "xerox",
    "tina",
    "arcade",
    "timor", "tim", "ti",
    "amy", "aramis",
    "best", "christmas", "beyond", "church",
    "apple", "desk", "echo", "car", "dog", "friend", "golf", "global",
    "ABCD", "XYZ", "Korea"
};
const char *test_strings_B[] =
{
    "act", "action", "age", "aged", "agenda", "cat", "stack", "stance", "star", "stare", "catfish"
};
```

```
/* main_trie.cpp (2) */

void main()
{
        ofstream fout;
        Trie<string> trieStr("TestTrie of Key Strings");
        int num_test_strings = 0;
        int trie_value;
        const char *pTest_Str;
        string sampleStr;
        TrieNode<string> *pTN;

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

        /* Testing Basic Operation in trie */
        fout << "Testing basic operations of trie inserting ..... " << endl;
        trieStr.insert("xyz", string("xyz"));
        trieStr.insert("ABCD", string("ABCD"));
        trieStr.insert("ABC", string("ABC"));
        trieStr.insert("AB", string("AB"));
        trieStr.insert("A", string("A"));
        trieStr.insert("xy", string("xy"));
        trieStr.insert("x", string("x"));
        trieStr.fprintTrie(fout);
```

```
/* main_trie.cpp (3) */

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

        /* Insert key strings into Trie_Str */
        num_test_strings = sizeof(test_strings_A) / sizeof(char *);
        fout << "\nInserting " << num_test_strings << " keywords into trie data structure.\n";
        for (int i = 0; i < num_test_strings; i++)
        {
                pTest_Str = test_strings_A[i];
                sampleStr = string(test_strings_A[i]);
                if ((pTest_Str == NULL) || (*pTest_Str == '\0'))
                        continue;
                trieStr.insert(pTest_Str, sampleStr);
                //fout << "Inserting " << i << "-th key_string " << pTest_Str << ", ";
                //trieStr.fprintTrie(fout);
                //fout.flush();
        }

        fout << "\nResult of the TrieAdd_InOder() for " << num_test_strings << "keywords : \n";
        trieStr.fprintTrie(fout);
```

```
/* main_trie.cpp (4) */

        fout << "\nTesting trie_find for " << num_test_strings << " keywords from trie data structure.\n";
        for (int i = 0; i < num_test_strings; i++)
        {
                pTest_Str = test_strings_A[i];
                if ((pTest_Str == NULL) || (*pTest_Str == '\0'))
                        continue;
                pTN = trieStr.find(pTest_Str);
                if (pTN != NULL)
                {
                        fout << "Trie_find (" << pTest_Str << ") = > trie_value(" << pTN->getValue() << ")\n";
                }
                else
                {
                        fout << "Trie_find (" << pTest_Str << ") = > not found !!\n";
                }
        }

        char prefix[] = "ab";
        List_String predictWords;
        List_String_Iter itr;
        predictWords.clear();
```

```
/* main_trie.cpp (5) */

        fout << "All predictive words with prefix (" << prefix << ") : ";
        trieStr.findPrefixMatch(prefix, predictWords);
        itr = predictWords.begin();
        for (int i = 0; i < predictWords.size(); i++)
        {
                fout << *itr << "  ";
                ++itr;
        }
        fout << endl;

        /* Testing TrieDeleteKey() */
        printf("\nTesting trie_delete_key for %d keywords from trie data structure.\n", num_test_strings);
        for (int i = 0; i < num_test_strings; i++)
        {
                pTest_Str = test_strings_A[i];
                if ((pTest_Str == NULL) || (*pTest_Str == '\0'))
                        continue;
                fout << "Trie-Deleting (key : " << pTest_Str << ") ...\n";
                trieStr.deletekeyWord(pTest_Str);
                //trieStr.fprintTrie(fout);
        }
        //trieStr.fprintTrie(fout);

        fout.close();
}
```

실행 결과

◆ trie 기본 구성 및 삭제 기능 시험

```
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 !
```

◆ Vocabulary 전체 삽입

```
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
 id
 lleviate
   oy
 mplify
     tude
 nalyze
  nealing
  onymous
 rbitrary
  gument
  tificial
 symmetric
    ptotic
 ttenuate
   ribute
 uthenticate
bandwidth
  rometer
template
  xtile
 hroughput
 olerance
  pology
 ransaction
     form
     port
   versal
  igonometric
ultra-sonic
versatile
 iolate
  rtualization
  scous
 olatile
 ulnerable
```

◆ predictive word 탐색 결과

```
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 (1)

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

<Key Points>

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

(2) trie 자료구조에 대한 설명

11.2 trie 자료구조를 구현하기 위한 class TrieNode에 대하여 그림과 pseudo code를 사용하여 설명하라.

<Key Points>

(1) class TrieNode의 데이터 멤버

(2) class TrieNode의 멤버함수들

(3) class TrieNode의 _fprint() 멤버함수

Oral Test (2)

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

<Key Points>
(1) _find() 멤버함수
(2) _traverse() 멤버함수
(3) findPrefixMatch() 멤버함수

11.4 trie 자료구조에서 주어진 키 문자열을 삽입 (insert)하는 절차에 대하여 그림과 pseudo code를 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