EngSci Press Project Final Report

Yunhao Qian

SN: 1005684225

April 9, 2020

1 Introduction

Hello.

2 Objectives

The core dictionary program should:

1. *Launch and response fast.* A slow start-up puts the user in a bad mood even before s/he starts using the program.

   Metric: Measure the time interval between a user request and its response. Shorter time in seconds is better. Start-up should take less than 1 second.

2. *Use memory efficiently.* Users might run the program on an outdated computer or a virtual machine, which typically has very limited memory. Large memory use hurts performance and can cause system failure.

   Metric: Measure the increased memory usage after loading the same dictionary dataset. Less memory in megabytes is better.

3. *Add dictionary entries easily.* The provided data have a lot of typos. Users like me might be unsatisfied and want to customize them. After following a clear and simple procedure, users should be able to add data files with the same format.

   Metric: Count the number of operations to load a CSV file into the dictionary dataset. Fewer operations are better.

The story writer program should:

1. *Produce grammatically correct sentences.* To generate meaningful and logical stories is beyond my ability. To tell my story writer apart from a monkey hitting keys, the only way is to force my production grammatically correct.

   Metric: Copy and paste the produced text into Microsoft Word. Green underlines flag grammatical errors. Fewer grammatical errors per sentence are better.

2. *Control the length of generated text accurately.* Sentence generation is slow, so it is a waste of time to work on unneeded sentences.

   Metric: Calculate the percentage difference between the user-specified length and the length of generated text. Smaller average difference is better.

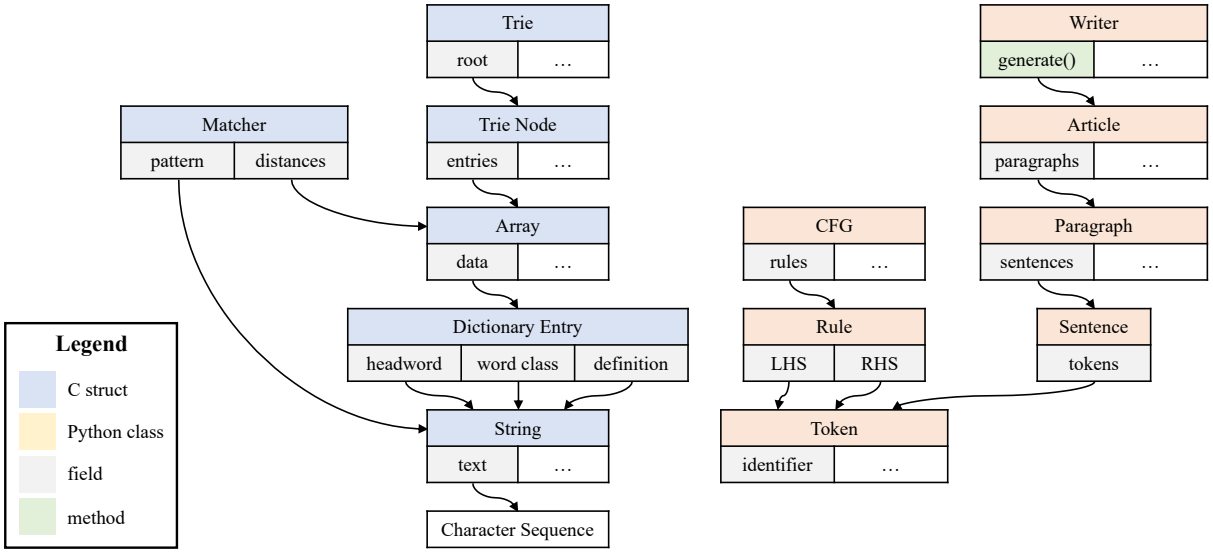

Figure 1: Overview of the project structure. Some fields and methods are omitted.

3 Detailed Framework

3.1 High-Level Overview

3.2 Languages

I use C for the core dictionary because it runs faster and provides more precise memory control. I initially wrote it in Python, but it took 3 seconds to launch and violated the time constraint. The bottleneck turns out to be CPU computation as opposed to disk IO. Moving to C should effectively speed it up since compiled languages typically compute much faster than interpreted languages.

I use Python for the story writer because it is easier to code, supports regular expression and features various sampling methods. Usage of these functionalities is described in <Section>. Python libraries such as NumPy have a mature and efficient C/Fortran back-end. Compared to reinvented wheels, they are faster, more robust and easier to debug. Moreover, exception mechanism in Python makes it simpler to handle special cases that appear in a natural language.

3.3 Data Structures

3.3.1 Dynamic Array

Many functions in EngSci Press require a resizable and contiguous array. The most straightforward implementation is a block memory which is reallocated on each resize. However, frequent `reallocs` slow down the program. To make a trade-off between fewer `reallocs` and more compact storage, my custom `Array` type applies an exponential resizing strategy. It re-

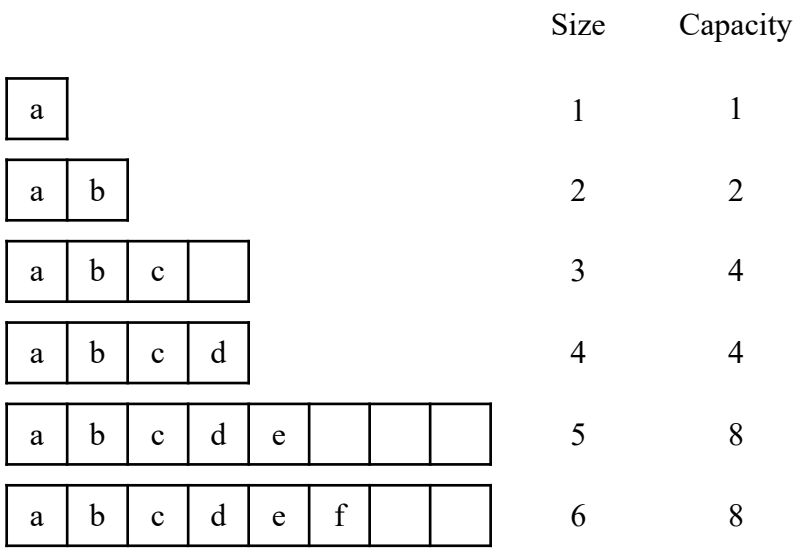

Figure 2: A dynamic array reserves space for future expansion.

serves more memory than its actual size. As shown in Figure 2, the memory space doubles when size $\geq$ capacity and halves when size $\leq$ capacity$/2$. A similar strategy is used for the `String` type.

For convenience, an `Array` of pointers is designed to hold an optional destructor and execute it upon every element deletion. The destructor function frees all the memory that an element uses, both directly and indirectly.

3.3.2 Trie

I choose trie to store, access and modify dictionary data because it is efficient and easy to implement. Trie is a tree-like data structure that implements mapping with string keys. As shown in Figure 3, each node holds a single character. The key of a node is represented by the character sequence along the root-node path.

Headwords are lower-cased as keys of dictionary entries, enabling case-insensitive search. Moreover, keys accept only characters whose ASCII codes fall in 32–64 or 97–122, because others are either upper-cased, or meaningless to appear in a dictionary headword. As a result, a trie node has 59 children at most.

For simplicity, mapping from a node to its children is implemented with a 59-element array of ordered child pointers. Fill `NULL` if a child does not exist. Such primitive implementation seems to hurt performance at first glance, as one has to check for many null pointers. However, because the accepted character set is small, a more advanced data structure, such as BST, usually brings more overhead as opposed to efficiency.

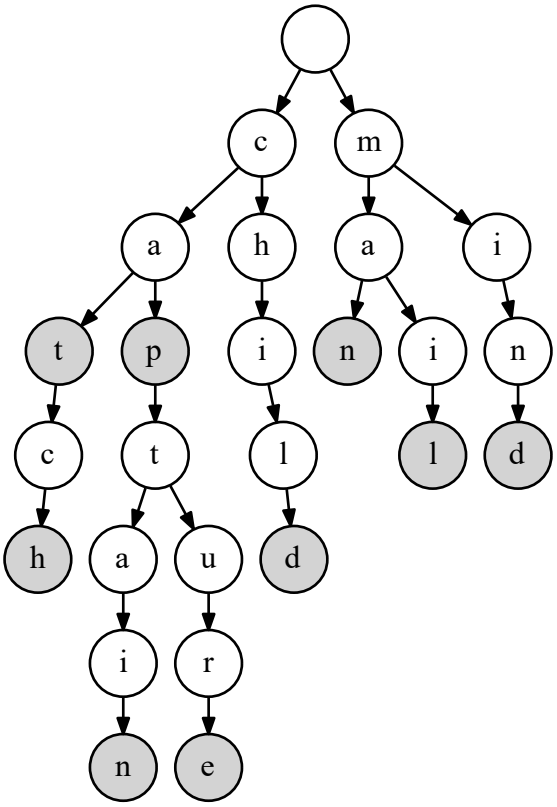

Figure 3: A trie. Each shadowed node represents an English word.

3.3.3 Levenshtein Automaton

3.4 Algorithms

3.4.1 Context-Free Grammar

EngSci Press Grammar (ESPG) is a context-free grammar (CFG) that generates English sentences. A CFG contains a start symbol (S) and describes many rewriting rules. The left-hand side (LHS) of a rule is a single token, and the right-hand side (RHS) is a sequence of at least one tokens. The rewriting process terminates when all tokens become English words (or terminals).

Stricter grammars such as regular grammar are not descriptive enough for a natural language, while more flexible ones can be challenging to implement for a generative purpose. For example, context-sensitive grammars, with more than one token on the LHS, can easily go into a blind alley, making it impossible to substitute all non-terminals.

Describing CFG with a Python `dict` is not hard, but wordy. To make my grammar more readable and maintainable, I design a simple description syntax as shown in <Figure>. The text is parsed with `re` (regular expression) module and expanded into a `class CFG` instance. To make the output more natural, this syntax supports optional tokens (i.e. RHS tokens that can be

omitted by a specified percentage chance) and weighted rules (i.e. with the same LHS, one rule has a higher execution chance than another).

3.4.2 Story Length Control

Properties of a CFG determine the length distribution of its generated sentences. CFG rules can be recursive, where the LHS token appears on the RHS. Such rules, as $A \to AB$, have a risk of falling into infinite loops: $A \to AB \to ABB \to ABBB \to \cdots$. A primitive yet effective patch for this is to set a convergence factor α. Weight of a rule decreases by α on each execution. As a result, non-recursive rules are preferred as a sentence grows longer, forcing a finite output.

When generating an article, users might also want to specify the word/paragraph count. However, imagine every paragraph has exactly the same number of sentences – such uniformity makes the output non-human-like. To address this issue, EngSci Press determines paragraph lengths by Poisson sampling, given the fact that lengths of human writings roughly follow a Poisson distribution. λ parameter of this distribution is user-defined, an indirect way to control story length.

4 Results

Hello.

5 Future Work/Conclusion

As mentioned in <Section>, EngSci Press generally meets my expectation. However, more objectives emerge as I work on this project, which might serve as a guide for future work

1. *Unicode support.* Many users look up words in a different language, so it is hard to avoid non-ASCII characters. Unfortunately, this need has not yet been satisfied as a result of limited Unicode support in C standard library. Porting from `char` to `wchar_t`, the most common approach, can cause a memory disaster, quadrupling the current 300MB to 1.2GB. I prefer variable-width encoding because it is more memory-efficient. It is very messy to implement and typically requires 3rd-party wheels.

2. *Compact trie.* Currently, a trie node can hold one character at most. However, it will be appealing to put multiple characters together if a group of adjacent nodes have only one child for each. It saves memory for sure, and might also run faster because of smaller tree heights.

3. *Gammatical agreement.* This phenomenon is common in a natural language (e.g. 3rd-person-singular verbs), but painful to describe in CFG.

A Complete Code

Contents

A.1 CMakeLists.txt

```
cmake_minimum_required(VERSION 3.2)
project(engsci-press C)

include_directories(src)

file(GLOB CORE_SOURCE src/core/*.c)
add_library(core STATIC ${CORE_SOURCE})

file(GLOB MAIN_SOURCE src/main/*.c)
add_executable(engsci_press ${MAIN_SOURCE})
target_link_libraries(engsci_press core)

file(GLOB TEST_SOURCE src/test/*.c)
foreach(source_name ${TEST_SOURCE})
    get_filename_component(executable_name ${source_name} NAME_WE)
    add_executable(${executable_name} ${source_name})
    target_link_libraries(${executable_name} core)
endforeach(source_name ${TEST_SOURCE})
```

A.2 .esp_rc

```
load ../Dictionary-in-csv/A.csv
load ../Dictionary-in-csv/B.csv
load ../Dictionary-in-csv/C.csv
load ../Dictionary-in-csv/D.csv
load ../Dictionary-in-csv/E.csv
load ../Dictionary-in-csv/F.csv
load ../Dictionary-in-csv/G.csv
load ../Dictionary-in-csv/H.csv
load ../Dictionary-in-csv/I.csv
load ../Dictionary-in-csv/J.csv
load ../Dictionary-in-csv/K.csv
load ../Dictionary-in-csv/L.csv
load ../Dictionary-in-csv/M.csv
load ../Dictionary-in-csv/N.csv
load ../Dictionary-in-csv/O.csv
load ../Dictionary-in-csv/P.csv
```

```
load ../Dictionary-in-csv/Q.csv
load ../Dictionary-in-csv/R.csv
load ../Dictionary-in-csv/S.csv
load ../Dictionary-in-csv/T.csv
load ../Dictionary-in-csv/U.csv
load ../Dictionary-in-csv/V.csv
load ../Dictionary-in-csv/W.csv
load ../Dictionary-in-csv/X.csv
load ../Dictionary-in-csv/Y.csv
load ../Dictionary-in-csv/Z.csv
```

A.3 src/core/global.h

```c
#ifndef CORE_GLOBAL_H_
#define CORE_GLOBAL_H_

#include <assert.h>
#include <ctype.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_LINE_SIZE 3000
#define ALPHABET_SIZE 59

#ifdef SUPPRESS_WARNINGS
#define WARN(...)
#else
#define WARN(...) fprintf(stderr, __VA_ARGS__)
#endif // SUPPRESS_WARNINGS

typedef void (*Destructor)(void *);

bool is_valid_key_char(char c, bool case_sensitive);
int char_to_index(char c);

#endif // CORE_GLOBAL_H_
```

A.4 src/core/global.c

```c
#include "core/global.h"

bool is_valid_key_char(char c, bool case_sensitive) {
    if (!case_sensitive && c >= 65 && c <= 90) {
```

```
        return true;
    }
    return c >= 32 && c <= 64 || c >= 97 && c <= 122;
}

int char_to_index(char c) {
    assert(is_valid_key_char(c, true) && "char_to_index: invalid character");
    if (c < 96) {
        return c - 32;
    } else {
        return c - 64;
    }
}
```

A.5 src/core/array.h

```
#ifndef CORE_ARRAY_H_
#define CORE_ARRAY_H_

#include "core/global.h"

typedef struct Array {
    void **data, **head;
    int size, capacity;
    Destructor destructor;
} Array;

Array *new_array(Destructor destructor);
void delete_array(void *array);

void array_append(Array *array, void *value);
void array_remove(Array *array, int index);

void array_reserve(Array *array, int capacity);

#endif // CORE_ARRAY_H_
```

A.6 src/core/array.c

```
#include "core/array.h"

Array *new_array(Destructor destructor) {
    Array *array = malloc(sizeof(Array));
    array->size = array->capacity = 0;
```

```
    array->destructor = destructor;
    return array;
}

void delete_array(void *p) {
    Array *array = p;
    if (array->size > 0) {
        if (array->destructor) {
            for (int i = 0; i < array->size; ++i) {
                array->destructor(array->data[i]);
            }
        }
        free(array->head);
    }
    free(array);
}

static void expand_array(Array *array) {
    if (array->capacity <= 0) {
        array->data = array->head = malloc(sizeof(void *));
        array->capacity = 1;
        return;
    }
    size_t block_size = array->size * sizeof(void *);
    if (array->size < array->capacity) {
        if (array->data + array->size >= array->head + array->capacity) {
            memmove(array->head, array->data, block_size);
        }
    } else {
        array->data = array->head = realloc(array->head, 2 * block_size);
        array->capacity *= 2;
    }
}

void array_append(Array *array, void *value) {
    expand_array(array);
    array->data[array->size] = value;
    ++array->size;
}

static void shrink_array(Array *array) {
    if (array->size <= 0) {
        free(array->head);
        array->capacity = 0;
        return;
```

```
    }
    if (array->size > array->capacity / 2) {
        return;
    }
    size_t block_size = array->size * sizeof(void *);
    void **head;
    if (array->data == array->head) {
        head = realloc(array->head, block_size);
    } else {
        head = malloc(block_size);
        memcpy(head, array->data, block_size);
        free(array->head);
    }
    array->data = array->head = head;
    array->capacity = array->size;
}

void array_remove(Array *array, int index) {
    assert(index >= 0 && index < array->size &&
           "array_remove: index out of range");
    if (array->destructor) {
        array->destructor(array->data[index]);
    }
    size_t move_size;
    if (index < array->size / 2) {
        move_size = index * sizeof(void *);
        memmove(array->data + 1, array->data, move_size);
        ++array->data;
    } else {
        move_size = (array->size - index - 1) * sizeof(void *);
        memmove(array->data + index, array->data + (index + 1), move_size);
    }
    --array->size;
    shrink_array(array);
}

void array_reserve(Array *array, int capacity) {
    assert(capacity >= array->size &&
           "array_reserve: capacity smaller than array size");
    if (capacity == array->capacity) {
        return;
    }
    size_t reserve_size = capacity * sizeof(void *);
    void **head;
    if (array->size <= 0) {
```

```
        if (array->capacity <= 0) {
            head = malloc(reserve_size);
        } else {
            head = realloc(array->head, reserve_size);
        }
    } else if (array->data == array->head) {
        head = realloc(array->head, reserve_size);
    } else {
        head = malloc(reserve_size);
        memcpy(head, array->data, array->size * sizeof(void *));
        free(array->head);
    }
    array->data = array->head = head;
    array->capacity = capacity;
}
```

A.7 src/core/string.h

```
#ifndef CORE_STRING_H_
#define CORE_STRING_H_

#include "core/array.h"

typedef struct String {
    char *text, *head;
    int size, capacity;
} String;

String *new_string(const char *text, int size);
void delete_string(void *string);

bool is_valid_key(const String *string, bool case_sensitive);
int string_index(const String *string, char value);

String *to_lower(const String *string);
void to_lower_in_place(String *string);

String *substring(const String *string, int start, int end);
void substring_in_place(String *string, int start, int end);

String *trim(const String *string);
void trim_in_place(String *string);

String *get_line(FILE *stream);

```

```
Array *split_string(const String *string);
String *join_strings(const Array *strings, char c);

bool string_start_with(const String *string, const char *prefix);

#endif // CORE_STRING_H_
```

A.8 src/core/string.c

```
#include "core/string.h"

String *new_string(const char *text, int size) {
    if (size < 0) {
        size = strlen(text);
    } else {
        assert(size <= strlen(text) &&
               "new_string: size larger than text length");
    }
    String *string = malloc(sizeof(String));
    string->text = string->head = malloc((size + 1) * sizeof(char));
    memcpy(string->text, text, size * sizeof(char));
    string->text[size] = '\0';
    string->size = string->capacity = size;
    return string;
}

void delete_string(void *p) {
    String *string = p;
    free(string->head);
    free(string);
}

bool is_valid_key(const String *string, bool case_sensitive) {
    if (string->size <= 0 || string_start_with(string, "--")) {
        return false;
    }
    for (int i = 0; i < string->size; ++i) {
        if (!is_valid_key_char(string->text[i], case_sensitive)) {
            return false;
        }
    }
    return true;
}

int string_index(const String *string, char value) {
```

```
    char *find = strchr(string->text, value);
    if (!find) {
        return -1;
    } else {
        return find - string->text;
    }
}

String *to_lower(const String *string) {
    String *lower = malloc(sizeof(String));
    lower->text = lower->head = malloc((string->size + 1) * sizeof(char));
    for (int i = 0; i <= string->size; ++i) {
        lower->text[i] = tolower(string->text[i]);
    }
    lower->size = lower->capacity = string->size;
    return lower;
}

void to_lower_in_place(String *string) {
    for (int i = 0; i < string->size; ++i) {
        string->text[i] = tolower(string->text[i]);
    }
}

static void shrink_string(String *string) {
    if (string->size > string->capacity / 2) {
        return;
    }
    size_t block_size = (string->size + 1) * sizeof(char);
    char *head;
    if (string->text == string->head) {
        head = realloc(string->head, block_size);
    } else {
        head = malloc(block_size);
        memcpy(head, string->text, block_size);
        free(string->head);
    }
    string->text = string->head = head;
    string->capacity = string->size;
}

String *substring(const String *string, int start, int end) {
    if (start >= end) {
        return new_string("", 0);
    }
```

```
    assert(start >= 0 && start < string->size &&
           "substring: start index out of range");
    assert(end >= 0 && end <= string->size &&
           "substring: end index out of range");
    return new_string(string->text + start, end - start);
}

void substring_in_place(String *string, int start, int end) {
    if (start == 0 && end == string->size) {
        return;
    }
    if (start >= end) {
        string->text = string->head;
        string->text[0] = '\0';
        string->size = 0;
        shrink_string(string);
        return;
    }
    assert(start >= 0 && start < string->size &&
           "substring_in_place: start index out of range");
    assert(end >= 0 && end <= string->size &&
           "substring_in_place: end index out of range");
    string->text += start;
    string->size = end - start;
    string->text[string->size] = '\0';
    shrink_string(string);
}

static void get_trim_indices(const String *string, int *start, int *end) {
    *end = 0;
    for (int i = string->size; i > 0; --i) {
        if (!isspace(string->text[i - 1])) {
            *end = i;
            break;
        }
    }
    *start = *end;
    for (int i = 0; i < *end; ++i) {
        if (!isspace(string->text[i])) {
            *start = i;
            break;
        }
    }
}

```

```
String *trim(const String *string) {
    int start, end;
    get_trim_indices(string, &start, &end);
    return substring(string, start, end);
}

void trim_in_place(String *string) {
    int start, end;
    get_trim_indices(string, &start, &end);
    substring_in_place(string, start, end);
}

String *get_line(FILE *stream) {
    static char buffer[MAX_LINE_SIZE + 1];
    if (!fgets(buffer, MAX_LINE_SIZE + 1, stream)) {
        return NULL;
    }
    int size = strlen(buffer);
    if (size > 0 && buffer[size - 1] == '\n') {
        --size;
    } else {
        int c;
        do {
            c = getc(stream);
        } while (c != '\n' && c != EOF);
    }
    if (size > 0 && buffer[size - 1] == '\r') {
        --size;
    }
    return new_string(buffer, size);
}

Array *split_string(const String *string) {
    Array *array = new_array(delete_string);
    int start;
    bool in_field = false;
    for (int i = 0; i < string->size; ++i) {
        if (isspace(string->text[i])) {
            if (in_field) {
                array_append(array, substring(string, start, i));
                in_field = false;
            }
        } else if (!in_field) {
            start = i;
            in_field = true;
```

```
        }
    }
    if (in_field) {
        array_append(array, substring(string, start, string->size));
    }
    return array;
}

String *join_strings(const Array *strings, char c) {
    assert(strings->size > 0 && "join_strings: string list is empty");
    int size = strings->size - 1;
    for (int i = 0; i < strings->size; ++i) {
        size += ((String *)strings->data[i])->size;
    }
    String *joined = malloc(sizeof(String)), *field;
    size_t block_size;
    joined->text = joined->head = malloc((size + 1) * sizeof(char));
    for (int i = 0, j = 0; i < strings->size; ++i) {
        if (i) {
            joined->text[j] = c;
            ++j;
        }
        field = strings->data[i];
        block_size = field->size * sizeof(char);
        memcpy(joined->text + j, field->text, block_size);
        j += block_size;
    }
    joined->text[size] = '\0';
    joined->size = joined->capacity = size;
    return joined;
}

bool string_start_with(const String *string, const char *prefix) {
    int size = strlen(prefix);
    if (size > string->size) {
        return false;
    }
    return !memcmp(string->text, prefix, size * sizeof(char));
}
```

A.9 src/core/dict_entry.h

```
#ifndef CORE_DICT_ENTRY_H_
#define CORE_DICT_ENTRY_H_

```

```h
#include "core/string.h"

typedef struct DictEntry {
    String *headword, *word_class, *definition;
} DictEntry;

DictEntry *new_dict_entry(const String *line);
void delete_dict_entry(void *entry);

bool confirm(bool default_yes, const char *message);
DictEntry *input_dict_entry(const String *headword);

void display_dict_entry(const DictEntry *entry);
void write_dict_entry(const DictEntry *entry, FILE *stream);

#endif // CORE_DICT_ENTRY_H_
```

A.10 src/core/dict_entry.c

```c
#include "core/dict_entry.h"

DictEntry *new_dict_entry(const String *line) {
    int open_index = string_index(line, '(');
    if (open_index < 0) {
        return NULL;
    }
    int close_index = -1;
    for (int i = open_index + 1, depth = 1; i < line->size; ++i) {
        if (line->text[i] == '(') {
            ++depth;
        } else if (line->text[i] == ')' && --depth <= 0) {
            close_index = i;
            break;
        }
    }
    if (close_index < 0) {
        return NULL;
    }
    DictEntry *entry = malloc(sizeof(DictEntry));
    int begin_index = 0, end_index = line->size;
    if (line->text[0] == '"' && line->text[line->size - 1] == '"') {
        ++begin_index;
        --end_index;
    }
    entry->headword = substring(line, begin_index, open_index);
```

```
    trim_in_place(entry->headword);
    if (!is_valid_key(entry->headword, false)) {
        WARN("Invalid headword: %s\n", entry->headword->text);
        delete_string(entry->headword);
        free(entry);
        return NULL;
    }
    entry->word_class = substring(line, open_index + 1, close_index);
    entry->definition = substring(line, close_index + 1, end_index);
    trim_in_place(entry->word_class);
    trim_in_place(entry->definition);
    return entry;
}

void delete_dict_entry(void *p) {
    DictEntry *entry = p;
    delete_string(entry->headword);
    delete_string(entry->word_class);
    delete_string(entry->definition);
    free(entry);
}

bool confirm(bool default_yes, const char *message) {
    if (message) {
        printf("%s ", message);
    }
    if (default_yes) {
        printf("([y]/n) ");
    } else {
        printf("(y/[n]) ");
    }
    bool returned;
    String *line = get_line(stdin);
    trim_in_place(line);
    to_lower_in_place(line);
    if (!strcmp(line->text, "y") || !strcmp(line->text, "yes")) {
        returned = true;
    } else if (!strcmp(line->text, "n") || !strcmp(line->text, "no")) {
        returned = false;
    } else {
        returned = default_yes;
    }
    delete_string(line);
    return returned;
}
```

```
72
73  DictEntry *input_dict_entry(const String *headword) {
74      if (!is_valid_key(headword, false)) {
75          WARN("Invalid headword: %s\nDo nothing.\n", headword->text);
76          return NULL;
77      }
78      DictEntry *entry = malloc(sizeof(DictEntry));
79      entry->headword = trim(headword);
80      printf("Word class: ");
81      entry->word_class = get_line(stdin);
82      printf("Definition: ");
83      entry->definition = get_line(stdin);
84      trim_in_place(entry->word_class);
85      trim_in_place(entry->definition);
86      printf("Will create the following entry:\n");
87      display_dict_entry(entry);
88      if (!confirm(true, "Continue?")) {
89          delete_dict_entry(entry);
90          printf("Do nothing.\n");
91          return NULL;
92      } else {
93          return entry;
94      }
95  }
96
97  void display_dict_entry(const DictEntry *entry) {
98      printf("%s\n%s\n%s\n", entry->headword->text, entry->word_class->text,
99             entry->definition->text);
100 }
101
102 void write_dict_entry(const DictEntry *entry, FILE *stream) {
103     fprintf(stream, "%s (%s) %s\n", entry->headword->text,
104             entry->word_class->text, entry->definition->text);
105 }
```

A.11 src/core/trie_node.h

```
1  #ifndef CORE_TRIE_NODE_H_
2  #define CORE_TRIE_NODE_H_
3
4  #include "core/dict_entry.h"
5
6  typedef struct TrieNode {
7      char letter;
8      Array *entries;
```

```
    struct TrieNode *parent, *children[ALPHABET_SIZE];
    int child_count;
} TrieNode;

TrieNode *new_trie_node(char letter, TrieNode *parent);
void delete_trie_node(void *node);

TrieNode *previous_trie_node(const TrieNode *node);
TrieNode *next_trie_node(const TrieNode *node);

TrieNode *trie_node_add_child(TrieNode *node, char letter);
void trie_node_remove_child(TrieNode *node, char letter);

#endif // CORE_TRIE_NODE_H_
```

A.12 src/core/trie_node.c

```
#include "core/trie_node.h"

TrieNode *new_trie_node(char letter, TrieNode *parent) {
    assert((!parent || is_valid_key_char(letter, true)) &&
           "new_trie_node: invalid character");
    TrieNode *node = malloc(sizeof(TrieNode));
    node->letter = letter;
    node->entries = new_array(delete_dict_entry);
    node->parent = parent;
    for (int i = 0; i < ALPHABET_SIZE; ++i) {
        node->children[i] = NULL;
    }
    node->child_count = 0;
    return node;
}

void delete_trie_node(void *p) {
    TrieNode *node = p;
    delete_array(node->entries);
    free(node);
}

static TrieNode *max_trie_leaf(TrieNode *node) {
    if (node->child_count <= 0) {
        return node;
    }
    for (int i = ALPHABET_SIZE - 1; i >= 0; --i) {
        if (node->children[i]) {
```

```
            return max_trie_leaf(node->children[i]);
        }
    }
    assert(false && "max_trie_leaf: unreachable code");
    return NULL;
}

TrieNode *previous_trie_node(const TrieNode *node) {
    TrieNode *parent = node->parent;
    if (!parent) {
        return NULL;
    }
    for (int i = char_to_index(node->letter) - 1; i >= 0; --i) {
        if (parent->children[i]) {
            return max_trie_leaf(parent->children[i]);
        }
    }
    return parent;
}

TrieNode *next_trie_node(const TrieNode *node) {
    if (node->child_count > 0) {
        for (int i = 0; i < ALPHABET_SIZE; ++i) {
            if (node->children[i]) {
                return node->children[i];
            }
        }
    }
    TrieNode *parent = node->parent;
    int index;
    while (parent) {
        index = char_to_index(node->letter);
        for (int i = index + 1; i < ALPHABET_SIZE; ++i) {
            if (parent->children[i]) {
                return parent->children[i];
            }
        }
        node = parent;
        parent = node->parent;
    }
    return NULL;
}

TrieNode *trie_node_add_child(TrieNode *node, char letter) {
    int index = char_to_index(letter);
```

```
    assert(!node->children[index] &&
           "trie_node_add_child: child already exists");
    node->children[index] = new_trie_node(letter, node);
    ++node->child_count;
    return node->children[index];
}

void trie_node_remove_child(TrieNode *node, char letter) {
    int index = char_to_index(letter);
    assert(node->child_count > 0 && node->children[index] &&
           "trie_node_remove_child: child does not exist");
    assert(node->children[index]->child_count <= 0 &&
           "trie_node_remove_child: remove a non-leaf child");
    delete_trie_node(node->children[index]);
    node->children[index] = NULL;
    --node->child_count;
}
```

A.13 src/core/trie.h

```
#ifndef CORE_TRIE_H_
#define CORE_TRIE_H_

#include "core/trie_node.h"

typedef struct Trie {
    TrieNode *root;
    int size;
} Trie;

Trie *new_trie();
void delete_trie(void *trie);

Array *trie_search(const Trie *trie, const String *word, bool
    case_sensitive);
void trie_insert(Trie *trie, DictEntry *entry);
void trie_remove(Trie *trie, const String *word, bool case_sensitive);

String *trie_predecessor(const Trie *trie, const String *word);
String *trie_successor(const Trie *trie, const String *word);

Array *traverse_trie(const Trie *trie);

#endif // CORE_TRIE_H_
```

A.14 src/core/trie.c

```c
#include "core/trie.h"

Trie *new_trie() {
    Trie *trie = malloc(sizeof(Trie));
    trie->root = new_trie_node('\0', NULL);
    trie->size = 0;
    return trie;
}

static void clear_trie_nodes(TrieNode *root) {
    assert(root && "clear_trie_nodes: root node is null");
    if (root->child_count > 0) {
        for (int i = 0; i < ALPHABET_SIZE; ++i) {
            if (root->children[i]) {
                clear_trie_nodes(root->children[i]);
            }
        }
    }
    delete_trie_node(root);
}

void delete_trie(void *p) {
    Trie *trie = p;
    clear_trie_nodes(trie->root);
    free(trie);
}

static TrieNode *get_trie_node(const Trie *trie, const String *word) {
    TrieNode *node = trie->root;
    for (int i = 0; i < word->size; ++i) {
        int index = char_to_index(tolower(word->text[i]));
        node = node->children[index];
        if (!node) {
            return NULL;
        }
    }
    return node;
}

Array *trie_search(const Trie *trie, const String *word, bool
    case_sensitive) {
    Array *entries = new_array(NULL);
    if (!is_valid_key(word, false)) {
```

```
        WARN("Invalid headword: %s\n", word->text);
        return entries;
    }
    TrieNode *node = get_trie_node(trie, word);
    if (!node || node->entries->size <= 0) {
        return entries;
    }
    if (case_sensitive) {
        DictEntry *entry;
        for (int i = 0; i < node->entries->size; ++i) {
            entry = node->entries->data[i];
            if (!strcmp(entry->headword->text, word->text)) {
                array_append(entries, entry);
            }
        }
    } else {
        for (int i = 0; i < node->entries->size; ++i) {
            array_append(entries, node->entries->data[i]);
        }
    }
    return entries;
}

void trie_insert(Trie *trie, DictEntry *entry) {
    assert(is_valid_key(entry->headword, false) &&
           "trie_insert: invalid entry headword");
    String *lowered = to_lower(entry->headword);
    TrieNode *node = trie->root;
    for (int i = 0; i < lowered->size; ++i) {
        int index = char_to_index(lowered->text[i]);
        if (!node->children[index]) {
            node = trie_node_add_child(node, lowered->text[i]);
        } else {
            node = node->children[index];
        }
    }
    delete_string(lowered);
    array_append(node->entries, entry);
    ++trie->size;
}

void trie_remove(Trie *trie, const String *word, bool case_sensitive) {
    assert(is_valid_key(word, false) && "trie_remove: invalid headword");
    TrieNode *node = get_trie_node(trie, word);
    if (!node) {
```

```
88          return;
89      }
90      if (case_sensitive) {
91          int i = 0;
92          DictEntry *entry;
93          while (i < node->entries->size) {
94              entry = node->entries->data[i];
95              if (!strcmp(entry->headword->text, word->text)) {
96                  array_remove(node->entries, i);
97                  --trie->size;
98              } else {
99                  ++i;
100             }
101         }
102     } else {
103         for (int i = node->entries->size - 1; i >= 0; --i) {
104             array_remove(node->entries, i);
105             --trie->size;
106         }
107     }
108     char letter;
109     while (node->entries->size <= 0 && node->child_count <= 0 && node->
    parent) {
110         letter = node->letter;
111         node = node->parent;
112         trie_node_remove_child(node, letter);
113     }
114 }
115
116 String *trie_predecessor(const Trie *trie, const String *word) {
117     TrieNode *node = get_trie_node(trie, word);
118     if (!node) {
119         WARN("Cannot find word: %s\n", word->text);
120         return NULL;
121     }
122     node = previous_trie_node(node);
123     while (node) {
124         if (node->entries->size > 0) {
125             DictEntry *entry = node->entries->data[0];
126             return to_lower(entry->headword);
127         }
128         node = previous_trie_node(node);
129     }
130     return NULL;
131 }
```

```

String *trie_successor(const Trie *trie, const String *word) {
    TrieNode *node = get_trie_node(trie, word);
    if (!node) {
        WARN("Cannot find word: %s\n", word->text);
        return NULL;
    }
    node = next_trie_node(node);
    while (node) {
        if (node->entries->size > 0) {
            DictEntry *entry = node->entries->data[0];
            return to_lower(entry->headword);
        }
        node = next_trie_node(node);
    }
    return NULL;
}

static void traverse_trie_nodes(const TrieNode *root, Array *entries) {
    for (int i = 0; i < root->entries->size; ++i) {
        array_append(entries, root->entries->data[i]);
    }
    if (root->child_count > 0) {
        for (int i = 0; i < ALPHABET_SIZE; ++i) {
            if (root->children[i]) {
                traverse_trie_nodes(root->children[i], entries);
            }
        }
    }
}

Array *traverse_trie(const Trie *trie) {
    Array *entries = new_array(NULL);
    traverse_trie_nodes(trie->root, entries);
    return entries;
}
```

A.15 src/core/matcher.h

```
#ifndef CORE_TRIE_H_
#define CORE_TRIE_H_

#include "core/trie_node.h"

typedef struct Trie {
```

```
    TrieNode *root;
    int size;
} Trie;

Trie *new_trie();
void delete_trie(void *trie);

Array *trie_search(const Trie *trie, const String *word, bool
    case_sensitive);
void trie_insert(Trie *trie, DictEntry *entry);
void trie_remove(Trie *trie, const String *word, bool case_sensitive);

String *trie_predecessor(const Trie *trie, const String *word);
String *trie_successor(const Trie *trie, const String *word);

Array *traverse_trie(const Trie *trie);

#endif // CORE_TRIE_H_
```

A.16 src/core/matcher.c

```
#include "core/trie.h"

Trie *new_trie() {
    Trie *trie = malloc(sizeof(Trie));
    trie->root = new_trie_node('\0', NULL);
    trie->size = 0;
    return trie;
}

static void clear_trie_nodes(TrieNode *root) {
    assert(root && "clear_trie_nodes: root node is null");
    if (root->child_count > 0) {
        for (int i = 0; i < ALPHABET_SIZE; ++i) {
            if (root->children[i]) {
                clear_trie_nodes(root->children[i]);
            }
        }
    }
    delete_trie_node(root);
}

void delete_trie(void *p) {
    Trie *trie = p;
    clear_trie_nodes(trie->root);
```

```
25      free(trie);
26  }
27
28  static TrieNode *get_trie_node(const Trie *trie, const String *word) {
29      TrieNode *node = trie->root;
30      for (int i = 0; i < word->size; ++i) {
31          int index = char_to_index(tolower(word->text[i]));
32          node = node->children[index];
33          if (!node) {
34              return NULL;
35          }
36      }
37      return node;
38  }
39
40  Array *trie_search(const Trie *trie, const String *word, bool
        case_sensitive) {
41      Array *entries = new_array(NULL);
42      if (!is_valid_key(word, false)) {
43          WARN("Invalid headword: %s\n", word->text);
44          return entries;
45      }
46      TrieNode *node = get_trie_node(trie, word);
47      if (!node || node->entries->size <= 0) {
48          return entries;
49      }
50      if (case_sensitive) {
51          DictEntry *entry;
52          for (int i = 0; i < node->entries->size; ++i) {
53              entry = node->entries->data[i];
54              if (!strcmp(entry->headword->text, word->text)) {
55                  array_append(entries, entry);
56              }
57          }
58      } else {
59          for (int i = 0; i < node->entries->size; ++i) {
60              array_append(entries, node->entries->data[i]);
61          }
62      }
63      return entries;
64  }
65
66  void trie_insert(Trie *trie, DictEntry *entry) {
67      assert(is_valid_key(entry->headword, false) &&
68             "trie_insert: invalid entry headword");
```

```
69      String *lowered = to_lower(entry->headword);
70      TrieNode *node = trie->root;
71      for (int i = 0; i < lowered->size; ++i) {
72          int index = char_to_index(lowered->text[i]);
73          if (!node->children[index]) {
74              node = trie_node_add_child(node, lowered->text[i]);
75          } else {
76              node = node->children[index];
77          }
78      }
79      delete_string(lowered);
80      array_append(node->entries, entry);
81      ++trie->size;
82  }
83
84  void trie_remove(Trie *trie, const String *word, bool case_sensitive) {
85      assert(is_valid_key(word, false) && "trie_remove: invalid headword");
86      TrieNode *node = get_trie_node(trie, word);
87      if (!node) {
88          return;
89      }
90      if (case_sensitive) {
91          int i = 0;
92          DictEntry *entry;
93          while (i < node->entries->size) {
94              entry = node->entries->data[i];
95              if (!strcmp(entry->headword->text, word->text)) {
96                  array_remove(node->entries, i);
97                  --trie->size;
98              } else {
99                  ++i;
100             }
101         }
102     } else {
103         for (int i = node->entries->size - 1; i >= 0; --i) {
104             array_remove(node->entries, i);
105             --trie->size;
106         }
107     }
108     char letter;
109     while (node->entries->size <= 0 && node->child_count <= 0 && node->
        parent) {
110         letter = node->letter;
111         node = node->parent;
112         trie_node_remove_child(node, letter);
```

```
    }
}

String *trie_predecessor(const Trie *trie, const String *word) {
    TrieNode *node = get_trie_node(trie, word);
    if (!node) {
        WARN("Cannot find word: %s\n", word->text);
        return NULL;
    }
    node = previous_trie_node(node);
    while (node) {
        if (node->entries->size > 0) {
            DictEntry *entry = node->entries->data[0];
            return to_lower(entry->headword);
        }
        node = previous_trie_node(node);
    }
    return NULL;
}

String *trie_successor(const Trie *trie, const String *word) {
    TrieNode *node = get_trie_node(trie, word);
    if (!node) {
        WARN("Cannot find word: %s\n", word->text);
        return NULL;
    }
    node = next_trie_node(node);
    while (node) {
        if (node->entries->size > 0) {
            DictEntry *entry = node->entries->data[0];
            return to_lower(entry->headword);
        }
        node = next_trie_node(node);
    }
    return NULL;
}

static void traverse_trie_nodes(const TrieNode *root, Array *entries) {
    for (int i = 0; i < root->entries->size; ++i) {
        array_append(entries, root->entries->data[i]);
    }
    if (root->child_count > 0) {
        for (int i = 0; i < ALPHABET_SIZE; ++i) {
            if (root->children[i]) {
                traverse_trie_nodes(root->children[i], entries);
```

```
            }
        }
    }
}

Array *traverse_trie(const Trie *trie) {
    Array *entries = new_array(NULL);
    traverse_trie_nodes(trie->root, entries);
    return entries;
}
```

A.17 src/test/global.h

```
#ifndef TEST_GLOBAL_H_
#define TEST_GLOBAL_H_

#include "core/array.h"
#include "core/dict_entry.h"
#include "core/matcher.h"
#include "core/string.h"
#include "core/trie.h"
#include "core/trie_node.h"

#define CHECK_BASE(equality, expect, actual, format)
        \
    do {
        \
        ++test_count;
        \
        if (equality) {
        \
            ++pass_count;
        \
        } else {
        \
            fprintf(stderr,
        \
                    "%s: %d\nexpect: " format "\nactual: " format "\n",
        \
                    __FILE__, __LINE__, expect, actual);
        \
        }
        \
    } while (false)

```

```
#define DISPLAY_TEST_RESULT()
        \
    printf("%d/%d (%g%%) passed\n", pass_count, test_count,
        \
            pass_count * 100.0 / test_count)

#define CHECK_BOOL(expect, actual)
        \
    CHECK_BASE((expect) == (actual), (expect) ? "true" : "false",
        \
                (actual) ? "true" : "false", "%s")

#define CHECK_INT(expect, actual)
        \
    CHECK_BASE((expect) == (actual), expect, actual, "%d")

#define CHECK_STRING(expect, actual)
        \
    CHECK_BASE(!strcmp(expect, (actual)->text), expect, (actual)->text, "%s
    ")

#endif // TEST_GLOBAL_H_
```

A.18 src/test/array.c

```
#include "test/global.h"

int test_count = 0, pass_count = 0;

#define N 50

int data[N];

void generate_data() {
    for (int i = 0; i < N; ++i) {
        data[i] = rand();
    }
}

void test_append() {
    generate_data();
    Array *array = new_array(free);
    int *element, *random;
    for (int i = 0; i < N; ++i) {
        element = malloc(sizeof(int));
```

```
        *element = data[i];
        random = malloc(sizeof(int));
        *random = rand();
        array_append(array, random);
        CHECK_INT(i + 1, array->size);
        array_append(array, element);
        CHECK_INT(i + 2, array->size);
        array_remove(array, i);
        CHECK_INT(i + 1, array->size);
    }
    for (int i = 0; i < N; ++i) {
        CHECK_INT(data[i], *(int *)(array->data[i]));
    }
    delete_array(array);
}

void test_remove() {
    generate_data();
    Array *array = new_array(free);
    int *element;
    for (int i = 0; i < N; ++i) {
        element = malloc(sizeof(int));
        *element = data[i];
        array_append(array, element);
    }
    for (int i = 0; i < N; ++i) {
        for (int j = i; j < N; ++j) {
            CHECK_INT(data[j], *(int *)(array->data[j - i]));
        }
        array_remove(array, 0);
    }
    CHECK_INT(0, array->size);
    delete_array(array);
}

void test_reserve() {
    generate_data();
    Array *array = new_array(free);
    array_reserve(array, N - 1);
    int *element;
    for (int i = 0; i < N; ++i) {
        element = malloc(sizeof(int));
        *element = data[i];
        CHECK_INT(N - 1, array->capacity);
        array_append(array, element);
```

```
    }
    CHECK_INT((N - 1) * 2, array->capacity);
    delete_array(array);
}

int main() {
    test_append();
    test_remove();
    test_reserve();
    DISPLAY_TEST_RESULT();
}
```

A.19 src/test/string.c

```
#include "test/global.h"

int test_count = 0, pass_count = 0;

#define N 10

char *texts[N] = {
    "pen pineapple apple pie",
    "  banana nanana  ",
    "iPhone XS Max",
    "\tIt was the age of wisdom.\tIt was the age of foolishness.\t",
    "gcc -g string.c -o string.o",
    "Engineering Science",
    " University of Toronto ",
    " \t \t \t \t ",
    "Designed by Apple in California",
    " ESC190 Teaching Team"};

String *strings[N];

void create_strings() {
    for (int i = 0; i < N; ++i) {
        strings[i] = new_string(texts[i], -1);
    }
}

void discard_strings() {
    for (int i = 0; i < N; ++i) {
        delete_string(strings[i]);
    }
}
```

```
32
33 char find_chars[N] = {'a', 'b', 'x', 'F', 'e', 'S', 'g', 'h', 'i', 't'};
34
35 int expect_indices[N] = {8, 2, 12, -1, -1, 12, -1, -1, 3, -1};
36
37 void test_index() {
38     int index;
39     create_strings();
40     for (int i = 0; i < N; ++i) {
41         index = string_index(strings[i], find_chars[i]);
42         CHECK_INT(expect_indices[i], index);
43     }
44     discard_strings();
45 }
46
47 char *lower_expect[N] = {
48     "pen pineapple apple pie",
49     "  banana nanana  ",
50     "iphone xs max",
51     "\tit was the age of wisdom.\tit was the age of foolishness.\t",
52     "gcc -g string.c -o string.o",
53     "engineering science",
54     " university of toronto ",
55     " \t \t \t \t ",
56     "designed by apple in california",
57     " esc190 teaching team"};
58
59 void test_to_lower() {
60     String *lowered;
61     create_strings();
62     for (int i = 0; i < N; ++i) {
63         lowered = to_lower(strings[i]);
64         CHECK_STRING(lower_expect[i], lowered);
65         delete_string(lowered);
66         to_lower_in_place(strings[i]);
67         CHECK_STRING(lower_expect[i], strings[i]);
68     }
69     discard_strings();
70 }
71
72 char *trim_expect[N] = {
73     "pen pineapple apple pie",
74     "banana nanana",
75     "iPhone XS Max",
76     "It was the age of wisdom.\tIt was the age of foolishness.",
```

```
    "gcc -g string.c -o string.o",
    "Engineering Science",
    "University of Toronto",
    "",
    "Designed by Apple in California",
    "ESC190 Teaching Team"};

void test_trim() {
    String *trimmed;
    create_strings();
    for (int i = 0; i < N; ++i) {
        trimmed = trim(strings[i]);
        CHECK_STRING(trim_expect[i], trimmed);
        delete_string(trimmed);
        trim_in_place(strings[i]);
        CHECK_STRING(trim_expect[i], strings[i]);
    }
    discard_strings();
}

int substring_indices[N][2] = {{4, 13}, {9, 15},  {7, 9}, {19, 25}, {7, 15},
                               {0, 11}, {15, 22}, {5, 8}, {9, 17},  {6, 16}};

char *substring_expect[N] = {"pineapple", "nanana",      "XS",      "wisdom",
                             "string.c",  "Engineering", "Toronto", "\t \t",
                             "by Apple",  "0 Teaching"};

void test_substring() {
    String *result;
    create_strings();
    for (int i = 0; i < N; ++i) {
        result = substring(strings[i], substring_indices[i][0],
                           substring_indices[i][1]);
        CHECK_STRING(substring_expect[i], result);
        delete_string(result);
        substring_in_place(strings[i], substring_indices[i][0],
                           substring_indices[i][1]);
        CHECK_STRING(substring_expect[i], strings[i]);
    }
    discard_strings();
}
```

```
118
119 int split_counts[N] = {4, 2, 3, 12, 5, 2, 3, 0, 5, 3};
120
121 char *split_fields[N][15] = {{"pen", "pineapple", "apple", "pie"},
122                              {"banana", "nanana"},
123                              {"iPhone", "XS", "Max"},
124                              {"It", "was", "the", "age", "of", "wisdom.", "
    It",
125                               "was", "the", "age", "of", "foolishness."},
126                              {"gcc", "-g", "string.c", "-o", "string.o"},
127                              {"Engineering", "Science"},
128                              {"University", "of", "Toronto"},
129                              {},
130                              {"Designed", "by", "Apple", "in", "California"
    },
131                              {"ESC190", "Teaching", "Team"}};
132
133 void test_split() {
134     Array *array;
135     create_strings();
136     for (int i = 0; i < N; ++i) {
137         array = split_string(strings[i]);
138         CHECK_INT(split_counts[i], array->size);
139         if (split_counts[i] != array->size) {
140             continue;
141         }
142         for (int j = 0; j < split_counts[i]; ++j) {
143             CHECK_STRING(split_fields[i][j], (String *)(array->data[j]));
144         }
145         delete_array(array);
146     }
147     discard_strings();
148 }
149
150 int join_counts[N] = {4, 2, 3, 12, 5, 2, 3, 1, 5, 3};
151
152 char *join_fields[N][15] = {{"pen", "pineapple", "apple", "pie"},
153                             {"banana", "nanana"},
154                             {"iPhone", "XS", "Max"},
155                             {"It", "was", "the", "age", "of", "wisdom.", "
    It",
156                              "was", "the", "age", "of", "foolishness."},
157                             {"gcc", "-g", "string.c", "-o", "string.o"},
158                             {"Engineering", "Science"},
159                             {"University", "of", "Toronto"},
```

```
                            {"t"},
                            {"Designed", "by", "Apple", "in", "California"
    },
                            {"ESC190", "Teaching", "Team"}};

char *join_expect[N] = {
    "pen pineapple apple pie",
    "banana nanana",
    "iPhone XS Max",
    "It was the age of wisdom. It was the age of foolishness.",
    "gcc -g string.c -o string.o",
    "Engineering Science",
    "University of Toronto",
    "t",
    "Designed by Apple in California",
    "ESC190 Teaching Team"};

void test_join() {
    Array *strings;
    String *field, *joined;
    for (int i = 0; i < N; ++i) {
        strings = new_array(delete_string);
        for (int j = 0; j < join_counts[i]; ++j) {
            field = new_string(join_fields[i][j], -1);
            array_append(strings, field);
        }
        joined = join_strings(strings, ' ');
        CHECK_STRING(join_expect[i], joined);
        delete_string(joined);
        delete_array(strings);
    }
}

const char *prefixes[N] = {"pen",        "  banana", "iPhone XS Max Pro",
                           "",           "gcc -g",   "Engineering Science",
                           "University", " \t",      "Designed by",
                           "ESC190 "};

bool start_expect[N] = {true, true,  false, true, true,
                        true, false, true,  true, false};

void test_start_with() {
    String *string;
    bool start;
    for (int i = 0; i < N; ++i) {
```

```
        string = new_string(texts[i], -1);
        start = string_start_with(string, prefixes[i]);
        delete_string(string);
        CHECK_BOOL(start_expect[i], start);
    }
}

int main() {
    test_index();
    test_to_lower();
    test_substring();
    test_trim();
    test_split();
    test_join();
    test_start_with();
    DISPLAY_TEST_RESULT();
}
```

A.20 src/test/dict_entry.c

```
#include "test/global.h"

int test_count = 0, pass_count = 0;

#define N 15

const char *lines[N] = {
    " Apple pie  ( n. )  A kind of food that is made of apple. ",
    "\"Banana ((n.) A long and yellow fruit.\"",
    "California (n.) A state of USA.",
    "\" Drag  ( v. ))  To pull forcefully. \"",
    "Delta-Epsilon (n.) A mathematical language that defines derivatives.",
    "\"Fine adj. Good.\"",
    "Gay (adj.) Happy and joyful.",
    "\"Hike     (   v.   )     Travel on foot.\"",
    " University of Toronto  ( n. )  A university in Ontario. ",
    "\"Iceberg (n.) A big piece of ice that floats on the sea.\"",
    " \"a ( ( b ) ) c\"",
    "a ( ( b ) ( ) ))  c",
    "a ( ( ( b ) ) c",
    "\" a b c () \"",
    "a b () b c c () \""};

bool expect_success[N] = {
    true, false, true, true, true,  false, true, true,
```

```
    true, true,  true, true, false, true,  true,
};

const char *expect_entries[N][3] = {
    {"Apple pie", "n.", "A kind of food that is made of apple."},
    {},
    {"California", "n.", "A state of USA."},
    {"Drag", "v.", ")  To pull forcefully."},
    {"Delta-Epsilon", "n.",
     "A mathematical language that defines derivatives."},
    {},
    {"Gay", "adj.", "Happy and joyful."},
    {"Hike", "v.", "Travel on foot."},
    {"University of Toronto", "n.", "A university in Ontario."},
    {"Iceberg", "n.", "A big piece of ice that floats on the sea."},
    {"\"a", "( b )", "c\""},
    {"a", "( b ) ( )", ")  c"},
    {},
    {"a b c", "", ""},
    {"a b", "", "b c c () \""}};

void test_new_dict_entry() {
    DictEntry *entry;
    for (int i = 0; i < N; ++i) {
        String *line = new_string(lines[i], -1);
        DictEntry *entry = new_dict_entry(line);
        delete_string(line);
        CHECK_BOOL(expect_success[i], (bool)entry);
        if (expect_success[i] && entry) {
            CHECK_STRING(expect_entries[i][0], entry->headword);
            CHECK_STRING(expect_entries[i][1], entry->word_class);
            CHECK_STRING(expect_entries[i][2], entry->definition);
        }
        if (entry) {
            delete_dict_entry(entry);
        }
    }
}

int main() {
    test_new_dict_entry();
    DISPLAY_TEST_RESULT();
}
```

A.21 src/test/trie.c

```
#include "test/global.h"

int test_count = 0, pass_count = 0;

Trie *dictionary;

void create_dictionary() {
    dictionary = new_trie();
    char file_name[] = "../Dictionary-in-csv/*.csv";
    int index = strchr(file_name, '*') - file_name;
    FILE *stream;
    String *line;
    DictEntry *entry;
    for (char c = 'A'; c <= 'Z'; ++c) {
        file_name[index] = c;
        stream = fopen(file_name, "r");
        if (!stream) {
            WARN("Cannot open file: %s\n", file_name);
            continue;
        }
        line = get_line(stream);
        while (line) {
            if (line->size > 0) {
                entry = new_dict_entry(line);
                if (!entry) {
                    WARN("Failed to parse the following line in %s:\n%s\n",
                            file_name, line->text);
                } else {
                    trie_insert(dictionary, entry);
                }
            }
            delete_string(line);
            line = get_line(stream);
        }
        fclose(stream);
    }
}

void save_dictionary(const char *file_name) {
    Array *entries = traverse_trie(dictionary);
    CHECK_INT(dictionary->size, entries->size);
    FILE *stream = fopen(file_name, "wb");
    for (int i = 0; i < entries->size; ++i) {
```

```
44          write_dict_entry((DictEntry *)(entries->data[i]), stream);
45      }
46      fclose(stream);
47      delete_array(entries);
48  }
49
50  void test_search() {
51      Array *entries = traverse_trie(dictionary), *results;
52      DictEntry *entry;
53      for (int i = 0; i < entries->size; ++i) {
54          entry = entries->data[i];
55          results = trie_search(dictionary, entry->headword, false);
56          CHECK_BOOL(true, results->size > 0);
57          delete_array(results);
58      }
59      delete_array(entries);
60  }
61
62  void test_predecessor() {
63      Array *entries = traverse_trie(dictionary);
64      String *s1, *s2;
65      s1 = to_lower(((DictEntry *)(entries->data[entries->size - 1]))->
    headword);
66      int entry_count = 0;
67      Array *results;
68      while (true) {
69          results = trie_search(dictionary, s1, false);
70          entry_count += results->size;
71          delete_array(results);
72          s2 = trie_predecessor(dictionary, s1);
73          if (!s2) {
74              delete_string(s1);
75              break;
76          }
77          CHECK_BOOL(true, strcmp(s1->text, s2->text) > 0);
78          delete_string(s1);
79          s1 = s2;
80      }
81      delete_array(entries);
82      CHECK_INT(dictionary->size, entry_count);
83  }
84
85  void test_successor() {
86      Array *entries = traverse_trie(dictionary);
87      String *s1, *s2;
```

```
88      s1 = to_lower(((DictEntry *)(entries->data[0]))->headword);
89      int entry_count = 0;
90      Array *results;
91      while (true) {
92          results = trie_search(dictionary, s1, false);
93          entry_count += results->size;
94          delete_array(results);
95          s2 = trie_successor(dictionary, s1);
96          if (!s2) {
97              delete_string(s1);
98              break;
99          }
100         CHECK_BOOL(true, strcmp(s1->text, s2->text) < 0);
101         delete_string(s1);
102         s1 = s2;
103     }
104     delete_array(entries);
105     CHECK_INT(dictionary->size, entry_count);
106 }
107
108 void test_remove() {
109     Array *entries = traverse_trie(dictionary);
110     Array *headwords = new_array(delete_string);
111     String *headword;
112     for (int i = 0; i < entries->size; ++i) {
113         headword = ((DictEntry *)(entries->data[i]))->headword;
114         array_append(headwords, new_string(headword->text, headword->size))
    ;
115     }
116     delete_array(entries);
117     for (int i = 0; i < headwords->size; ++i) {
118         trie_remove(dictionary, (String *)(headwords->data[i]), true);
119     }
120     CHECK_INT(0, dictionary->size);
121     delete_array(headwords);
122 }
123
124 int main() {
125     create_dictionary();
126     save_dictionary("rewrite_dictionary.txt");
127     test_search();
128     test_predecessor();
129     test_successor();
130     test_remove();
131     delete_trie(dictionary);
```

```
    DISPLAY_TEST_RESULT();
}
```

A.22 src/test/matcher.c

```
#include "test/global.h"

int test_count = 0, pass_count = 0;

Trie *dictionary;

// Copy and paste from test/trie.c.
void create_dictionary() {
    dictionary = new_trie();
    char file_name[] = "../Dictionary-in-csv/*.csv";
    int index = strchr(file_name, '*') - file_name;
    FILE *stream;
    String *line;
    DictEntry *entry;
    for (char c = 'A'; c <= 'Z'; ++c) {
        file_name[index] = c;
        stream = fopen(file_name, "r");
        if (!stream) {
            WARN("Cannot open file: %s\n", file_name);
            continue;
        }
        line = get_line(stream);
        while (line) {
            if (line->size > 0) {
                entry = new_dict_entry(line);
                if (!entry) {
                    WARN("Failed to parse the following line in %s:\n%s\n",
                         file_name, line->text);
                } else {
                    trie_insert(dictionary, entry);
                }
            }
            delete_string(line);
            line = get_line(stream);
        }
        fclose(stream);
    }
}

void test_closest_match() {
```

```
    Array *entries = traverse_trie(dictionary);
    DictEntry *entry;
    String *lowered, *result;
    for (int i = 0; i < entries->size; ++i) {
        entry = entries->data[i];
        lowered = to_lower(entry->headword);
        result = trie_closest_match(dictionary, entry->headword, -1);
        CHECK_BOOL(true, result != NULL);
        if (result) {
            CHECK_STRING(lowered->text, result);
            delete_string(result);
        }
        delete_string(lowered);
    }
    delete_array(entries);
}

int main() {
    create_dictionary();
    test_closest_match();
    delete_trie(dictionary);
    DISPLAY_TEST_RESULT();
}
```

A.23 src/main/api.h

```
#ifndef MAIN_API_H_
#define MAIN_API_H_

#include "core/array.h"
#include "core/dict_entry.h"
#include "core/matcher.h"
#include "core/string.h"
#include "core/trie.h"
#include "core/trie_node.h"

#ifndef ESP_RC_PATH
#define ESP_RC_PATH "../.esp_rc"
#endif // ESP_RC_PATH

typedef enum EspMode {
    ESP_MODE_INTERACTIVE,
    ESP_MODE_BACKGROUND,
    ESP_MODE_COMMAND_LINE
} EspMode;
```

```

void esp_initialize(EspMode mode);
void esp_cleanup(EspMode mode);

bool esp_parse_arguments(Array *arguments, EspMode mode);

void esp_on_load(Array *arguments, EspMode mode);
void esp_on_search(Array *arguments, EspMode mode);
void esp_on_insert(Array *arguments, EspMode mode);
void esp_on_remove(Array *arguments, EspMode mode);
void esp_on_neighbour(Array *arguments, EspMode mode);
void esp_on_prefix(Array *arguments, EspMode mode);
void esp_on_match(Array *arguments, EspMode mode);
void esp_on_size(Array *arguments, EspMode mode);
void esp_on_save(Array *arguments, EspMode mode);
bool esp_on_exit(Array *arguments, EspMode mode);

#endif // MAIN_API_H_
```

A.24 src/main/api.c

```
#include "main/api.h"
#include "main/utility.h"

Trie *dictionary;

void esp_initialize(EspMode mode) {
    if (mode == ESP_MODE_INTERACTIVE) {
        printf("
   *============================================\n"
               "               ||  EngSci Press Dictionary by Yunhao Qian
   ||\n"
               "
   *============================================\n"
               "\nStarting...\n\n");
    }
    dictionary = new_trie();
    FILE *stream = fopen(ESP_RC_PATH, "r");
    if (stream) {
        String *line = get_line(stream);
        Array *arguments;
        while (line) {
            arguments = split_string(line);
            esp_parse_arguments(arguments, ESP_MODE_BACKGROUND);
            delete_array(arguments);
```

```
            delete_string(line);
            line = get_line(stream);
        }
        fclose(stream);
    }
}

void esp_cleanup(EspMode mode) {
    if (mode == ESP_MODE_INTERACTIVE) {
        printf("\nExiting...\n");
    }
    delete_trie(dictionary);
}

bool esp_parse_arguments(Array *arguments, EspMode mode) {
    if (arguments->size <= 0) {
        return true;
    }
    String *leading = to_lower(arguments->data[0]);
    array_remove(arguments, 0);
    bool returned = true;
    if (!strcmp(leading->text, "load")) {
        esp_on_load(arguments, mode);
    } else if (!strcmp(leading->text, "search")) {
        esp_on_search(arguments, mode);
    } else if (!strcmp(leading->text, "insert")) {
        esp_on_insert(arguments, mode);
    } else if (!strcmp(leading->text, "remove")) {
        esp_on_remove(arguments, mode);
    } else if (!strcmp(leading->text, "neighbour")) {
        esp_on_neighbour(arguments, mode);
    } else if (!strcmp(leading->text, "prefix")) {
        esp_on_prefix(arguments, mode);
    } else if (!strcmp(leading->text, "match")) {
        esp_on_match(arguments, mode);
    } else if (!strcmp(leading->text, "size")) {
        esp_on_size(arguments, mode);
    } else if (!strcmp(leading->text, "save")) {
        esp_on_save(arguments, mode);
    } else if (!strcmp(leading->text, "exit")) {
        returned = esp_on_exit(arguments, mode);
    } else {
        WARN("Unknown leading argument: %s\n", leading->text);
    }
    delete_string(leading);
```

```
    return returned;
}

void esp_on_load(Array *arguments, EspMode mode) {
    if (mode == ESP_MODE_COMMAND_LINE) {
        WARN_NOT_SUPPORTED("load", "command-line");
        return;
    }
    if (arguments->size < 1) {
        WARN_MISSING("file name");
        return;
    }
    if (arguments->size > 1) {
        WARN_REDUNDANT(arguments, 1);
    }
    const char *file_name = ((String *)arguments->data[0])->text;
    FILE *stream = fopen(file_name, "r");
    if (!stream) {
        WARN("Cannot open file: %s\nDo nothing.\n", file_name);
        return;
    }
    String *line = get_line(stream);
    DictEntry *entry;
    int count = 0;
    while (line) {
        if (line->size > 0) {
            entry = new_dict_entry(line);
            if (!entry) {
                WARN("Failed to parse the following line in %s:\n%s\n",
                     file_name, line->text);
            } else {
                trie_insert(dictionary, entry);
                ++count;
            }
        }
        delete_string(line);
        line = get_line(stream);
    }
    fclose(stream);
    if (mode == ESP_MODE_INTERACTIVE) {
        printf("%d entries loaded from %s\n", count, file_name);
    }
}

void esp_on_search(Array *arguments, EspMode mode) {
```

```
    if (mode == ESP_MODE_BACKGROUND) {
        WARN_NOT_SUPPORTED("search", "background");
        return;
    }
    if (arguments->size <= 0) {
        WARN_MISSING("headword");
        return;
    }
    String *word = join_strings(arguments, ' ');
    bool case_sensitive = false;
    for (int i = 0; i < word->size; ++i) {
        if (isupper(word->text[i])) {
            case_sensitive = true;
            break;
        }
    }
    Array *results = trie_search(dictionary, word, case_sensitive);
    if (results->size <= 0) {
        WARN("Find no entry named: %s\n", word->text);
        word_hint(word, dictionary, case_sensitive);
    } else {
        for (int i = 0; i < results->size; ++i) {
            putchar('\n');
            display_dict_entry(results->data[i]);
        }
        putchar('\n');
    }
    delete_array(results);
    delete_string(word);
}

void esp_on_insert(Array *arguments, EspMode mode) {
    if (mode == ESP_MODE_BACKGROUND) {
        WARN_NOT_SUPPORTED("insert", "background");
        return;
    }
    if (mode == ESP_MODE_COMMAND_LINE) {
        WARN_NOT_SUPPORTED("insert", "command-line");
        return;
    }
    if (arguments->size <= 0) {
        WARN_MISSING("headword");
        return;
    }
    String *headword = join_strings(arguments, ' ');
```

```
157     DictEntry *entry = input_dict_entry(headword);
158     if (entry) {
159         trie_insert(dictionary, entry);
160     }
161     delete_string(headword);
162 }
163 
164 void esp_on_remove(Array *arguments, EspMode mode) {
165     if (mode == ESP_MODE_COMMAND_LINE) {
166         WARN_NOT_SUPPORTED("remove", "command-line");
167         return;
168     }
169     if (arguments->size <= 0) {
170         WARN_MISSING("headword");
171         return;
172     }
173     String *headword = join_strings(arguments, ' ');
174     bool case_sensitive = false;
175     for (int i = 0; i < headword->size; ++i) {
176         if (isupper(headword->text[i])) {
177             case_sensitive = true;
178             break;
179         }
180     }
181     Array *results = trie_search(dictionary, headword, case_sensitive);
182     int remove_count = results->size;
183     bool shall_remove = true;
184     if (remove_count <= 0) {
185         WARN("Find no entry named: %s\n", headword->text);
186         word_hint(headword, dictionary, case_sensitive);
187         shall_remove = false;
188     } else if (mode == ESP_MODE_INTERACTIVE) {
189         printf("The following entries will be removed:\n");
190         for (int i = 0; i < remove_count; ++i) {
191             putchar('\n');
192             display_dict_entry(results->data[i]);
193         }
194         shall_remove = confirm(true, "\nWant to continue?");
195     }
196     delete_array(results);
197     if (shall_remove) {
198         trie_remove(dictionary, headword, case_sensitive);
199         if (mode == ESP_MODE_INTERACTIVE) {
200             printf("%d entries removed.\n", remove_count);
201         }
```

```
202     } else if (mode == ESP_MODE_INTERACTIVE) {
203         printf("Do nothing.\n");
204     }
205 }
206 
207 void esp_on_neighbour(Array *arguments, EspMode mode) {
208     if (mode == ESP_MODE_BACKGROUND) {
209         WARN_NOT_SUPPORTED("neighbour", "background");
210         return;
211     }
212     if (arguments->size <= 0) {
213         WARN_MISSING("headword");
214         return;
215     }
216     int radius = 10;
217     if (string_start_with(arguments->data[0], "--")) {
218         if (arguments->size <= 1) {
219             WARN_MISSING("headword");
220             return;
221         }
222         int number = parse_unsigned_int_flag(arguments->data[0]);
223         if (number < 0) {
224             WARN("Invalid flag: %s\n", ((String *)arguments->data[0])->text
    );
225         } else {
226             radius = number;
227         }
228         array_remove(arguments, 0);
229     }
230     String *word = join_strings(arguments, ' ');
231     delete_string(word);
232 }
233 
234 void esp_on_prefix(Array *arguments, EspMode mode) {
235     if (mode == ESP_MODE_BACKGROUND) {
236         WARN_NOT_SUPPORTED("prefix", "background");
237         return;
238     }
239     if (arguments->size <= 0) {
240         WARN_MISSING("prefix string");
241         return;
242     }
243     int max_count = 10;
244     if (string_start_with(arguments->data[0], "--")) {
245         if (arguments->size <= 1) {
```

```
246             WARN_MISSING("prefix string");
247             return;
248         }
249         int number = parse_unsigned_int_flag(arguments->data[0]);
250         if (number < 0) {
251             WARN("Invalid flag: %s\n", ((String *)arguments->data[0])->text
    );
252         } else {
253             max_count = number;
254         }
255         array_remove(arguments, 0);
256     }
257     String *prefix = join_strings(arguments, ' ');
258     delete_string(prefix);
259 }
260 
261 void esp_on_match(Array *arguments, EspMode mode) {
262     if (mode == ESP_MODE_BACKGROUND) {
263         WARN_NOT_SUPPORTED("match", "background");
264         return;
265     }
266     if (arguments->size <= 0) {
267         WARN_MISSING("headword");
268         return;
269     }
270     int tolerance = -1;
271     if (string_start_with(arguments->data[0], "--")) {
272         if (arguments->size <= 1) {
273             WARN_MISSING("headword");
274             return;
275         }
276         int number = parse_unsigned_int_flag(arguments->data[0]);
277         if (number < 0) {
278             WARN("Invalid flag: %s\n", ((String *)arguments->data[0])->text
    );
279         } else {
280             tolerance = number;
281         }
282         array_remove(arguments, 0);
283     }
284     String *pattern = join_strings(arguments, ' ');
285     String *matched = trie_closest_match(dictionary, pattern, tolerance);
286     if (!matched) {
287         WARN("Find no entry similar to: %s\n", pattern->text);
288     } else {
```

```
289         printf("%s\n", matched->text);
290         delete_string(matched);
291     }
292     delete_string(pattern);
293 }
294 
295 void esp_on_size(Array *arguments, EspMode mode) {
296     if (mode == ESP_MODE_BACKGROUND) {
297         WARN_NOT_SUPPORTED("size", "background");
298         return;
299     }
300     if (arguments->size > 0) {
301         WARN_REDUNDANT(arguments, 0);
302     }
303     printf("Dictionary size: %d\n", dictionary->size);
304 }
305 
306 void esp_on_save(Array *arguments, EspMode mode) {
307     if (arguments->size < 1) {
308         WARN_MISSING("file name");
309         return;
310     }
311     if (arguments->size > 1) {
312         WARN_REDUNDANT(arguments, 1);
313     }
314     if (mode == ESP_MODE_INTERACTIVE && dictionary->size <= 0 &&
315         !confirm(false, "The dictionary is empty. Continue?")) {
316         printf("Do nothing.\n");
317         return;
318     }
319     const char *file_name = ((String *)arguments->data[0])->text;
320     FILE *stream = fopen(file_name, "wb");
321     if (!stream) {
322         WARN("Cannot open file: %s\nDo nothing.\n", file_name);
323         return;
324     }
325     Array *entries = traverse_trie(dictionary);
326     for (int i = 0; i < entries->size; ++i) {
327         write_dict_entry(entries->data[i], stream);
328     }
329     delete_array(entries);
330     fclose(stream);
331     if (mode == ESP_MODE_INTERACTIVE) {
332         printf("%d entries saved to %s.\n", dictionary->size, file_name);
333     }
```

```
}

bool esp_on_exit(Array *arguments, EspMode mode) {
    if (mode == ESP_MODE_BACKGROUND) {
        WARN_NOT_SUPPORTED("exit", "background");
        return true;
    }
    if (mode == ESP_MODE_COMMAND_LINE) {
        WARN_NOT_SUPPORTED("exit", "command-line");
        return true;
    }
    if (arguments->size > 0) {
        WARN_REDUNDANT(arguments, 0);
    }
    return !confirm(true, "Are you sure you want to exit?");
}
```

A.25 src/main/utility.h

```
#ifndef MAIN_UTILITY_H_
#define MAIN_UTILITY_H_

#include "main/api.h"

#define WARN_NOT_SUPPORTED(argument, mode)
        \
    WARN("Does not support \"%s\" in %s mode.\n", argument, mode);

#define WARN_MISSING(expected)
        \
    WARN("Missing argument: %s expected.\n", expected)

#define WARN_REDUNDANT(arguments, start_index)
        \
    WARN("Redundant arguments: ignore arguments since \"%s\".\n",
        \
          ((String *)(arguments)->data[start_index])->text)

int parse_unsigned_int_flag(const String *string);

void word_hint(const String *string, const Trie *dictionary,
               bool case_sensitive);

#endif // MAIN_UTILITY_H_
```

A.26 src/main/utility.c

```c
#include "main/utility.h"

int parse_unsigned_int_flag(const String *string) {
    assert(string_start_with(string, "--") &&
           "parse_unsigned_int_flag: not a flag");
    if (string->size <= 2) {
        return -1;
    }
    String *flag = substring(string, 2, string->size);
    for (int i = 0; i < flag->size; ++i) {
        if (!isdigit(flag->text[i])) {
            delete_string(flag);
            return -1;
        }
    }
    int number = atoi(flag->text);
    delete_string(flag);
    return number;
}

void word_hint(const String *string, const Trie *dictionary,
               bool case_sensitive) {
    String *matched = trie_closest_match(dictionary, string, -1);
    if (!matched) {
        return;
    }
    if (case_sensitive) {
        String *lowered = to_lower(string);
        if (!strcmp(matched->text, lowered->text)) {
            printf("Tip: use lower-case word for case-insensitive "
                   "search/remove.\n");
            delete_string(lowered);
            return;
        }
        delete_string(lowered);
    }
    printf("Did you mean: %s\n", matched->text);
    delete_string(matched);
}
```

A.27 src/main/main.c

```c
#include "main/api.h"
```

```

int main(int argc, const char **argv) {
    Array *arguments;
    if (argc > 1) {
        arguments = new_array(delete_string);
        for (int i = 1; i < argc; ++i) {
            array_append(arguments, new_string(argv[i], -1));
        }
        esp_initialize(ESP_MODE_COMMAND_LINE);
        esp_parse_arguments(arguments, ESP_MODE_COMMAND_LINE);
        delete_array(arguments);
        esp_cleanup(ESP_MODE_COMMAND_LINE);
        return 0;
    }
    esp_initialize(ESP_MODE_INTERACTIVE);
    String *line;
    bool shall_continue;
    do {
        printf(">>> ");
        line = get_line(stdin);
        if (!line) {
            break;
        }
        arguments = split_string(line);
        shall_continue = esp_parse_arguments(arguments,
    ESP_MODE_INTERACTIVE);
        delete_array(arguments);
        delete_string(line);
    } while (shall_continue);
    esp_cleanup(ESP_MODE_INTERACTIVE);
}
```

A.28 src/writer/grammar.py

```
from random import choices, random
from re import compile, match


float_pattern = compile(r'([0-9]*\.)?[0-9]+)\s*\:\s+')
token_pattern = compile(
    r'(([^\s\"\(\)]|(\(([^\(\)]|(\"([^\"]|\\\")+\"))+\))|(\"([^\"]|\\\")*\"))
    +)\s*')


class Token:
```

```
11
12      __slots__ = 'identifier', 'terminal', 'optional', 'probability'
13
14      def __init__(self, string):
15          self.terminal = False
16          self.optional = False
17          self.probability = 1
18          if string.startswith('(') and string.endswith(')'):
19              string = string[1:-1]
20              matched = float_pattern.match(string)
21              factor = 1
22              if matched:
23                  string = string[matched.end():]
24                  factor = float(matched.group(1))
25              self.__init__(string)
26              self.optional = True
27              self.probability *= factor
28          elif string.startswith('"') and string.endswith('"'):
29              self.identifier = string[1:-1].replace('\\"', '"')
30              self.terminal = True
31          else:
32              self.identifier = string
33
34      def __eq__(self, other):
35          if not isinstance(other, Token):
36              return False
37          return self.identifier == other.identifier and \
38              self.terminal == other.terminal
39
40      def __hash__(self):
41          return hash((self.identifier, self.terminal))
42
43      def __str__(self):
44          string = self.identifier.replace('"', '\\"')
45          if self.terminal:
46              string = '"{}"'.format(string)
47          if self.optional:
48              if self.probability == 1:
49                  string = '({})'.format(string)
50              else:
51                  string = '({}: {})'.format(self.probability, string)
52          return string
53
54
55  class Rule:
```

```
56
57      __slots__ = 'lhs', 'rhs', 'weight'
58
59      def __init__(self, string):
60          matched = float_pattern.match(string)
61          if matched:
62              string = string[matched.end():]
63              self.weight = float(matched.group(1))
64          else:
65              self.weight = 1
66          matched = token_pattern.match(string)
67          string = string[matched.end():]
68          self.lhs = Token(matched.group(1))
69          string = string[match(r'->\s*', string).end():]
70          self.rhs = []
71          while True:
72              matched = token_pattern.match(string)
73              if not matched:
74                  break
75              string = string[matched.end():]
76              self.rhs.append(Token(matched.group(1)))
77
78      def __eq__(self, other):
79          if not isinstance(other, Rule):
80              return False
81          return self.lhs, tuple(self.rhs), self.weight == \
82              other.lhs, tuple(other.rhs), other.weight
83
84      def __hash__(self):
85          return hash((self.lhs, tuple(self.rhs), self.weight))
86
87      def __str__(self):
88          elements = []
89          if self.weight != 1:
90              elements.append(str(self.weight) + ':')
91          elements += [str(self.lhs), '->']
92          for element in self.rhs:
93              elements.append(str(element))
94          return ' '.join(elements)
95
96
97  class CFG:
98
99      __slots__ = 'rules', 'convergence'
100
```

```
    def __init__(self, string=None):
        self.rules = {}
        self.convergence = 1
        if string:
            self.load_lines(string)

    def __str__(self):
        elements = []
        if self.convergence != 1:
            elements.append('convergence = {}'.format(self.convergence))
        for rule_list in self.rules.values():
            for rule in rule_list:
                elements.append(str(rule))
        return '\n'.join(elements)

    def load_line(self, line):
        line = line.strip()
        if line == '' or line.startswith('//'):
            return
        matched = match(r'convergence\s*=\s*(([0-9]*\.)?[0-9]+)\s*', line)
        if matched:
            self.convergence = float(matched.group(1))
            return
        rule = Rule(line)
        if rule.lhs in self.rules:
            self.rules[rule.lhs].append(rule)
        else:
            self.rules[rule.lhs] = [rule]

    def load_lines(self, string):
        for line in string.split('\n'):
            try:
                self.load_line(line)
            except:
                print('Failed to parse line: {}'.format(line))

    def generate(self, start=Token('S'), max_length=-1):
        weight_dict = {}
        for rule_list in self.rules.values():
            for rule in rule_list:
                weight_dict[rule] = rule.weight
        stack = [start]
        terminals = []
        while len(stack) > 0:
            token = stack.pop()
```

```
            if token.optional and random() > token.probability:
                continue
            if token.terminal:
                terminals.append(token.identifier)
                continue
            try:
                rule_list = self.rules[token]
            except:
                raise Exception('Failed to find rule for: {}'.format(token)
    )
            weights = [weight_dict[rule] for rule in rule_list]
            rule = choices(rule_list, weights, k=1)[0]
            weight_dict[rule] *= self.convergence
            stack += rule.rhs[::-1]
            if max_length > 0 and len(terminals) > max_length:
                raise Exception('Exceed max length: {}'.format(max_length))
        return terminals


demo_grammar = '''
convergence = 0.3

0.9: S -> Clause "."
0.1: S -> Clause "while" S

Clause -> NP VP

0.9: NP -> Det (0.6: Adj) N
0.1: NP -> NP "and" NP

VP -> V NP
VP -> V

Det -> "a"
Det -> "the"

Adj -> "smart"
Adj -> "tired"
Adj -> "brown"

N -> "student"
N -> "laptop"
N -> "car"

V -> "drives"
```

```
V -> "walks"
V -> "leaves"
'''


if __name__ == '__main__':
    cfg = CFG(demo_grammar)
    print(cfg)
    while True:
        input('-------------------------------------------------------------
    ')
        try:
            tokens = cfg.generate(max_length=30)
        except Exception as exception:
            print(exception)
        else:
            tokens[0] = tokens[0].capitalize()
            print(' '.join(tokens[:-1]) + tokens[-1])
```

A.29 src/writer/espg_base.txt

```
// ===== Sentence =====
S -> NP-Sg VP-Sg "."
S -> NP-Pl VP-Pl "."
0.4: S -> VP-Pl "!"
0.2: S -> Aux-Sg NP-Sg VP-Pl "?"
0.2: S -> Aux-Pl NP-Pl VP-Pl "?"
0.1: S -> Wh-NP-Sg Aux-Sg NP-Sg VP-Pl "?"
0.1: S -> Wh-NP-Pl Aux-Pl NP-Pl VP-Pl "?"

// ===== Noun Phrase =====
0.2: NP-Sg -> Pronoun-Sg
// NP-Sg -> Proper-Noun-Sg
NP-Sg -> Det-Sg (0.5: AP) Nominal-Sg
0.2: NP-Pl -> Pronoun-Pl
// NP-Pl -> Proper-Noun-Pl
NP-Pl -> Det-Pl (0.5: AP) Nominal-Pl

// ===== Nominal =====
Nominal-Sg -> Noun-Sg
0.3: Nominal-Sg -> Nominal-Sg PP
0.3: Nominal-Sg -> Nominal-Sg Gerund-VP
0.3: Nominal-Sg -> Nominal-Sg Rel-Clause-Sg
Nominal-Pl -> Noun-Pl
0.3: Nominal-Pl -> Nominal-Pl PP
```

```
0.3: Nominal-Pl -> Nominal-Pl Gerund-VP
0.3: Nominal-Pl -> Nominal-Pl Rel-Clause-Pl

// ===== Gerundive Verb =====
Gerund-VP -> Gerund-V
Gerund-VP -> Gerund-V NP-Sg
Gerund-VP -> Gerund-V NP-Pl
Gerund-VP -> Gerund-V PP
Gerund-VP -> Gerund-V NP-Sg PP
Gerund-VP -> Gerund-V NP-Pl PP

// ===== Relative Clause =====
Rel-Clause-Sg -> Rel-Pronoun VP-Sg
Rel-Clause-Pl -> Rel-Pronoun VP-Pl

// ===== Verb Phrase =====
VP-Sg -> Verb-I-Sg
VP-Sg -> Verb-T-Sg NP-Sg
VP-Sg -> Verb-T-Sg NP-Pl
VP-Sg -> Verb-T-Sg NP-Sg PP
VP-Sg -> Verb-T-Sg NP-Pl PP
VP-Sg -> Verb-I-Sg PP
VP-Pl -> Verb-I-Pl
VP-Pl -> Verb-T-Pl NP-Sg
VP-Pl -> Verb-T-Pl NP-Pl
VP-Pl -> Verb-T-Pl NP-Sg PP
VP-Pl -> Verb-T-Pl NP-Pl PP
VP-Pl -> Verb-I-Pl PP

// ===== Adjective Phrase =====
AP -> Adj
0.2: AP -> Adv AP

// ===== Prepositional Phrase =====
PP -> Preposition NP-Sg
PP -> Preposition NP-Pl

// ===== Determiner =====
5: Det-Sg -> "the"
5: Det-Pl -> "the"
5: Det-Sg -> "a"
4: Deg-Sg -> "this"
4: Deg-Sg -> "that"
4: Det-Pl -> "these"
4: Det-Pl -> "those"
```

```
Det-Sg -> "my"
Det-Pl -> "my"
Det-Sg -> "your"
Det-Pl -> "your"
Det-Sg -> "his"
Det-Pl -> "his"
Det-Sg -> "her"
Deg-Pl -> "her"
Det-Sg -> "its"
Det-Pl -> "its"
Det-Sg -> "our"
Det-Pl -> "our"
Det-Sg -> "their"
Det-Pl -> "their"
2: Det-Pl -> "a" "few"
2: Det-Pl -> "many"
2: Det-Pl -> "a" "lot" "of"
3: Det-Pl -> "some"
Det-Sg -> "any"
Det-Sg -> "one"
Det-Pl -> "all"
Det-Sg -> "each"
Det-Sg -> "every"
Det-Sg -> "another"
Det-Sg -> NP-Sg "'s"
Det-Pl -> NP-Sg "'s"

// ===== Auxiliary Verb =====
Aux-Sg -> "has"
Aux-Pl -> "have"
Aux-Sg -> "had"
Aux-Pl -> "had"
Aux-Sg -> "did"
Aux-Pl -> "did"
Aux-Sg -> "will"
Aux-Pl -> "will"
Aux-Sg -> "should"
Aux-Pl -> "should"
Aux-Sg -> "would"
Aux-Pl -> "would"
Aux-Sg -> "may"
Aux-Pl -> "may"
Aux-Sg -> "might"
Aux-Pl -> "might"
Aux-Sg -> "must"
```

```
Aux-Pl -> "must"
Aux-Sg -> "can"
Aux-Pl -> "can"
Aux-Sg -> "could"
Aux-Pl -> "could"
Aux-Sg -> "does"
Aux-Pl -> "do"
Aux-Sg -> "need"
Aux-Pl -> "need"

// ===== Wh- Noun Phrase ====
Wh-NP-Sg -> "when"
Wh-NP-Pl -> "when"
Wh-NP-Sg -> "who"
Wh-NP-Pl -> "who"
Wh-NP-Sg -> "where"
Wh-NP-Pl -> "where"
Wh-NP-Sg -> "what"
Wh-NP-Pl -> "what"
Wh-NP-Sg -> "what" Noun-Sg
Wh-NP-Pl -> "what" Noun-Pl
Wh-NP-Sg -> "whose" Noun-Sg
Wh-NP-Pl -> "whose" Noun-Pl
Wh-NP-Sg -> "which" Noun-Sg
Wh-NP-Pl -> "which" Noun-Pl

// ===== Pronoun =====
4: Pronoun-Pl -> "you"
Pronoun-Sg -> "yours"
Pronoun-Pl -> "yours"
2: Pronoun-Pl -> "yourself"
Pronoun-Sg -> "him"
Pronoun-Sg -> "his"
Pronoun-Pl -> "his"
2: Pronoun-Sg -> "himself"
Pronoun-Sg -> "her"
Pronoun-Sg -> "hers"
Pronoun-Pl -> "hers"
2: Pronoun-Sg -> "herself"
4: Pronoun-Sg -> "it"
Pronoun-Sg -> "its"
Pronoun-Pl -> "its"
Pronoun-Sg -> "itself"
4: Pronoun-Sg -> "ours"
4: Pronoun-Pl -> "ours"
```

```
2: Pronoun-Pl -> "ourself"
Pronoun-Sg -> "theirs"
Pronoun-Pl -> "theirs"
2: Pronoun-Pl -> "themselves"

// ===== Relative Pronoun =====
Rel-Pronoun -> "who"
Rel-Pronoun -> "which"
Rel-Pronoun -> "that"

// ===== In espg_lexicon.txt =====
// Noun-Sg
// Noun-Pl
// Gerund-V
// Verb-I-Sg
// Verb-T-Sg
// Verb-I-Pl
// Verb-T-Pl
// Preposition
// Adj
// Adv
```

A.30 src/writer/lexicon.py

```python
from collections import defaultdict


class DictEntry:

    __slots__ = 'headword', 'word_class'

    def __init__(self, headword, word_class):
        self.headword = headword
        self.word_class = word_class

    @staticmethod
    def from_line(line):
        if line.startswith('"') and line.endswith('"'):
            line = line[1:-1]
        left_index = line.index('(')
        if left_index < 0:
            raise Exception('missing word class', line)
        right_index = -1
        depth = 1
        for i in range(left_index + 1, len(line)):
```

```
            if line[i] == '(':
                depth += 1
            elif line[i] == ')':
                depth -= 1
                if depth == 0:
                    right_index = i
                    break
        if right_index < 0:
            raise Exception('mismatched brackets', line)
        headword = line[:left_index].strip()
        if headword == '':
            raise Exception('empty headword', line)
        word_class = line[left_index + 1:right_index].strip()
        return DictEntry(headword, word_class)


pos_tag_to_word_classes = {
    'Proper-Noun-Sg': set(),
    'Proper-Noun-Pl': set(),
    'Noun-Sg': {
        'n. & v',
        'n.& v.',
        'n & v.',
        'n. & v. t.',
        'n.',
        'n. sing & pl.',
        'a & n.',
        'n. & v. i.',
        'n. /',
        'n. / interj.',
        'n. & v.',
        'sing. or pl.',
        'n.sing & pl.',
        'n',
        'n., a., & v.',
        'n. & a.',
        'sing. & pl.',
        'n .',
        'v. t. & n.',
        'n. sing. & pl.',
        'a., n., & adv.',
        'n. & adv.',
        'n. / v. t. & i.',
        'n.sing. & pl.',
        'n. .',
```

```
        'v.& n.',
        'n. & interj.',
        'adv. & n.',
        'n. Chem.',
        'v. i. & n.',
        'n.',
        'sing.',
        'N.',
        'n./',
        'adv., & n.',
        'a. / n.',
        'v. & n.',
        'a., adv., & n.',
        'n..',
        'n. sing. & pl',
        'interj. & n.',
        'n. sing.',
        'n. & i.',
        'imperative sing.',
        'syntactically sing.'
    },
    'Noun-Pl': {
        'n. pl.',
        'n. sing & pl.',
        'n.pl.',
        'sing. or pl.',
        'n.sing & pl.',
        'sing. & pl.',
        'n. pl',
        'n. sing. & pl.',
        'n.sing. & pl.',
        'n pl.',
        'n., sing. & pl.',
        'n. collect. & pl.',
        'n. sing. & pl',
        'n. pl.',
        'sing. / pl.'
    },
    'Gerund-V': {
        'p. pr. & v. n.',
        'p. pr. &, vb. n.',
        'imp. & p. p. Fenced (/); p. pr. & vb. n.',
        'imp. & p. p. & vb. n.',
        'p, pr. & vb. n.',
        'p. pr. a. & vb. n.',
```

```
        'p. pr. vb. n.',
        'imp. & p. pr. & vb. n.',
        'pr.p. & vb. n.',
        'p. pr. / vb. n.',
        'p]. pr. & vb. n.',
        'p. pr.& vb. n.',
        'p. pr. &vb. n.',
        'p. pr. & vb/ n.',
        'P. pr. & vb. n.',
        'p. pr. & vvb. n.',
        'p. a. & vb. n.',
        'p. pr. &. vb. n.',
        'p. pr. & pr. & vb. n.',
        'vb. n.',
        'p. p. & vb. n.',
        'p pr. & vb. n.',
        'imp. & p. p. Adored (/); p. pr. & vb. n.',
        'p. pr & vb. n.'
    },
    'Verb-I-Sg': {
        '3d sing.pr.',
        'subj. 3d pers. sing.',
        '3d sing.',
        '3d pers. sing. pres.',
        '3d sing. pr.',
        'pres. indic. sing., 1st & 3d pers.',
        'Sing. pres. ind.',
        '3d sing.',
        'pres. sing.'
    },
    'Verb-T-Sg': {
        '3d sing.pr.',
        'subj. 3d pers. sing.',
        '3d sing.',
        '3d pers. sing. pres.',
        '3d sing. pr.',
        'pres. indic. sing., 1st & 3d pers.',
        'Sing. pres. ind.',
        '3d sing.',
        'pres. sing.'
    },
    'Verb-I-Pl': {
        'v. t. / i.',
        'v. i.,',
        'n. & v. i.',
```

```
        'v. i. & i.',
        'v. i.',
        'v.t & i.',
        'v.i',
        'v. t. / v. i.',
        'v.i.',
        'v. t.& i.',
        'n. / v. t. & i.',
        'v. i.',
        'v. t. & v. i.',
        'v. i. & n.',
        'v. i. & auxiliary.',
        'v. t. & i.',
        'v. i. & t.',
        'v. i. / auxiliary'
    },
    'Verb-T-Pl': {
        'v. t. / i.',
        'a. & v. t.',
        'v. t. &',
        'n. & v. t.',
        'v. t..',
        'v. t. v. t.',
        'v.t & i.',
        'v. t. / v. i.',
        'v.t',
        'v. t. & n.',
        'v. t.& i.',
        'n. / v. t. & i.',
        'v. t. & v. i.',
        'v./t.',
        'v. t.',
        'v. t. / auxiliary',
        'v. t.',
        'v. i. & t.',
        'v.t.'
    },
    'Preposition': {
        'prep., adv., & conj.',
        'prep., adv., conj. & n.',
        'adv. & prep.',
        'prep. & conj., but properly a participle',
        'prep., adv. & a.',
        'prep., adv. & conj.',
        'prep. & adv.',
```

```
        'adv., prep., & conj.',
        'prep.',
        'adv. or prep.',
        'prep. & conj.',
        'conj. & prep.'
    },
    'Adj': {
        'adj.',
        'pron. / adj.',
        'a.',
        'p. p. / a.',
        'a. & v. t.',
        'adv. & a.',
        'p. p & a.',
        'p. p. & a.',
        'a. / a. pron.',
        'P. p. & a.',
        'pron. & a.',
        'a & n.',
        'a/',
        'adv. / a.',
        'a. & a. pron.',
        'a & p. p.',
        'p. & a.',
        'prep., adv. & a.',
        'a. .',
        'a. superl.',
        'v. & a.',
        'a. & adv.',
        'n., a., & v.',
        'a. a.',
        'pron., a., conj., & adv.',
        'n. & a.',
        'p. pr. a. & vb. n.',
        'a. & v.',
        'a., n., & adv.',
        'a. Vigorously',
        'a. & n.',
        'a.',
        'a. / adv.',
        'a & adv.',
        'a. Vibrating',
        'a. or pron.',
        'a. / pron.',
        'imp., p. p., & a.',
```

```
        'a',
        'p. p. & a',
        'a. / n.',
        'pron., a., & adv.',
        'a., adv., & n.',
        'a. & p. p.',
        'a. & pron.'
    },
    'Adv': {
        'prep., adv., & conj.',
        'prep., adv., conj. & n.',
        'adv. & a.',
        'adv. In combination or cooperation',
        'adv. / interj.',
        'interrog. adv.',
        'adv. & prep.',
        'adv. In a vanishing manner',
        'adv. / a.',
        'prep., adv. & a.',
        'a. & adv.',
        'pron., a., conj., & adv.',
        'prep., adv. & conj.',
        'conj. / adv.',
        'adv.',
        'prep. & adv.',
        'interj., adv., or a.',
        'a., n., & adv.',
        'interj., adv., & n.',
        'n. & adv.',
        'a. / adv.',
        'adv., prep., & conj.',
        'adv. & n.',
        'a & adv.',
        'adv. or prep.',
        'adv., & n.',
        'pron., a., & adv.',
        'a., adv., & n.',
        'interj. & adv.',
        'adv. / conj.',
        'adv. & conj.'
    }
}


def create_lexicon(lines):
```

```
    lexicon = {}
    for tag in pos_tag_to_word_classes:
        lexicon[tag] = set()
    for line in lines:
        line = line.strip()
        if line == '':
            continue
        try:
            entry = DictEntry.from_line(line)
        except:
            print('Failed to parse line: {}'.format(line))
            continue
        for tag in pos_tag_to_word_classes:
            word_class_set = pos_tag_to_word_classes[tag]
            if entry.word_class in word_class_set:
                lexicon[tag].add(entry.headword.lower().replace('"', '\\"')
    )
    return lexicon


def write_lexicon(lexicon, stream):
    for tag in pos_tag_to_word_classes:
        for terminal in lexicon[tag]:
            stream.write('{} -> "{}"\n'.format(tag, terminal))


if __name__ == '__main__':
    try:
        output = open('espg_lexicon.txt', 'w')
    except:
        print('Failed to open file: {}'.format('espg_lexicon.txt'))
        exit()
    for i in range(ord('A'), ord('Z') + 1):
        file_name = ('../../Dictionary-in-csv/{}.csv'.format(chr(i)))
        try:
            with open(file_name, 'r') as stream:
                lines = open(file_name, 'r').readlines()
        except:
            print('Failed to open file: {}'.format(file_name))
        else:
            lexicon = create_lexicon(lines)
            write_lexicon(lexicon, output)
    output.close()
```

A.31 src/writer/writer.py

```
from string import punctuation
from numpy.random import poisson


class Sentence:

    __slots__ = 'tokens'

    def __init__(self, tokens):
        self.tokens = tokens

    def __str__(self):
        elements = []
        for token in self.tokens:
            if len(elements) > 0:
                if token[0] in punctuation or elements[-1][-1] in
   punctuation:
                    elements[-1] += token
                else:
                    elements.append(token)
            else:
                elements.append(token.capitalize())
        return ' '.join(elements)

    def word_count(self):
        count = 0
        for token in self.tokens:
            if token not in punctuation:
                count += 1
        return count


class Paragraph:

    __slots__ = 'sentences', 'indent'

    def __init__(self, indent):
        self.sentences = []
        self.indent = indent

    def __str__(self):
        elements = []
        for sentence in self.sentences:
```

```
            elements.append(str(sentence))
        return ' ' * self.indent + ' '.join(elements)

    def add_sentence(self, tokens):
        self.sentences.append(Sentence(tokens))

    def word_count(self):
        count = 0
        for sentence in self.sentences:
            count += sentence.word_count()
        return count


class Article:

    __slots__ = 'paragraphs', 'title', 'spacing', 'indent'

    def __init__(self, title, spacing, indent):
        self.paragraphs = []
        self.title = title
        self.spacing = spacing
        self.indent = indent

    def __str__(self):
        elements = []
        if self.title:
            elements.append(self.title)
        for paragraph in self.paragraphs:
            if len(paragraph.sentences) == 0:
                continue
            elements.append(str(paragraph))
        return ('\n' * (self.spacing + 1)).join(elements)

    def add_sentence(self, tokens):
        if len(self.paragraphs) == 0:
            self.new_paragraph()
        self.paragraphs[-1].add_sentence(tokens)

    def new_paragraph(self, indent=None):
        if len(self.paragraphs) > 0 and \
                len(self.paragraphs[-1].sentences) == 0:
            self.paragraphs.pop()
        if indent == None:
            indent = self.indent
        self.paragraphs.append(Paragraph(indent))
```

```
88
89      def word_count(self):
90          count = 0
91          for paragraph in self.paragraphs:
92              count += paragraph.word_count()
93          return count
94
95
96  class Writer:
97
98      __slots__ = 'grammar', 'paragraphs_per_article',\
99          'sentences_per_paragraph', 'tokens_per_sentence'
100
101     def __init__(self, grammar, paragraphs_per_article=5,
102                  sentences_per_paragraph=10, tokens_per_sentence=20):
103         self.grammar = grammar
104         self.paragraphs_per_article = paragraphs_per_article
105         self.sentences_per_paragraph = sentences_per_paragraph
106         self.tokens_per_sentence = tokens_per_sentence
107
108     def generate(self, title=None, spacing=1, indent=4):
109         article = Article(title, spacing, indent)
110         for i in range(poisson(self.paragraphs_per_article)):
111             for j in range(poisson(self.sentences_per_paragraph)):
112                 attempt_count = 0
113                 while True:
114                     try:
115                         max_length = poisson(self.tokens_per_sentence)
116                         tokens = self.grammar.generate(max_length=
    max_length)
117                     except:
118                         attempt_count += 1
119                         if attempt_count > 50:
120                             raise Exception('Too many attempts')
121                     else:
122                         article.add_sentence(tokens)
123                         break
124             article.new_paragraph()
125         return article
126
127
128 if __name__ == '__main__':
129     from grammar import CFG, demo_grammar
130     article = Writer(CFG(demo_grammar)).generate('Demo Article')
131     print(article)
```

```
132     print()
133     print('Word count: {}'.format(article.word_count()))
```

A.32 src/writer/main.py

```
1  from grammar import CFG
2  from writer import Writer
3
4
5  def create_espg():
6      espg = CFG()
7      with open('espg_base.txt', 'r') as stream:
8          lines = stream.read()
9      espg.load_lines(lines)
10     with open('espg_lexicon.txt', 'r') as stream:
11         lines = stream.read()
12     espg.load_lines(lines)
13     return espg
14
15
16 if __name__ == '__main__':
17     espg = create_espg()
18     writer = Writer(espg, 3, 10, 15)
19     while True:
20         input('------------------------------------------------------------
       ')
21         article = writer.generate()
22         print(article)
23         print()
24         print('Word count: {}'.format(article.word_count()))
```