自然言語処理 parser 試作に関する報告書

島田 歩 [†]

[†]*ayumu.shimada@gmail.com*

1　始めに

本実験報告書は以下の章立てで構成されている:

第 1 章　始めに
第 2 章　実験の目的
第 3 章　自然言語処理の概要
第 4 章　実験の結果
第 5 章　考察
第 6 章　まとめ

また，巻末には，参考文献リストと付録および手書きの図がある．

特に，第 5 章の考察では，文の解釈を一意に特定できない問題と非文が生成される問題について述べている．

2　実験の目的

自然言語処理において，基本技術の一つになっている構文解析技法について，その原理を学ぶ[1]．また，自然言語処理における構文解析の困難さについても実際に体験する．

3　自然言語処理の概要

自然言語処理 (natural language processing: NLP) とは，人間が書いたり話したりしたりする文章をコンピュータに処理させることである[1]．これは，文章の中の単語の品詞を決定するための形態素解析，単語間のつながりを調べる構文解析，単語や文の意味を解析する意味解析および必要な応答文を生成する等の処理を含む[1]．

また，Chomsky によれば，言語は，形式言語を用いて階層化できる（図 **1**）．なお，正規文法は有限オートマトンで，文脈自由文法 (context-free grammar: CFG) はプッシュダウンオートマトンで，それぞれ認識できる．

3.1　有限オートマトン

有限オートマトンは入力記号列と現在の状態に応じて次の状態へ遷移する．入力記号列を入力し終えたときに受理状態にあれば，その入力記号列は受理可能である．

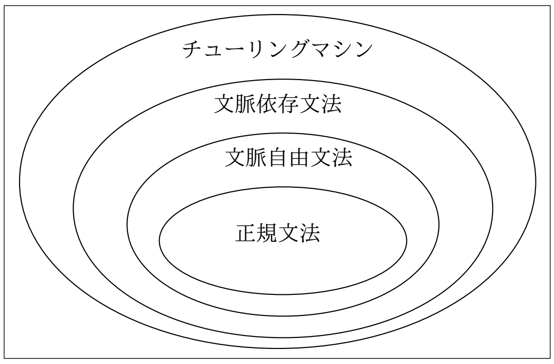


図 1　Chomsky の言語階層[1]

3.2　プッシュダウンオートマトン

有限オートマトンに補助記憶装置としてプッシュダウンスタックを取り付けたものがプッシュダウンオートマトンである．入力記号列を入力し終えたときに受理状態にあり，かつ，スタックが空であれば，その入力記号列は受理可能である．

3.3　あいまい性

文脈自由文法では，構文木が複数得られることがある．この場合，その文法はあいまいであると言う．

4　実験の結果

4.1　ネットワーク文法の実験

4.1.1　手による解析

表 **1** の文法規則と表 **2** の単語辞書を用いて “I saw a girl with a telescope .” を形態素解析した結果を図 **2** に示す．

4.1.2　プログラムによる解析

文法規則と単語辞書に表 **1**, 表 **2** および表 **3** を用い，作成したプログラムで “I saw a girl with a telescope .” と “the child runs quickly to the large house .” の 2 文を形態素解析した結果を示す．

4.2　文脈自由文法の実験

4.2.1　手による解析

表 **4** の文法規則と表 **2** の単語辞書を用いて top-down 法および CYK 法により構文解析した結果を，それぞ

2017 年 12 月 1 日 仮提出 ; 2017 年 12 月 6 日 受理 ;
2017 年 12 月 12 日 更新

```
         I      saw    a     girl   with   a     telescope   .
|        |      |      |     |      |      |     |           |
start    NOUN   VERB   DET   NOUN   PREP   DET   NOUN        end

         the   child   runs   quickly   to     the   large   house   .
|        |     |       |      |         |      |     |       |       |
start    DET   NOUN    VERB   ADV       PREP   DET   ADJ     NOUN    end
```

図 3　プログラムによる形態素解析結果

表 1　ネットワーク文法の文法規則[1)]

start	→	DET (冠詞)
start	→	NOUN (名詞)
DET (冠詞)	→	NOUN (名詞)
DET (冠詞)	→	ADJ (形容詞)
DET (冠詞)	→	NOUN (名詞)
ADJ (形容詞)	→	NOUN (名詞)
NOUN (名詞)	→	PREP (前置詞)
PREP (前置詞)	→	NOUN (名詞)
PREP (前置詞)	→	ADJ (形容詞)
PREP (前置詞)	→	DET (冠詞)
NOUN (名詞)	→	VERB (動詞)
VERB (動詞)	→	ADV (副詞)
VERB (動詞)	→	DET (冠詞)
ADV (副詞)	→	PREP (前置詞)
NOUN (名詞)	→	ADV (副詞)
ADV (副詞)	→	NOUN (名詞)
VERB (動詞)	→	end
NOUN (名詞)	→	end

表 2　“I saw a girl with a telescope.” の単語辞書

I	→	NOUN (名詞)
saw	→	VERB (動詞)
a	→	DET (冠詞)
girl	→	NOUN (名詞)
with	→	PREP (前置詞)
telescope	→	NOUN (名詞)
.	→	end

れ図 **4**, 図 **5** に示す.

4.2.2　プログラムによる解析

表 **5** の文法規則と表 **3** の単語辞書を用い，作成したプログラムで “the child runs quickly to the large house” を CYK 法により構文解析した結果得られた構文木と S 式をそれぞれ図 **6** と図 **7** に示す.

表 3　“the child runs quickly to the large house.” の単語辞書

the	→	DET (冠詞)
child	→	NOUN (名詞)
runs	→	VERB (動詞)
quickly	→	ADV (副詞)
to	→	PREP (前置詞)
large	→	ADJ (形容詞)
house	→	NOUN (名詞)
.	→	end

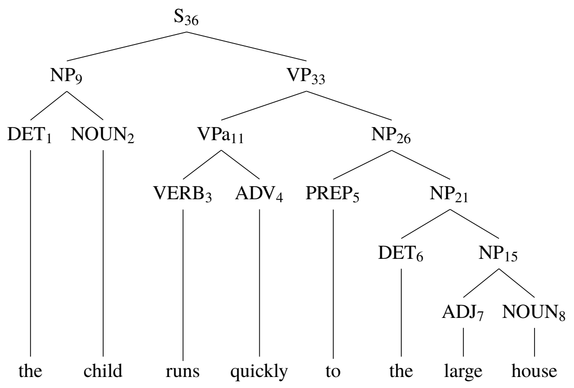


図 6　“the child runs quickly to the large house” の構文木

さらに，表 **4** の文法規則と表 **2** の単語辞書を用い，“I saw a girl with a telescope” を構文解析した結果得られた構文木と S 式をそれぞれ図 **8a** と図 **7** に示す.

5　考察

5.1　“I saw a girl with a telescope.” の 2 つの解釈

“I saw a girl with a telescope.” には大きくわけて，以下の 2 つの意味（解釈）がある:

- 私は望遠鏡を持った少女に会った.（図 **8a**）
- 私は望遠鏡で少女を見た.（図 **8b**）

今回の実験の結果得られた解釈は前者だが，表 **4** の文法規則に VP → VP + PP を追加することにより，後者の解釈となる構文木（図 **8b**）と S 式（図 **9**）が得られる.

表 **4**　“I saw a girl with a telescope” の文法規則

S (文)	→	NP (名詞句)	+	VP (動詞句)
NP (名詞句)	→	NOUN (名詞)		
NP (名詞句)	→	DET (冠詞)	+	NOUN (名詞)
VP (動詞句)	→	VERB (動詞)		
VP (動詞句)	→	VERB (動詞)	+	NP (名詞句)
VP (動詞句)	→	VERB (動詞)	+	SS
SS	→	NP (名詞句)	+	PP (前置詞句)
PP (前置詞句)	→	PREP (前置詞)	+	NP (名詞句)

表 **5**　“the child runs quickly to the large house” の文法規則

S (文)	→	NP (名詞句)	+	VP (動詞句)
NP (名詞句)	→	DET (冠詞)	+	NOUN (名詞)
NP (名詞句)	→	DET (冠詞)	+	NP (名詞句)
NP (名詞句)	→	PREP (前置詞)	+	NP (名詞句)
NP (名詞句)	→	ADJ (形容詞)	+	NOUN (名詞)
VP (動詞句)	→	VERB (動詞)	+	NP (名詞句)
VP (動詞句)	→	VPa (動詞句)	+	NP (名詞句)
VP (動詞句)	→	VERB (動詞)	+	ADV (副詞)
VP (動詞句)	→	VERB (動詞)		
VPa (動詞句)	→	VERB (動詞)	+	ADV (副詞)
VPa (動詞句)	→	VERB (動詞)	+	NP (名詞句)

```
(S36(NP9(DET1 "the")(NOUN2 "child"))(VP33(VPa11(VERB3 "runs")(ADV4 "quickly"))
    (NP26(PREP5 "to")(NP21(DET6 "the")(NP15(ADJ7 "large")(NOUN8 "house"))))))

(S28(NP1(NOUN1 "I"))(VP27(VERB2 "saw")(SS25(NP10(DET3 "a")(NOUN4 "girl"))
    (PP18(PREP5 "with")(NP13(DET6 "a")(NOUN7 "telescope"))))))
```

図 **7**　プログラムによる構文解析結果（印刷の関係上，出力は改行している．）

なお，複数の S 式を得るためには，合致する文法規則を見つけるとそこで中止して次のセルに移動するような実装では不可能で，必ず全ての文法規則についてチェックを行い，合致する文法規則を全て記憶しておく必要がある．本プログラムはそれを考慮した実装になっているため，複数の S 式が出力可能である．

しかしながら，複数の S 式が出力されるということは文の解釈を一意に特定できないということであり，この問題の解決は困難であると考えられる．

5.2　“Time flies like an arrow.” の解釈

表 6 の文法規則と**表 7** の単語辞書を用い “Time flies like an arrow” を構文解析した結果，4 つの構文木（**図 10**）と S 式（**図 11**）が得られた．これらの解釈はそれぞれ以下である:

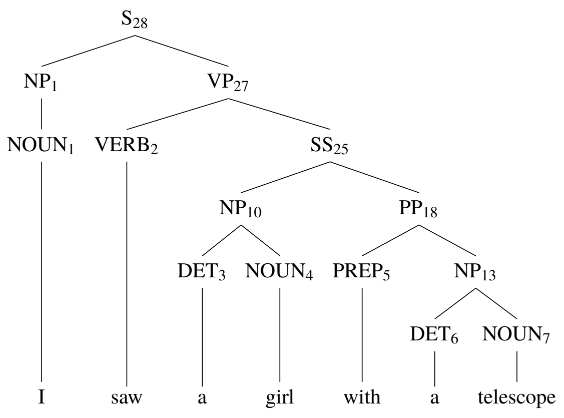
- 時は矢のように過ぎる． ⇒ 光陰矢のごとし．（**図 10a**）
- time fly（トキバエ？）たちは矢を好む．（**図 10b**）
- “矢のようなハエ” たちの速度を測定せよ．（**図 10c**）
- 矢（の速度を測定するとき）のように，ハエたちの速度を測定せよ．（**図 10d**）

さらに，“Time” が雑誌の名前である可能性も考慮すると，さらに別の解釈も可能である．

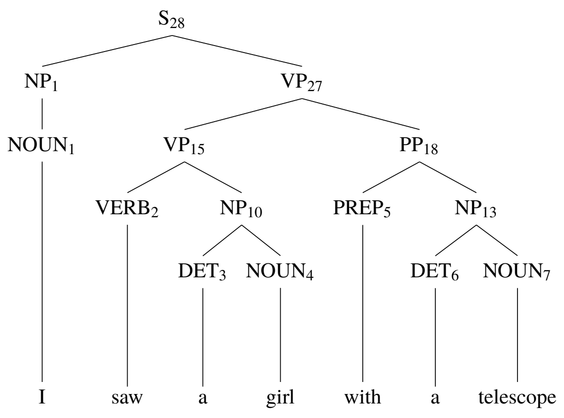
実際には，「光陰矢のごとし」以外の候補はいわゆる非文とみなせるが，候補の絞り込みには別の方法を用いる必要性が示唆される．

5.3　ネットワーク文法と文脈自由文法の能力の比較

第 3 章で触れたように，形式文法は階層構造をなしており，文脈自由文法はネットワーク文法を含む．今回作



(a) 私は望遠鏡を持った少女に会った．

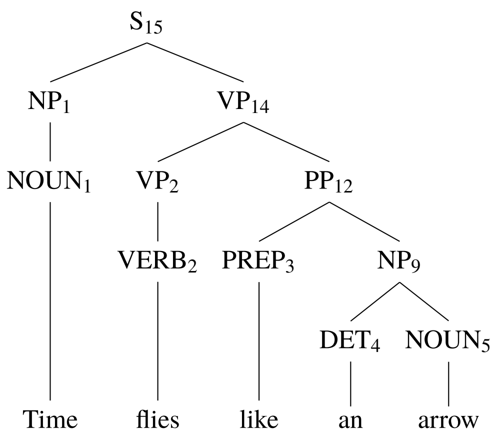


(b) 私は望遠鏡で少女を見た．

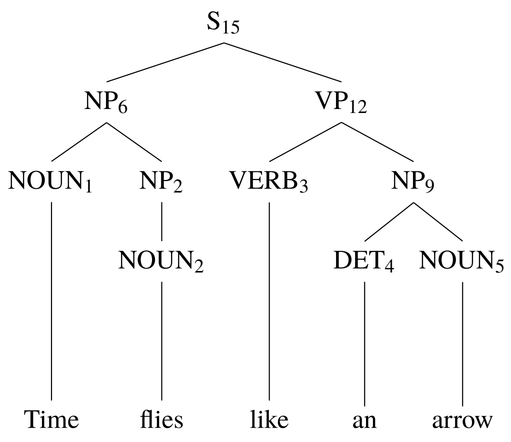
図 8　“I saw a girl with a telescope” の構文木

```
(S28(NP1(NOUN1 "I"))(VP27(VP15(VERB2 "saw")(NP10(DET3 "a")(NOUN4 "girl")))
    (PP18(PREP5 "with")(NP13(DET6 "a")(NOUN7 "telescope")))))
```

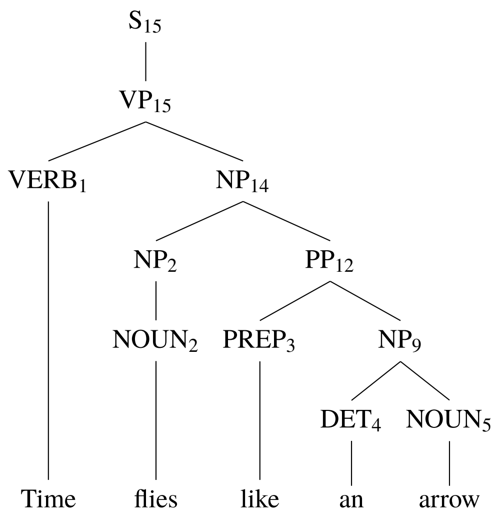
図 9　私は望遠鏡で少女を見た（**図 8b**）という解釈になる S 式（印刷の関係上，出力は改行している．）



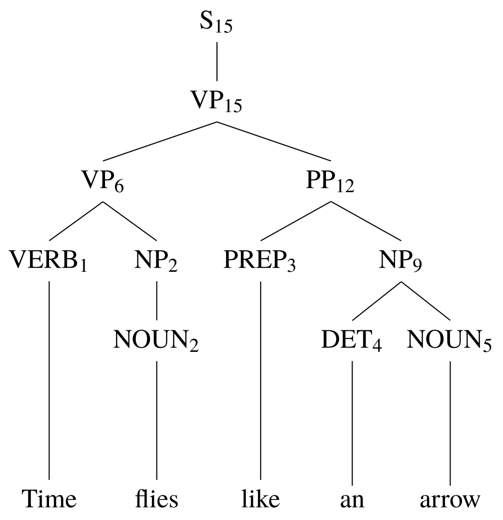
(a) 光陰矢のごとし．



(b) time fly（トキバエ？）たちは矢を好む．



(c) “矢のようなハエ” たちの速度を測定せよ．



(d) 矢（の速度を測定するとき）のように，ハエたちの速度を測定せよ．

図 10　“Time flies like an arrow” の構文木

```
(S15(NP1(NOUN1 "Time"))(VP14(VP2(VERB2 "flies"))
    (PP12(PREP3 "like")(NP9(DET4 "an")(NOUN5 "arrow")))))

(S15(NP6(NOUN1 "Time")(NP2(NOUN2 "flies")))
    (VP12(VERB3 "like")(NP9(DET4 "an")(NOUN5 "arrow"))))

(S15(VP15(VERB1 "Time")(NP14(NP2(NOUN2 "flies"))
    (PP12(PREP3 "like")(NP9(DET4 "an")(NOUN5 "arrow"))))))

(S15(VP15(VP6(VERB1 "Time")(NP2(NOUN2 "flies")))
    (PP12(PREP3 "like")(NP9(DET4 "an")(NOUN5 "arrow")))))
```

図 11　“Time flies like an arrow” の S 式（印刷の関係上，出力は改行している.）

表 6　“Time flies like an arrow” の文法規則

S	→	NP	+	VP
S	→	VP		
NP	→	NOUN		
NP	→	DET	+	NOUN
NP	→	NOUN	+	NP
NP	→	NP	+	PP
VP	→	VERB		
VP	→	VERB	+	NP
VP	→	VP	+	PP
PP	→	PREP	+	NP

表 7　“Time flies like an arrow” の単語辞書

Time	→	NOUN
Time	→	VERB
flies	→	NOUN
flies	→	VERB
like	→	PREP
like	→	VERB
an	→	DET
arrow	→	NOUN

成したプログラムにおいても，CYK 法による paser を作成する際も，形態素解析の部分にはネットワーク文法のプログラムを流用したことから，これが確認できる.

ネットワーク文法においても，遷移規則（文法規則）によって簡易的に構文解析ができたが，構文木（S 式）まで得るためには文脈自由文法を用いる必要がある.

しかしながら，文脈自由文法も，今回の考察でも分かったように，文法規則を作成することそのものも困難であるし，複数の構文木が得られてあいまいさが残る問題もある.

5.4　実験の感想

C 言語での実装は比較的大変であったが，オートマトンと構文解析について理解が深まり，本実験の目的は達せられたのではないかと思う.

6　まとめ

構文解析そのものはプログラムにより実現できたが，構文木が複数得られることから，文の解釈を一意に特定することや非文の生成を防ぐことの困難さが示唆された.

参考文献

1) 実験手順書.

付録 A　ネットワーク文法のプログラム

ソースコード **1**　Makefile

```
CC = gcc
CFLAGS = -Wall

all: main

main: main.o dict_builder.o syntax_builder.o splitter.o post.o network.o output.o
	$(CC) $(CFLAGS) -o $@ $^

%.o: %.c global.h
	$(CC) $(CFLAGS) -c $<

clean:
	$(RM) *.o

run: all
	cat data.txt | ./main dict.csv syntax.csv > result.txt
```

ソースコード **2**　global.h

```c
#ifndef GLOBAL_H
#define GLOBAL_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define bool  int
#define true  1
#define false 0

typedef struct {
  char *word;
  char *type;
} WORD;

typedef struct {
  char *current;
  char *next;
} SYNTAX;

extern WORD START;
extern WORD DUMMY;
extern SYNTAX DUMMY_SYNTAX;

bool type_analyzer(WORD *words, const WORD *dict);
bool syntax_analyzer(const WORD *words, const SYNTAX *syntax);
WORD* sentence_splitter(char *sentence);
WORD* build_dict(const char *srcfile);
SYNTAX* build_syntax(const char *srcfile);
void print_words(const WORD *words, const size_t buf_size);

#endif
```

ソースコード **3**　main.c

```c
#include "global.h"

WORD START;
WORD DUMMY;
SYNTAX DUMMY_SYNTAX;

```

```c

int main(int argc, char **argv) {
  const size_t DEFAULT_LINE_LENGTH = 1024;

  size_t max_line_length;
  if (argc > 3) {
    max_line_length = strtoul(argv[3], NULL, 10);
  } else {
    max_line_length = DEFAULT_LINE_LENGTH;
  }
  fprintf(stderr, "[INFO]:␣up␣to␣%ld␣characters␣are␣allowed␣for␣each␣sentence\n",
      max_line_length - 1);

  fprintf(stderr, "[INFO]:␣each␣sentence␣must␣be␣separated␣by␣newline␣character␣\'\\
      n\'\n");
  fprintf(stderr, "[INFO]:␣each␣word␣must␣be␣separated␣by␣whitespace␣characters\n");

  // init global variables
  DUMMY.word = "";
  DUMMY.type = "?";
  DUMMY_SYNTAX.current = "";
  DUMMY_SYNTAX.next = "";
  START.word = "␣";
  START.type = "start";

  WORD *dict = build_dict(argv[1]);
  SYNTAX *syntax = build_syntax(argv[2]);

  WORD *words = NULL;
  char *sentence = malloc(sizeof(char) * max_line_length);

  while (fgets(sentence, max_line_length, stdin) != NULL) {
    fprintf(stderr, "[INFO]:␣Parsing␣started.\n");
    words = sentence_splitter(sentence);

    bool typeOK = type_analyzer(words, dict);
    print_words(words, max_line_length);

    if (typeOK == true) {
      if (syntax_analyzer(words, syntax) == true) {
        fprintf(stderr, "[INFO]:␣\x1b[42mSentence␣OK\x1b[49m\x1b[32m␣(sentence␣is␣
            acceptable.)\x1b[39m\n");
      } else {
        fprintf(stderr, "\x1b[31m[WARN]:␣\x1b[39m\x1b[41mSentence␣NG\x1b[49m\x1b[31m
            ␣(sentence␣is␣not␣acceptable.)\x1b[39m\n");
      }
      fprintf(stderr, "[INFO]:␣Parsing␣finished.\n");
    } else {
      fprintf(stderr, "\x1b[31m[WARN]:␣can't␣start␣syntax␣checking␣because␣of␣
          unknown␣words.\x1b[39m\n");
      fprintf(stderr, "\x1b[31m[WARN]:␣Parsing␣aborted.\x1b[39m\n");
    }

    free(words);
  }

  fprintf(stderr, "[INFO]:␣exiting...");
  fflush(stderr);

  /*
  // no need?
  free(sentence);
  free(syntax[0].current);
  free(syntax);
  free(dict[0].word);
  free(dict);
  */

  fprintf(stderr, "␣Goodbye.\n");

  return 0;
}
```

ソースコード **4**　dict_builder.c

```
#include "global.h"


WORD* build_dict(const char *srcfile) {
  const int FGETS_MARGIN = 2; // do NOT decrease to < 2 (do not change)

  fprintf(stderr, "[INFO]:␣building␣dictionary␣database...");
  fflush(stderr);

  FILE *fp;
  if ((fp = fopen(srcfile, "r")) == NULL) {
    fprintf(stderr, "\n[ERROR:]␣Failed␣to␣open␣dictionary␣file.\n");
    exit(EXIT_FAILURE);
  }

  // get file size
  fseek(fp, 0, SEEK_END);
  int size = ftell(fp) + FGETS_MARGIN;
  fseek(fp, 0, SEEK_SET);

  // checking csv and copying to memory
  char *bin = malloc(sizeof(char) * size);
  int count = 0;
  char *ptr;
  int remain;
  int entry = 0;
  char c;
  for (ptr = bin, remain = size; fgets(ptr, remain, fp) != NULL; /**/) {
    if (sscanf(ptr, "␣%*[^,],%*s%c%n", &c, &count) == 1) { // csv format checking
      entry++;
      // count = strlen(ptr);
      ptr[count - 1] = '\0'; // %c (CR or LF (or space?)) -> '\0'
      ptr += count;
      remain -= count;
    } else {
      ; // do nothing
    }
  }

  WORD *dict = malloc(sizeof(WORD) * (entry + 1));
  int n[4] = {0};
  int i = 0;
  for (ptr = bin; sscanf(ptr, "␣%n%*[^,]%n,%n%*s%n", &n[0], &n[1], &n[2], &n[3]) !=
      EOF; ptr += n[3] + 1) {
    // fprintf(stderr, "%d, %d, %d, %d\n", n[0], n[1], n[2], n[3]);
    dict[i].word = ptr + n[0];
    dict[i].type = ptr + n[2];
    i++;
    ptr[n[1]] = '\0';
    ptr[n[3]] = '\0';
  }
  dict[i] = DUMMY;

  fclose(fp);
  fprintf(stderr, "␣done.\n");
  fprintf(stderr, "[INFO]:␣Dictionary␣database␣has␣%d␣entries.\n", i);

```

```c
  /*
  // write out to dict.bin (for debug)
  if ((fp = fopen("dict.bin", "wb")) == NULL) {
    fprintf(stderr, "\n[ERROR:] Failed to write dictionary file.\n");
    exit(EXIT_FAILURE);
  }
  fwrite(dict, sizeof(WORD), entry + 1, fp);
  fwrite(bin, sizeof(char), size, fp);
  fclose(fp);
  */

  return dict;
}
```

ソースコード **5**　syntax_builder.c

```c
#include "global.h"


SYNTAX* build_syntax(const char *srcfile) {
  const int FGETS_MARGIN = 2; // do NOT decrease to < 2 (do not change)

  fprintf(stderr, "[INFO]:␣building␣syntax␣database...");
  fflush(stderr);

  FILE *fp;
  if ((fp = fopen(srcfile, "r")) == NULL) {
    fprintf(stderr, "\n[ERROR]:␣Failed␣to␣open␣syntax␣file.\n");
    exit(EXIT_FAILURE);
  }

  // get file size
  fseek(fp, 0, SEEK_END);
  int size = ftell(fp) + FGETS_MARGIN;
  fseek(fp, 0, SEEK_SET);

  // checking csv and copying to memory
  char *bin = malloc(sizeof(char) * size);
  int count = 0;
  char *ptr;
  int remain;
  int entry = 0;
  char c;
  for (ptr = bin, remain = size; fgets(ptr, remain, fp) != NULL; /**/) {
    if (sscanf(ptr, "␣%*[^,],%*s%c%n", &c, &count) == 1) { // csv format checking
      entry++;
      // count = strlen(ptr);
      ptr[count - 1] = '\0'; // %c (CR or LF (or space?)) -> '\0'
      ptr += count;
      remain -= count;
    } else {
      ; // do nothing
    }
  }

  SYNTAX *syntax = malloc(sizeof(SYNTAX) * (entry + 1));
  int n[4] = {0};
  int i = 0;
  for (ptr = bin; sscanf(ptr, "␣%n%*[^,]%n,%n%*s%n", &n[0], &n[1], &n[2], &n[3]) != EOF; ptr += n[3] + 1) {
    // fprintf(stderr, "%d, %d, %d, %d\n", n[0], n[1], n[2], n[3]);
    syntax[i].current = ptr + n[0];
    syntax[i].next    = ptr + n[2];
    i++;
    ptr[n[1]] = '\0';
    ptr[n[3]] = '\0';
  }
  syntax[i] = DUMMY_SYNTAX;

  fclose(fp);
  fprintf(stderr, "␣done.\n");
  fprintf(stderr, "[INFO]:␣Syntax␣database␣has␣%d␣entries.\n", i);

  /*
  // write out to syntax.bin (for debug)
  if ((fp = fopen("syntax.bin", "wb")) == NULL) {
    fprintf(stderr, "\n[ERROR:] Failed to write syntax file.\n");
    exit(EXIT_FAILURE);
  }
  fwrite(syntax, sizeof(SYNTAX), entry + 1, fp);
  fwrite(bin, sizeof(char), size, fp);
  fclose(fp);
  */

  return syntax;
}
```

ソースコード **6**　splitter.c

```c
#include "global.h"


WORD* sentence_splitter(char *sentence) {
  char *ptr;
  int n[2] = {0};

  // check wordage
  int wordage = 0;
  for (ptr = sentence; sscanf(ptr, "␣%*s%n", &n[1]) != EOF; ptr += n[1]) {
    wordage++;
  }

  // memory allocation
  WORD *words = malloc(sizeof(WORD) * (wordage + 2));
  // wordage + 2?: last word will be DUMMY; first word will be START.

  // word separation
  int i = 1; // words[0] (i = 0) is START
  n[1] = 0;
  for (ptr = sentence; sscanf(ptr, "␣%n%*s%n", &n[0], &n[1]) != EOF; ptr += n[1] + 1) {
    words[i].word = ptr + n[0];
    i++;
    if (ptr[n[1]] == '\0') { // if reach the end of sentence
      break;
    } else {
      ptr[n[1]] = '\0';
    }
  }

  // head and tail editing
  words[0] = START;
  words[i] = DUMMY;

  return words;
}
```

ソースコード **7**　post.c

```c
#include "global.h"


```

```c
bool type_analyzer(WORD *words, const WORD *dict) {
  fprintf(stderr, "[INFO]:␣Part-of-Speech␣tagging␣(POST)␣has␣been␣executed.\n");

  int i, j;
  bool typeFound = false;
  bool noUnknown = true;
  for (i = 1; words[i].word != DUMMY.word; i++) { // words[0] is START
    fprintf(stderr, "[DEBUG]:␣%s␣=>␣", words[i].word); fflush(stderr);
    for (j = 0; dict[j].word != DUMMY.word; j++) {
      if (strcmp(words[i].word, dict[j].word) == 0) {
        fprintf(stderr, "%s\n", dict[j].type); // [DEBUG]:
        words[i].type = dict[j].type;
        typeFound = true;
        break;
      }
    }

    if (typeFound == true) {
      typeFound = false;
    } else {
      fprintf(stderr, "\x1b[41m?\x1b[49m\n"); // [DEBUG]:
      fprintf(stderr, "\x1b[31m[ERROR]:␣Unknown␣word␣\"%s\"␣detected.\x1b[39m\n", words[i].word);
      noUnknown = false;
      words[i].type = DUMMY.type;
    }
  }

  fprintf(stderr, "[INFO]:␣POST␣finished.\n");
  return noUnknown;
}
```

ソースコード 8　network.c

```c
#include "global.h"


bool syntax_analyzer(const WORD *words, const SYNTAX *syntax) {
  fprintf(stderr, "[INFO]:␣Syntax␣checking␣started.\n");

  int i, j;
  bool transOK = false;
  for (i = 0; words[i+1].type != DUMMY.type; i++) {
    fprintf(stderr, "[DEBUG]:␣%s␣(%s)␣->␣%s␣(%s)␣=>␣", words[i].type, words[i].word, words[i+1].type, words[i+1].word); fflush(stderr);
    for (j = 0; syntax[j].current != DUMMY_SYNTAX.current; j++) {
      if (strcmp(words[i].type, syntax[j].current) == 0) {
        if (strcmp(words[i+1].type, syntax[j].next) == 0) {
          fprintf(stderr, "\x1b[42mOK\x1b[49m\n"); // [DEBUG]:
          transOK = true;
          break;
        }
      }
    }

    if (transOK == true) {
      transOK = false;
    } else {
      fprintf(stderr, "\x1b[41mNG\x1b[49m\n"); // [DEBUG]:
      fprintf(stderr, "\x1b[31m[WARN]:␣Syntax␣error␣detected.\x1b[39m\n");
      return false;
    }
  }

  fprintf(stderr, "[INFO]:␣Syntax␣checking␣finished.\n");
  return true;
}
```

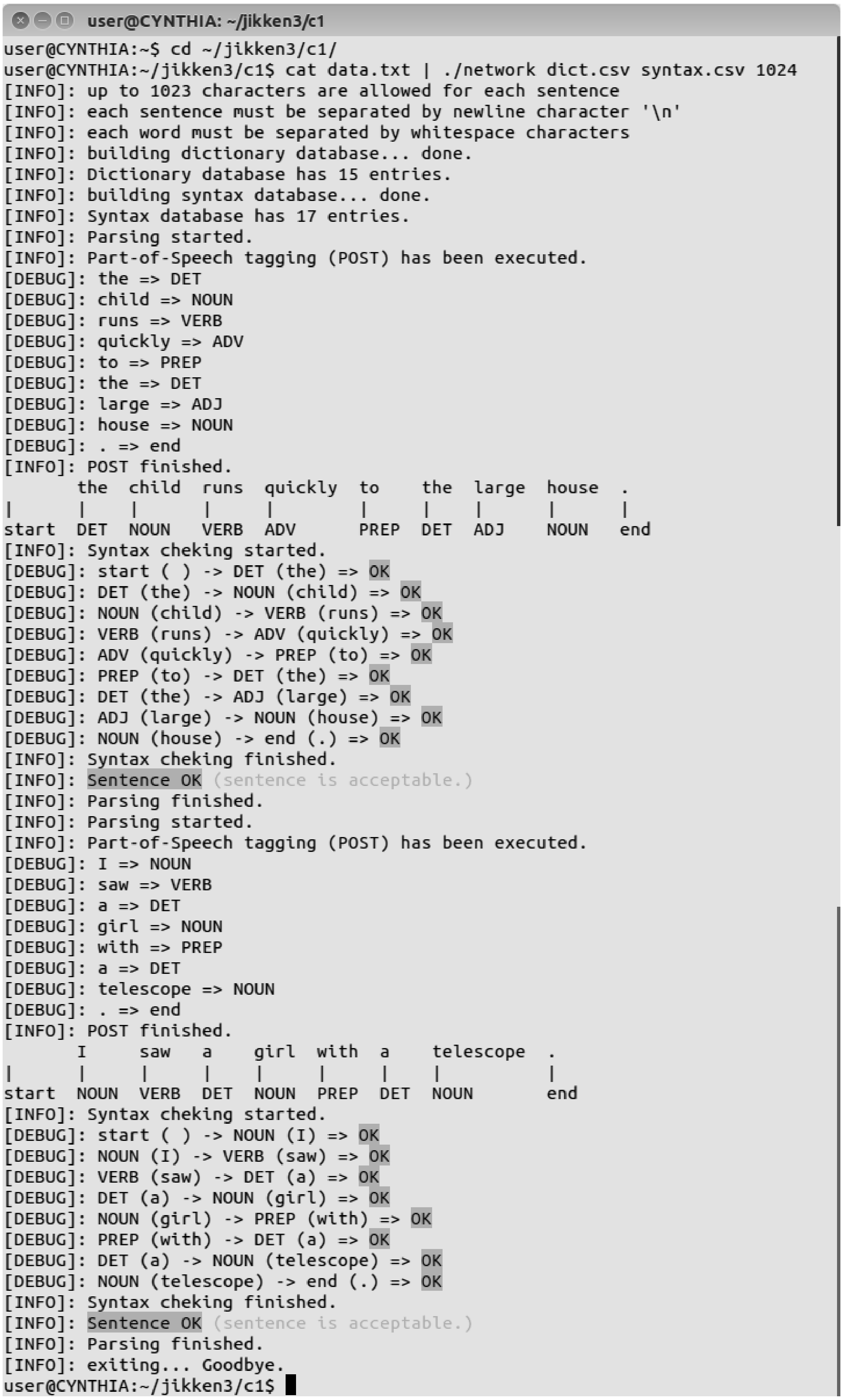
ソースコード **9**　output.c

```
#include "global.h"


void print_words(const WORD *words, const size_t buf_size) {
  const int WORD_SEPARATION = 2;
  const int LINES = 3;

  int i;

  // memory allocation
  char *line[LINES];
  line[0] = malloc(sizeof(char) * buf_size * LINES); // get heap area

  for (i = 1; i < LINES; i++) {
    line[i] = line[0] + i * buf_size;
  }

  // initializing couters and lines
  int count[LINES];
  for (i = 0; i < LINES; i++) {
    line[i][0] = '\0';
    count[i] = 0;
  }

  // padding
  int max = 0;
  int j, k;
  int remain = buf_size;
  for (i = 0; words[i].word != DUMMY.word; i++) {
    count[0] += snprintf(line[0] + count[0], remain, "%s", words[i].word);
    count[1] += snprintf(line[1] + count[1], remain, "|");
    count[2] += snprintf(line[2] + count[2], remain, "%s", words[i].type);

    // getting the longest elements
    for (j = 0; j < LINES; j++) {
      if (count[j] > buf_size - 1) { // minus 1?: last element must be terminated
           with '\0'
        // Because snprintf doesn't return the actual number of written characters,
        // count[] may exceeds buf_size.
        count[j] = buf_size - 1;
      }
      if (max < count[j]) {
        max = count[j];
      }
    }

    // padding with spaces
    for (j = 0; j < LINES; j++) {
      while (count[j] < max) {
        line[j][count[j]++] = '␣';
      }
    }
    // now, count[] == max

    remain = buf_size - max;

    if (remain > WORD_SEPARATION) {
      for (j = 0; j < LINES; j++) {
        for (k = 0; k < WORD_SEPARATION; k++) {
          line[j][count[j]++] = '␣';
        }
      }
      remain -= WORD_SEPARATION;
    } else {
      break;
    }
  }

  // terminating and printing each line
  for (i = 0; i < LINES; i++) {
    if (buf_size - count[i] > WORD_SEPARATION) {
      line[i][count[i] - WORD_SEPARATION] = '\0'; // remove last padding
    } else {
      line[i][count[i]] = '\0';
    }
    printf("%s\n", line[i]);
  }

  free(line[0]);
  return;
}
```

A.1　一文の長さの制限変更について

初期状態では 1 文の長さは 1023 文字に制限されているが，再コンパイルの必要なく，第 3 引数に n を渡すことで $n-1$ 文字までの制限に変更することができる．

これは文脈自由文法のプログラムにおいても同様である．



```
user@CYNTHIA: ~/jikken3/c1
user@CYNTHIA:~$ cd ~/jikken3/c1/
user@CYNTHIA:~/jikken3/c1$ cat data.txt | ./network dict.csv syntax.csv 1024
[INFO]: up to 1023 characters are allowed for each sentence
[INFO]: each sentence must be separated by newline character '\n'
[INFO]: each word must be separated by whitespace characters
[INFO]: building dictionary database... done.
[INFO]: Dictionary database has 15 entries.
[INFO]: building syntax database... done.
[INFO]: Syntax database has 17 entries.
[INFO]: Parsing started.
[INFO]: Part-of-Speech tagging (POST) has been executed.
[DEBUG]: the => DET
[DEBUG]: child => NOUN
[DEBUG]: runs => VERB
[DEBUG]: quickly => ADV
[DEBUG]: to => PREP
[DEBUG]: the => DET
[DEBUG]: large => ADJ
[DEBUG]: house => NOUN
[DEBUG]: . => end
[INFO]: POST finished.
       the  child  runs  quickly  to    the  large  house  .
|      |    |      |     |        |     |    |      |      |
start  DET  NOUN   VERB  ADV      PREP  DET  ADJ    NOUN   end
[INFO]: Syntax cheking started.
[DEBUG]: start ( ) -> DET (the) => OK
[DEBUG]: DET (the) -> NOUN (child) => OK
[DEBUG]: NOUN (child) -> VERB (runs) => OK
[DEBUG]: VERB (runs) -> ADV (quickly) => OK
[DEBUG]: ADV (quickly) -> PREP (to) => OK
[DEBUG]: PREP (to) -> DET (the) => OK
[DEBUG]: DET (the) -> ADJ (large) => OK
[DEBUG]: ADJ (large) -> NOUN (house) => OK
[DEBUG]: NOUN (house) -> end (.) => OK
[INFO]: Syntax cheking finished.
[INFO]: Sentence OK (sentence is acceptable.)
[INFO]: Parsing finished.
[INFO]: Parsing started.
[INFO]: Part-of-Speech tagging (POST) has been executed.
[DEBUG]: I => NOUN
[DEBUG]: saw => VERB
[DEBUG]: a => DET
[DEBUG]: girl => NOUN
[DEBUG]: with => PREP
[DEBUG]: a => DET
[DEBUG]: telescope => NOUN
[DEBUG]: . => end
[INFO]: POST finished.
       I     saw   a    girl  with  a    telescope  .
|      |     |     |    |     |     |    |          |
start  NOUN  VERB  DET  NOUN  PREP  DET  NOUN       end
[INFO]: Syntax cheking started.
[DEBUG]: start ( ) -> NOUN (I) => OK
[DEBUG]: NOUN (I) -> VERB (saw) => OK
[DEBUG]: VERB (saw) -> DET (a) => OK
[DEBUG]: DET (a) -> NOUN (girl) => OK
[DEBUG]: NOUN (girl) -> PREP (with) => OK
[DEBUG]: PREP (with) -> DET (a) => OK
[DEBUG]: DET (a) -> NOUN (telescope) => OK
[DEBUG]: NOUN (telescope) -> end (.) => OK
[INFO]: Syntax cheking finished.
[INFO]: Sentence OK (sentence is acceptable.)
[INFO]: Parsing finished.
[INFO]: exiting... Goodbye.
user@CYNTHIA:~/jikken3/c1$ 
```

図 **12**　ネットワーク文法のプログラムの動作風景

付録 B　文脈自由文法のプログラム

ソースコード **10**　Makefile

```
CC = gcc
CFLAGS = -Wall

all: main

main: main.o dict_builder.o syntax_builder.o splitter.o post.o cyk.o output.o
	$(CC) $(CFLAGS) -o $@ $^

%.o: %.c global.h
	$(CC) $(CFLAGS) -c $<

clean:
	$(RM) *.o

run: all
	cat child.txt | ./main dict.csv syntax_child.csv > result.txt
	cat telescope.txt | ./main dict.csv syntax_telescope.csv >> result.txt
	cat flies.txt | ./main dict.csv syntax_flies.csv >> result.txt
```

ソースコード **11**　global.h

```
#ifndef GLOBAL_H
#define GLOBAL_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define bool  int
#define true  1
#define false 0

#define DEFAULT_CAPACITY_OF_ARRAY 64

typedef struct {
  char *word;
  char **type;
  bool type_is_new_array;
} WORD;

typedef struct { // ex.) NP -> DET + NOUN
  char *lhs;  // left-hand side  (NP)
  char *rhs1; // right-hand side (DET)
  char *rhs2; // right-hand side (NOUN)
} SYNTAX;

typedef struct type {
  int index;
  char *type;
  struct cell *cell;
  struct type *prev_left_type;
  struct type *prev_right_type;
} TYPE;

typedef struct cell {
  int index;
  char *word;
  TYPE *type;
  bool type_is_new_array;
} CELL;

```

```c
extern WORD DUMMY;
extern SYNTAX DUMMY_SYNTAX;
extern CELL DUMMY_CELL;
extern int WORDAGE; // the number or words
extern int CELL_LENGTH; // the number or cells

bool type_analyzer(WORD *words, const WORD *dict);
CELL* syntax_analyzer(const WORD *words, const SYNTAX *syntax);
WORD* sentence_splitter(char *sentence);
WORD* build_dict(const char *srcfile);
SYNTAX* build_syntax(const char *srcfile);
void print_words(const WORD *words, const size_t buf_size, const int *type_num);
void print_sexps(const TYPE *type, int *type_num, int indent);

#endif
```

ソースコード **12**　main.c

```c
#include "global.h"

// global variables
WORD DUMMY;
SYNTAX DUMMY_SYNTAX;
CELL DUMMY_CELL;


int main(int argc, char **argv) {
  size_t max_line_length;
  if (argc > 3) {
    max_line_length = strtoul(argv[3], NULL, 10);
  } else {
    max_line_length = 1024;
  }
  fprintf(stderr, "[INFO]:␣up␣to␣%ld␣characters␣are␣allowed␣for␣each␣sentence\n",
      max_line_length - 1);

  fprintf(stderr, "[INFO]:␣each␣sentence␣must␣be␣separated␣by␣newline␣character␣\'\\
      n\'\n");
  fprintf(stderr, "[INFO]:␣each␣word␣must␣be␣separated␣by␣whitespace␣characters\n");

  // init global variables
  int i;
  DUMMY.word = "";
  DUMMY.type = (char **)"?";
  DUMMY.type_is_new_array = false;
  DUMMY_SYNTAX.lhs = "";
  DUMMY_SYNTAX.rhs1 = "";
  DUMMY_SYNTAX.rhs2 = "";

  TYPE dummy_type;
  DUMMY_CELL.index = -1;
  DUMMY_CELL.word = DUMMY.word;
  DUMMY_CELL.type = &dummy_type;
  DUMMY_CELL.type_is_new_array = false;
  dummy_type.index = -1;
  dummy_type.type = DUMMY.type[0];
  dummy_type.cell = &DUMMY_CELL;
  dummy_type.prev_left_type = &dummy_type;
  dummy_type.prev_right_type = &dummy_type;

  WORD *dict = build_dict(argv[1]);
  SYNTAX *syntax = build_syntax(argv[2]);

  WORD *words = NULL;
  char *sentence = malloc(sizeof(char) * max_line_length);

  while (fgets(sentence, max_line_length, stdin) != NULL) {
    fprintf(stderr, "[INFO]:␣Parsing␣started.\n");
    words = sentence_splitter(sentence);

    bool typeOK = type_analyzer(words, dict);
    // print_words(words, max_line_length, NULL);

    CELL *cells = NULL;
    int *type_num = malloc(sizeof(int) * WORDAGE);
    bool sentenceOK = false;
    if (typeOK == true) {
      cells = syntax_analyzer(words, syntax);
      if (cells != NULL) {
        for (i = 0; cells[CELL_LENGTH - 1].type[i].type != DUMMY_CELL.type[0].type; i++) {
          if (strcmp(cells[CELL_LENGTH - 1].type[i].type, "S") == 0) {
            sentenceOK = true;
            print_sexps(&(cells[CELL_LENGTH - 1].type[i]), type_num, 0);
            // printf("\n");
            // print_words(words, max_line_length, type_num);
            // printf("\n");
          }
        }
        if (sentenceOK == true) {
          fprintf(stderr, "[INFO]:␣\x1b[42mSentence␣OK\x1b[49m\x1b[32m␣(sentence␣is␣acceptable.)\x1b[39m\n");
        } else {
          fprintf(stderr, "\x1b[31m[WARN]:␣\x1b[39m\x1b[41mSentence␣NG\x1b[49m\x1b[31m␣(sentence␣is␣not␣acceptable.)\x1b[39m\n");
        }
        fprintf(stderr, "[INFO]:␣Parsing␣finished.\n");
      } else {
        fprintf(stderr, "\x1b[31m[WARN]:␣Parsing␣aborted.\x1b[39m\n");
      }
    } else {
      fprintf(stderr, "\x1b[31m[WARN]:␣can't␣start␣syntax␣checking␣because␣of␣unknown␣words.\x1b[39m\n");
      fprintf(stderr, "\x1b[31m[WARN]:␣Parsing␣aborted.\x1b[39m\n");
    }

    for (i = 0; cells[i].word != DUMMY_CELL.word; i++) {
      if (cells[i].type_is_new_array == true) {
        free(cells[i].type);
      }
    }
    free(cells);
    free(type_num);
    for (i = 0; words[i].word != DUMMY.word; i++) {
      if (words[i].type_is_new_array == true) {
        free(words[i].type);
      }
    }
    free(words);
  }

  fprintf(stderr, "[INFO]:␣exiting...");
  fflush(stderr);

  /*
  // no need?
  free(sentence);
  free(syntax[0].lhs);
  free(syntax);
  free(dict[0].word);
  free(dict[0].type);
  free(dict);
  */

  fprintf(stderr, "␣Goodbye.\n");

  return 0;
}
```

ソースコード **13**　dict_builder.c

```c
#include "global.h"


WORD* build_dict(const char *srcfile) {
  const int FGETS_MARGIN = 2; // do NOT decrease to < 2 (do not change)

  fprintf(stderr, "[INFO]:␣building␣dictionary␣database...");
  fflush(stderr);

  FILE *fp;
  if ((fp = fopen(srcfile, "r")) == NULL) {
    fprintf(stderr, "\n[ERROR:]␣Failed␣to␣open␣dictionary␣file.\n");
    exit(EXIT_FAILURE);
  }

  // get file size
  fseek(fp, 0, SEEK_END);
  int size = ftell(fp) + FGETS_MARGIN;
  fseek(fp, 0, SEEK_SET);

  // checking csv and copying to memory
  char *bin = malloc(sizeof(char) * size);
  int count = 0;
  char *ptr;
  int remain;
  int entry = 0;
  char c;
  for (ptr = bin, remain = size; fgets(ptr, remain, fp) != NULL; /**/) {
    if (sscanf(ptr, "␣%*[^,],%*s%c%n", &c, &count) == 1) { // csv format checking
      entry++;
      // count = strlen(ptr);
      ptr[count - 1] = '\0'; // %c (CR or LF (or space?)) -> '\0'
      ptr += count;
      remain -= count;
    } else {
      ; // do nothing
    }
  }

  WORD *dict = malloc(sizeof(WORD) * (entry + 1));
  char **type = malloc(sizeof(char *) * entry);
  int n[4] = {0};
  int i = 0;
  for (ptr = bin; sscanf(ptr, "␣%n%*[^,]%n,%n%*s%n", &n[0], &n[1], &n[2], &n[3]) !=
      EOF; ptr += n[3] + 1) {
    // fprintf(stderr, "%d, %d, %d, %d\n", n[0], n[1], n[2], n[3]);
    dict[i].type = &type[i];
    dict[i].word = ptr + n[0];
    dict[i].type[0] = ptr + n[2];
    i++;
    ptr[n[1]] = '\0';
    ptr[n[3]] = '\0';
  }
  dict[i] = DUMMY;
  dict[0].type_is_new_array = true;

  fclose(fp);
  fprintf(stderr, "␣done.\n");
  fprintf(stderr, "[INFO]:␣Dictionary␣database␣has␣%d␣entries.\n", i);

  /*
  // write out to dict.bin (for debug)
  if ((fp = fopen("dict.bin", "wb")) == NULL) {
    fprintf(stderr, "\n[ERROR:] Failed to write dictionary file.\n");
    exit(EXIT_FAILURE);
  }
  fwrite(dict, sizeof(WORD), entry + 1, fp);
  fwrite(bin, sizeof(char), size, fp);
  fclose(fp);
  */

  return dict;
}
```

ソースコード **14**　syntax_builder.c

```
#include "global.h"


SYNTAX* build_syntax(const char *srcfile) {
  const int FGETS_MARGIN = 2; // do NOT decrease to < 2 (do not change)

  fprintf(stderr, "[INFO]:␣building␣phrase␣structure␣rules␣database...");
  fflush(stderr);

  FILE *fp;
  if ((fp = fopen(srcfile, "r")) == NULL) {
    fprintf(stderr, "\n[ERROR]:␣Failed␣to␣open␣syntax␣file.\n");
    exit(EXIT_FAILURE);
  }

  // get file size
  fseek(fp, 0, SEEK_END);
  int size = ftell(fp) + FGETS_MARGIN;
  fseek(fp, 0, SEEK_SET);

  // checking csv and copying to memory
  char *bin = malloc(sizeof(char) * size);
  int count = 0;
  char *ptr;
  int remain;
  int entry = 0;
  char c;
  for (ptr = bin, remain = size; fgets(ptr, remain, fp) != NULL; /**/) {
    // csv format checking
    if (sscanf(ptr, "␣%*[^,],%*s%c%n", &c, &count)         == 1 || // a,b,
        sscanf(ptr, "␣%*[^,],%*[^,],%*s%c%n", &c, &count) == 1) { // a,b,c
      entry++;
      // count = strlen(ptr);
      ptr[count - 1] = '\0'; // %c (CR or LF (or space?)) -> '\0'
      ptr += count;
      remain -= count;
    } else {
      ; // do nothing
    }
  }

  SYNTAX *syntax = malloc(sizeof(SYNTAX) * (entry + 1));
  int n[6] = {0};
  int i = 0;
  ptr = bin;
  for (;;) { // infinite loop
    if (sscanf(ptr, "␣%n%*[^,]%n,%n%*[^,]%n,%n%*s%n", &n[0], &n[1], &n[2], &n[3], &n
          [4], &n[5]) != EOF) {
        // fprintf(stderr, "%d, %d, %d, %d\n", n[0], n[1], n[2], n[3], n[4], n[5]);
        syntax[i].lhs = ptr + n[0];
        syntax[i].rhs1 = ptr + n[2];
        syntax[i].rhs2 = ptr + n[4];
        i++;
        ptr[n[1]] = '\0';
        ptr[n[3]] = '\0';
        ptr[n[5]] = '\0';
        ptr += n[5] + 1;
      } else
      if (sscanf(ptr, "␣%n%*[^,]%n,%n%*s%n", &n[0], &n[1], &n[2], &n[3]) != EOF) {
        // fprintf(stderr, "%d, %d, %d, %d\n", n[0], n[1], n[2], n[3]);
        syntax[i].lhs = ptr + n[0];
        syntax[i].rhs1 = ptr + n[2];
        syntax[i].rhs2 = DUMMY_SYNTAX.rhs2;
        i++;
        ptr[n[1]] = '\0';
        ptr[n[3]] = '\0';
        if (ptr[n[3] - 1] == ',') {
          // remove comma (rhs1 may include comma at its tail)
          ptr[n[3] - 1] = '\0';
        }
        ptr += n[3] + 1;
      } else {
        break;
      }
    }
    syntax[i] = DUMMY_SYNTAX;

    fclose(fp);
    fprintf(stderr, "␣done.\n");
    fprintf(stderr, "[INFO]:␣Syntax␣database␣has␣%d␣phrase␣structure␣rules.\n", i);

    /*
    // write out to syntax.bin (for debug)
    if ((fp = fopen("syntax.bin", "wb")) == NULL) {
      fprintf(stderr, "\n[ERROR:] Failed to write syntax file.\n");
      exit(EXIT_FAILURE);
    }
    fwrite(syntax, sizeof(SYNTAX), entry + 1, fp);
    fwrite(bin, sizeof(char), size, fp);
    fclose(fp);
    */

    return syntax;
}
```

ソースコード **15**　splitter.c

```
#include "global.h"

int WORDAGE;


WORD* sentence_splitter(char *sentence) {
  char *ptr;
  int n[2] = {0};

  // check wordage
  WORDAGE = 0;
  for (ptr = sentence; sscanf(ptr, "␣%*s%n", &n[1]) != EOF; ptr += n[1]) {
    WORDAGE++;
  }

  // memory allocation
  WORD *words = malloc(sizeof(WORD) * (WORDAGE + 1));
  // WORDAGE + 1?: last word will be DUMMY.

  // word separation
  int i = 0;
  n[1] = 0;
  for (ptr = sentence; sscanf(ptr, "␣%n%*s%n", &n[0], &n[1]) != EOF; ptr += n[1] +
      1) {
    words[i].word = ptr + n[0];
    i++;
    if (ptr[n[1]] == '\0') { // if reach the end of sentence
      break;
    } else {
      ptr[n[1]] = '\0';
    }
  }

  // head and tail editing
  words[i] = DUMMY;

  return words;
}
```

ソースコード **16**　post.c

```
#include "global.h"


bool type_analyzer(WORD *words, const WORD *dict) {
  fprintf(stderr, "[INFO]:␣Part-of-Speech␣tagging␣(POST)␣has␣been␣executed.\n");

  int i, j, k;
  bool typeFound = false;
  bool noUnknown = true;

  int array_size = DEFAULT_CAPACITY_OF_ARRAY;
  char **type = malloc(sizeof(char *) * array_size);
  int type_index = 0;
  for (i = 0; words[i].word != DUMMY.word; i++) {
    outer: // outer loop
    words[i].type = &type[type_index];
    if (type_index == 0) {
      words[i].type_is_new_array = true;
    }
    k = 0;
    // fprintf(stderr, "[DEBUG]: %s => ", words[i].word); fflush(stderr);
    for (j = 0; dict[j].word != DUMMY.word; j++) {
      if (strcmp(words[i].word, dict[j].word) == 0) {
        typeFound = true;
        if (type_index >= (array_size - 1)) { // -1?: last element is preserved for
            dummy element.
          array_size *= 2; // new array will be expanded.
          fprintf(stderr, "[DEBUG]:␣words[i].type␣is␣full.␣increasing␣array␣size␣to␣
              %d␣elements.\n", array_size);
          if (k == type_index) {
            free(type);
          }
          type = malloc(sizeof(char *) * array_size);
          type_index = 0;
          goto outer; // continue from outer loop
        }
        words[i].type[k] = dict[j].type[0];
        // fprintf(stderr, "%s\n", words[i].type[k]); // [DEBUG]:
        k++;
        type_index++;
      }
```

```c
    }
    words[i].type[k] = DUMMY.type[0];
    k++;
    type_index++;

    if (typeFound == true) {
      typeFound = false;
    } else {
      fprintf(stderr, "[DEBUG]:␣%s␣=>␣", words[i].word);
      fprintf(stderr, "\x1b[41m?\x1b[49m\n"); // [DEBUG]:
      fprintf(stderr, "\x1b[31m[ERROR]:␣Unknown␣word␣\"%s\"␣detected.\x1b[39m\n", words[i].word);
      noUnknown = false;
    }
  }

  fprintf(stderr, "[INFO]:␣POST␣finished.\n");
  return noUnknown;
}
```

ソースコード **17**　cyk.c

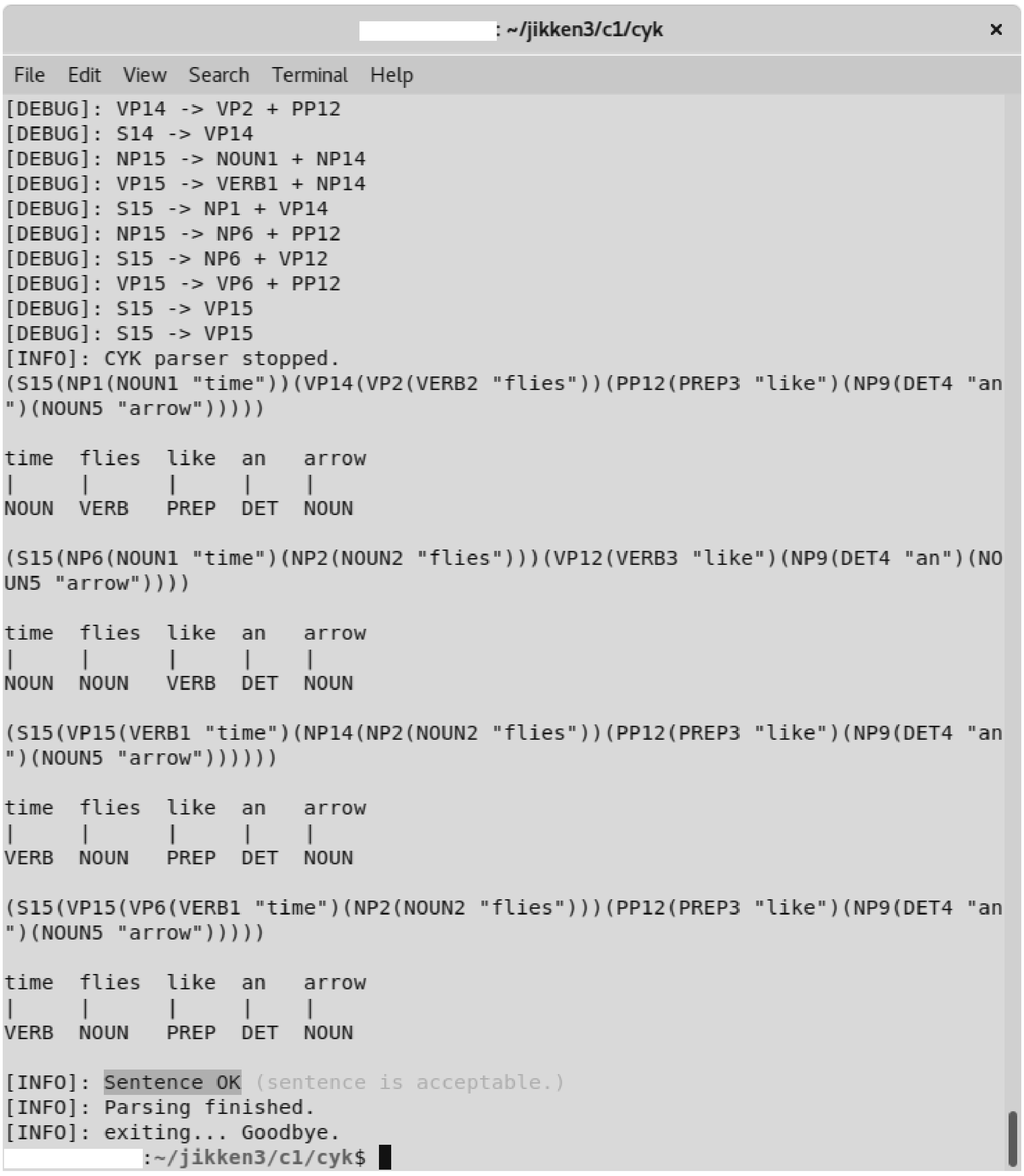
```c
#include "global.h"

// global variables
int CELL_LENGTH;


int getIndex(const int x, const int y) {
// this method for upper triangular matrix
  int index = 0;
  int column = 0;
  int i, j;
  for (i = 0, j = 0; i < WORDAGE && j < WORDAGE; /* */) {
    if (i == x && j == y) {
      return index;
    }

    if (j == WORDAGE - 1) {
      i = 0;
      j = ++column;
    } else {
      i++;
      j++;
    }
    index++;
  }

  return -1;
}


bool setNewArrayIfNeeded(TYPE **type, int *type_index, int *array_size, const int *k) {
  if (*type_index >= (*array_size - 1)) { // -1?: last element is preserved for dummy element.
    *array_size *= 2; // new array will be expanded.
    fprintf(stderr, "[DEBUG]:␣cells[index].type␣is␣full.␣increasing␣array␣size␣to␣%d␣elements.\n", *array_size);
    if (*k == *type_index) {
      free(*type);
    }
    *type = malloc(sizeof(TYPE) * *array_size);

    int i;
    for (i = 0; i < *array_size; i++) { // init type[] with DUMMY_CELL.type[0]
```

```
      memcpy(&(*type)[i], &DUMMY_CELL.type[0], sizeof(TYPE));
    }

    *type_index = 0;
    return true;
  } else {
    return false;
  }
}


CELL* syntax_analyzer(const WORD *words, const SYNTAX *syntax) {
  fprintf(stderr, "[INFO]:␣CYK␣parser␣started.\n");

  // memory allocation
  CELL_LENGTH = (WORDAGE * (WORDAGE + 1)) / 2; // upper triangular matrix (n*(n
      +1)/2)
  if (CELL_LENGTH == 0) {
    fprintf(stderr, "\x1b[31m[WARN]:␣CYK␣parser␣stopped␣(word␣not␣found).\x1b[39m\n"
        );
    return NULL;
  }
  CELL *cells = malloc(sizeof(CELL) * CELL_LENGTH);

  int index;

  // init cells
  for (index = 0; index < CELL_LENGTH; index++) {
    cells[index] = DUMMY_CELL; // for unused-member initializing
    cells[index].index = index;
  }

  int i, j; // for loop
  int k; // cells[index].type[k]
  int p, q, a; // comparison cells[a] (x, y) = (p, q)
  int u, v, b; // comparison cells[b] (x, y) = (u, v)
  int l, m, n; // for loop
  int column = 0;
  index = 0;
  int array_size = DEFAULT_CAPACITY_OF_ARRAY;
  TYPE *type = malloc(sizeof(TYPE) * array_size);
  for (i = 0; i < array_size; i++) { // init type[] with DUMMY_CELL.type[0]
    memcpy(&type[i], &DUMMY_CELL.type[0], sizeof(TYPE));
  }
  int type_index = 0;
  for (i = 0, j = column; i < WORDAGE && j < WORDAGE; /* */) {
    outer: // outer loop
    cells[index].type = &type[type_index];
    if (type_index == 0) {
      cells[index].type_is_new_array = true;
    }
    k = 0;

    // (cells[index].word <= words[index].word)
    if (index < WORDAGE) {
      cells[index].word = words[index].word;
      for (i = 0; words[index].type[i] != DUMMY.type[0]; i++) {
        if (setNewArrayIfNeeded(&type, &type_index, &array_size, &k) == true) {
          goto outer; // continue from outer loop
        }

        cells[index].type[k].index = k;
        cells[index].type[k].type = words[index].type[i];
        cells[index].type[k].cell = &cells[index];
        fprintf(stderr, "[DEBUG]:␣%s%d␣\"%s\"\n", cells[index].type[k].type, cells[
            index].index + 1, cells[index].word);
        k++;
        type_index++;
      }
    }

    // ex.) S -> NP + VP; VP -> VP + NP
    p = i;
    v = j;
    for (q = i, u = i + 1; q < j && u <= j; q++, u++) {
      a = getIndex(p, q);
      b = getIndex(u, v);
      // fprintf(stderr, "[DEBUG]: %d: (%d, %d) = %d, (%d, %d) = %d\n", index+1, p
          +1, q+1, a+1, u+1, v+1, b+1);

      for (l = 0; cells[a].type[l].type != DUMMY_CELL.type[0].type; l++) {
        for (m = 0; cells[b].type[m].type != DUMMY_CELL.type[0].type; m++) {
          for (n = 0; syntax[n].lhs != DUMMY_SYNTAX.lhs; n++) {
            if (strcmp(cells[a].type[l].type, syntax[n].rhs1) == 0 && strcmp(cells[b
                ].type[m].type, syntax[n].rhs2) == 0) {
              if (setNewArrayIfNeeded(&type, &type_index, &array_size, &k) == true)
                  {
                goto outer; // continue from outer loop
              }

              cells[index].type[k].index = k;
              cells[index].type[k].type = syntax[n].lhs;
              cells[index].type[k].cell = &cells[index];
              cells[index].type[k].prev_left_type = &cells[a].type[l];
              cells[index].type[k].prev_right_type = &cells[b].type[m];
              fprintf(stderr, "[DEBUG]:␣%s%d␣->␣%s%d␣+␣%s%d\n",
                cells[index].type[k].type, cells[index].type[k].cell->index + 1,
                cells[index].type[k].prev_left_type->type, cells[index].type[k].
                    prev_left_type->cell->index + 1,
                cells[index].type[k].prev_right_type->type, cells[index].type[k].
                    prev_right_type->cell->index + 1
              );
              k++;
              type_index++;
            }
          }
        }
      }
    }

    // self-referencing (ex.) VP -> VERB; S -> VP
    for (l = 0; cells[index].type[l].type != DUMMY_CELL.type[0].type; l++) {
      for (n = 0; (syntax[n].lhs != DUMMY_SYNTAX.lhs); n++) {
        if (strcmp(cells[index].type[l].type, syntax[n].rhs1) == 0 && syntax[n].rhs2
            == DUMMY_SYNTAX.rhs2) {
          if (setNewArrayIfNeeded(&type, &type_index, &array_size, &k) == true) {
            goto outer; // continue from outer loop
          }

          cells[index].type[k].index = k;
          cells[index].type[k].type = syntax[n].lhs;
          cells[index].type[k].cell = &cells[index];
          cells[index].type[k].prev_left_type = &cells[index].type[l];
          fprintf(stderr, "[DEBUG]:␣%s%d␣->␣%s%d\n",
            cells[index].type[k].type, cells[index].type[k].cell->index + 1,
            cells[index].type[k].prev_left_type->type, cells[index].type[k].
                prev_left_type->cell->index + 1
          );
          k++;
          type_index++;
        }
      }
    }

    if (j == WORDAGE - 1) {
      i = 0;
      j = ++column;
    } else {
      i++;
      j++;
    }

    if (k == 0) { // when k == 0, the cell is empty.
      if (type_index == 0) { // type_index == 0 means the head of type[].
        type_index++;
      } else {
        cells[index].type--; // cells[index].type points last dummy element of
            previous cell (cells[index - 1])
      }                      //   to save memory resource
    } else {
      type_index++;  // make last element dummy
    }

    setNewArrayIfNeeded(&type, &type_index, &array_size, &k);
    index++;
  }

  fprintf(stderr, "[INFO]:␣CYK␣parser␣stopped.\n");
  return cells;
}
```

ソースコード **18**　output.c

```
#include "global.h"


void _print_sexps(const TYPE *type, int *word_num, int *type_num, int indent, int
    depth) {
  if (type->type == DUMMY_CELL.type[0].type) {
    return;
  }

  if (indent > 0) {
    int i, j;
    printf("\n");
    for (i = 0; i < depth; i++) {
      for (j = 0; j < indent; j++) {
        printf("␣");
      }
    }
  }
  printf("(%s%d", type->type, (type->cell)->index + 1);

  if ((type->cell)->word != DUMMY_CELL.word) {
    if (type->prev_left_type == &DUMMY_CELL.type[0]) {
      printf("␣\"%s\"", (type->cell)->word);
      if (type_num != NULL) {
        //fprintf(stderr, "%d, ", type->index);
        type_num[(*word_num)++] = type->index;
      }
    }
  }

  _print_sexps(type->prev_left_type, word_num, type_num, indent, depth + 1);
  _print_sexps(type->prev_right_type, word_num, type_num, indent, depth + 1);
  printf(")");

  return;
```

```
}


void print_sexps(const TYPE *type, int *type_num, int indent) {
  int word_num = 0;
  _print_sexps(type, &word_num, type_num, indent, 0);
  printf("\n");
  return;
}


// legacy
void print_words(const WORD *words, const size_t buf_size, const int *type_num) {
  const int WORD_SEPARATION = 2;
  const int LINES = 3;

  int i;

  // memory allocation
  char *line[LINES];
  line[0] = malloc(sizeof(char) * buf_size * LINES); // get heap area

  for (i = 1; i < LINES; i++) {
    line[i] = line[0] + i * buf_size;
  }

  // initializing couters and lines
  int count[LINES];
  for (i = 0; i < LINES; i++) {
    line[i][0] = '\0';
    count[i] = 0;
  }

  // padding
  int max = 0;
  int j, k;
  int remain = buf_size;
  for (i = 0; words[i].word != DUMMY.word; i++) {
    count[0] += snprintf(line[0] + count[0], remain, "%s", words[i].word);
    count[1] += snprintf(line[1] + count[1], remain, "|");
    if (type_num == NULL) {
      count[2] += snprintf(line[2] + count[2], remain, "%s", words[i].type[0]);
    } else {
      count[2] += snprintf(line[2] + count[2], remain, "%s", words[i].type[type_num[
          i]]);
    }

    // getting the longest elements
    for (j = 0; j < LINES; j++) {
      if (count[j] > buf_size - 1) { // minus 1?: last element must be terminated
          with '\0'
        // Because snprintf doesn't return the actual number of written characters,
        // count[] may exceeds buf_size.
        count[j] = buf_size - 1;
      }
      if (max < count[j]) {
        max = count[j];
      }
    }

    // padding with spaces
    for (j = 0; j < LINES; j++) {
      while (count[j] < max) {
        line[j][count[j]++] = '␣';
      }
    }
    // now, count[] == max

    remain = buf_size - max;

    if (remain > WORD_SEPARATION) {
      for (j = 0; j < LINES; j++) {
        for (k = 0; k < WORD_SEPARATION; k++) {
          line[j][count[j]++] = '␣';
        }
      }
      remain -= WORD_SEPARATION;
    } else {
      break;
    }
  }

  // terminating and printing each line
  // fprintf(stderr, "[INFO]:\n");
  for (i = 0; i < LINES; i++) {
    if (buf_size - count[i] > WORD_SEPARATION) {
      line[i][count[i] - WORD_SEPARATION] = '\0'; // remove last padding
    } else {
      line[i][count[i]] = '\0';
    }
    fprintf(stdout, "%s\n", line[i]);
  }

  free(line[0]);
  return;
}
```


[redacted]: ~/jikken3/c1/cyk
File Edit View Search Terminal Help

```
[DEBUG]: VP14 -> VP2 + PP12
[DEBUG]: S14 -> VP14
[DEBUG]: NP15 -> NOUN1 + NP14
[DEBUG]: VP15 -> VERB1 + NP14
[DEBUG]: S15 -> NP1 + VP14
[DEBUG]: NP15 -> NP6 + PP12
[DEBUG]: S15 -> NP6 + VP12
[DEBUG]: VP15 -> VP6 + PP12
[DEBUG]: S15 -> VP15
[DEBUG]: S15 -> VP15
[INFO]: CYK parser stopped.
(S15(NP1(NOUN1 "time"))(VP14(VP2(VERB2 "flies"))(PP12(PREP3 "like")(NP9(DET4 "an
")(NOUN5 "arrow")))))

time  flies  like  an   arrow
|     |      |     |    |
NOUN  VERB   PREP  DET  NOUN

(S15(NP6(NOUN1 "time")(NP2(NOUN2 "flies")))(VP12(VERB3 "like")(NP9(DET4 "an")(NO
UN5 "arrow"))))

time  flies  like  an   arrow
|     |      |     |    |
NOUN  NOUN   VERB  DET  NOUN

(S15(VP15(VERB1 "time")(NP14(NP2(NOUN2 "flies"))(PP12(PREP3 "like")(NP9(DET4 "an
")(NOUN5 "arrow"))))))

time  flies  like  an   arrow
|     |      |     |    |
VERB  NOUN   PREP  DET  NOUN

(S15(VP15(VP6(VERB1 "time")(NP2(NOUN2 "flies")))(PP12(PREP3 "like")(NP9(DET4 "an
")(NOUN5 "arrow")))))

time  flies  like  an   arrow
|     |      |     |    |
VERB  NOUN   PREP  DET  NOUN

[INFO]: Sentence OK (sentence is acceptable.)
[INFO]: Parsing finished.
[INFO]: exiting... Goodbye.
[redacted]:~/jikken3/c1/cyk$
```

図 **13**　文脈自由文法のプログラムの動作風景

B.1　インデント機能の使用

S 式出力用の関数 `print_sexps` はインデント機能を有している．`main` 関数内で

```
print_sexps(&(cells[CELL_LENGTH - 1].type[i]), type_num, 0);
```

と呼び出している部分の最後の引数を例えば

```
print_sexps(&(cells[CELL_LENGTH - 1].type[i]), type_num, 4);
```

とすれば 4 文字でのインデントが有効になり，簡易的に木構造を確認できる（図 **14**）.

```
(S36
    (NP9
        (DET1 "the")
        (NOUN2 "child"))
    (VP33
        (VPa11
            (VERB3 "runs")
            (ADV4 "quickly"))
        (NP26
            (PREP5 "to")
            (NP21
                (DET6 "the")
                (NP15
                    (ADJ7 "large")
                    (NOUN8 "house"))))))
```

図 **14**　インデント機能を有効にした場合の出力