1.プログラムの仕様

・プログラムの機能

問題をファイルから読み込む機能。
ゲームの最後に間違えた問題、正解数、間違えた数、正解率を表示する機能。

・プログラムの使い方

オプションでファイル名を指定するとそのファイルから問題を読み込む。デフォルトでは同じ階層にある″toeic1500.dat”を使用する。ファイルのフォーマットは″toeic1500.dat”と同じでなければならない。

・使用したデータ構造

使用したキーは、アルファベットの位置(a なら 0、b なら 1、...、z なら 25)をインデックスとする配列で管理される。インデックスの位置の値が 1 なら使用済み、そうでないなら未使用とした。

失敗した問題のデータはリストで保持される。

ハングマンに必要なデータは構造体″hangman_context”で管理される。
この構造体の全ての操作は″hangman_*”関数を通して行われるべき。

result 列挙体は″succeeded”と″failed”を持ち、関数の実行結果を返すのに使われる。

・各関数の仕様

size_t FILE_count_line(FILE* f);
ファイル″f”の行数を返す。

char* FILE_get_line_at(FILE* f, size_t line);
ファイル″f”の″line”行目を返す。メモリは呼び出した側が解放しなければいけない。

void hangman_init(hangman_context* hm);
“hm”を初期化する。必ず呼ばれなければいけない。

void hangman_destroy_context(hangman_context* hm);
“hm”に関連するリソースの解放をする。以降、″hm”は初期化するまで操作されるべきではない。

result hangman_set_problem(hangman_context* hm, char const* file_name);
“file_name”からランダムに 1 つの単語を選び、それを正解として″hm”に設定する。

void hangman_check_alphabet(hangman_context* hm, char alphabet);
”hm”に″alphabet”を入力し、″hm”を更新する。

char const* hangman_get_answer(hangman_context const* hm);
正解を取得する。

char const* hangman_get_disp(hangman_context const* hm);
画面に表示すべき正解状況を返す。例えば、正解が"apple"で、p がすでに入力されていた場合"-pp--"という文字列が返される。

char const* hangman_get_nihongo(hangman_context const* hm);
正解の英単語の日本語の意味を返す。

int hangman_get_remaining(hangman_context const* hm);
残りの間違えて良い回数を返す。

int hangman_is_clear(hangman_context const* hm);
クリアされているかを調べる。クリアされている場合は 0、そうでない時は負の数を返す。

void hangman_print_used(hangman_context const* hm);
すでに使用されているアルファベットを表示する。

void ctrl_hangman(char const* file_name);
"file_name"から問題を読み取って、ユーザーが望む限りハングマンを継続する。

result play_hangman(hangman_context* hm);
hangman_context 操作関数を使い"hm"を操作し、ハングマンを進行させる。
プレイヤーが正解したら"succeeded"を返す。正解できなかったら"failed"を返す。

void string_list_create(string_list_node** list);
リストを作成し、*list が作成されたリストを指すようにする。

void string_list_destroy(stirng_list_node* list);
"list"に関するメモリを解放する。

stirng_list_node* string_list_add(string_list_node* list, char const* str);
"str"を新しく確保したメモリにコピーし、"list"の最後に追加する。
追加されたノードのアドレスを返す。

void print_failed_problem(stirng_list_node* head);
間違えた問題の解答と日本語訳をすべて表示する。

2.プログラムの正しさの検証

厳密に正しく動くかテストするのは大変なので、何回か実際に遊んで問題なくすることを確かめた。
ファイルがない場合もそのことを表示して正常に終了するし、メモリリークにも細心の注意を払ってプログラムを書いた。
想定されていないフォーマットのファイルの入力には対応していない。

3.問題点

間違えた問題を答えと日本語訳を含めて、リストで保持するのはメモリの無駄だと思った。ファイルに書き出して、終了時にそのファイルの内容を表示するようにするべきだった。

正解判定はプレイヤーの入力が合った時に 1 回のみ行い、結果をフラグに格納すると、正解判定を要求された時に、いちいち計算する必要がなくなるので、良いと思った。

入力されるファイルが想定外のフォーマットの場合の動作が未定義なので、ファイルのフォーマットをチェックできるようにしたい。

4.プログラムリスト

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>


//A node that consists of a list to store problems the player failed.

typedef struct string_list_node{

	char *str_;


	struct string_list_node* next_;

}string_list_node;


//Creates a list object and let the passed pointer point to the list

void string_list_create(string_list_node** list){

	*list = malloc(sizeof(string_list_node));

	(*list)->str_ = NULL;

	(*list)->next_ = NULL;

}


void print_failed_problems(string_list_node* head){

	string_list_node* node = head->next_;

	while(node != NULL){

		printf("%s\n", node->str_);
```

```
                node = node->next_;
                printf("%s\n\n", node->str_);
                node = node->next_;
        }
}

//Deallocates memory
void string_list_destroy(string_list_node* target){
        if(target == NULL){
                return;
        }
        string_list_node* next = target->next_;
        free(target->str_);
        free(target);

        string_list_destroy(next);
}

//Adds an string next to the "list"
string_list_node* string_list_add(string_list_node* list, char const* str){
        list->next_ = malloc(sizeof(string_list_node));
        list->next_->str_ = malloc(strlen(str) + 1);
        list->next_->next_ = NULL;
        strcpy(list->next_->str_, str);

        return list->next_;
}

char getChar(void);

typedef enum{
```

```
	used,
	unused,
}alphabet_status;

typedef enum{
	succeeded,
	failed,
}result;

typedef struct{
	//Use this pointer to deallocate the memory. This poins to memory storing the line of a current problem.
	char *to_free_;

	//These 2 pointers point to a string representing an answer and the meaning in Japanese.
	char const* answer_;
	char const* nihongo_;

	//A string to be displayed, must be freed.
	char* disp_;

	//The length of a answer word
	int answer_length_;

	//The number of times the player can fail.
	int remaining_;

	//To manage used alphabets
	alphabet_status alphabet_map_[26];
}hangman_context;
```

```
//Counts the total line of "f"
size_t FILE_count_line(FILE* f){
	int temp;
	int line_ = 0;

	for(temp = fgetc(f); temp != EOF; temp = fgetc(f)){
		if(temp == '\n')
			line_++;
	}
	fseek(f, 0, SEEK_SET);

	return line_;
}

//Return value: A pointer to memory storing a string of the desired line. The pointer should be freed by yourself.
//If the "line" is greater than the actual number of f's line, behaviour is undefined.
char* FILE_get_line_at(FILE* f, size_t line){
	int i = 0;
	char* read;

	line--;
	while(line > 0){
		if(fgetc(f) == '\n')
			line--;
	}

	int num_char = 0;
	while(fgetc(f) != '\n'){
		num_char++;
```

```
        }

        num_char++;
        fseek(f, -num_char, SEEK_CUR);

        read = malloc(num_char+1);

        fgets(read, num_char, f);

        return read;
}

//Initilizes a hangman context.
void hangman_init(hangman_context* hm){
        hm->remaining_ = 7;

        hm->answer_ = NULL;
        hm->nihongo_ = NULL;
        hm->disp_ = NULL;

        hm->answer_length_ = 0;

        int i;
        for(i = 0; i < 26; i++){
                hm->alphabet_map_[i] = unused;
        }
}

//Sets a problem from "file_name".
//Returns 0 on success or -1 on failure.
result hangman_set_problem(hangman_context* hm, char const* file_name){
```

```
	FILE* f = fopen(file_name, "r");
	if(f == NULL){
		fprintf(stderr, "could not read the file:%s\n", file_name);
		return failed;
	}
	int file_line_count = FILE_count_line(f);

	char* line;
	int i;

	int problem_number = rand() % file_line_count + 1;
	hm->to_free_ = line = FILE_get_line_at(f, problem_number);

	while(*(line++) != ' ');
	hm->answer_ = line;

	while(*(line++) != ' ');
	*(line-1) = '\0';

	hm->nihongo_ = line;

	hm->disp_ = malloc(strlen(hm->nihongo_)+1);
	for(i = 0; i < strlen(hm->answer_); i++){
		hm->disp_[i] = '-';
	}
	hm->disp_[i] = '\0';

	fclose(f);

	return succeeded;
}
```

```
//Checks if "alphabet" is valid.
//If "alphabet" is valid, "hm" will be modified.
void hangman_check_alphabet(hangman_context* hm, char const alphabet){
	int i;
	char not_hit = 1;

	if(alphabet < 'a' || alphabet > 'z')
		return;

	if(hm->alphabet_map_[alphabet-'a'] == used)
		return;

	hm->alphabet_map_[alphabet-'a'] = used;

	for(i = 0; hm->answer_[i] != '\0'; i++){
		if(alphabet == hm->answer_[i]){
			not_hit = 0;
			hm->disp_[i] = hm->answer_[i];
		}
	}

	if(not_hit)
		hm->remaining_ -= 1;
}

//Prints used alphabets.
void hangman_print_used(hangman_context const* hm){
	int i;
	for(i = 0; i < 26; i++){
		if(hm->alphabet_map_[i] == used){
```

```
                printf("%c", 'a' + i);
            }
        }
}

//Returns 1 if the player solved a problem or 0 the player not solved.
int hangman_is_clear(hangman_context const* hm){
        int i;
        for(i = 0; hm->disp_[i]; i++){
            if(hm->disp_[i] == '-'){
                return 0;
            }
        }
        return 1;
}

//Returns the answer of a current problem.
char const* hangman_get_answer(hangman_context const* hm){
        return hm->answer_;
}

//Returns the meaning in Japanese of an answer.
char const* hangman_get_nihongo(hangman_context const* hm){
        return hm->nihongo_;
}

//Returns a string to be displayed.
char const* hangman_get_disp(hangman_context const* hm){
        return hm->disp_;
}
```

```
//Returns a number of remaining.
int hangman_get_remaining(hangman_context const* hm){
	return hm->remaining_;
}

//Deallocate the memory that a hangmane_context used.
void hangman_destroy_context(hangman_context * hm){
	free(hm->to_free_);
	free(hm->disp_);
}

//Returns "success" if the player solved the problem or "failed" the player could not solved the problem.
result play_hangman(hangman_context* hm){
	char input;
	while(hangman_get_remaining(hm) > 0){
		if(hangman_is_clear(hm)){
			printf("\n%s!!!\n\n", hangman_get_answer(hm));
			return succeeded;
		}

		printf("\nremaining: %d\n", hangman_get_remaining(hm));

		printf("used: ");
		hangman_print_used(hm);

		printf("\n%s\n", hangman_get_disp(hm));

		printf("input: ");
		input = getChar();
		printf("\n");
```

```
            hangman_check_alphabet(hm, input);
        }

        return failed;
}

//Contorls hangman game.
void ctrl_hangman(char const* file_name){
        hangman_context hm;
        char tudukeru = 'y';
        int success_count = 0;
        int play_count = 0;

        string_list_node* failed_problems;
        string_list_create(&failed_problems);


        //this pointer does not have ownership
        string_list_node* ite = failed_problems;

        while(tudukeru != 'n'){
                play_count++;

                hangman_init(&hm);
                if(hangman_set_problem(&hm, file_name) == failed){
                        string_list_destroy(failed_problems);
                        return;
                }

                if(play_hangman(&hm) == succeeded){
```

```
                success_count++;
        }else{
                ite = string_list_add(ite, hangman_get_answer(&hm));
                ite = string_list_add(ite, hangman_get_nihongo(&hm));
        }

        hangman_destroy_context(&hm);

        printf("continue? input 'n': no, another: yes: ");
        tudukeru = getChar();
    }
    printf("\n\n-------result---------\n");
    printf("seikairistu:%d/%d(%3lf)\n\n", success_count, play_count, (double)success_count / play_count);

    print_failed_problems(failed_problems);
    string_list_destroy(failed_problems);
}


int main(int argc, char *argv[])
{
    srand(time(NULL));
    if(argc == 2)
        ctrl_hangman(argv[1]);
    else
        ctrl_hangman("./toeic1500.dat");
}
```