```
#include<stdio.h>
#include<stdlib.h>
#include<assert.h>
#include<unistd.h>
#include <string.h>
#define TAILLEMAX 2000

struct rule {
    int cur_state;
    int symbol;
    int new_symbol;
    int direction;
    int new_state;
};
typedef struct rule rule;

struct vect_rule {
    int nb_elem;
    rule *p;
};
typedef struct vect_rule vect_rule;

struct vect_tape{
    int nb_elem;
    char *p;
};
typedef struct vect_tape vect_tape;


vect_tape init (char * file_tape){
    FILE * file;
    int i, taille;
    char init_tape[TAILLEMAX], c;
    vect_tape output_tape;

    memset(init_tape, 2, TAILLEMAX);
    file = fopen(file_tape, "r");
    if (file == NULL){
        printf("Error loading the file : %s. Either it was
            ↪ missplled or it does not exist.\n", file_tape);
        exit(0);
    }else {
        fseek(file, 0L, SEEK_END);
        taille =ftell(file);
        if (taille == 0){
            printf("This tape file is empty, aborting...\n");
            exit(0);
        }
        fseek(file, 0L, SEEK_SET);
        for (i =0; i < taille; i ++){
            fscanf(file, "%c", &c);
            fseek(file, 0L, SEEK_CUR);
            init_tape[i] = atoi(&c);
        }
        output_tape.nb_elem = i+1;
        output_tape.p = malloc(output_tape.nb_elem * sizeof(
            ↪ char));
        for(i = 0; i < output_tape.nb_elem; i++){
            output_tape.p[i+5] = init_tape[i];
        }
    }
    fclose(file);
    return output_tape;
}

vect_rule rule_generator (char * file_rule){
    int line_number = 0;
    int ligne;
    char tmp;
    vect_rule output_rules;
    FILE * file;
    file = fopen(file_rule, "r");

    for(tmp = getc(file); tmp != EOF; tmp = getc(file)){
        if ( tmp == '\n')
            line_number++;
    }
    output_rules.p = malloc(line_number*sizeof(rule));
    assert(output_rules.p);
    output_rules.nb_elem = line_number;
    fseek(file, 0, SEEK_SET);
    for (ligne = 0; ligne < line_number; ligne++ ){
        fscanf(file, "%d %d %d %d %d", &output_rules.p[ligne].
            ↪ cur_state, &output_rules.p[ligne].symbol, &
            ↪ output_rules.p[ligne].new_symbol, &output_rules.p
            ↪ [ligne].direction, &output_rules.p[ligne].
            ↪ new_state);
    }
    fclose(file);
    return output_rules;
}


vect_tape size_increase(vect_tape init_tape){
    int i;
    vect_tape output;

    output.nb_elem = TAILLEMAX*2;
    output.p = malloc(output.nb_elem * sizeof(char));
    for( i = 0; i<(TAILLEMAX *2); i++){
        output.p[i] = init_tape.p[i];
    }
    free(init_tape.p);
    return output;
}



int turing_machine (vect_tape init_tape, vect_rule rule_list,
    ↪ int cur_state, int verbose){
    int head_pos, i, n;
    char rule_found;
    head_pos = 5;
    while(1){

        if(head_pos == TAILLEMAX)
            init_tape = size_increase(init_tape);

        rule_found = 0;
        for (i = 0; i < rule_list.nb_elem; i++){
            if (cur_state == rule_list.p[i].cur_state &&
                ↪ init_tape.p[head_pos] == rule_list.p[i].symbol
                ↪ ){
```

```
                rule_found = 1;
                init_tape.p[head_pos] = rule_list.p[i].
                    ↪ new_symbol;
                if (rule_list.p[i].direction)
                    head_pos++;
                else
                    head_pos--;
                cur_state = rule_list.p[i].new_state;
                break;
            }
        }
        if(!rule_found){
            printf("\nNo rule found for the current state : %d ,
                ↪  job is done.", cur_state);
            printf("\nExited with :\n");
            for(n = 5; n > 0; n--){
                if (init_tape.p[n-1] == 1)
                    printf("%d ", init_tape.p[n-1]);
            }
            for (n = 0; n < init_tape.nb_elem-1; n ++){
                printf("%d ", init_tape.p[n+5]);
            };
            printf("\n");
            return 0;
        }
        if (verbose == 1){
            printf("state is : %d. Head is at position %d \n",
                ↪ cur_state, head_pos);
            printf("current tape is : ");
            for(n = 5; n > 0; n--){
                if (init_tape.p[n-1] == 1)
                    printf("%d ", init_tape.p[n-1]);
            }
            for (n = 0; n < init_tape.nb_elem-1; n ++){
                printf("%d ", init_tape.p[n+5]);
            };
            printf("\n");
        }
    }
}
```

```
int main (int argc, char *argv[]){
    int verbose, n;
    vect_tape init_tape;
    vect_rule rule_list;

    if(argv[4] && !strcmp(argv[4], "-v"))
        verbose = 1;
    else{
        printf("Note : call with -v for detailed processing.\n\
            ↪ n");
        verbose = 0;
    }
    if (argc < 4|| argc > 5 || !strcmp(argv[1], "-help")){
        printf("Usage : %s <tape_file> <rule_file> <
            ↪ initial_state> <-v> (last argument is optional
            ↪ and enables verbose mode)\n\nRule pattern must be
            ↪  : current_state found_symbol new_symbol
            ↪ movement_direction(where 0 is left and 1 is right
            ↪ ) new_state WITH SPACES. \nTape pattern must be a
            ↪  a chain of boolean numbers not separated by
            ↪ anything. The \'2\' symbol will represent blank
            ↪ spaces, rules must be set accordingly. \n\nHead
            ↪ starting position is at the begining of tape. The
            ↪  machine will halt anytime it detects a symbol
            ↪ for which it doesn't know any rule to apply.\n",
            ↪ argv[0]);
        exit(0);
    }
    init_tape = init(argv[1]);
    printf("initial call done with : ");
    for (n = 0; n <= init_tape.nb_elem-2; n ++){
        printf("%d ", init_tape.p[n+5]);
    };
    printf("\n");
    rule_list = rule_generator(argv[2]);
    printf("Processing...\n\n");
    turing_machine(init_tape, rule_list, atoi(argv[3]), verbose
        ↪ );
    return 0;
}
```