~/Documents/Saint Louis/TIPE/TIPE2/machineDeTuringRoutiers.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

struct stat st = {0};

//STRUCTURE TABLEAU DYNAMIQUE ENTIER

struct Tape{
    int size;
    int* tab;
    int blank;
};

typedef struct Tape Tape;

Tape* createTape(int blank){
    Tape* tabdynTape = malloc(sizeof(Tape));
    tabdynTape->blank = blank;
    tabdynTape->size = 16;
    tabdynTape->tab = malloc(sizeof(int)*16);
    return(tabdynTape);
}



//STRUCTURE TABLEAU DYNAMIQUE STRING

struct dynamicStringTable{
    int size;
    char** tab;
    int numbOfElem;
```

```
};

typedef struct dynamicStringTable dynamicStringTable;

dynamicStringTable* createdynamicStringTable(){
    dynamicStringTable* tabdyn = malloc(sizeof(dynamicStringTable));
    tabdyn->size = 16;
    tabdyn->numbOfElem = 0;
    tabdyn->tab = malloc(sizeof(int)*16);
    return(tabdyn);
}

void expanddynamicStringTable(dynamicStringTable* tabdyn){
    tabdyn->size *= 2;
    tabdyn->tab = realloc(tabdyn->tab, sizeof(int)*tabdyn->size);
}


// STRUCTURE NEXTSTEP POUR FAIRE UN TUPLE (nextState, nextDirection, changedChar) = int newState, (-1|0|1), (int newChar)

struct nextStep{
    int nextState;
    int nextDirection;
    int nextChar;
    bool defined;
};

typedef struct nextStep nextStep;

// STRUCTURE TEMPTRANSITIONS POUR PLUS FACILEMENT INITIALISER UNE MACHINE DE TURING

struct tempTransitions{
    int currentState;
    int currentChar;
    int nextState;
    int nextDirection;
    int nextChar;
};

typedef struct tempTransitions tempTransitions;
```

```

// STRUCTURE DE MACHINE DE TURING

struct turing{
    // pour avoir tout du même type on utilise des int pour les char
    Tape* tape;
    // possibilité de les transformer avec
    char** charConverter;
    // état sous la forme d'un int, aussi possibilité de les transformer avec
    int currentState;
    char** stateConverter;
    // transitions sous la forme d'un tableau-ception ([currentState][readchar]=nextStep)
    nextStep** transitions;
    // états finals dans un boolswitch
    bool* finalStates;
    // bool qui dit si la machine de turing s'est arrêté
    bool halted;
    // l'indice de la position de la tête (peut-être qu'on changera en un pointeur pour l'arithmétique pointeur)
    int headPosition;
    // symbol blank qui sera celui qui remplit la machine dans les infinits
    int blankChar;
};

typedef struct turing turing;



// ALGOS ÉLÉMENTAIRES POUR UTILISER LA STRUCTURE MACHINE DE TURING


Tape* makeEmptyTape(int blank) {
    Tape* tape = createTape(blank);
    for (int i = 0; i<16; i++) {
        tape->tab[i] = blank;
    }
    return(tape);
};


void expandTapeRight(turing* turingMachine){
```

```
    int oldSize = turingMachine->tape->size;
    turingMachine->tape->size *= 2;
    turingMachine->tape->tab = realloc(turingMachine->tape->tab, sizeof(int)*turingMachine->tape->size);
    for (int i = oldSize; i < turingMachine->tape->size; i++) {
        turingMachine->tape->tab[i] = turingMachine->tape->blank;
    }
}

void expandTapeLeft(turing* turingMachine){
    int oldSize = turingMachine->tape->size;
    turingMachine->tape->size *=2;
    int* newTab = malloc(sizeof(int)*turingMachine->tape->size);
    for (int i = 0; i<oldSize; i++) {
        newTab[i] = turingMachine->tape->blank;
    }
    for (int i = 0; i<oldSize; i++) {
        newTab[i+oldSize] = turingMachine->tape->tab[i];
    }
    turingMachine->headPosition += oldSize;
    free(turingMachine->tape->tab);
    turingMachine->tape->tab = newTab;
}

void modifyTape(turing* turingMachine, int newChar, int newCharIndex){
    while (newCharIndex >= turingMachine->tape->size) {
        expandTapeRight(turingMachine);
    }
    turingMachine->tape->tab[newCharIndex] = newChar;
    while (newCharIndex < 0) {
        int oldSize = turingMachine->tape->size;
        expandTapeLeft(turingMachine);
        modifyTape(turingMachine, newChar, newCharIndex + oldSize);
    }
}

void customizeStartingTape(turing* turingMachine, int* startingTab, int startingTabLength) {
    for (int i = 0; i<startingTabLength; i++) {
        modifyTape(turingMachine, startingTab[i], i);
    }
}
```

```
157
158 turing* makeTuringMachine(/*int* alphabet,*/ int alphabetSize, int blank, /*on ne précise pas les symboles permis d'appar
159     //FOR NOW BOTH ALPHABET AND STATES ARE INT, CONVERSION WILL COME LATER
160
161     turing* turingMachine = malloc(sizeof(turing)); //allocate turing structure
162
163
164
165
166     // initialise an empty tape
167     turingMachine->tape = makeEmptyTape(blank);
168     customizeStartingTape(turingMachine, startingTape, startingTapeLength);
169
170
171     // set initial state
172     turingMachine->currentState = starterState;
173
174
175
176     turingMachine->transitions = malloc(sizeof(nextStep*)*stateSize);
177     for (int i = 0; i<stateSize; i++) {
178         turingMachine->transitions[i] = malloc(sizeof(nextStep)*alphabetSize);
179         for (int j = 0; j<alphabetSize;j++) {
180             turingMachine->transitions[i][j].defined = false;
181         }
182     }   // allocate transition table
183
184     // fill transition table
185     for (int i = 0; i<transitionsSize; i++) {
186
187         turingMachine->transitions[transitions[i].currentState][transitions[i].currentChar].nextState =  transitions[i].n
188         turingMachine->transitions[transitions[i].currentState][transitions[i].currentChar].nextDirection =  transitions[
189         turingMachine->transitions[transitions[i].currentState][transitions[i].currentChar].nextChar =  transitions[i].ne
190         turingMachine->transitions[transitions[i].currentState][transitions[i].currentChar].defined =  true;
191     }
192
193
194     // make bool switch for final states
195     turingMachine->finalStates = malloc(sizeof(bool)*stateSize);
196     for (int i = 0; i<stateSize; i++) {
```

```
197         turingMachine->finalStates[i] = false;
198     }
199     for (int i = 0; i<finalStatesSize; i++) {
200         turingMachine->finalStates[finalStates[i]] = true;
201     }
202 
203 
204     // set halted to false
205     turingMachine->halted = false;
206 
207 
208     // address for starting point (as given in input)
209     turingMachine->headPosition = headPosition;  // version pointeur : &(turingMachine->tape->tab[headPosition]);
210 
211 
212     // set blank character
213     turingMachine->blankChar = blank;
214 
215     // make int to char converters
216     /*
217     turingMachine->stateConverter = malloc(sizeof(char*)*stateSize);
218     turingMachine->stateConverter[0] = starterState;
219     int adjust = 1;
220     for (int i = 0; i<stateSize-1; i++) {
221         if (strcmp(starterState, states[i-adjust]) == 0) {
222             adjust = 0;
223         } else {
224         turingMachine->stateConverter[i+adjust] = states[i];
225         }
226     }
227 
228 
229     turingMachine->charConverter = malloc(sizeof(char*)*alphabetSize);
230     turingMachine->charConverter[0] = blank;
231     for (int i = 0; i<alphabetSize-1; i++) {
232         if (strcmp(blank, alphabet[i-adjust]) == 0) {
233             adjust = 0;
234         } else {
235         turingMachine->charConverter[i+adjust] = alphabet[i];
236         }
```

```
    }
    */


    return(turingMachine);
};

void printMachine(turing* turingMachine){
    int screenWidth = 75; // to fill with ------
    printf("Current State : %d\n", turingMachine->currentState);
    for (int i = 0; i < screenWidth; i++) {
        printf("_");
    }
    printf("\n");
    for (int j = 0; j<turingMachine->tape->size; j++) {
        if (turingMachine->headPosition == j) {
            printf("|");
            printf("\033[0;31m");
            printf("%d", turingMachine->tape->tab[j]);
            printf("\033[0m");
        } else {
            printf("|%d", turingMachine->tape->tab[j]);
        }
    }
    printf("|\n");
    for (int i = 0; i < screenWidth; i++) {
        printf("¯");
    }
    printf("\n");
}


void oneStep(turing* turingMachine) {
    // make sure machine is not halted
    if (!turingMachine->halted) { // analyze current + expected transition
        int readCharacter = turingMachine->tape->tab[turingMachine->headPosition];
        int currentState = turingMachine->currentState;
        nextStep next = turingMachine->transitions[currentState][readCharacter];
        if (!next.defined){ // check if transition defined
            turingMachine->halted = true;
```

```
        } else {
            turingMachine->currentState = next.nextState;  // update state
            modifyTape(turingMachine, next.nextChar, turingMachine->headPosition); // update tape
            turingMachine->headPosition += next.nextDirection; // update head
            // expand tape if necessary based on head
            if (turingMachine->headPosition == -1) {
                expandTapeLeft(turingMachine);
            }
            if (turingMachine->headPosition >= turingMachine->tape->size) {
                expandTapeRight(turingMachine);
            }

            // halt if in final state
            if (turingMachine->finalStates[turingMachine->currentState]) {
                turingMachine->halted = true;
            }

        }

    }

}

void runMachine(turing* turingMachine) {
    printMachine(turingMachine);
    while (!turingMachine->halted) {
        oneStep(turingMachine);
        printMachine(turingMachine);
    }
}

struct printParameters {
    int* fileCount;
    char* OutputDir;
    int leftBorderChar;
    int rightBorderChar;
    int printState;
};

typedef struct printParameters printParameters;
```

```
317
318
319
320 void writeMachineToFile(turing* turingMachine, printParameters* printParams) {
321     printParams->fileCount[0]++;
322     //FILE* ptr = fopen(("/%s/input/%s%d.txt", OutputDir, OutputDir, fileCount),"w");
323     /*
324     char bufmake[0x200];
325     snprintf(bufmake, sizeof(bufmake), "./%s/Input/", printParams->OutputDir);
326     if (stat(bufmake, &st) == -1) {
327         mkdir(bufmake, 0700);
328     }
329     */
330     char buf[0x200];
331     snprintf(buf, sizeof(buf), "./%s/Input/%s%d.txt", printParams->OutputDir, printParams->OutputDir, printParams->fileCo
332     FILE* ptr = fopen(buf, "w");
333     int customHead = 0;
334     while (turingMachine->tape->tab[customHead] != printParams->leftBorderChar) {
335         customHead++;
336     }
337     customHead++;
338     while (turingMachine->tape->tab[customHead] != printParams->rightBorderChar) {
339         char buf1[0x100];
340         snprintf(buf1, sizeof(buf1), "%d\n", turingMachine->tape->tab[customHead]);
341         fputs(buf1, ptr);
342         customHead++;
343     }
344     fputs("end\n", ptr);
345     fclose(ptr);
346 }
347
348 void runMachineWithOptionalPrint(turing* turingMachine, printParameters* printParams) {
349     char bufmake[0x200];
350     snprintf(bufmake, sizeof(bufmake), "./%s", printParams->OutputDir);
351     mkdir(bufmake, 0700);
352     char bufmake1[0x200];
353     snprintf(bufmake1, sizeof(bufmake1), "./%s/Input/", printParams->OutputDir);
354     mkdir(bufmake1, 0700);
355     writeMachineToFile(turingMachine, printParams);
356     while (!turingMachine->halted) {
```

```
        oneStep(turingMachine);
        if (turingMachine->currentState == printParams->printState) {
            writeMachineToFile(turingMachine, printParams);
        }
    }
    char command0[400];
    snprintf(command0, sizeof(command0), "cp ./Image-Converter.py ./%s/Image-Converter.py", printParams->OutputDir);
    system(command0);
    char command[400];
    snprintf(command, sizeof(command), "python3 ./%s/Image-Converter.py ./%s/Input", printParams->OutputDir, printParams-
    system(command);
}

void runMachineWithOptionalPrintSlowy(turing* turingMachine, printParameters* printParams) {
    char bufmake[0x200];
    snprintf(bufmake, sizeof(bufmake), "./%s", printParams->OutputDir);
    mkdir(bufmake, 0700);
    char bufmake1[0x200];
    snprintf(bufmake1, sizeof(bufmake1), "./%s/Input/", printParams->OutputDir);
    mkdir(bufmake1, 0700);
    writeMachineToFile(turingMachine, printParams);
    while (!turingMachine->halted) {
        oneStep(turingMachine);
        if (turingMachine->currentState == printParams->printState) {
            writeMachineToFile(turingMachine, printParams);
        }
    }
    char command0[400];
    snprintf(command0, sizeof(command0), "cp ./Image-ConverterSlowy.py ./%s/Image-ConverterSlowy.py", printParams->Output
    system(command0);
    char command[400];
    snprintf(command, sizeof(command), "python3 ./%s/Image-ConverterSlowy.py ./%s/Input", printParams->OutputDir, printPa
    system(command);
}



void runMachineWithOptionalPrint2D(turing* turingMachine, printParameters* printParams, int howManyPrintsMax) {
    //le char 0 est le blank
    //le char 1 et 2 sont les print
```

```
    //le char 3 est pour la barrière du centre
    //les char du centre sont 4,5,6 pour vide right et down
    //le char right est 7
    //le char down est 8
    char bufmake[0x200];
    snprintf(bufmake, sizeof(bufmake), "./%s", printParams->OutputDir);
    mkdir(bufmake, 0700);
    char bufmake1[0x200];
    snprintf(bufmake1, sizeof(bufmake1), "./%s/Input/", printParams->OutputDir);
    mkdir(bufmake1, 0700);
    writeMachineToFile(turingMachine, printParams);
    int prints = 1;
    while (!turingMachine->halted && prints < howManyPrintsMax) {
        oneStep(turingMachine);
        if (turingMachine->currentState == printParams->printState) {
            writeMachineToFile(turingMachine, printParams);
            prints++;
            //optional
            printMachine(turingMachine);
        }
    }
    char command0[400];
    snprintf(command0, sizeof(command0), "cp ./Image-Converter-2D.py ./%s/Image-Converter-2D.py", printParams->OutputDir)
    system(command0);
    char command[400];
    snprintf(command, sizeof(command), "python3 ./%s/Image-Converter-2D.py ./%s/Input", printParams->OutputDir, printPara
    system(command);
}


int makeASpeedyCar(int* alphabetSize, int blank, int* stateSize, int emptyHandState, int carSpeed, int* transitionCount,
    // 2 solutions, retirer n voitures puis faire apparaitre n voitures, ou retirer une voiture 1 par 1
    //j'ai fait option 2

    // goingLeft
    transitions[*transitionCount].currentState = goingLeftState; transitions[*transitionCount].currentChar = *alphabetSiz

    //pick up the tail
    transitions[*transitionCount].currentState = emptyHandState; transitions[*transitionCount].currentChar = *alphabetSiz
    //for every amount of back and forths left to do
```

```
    for (int i = 0; i < carSpeed-1; i++) {
        //propagate forward
        transitions[*transitionCount].currentState = (*stateSize)-1; transitions[*transitionCount].currentChar = *alphabe

        //if printChar end simulation
        transitions[*transitionCount].currentState = (*stateSize)-1; transitions[*transitionCount].currentChar = rightPri

        //place
        transitions[*transitionCount].currentState = (*stateSize)-1; transitions[*transitionCount].currentChar = blank; t
        //propagate backwards
        transitions[*transitionCount].currentState = (*stateSize)-1; transitions[*transitionCount].currentChar = *alphabe
        //go forward to tail
        transitions[*transitionCount].currentState = (*stateSize)-1; transitions[*transitionCount].currentChar = blank; t
        //pick up car then loop
        transitions[*transitionCount].currentState = (*stateSize)-1; transitions[*transitionCount].currentChar = *alphabe
    }
    //propagate a final time
    transitions[*transitionCount].currentState = (*stateSize)-1; transitions[*transitionCount].currentChar = *alphabetSiz
    //place and start going forwards again
    transitions[*transitionCount].currentState = (*stateSize)-1; transitions[*transitionCount].currentChar = blank; trans
    return((*alphabetSize)++);
}

void placeASpeedyCar(int carSize, int carStartIndex, int carSymbol, int* tape) {
    for (int i = carStartIndex; i < carStartIndex + carSize; i++) {
        tape[i] = carSymbol;
    }
}

void addASpeedyCar(int* alphabetSize, int blank, int* stateSize, int emptyHandState, int carSpeed, int* transitionCount,
    placeASpeedyCar(carSize, carStartIndex, makeASpeedyCar(alphabetSize, blank, stateSize, emptyHandState, carSpeed, tran
}



//si jamais une voiture est à moins de deux fois la distance qu'on va avancer d'une autre voiture, prendre la vitesse de
int instantCrashAvoid(int* alphabetSize, int blank, int* stateSize, int emptyHandState, int offset, int maxSpeed, int* tr
    (*alphabetSize) += maxSpeed;
    //state i*2+offset = contamination associé à vi cycle sur 0
```

```
    for(int i = 1; i<= maxSpeed; i++) {
        //printf("%d\n", *transitionCount);
        //left until car
        transitions[*transitionCount].currentState = i*2+offset; transitions[*transitionCount].currentChar = blank; trans
        for (int j = i+1; j<= maxSpeed; j++){//voiture de derrière plus vite
        //if car start contaminating
        transitions[*transitionCount].currentState = i*2+offset; transitions[*transitionCount].currentChar = j; transitio
        //continue contaminating
        transitions[*transitionCount].currentState = i*2+1+offset; transitions[*transitionCount].currentChar = j; transit
        //start going Right again
        //printf("%d lol\n", emptyHandState);
        transitions[*transitionCount].currentState = i*2+1+offset; transitions[*transitionCount].currentChar = blank; tra
        }

    }
    (*stateSize) += 2*maxSpeed+1;

    //printf("%d XD\n", *stateSize);
    for(int i = 1; i<= maxSpeed; i++) {
        //printf("%d\n", *transitionCount);
        //start going to head
        transitions[*transitionCount].currentState = emptyHandState; transitions[*transitionCount].currentChar = i; trans
        //keep going to head
        transitions[*transitionCount].currentState = *stateSize - 1; transitions[*transitionCount].currentChar = i; trans
        //if at end

        //printf("%d\n", *stateSize);
        //printf("%d\n", i);
        for (int j = 0; j<i; j++) {
            transitions[*transitionCount].currentState = *stateSize - 1; transitions[*transitionCount].currentChar = blan
            for (int k = 1; k<i; k++){
               transitions[*transitionCount].currentState = *stateSize - 1; transitions[*transitionCount].currentChar = k
            }
        }
        transitions[*transitionCount].currentState = *stateSize - 1; transitions[*transitionCount].currentChar = blank; t
        transitions[*transitionCount].currentState = *stateSize - 1; transitions[*transitionCount].currentChar = blank; t
        //when see car
        transitions[*transitionCount].currentState = *stateSize - 1; transitions[*transitionCount].currentChar = i; trans
        transitions[*transitionCount].currentState = *stateSize - 1; transitions[*transitionCount].currentChar = i; trans
        //when see empty
```

```
517         transitions[*transitionCount].currentState = *stateSize - 1; transitions[*transitionCount].currentChar = blank; t
518
519         // goingLeft
520         transitions[*transitionCount].currentState = goingLeftState; transitions[*transitionCount].currentChar = i; trans
521
522         //pick up the tail
523         transitions[*transitionCount].currentState = *stateSize - 1; transitions[*transitionCount].currentChar = i; trans
524         //for every amount of back and forths left to do
525         for (int j = 0; j < i-1; j++) {
526             //propagate forward
527             transitions[*transitionCount].currentState = (*stateSize)-1; transitions[*transitionCount].currentChar = i; t
528
529             //if printChar end simulation
530             transitions[*transitionCount].currentState = (*stateSize)-1; transitions[*transitionCount].currentChar = righ
531
532             //place
533             transitions[*transitionCount].currentState = (*stateSize)-1; transitions[*transitionCount].currentChar = blan
534             //propagate backwards
535             transitions[*transitionCount].currentState = (*stateSize)-1; transitions[*transitionCount].currentChar = i; t
536             //go forward to tail
537             transitions[*transitionCount].currentState = (*stateSize)-1; transitions[*transitionCount].currentChar = blan
538             //pick up car then loop
539             transitions[*transitionCount].currentState = (*stateSize)-1; transitions[*transitionCount].currentChar = i; t
540         }
541         //propagate a final time
542         transitions[*transitionCount].currentState = (*stateSize)-1; transitions[*transitionCount].currentChar = i; trans
543         //place and start going forwards again
544         transitions[*transitionCount].currentState = (*stateSize)-1; transitions[*transitionCount].currentChar = blank; t
545
546
547     }
548     return(*alphabetSize);
549 }
550
551
552
553
554
555 int main()
556 {
```

```
557
558
559 //-----------------------------------------------------------------------------------------------------------------
560
561 // 3 STATE BUSY BEAVER TEST
562
563 /*
564
565 int BB3alphabetSize = 2;
566 int BB3blank = 0;
567 int BB3stateSize = 4; // includes HALT
568 int BB3starterState = 0;
569 int BB3finalStates[1] = {3};
570 int BB3finalStatesSize = 1;
571 tempTransitions* BB3transitions = malloc(sizeof(tempTransitions)*6);
572 BB3transitions[0].currentState = 0; BB3transitions[0].currentChar = 0; BB3transitions[0].nextState = 1; BB3transitions[0]
573 BB3transitions[1].currentState = 0; BB3transitions[1].currentChar = 1; BB3transitions[1].nextState = 2; BB3transitions[1]
574 BB3transitions[2].currentState = 1; BB3transitions[2].currentChar = 0; BB3transitions[2].nextState = 0; BB3transitions[2]
575 BB3transitions[3].currentState = 1; BB3transitions[3].currentChar = 1; BB3transitions[3].nextState = 1; BB3transitions[3]
576 BB3transitions[4].currentState = 2; BB3transitions[4].currentChar = 0; BB3transitions[4].nextState = 1; BB3transitions[4]
577 BB3transitions[5].currentState = 2; BB3transitions[5].currentChar = 1; BB3transitions[5].nextState = 3; BB3transitions[5]
578 int BB3transitionsSize = 6;
579 int BB3startingTape[1] = {0};
580 int BB3startingTapeLength = 1;
581 int BB3headPosition = 0;
582
583 turing* BB3machine = makeTuringMachine(BB3alphabetSize, BB3blank, BB3stateSize, BB3starterState, BB3finalStates, BB3final
584 runMachine(BB3machine);
585
586 */
587
588
589 //-----------------------------------------------------------------------------------------------------------------
590
591 // PROBLÈME 1
592 /*
593 int PB1alphabetSize = 3;
594 int PB1blank = 0;
595 int PB1stateSize = 3; // includes HALT
596 int PB1starterState = 0;
```

```
int PB1finalStates[1] = {2};
int PB1finalStatesSize = 1;
tempTransitions* PB1transitions = malloc(sizeof(tempTransitions)*6);
PB1transitions[0].currentState = 0; PB1transitions[0].currentChar = 0; PB1transitions[0].nextState = 0; PB1transitions[0]
PB1transitions[1].currentState = 0; PB1transitions[1].currentChar = 1; PB1transitions[1].nextState = 1; PB1transitions[1]
PB1transitions[2].currentState = 0; PB1transitions[2].currentChar = 2; PB1transitions[2].nextState = 2; PB1transitions[2]
PB1transitions[3].currentState = 1; PB1transitions[3].currentChar = 0; PB1transitions[3].nextState = 0; PB1transitions[3]
PB1transitions[4].currentState = 1; PB1transitions[4].currentChar = 1; PB1transitions[4].nextState = 1; PB1transitions[4]
PB1transitions[5].currentState = 1; PB1transitions[5].currentChar = 2; PB1transitions[5].nextState = 2; PB1transitions[5]
int PB1transitionsSize = 6;
// en fonction des tests qu'on veut
int PB1startingTape[30] = {0,1,0,1,1,1,1,1,0,0,0,1,1,1,0,0,0,0,1,1,0,0,1,0,1,0,1,0,0,2};
int PB1startingTapeLength = 30;
int PB1headPosition = 0;

turing* PB1machine = makeTuringMachine(PB1alphabetSize, PB1blank, PB1stateSize, PB1starterState, PB1finalStates, PB1final
runMachine(PB1machine);
*/


//-------------------------------------------------------------------------------------------------------------

//FANCY PRINT TEST
/*
int FPTestalphabetSize = 4; //0, 1 | print : 2 , 3
int FPTestblank = 0;
int FPTeststateSize = 5; //handEmpty, handFull, goingBackLeft, print, HALT
int FPTeststarterState = 0;
int FPTestfinalStates[1] = {4};
int FPTestfinalStatesSize = 1;
tempTransitions* FPTesttransitions = malloc(sizeof(tempTransitions)*10);
FPTesttransitions[0].currentState = 0; FPTesttransitions[0].currentChar = 0; FPTesttransitions[0].nextState = 0; FPTesttr
FPTesttransitions[1].currentState = 0; FPTesttransitions[1].currentChar = 1; FPTesttransitions[1].nextState = 1; FPTesttr
FPTesttransitions[2].currentState = 0; FPTesttransitions[2].currentChar = 3; FPTesttransitions[2].nextState = 3; FPTesttr
FPTesttransitions[3].currentState = 3; FPTesttransitions[3].currentChar = 3; FPTesttransitions[3].nextState = 2; FPTesttr
FPTesttransitions[4].currentState = 2; FPTesttransitions[4].currentChar = 0; FPTesttransitions[4].nextState = 2; FPTesttr
FPTesttransitions[5].currentState = 2; FPTesttransitions[5].currentChar = 1; FPTesttransitions[5].nextState = 2; FPTesttr
FPTesttransitions[6].currentState = 2; FPTesttransitions[6].currentChar = 2; FPTesttransitions[6].nextState = 0; FPTesttr
FPTesttransitions[7].currentState = 1; FPTesttransitions[7].currentChar = 0; FPTesttransitions[7].nextState = 0; FPTesttr
FPTesttransitions[8].currentState = 1; FPTesttransitions[8].currentChar = 1; FPTesttransitions[8].nextState = 1; FPTesttr
```

```
FPTesttransitions[9].currentState = 1; FPTesttransitions[9].currentChar = 3; FPTesttransitions[9].nextState = 4; FPTesttr

int FPTesttransitionsSize = 10;
int* FPTeststartingTape = malloc(sizeof(int)*200);
int FPTeststartingTapeLength = 200;
FPTeststartingTape[0] = 2;
FPTeststartingTape[199] = 3;
for (int i = 1; i<50; i++) {
    FPTeststartingTape[i] = 0;
}
for (int i = 50; i<150; i++) {
    FPTeststartingTape[i] = 1;
}
for (int i = 150; i<199; i++) {
    FPTeststartingTape[i] = 0;
}
int FPTestheadPosition = 1;

turing* FPTestmachine = makeTuringMachine(FPTestalphabetSize, FPTestblank, FPTeststateSize, FPTeststarterState, FPTestfin

// Run FPTest
//runMachine(FPTestmachine);



printParameters* FPTestprintParams = malloc(sizeof(printParameters));
int FPTestfileCount = 10000;
FPTestprintParams->fileCount = &FPTestfileCount;
FPTestprintParams->OutputDir = "FPTest";
FPTestprintParams->leftBorderChar = 2;
FPTestprintParams->rightBorderChar = 3;
FPTestprintParams->printState = 3;



runMachineWithOptionalPrint(FPTestmachine, FPTestprintParams);
//writeMachineToFile(FPTestmachine, FPTestprintParams);

printf("finished\n");

```

```
*/


//-------------------------------------------------------------------------------------------------------------

// PROBLÈME 2
///*
int speedyCarsTestalphabetSize = 3; //0 | print : 1 , 2
int speedyCarsTestblank = 0;
int speedyCarsTeststateSize = 4; //handEmpty, goingBackLeft, print, HALT
int speedyCarsTeststarterState = 0;
int speedyCarsTestfinalStates[1] = {3};
int speedyCarsTestfinalStatesSize = 1;
tempTransitions* speedyCarsTesttransitions = malloc(sizeof(tempTransitions)*1000);
speedyCarsTesttransitions[0].currentState = 0; speedyCarsTesttransitions[0].currentChar = 0; speedyCarsTesttransitions[0]
speedyCarsTesttransitions[1].currentState = 0; speedyCarsTesttransitions[1].currentChar = 2; speedyCarsTesttransitions[1]
speedyCarsTesttransitions[2].currentState = 2; speedyCarsTesttransitions[2].currentChar = 2; speedyCarsTesttransitions[2]
speedyCarsTesttransitions[3].currentState = 1; speedyCarsTesttransitions[3].currentChar = 0; speedyCarsTesttransitions[3]
speedyCarsTesttransitions[4].currentState = 1; speedyCarsTesttransitions[4].currentChar = 1; speedyCarsTesttransitions[4]

int speedyCarsTesttransitionsSize = 5;
int* speedyCarsTeststartingTape = malloc(sizeof(int)*3000);
int speedyCarsTeststartingTapeLength = 3000;
speedyCarsTeststartingTape[0] = 1;
for (int i = 1; i<3000; i++) {
    speedyCarsTeststartingTape[i] = 0;
}
for (int i = 2980; i<3000; i++) {
    speedyCarsTeststartingTape[i] = 2;
}


int speedyCarsTestheadPosition = 1;

printParameters* speedyCarsTestprintParams = malloc(sizeof(printParameters));
int speedyCarsTestfileCount = 10000;
speedyCarsTestprintParams->fileCount = &speedyCarsTestfileCount;
speedyCarsTestprintParams->leftBorderChar = 1;
speedyCarsTestprintParams->rightBorderChar = 2;
speedyCarsTestprintParams->printState = 2;
```

```
717
718
719
720
721 //two 100 length cars, starting 100 apart; speeds : first = 5 second = 3
722 /*
723 speedyCarsTestprintParams->OutputDir = "twoSpeedyCars";
724 addASpeedyCar(&speedyCarsTestalphabetSize, speedyCarsTestblank, &speedyCarsTeststateSize, 0, 5, &speedyCarsTesttransition
725 addASpeedyCar(&speedyCarsTestalphabetSize, speedyCarsTestblank, &speedyCarsTeststateSize, 0, 3, &speedyCarsTesttransition
726 */
727
728 //three 100 length cars, starting 200 apart; speeds : first = 12 second = 10 third = 8
729
730 speedyCarsTestprintParams->OutputDir = "threeSpeedyCarsCrashing";
731 addASpeedyCar(&speedyCarsTestalphabetSize, speedyCarsTestblank, &speedyCarsTeststateSize, 0, 12, &speedyCarsTesttransitio
732 addASpeedyCar(&speedyCarsTestalphabetSize, speedyCarsTestblank, &speedyCarsTeststateSize, 0, 10, &speedyCarsTesttransitio
733 addASpeedyCar(&speedyCarsTestalphabetSize, speedyCarsTestblank, &speedyCarsTeststateSize, 0, 8, &speedyCarsTesttransition
734
735
736
737 //three 100 length cars, starting 100 apart; speeds : first = 5 second = 10 third = 20
738 /*
739 speedyCarsTestprintParams->OutputDir = "threeSpeedyCarsDispersing";
740 addASpeedyCar(&speedyCarsTestalphabetSize, speedyCarsTestblank, &speedyCarsTeststateSize, 0, 7, &speedyCarsTesttransition
741 addASpeedyCar(&speedyCarsTestalphabetSize, speedyCarsTestblank, &speedyCarsTeststateSize, 0, 10, &speedyCarsTesttransitio
742 addASpeedyCar(&speedyCarsTestalphabetSize, speedyCarsTestblank, &speedyCarsTeststateSize, 0, 20, &speedyCarsTesttransitio
743 */
744
745 turing* speedyCarsTestmachine = makeTuringMachine(speedyCarsTestalphabetSize, speedyCarsTestblank, speedyCarsTeststateSiz
746
747
748
749
750 //runMachine(speedyCarsTestmachine);
751
752
753 runMachineWithOptionalPrint(speedyCarsTestmachine, speedyCarsTestprintParams);
754
755 //*/
756
```

```
757
758 //-----------------------------------------------------------------------------------------------------------------
759
760 // PROBLÈME 3
761 ///*
762 int slowyCarsTestalphabetSize = 3; //0 | print : 1 , 2
763 int slowyCarsTestblank = 0;
764 int slowyCarsTeststateSize = 4; //handEmpty, goingBackLeft, print, HALT
765 int slowyCarsTeststarterState = 0;
766 int slowyCarsTestfinalStates[1] = {3};
767 int slowyCarsTestfinalStatesSize = 1;
768 tempTransitions* slowyCarsTesttransitions = malloc(sizeof(tempTransitions)*100000);
769 slowyCarsTesttransitions[0].currentState = 0; slowyCarsTesttransitions[0].currentChar = 0; slowyCarsTesttransitions[0].ne
770 slowyCarsTesttransitions[1].currentState = 0; slowyCarsTesttransitions[1].currentChar = -2; slowyCarsTesttransitions[1].n
771 slowyCarsTesttransitions[2].currentState = 2; slowyCarsTesttransitions[2].currentChar = -2; slowyCarsTesttransitions[2].n
772 slowyCarsTesttransitions[3].currentState = 1; slowyCarsTesttransitions[3].currentChar = 0; slowyCarsTesttransitions[3].ne
773 slowyCarsTesttransitions[4].currentState = 1; slowyCarsTesttransitions[4].currentChar = -1; slowyCarsTesttransitions[4].n
774
775 int slowyCarsTesttransitionsSize = 5;
776 int* slowyCarsTeststartingTape = malloc(sizeof(int)*3000);
777
778 int slowyCarsTeststartingTapeLength = 3000;
779
780 int slowyCarsTestheadPosition = 1;
781
782 slowyCarsTeststartingTape[0] = -1;
783
784
785 for (int i = 1; i<3000; i++) {
786     slowyCarsTeststartingTape[i] = 0;
787 }
788
789
790
791 for (int i = 2980; i<3000; i++) {
792     slowyCarsTeststartingTape[i] = -2;
793 }
794
795
796
```

```
printParameters* slowyCarsTestprintParams = malloc(sizeof(printParameters));
int slowyCarsTestfileCount = 10000;
slowyCarsTestprintParams->fileCount = &slowyCarsTestfileCount;
slowyCarsTestprintParams->leftBorderChar = -1;
slowyCarsTestprintParams->rightBorderChar = -2;
slowyCarsTestprintParams->printState = 2;


slowyCarsTestprintParams->OutputDir = "threeSpeedyCarsSlowing";

instantCrashAvoid(&slowyCarsTestalphabetSize, slowyCarsTestblank, &slowyCarsTeststateSize, 0, slowyCarsTesttransitionsSiz

placeASpeedyCar(100, 100, 12, slowyCarsTeststartingTape);
placeASpeedyCar(100, 350, 10, slowyCarsTeststartingTape);
placeASpeedyCar(100, 550, 8, slowyCarsTeststartingTape);

turing* slowyCarsTestmachine = makeTuringMachine(slowyCarsTestalphabetSize*30, slowyCarsTestblank, slowyCarsTeststateSize

//runMachine(slowyCarsTestmachine);


//runMachineWithOptionalPrintSlowy(slowyCarsTestmachine, slowyCarsTestprintParams);



//-------------------------------------------------------------------------------------------------------

//PROBLEM 4

int PB4alphabetSize = 9; // 0 = empty, 1 = borderLeft, 2 = borderRight, 3 = borderCenter, 4 = centerEmpty, 5 = centerRigh
int PB4blank = 0;
int PB4stateSize = 23; // 0 = phase1.handEmpty, 1 = phase1.handFull, 2 = phase3.handEmpty, 3 = phase2.libérerCentre, 4 =
                       // 10 = phase5.handEmpty, 11 = phase5.handFull, 12 = phase7.handEmpty, 13 = phase6.libérerCentre, 1
                       // 20 = phase8.retour
                       // 21 = phase8.PrintState
                       // 22 = HaltState (not used)

//phase 1 : 0 = phase1.handEmpty, 1 = phase1.handFull, 7 = phase1.retourenarrière, 8 = phase1.repartirenavant
//phase 2 : 3 = phase2.libérerCentre, 4 = phase2.revenir-handfull, 6 = phase2.placercentre, 9 = phase2.revenir-handEmpty
//phase 3 : 2 = phase3.handEmpty, 5 = phase3.handFull
```

```
//phase 4 : instantané
//phase 5 : 10 = phase5.handEmpty, 11 = phase5.handFull, 17 = phase5.retourenarrière, 18 = phase5.repartirenavant
//phase 6 : 13 = phase6.libérerCentre, 14 = phase6.revenir-handfull, 16 = phase6.placercentre, 19 = phase6.revenir-handEm
//phase 7 : 12 = phase7.handEmpty, 15 = phase7.handFull
//phase 8 : 20 = phase8.retour, 21 = phase8.PrintState

int PB4starterState = 0;
int PB4finalStates[1] = {22}; // NEED TO DO IT
int PB4finalStatesSize = 1;
tempTransitions* PB4transitions = malloc(sizeof(tempTransitions)*89);
//phase 1 déplacer vers la droite
//phase 2 faire correspondre le centre si changement
//phase 3 finir le déplacement vers la droite
//phase 4 aller sur la bande verticale
//phase 5 déplacer vers le bas
//phase 6 faire correspondre le centre si changement
//phase 7 finir le déplacement vers le bas
//phase 8 revenir au début. Vinaver.exe

//PHASE 1 :
//si main vide
PB4transitions[0].currentState = 0; PB4transitions[0].currentChar = 0; PB4transitions[0].nextState = 0; PB4transitions[0]
PB4transitions[1].currentState = 0; PB4transitions[1].currentChar = 7; PB4transitions[1].nextState = 1; PB4transitions[1]
PB4transitions[2].currentState = 0; PB4transitions[2].currentChar = 6; PB4transitions[2].nextState = 2; PB4transitions[2]
PB4transitions[3].currentState = 0; PB4transitions[3].currentChar = 4; PB4transitions[3].nextState = 2; PB4transitions[3]
PB4transitions[4].currentState = 0; PB4transitions[4].currentChar = 5; PB4transitions[4].nextState = 3; PB4transitions[4]

//si main pleine
PB4transitions[5].currentState = 1; PB4transitions[5].currentChar = 0; PB4transitions[5].nextState = 0; PB4transitions[5]
PB4transitions[6].currentState = 1; PB4transitions[6].currentChar = 7; PB4transitions[6].nextState = 1; PB4transitions[6]
PB4transitions[7].currentState = 1; PB4transitions[7].currentChar = 5; PB4transitions[7].nextState = 5; PB4transitions[7]
PB4transitions[8].currentState = 1; PB4transitions[8].currentChar = 6; PB4transitions[8].nextState = 7; PB4transitions[8]
PB4transitions[9].currentState = 1; PB4transitions[9].currentChar = 4; PB4transitions[9].nextState = 6; PB4transitions[9]

//backtrace
PB4transitions[10].currentState = 7; PB4transitions[10].currentChar = 7; PB4transitions[10].nextState = 7; PB4transitions
PB4transitions[11].currentState = 7; PB4transitions[11].currentChar = 0; PB4transitions[11].nextState = 8; PB4transitions
PB4transitions[12].currentState = 8; PB4transitions[12].currentChar = 7; PB4transitions[12].nextState = 8; PB4transitions
PB4transitions[13].currentState = 8; PB4transitions[13].currentChar = 6; PB4transitions[13].nextState = 2; PB4transitions

```

```
877
878 //PHASE 2 :
879 //si ordre de vider
880     //si centre
881 PB4transitions[14].currentState = 3; PB4transitions[14].currentChar = 5; PB4transitions[14].nextState = 4; PB4transitions
882     //sinon
883 PB4transitions[15].currentState = 3; PB4transitions[15].currentChar = 0; PB4transitions[15].nextState = 3; PB4transitions
884 PB4transitions[16].currentState = 3; PB4transitions[16].currentChar = 3; PB4transitions[16].nextState = 3; PB4transitions
885 PB4transitions[17].currentState = 3; PB4transitions[17].currentChar = 7; PB4transitions[17].nextState = 3; PB4transitions
886 PB4transitions[18].currentState = 3; PB4transitions[18].currentChar = 8; PB4transitions[18].nextState = 3; PB4transitions
887     //revenir
888 PB4transitions[19].currentState = 4; PB4transitions[19].currentChar = 0; PB4transitions[19].nextState = 4; PB4transitions
889 PB4transitions[20].currentState = 4; PB4transitions[20].currentChar = 3; PB4transitions[20].nextState = 4; PB4transitions
890 PB4transitions[21].currentState = 4; PB4transitions[21].currentChar = 7; PB4transitions[21].nextState = 4; PB4transitions
891 PB4transitions[22].currentState = 4; PB4transitions[22].currentChar = 8; PB4transitions[22].nextState = 4; PB4transitions
892     //si centre se mettre en phase3.handFull
893 PB4transitions[23].currentState = 4; PB4transitions[23].currentChar = 4; PB4transitions[23].nextState = 5; PB4transitions
894
895 //si ordre de rajouter
896     //si centre
897 PB4transitions[24].currentState = 6; PB4transitions[24].currentChar = 4; PB4transitions[24].nextState = 9; PB4transitions
898     //sinon
899 PB4transitions[25].currentState = 6; PB4transitions[25].currentChar = 0; PB4transitions[25].nextState = 6; PB4transitions
900 PB4transitions[26].currentState = 6; PB4transitions[26].currentChar = 3; PB4transitions[26].nextState = 6; PB4transitions
901 PB4transitions[27].currentState = 6; PB4transitions[27].currentChar = 7; PB4transitions[27].nextState = 6; PB4transitions
902 PB4transitions[28].currentState = 6; PB4transitions[28].currentChar = 8; PB4transitions[28].nextState = 6; PB4transitions
903     //revenir
904 PB4transitions[29].currentState = 9; PB4transitions[29].currentChar = 0; PB4transitions[29].nextState = 9; PB4transitions
905 PB4transitions[30].currentState = 9; PB4transitions[30].currentChar = 3; PB4transitions[30].nextState = 9; PB4transitions
906 PB4transitions[31].currentState = 9; PB4transitions[31].currentChar = 7; PB4transitions[31].nextState = 9; PB4transitions
907 PB4transitions[32].currentState = 9; PB4transitions[32].currentChar = 8; PB4transitions[32].nextState = 9; PB4transitions
908     //si centre se mettre en phase3.handFull
909 PB4transitions[33].currentState = 9; PB4transitions[33].currentChar = 5; PB4transitions[33].nextState = 2; PB4transitions
910
911 //PHASE 3
912 //HandEmpty
913 PB4transitions[34].currentState = 2; PB4transitions[34].currentChar = 0; PB4transitions[34].nextState = 2; PB4transitions
914 PB4transitions[35].currentState = 2; PB4transitions[35].currentChar = 7; PB4transitions[35].nextState = 5; PB4transitions
915
916 //HandFull
```

```
PB4transitions[36].currentState = 5; PB4transitions[36].currentChar = 0; PB4transitions[36].nextState = 2; PB4transitions
PB4transitions[37].currentState = 5; PB4transitions[37].currentChar = 7; PB4transitions[37].nextState = 5; PB4transitions

//PHASE 4
//center border
PB4transitions[38].currentState = 2; PB4transitions[38].currentChar = 3; PB4transitions[38].nextState = 10; PB4transition
PB4transitions[39].currentState = 5; PB4transitions[39].currentChar = 3; PB4transitions[39].nextState = 10; PB4transition

//changements : inversion des caractères 7 et 8 et des caractères 5 et 6
//              copiage du centre se fait dans le sens inverse
//              décallage des états +10

//PHASE 5 :
//si main vide
PB4transitions[40].currentState = 10; PB4transitions[40].currentChar = 0; PB4transitions[40].nextState = 10; PB4transitio
PB4transitions[41].currentState = 10; PB4transitions[41].currentChar = 8; PB4transitions[41].nextState = 11; PB4transitio
PB4transitions[42].currentState = 10; PB4transitions[42].currentChar = 5; PB4transitions[42].nextState = 12; PB4transitio
PB4transitions[43].currentState = 10; PB4transitions[43].currentChar = 4; PB4transitions[43].nextState = 12; PB4transitio
PB4transitions[44].currentState = 10; PB4transitions[44].currentChar = 6; PB4transitions[44].nextState = 13; PB4transitio

//si main pleine
PB4transitions[45].currentState = 11; PB4transitions[45].currentChar = 0; PB4transitions[45].nextState = 10; PB4transitio
PB4transitions[46].currentState = 11; PB4transitions[46].currentChar = 8; PB4transitions[46].nextState = 11; PB4transitio
PB4transitions[47].currentState = 11; PB4transitions[47].currentChar = 6; PB4transitions[47].nextState = 15; PB4transitio
PB4transitions[48].currentState = 11; PB4transitions[48].currentChar = 5; PB4transitions[48].nextState = 17; PB4transitio
PB4transitions[49].currentState = 11; PB4transitions[49].currentChar = 4; PB4transitions[49].nextState = 16; PB4transitio

//backtrace
PB4transitions[50].currentState = 17; PB4transitions[50].currentChar = 8; PB4transitions[50].nextState = 17; PB4transitio
PB4transitions[51].currentState = 17; PB4transitions[51].currentChar = 0; PB4transitions[51].nextState = 18; PB4transitio
PB4transitions[52].currentState = 18; PB4transitions[52].currentChar = 8; PB4transitions[52].nextState = 18; PB4transitio
PB4transitions[53].currentState = 18; PB4transitions[53].currentChar = 5; PB4transitions[53].nextState = 12; PB4transitio


//PHASE 6 :
//si ordre de vider
    //si centre
PB4transitions[54].currentState = 13; PB4transitions[54].currentChar = 6; PB4transitions[54].nextState = 14; PB4transitio
    //sinon
PB4transitions[55].currentState = 13; PB4transitions[55].currentChar = 0; PB4transitions[55].nextState = 13; PB4transitio
```

```
PB4transitions[56].currentState = 13; PB4transitions[56].currentChar = 3; PB4transitions[56].nextState = 13; PB4transitio
PB4transitions[57].currentState = 13; PB4transitions[57].currentChar = 8; PB4transitions[57].nextState = 13; PB4transitio
PB4transitions[58].currentState = 13; PB4transitions[58].currentChar = 7; PB4transitions[58].nextState = 13; PB4transitio
    //revenir
PB4transitions[59].currentState = 14; PB4transitions[59].currentChar = 0; PB4transitions[59].nextState = 14; PB4transitio
PB4transitions[60].currentState = 14; PB4transitions[60].currentChar = 3; PB4transitions[60].nextState = 14; PB4transitio
PB4transitions[61].currentState = 14; PB4transitions[61].currentChar = 8; PB4transitions[61].nextState = 14; PB4transitio
PB4transitions[62].currentState = 14; PB4transitions[62].currentChar = 7; PB4transitions[62].nextState = 14; PB4transitio
    //si centre se mettre en phase3.handFull
PB4transitions[63].currentState = 14; PB4transitions[63].currentChar = 4; PB4transitions[63].nextState = 15; PB4transitio

//si ordre de rajouter
    //si centre
PB4transitions[64].currentState = 16; PB4transitions[64].currentChar = 4; PB4transitions[64].nextState = 19; PB4transitio
    //sinon
PB4transitions[65].currentState = 16; PB4transitions[65].currentChar = 0; PB4transitions[65].nextState = 16; PB4transitio
PB4transitions[66].currentState = 16; PB4transitions[66].currentChar = 3; PB4transitions[66].nextState = 16; PB4transitio
PB4transitions[67].currentState = 16; PB4transitions[67].currentChar = 8; PB4transitions[67].nextState = 16; PB4transitio
PB4transitions[68].currentState = 16; PB4transitions[68].currentChar = 7; PB4transitions[68].nextState = 16; PB4transitio
    //revenir
PB4transitions[69].currentState = 19; PB4transitions[69].currentChar = 0; PB4transitions[69].nextState = 19; PB4transitio
PB4transitions[70].currentState = 19; PB4transitions[70].currentChar = 3; PB4transitions[70].nextState = 19; PB4transitio
PB4transitions[71].currentState = 19; PB4transitions[71].currentChar = 8; PB4transitions[71].nextState = 19; PB4transitio
PB4transitions[72].currentState = 19; PB4transitions[72].currentChar = 7; PB4transitions[72].nextState = 19; PB4transitio
    //si centre se mettre en phase3.handFull
PB4transitions[73].currentState = 19; PB4transitions[73].currentChar = 6; PB4transitions[73].nextState = 12; PB4transitio

//PHASE 7
//HandEmpty
PB4transitions[74].currentState = 12; PB4transitions[74].currentChar = 0; PB4transitions[74].nextState = 12; PB4transitio
PB4transitions[75].currentState = 12; PB4transitions[75].currentChar = 8; PB4transitions[75].nextState = 15; PB4transitio

//HandFull
PB4transitions[76].currentState = 15; PB4transitions[76].currentChar = 0; PB4transitions[76].nextState = 12; PB4transitio
PB4transitions[77].currentState = 15; PB4transitions[77].currentChar = 8; PB4transitions[77].nextState = 15; PB4transitio

//PHASE 8
//Right border
PB4transitions[78].currentState = 12; PB4transitions[78].currentChar = 2; PB4transitions[78].nextState = 20; PB4transitio
PB4transitions[79].currentState = 15; PB4transitions[79].currentChar = 2; PB4transitions[79].nextState = 20; PB4transitio
```

```
//revenir
PB4transitions[80].currentState = 20; PB4transitions[80].currentChar = 0; PB4transitions[80].nextState = 20; PB4transitio
PB4transitions[81].currentState = 20; PB4transitions[81].currentChar = 3; PB4transitions[81].nextState = 20; PB4transitio
PB4transitions[82].currentState = 20; PB4transitions[82].currentChar = 4; PB4transitions[82].nextState = 20; PB4transitio
PB4transitions[83].currentState = 20; PB4transitions[83].currentChar = 5; PB4transitions[83].nextState = 20; PB4transitio
PB4transitions[84].currentState = 20; PB4transitions[84].currentChar = 6; PB4transitions[84].nextState = 20; PB4transitio
PB4transitions[85].currentState = 20; PB4transitions[85].currentChar = 7; PB4transitions[85].nextState = 20; PB4transitio
PB4transitions[86].currentState = 20; PB4transitions[86].currentChar = 8; PB4transitions[86].nextState = 20; PB4transitio

//Border gauche = recommencer
PB4transitions[87].currentState = 20; PB4transitions[87].currentChar = 1; PB4transitions[87].nextState = 21; PB4transitio
PB4transitions[88].currentState = 21; PB4transitions[88].currentChar = 1; PB4transitions[88].nextState = 0; PB4transition



int PB4transitionsSize = 89;
int PB4headPosition = 1;

printParameters* PB4printParams = malloc(sizeof(printParameters));
int PB4fileCount = 10000;
PB4printParams->fileCount = &PB4fileCount;
PB4printParams->leftBorderChar = 1;
PB4printParams->rightBorderChar = 2;
PB4printParams->printState = 21;


// en fonction des tests qu'on veut
//test simple

int PB4startingTape[65] = {1,7,7,0,0,7,7,7,7,0,0,0,7,7,7,0,4,0,0,0,7,7,7,7,7,0,0,0,0,0,0,0,3,0,0,0,8,8,8,8,8,8,8,0,0,0,0,
PB4printParams->OutputDir = "CrossRoadTest";
int PB4startingTapeLength = 65;

//accumulation
/*
int PB4startingTape[305] =
{1,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,0,7,
int PB4startingTapeLength = 305;
PB4printParams->OutputDir = "CrossRoadTestDots";
```

```
1036 */
1037
1038
1039
1040
1041 turing* PB4machine = makeTuringMachine(PB4alphabetSize, PB4blank, PB4stateSize, PB4starterState, PB4finalStates, PB4final
1042 //runMachine(PB4machine);
1043 //runMachineWithOptionalPrint2D(PB4machine, PB4printParams, 100);
1044
1045 //-----------------------------------------------------------------------------------------------------------------
1046
1047 return 0;
1048 }
```