PRÁCTICA GIT- GITHUB

Yordi Carbajal

David Vargas

Jose María Lagreca

Iker Lorente

Valeska Castañeda

Daniel Gardeta

1DAW 10/04/2025

Índice

EJERCICIO 1

a.

Se define un tipo enumerado tOrientacio con tres valores posibles: horizontal, vertical y diagonal. Luego, se crea un tipo tPosition como tupla que guarda una posición vertical, una horizontal y una orientación.

```
tipus
tOrientacio ={HORITZONTAL, VERTICAL, DIAGONAL}
tPosition=
    tupla
        vertical: enter;
        horitzontal: enter;
        orientacio: tOrientacio;
ftupla
ftipus
```

b.

Se define una constante para la longitud máxima de una palabra (15) y un tipo tWord que guarda la palabra, si ha sido encontrada y su posición en la sopa de letras.

```
const
    MAXPARAULA:enter=15;
fconst
tipus
    tWord=
        tupla
            paraula:char[MAXPARAULA];
            paraulaTrobada:boolean;
            posicion:tPosition;
        ftupla
ftipus
```

EJERCICIO 1

c.

Se crean constantes para el tamaño máximo de filas y columnas (50). Se crea el tipo tSoup, que representa una sopa de lletres con una matriz de caracteres, el número de filas y columnas, y tEstatSopa que indica si está vacía, llena, con caracteres aleatorios o no al 100% completada.

```
const
MAXFILA: enter = 50; MAXCOLUMNA: enter = 50;
fconst
tipus
tEstat={buida, omplerta, carAleat, ompAcercar} tSoup= tupla lletres: char[MAXFILA][MAXCOLUMNA]; files:
enter; columnes: enter; estat: tEstatSopa;
ftupla
ftipus
```

d.

Se define una constante que indica el número máximo de palabras a buscar (15). Luego, se crea el tipo tSearch, una tupla que contiene un array de palabras (tWord) que representan las palabras a buscar en la sopa de letras.

```
const
    MAXTOTALPARAU:enter=15;
fconst
tipus
    tSearch=
        tupla
            paraules: taula[MAXTOTALPARAU] de tWord
        ftupla
ftipus
```

e.

Se define el tipo tGame como una tupla que contiene una sopa de letras (tSopa) y las palabras a buscar (tSearch).

```
tipus
tGame= tupla sopa:tSopa; paraules:tSearch; ftupla
ftipus
```

EJERCICIO 2

a. Declarem una acció/funció getCharacter que a partir d’una sopa tSoup, retorni el caràcter que es troba en una fila i columna donades.

```
funcio getCharacter( entsor soup: tSoup, ent fila: enter, ent columna: enter ): caracter
    var
        c: caracter;
    fvar
    c := soup.matriz[fila][columna];
    retorna c;
ffuncio
```

b. Declareu una acció/funció setCharacter que col·loqui un caràcter donat en una determinada posició d’una tSoup.

```
accio setCharacter( entsor soup: tSoup, ent fila: enter, ent columna: enter, ent c: caracter )
    fvar
    soup.matriz[fila][columna] := c;
faccio
```

c. Declareu una acció/funció setWord que col·loqui una tWord en una determinada posició d’una tSoup, i amb una orientació donada. A més, la paraula és marcada com a “no trobada” i inicialitzada amb la seva posició i orientació.

```
accio setWord( entsor soup: tSoup, entsor word: tWord, ent fila: enter, ent columna: enter, ent ori: tOrientation )
    var
        i, len: enter;
    fvar
    word.trobada := fals;
    word.posFila := fila;
    word.posColumna := columna;
    word.orientacio := ori;
    len := getWordLength(word);
    per i:= 0 fins len - 1 fer
        si ori = HORIZONTAL llavors
            soup.matriz[fila][columna + i] := word.texto[i];
        sino si ori = VERTICAL llavors
            soup.matriz[fila + i][columna] := word.texto[i];
        sino si ori = DIAGONAL llavors
            soup.matriz[fila + i][columna + i] := word.texto[i];
        fsi
    fper
faccio
```

EJERCICIO 2

d. Declareu una acció/funció initSoup que inicialitzi una sopa tSoup. amb les dimensions donades.

```
accio initSoup( ent n: enter, ent m: enter, sor soup: tSoup )
var
    i, j: enter;
fvar
    soup.filas := n;
    soup.columnas := m;
    redimensionar soup.matriz a [n][m];
    per i:= 0 fins n - 1 fer
        per j:= 0 fins m - 1 fer
            soup.matriz[i][j] := ' ';
        fper
    fper
faccio
```

e. Declareu una acció/funció readWord que llegeixi i retorni una paraula tWord de l’entrada estàndard.

```
funcio readWord(): tWord
var
    word: tWord;
fvar
    word.texto := readString();
    retorna word;
ffuncio
```

EJERCICIO 3

```
algorisme fillSoup
var
    sopa: tSoup;
    search: tSearch;
    paraula: tWord;
    n, m, w, i: enter;
fvar
    n := readInteger();
    m := readInteger();
    initSoup(n, m, sopa);
    initSearch(search);
    fillWithRandomCharacters(n, m, sopa);
    w := readInteger();
    per i:=0 fins w-1 fer
        paraula := readWord();
        addWordToSearch(search, paraula);
        hideWordIntoSoup(paraula, n, m, sopa);
    fper
    writeSoup(n, m, sopa);
falgorisme

accio fillWithRandomCharacters( ent n: enter, ent m: enter, entsor sopa: tSoup )
var
    i, j: enter;
    c: caracter;
fvar
    per i:=0 fins n-1 fer
        per j:=0 fins m-1 fer
            c := getRandomCharacter();
            setCharacter(sopa, i, j, c);
        fper
    fper faccio

accio initSearch( sor search: tSearch )
fvar
    redimensionar search.paraules a [0];
faccio

accio addWordToSearch( entsor search: tSearch, ent word: tWord )
var
    mida: enter;
fvar
    mida := longitud(search.paraules);
    redimensionar search.paraules a [mida+1];
    search.paraules[mida] := word;
faccio

funcio getRandomCharacter(): caracter
var
    codi: enter;
fvar
    codi := getRandomNumber(65, 90);
    retorna chr(codi);
ffuncio
```

EJERCICIO 4

A. Diseñar una acción reverseWord que invierta el orden de los caracteres de una palabra dada.

```
accio reverseWord(entsor word: tWord)
var len, i: enter;
fvar
len := getWordLength(word);
per i := 0 fins len / 2 fer
    var
        tmp: caracter;
    fvar
    tmp := word.paraula[i];
    word.paraula[i] := word.paraula[len - 1 - i];
    word.paraula[len - 1 - i] := tmp;
fper
faccio
```

B. Modificar el algoritmo del ejercicio 3 para conseguir el efecto descrito, utilizando la acción reverseWord.

```
accio hideWordIntoSoup(entsor word: tWord, ent n: enter, ent m: enter, entsor soup: tSoup)
var
    len, ori, i, j: enter;
fvar
len := getWordLength(word);
ori := getRandomNumber(1, 3);
si
getRandomNumber(0, 1) = 1 llavors
    reverseWord(word);
fsi
si
ori = VERTICAL llavors
    i := getRandomNumber(0, n - len + 1);
    j := getRandomNumber(0, m);
sino si ori = HORIZONTAL llavors
    i := getRandomNumber(0, n);
    j := getRandomNumber(0, m - len + 1);
sino
    i := getRandomNumber(0, n - len + 1);
    j := getRandomNumber(0, m - len + 1);
fsi
setWord(soup, word, i, j, ori);
word.trobada := fals;
faccio
```

Ejercicio 5

Código Final pasado a C:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <ctype.h>

#define MAX_ROWS 50
#define MAX_COLUMNS 50
#define MAX_WORDS 15
#define MAX_WORD_LENGTH 15
#define POINT '.'

// Tipos de orientación
typedef enum {
    HORIZONTAL = 1,
    VERTICAL = 2,
    DIAGONAL = 3
} Orientation;

// Posición de una palabra en la sopa
typedef struct {
    int row;
    int col;
    Orientation orientation;
} tPosition;

// Representación de una palabra
typedef struct {
    char text[MAX_WORD_LENGTH + 1];
    int found;
    tPosition position;
} tWord;

// Representación de la sopa de letras
typedef enum {EMPTY, RANDOM_FILLED, READY} tState;
typedef struct {
    char grid[MAX_ROWS][MAX_COLUMNS];
    int rows;
    int columns;
    tState state;
} tSoup;

// Lista de palabras a buscar
typedef struct {
    tWord words[MAX_WORDS];
    int count;
} tSearch;

// Información del juego
typedef struct {
    tSoup soup;
    tSearch search;
} tGame;

// === Funciones de utilidades ===
int getRandomNumber(int min, int max) {
    return min + rand() % (max - min + 1);
}

char getRandomCharacter() {
    return 'A' + getRandomNumber(0, 25);
}

void reverseWord(tWord* word) {
    int len = strlen(word->text), i;
    for (i = 0; i < len / 2; i++) {
        char tmp = word->text[i];
        word->text[i] = word->text[len - 1 - i];
        word->text[len - 1 - i] = tmp;
    }
}

// === Funciones relacionadas con tSoup ===
char getCharacter(tSoup* soup, int row, int col) {
    return soup->grid[row][col];
}

void setCharacter(tSoup* soup, int row, int col, char c) {
    soup->grid[row][col] = c;
}

void initSoup(int n, int m, tSoup* soup) {
    int i, j;
    soup->rows = n;
    soup->columns = m;
    soup->state = EMPTY;
    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
            soup->grid[i][j] = ' ';
}

void fillWithRandomCharacters(int n, int m, tSoup* soup) {
    int i, j;
    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
            soup->grid[i][j] = getRandomCharacter();
    soup->state = RANDOM_FILLED;
}

void writeSoup(tSoup* soup) {
    int i, j;
    printf("< %d %d\n", soup->rows, soup->columns);
    for (i = 0; i < soup->rows; i++) {
        for (j = 0; j < soup->columns; j++) {
            printf("%c ", soup->grid[i][j]);
        }
        printf("\n");
    }
    printf(">\n");
}

// === Funciones relacionadas con palabras y búsqueda ===
void initSearch(tSearch* search) {
    search->count = 0;
}

tWord readWord() {
    tWord word;
    scanf("%s", word.text);
    word.found = 0;
    return word;
}

void addWordToSearch(tSearch* search, tWord word) {
    if (search->count < MAX_WORDS) {
        search->words[search->count++] = word;
    }
}

void setWord(tSoup* soup, tWord* word, int row, int col, Orientation orientation) {
    int len = strlen(word->text), i;
    word->position.row = row;
    word->position.col = col;
    word->position.orientation = orientation;
    word->found = 0;
    for (i = 0; i < len; i++) {
        int r = row, c = col;
        if (orientation == VERTICAL) r += i;
        else if (orientation == HORIZONTAL) c += i;
        else if (orientation == DIAGONAL) { r += i; c += i; }
        soup->grid[r][c] = word->text[i];
    }
}

// === Función principal de ocultar palabra en sopa ===
void hideWordIntoSoup(tWord* word, int n, int m, tSoup* soup) {
    int len = strlen(word->text);
    Orientation ori = getRandomNumber(1, 3);
    // 50% de probabilidad de invertir la palabra
    if (getRandomNumber(0, 1)) {
        reverseWord(word);
    }
    int i, j;
    if (ori == VERTICAL) {
        i = getRandomNumber(0, n - len);
        j = getRandomNumber(0, m - 1);
    } else if (ori == HORIZONTAL) {
        i = getRandomNumber(0, n - 1);
        j = getRandomNumber(0, m - len);
    } else { // DIAGONAL
        i = getRandomNumber(0, n - len);
        j = getRandomNumber(0, m - len);
    }
    setWord(soup, word, i, j, ori);
    word->found = 0;
}

// === Lectura de entrada y ejecución ===
void fillSoup() {
    tSoup soup;
    tSearch search;
    tWord word;
    int n, m, w;
    printf("Introduce filas y columnas (ej: 10 10):\n");
    scanf("%d %d", &n, &m);
    initSoup(n, m, &soup);
    fillWithRandomCharacters(n, m, &soup);
    printf("Introduce cuántas palabras quieres ocultar:\n");
    scanf("%d", &w);
    initSearch(&search);
    int i;
    for (i = 0; i < w; i++) {
        printf("Introduce la palabra %d:\n", i + 1);
        word = readWord();
        addWordToSearch(&search, word);
        hideWordIntoSoup(&search.words[search.count - 1], n, m, &soup);
    }
    soup.state = READY;
    writeSoup(&soup);
}

// === Algoritmo de resolució (EXERCICI 5) ===
typedef struct {
    char letters[MAX_ROWS][MAX_COLUMNS];
    int nRows, nCols;
} tLetterSoup;

typedef struct {
    char letters[MAX_WORD_LENGTH + 1];
    int length;
} tString;

int isUpperCaseLetter(char c) {
    return c >= 'A' && c <= 'Z';
}

void readInputSoup(tLetterSoup* soup) {
    int i, j;
    for (i = 0; i < soup->nRows; i++) {
        for (j = 0; j < soup->nCols; j++) {
            char c;
            do {
                scanf(" %c", &c);
            } while (!isUpperCaseLetter(c));
            soup->letters[i][j] = c;
        }
    }
}

void initOutputSoup(tLetterSoup* soup) {
    int i, j;
    for (i = 0; i < soup->nRows; i++) {
        for (j = 0; j < soup->nCols; j++) {
            soup->letters[i][j] = POINT;
        }
    }
}

tString readString() {
    tString str;
    char c;
    str.length = 0;
    do scanf(" %c", &c); while (!isUpperCaseLetter(c));
    while (isUpperCaseLetter(c) && str.length < MAX_WORD_LENGTH) {
        str.letters[str.length++] = c;
        scanf("%c", &c);
    }
    str.letters[str.length] = '\\0';
    return str;
}

void checkWord(tLetterSoup* soup, tString* word, int x, int y, int dx, int dy, int* found) {
    int i = 0;
    int match = 1;
    while (i < word->length && match) {
        if (x < 0 || y < 0 || x >= soup->nRows || y >= soup->nCols || soup->letters[x][y] != word->letters[i]) {
            match = 0;
        } else {
            x += dx;
            y += dy;
            i++;
        }
    }
    *found = match && (i == word->length);
}

void lookForWord(tLetterSoup* soup, tString* word, int x, int y, int* dx, int* dy, int* found) {
    *found = 0;
    int dX, dY;
    for (dX = -1; dX <= 1 && !*found; dX++) {
        for (dY = -1; dY <= 1 && !*found; dY++) {
            if (dX == 0 && dY == 0) continue;
            checkWord(soup, word, x, y, dX, dY, found);
            if (*found) {
                *dx = dX;
                *dy = dY;
            }
        }
    }
}

void putWord(tLetterSoup* soup, tString* word, int x, int y, int dx, int dy) {
    int i, j;
    for (i = 0; i < word->length; i++) {
        soup->letters[x][y] = word->letters[i];
        x += dx;
        y += dy;
    }
}

void writeOutputSoup(tLetterSoup* soup) {
    int i, j;
    printf("< %d %d\n", soup->nRows, soup->nCols);
    for (i = 0; i < soup->nRows; i++) {
        for (j = 0; j < soup->nCols; j++) {
            printf("%c ", soup->letters[i][j]);
        }
        printf("\n");
    }
    printf(">\n");
}

void solveSoup() {
    tLetterSoup inputSoup, outputSoup;
    tString currentWord;
    int w;
    int found, dx, dy;
    int i, x, y;
    printf("== RESOLVER SOPA DE LETRAS ==\\n");
    printf("Introduce el contenido de la sopa en el siguiente formato:\\n");
    printf("< n m\\n");
    printf("Letras (en mayúsculas, separadas por espacio)\\n");
    printf("... (n filas en total)\\n");
    printf(">\\n");
    printf("Después, el número de palabras a buscar y las palabras (una por línea).\\n");
    printf("Ejemplo:\\n");
    printf("< 5 5\\nA B C D E\\nF G H I J\\nK L M N O\\nP Q R S T\\nU V W X
Y\\n>\\n3\\nPERRO\\nGATO\\nLUNA\\n");
    printf("\\nEsperando entrada...\\n");
    fflush(stdin);

    // Lectura del tamaño
    scanf("< %d %d", &inputSoup.nRows, &inputSoup.nCols);
    printf("Tamaño de la sopa: %d filas x %d columnas\\n", inputSoup.nRows, inputSoup.nCols);

    // Lectura de la sopa
    printf("Leyendo letras de la sopa...\\n");
    readInputSoup(&inputSoup);
    fflush(stdin);

    outputSoup.nRows = inputSoup.nRows;
    outputSoup.nCols = inputSoup.nCols;
    initOutputSoup(&outputSoup);
    fflush(stdin);

    // Lectura del número de palabras
    printf("Introduce el numero de palabras a buscar: \\n");
    scanf("%d", &w);
    printf("Se buscarán %d palabra(s)...\\n", w);
    fflush(stdin);

    for (i = 0; i < w; i++) {
        printf("Introduce la palabra %d:\\n", i + 1);
        currentWord = readString();
        printf("Buscando palabra: %s\\n", currentWord.letters);
        found = 0;
        for (x = 0; x < inputSoup.nRows && !found; x++) {
            for (y = 0; y < inputSoup.nCols && !found; y++) {
                lookForWord(&inputSoup, &currentWord, x, y, &dx, &dy, &found);
                if (found) {
                    printf("? Palabra encontrada en (%d, %d) con dirección (%d, %d)\\n", x, y, dx, dy);
                    putWord(&outputSoup, &currentWord, x, y, dx, dy);
                }
            }
        }
        if (!found) {
            printf("? No se encontró la palabra: %s\\n", currentWord.letters);
        }
    }
    fflush(stdin);
    printf("\\nSopa con palabras encontradas:\\n");
    writeOutputSoup(&outputSoup);
}

// === MAIN ===
int main() {
    srand(time(NULL));
    fillSoup(); // Ejercicio 3 y 4
    solveSoup(); // Ejercicio 5
    return 0;
}
```