Ejercicios de la sesión 4 de C resueltos

1. Usando el procedimiento del ejercicio 2 del seminario anterior, escribe un programa que multiplique dos matrices A y B leídas de ficheros, la primera de tamaño n×m y la segunda de tamaño m×p. Las matrices deben leerse cada una de un fichero distinto y el resultado debe estar en otro fichero. El formato del fichero es el siguiente: en la primera línea habrán 2 enteros, indicando el tamaño de la matriz en cada dimensión. A continuación vienen las filas de la matriz, cada una en una misma línea. Los valores son números reales. El resultado debe estar en otro fichero con el mismo formato.

   Por ejemplo, un fichero de definición de una matríz podría ser el siguiente:

   ```
   3 4
   1     0.1   0     1.2
   0     1.2   3.4   0.3
   2.8   2.4   5.3   5.2
   ```

   Los nombres de los ficheros serán los argumentos del programa o (si no existen) se solicitarán a través del teclado.

Solución: *Una posible implementación sería ésta*:

```c
#include <stdlib.h>
#include <limits.h>
#include <string.h>

/**
 * Crea una matriz de doubles de tamaño N x M, reservando la memoria de
 * forma dinámica.
 */
double *creaMatriz(int m, int n) {
    return calloc(m*n, sizeof (double));
}

/**
 * Multiplica dos matrices A y B, escribiendo el resultado en una tercera C.
 * Las matrices a multiplicar deben ser de dimensiones compatibles, en otro
 * caso se devuelve el valor NULL como resultado. Si todo funciona
correctamente,
 * se devuelve como resultado un puntero a C. Si C, que se pasa como
parámetro,
 * tiene como valor NULL, se reserva espacio para una nueva matriz y se
devuelve
 * el puntero como resultado.
 * Parametros:
 * - double *A: puntero a la matriz A
 * - int m: filas de A
 * - int n: columnas de A
 * - double *B: puntero a la matriz B
 * - int r: filas de B
 * - int s: columnas de B
 * - double *C: puntero a la matriz C (puede ser NULL si se quiere crear)
 * - int *t: puntero a filas de C (para poder escribirlo en caso de crearla)
 * - int *u: puntero a columnas de C (para poder escribirlo en caso de
crearla)
```

```
 *
 * Devuelve: NULL en caso de error. Un puntero a la matriz resultante en otro
 * caso. Si C era NULL, se creará una nueva matriz. En *t y *u se escribiran
 * las dimensiones de la nueva matriz.
 */
double *multiplicaMatrices(double *A, int m, int n, double *B, int r, int s,
        double *C, int *t, int *u) {
    int i, j, k;

    if (n != r)
        return NULL; //Dimensiones incompatibles

    if (C && (m>*t || s>*u))
        return NULL; //Dimensiones del C existente incorrectas

    if (!C) { // Si C es NULL, creamos la matriz resultado
        C = creaMatriz(m, s);
        *t = m;
        *u = s;
    }

    for (i = 0; i < m; i++) { //Recorremos filas
        for (j = 0; j < s; j++) { //Recorremos columnas
            C[i * s + j] = 0; //Inicializamos el valor para la posicion i,j
            for (k = 0; k < n; k++) {
                C[i * s + j] += A[i * n + k] * B[k * s + j];
            }
        }
    }

    return C;
}

/*
 * Lee una matriz del fichero indicado “nombreFichero”, reservando la memoria
 * en “M” e indicando las dimensiones en “m” y “n”. Devuelve M=NULL en caso de
 * error, indicando el codigo de error en n.
 */
void leeMatriz(char *nombreFichero, double **M, int *m, int *n) {
    FILE *f;
    f = fopen(nombreFichero, "r");
    if (!f) {
        *M = NULL;
        *m = 0;
        *n = 1; // El fichero no se ha podido abrir
        return;
    }
    if (fscanf(f, "%d %d\n", m, n) != 2) {
        fclose(f);
        *M = NULL;
        *m = 0;
        *n = 2; // Las dimensiones no se han podido leer
        return;
    }
    *M = creaMatriz(*m, *n);
    if (!*M) {
        fclose(f);
        *M = NULL;
        *m = 0;
```

```
        *n = 3; //Imposible reservar la memoria
        return;
    }

    int i, j;
    float v;
    for (i = 0; i<*m; i++) {
        for (j = 0; j<*n; j++) {
            fscanf(f, "%f", &v);
            (*M)[i * (*n) + j] = v;
        }
    }
    fclose(f);
}

/*
 * Escribe la matriz “M” con dimensiones “m” X “n” en el fichero indicado
 * “nombreFichero”. Devuelve 0 en caso de ejecución correcta.
 */
int escribeMatriz(char *nombreFichero, double *M, int m, int n) {
    FILE *f;
    int i, j;

    f = fopen(nombreFichero, "w");
    if (!f) {
        return 1; //No se puede crear el fichero
    }
    fprintf(f, "%d %d\n", m, n);
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            fprintf(f, "%g ", M[i * n + j]);
        }
        putc('\n', f);
    }
    fclose(f);
    return 0;
}

/*
 * Imprime una matriz por la salida estándar
 */
void imprimeMatriz(double *M, int m, int n) {
    int i, j;
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            printf("%g ", M[i * n + j]);
        }
        putchar('\n');
    }
}

/* Principal */
int main(int argc, char *argv[]) {
    double *A, *B, *C = NULL;
    int am, an, bm, bn, cm, cn; // Dimensiones
    char *matrizA, *matrizB, *matrizC;
    int sz;

    if (argc < 2) {
```

```
        printf("Indique el nombre del fichero con la matriz A: ");
        matrizA = malloc(PATH_MAX + 1);
        fgets(matrizA, PATH_MAX, stdin);
        if (!(sz = strlen(matrizA))) {
            return EXIT_SUCCESS;
        }
        if (matrizA[sz - 1] == '\n')
            matrizA[sz - 1] = '\0';

    } else {
        matrizA = argv[1];
    }

    leeMatriz(matrizA, &A, &am, &an);
    if (!A) {
        printf("¡Imposible leer A de %s, código de error %d!\n", matrizA, an);
        return (EXIT_FAILURE);
    }
    printf("Leido A =\n");
    imprimeMatriz(A, am, an);

    if (argc < 3) {
        printf("Indique el nombre del fichero con la matriz B: ");
        matrizB = malloc(PATH_MAX + 1);
        fgets(matrizB, PATH_MAX, stdin);
        if (!(sz = strlen(matrizB))) {
            return EXIT_SUCCESS;
        }
        if (matrizB[sz - 1] == '\n')
            matrizB[sz - 1] = '\0';

    } else {
        matrizB = argv[2];
    }
    leeMatriz(matrizB, &B, &bm, &bn);
    if (!B) {
        printf("¡Imposible leer B de %s, código de error %d!\n", matrizB, bn);
        return (EXIT_FAILURE);
    }
    printf("Leido B =\n");
    imprimeMatriz(B, bm, bn);

    C = multiplicaMatrices(A, am, an, B, bm, bn, C, &cm, &cn);
    if (!C) {
        printf("¡Imposible multiplicar AxB!\n");
        return (EXIT_FAILURE);
    }
    printf("\nC = AxB =\n");
    imprimeMatriz(C, cm, cn);

    if (argc < 4) {
        printf("Indique el nombre del fichero donde guardar la matriz C: ");
        matrizC = malloc(PATH_MAX + 1);
        fgets(matrizC, PATH_MAX, stdin);
        if (!(sz = strlen(matrizC))) {
            return EXIT_SUCCESS;
        }
        if (matrizC[sz - 1] == '\n')
            matrizC[sz - 1] = '\0';
```

```
    } else {
        matrizC = argv[3];
    }
    if (escribeMatriz(matrizC, C, cm, cn)) {
        printf("¡Imposible escribir C en %s!\n", matrizC);
        return (EXIT_FAILURE);
    }

    /* Liberamos la memoria */
    free(A);
    free(B);
    free(C);

    if (argc < 2)
        free(matrizA);
    if (argc < 3)
        free(matrizB);
    if (argc < 4)
        free(matrizC);

    return (EXIT_SUCCESS);
}
```

2. Dado el programa escrito en el ejercicio anterior (cálculo del producto de dos matrices) dividir el código en tres módulos: Ficheros (funciones de entrada salida), Matematico (tipo de datos matriz y producto de matrices) y Principal (programa principal). Crea un fichero makefile para generar automáticamente el programa ejecutable.

Solución: *Una posible implementación sería ésta*:

```
/*
 * Fichero:   matematico.h
 */

#ifndef _MATEMATICO_H
#define _MATEMATICO_H

/**
 * Crea una matriz de doubles de tamaño N x M, reservando la memoria de
 * forma dinámica.
 */
double *creaMatriz(int m, int n);

/**
 * Multiplica dos matrices A y B, escribiendo el resultado en una tercera C.
 * Las matrices a multiplicar deben ser de dimensiones compatibles, en otro
 * caso se devuelve el valor NULL como resultado. Si funciona correctamente,
 * se devuelve como resultado un puntero a C. Si C tiene como valor NULL,
 * se reserva espacio para una nueva matriz y se devuelve el puntero.
 * Parametros:
 * - double *A: puntero a la matriz A
 * - int m: filas de A
 * - int n: columnas de A
 * - double *B: puntero a la matriz B
 * - int r: filas de B
 * - int s: columnas de B
```

```
 * - double *C: puntero a la matriz C (puede ser NULL si se quiere crear)
 * - int *t: puntero a filas de C (para poder escribirlo en caso de crearla)
 * - int *u: puntero a columnas de C (para poder escribirlo)
 *
 * Devuelve: NULL en caso de error. Un puntero a la matriz resultante en otro
 * caso. Si C era NULL, se creará una nueva matriz. En *t y *u se escribiran
 * las dimensiones de la nueva matriz.
 */
double *multiplicaMatrices(double *A, int m, int n, double *B, int r, int s,
        double *C, int *t, int *u);

#endif  /* _MATEMATICO_H */
```

```
/*
 * Fichero:   matematico.c
 */

#include "matematico.h"

#include <stdlib.h>

/**
 * Crea una matriz de doubles de tamaño N x M, reservando la memoria de
 * forma dinámica.
 */
double *creaMatriz(int m, int n) {
    return calloc(m*n, sizeof (double));
}

/**
 * Multiplica dos matrices A y B, escribiendo el resultado en una tercera C.
 * Las matrices a multiplicar deben ser de dimensiones compatibles, en otro
 * caso se devuelve el valor NULL como resultado. Si todo funciona
correctamente,
 * se devuelve como resultado un puntero a C. Si C, que se pasa como
parámetro,
 * tiene como valor NULL, se reserva espacio para una nueva matriz y se
devuelve
 * el puntero como resultado.
 * Parametros:
 * - double *A: puntero a la matriz A
 * - int m: filas de A
 * - int n: columnas de A
 * - double *B: puntero a la matriz B
 * - int r: filas de B
 * - int s: columnas de B
 * - double *C: puntero a la matriz C (puede ser NULL si se quiere crear)
 * - int *t: puntero a filas de C (para poder escribirlo en caso de crearla)
 * - int *u: puntero a columnas de C (para poder escribirlo en caso de
crearla)
 *
 * Devuelve: NULL en caso de error. Un puntero a la matriz resultante en otro
 * caso. Si C era NULL, se creará una nueva matriz. En *t y *u se escribiran
 * las dimensiones de la nueva matriz.
 */
double *multiplicaMatrices(double *A, int m, int n, double *B, int r, int s,
```

```
        double *C, int *t, int *u) {
    int i, j, k;

    if (n != r)
        return NULL; //Dimensiones incompatibles

    if (C && (m>*t || s>*u))
        return NULL; //Dimensiones del C existente incorrectas

    if (!C) { // Si C es NULL, creamos la matriz resultado
        C = creaMatriz(m, s);
        *t = m;
        *u = s;
    }

    for (i = 0; i < m; i++) { //Recorremos filas
        for (j = 0; j < s; j++) { //Recorremos columnas
            C[i * s + j] = 0; //Inicializamos el valor para la posicion i,j
            for (k = 0; k < n; k++) {
                C[i * s + j] += A[i * n + k] * B[k * s + j];
            }
        }
    }

    return C;
}
```

```
/*
 * File:   ficheros.h
 */

#ifndef _FICHEROS_H
#define _FICHEROS_H

/*
 * Lee una matriz del fichero indicado “nombreFichero”, reservando la memoria
 * en “M” e indicando las dimensiones en “m” y “n”. Devuelve M=NULL en caso de
 * error, indicando el código de error en n.
 */
void leeMatriz(char *nombreFichero, double **M, int *m, int *n);

/*
 * Escribe la matriz “M” con dimensiones “m” X “n” en el fichero indicado
 * “nombreFichero”. Devuelve 0 en caso de ejecución correcta.
 */
int escribeMatriz(char *nombreFichero, double *M, int m, int n);

#endif  /* _FICHEROS_H */
```

```
/*
 * File:   ficheros.c
 */
#include "ficheros.h"

#include <stdio.h>
#include "matematico.h"
```

```c
/*
 * Lee una matriz del fichero indicado “nombreFichero”, reservando la memoria
 * en “M” e indicando las dimensiones en “m” y “n”. Devuelve M=NULL en caso de
 * error, indicando el código de error en n.
 */
void leeMatriz(char *nombreFichero, double **M, int *m, int *n) {
    FILE *f;
    f = fopen(nombreFichero, "r");
    if (!f) {
        *M = NULL;
        *n = 1; // El fichero no se ha podido abrir
        return;
    }
    if (fscanf(f, "%d %d\n", m, n) != 2) {
        fclose(f);
        *M = NULL;
        *n = 2; // Las dimensiones no se han podido leer
        return;
    }
    *M = creaMatriz(*m, *n);
    if (!*M) {
        fclose(f);
        *M = NULL;
        *n = 3; //Imposible reservar la memoria
        return;
    }

    int i, j;
    float v;
    for (i = 0; i<*m; i++) {
        for (j = 0; j<*n; j++) {
            fscanf(f, "%f", &v);
            (*M)[i * (*n) + j]=v;
        }
    }
    fclose(f);
}

/*
 * Escribe la matriz “M” con dimensiones “m” X “n” en el fichero indicado
 * “nombreFichero”. Devuelve 0 en caso de ejecución correcta.
 */
int escribeMatriz(char *nombreFichero, double *M, int m, int n) {
    FILE *f;
    int i, j;

    f = fopen(nombreFichero, "w");
    if (!f) {
        return 1; //No se puede crear el fichero
    }
    fprintf(f, "%d %d\n", m, n);
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            fprintf(f, "%g ", M[i * n + j]);
        }
        putc('\n', f);
    }
    fclose(f);
```

```
    return 0;
}
```

```
/*
 * File:   principal.c
 */

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

#include "matematico.h"
#include "ficheros.h"

/*
 * Imprime una matriz por la salida estándar
 */
void imprimeMatriz(double *M, int m, int n) {
    int i, j;
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            printf("%g ", M[i * n + j]);
        }
        putchar('\n');
    }
}

/*
 * Programa de prueba
 */
int main(int argc, char *argv[]) {
    double *A, *B, *C = NULL;
    int am, an, bm, bn, cm, cn; // Dimensiones
    char *matrizA, *matrizB, *matrizC;
    int sz;

    if (argc < 2) {
        printf("Indique el nombre del fichero con la matriz A: ");
        matrizA = malloc(PATH_MAX + 1);
        fgets(matrizA, PATH_MAX, stdin);
        if (!(sz = strlen(matrizA))) {
            return EXIT_SUCCESS;
        }
        if (matrizA[sz - 1] == '\n')
            matrizA[sz - 1] = '\0';

    } else {
        matrizA = argv[1];
    }

    leeMatriz(matrizA, &A, &am, &an);
    if (!A) {
        printf("¡Imposible leer A de %s, código de error %d!\n", matrizA, an);
        return (EXIT_FAILURE);
    }
    printf("Leido A =\n");
    imprimeMatriz(A, am, an);
```

```
    if (argc < 3) {
        printf("Indique el nombre del fichero con la matriz B: ");
        matrizB = malloc(PATH_MAX + 1);
        fgets(matrizB, PATH_MAX, stdin);
        if (!(sz = strlen(matrizB))) {
            return EXIT_SUCCESS;
        }
        if (matrizB[sz - 1] == '\n')
            matrizB[sz - 1] = '\0';

    } else {
        matrizB = argv[2];
    }
    leeMatriz(matrizB, &B, &bm, &bn);
    if (!B) {
        printf("¡Imposible leer B de %s, código de error %d!\n", matrizB, bn);
        return (EXIT_FAILURE);
    }
    printf("Leido B =\n");
    imprimeMatriz(B, bm, bn);

    C = multiplicaMatrices(A, am, an, B, bm, bn, C, &cm, &cn);
    if (!C) {
        printf("¡Imposible multiplicar AxB!\n");
        return (EXIT_FAILURE);
    }
    printf("\nC = AxB =\n");
    imprimeMatriz(C, cm, cn);

    if (argc < 4) {
        printf("Indique el nombre del fichero donde guardar la matriz C: ");
        matrizC = malloc(PATH_MAX + 1);
        fgets(matrizC, PATH_MAX, stdin);
        if (!(sz = strlen(matrizC))) {
            return EXIT_SUCCESS;
        }
        if (matrizC[sz - 1] == '\n')
            matrizC[sz - 1] = '\0';

    } else {
        matrizC = argv[3];
    }
    if (escribeMatriz(matrizC, C, cm, cn)) {
        printf("¡Imposible escribir C en %s!\n", matrizC);
        return (EXIT_FAILURE);
    }

    /* Liberamos la memoria */
    free(A);
    free(B);
    free(C);

    if (argc < 2)
        free(matrizA);
    if (argc < 3)
        free(matrizB);
    if (argc < 4)
        free(matrizC);
```

```
    return (EXIT_SUCCESS);
}
```

```
## -*- Makefile -*-

CFLAGS = -g -Wall

# Enlaza todos los .o en un fichero final ejecutable
a.out: principal.o matematico.o ficheros.o
    gcc $(CFLAGS) -o $@ principal.o matematico.o ficheros.o


# Compila ficheros fuente en  ficheros .o
principal.o: principal.c matematico.h ficheros.h
    gcc $(CFLAGS) -c principal.c

matematico.o: matematico.c matematico.h
    gcc $(CFLAGS) -c matematico.c

ficheros.o: ficheros.c ficheros.h matematico.h
    gcc $(CFLAGS) -c ficheros.c


#### Para limpiar el directorio de ficheros generados ####
clean:
    rm -f a.out *.o
```

3. Escribe un programa que dado un nombre de fichero, que se pasará como parámetro en la línea de comandos, haga que la primera letra de cada palabra esté en mayúsculas. Se deberá poder compilar en "modo debug", mostrando en ese caso los principales pasos de ejecución. Crea un fichero makefile para generar automáticamente el programa ejecutable.

Solución: *Una posible implementación sería ésta*:

```
/*
 * File:   ponmayusculas.h
 */

#ifndef _PONMAYUSCULAS_H
#define _PONMAYUSCULAS_H

/**
 * Abre el fichero *nombreFichero* en modo lectura escritura y cambia la
 * primera letra de capa palabra a mayúsculas.
 * Devuelve 0 si todo va bien, un código de error en otro caso.
 */
int ponMayusculas(char *nombreFichero);

#endif  /* _PONMAYUSCULAS_H */
```

```
/*
 * File:   ponmayusculas.c
 */
```

```c
#include "ponmayusculas.h"

#include <stdio.h>
#include <ctype.h>

int ponMayusculas(char* nombreFichero) {
    FILE *f;
    int c;

    f = fopen(nombreFichero, "r+");
    if (!f) {
#ifdef DEBUG
        fprintf(stderr, "El fichero %s no se ha podido abrir correctamente\n",
                nombreFichero);
#endif
        return 1;
    }

#ifdef DEBUG
    fprintf(stderr, "Fichero %s abierto correctamente\n", nombreFichero);
#endif

    while (!feof(f)) {
#ifdef DEBUG
        fprintf(stderr, "Buscando inicio de palabra... ");
#endif
        while (!isalnum(c = getc(f)) && !feof(f));//Buscamos principio palabra
        if (feof(f))
            break;
#ifdef DEBUG
        fprintf(stderr, "encontrado '%c', convirtiendo en '%c'\n",
            c, toupper(c));
#endif
        fseek(f, -1, SEEK_CUR); //Nos movemos uno atrás para sobreescribir
        putc(toupper(c), f); //Sobreescribimos con la versión en mayúsculas
        while (isalpha(c = getc(f)) && !feof(f)); //Saltamos el resto palabra
    }
#ifdef DEBUG
    fprintf(stderr, "\nFichero %s procesado completamente\n", nombreFichero);
#endif

    fclose(f);

    return 0;
}
```

```c
/*
 * File:   principal.c
 */

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

#include "ponmayusculas.h"

void imprimeUso(char *ejecutable) {
    fprintf(stderr, "Uso del programa:\n\t%s <nombre_fichero_a_procesar>\n",
            ejecutable);
```

```
    fputs(" Este programa modifica el 'fichero_a_procesar' convirtiendo",
            stderr);
    fputs("cada primera letra de cada palabra a mayusculas.", stderr);
}

int main(int argc, char** argv) {
    if (argc < 2) {
        imprimeUso(argv[0]);
        return (EXIT_FAILURE);
    }
    if (ponMayusculas(argv[1])) {
        fprintf(stderr, "El fichero %s ha sido imposible de procesar\n",
            argv[1]);
        imprimeUso(argv[0]);
        return (EXIT_FAILURE);
    }
    return (EXIT_SUCCESS);
}
```

```
## -*- Makefile -*-

CFLAGS = -g -Wall -DDEBUG
#CFLAGS = -g -Wall

# Enlaza todos los .o en un fichero final ejecutable
a.out: principal.o ponmayusculas.o
    gcc $(CFLAGS) -o $@ principal.o ponmayusculas.o


# Compila ficheros fuente en  ficheros .o
principal.o: principal.c ponmayusculas.h
    gcc $(CFLAGS) -c principal.c

ponmayusculas.o: ponmayusculas.c ponmayusculas.h
    gcc $(CFLAGS) -c ponmayusculas.c

#### Para limpiar el directorio de ficheros generados ####
clean:
    rm -f a.out *.o
```