Memoria Práctica 3 EDAT

Alejandro Monterrubio Navarro

Héctor Martín Cuadra

Grupo 1272

**Objetivo:**

Esta práctica se basa en crear una pequeña base de datos de una manera muy simple a través de la creación de tablas y de índices. El proceso sería crear una tabla y un índice a esa tabla, después vamos insertando datos para finalmente imprimir un árbol. Para guardar la información no la guardamos en la memoria interna, sino que la vamos leyendo y vamos modificando según la necesidad.

**Algoritmos a implementar:**

**CREATETABLE:** Crea una tabla.

bool createTable(const char \* tableName) {

    FILE \*f;

    int num = -1;

    /\*Comprueba que se abre bien\*/

    if (!fopen(tableName, "r")) {

        f = fopen(tableName, "w");

        fwrite(&num, sizeof(int), 1, f);

    }

    fclose(f);

    return true;

}

**CREATEINDEX:** A partir de la tabla se crea el índice.

bool createIndex(const char \*indexName) {

    FILE \*f;

    int num = -1;

    /\*Comprueba que se abre bien\*/

    if (!fopen(indexName, "r")) {

        f = fopen(indexName, "w");

        fwrite(&num, sizeof(int), 1, f);

        fwrite(&num, sizeof(int), 1, f);

    }

    fclose(f);

    return true;

}

**PRINTTREE Y PRINTNODE:** Se va llamando a la función print node recursivamente mientras se va imprimiendo un árbol binario.

/\*Printea nodo\*/

void printnode(size\_t \_level, size\_t level, FILE \* indexFileHandler, int node\_id, char side) {

    int posicion = 0, izq = 0, der = 0, offset = 0, i = 0;

    char book\_id[PK\_SIZE];

    if (\_level > level) {

        return;

    }

    if (node\_id == -1) {

        return;

    }

    /\*Busca posición del nodo\*/

    posicion = node\_id \* (4 + 4 \* 4) + 8;

    fseek(indexFileHandler, posicion, SEEK\_SET);

    fread(book\_id, sizeof(book\_id), 1, indexFileHandler);

    fread(&izq, sizeof(int), 1, indexFileHandler);

    fread(&der, sizeof(int), 1, indexFileHandler);

    fread(&offset, sizeof(int), 1, indexFileHandler);

    fread(&offset, sizeof(int), 1, indexFileHandler);

    for (i=0; (size\_t) i < \_level; i++ )

        printf("     ");

    printf("%c %s (%d): %d\n", side, book\_id, node\_id, offset);

    /\*Printea nodos a los lados\*/

    printnode(\_level+1, level, indexFileHandler, izq, 'i');

    printnode(\_level+1, level, indexFileHandler, der, 'd');

    return;

}

/\*Printea el árbol\*/

void printTree(size\_t level, const char \* indexName) {

    FILE \*f;

    int raiz;

    if (!(f = fopen(indexName, "r+"))) {

        return;

    }

    fread(&raiz, sizeof(int), 1 , f);

    printnode(0, level, f, raiz, '  ');

    fclose(f);

    return;

}

**FINDKEY:** Busca la clave en el índice y devuelve true si la encuentra y false si no lo hace.

bool findKey(const char \* book\_id, const char \*indexName, int \* nodeIDOrDataOffset) {

    char book\_read[PK\_SIZE];

    int node, res, i;

    FILE \*f;

    \*nodeIDOrDataOffset = -1;

    if (!book\_id || !indexName) {

        return false;

    }

    if (!(f=fopen(indexName,"r+"))) {

        return false;

    }

    fread(&node,sizeof(int), 1, f);

    while (node != -1) {

        \*nodeIDOrDataOffset = node;

        fseek(f, node\*20+8, SEEK\_SET);

        fread(book\_read, sizeof(book\_read), 1, f);

        res = strncmp(book\_id,book\_read,PK\_SIZE);

        if (res == 0) {

            for (i=0; i<4; i++) {

                fread(nodeIDOrDataOffset, sizeof(int), 1, f);

            }

            fclose(f);

            return true;

        }

        else if (res < 0) {

            fread(&node, sizeof(int), 1, f);

        }

        else {

            for (i=0; i<2; i++) {

                fread(&node,sizeof(int), 1, f);

            }

        }

    }

    fclose(f);

    return false;

}

**ADDINDEXENTRY:** Actualiza la información de la estructura con lo que le manda addTableEntry.

bool addIndexEntry(char \* book\_id,  int bookOffset, char const \* indexName) {

    FILE \*f;

    int nodeIDOrDataOffset, numnodes, raiz, borrado, a = -1, siguienteBorrado;

    char padre[PK\_SIZE];

    if (!(f=fopen(indexName,"r+"))) {

        return false;

    }

    if (findKey(book\_id,indexName,&nodeIDOrDataOffset) == true) {

        fclose(f);

        return false;

    }

    fread(&raiz, sizeof(int), 1, f);

    fread(&borrado, sizeof(int), 1, f);

    if (borrado != -1) {

        fseek(f,nodeIDOrDataOffset\*20+8, SEEK\_SET);

        fread(padre, sizeof(padre), 1, f);

        if (strncmp(book\_id,padre,PK\_SIZE) < 0) {

            fwrite(&borrado, sizeof(int), 1, f);

        }

        else {

            fseek(f,ftell(f)+4,SEEK\_SET);

            fwrite(&borrado, sizeof(int), 1, f);

        }

        fseek(f, borrado\*20+12, SEEK\_SET);

        fread(&siguienteBorrado, sizeof(int), 1, f);

        fseek(f, 4, SEEK\_SET);

        fwrite(&siguienteBorrado, sizeof(int), 1, f);

        fseek(f, borrado\*20+8, SEEK\_SET);

        fwrite(book\_id, PK\_SIZE, 1, f);

        fwrite(&a, sizeof(int), 1, f);

        fwrite(&a, sizeof(int), 1, f);

        fwrite(&nodeIDOrDataOffset, sizeof(int), 1, f);

        fwrite(&bookOffset, sizeof(int), 1, f);

    }

    else {

        fseek(f, 0, SEEK\_END);

        numnodes = (ftell(f)-8)/20;

        if (raiz == -1) {

            fseek(f, 0, SEEK\_SET);

            fwrite(&numnodes, sizeof(int), 1, f);

        }

        else {

            fseek(f, nodeIDOrDataOffset\*20+8, SEEK\_SET);

            fread(padre, sizeof(padre), 1, f);

            if (strncmp(book\_id, padre, PK\_SIZE) < 0) {

                fwrite(&numnodes, sizeof(int), 1, f);

            }

            else {

                fseek(f,ftell(f)+4,SEEK\_SET);

                fwrite(&numnodes, sizeof(int), 1, f);

            }

        }

            fseek(f, 0, SEEK\_END);

            fwrite(book\_id, PK\_SIZE, 1, f);

            fwrite(&a, sizeof(int), 1, f);

            fwrite(&a, sizeof(int), 1, f);

            fwrite(&nodeIDOrDataOffset, sizeof(int), 1, f);

            fwrite(&bookOffset, sizeof(int), 1, f);

    }

    fclose(f);

    return true;

}

**ADDTABLEENTRY:** Añade la información a la estructura y se la envía a addIndexEntry para que la actualice.

bool addTableEntry(Book \* book, const char \* dataName, const char \* indexName) {

    FILE \*f;

    int borradoActual, nodeIDOrDataOffset;

    if (!(f=fopen(dataName,"r+"))) {

        fprintf(f,"Fallo al leer la clave");

        return false;

    }

    if(findKey(book->book\_id, indexName, &nodeIDOrDataOffset)==true){

        fclose(f);

        return false;

    }

    /\*Aquí iría el borrado, pero como no tengo el tester no se como hacerlo sin ver los fallos y si dejo el código de prueba me da muchos errores\*/

    fseek(f, 0, SEEK\_END);

    borradoActual=ftell(f);

    fwrite(book->book\_id, 4, 1, f);

    fwrite(&book->title\_len, 4, 1, f);

    fwrite(book->title, book->title\_len, 1, f);

    addIndexEntry(book->book\_id, borradoActual, indexName);

    fclose(f);

    return true;

}

**Interfaz de usuario (Código):** Es un menú que deja elegir entre las distintas opciones y ejecuta las funciones.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "utils.h"

#define MAX 256

int ShowMainMenu() {

    int nSelected = 0;

    char buf[16];

    do {

        printf("\n (1) use\n"

            " (2) insert\n"

            " (3) print\n"

            " (4) exit\n\n"

            "Introduce un número > ");

        if (!fgets(buf, 16, stdin))

            /\* reading input failed, give up: \*/

            nSelected =0;

        else

            /\* have some input, convert it to integer: \*/

            nSelected = atoi(buf);

        printf("\n");

        if ((nSelected < 1) || (nSelected > 4)) {

            printf("Has introducido un valor no válido. Prueba otra vez\n\n\n");

        }

    } while ((nSelected < 1) || (nSelected > 4));

    return nSelected;

}

int main(void) {

    int nChoice = 0, altura, use = 0;

    char tablename[MAX], indexname[MAX], reserva[16];

    Book book;

    char titulo[256];

    do {

        nChoice = ShowMainMenu();

        switch (nChoice) {

            case 1: {

                printf ("Introduce el nombre de la tabla: \n");

                scanf("%s",tablename);

                fgets(reserva, 16, stdin);

                strcpy(indexname,tablename);

                strcat(indexname,".idx");

                strcat(tablename,".dat");

                createTable(tablename);

                createIndex(indexname);

                use = 1;

                printf("\n");

            }

                break;

            case 2: {

                if (use == 0) {

                    printf("No has seleccionado una tabla.\n");

                }

                else {

                    printf("Introduce book id: \n");

                    scanf("%s",book.book\_id);

                    fgets(reserva, 16, stdin);

                    printf("Introduce el título: \n");

                    scanf("%s",titulo);

                    fgets(reserva, 16, stdin);

                    fflush(stdin);

                    if ((book.title = malloc(strlen(titulo))) == NULL) {

                        return false;

                    }

                    book.title\_len = strlen(titulo);

                    strncpy(book.title, titulo, book.title\_len);

                    addTableEntry(&book, tablename, indexname);

                    free(book.title);

                    printf("\n");

                }

            }

                break;

            case 3: {

                printf("Introduce la altura del árbol: \n");

                scanf("%d",&altura);

                fgets(reserva, 16, stdin);

                printTree(altura,indexname);

                printf("\n");

            }

                break;

            case 4: {

                printf("Adios\n\n");

            }

                break;

        }

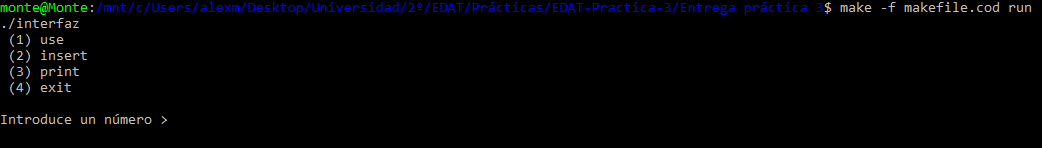
    } while (nChoice != 4);

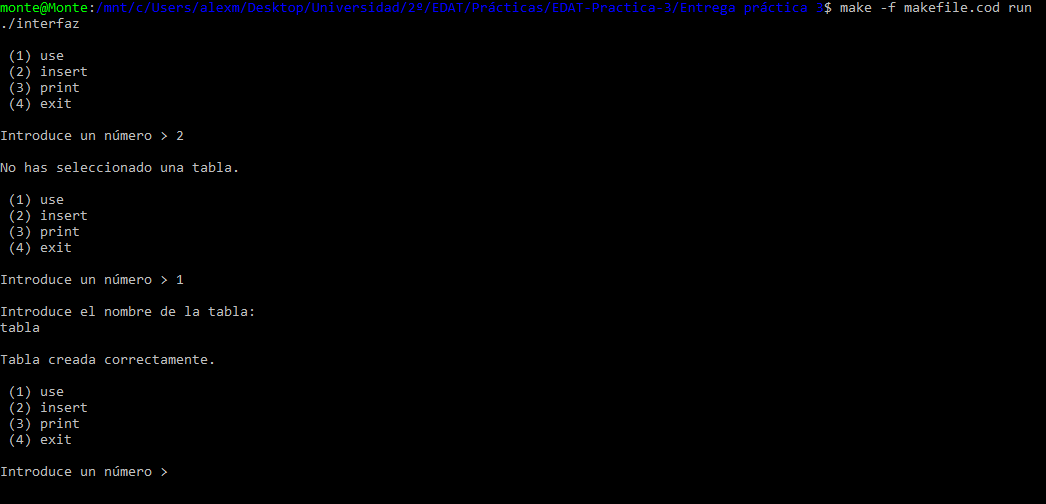
    return 0;

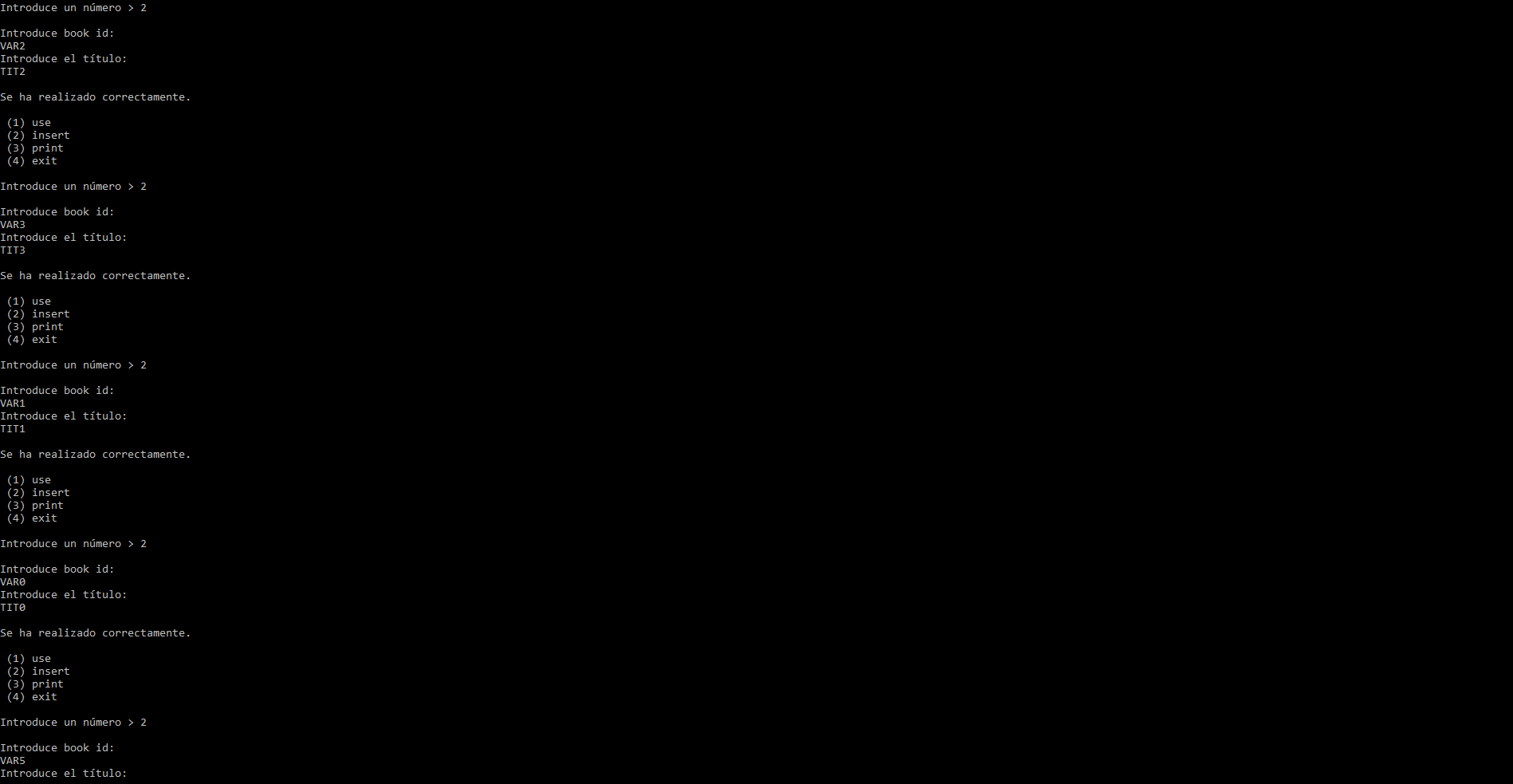
}

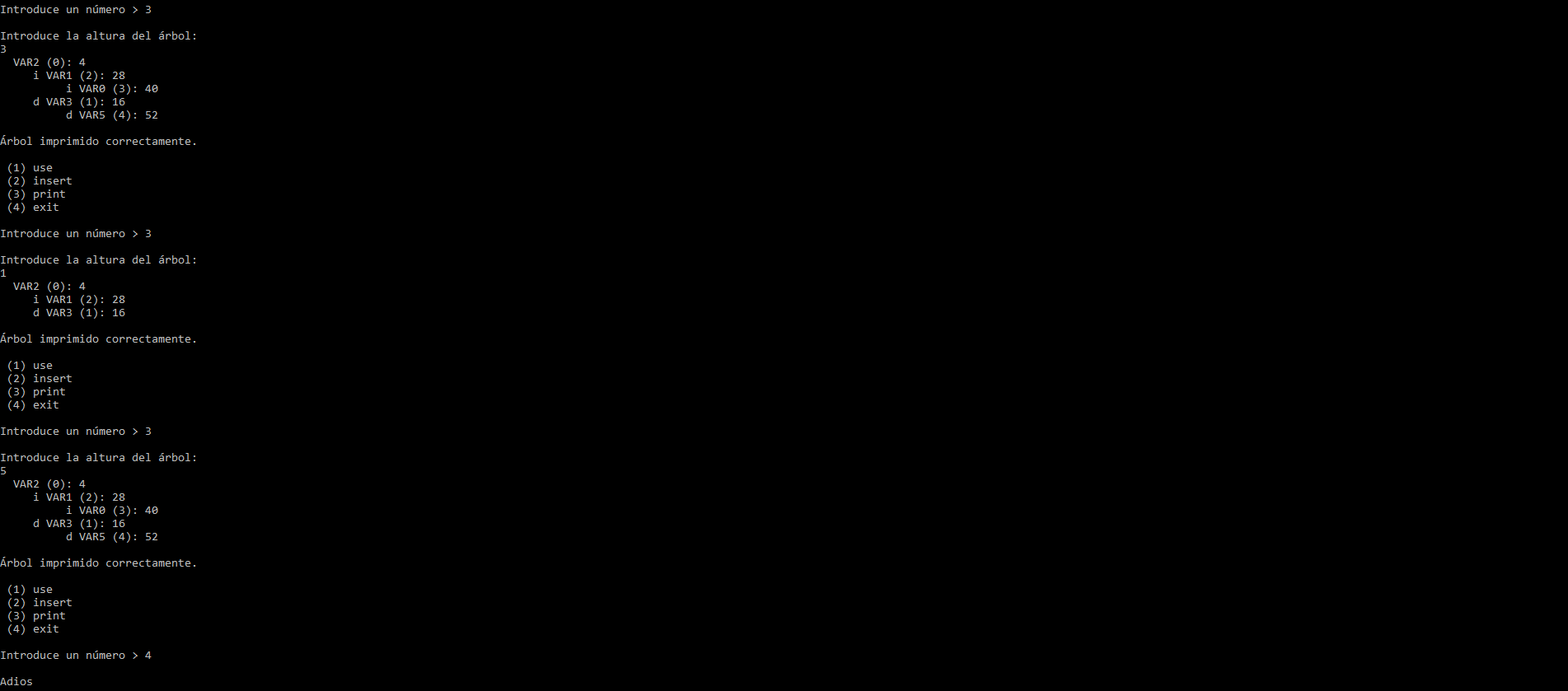
**Funcionamiento funciones y código:**

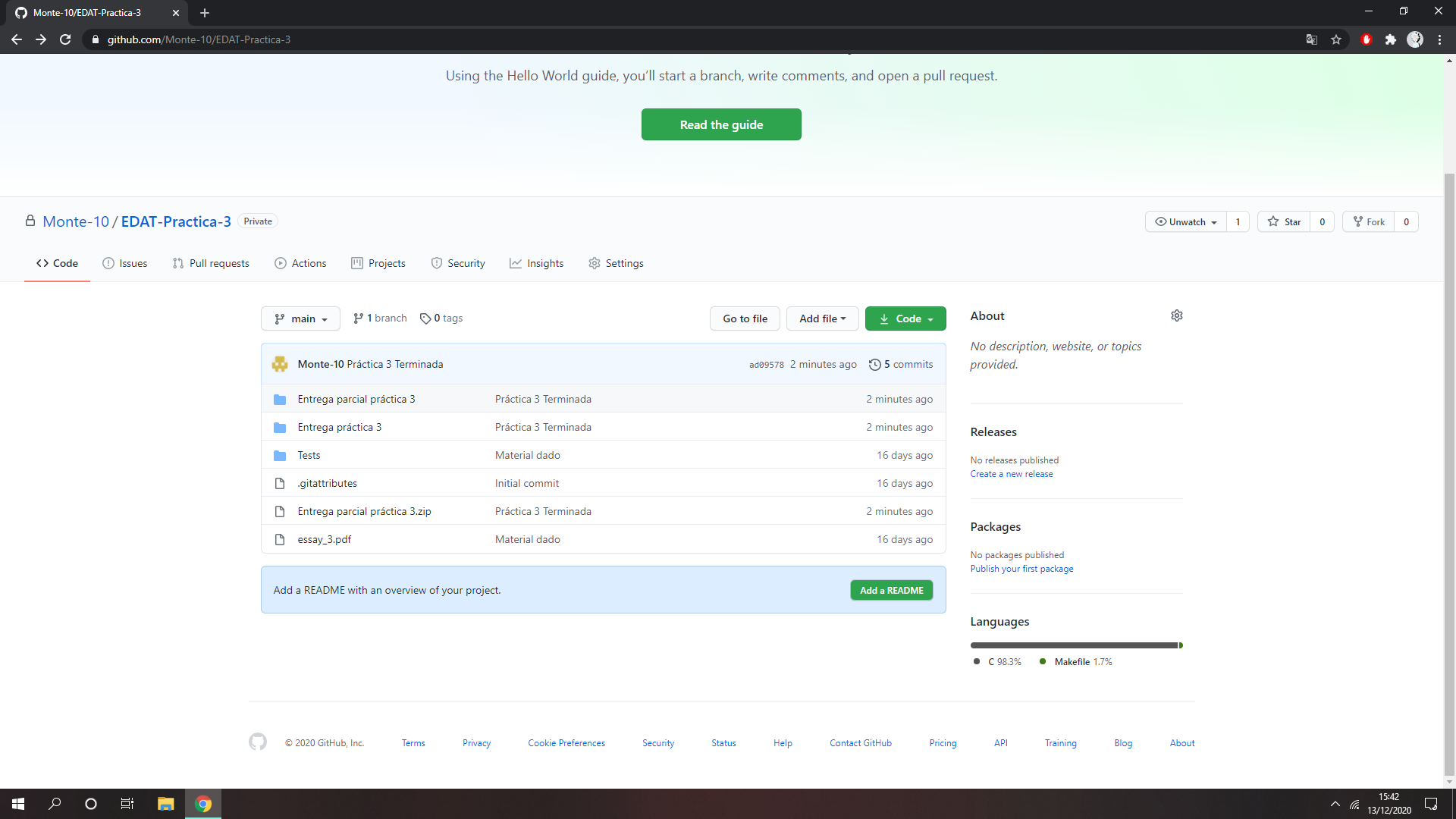
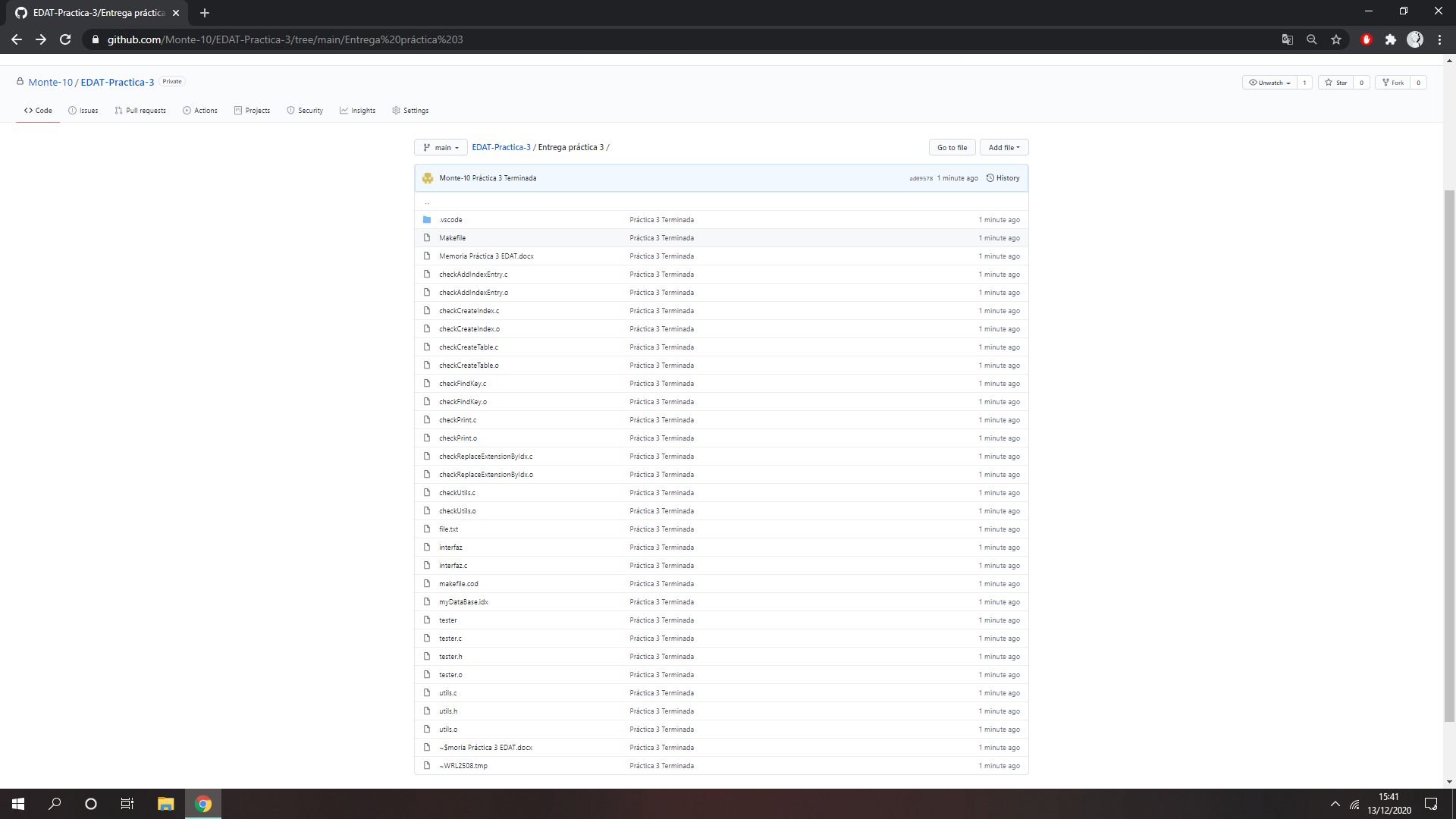
**Interfaz:**

****

**Use:**

**Insert:**

**Print:**

**Capturas Git:**