Universidad Autónoma de Madrid

Departamento de Informática

Estructura de Datos
Memorias Práctica - 2

Hecho por: Diego Rodríguez

y Alejandro García

Realización del menú y conexión a la base de datos:

El menú está basado en el que se nos ha proporcionado en el material de la práctica. Se estructura en varios submenús, todos ubicados en un fichero independiente llamado funciones.c; incluye las funciones dedicadas al menú además de algunas que son comunes a todas las consultas. Los submenús están dedicados a consultas sobre cada uno de los aspectos de la base de datos: productos, pedidos (orders) y clientes. Cada consulta tiene una función asociada que gestiona el trato con el usuario y otra que realiza la consulta, por ejemplo, ProductsStock y ProductsStockQ. Las primeras se ubican en funciones.c, mientras que las segundas se encuentran en un fichero distinto dedicado al submenú en el que se encuentran products.c, orders.c o customers.c.

Por lo tanto, en menu.c solo se realiza la conexión y la desconexión a la base de datos, para evitar conectarse y desconectarse cada vez que se realiza una consulta, y una llamada a la función Menu() que inicia el menú.

Funcionamiento del Makefile:

Siguiendo las indicaciones del enunciado el Makefile tiene dos funcionalidades. Por un lado, con el comando *make all* crea la base de datos y con el comando *make compile* compila los ficheros en C destinados a la gestión del menú. Además, incluye los comandos *make dropdb, createdb, restore y shell* para trabajar con la base de datos de forma más específica que con *make al*l, que realiza las cuatro tareas.

Implementación de las consultas:

Para la realización de las consultas nos hemos inspirado en el ejemplo obdc-example4.c. Además, para poder cambiar las consultas de SQL sin cambiar el código C, hemos creado un archivo de sql, queries.sql, donde están redactadas. Esta implementación tiene sus desventajas, por ejemplo, las consultas tienen que estar en una línea y colocadas de una forma particular. Sin embargo, permite tener todas las consultas en un mismo archivo y cambiarlas si se quieren hacer de una forma distinta, conservando el número de parámetros y de columnas de salida y sus tipos.

```
/*Archivo con las consultas ya escritas para evitar copiarlas tal cual en la función:*/
/*EN el submenu Products Consulta STOCK */
SELECT products.quantityinstock FROM products WHERE products.productcode=?
/*EN el submenu Products Consulta FIND */
SELECT products.productcode, products.productname from products where products.productname like '%?%'
/*EN el submenu Orders Consulta OPEN */
SELECT orders.ordernumber FROM orders WHERE orders.shippeddate IS NULL
/*EN el submenu Orders Consulta RANGE */
SELECT orders.ordernumber, orders.orderdate, orders.shippeddate FROM orders WHERE ? <= orders.orderdate and  orders.orderdate <= ?
/*EN el submenu Orders Consulta DETAILS */
SELECT orders.orderdate,orders.status FROM orders WHERE orders.ordernumber= ?
/*Segunda Query de Details*/
SELECT SUM(orderdetails.priceeach*orderdetails.quantityordered) FROM orderdetails WHERE orderdetails.ordernumber= ?
/*Tercera Query de Details*/
SELECT orderdetails.productcode, orderdetails.quantityordered, orderdetails.priceeach FROM orderdetails WHERE orderdetails.ordernumber = ?
/*EN el submenu Customers Consulta FIND */
SELECT customers.customername, customers.contactfirstname, customers.contactlastname, customers.customernumber FROM customers
WHERE (customers.contactfirstname like '%?%' OR customers.contactlastname like '%?%')
/*EN el submenu Customers Consulta LIST PRODUCTS */
WITH codProductos AS (SELECT orderdetails.productcode, orderdetails.quantityordered FROM orderdetails WHERE orderdetails.ordernumber IN (SELECT orders.ordernumber FROM orders where orders.customernumber=?))
SELECT products.productname, SUM(codProductos.quantityordered) FROM codProductos, products WHERE products.productcode = codProductos.productcode GROUP BY products.productcode ORDER BY products.productcode
/*EN el submenu Customers Consulta BALANCE */
WITH pagado AS (SELECT SUM(orderdetails.priceeach*orderdetails.quantityordered) AS suma FROM orderdetails WHERE orderdetails.ordernumber IN (SELECT orders.ordernumber FROM orders
WHERE orders.customernumber= ? )), comprado AS (SELECT SUM(payments.amount) AS suma FROM payments WHERE payments.customernumber = ? ) SELECT comprado.suma-pagado.suma from comprado, pagado
```

La razón por la que se debe mantener el número y el tipo de parámetros de entrada es que las consultas se realizan utilizando las siguientes funciones del fichero odbc.c: SQLBindParameter() y SQLBindCol(). Para realizar una consulta es necesario llamar a la primera por cada parámetro de entrada y a la segunda para cada columna de retorno.

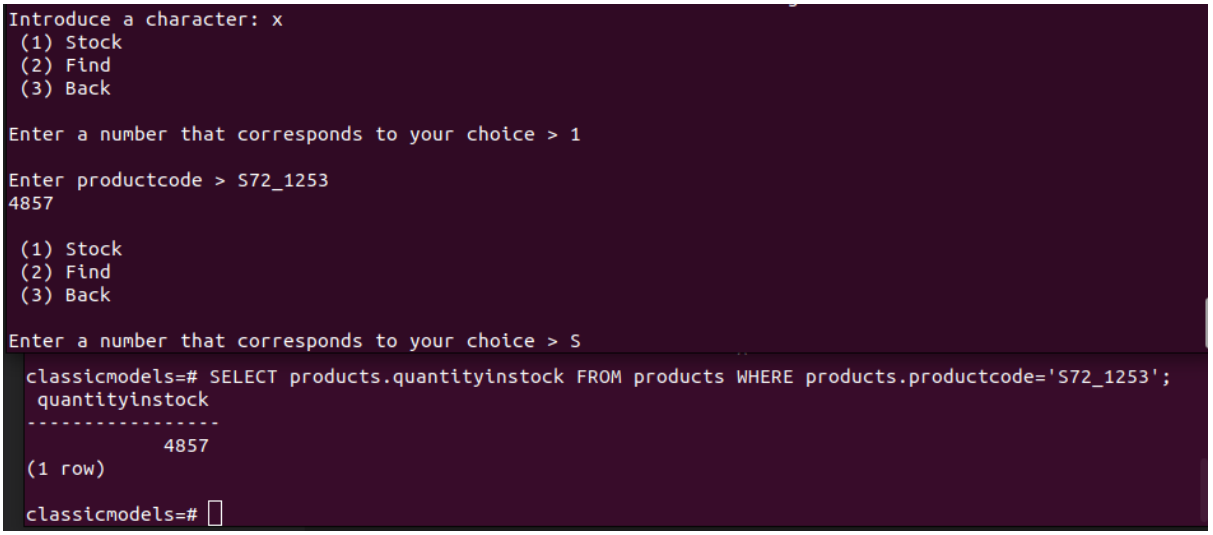
Por otro lado, la lectura de las consultas la realiza la función readQuery()que las lee del archivo queries.sql. Esta función toma como parámetros un número que indica la consulta que se quiere realizar y un puntero a char donde se almacena la query.

```
int readQuery(int numQ, char *querr){
    int i;
    FILE *f=NULL;

    if(numQ<1||!querr){
        printf("Error al leer la consulta");
        return ERROR;
    }

    f=fopen(nombre, "r");

    if(!f){ /*Control de errores*/
        printf("Error");
        return ERROR;
    }

    for(i=0; i<(1+2*numQ); i++) (void)fgets(querr, TAM, f); /*Almacena en query la query deseada*/

   (void) fclose(f);

    return OK;
}
```

Otro aspecto común entre todas las consultas es el uso de initQuery()y endQuery()que son las encargadas de inicializar la query y cerrarla, es decir reserva memoria para el handle (puntero a la consulta en SQL) y la libera. Su funcionalidad es reservar memoria para el handle (puntero a la consulta) y liberar dicho puntero, respectivamente. A parte de eso la estructura básica de cada consultas es: leer la consulta con readQuery(), inicializarla con initQuery, introducir los parámetros con SQLBindParameter(), llamar a la función SQLExecute(), extraer las columnas SQLBindCol(), extraer los resultados usando SQLFetch() y cerrar la query primero con SQLCloseCursor()y luego con endQuery(). En este ejemplo de la consulta Balance del submenú customers, se aprecia este formato.

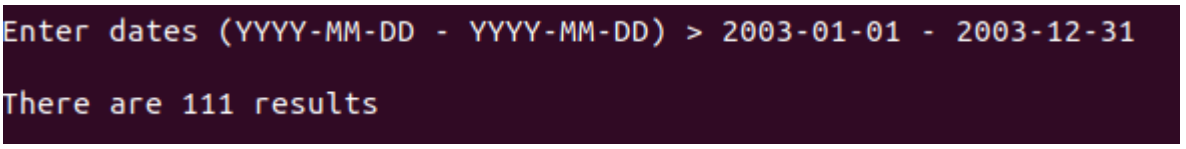
```c
void ProductsStockQuery(SQLHDBC dbc,char * productcode) {
    char cantidad[BufferLength]="\0"; /* odbc.c */
    char *querr=NULL, aux;
    int ret=-1;
    SQLHSTMT stmt = NULL;

    fflush(stdout);
    /*Función hacer query extraer el número de unidades de el productcode*/
    querr=(char*)calloc(TAM, sizeof(char));
    if(readQuery(QueryStockProducts, querr)==ERROR){ /*Control de errores*/
        printf("Error al leer la consulta");
        return ;
    }

    /*Inicilizar*/
    stmt=initQuery(dbc, querr);
    if(stmt==NULL){
        free(querr);
        printf("Error al almacenar la query FindStock");
        return;
    }

        (void) SQLBindParameter(stmt, 1, SQL_PARAM_INPUT,
                                SQL_C_CHAR, SQL_CHAR,
                                0, 0, productcode, BufferLength, NULL);

        (void) SQLExecute(stmt);

        (void) SQLBindCol(stmt, 1, SQL_C_CHAR, (SQLCHAR*) cantidad, BufferLength, NULL);
        /* Loop through the rows in the result-set */
        if (SQL_SUCCEEDED(ret = SQLFetch(stmt))) {

            printf("%s\n\n", cantidad);
        }
        else{
            printf("Error al ejecutar la query");
        }

    /*Cerrar esto*/
    (void) SQLCloseCursor(stmt);
    scanf("%c", &aux);
    free(querr); /*Liberamos la memoria empleada*/


    if(-1==endQuery(stmt)) printf("Error al liberar stmt");
```

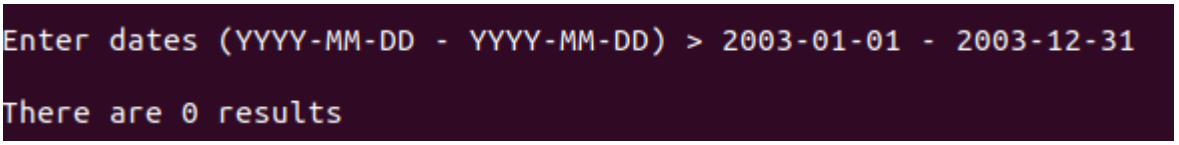
Como método general de comprobación, hemos realizado las consultas en SQL y hemos cotejado el resultado con el del programa en C. Tras varias pruebas de este estilo, podemos asegurar que la implementación en C de las consultas es correcta. Sin embargo, aún nos faltaría comprobar el funcionamiento de la consulta en SQL, para eso recurrimos al método de comprobación de la práctica anterior, pequeñas modificaciones de la base de datos probando casos límite. Tristemente, es difícil asegurar que una consulta de SQL funciona perfectamente para cualquier estado de la relación.



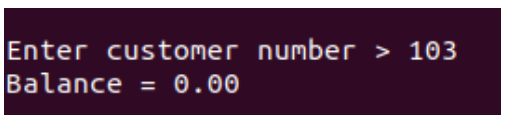
Ejemplo de la comprobación de la consulta ProductStock hecha en SQL y en menu.c.

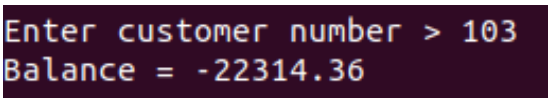
A continuación mostramos algunos ejemplos de comprobaciones, por ejemplo para asegurarnos de que Orders Range funciona adecuadamente hemos cambiado algunas fechas del año 2003 a 2004 con(UPDATE orders SET orderdate='2004-01-01' WHERE orders.orderdate <'2003-12-31' and orders.orderdate > '2003-01-01'), y como se puede ver el número de pedidos realizados en 2003 se reduce a 0 y de esta forma lo refleja la query.



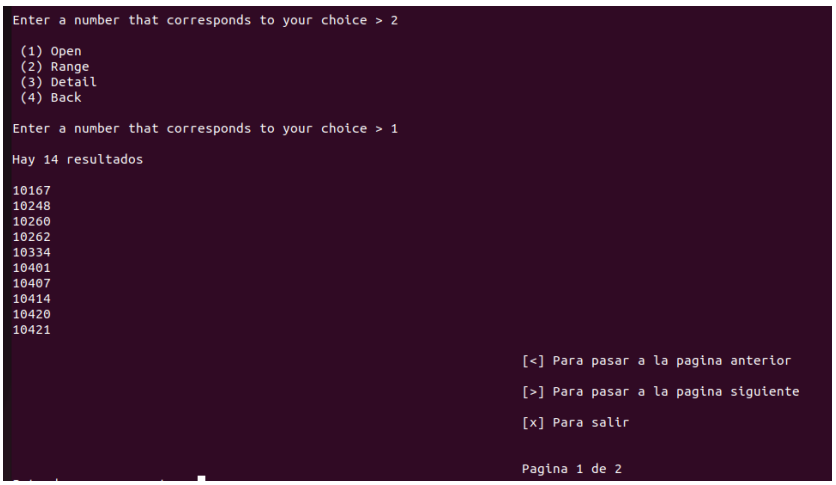


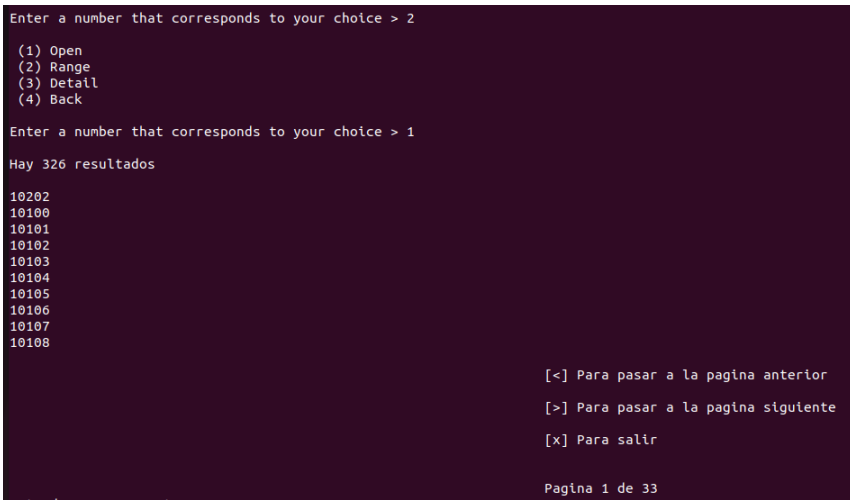
Otra comprobación ha sido en la consulta Customer Balance, cambiar todos los pagos de un cliente a 0 con (UPDATE payments SET amount=0 WHERE payments.customernumber=103) como se puede apreciar solo aparecen sus compras en negativo:



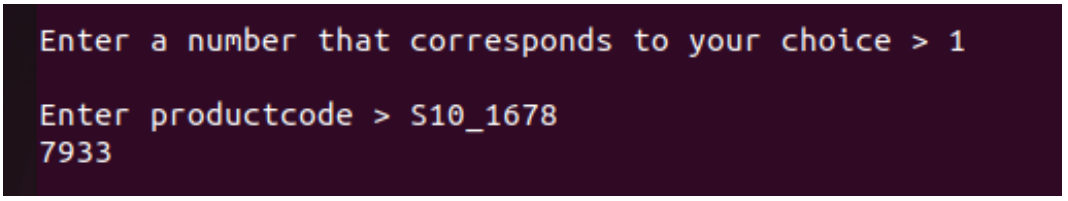


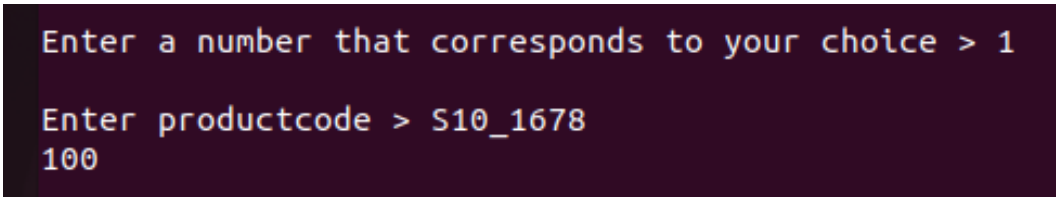
Para comprobar la función Open hemos hecho que todos los pedidos se almacenen como pedidos abiertos (UPDATE orders SET shippeddate=NULL;). Tras ello, vemos como el número de resultados aumenta de 14 a 326:





Por otro lado, para asegurarnos de que FindStock funciona correctamente, únicamente hemos cambiado la cantidad de producto disponible de aquel asociado al código S10_1678 (UPDATE products SET quantityinstock=100 WHERE products.productcode=S10_1678;). Vemos que el programa indica la nueva cantidad correctamente:





Como hemos dicho antes las consultas se dividen en tres submenús distintos, ahora veremos los distintos desafíos que nos hemos enfrentado al realizar cada consulta.

Submenú Products:

Products Stock:

Esta fue la primera consulta que implementamos, por lo que le dedicamos bastante tiempo a familiarizarnos con las nuevas funciones del fichero odbc.c y la conversión entre tipos de SQL a C. Tras mucho trabajo tratando de entender todos los parámetros de entrada de las funciones SQLBindParameter() y SQLBindCol(), fuimos capaces de realizar con éxito nuestra primera consulta.

Products Find:

Esta consulta además de devolver dos columnas, no devuelve una única fila como hacía la anterior, si no varias tupla. Esto provoca que en vez de usar un if para comprobar si la consulta había encontrado algún resultado, tuvimos que usar un while para recorrer la tabla e imprimir dichas filas. Esta es la primera función en la que aplicamos la paginación luego debemos reservar memoria para el resultado, debido a las características de la base de datos nunca va a ser necesario llamar a la función AumentarReserva() encargada de reservar más memoria en caso de que fuese necesario. Dicha memoria se libera dentro de la función paginaciónyLiberacion(), encargada de imprimir la tabla resultado.

Submenú Orders:

Orders Open:

Quizás esta fuese la consulta más sencilla de realizar al no tener parámetros de entrada y haber realizado la consulta anterior con la que aprendimos a devolver columnas con más de una tupla. Por tanto no supuso demasiado esfuerzo. En esta función también hemos usado la paginación.

Orders Range:

En este caso, fuera de la realización de la consulta en SQL, la implementación en C tampoco incluye nada que no hubiésemos hecho en alguna de las anteriores. Aunque, esta es una consulta muy completa con dos parámetros de entrada diferentes y devuelve tres columnas con varias tuplas. Además, aprendimos a sustituir los valores NULL, por un espacio en blanco a la hora de imprimir los resultados en la función paginaciónyLiberacion().

Orders Details:

Mientras que las anteriores consultas de este submenú no han sido demasiado complicadas, esta sí supuso más trabajo. En primer lugar, para extraer todos los datos que nos piden la dividimos en tres

consultas distintas en SQL, por tanto dentro de esta función se realizan tres queries distintas cuyos retornos luego se gestionan en C. En segundo lugar, es importante para hacer un uso eficiente de la memoria, cerrar y liberar, con la función endQuery(), los punteros (handle) a la query para poder reutilizar la variable para la siguiente consulta.

Submenú Customers:

Customers Find:

Tras haber realizado ya seis consultas no necesitamos demasiado tiempo para realizar esta query. En esta consulta volvemos a utilizar la función PrepararQuery(), ya que en nuestra implementación de la consulta utilizamos un LIKE.

Customers List Products:

Esta es otra consulta donde la mayor complicación está en el código SQL, por tanto para asegurarnos del correcto funcionamiento del a query la probamos en pgAdmin. En esta consulta la implementación sigue el modelo ya descrito y no presenta ninguna novedad importante respecto a las consultas ya realizadas.

Customers Balance:

Esta última consulta es muy similar a las demás. Su única peculiaridad es el uso del tipo *double* para ser capaz de almacenar los decimales. Aparte de esto, en la consulta se introduce el mismo parámetro dos veces por lo que necesitamos llamar dos veces a SQLBindParameter().

Añado la un pago de 10 euros al cliente 103

Paginación:

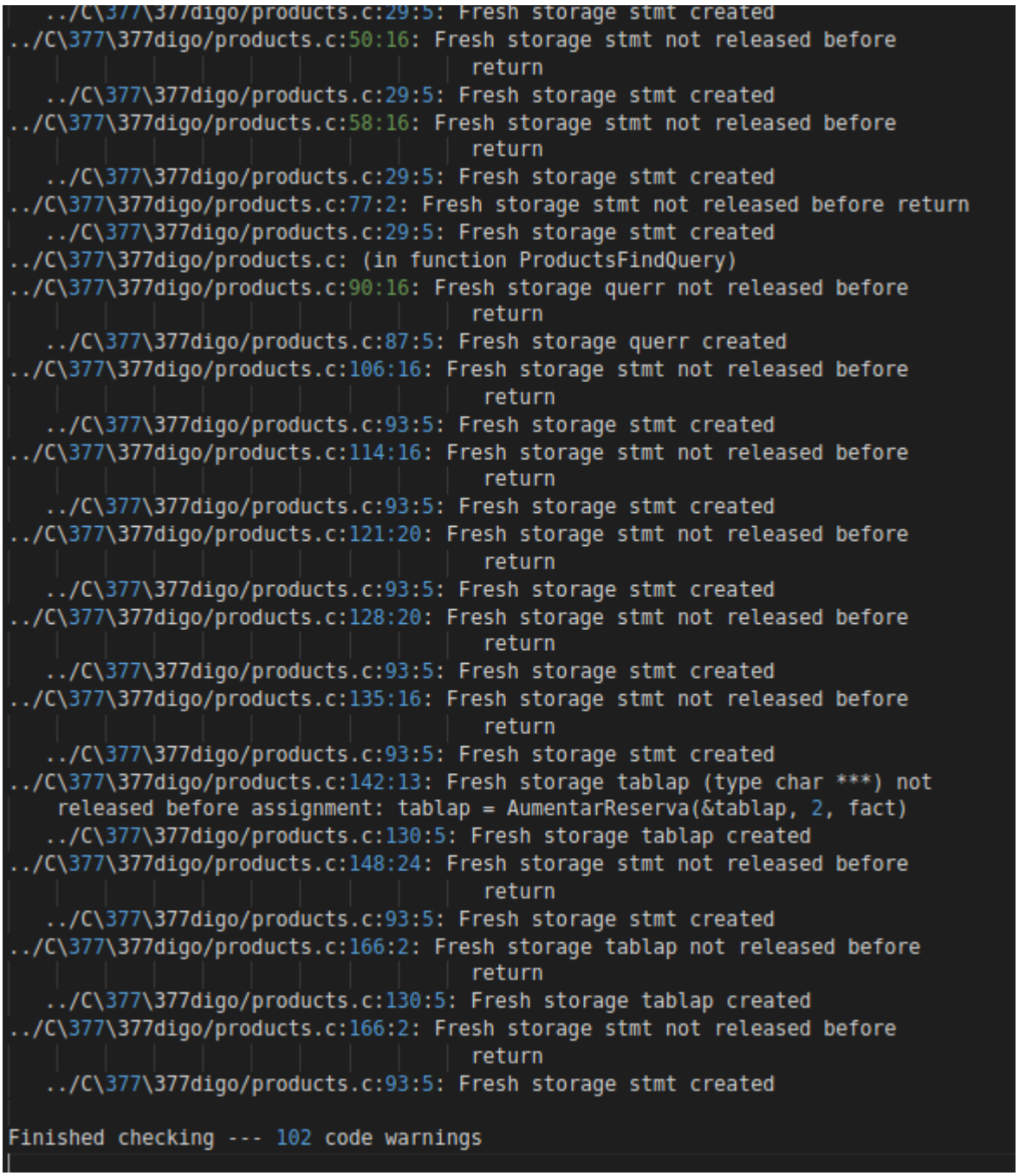
Con el objetivo de hacer una interfaz más atractiva para el usuario, hemos desarrollado una función llamada paginaciónyLiberacion(). Con ella los resultados de las consultas se muestran en grupos de diez. Cada grupo ocupa una página, se cambia la página usando los símbolos < y > , y si se quiere salir se introduce x. Es posible que los resultados los imprima en grupos más pequeños o más grandes alterando la macro pagTAM =10. Esta función se ubica en el fichero funciones.c y toma como argumentos un puntero a la tabla donde están copiados todos los resultados, el número de columnas, el de filas y un factor que se usará en la función freeTabla que libera la tabla y necesita saber el número de filas reservadas (TAM*fact) donde TAM=1000.

```c
void paginacionyLiberacion(char***ptabla,int c, int f,int fact){
    int i, j, pag=0;
    char ***tabla=NULL;
    char letra[BufferLength];

    tabla=ptabla;
    printf("\nThere are %d results\n\n", f);
    if(f<=pagTAM){ /*Si hay pagTAM filas o menos*/
        for(i=0; i<f; i++){
            for(j=0; j<c; j++){
                printf("%s ", tabla[j][i]);
            }
            printf("\n");
            }
        printf("\n");
        freeTabla(ptabla,c,fact);
        return;
    }
    else{ /*Si hay mas de pagTAM filas*/
        do{
            for(i=pag; (i<f) && (i<pag+pagTAM); i++){
                for(j=0; j<c; j++){
                    printf("%s ", tabla[j][i]);
                }
                printf("\n");
            }

            printf("\n                                                    [<] Move to the previous page\n");
            printf("\n                                                    [>] Move to the next page\n");
            printf("\n                                                    [x] Exit\n\n");
            printf("\n                                                    Page %d out of %d\n", pag/pagTAM+1, (f-1)/pagTAM+1);
            printf("Introduce a character: ");



            if(!fgets(letra, BufferLength, stdin)) return;

            while ((letra[0]!='<')&&(letra[0]!='>')&&(letra[0]!='x')){
                printf("\nWrong character");
                printf("\n                                                    [<] Move to the previous page\n");
                printf("\n                                                    [>] Move to the next page\n");
                printf("\n                                                    [x] Exit\n\n");
                printf("\n                                                    Page %d out of %d\n", pag/pagTAM+1, (f-1)/pagTAM+1);
                printf("Introduce a character: ");
                if(!fgets(letra, BufferLength, stdin)) return;
            }
            if(letra[0]=='>'){
                pag+=pagTAM;
                if(pag>=f) pag=0;
            }
            else if(letra[0]=='<'){
                pag-=pagTAM;
                if(pag<0) pag=f-f%pagTAM;
                if(pag==f) pag=f-10;
            }

        }while(letra[0]!='x');
    }
    freeTabla(ptabla,c,fact);
}
```

Comentarios:

Adicionalmente tras analizar nuestro código con *splint* vemos que nos devuelve 102 avisos, casi todos de ellos derivados de la reserva dinámica de memoria que redimensiona una tabla, otros pocos se originan en el archivo odbc.c no hemos querido cambiar.

Por otro lado, para comprobar que los errores de manejo de memoria que indicaba *splint* sobre la gestión de memoria estaban solucionados, hemos utilizado *valgrind* y nos hemos cerciorado que nuestro programa no tiene fugas a pesar de que odbc.c si las tenga.



Conclusión:

En esta práctica hemos aprendido a implementar un sencillo menú para que una base de datos sea más accesible para los usuarios. Además, hemos adquirido nociones básicas acerca del uso de la interfaz de programación ODBC para acceder a una base de datos de forma indirecta. La práctica nos ha permitido atisbar la labor que se realiza en la vida real, ya que la mayoría de usuarios deben ser capaces de realizar consultas sin tener que conocer ni el modelo relacional de la base de datos ni el lenguaje SQL.