Estructuras de datos II

Manual de funciones proyecto 2

Presentado por:

Jhon Sebastian Nieto Gil

Ricardo Andrés Villalobos Marulanda

Corporación de Estudios Tecnológicos (COTECNOVA)

Programa: Ingeniería en sistemas

Profesor (a):

Carlos londoño

Cartago

2017

Función main: Función principal del sistema, inicializa el proyecto llama a la función menú.

```c
/* Se cargan las libreria y la función principal del proyecto.
*  Fecha 18/10/2017
*  Elaborado por: John Sebastian Nieto gil
*  Elaborado por: Ricardo Andres Villalobos
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "pilaAndCola.h"
#include "menu.h"

int main()
{
    menu();
}
```

Void menú: Muestra un menú para las opciones correspondientes de estructuras pilas y colas:

Se debe seleccionar una opción de las 3 allí correspondientes.

```c
/**
 * Muestra un menu para las opciones correspondientes
 * de estructuras pilas y colas
 */
void menu()
{
    Nodo * pila = NULL;
    Nodo * inicio = NULL;
    Nodo * fin = NULL;

    int opcion = 0;
    int subOpcion = 0;
    int opcionEstructura = 0;
    int opCantidad = 0;
    int copiaOpcion = 0;
    int cantidad = 0;
    int dato = 0;
    char nombre[14];


    do
    {
        system("clear");
        printf("\n\t---Menu principal--");
        printf("\n 1. Pilas");
        printf("\n 2. Colas");
        printf("\n 0. Salir");
        printf("\n\nOpcion: ");
        scanf("%d", &opcion);
```

Despliega una serie de opciones tanto para las estructuras de pilas como las de colas y retorna la opcion seleccionada.

```
/**
 * Imprime las operaciones de la estructura
 * y retorna la seleccionada
 */
int menuEstructura()
{
    int opcion = 0;
    system("clear");
    printf("\n\t\t\t---MENU---");
    printf("\n\t1. Generar Elementos");
    printf("\n\t2. Cargar Elementos");
    printf("\n\t3. Editar Elementos");
    printf("\n\t4. Eliminar Elementos");
    printf("\n\t5. Eliminar estructura");
    printf("\n\t6. Listar Elementos");
    printf("\n\t7. Buscar Elementos");
    printf("\n\t8. Ordenar estructura");
    printf("\n\t9. Salvar Datos");
    printf("\n\t10. Restaurar Datos");
    printf("\n\t0. Salir");
    printf("\n\n\tIngrese una opcion\n");
    scanf("%d", &opcion);

    return opcion;
}
```

Imprime las opciones con las cantidades de datos que servirán para generar o cargar los datos posteriores y esta retorna la opción seleccionada.

```
/**
 * Imprime as opciones con las cantidades
 * y retorna la seleccionada
 */
int menuCantidad()
{
    system("clear");
    int opcion = 0;
    printf("\n\t\t---Cantidad---");
    printf("\n 1. Un millon");
    printf("\n 2. Dos millones");
    printf("\n 3. Cinco millones");
    printf("\n 4. Diez millones");
    printf("\n 5. Veinte millones");
    printf("\n\nOpcion: ");
    scanf("%d", &opcion);

    return opcion;
}
```

Despliega dos opciones con los métodos de ordenamiento y retorna el seleccionado.

```c
/**
 * Imprime dos opciones con los metodos de ordenamiento
 * y retorna el seleccionado
 */
int menuOrdenar()
{
    system("clear");
    int opcion = 0;
    printf("\n\t\t---Metodo de ordenamiento---");
    printf("\n 1. Metodo directo(burbuja)");
    printf("\n 2. Metodo Rapido");
    printf("\n 0. Salir");
    printf("\n\nOpcion: ");
    scanf("%d", &opcion);

    return opcion;
}
```

Retorna la cantidad según la opción seleccionada anteriormente en el menú cantidad.

```c
/**
 * Retorna una cantidad segun la opcion escogida
 */
int opcionesCantidad(int opcion)
{
    int cantidad = 0;
    switch(opcion)
    {
        case 1:
            cantidad = 1000000;
            break;
        case 2:
            cantidad = 2000000;
            break;
        case 3:
            cantidad = 5000000;
            break;
        case 4:
            cantidad = 10000000;
            break;
        case 5:
            cantidad = 20000000;
            break;
    }

    return cantidad;
}
```

Copia en un arreglo de caracteres el nombre del archivo según la opción escogida anteriormente en el menú cantidad.

```c
/**
 * copia un nombre de archivo en un arreglo de caracteres
 * según una opcion escogida
 */
void nombreArchivo(int opcion, char *nombre)
{
    switch(opcion)
    {
        case 1:
            strcpy(nombre, "archivos/1millon.txt");
            break;
        case 2:
            strcpy(nombre, "archivos/2millones.txt");
            break;
        case 3:
            strcpy(nombre, "archivos/5millones.txt");
            break;
        case 4:
            strcpy(nombre, "archivos/10millones.txt");
            break;
        case 5:
            strcpy(nombre, "archivos/20millones.txt");
            break;
    }
}
```

Se definen las simplificaciones para reservar MEMORIA y mostrar el tiempo de procesamiento para las funciones.

Se establecen el prototipo de las funciones

```
/* Se definen las funciones para las operaciones correspondientes a pilas y colas.
*  Fecha 18/10/2017
*  Elaborado por: John Sebastian Nieto gil
*  Elaborado por: Ricardo Andres Villalobos
*/
#define RESERVAR_MEMORIA (Nodo *)malloc(sizeof(Nodo))
#define MENSAJE_TIEMPO printf("\nTiempo transcurrido: %.0f milisegundos.", ((double)clock() - tiempoInicio) / CLOCKS_PER_SEC * 1000)

struct Nodo
{
    int dato;
    Nodo* siguiente;
};

void insertarElementoPila(int, Nodo *&);
void insertarDatosCola(int, Nodo* &, Nodo* &);
void eliminarElementoPila(Nodo *&);
void eliminarElementoCola(Nodo* &, Nodo* &);
void eliminarPila(Nodo *&);
void eliminarCola(Nodo* &, Nodo* &);
void listarElementos(Nodo *);
void buscarElementos(Nodo *, int);
void editarElemento(Nodo *&, int);
bool estructuraVacia(Nodo *);
int pedirDato();
void cargarDatosPila(char*, Nodo* &);
void generarDatosPila(int, Nodo* &);
void cargarDatosCola(char*, Nodo* &, Nodo* &);
void generarDatosCola(int, Nodo* &, Nodo* &);
void ordenarDatosDirecto(Nodo* &);
void ordenarDatosRapido(Nodo * &);
int salvarDatos(Nodo* , int);
int selectDigit(int, int);
void iniciarArreglo(Nodo* [], int);
```

Cargar los datos desde un archivo específico a una pila.

```
/**
 * Carga los datos desde un archivo a una pila
 */
void cargarDatosPila(char* nombreArchivo, Nodo* &estructura)
{
    clock_t tiempoInicio;
    tiempoInicio = clock();
    FILE * archivo;
    archivo = fopen(nombreArchivo, "rb");
    int dato;
    while(fread(&dato, sizeof(dato), 1, archivo))
    {
        insertarElementoPila(dato, estructura);
    }

    printf("\nSe cargaron correctamente los datos\n");
    MENSAJE_TIEMPO;
}
```

Genera datos aleatorios según la cantidad recibida y los inserta en una pila.

```
/**
 * Genera datos aleatorios y los ingresa en una pila
 */
void generarDatosPila(int cantidad, Nodo* &estructura)
{
    clock_t tiempoInicio;
    tiempoInicio = clock();
    int dato = 0;

    srand(clock());
    for(int i = 0; i < cantidad; i+=1)
    {
        dato = (rand()%1999999 - 999999);
        insertarElementoPila(dato, estructura);
    }

    printf("\nSe generaron correctamente %d de datos\n", cantidad);
    MENSAJE_TIEMPO;
}
```

Cargar los datos desde un archivo específico a una cola.

```
/**
 * Carga los datos desde un archivo a una cola
 */
void cargarDatosCola(char* nombreArchivo, Nodo* &estructura, Nodo* &fin)
{
    clock_t tiempoInicio;
    tiempoInicio = clock();
    FILE * archivo;
    archivo = fopen(nombreArchivo, "rb");
    int dato;
    while(fread(&dato, sizeof(dato), 1, archivo))
    {
        insertarDatosCola(dato, estructura, fin);
    }
    printf("\nSe cargaron correctamente los datos\n");
    MENSAJE_TIEMPO;
}
```

Genera datos aleatorios según la cantidad recibida y los inserta en una cola.

```
/**
 * Genera datos aleatorios y los ingresa en una pila
 */
void generarDatosCola(int cantidad, Nodo* &estructura, Nodo* &fin)
{
    clock_t tiempoInicio;
    tiempoInicio = clock();
    int dato = 0;

    srand(clock());
    for(int i = 0; i < cantidad; i+=1)
    {
        dato = (rand()%1999999 - 999999);
        insertarDatosCola(dato, estructura, fin);
    }
    printf("\nSe generaron correctamente %d de datos\n", cantidad);
    MENSAJE_TIEMPO;
}
```

Inserta un nuevo elemento al inicio de la pila

```
/**
 * Insertar un nuevo elemento al inicio de la pila
 */
void insertarElementoPila(int dato, Nodo* &pila)
{
    Nodo* nuevoNodo = RESERVAR_MEMORIA;

    nuevoNodo->dato = dato;
    nuevoNodo->siguiente = pila;
    pila = nuevoNodo;
}
```

Inserta un nuevo elemento al inicio de la cola.

```
/**
 * Inserta un nuevo elemento al fina de la cola
 */
void insertarDatosCola(int numero, Nodo* &inicio, Nodo* &fin)
{

    Nodo * nuevoNodo = RESERVAR_MEMORIA;
    nuevoNodo->dato = numero;
    nuevoNodo->siguiente = NULL;

    if(estructuraVacia(inicio))
    {
        inicio = nuevoNodo;
    }
    else
    {
        fin->siguiente = nuevoNodo;
    }
    fin = nuevoNodo;
}
```

Busca un elemento en la pila o cola según el dato enviado en el menú principal. Si lo encuentra imprime su posición y la dirección de memoria., Luego pregunta si desea editar el dato o seguir buscando otra coincidencia.

```
/*
 * Editar un elemento en la estructura(pila, cola)
 */
void editarElemento(Nodo* &estructura, int dato)
{
    Nodo* aux = estructura;
    int contador = 0;
    int nuevoDato = 0;
    int posicion = 0;
    char comprobar;

    if(estructuraVacia(estructura))
    {
        printf("\nPila vacia");
    }
    else
    {
        while(aux != NULL)
        {
            if(aux->dato == dato)
            {
                contador++;
                printf("\n %d dato  encontrado, posicion: %d, direccion en memoria: %p", contador, posicion+1, &aux->dato);
                printf("\n¿Desea editar este elemento?. S o N: ");
                getchar();
                scanf("%c", &comprobar);
                if(comprobar == 's' || comprobar == 'S')
                {
                    nuevoDato = pedirDato();
                    aux->dato = nuevoDato;
                    break;
                }
            }
            posicion += 1;
            aux = aux->siguiente;
        }
        if(contador == 0)
        {
            printf("\nDato no encontrado");
        }
    }
```

Elimina el primer elemento insertado en la pila

```
/**
 * Eliminar el primer elemento de la pila
 */

void eliminarElementoPila(Nodo* &pila)
{
    Nodo *aux = pila;

    if(estructuraVacia(pila))
    {
        printf("\nPila vacia");
    }
    else
    {
        pila = aux->siguiente;
        free(aux);
    }
}
```

Elimina el primer elemento insertado en la cola.

```
/**
 * Elimina el primer elemento de la cola
 */
void eliminarElementoCola(Nodo* &inicio, Nodo* &fin)
{
    Nodo* aux = inicio;
    if(estructuraVacia(inicio))
    {
        printf("\nCola vacia");
    }
    else
    {
        if(inicio == fin)
        {
            inicio = NULL;
            fin == NULL;
        }
        else
        {
            inicio = inicio->siguiente;
        }
        free(aux);
    }
}
```

Elimina todos elementos ingresados en la pila.

```
/**
 * Eliminar completamente la pila
 */

void eliminarPila(Nodo* &pila)
{
    clock_t tiempoInicio;
    tiempoInicio = clock();

    if(estructuraVacia(pila))
    {
        printf("\nPila vacia");
    }
    else
    {
        while(pila != NULL)
        {
            eliminarElementoPila(pila);
        }
    }


    printf("\nSe elimino la pila\n");
    MENSAJE_TIEMPO;
}
```

Elimina todos elementos ingresados en la cola.

```
/**
 * Elimina todos los elementos de la cola.
 */
void eliminarCola(Nodo* &inicio, Nodo* &fin)
{
    clock_t tiempoInicio;
    tiempoInicio = clock();

    if(estructuraVacia(inicio))
    {
        printf("\nPila vacia");
    }
    else
    {
        while(inicio != NULL)
        {
            eliminarElementoCola(inicio, fin);
        }
    }
    printf("\nSe elimino la Cola\n");
    MENSAJE_TIEMPO;
}
```

Lista todos los elementos de una estructura. Si se va a listar una pila se debe enviar el nodo pila y si se va enviar una cola se enviar el nodo inicio cola.

```
/**
 * Muestra todos los elementos de la estructura(pila, cola)
 */

void listarElementos(Nodo* estructura)
{
    clock_t tiempoInicio;
    tiempoInicio = clock();

    Nodo *aux = estructura;

    if(estructuraVacia(estructura))
    {
        printf("\nLa estructura esta vacia");
    }
    else
    {
        while(aux != NULL)
        {
            printf("\n %d", aux->dato);
            aux = aux->siguiente;
        }
    }
    MENSAJE_TIEMPO;
}
```

Busca un elemento en la pila o cola según el dato enviado en el menú principal. Si lo encuentra imprime su posición y la dirección de memoria, Luego continua buscando otra coincidencia

```c
/**
 * Busca un elemento en la estructura(pila, cola)
 */

void buscarElementos(Nodo* estructura, int dato)
{
    clock_t tiempoInicio;
    tiempoInicio = clock();

    Nodo *aux = estructura;
    int contador = 0;
    int posicion = 0;

    if(estructuraVacia(estructura))
    {
        printf("Pila vacia\n");
    }
    else
    {
        while(aux != NULL)
        {
            if(aux->dato == dato)
            {
                contador += 1;
                printf("\n %d dato  encontrado, posicion: %d, direccion en memoria: %p", contador, posicion+1, &aux->dato);
                //break;
            }

            aux = aux->siguiente;
            posicion += 1;
        }
        if(contador == 0)
        {
            printf("\nNo se encontro el dato");
        }
    }
    MENSAJE_TIEMPO;
}
```

Comprueba si la estructura es nula.

```c
/**
 * Comprobar si la estructura(pila, cola) esta vacía
 */

bool estructuraVacia(Nodo *estructura)
{
    if(estructura == NULL)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Solicita un dato y lo retorna.

```
/**
 * Ingresa un nuevo dato solicitado por teclado
 */

int pedirDato()
{
    int dato;

    printf("\nIngrese un dato: ");
    scanf("%d", &dato);

    return dato;
}
```

Ordena la estructura revisando cada elemento de la lista que va a ser ordenada con el siguiente , intercambiándolos de posición hasta que finalice su ciclo On² (Burbuja)

```
/**
 * Ordena la estructura(pila, cola), utlizando un metodo de
 * ordenamiento directo(burbuja).
 */
void ordenarDatosDirecto(Nodo* &estructura)
{
    clock_t tiempoInicio;
    tiempoInicio = clock();

    int aux;
    Nodo* auxNodo = estructura;
    Nodo* copiaSiguiente = NULL;

    if(estructuraVacia(estructura))
    {
        printf("\nLa pila esta vacia");
    }
    else
    {
        while(auxNodo != NULL)
        {
            copiaSiguiente = auxNodo->siguiente;
            while(copiaSiguiente != NULL)
            {

                if(auxNodo->dato > copiaSiguiente->dato)
                {
                    aux = auxNodo->dato;
                    auxNodo->dato = copiaSiguiente->dato;
                    copiaSiguiente->dato = aux;
                }

                copiaSiguiente = copiaSiguiente->siguiente;
            }
            auxNodo = auxNodo->siguiente;
        }
    }
    printf("\nDatos ordenados\n");
    MENSAJE_TIEMPO;
}
```

Ordena la estructura procesando sus dígitos de forma individual, Según el digito se ingresa en una cola que es contenida en una posición del vector para este digito y luego se pasan a la estructura principal de forma ordenada, Despues continúa el mismo proceso con el siguiente digito del número.

```
 * Ordena la estructura(pila, cola), utlizando un metodo de
 * ordenamiento rápido(radix).
 */
void ordenarDatosRapido(Nodo* &estructura)
{
    clock_t tiempoInicio;
    tiempoInicio = clock();

    Nodo* auxNodo = estructura;
    Nodo* copiaSiguiente = NULL;
    Nodo* vector[19];
    Nodo* vectorFin[19];
    int divisor = 1;

    if(estructuraVacia(estructura))
    {
        printf("\nEstructura vacia");
    }
    else
    {
        iniciarArreglo(vector, 19);
        iniciarArreglo(vectorFin, 19);

        for(int i = 0; i < 6; i++)
        {
            copiaSiguiente = auxNodo;
            while(auxNodo != NULL)
            {
                /* vector[selectDigit(auxNodo->dato, divisor)+9] -> la función selectDigit me retorna un digito de un numero,
                y a este le sumo 9; con esto estoy diciendo que a esa posicion del vector(9+digito) se ve a gurdar el número.
                La suma del nueve se debe a q si selectDigit retorna un negativo que maximo puede ser -9 se guardara en la posicio 0 ya
                que -9 es el numero menor de todos. */
                insertarDatosCola(auxNodo->dato, vector[selectDigit(auxNodo->dato, divisor)+9], vectorFin[selectDigit(auxNodo->dato, divisor)+9])
```

```
                {
                    /* vector[selectDigit(auxNodo->dato, divisor)+9] -> la función selectDigit me retorna un digito de un numero,
                    y a este le sumo 9; con esto estoy diciendo que a esa posicion del vector(9+digito) se ve a gurdar el número.
                    La suma del nueve se debe a q si selectDigit retorna un negativo que maximo puede ser -9 se guardara en la posicio 0 ya
                    que -9 es el numero menor de todos. */
                    insertarDatosCola(auxNodo->dato, vector[selectDigit(auxNodo->dato, divisor)+9], vectorFin[selectDigit(auxNodo->dato, divisor)+9]);
                    auxNodo = auxNodo->siguiente;
                }

                auxNodo = copiaSiguiente;
                //eliminarPila(auxNodo);
                for(int j = 0; j < 19; j++)
                {
                    if(!estructuraVacia(vector[j]))
                    {
                        while(vector[j] != NULL)
                        {
                            //insertarElementoPila(vector[j]->dato, auxNodo);
                            auxNodo->dato = vector[j]->dato;
                            vector[j] = vector[j]->siguiente;
                            auxNodo = auxNodo->siguiente;
                        }
                        while(vector[j] != NULL)
                        {
                            eliminarElementoCola(vector[j], vectorFin[j]);
                        }
                        //eliminarCola(vector[j], vectorFin[j]);
                    }
                    vector[j] = NULL;
                }
                    auxNodo = copiaSiguiente;

                divisor *= 10;
            }

            printf("\nDatos ordenados\n");
        }

        MENSAJE_TIEMPO;
    }
```

Retorna el digito de un número.

```
/**
 * Retorna el dígito de un número
 */
int selectDigit(int number, int divisor)
{
    return (number/divisor%10);
}
```

Inicializa todos los elementos del arreglo en NULL

```
/**
 * Inicializa todos los elementos del arreglo en null
 */
void iniciarArreglo(Nodo *arreglo[], int cantidad)
{
    for(int i = 0; i < cantidad; i++)
    {
        arreglo[i] = NULL;
    }
}
```

Guarda los elementos de la estructura principal en un archivo. (La variable tipo define lo que se va aguardar sea una pila o una cola)

```
/**
 * Guarda los elementos de la estructura(pila, cola) en una archivo.
 */
int salvarDatos(Nodo* estructura, int tipo)
{
    clock_t tiempoInicio;
    tiempoInicio = clock();

    Nodo *aux = estructura;
    FILE * archivo;
    int dato = 0;
    char nombre[23];

    if(tipo == 1)
    {
        strcpy(nombre, "archivos/datosPila.txt");
    }
    else
    {
        strcpy(nombre, "archivos/datosCola.txt");
    }

    archivo = fopen(nombre, "wb");

    if(archivo == NULL)
    {
        printf("No se pudo crear el archivo");
        return false;
    }

    if(estructuraVacia(estructura))
    {
        printf("\nLa estructura esta vacia");
    }
    else
    {
        while(aux != NULL)
        {
            dato = aux->dato;
            fwrite(&dato, sizeof(dato), 1, archivo);
            aux = aux->siguiente;
        }
    }
    printf("Se guardaron correctamente los datos\n");
    MENSAJE_TIEMPO;
    return 0;
}
```