

A dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the date.

10/19/2021

Listas Simples

Practica Tema 5

Several thin, curved lines in dark blue and light grey that sweep upwards from the bottom left corner.

Alejandro Colmenero Moreno
PROGRAMACIÓN II

ÍNDICE

EJERCICIO 1.....	2
EJERCICIO 2.....	5
EJERCICIO 3.....	8
EJERCICIO 4.....	12
EJERCICIO 5.....	14
EJERCICIO 6.....	16
BIBLIOGRAFÍA	19

EJERCICIO 1

1. Insertar al principio - Realizar un programa en C que vaya pidiendo un entero y una cadena y vaya metiendo los datos en una lista simple por el principio. Tras insertar, que pregunte si meter otro elemento o acabar. Si se elige esta opción, se mostrará el contenido de la lista al completo y se liberará el espacio ocupado.

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

typedef struct estructura{
    int codigo;
    char nombre[10];
    struct estructura *siguiente;
} nodo;

nodo* nuevo();

nodo* nuevo(){
    nodo *p;
    p = (nodo*) malloc(sizeof(nodo));
    p->siguiente = NULL;
    return(p);
}

int main()
{
    char continuar[] = "Si";
    char nombre[10];
    int codigo = 0;
    int indice = 0;

    nodo* nodoAnterior = nuevo();
    nodo* nodoActual = nuevo();

    while( strcmp(&continuar,"Si") == 0 || strcmp(&continuar,"si") == 0 ){

        indice++;

        // PEDIR DATOS
        printf("Dime codigo: ");
        scanf("%i", &codigo);
        printf("Dime nombre: ");
        scanf("%s", &nombre);
        fseek(stdin,0,SEEK_END);

        // ASIGNAR AL AUXILIAR
        nodoActual = nuevo();
        nodoActual->codigo = codigo;
```

```

strcpy(nodoActual->nombre, nombre);

if(indice == 1){
    // INDICO EL PRIMER NODO
} else {
    // ASIGNO EL SIGUIENTE ES EL ANTERIOR
    nodoActual->siguiente = nodoAnterior;
}

nodoActual->siguiente = nodoAnterior;
// NODO ANTERIOR SERÁ EL CREADO AHORA
nodoAnterior = nodoActual;

// CONTINUAR
printf("Sigo?\n");
scanf("%s", &continuar);
printf("\n");
}

// IMPRIMIR
nodo* primerNodo = nodoActual;
while(nodoActual->siguiente!=NULL){
    printf("C: %d \t N: %s\n", nodoActual->codigo, nodoActual->nombre);
    nodoActual = nodoActual->siguiente;
}
primerNodo = NULL;
return 0;
}

```

```
Dime codigo: 1
Dime nombre: a
Sigo?
Si
```

```
Dime codigo: 2
Dime nombre: b
Sigo?
Si
```

```
Dime codigo: 3
Dime nombre: c
Sigo?
Si
```

```
Dime codigo: 4
Dime nombre: d
Sigo?
No
```

```
C: 4      N: d
C: 3      N: c
C: 2      N: b
C: 1      N: a
```

```
Process returned 0 (0x0)   execution time : 17.268 s
Press any key to continue.
```

EJERCICIO 2

2. Insertar al final - Realizar un programa en C que vaya pidiendo un entero y una cadena y vaya metiendo los datos en una lista simple por el final. Tras insertar, que pregunte si meter otro elemento o acabar. Si se elige esta opción, se mostrará el contenido de la lista al completo y se liberará el espacio ocupado.

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

typedef struct estructura{
    int codigo;
    char nombre[10];
    struct estructura *siguiente;
} nodo;

nodo* nuevo();

nodo* nuevo(){
    nodo *p;
    p = (nodo*) malloc(sizeof(nodo));
    p->siguiente = NULL;
    return(p);
}

int main()
{
    char continuar[] = "Si";
    char nombre[10];
    int codigo = 0;
    int indice = 0;

    nodo* nodoAnterior;
    nodo* nodoPrimero;
    nodo* nodoActual = nuevo();

    while( strcmp(&continuar,"Si") == 0 || strcmp(&continuar,"si") ==
0 || strcmp(&continuar,"s") == 0 ){

        indice++;

        // PEDIR DATOS
        printf("Dime codigo: ");
        scanf("%i", &codigo);
        printf("Dime nombre: ");
        scanf("%s", &nombre);
        fseek(stdin,0,SEEK_END);

        // ASIGNAR AL AUXILIAR
        nodoActual = nuevo();
        nodoActual->codigo = codigo;
```

```

strcpy(nodoActual->nombre, nombre);

// si es el primer nodo.
if(indice == 1){
    // INDICO EL PRIMER NODO
    nodoPrimero = nodoActual;
} else {
    // ASIGNO EL SIGUIENTE ES EL ANTERIOR
    nodoAnterior->siguiente = nodoActual;
}

// GUARDO EL NODO ANTERIOR
// ASI LE ASIGNO SU "SIGUIENTE" ÉL
nodoAnterior = nodoActual;

// CONTINUAR
printf("Sigo?.\n");
scanf("%s", &continuar);
printf("\n");
}

// creo un nuevo NODO apuntando el PRIMER NODO CREADO
nodo* auxiliar = nuevo();
auxiliar = nodoPrimero;

while(auxiliar!=NULL){
    printf("C: %d \t N: %s\n", auxiliar->codigo, auxiliar->nombre);
    auxiliar = auxiliar->siguiente;
}
nodoPrimero = NULL;

return 0;
}

```

```
Dime codigo: 1
Dime nombre: a
Sigo?.
Si
```

```
Dime codigo: 2
Dime nombre: b
Sigo?.
Si
```

```
Dime codigo: 3
Dime nombre: c
Sigo?.
Si
```

```
Dime codigo: 4
Dime nombre: d
Sigo?.
No
```

```
C: 1      N: a
C: 2      N: b
C: 3      N: c
C: 4      N: d
```

```
Process returned 0 (0x0)   execution time : 13.903 s
Press any key to continue.
```


EJERCICIO 3

En este código nos van pidiendo nombre y código de cada nodo. La primera vez que te lo pide, le da los datos a un NODO ya creado llamado PrimerNodo. La segunda vez hace lo mismo, pero a un nodo auxiliar, y le hago que el atributo “siguiente” de PrimerNodo apunte a este recién creado, al auxiliar.

A partir del tercero te empieza a preguntare dónde quieres introducir un nuevo nodo, y lo introduce después del nodo cuyo índice ha indicado.

En cada petición llama a una función que imprime por pantalla todos los nodos. Esto es posible gracias a que cree una variable NODO concreta para ser el primer nodo de la lista. Se le pasa por parámetros a la función este PrimerNodo y va imprimiendo los datos accediendo a la propiedad siguiente hasta que esta sea NULL.

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

typedef struct estructura{
    int codigo;
    char nombre[10];
    struct estructura *siguiente;
} nodo;

nodo* nuevo();

nodo* nuevo(){
    nodo *p;
    p = (nodo*) malloc(sizeof(nodo));
    p->siguiente = NULL;
    return(p);
}

mostrar_lista(nodo *nodoPrimero);
mostrar_lista(nodo *nodoPrimero){
    // creo un nuevo NODO apuntando el PRIMER NODO CREADO
    nodo* auxiliar = nuevo();
    auxiliar = nodoPrimero;

    while(auxiliar!=NULL){
        printf("C: %d \t N: %s\n", auxiliar->codigo, auxiliar->nombre);
        auxiliar = auxiliar->siguiente;
    }
}

int main()
{
    char continuar[] = "Si";
    char nombre[10];
    int codigo = 0;
    int indice = 0;
    int indiceAgrega = 0;
```

```

nodo* nodoPrimero = nuevo();

nodo* nodoIndiceAgregar = nuevo();

while( strcmp(&continuar,"Si") == 0 || strcmp(&continuar,"si") ==
0 || strcmp(&continuar,"s") == 0 ){

    // PEDIR DATOS
    printf("Dime codigo: ");
    scanf("%i", &codigo);
    fseek(stdin,0,SEEK_END);
    printf("Dime nombre: ");
    scanf("%s", &nombre);
    fseek(stdin,0,SEEK_END);

    if(indice == 0){ // CREAM PRIMERO

        nodoPrimero->codigo = codigo;
        strcpy(nodoPrimero->nombre, &nombre);
        nodoPrimero->siguiente = NULL;

        mostrar_lista(nodoPrimero);

    } else if(indice == 1) { // CREAM OTRO, PRIMERO->SIGUIENTE = ESTE OTRO

        nodo* nodoActual = nuevo();
        nodoActual->codigo = codigo;
        strcpy(nodoActual->nombre, nombre);
        nodoPrimero->siguiente = nodoActual;

        mostrar_lista(nodoPrimero);

    } else {

        nodo* nodoActual = nuevo();
        nodoActual->codigo = codigo;
        strcpy(nodoActual->nombre, nombre);

        printf("Dime indice: ");
        for(int x = 0; x < indice; x++){
            printf("[%i]",x);
        }
        printf("\n");
        scanf("%i", &indiceAgregar);
        if(indiceAgregar<0) {
            indiceAgregar = 0;
        }
        if(indiceAgregar>=indice) {
            indiceAgregar = indice;
            printf("a");
        }
    }
}

```

```

    printf("Insertado en indice %i.\n",indiceAgregar);

    // indico a cual AGREGAR
    nodoIndiceAgregar = nodoPrimero;
    for(int x = 0; x < indiceAgregar; x++){
        nodoIndiceAgregar = nodoIndiceAgregar->siguiente;
    }

    // HAGO EL SWITCH
    // el creado ahora, tendrá de siguiente el siguiente del
    nodoActual->siguiente = nodoIndiceAgregar->siguiente;
    nodoIndiceAgregar->siguiente = nodoActual;

    mostrar_lista(nodoPrimero);
}

// CONTINUAR
printf("Sigo?.\n");
scanf("%s", &continuar);
printf("\n");
indice++;
}

mostrar_lista(nodoPrimero);

return 0;
}

```

```
Dime codigo: 1
Dime nombre: a
C: 1      N: a
Sigo?.
Si

Dime codigo: 2
Dime nombre: b
C: 1      N: a
C: 2      N: b
Sigo?.
Si

Dime codigo: 3
Dime nombre: c
Dime indice: [0][1]
1
Insertado en indice 1.
C: 1      N: a
C: 2      N: b
C: 3      N: c
Sigo?.
Si

Dime codigo: 4
Dime nombre: d
Dime indice: [0][1][2]
0
Insertado en indice 0.
C: 1      N: a
C: 4      N: d
C: 2      N: b
C: 3      N: c
Sigo?.
Si

Dime codigo: 5
Dime nombre: e
Dime indice: [0][1][2][3]
2
Insertado en indice 2.
C: 1      N: a
C: 4      N: d
C: 2      N: b
C: 5      N: e
C: 3      N: c
Sigo?.
```

EJERCICIO 4

4. Eliminar primer elemento - Realizar un programa en C que genere de forma aleatoria una lista simple de tres elementos, muestre la lista cómo ha quedado, borre el primer elemento, libere el espacio ocupado por dicho elemento, vuelva a mostrar la lista resultante y por último libere el espacio ocupado por el resto de la lista.

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

typedef struct estructura{
    int codigo;
    char nombre[10];
    struct estructura *siguiente;
} nodo;

nodo* nuevo();

nodo* nuevo(){
    nodo *p;
    p = (nodo*) malloc(sizeof(nodo));
    p->siguiente = NULL;
    return(p);
}

mostrar_lista(nodo *nodoPrimero);
mostrar_lista(nodo *nodoPrimero){
    // creo un nuevo NODO apuntando el PRIMER NODO CREADO
    nodo* auxiliar = nuevo();
    auxiliar = nodoPrimero;

    while(auxiliar!=NULL){
        printf("C: %d \t N: %s\n", auxiliar->codigo, auxiliar->nombre);
        auxiliar = auxiliar->siguiente;
    }
}

int main()
{
    char continuar[] = "Si";
    char nombre[10];
    int codigo = 0;
    int indice = 0;
    int indiceAgrega = 0;

    nodo* nodo1 = nuevo();
    nodo* nodo2 = nuevo();
    nodo* nodo3 = nuevo();

    // RELLENAR
```

```

nodo1->codigo = 1;
strcpy(nodo1->nombre, "a");
nodo1->siguiente = nodo2;

nodo2->codigo = 2;
strcpy(nodo2->nombre, "b");
nodo2->siguiente = nodo3;


nodo3->codigo = 3;
strcpy(nodo3->nombre, "c");

mostrar_lista(nodo1);
printf("\n");

// QUITAR PRIMER
nodo1 = NULL;
mostrar_lista(nodo2);

return 0;
}

```

 C:\Users\pycol\OneDrive\Escritorio\Formaci³n\Experto Universitario Programaci³n

```

C: 1      N: a
C: 2      N: b
C: 3      N: c

```

```

C: 2      N: b
C: 3      N: c

```

```

Process returned 0 (0x0)   execution time : 0.026 s
Press any key to continue.

```

EJERCICIO 5

5. Eliminar último elemento - Realizar un programa en C que genere de forma aleatoria una lista simple de tres elementos, muestre la lista cómo ha quedado, borre el último elemento, libere el espacio ocupado por dicho elemento, vuelva a mostrar la lista resultante y por último libere el espacio ocupado por el resto de la lista.

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

typedef struct estructura{
    int codigo;
    char nombre[10];
    struct estructura *siguiente;
} nodo;

nodo* nuevo();

nodo* nuevo(){
    nodo *p;
    p = (nodo*) malloc(sizeof(nodo));
    p->siguiente = NULL;
    return(p);
}

mostrar_lista(nodo *nodoPrimero);
mostrar_lista(nodo *nodoPrimero){
    // creo un nuevo NODO apuntando el PRIMER NODO CREADO
    nodo* auxiliar = nuevo();
    auxiliar = nodoPrimero;

    while(auxiliar!=NULL){
        printf("C: %d \t N: %s\n", auxiliar->codigo, auxiliar->nombre);
        auxiliar = auxiliar->siguiente;
    }
}

int main()
{
    char continuar[] = "Si";
    char nombre[10];
    int codigo = 0;
    int indice = 0;
    int indiceAgrega = 0;

    nodo* nodo1 = nuevo();
    nodo* nodo2 = nuevo();
    nodo* nodo3 = nuevo();
```

```

// RELLENAR
nodo1->codigo = 1;
strcpy(nodo1->nombre, "a");
nodo1->siguiente = nodo2;

nodo2->codigo = 2;
strcpy(nodo2->nombre, "b");
nodo2->siguiente = nodo3;

nodo3->codigo = 3;
strcpy(nodo3->nombre, "c");

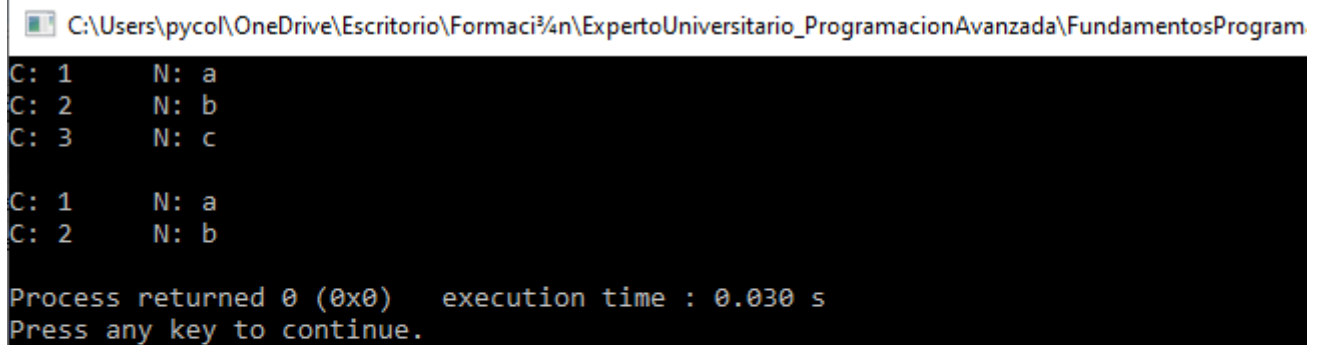
mostrar_lista(nodo1);
printf("\n");

nodo2->siguiente = NULL;
nodo3 = NULL;

mostrar_lista(nodo1);

return 0;
}

```



```

C:\Users\pycol\OneDrive\Escritorio\Formaci3/4n\Experto Universitario Programacion Avanzada\Fundamentos Program
C: 1      N: a
C: 2      N: b
C: 3      N: c

C: 1      N: a
C: 2      N: b

Process returned 0 (0x0)   execution time : 0.030 s
Press any key to continue.

```


EJERCICIO 6

6. Eliminar un elemento - Realizar un programa en C que genere de forma aleatoria una lista simple de cinco elementos, muestre la lista cómo ha quedado, pregunte por alguno de los elementos mostrados para borrarlo, borre dicho elemento, libere el espacio ocupado por ese elemento, vuelva a mostrar la lista resultante y por último libere el espacio ocupado por el resto de la lista. Si el elemento a eliminar no existiese, se mostrará error y se volverá a preguntar por otro elemento a borrar.

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

typedef struct estructura{
    int codigo;
    char nombre[10];
    struct estructura *siguiente;
} nodo;

nodo* nuevo();

nodo* nuevo(){
    nodo *p;
    p = (nodo*) malloc(sizeof(nodo));
    p->siguiente = NULL;
    return(p);
}

mostrar_lista(nodo *nodoPrimero);
mostrar_lista(nodo *nodoPrimero){
    // creo un nuevo NODO apuntando el PRIMER NODO CREADO
    nodo* auxiliar = nuevo();
    auxiliar = nodoPrimero;

    int x = 0;
    while(auxiliar!=NULL){
        printf("%d)Codigo: %d \t Nombre: %s\n", ++x, auxiliar->codigo, auxiliar->nombre);
        auxiliar = auxiliar->siguiente;
    }
}

int main()
{
    char *texto;
    int nbytes;
    texto = "abcdefghij";

    nodo* nodoPrimero = nuevo();
    int indiceBorrar = -1;
    int num = rand() % 10 + 1;
```

```

char letra = texto[0];

for(int x = 0; x < 5; x++){

    num = rand() % 10 + 1;

    if(x == 0){
        // primero
        nodo* nodoActual = nuevo();
        //nodoActual->nombre = texto[0];
        nodoActual->nombre[0] = texto[num-1];
        nodoActual->codigo = num;
        nodoActual->siguiente = NULL;

        nodoPrimero = nodoActual;
    } else {
        // Segundo
        nodo* nodoActual = nuevo();
        nodoActual->nombre[0] = texto[num-1];
        nodoActual->codigo = num;

        nodo* auxiliar = nodoPrimero;
        while( auxiliar->siguiente != NULL){
            auxiliar = auxiliar->siguiente;
        }
        auxiliar->siguiente = nodoActual;

    }
}

mostrar_lista(nodoPrimero);
printf("\n");


int repetir = 0;
while(repetir == 0){
    printf("Dime INDICE a borrar: ");
    scanf("%i", &indiceBorrar);
    fseek(stdin,0,SEEK_END);
    if(indiceBorrar>=0 && indiceBorrar <= 4) {
        repetir = 1;
    }
}

nodo* auxiliar = nodoPrimero;
for(int x = 0; x < indiceBorrar-1; x++){
    auxiliar = nodoPrimero->siguiente;
}
auxiliar->siguiente = auxiliar->siguiente->siguiente;
printf("Borrado en indice %i.\n", indiceBorrar);

```

```
printf("\n");

mostrar_lista(nodoPrimero);


return 0;
}
```

C:\Users\pycol\OneDrive\Escritorio\Formaci3/4n\ExpertoUniversitario_ProgramacionAvanzada\FundamentosProgram:

```
1)Codigo: 8      Nombre: h
2)Codigo: 5      Nombre: e
3)Codigo: 1      Nombre: a
4)Codigo: 10     Nombre: j
5)Codigo: 5      Nombre: e

Dime INDICE a borrar: -234
Dime INDICE a borrar: fewe
Dime INDICE a borrar: 3845
Dime INDICE a borrar: 2
Borrado en indice 2.

1)Codigo: 8      Nombre: h
2)Codigo: 5      Nombre: e
3)Codigo: 10     Nombre: j
4)Codigo: 5      Nombre: e

Process returned 0 (0x0)   execution time : 13.337 s
Press any key to continue.
```

BIBLIOGRAFÍA

N/A