Universidad Autónoma de San Luis Potosí

Facultad de Ingeniería

Área en Ciencias de la Computación

Estructuras de Datos II

Semestre: Agosto – Diciembre 2022

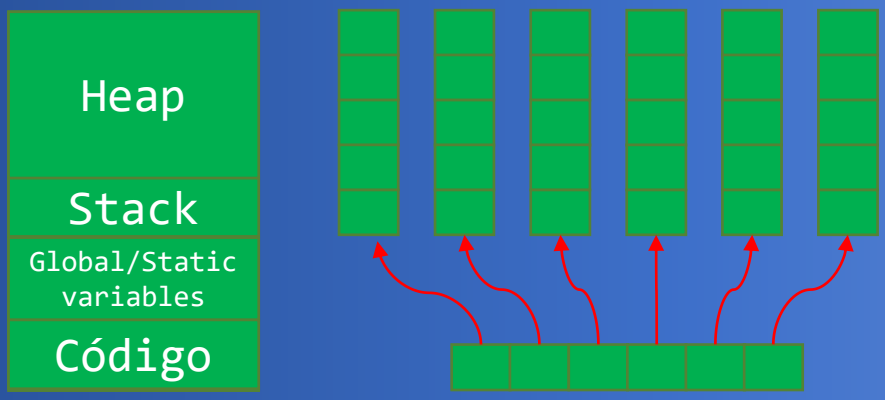
Alumno: Esparza Castañeda Hugo

Profesor: M. en C. Froylán Eloy Hernández Castro

Índice General de Temas

Martes 16 Agosto 2022

1. APUNTADORES

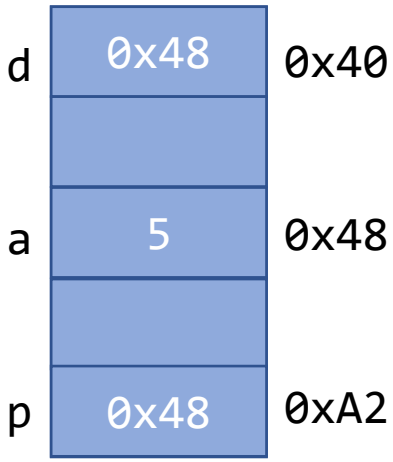


1.1 Introducción a apuntadores

Objetivo: Conocer el concepto de apuntadores en el paso de parámetros de funciones y gestión de memoria (Dinámica), así como ser capaz de utilizarlos.

Apuntador: un apuntador es una variable que contiene una dirección de memoria pero su contenido es otra dirección de memoria.

```c
int main(){
    int a;
    a = 5;
    int *p;
    p = &a;
    int *d;
    d = p;
    return 0;
}
```




```c
printf("%d", a;); //output: 5
printf("%d", *p;); //output: 5
printf("%p", p;); //output: 0x48
printf("%d", *d;); //output: 5
printf("%p", d;); //output: 0x48
```

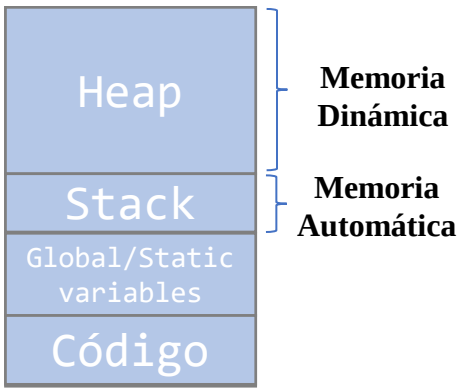
a = 10; ⇔ *p = 10

scanf(“%d”, p) ⇔ scanf(“%d”, &a)

1.2 Parámetros por valor/referencia

Jueves 18 Agosto 2022

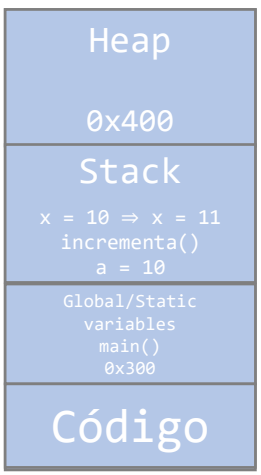
Memoria



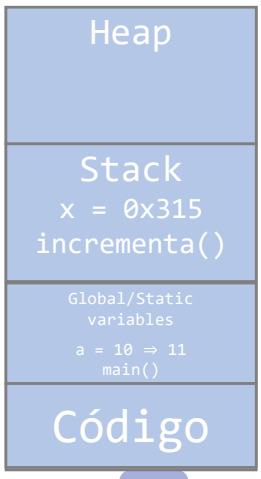

```c
#include <stdio.h>
void incrementa(int *);
int main(){
    int a;
    a = 10;
    incrementa(&a);
    printf("%d", a);
    return 0;
}

void incrementa(int *x){
    *x = *x + 1;
    return;
}
```

Por valor



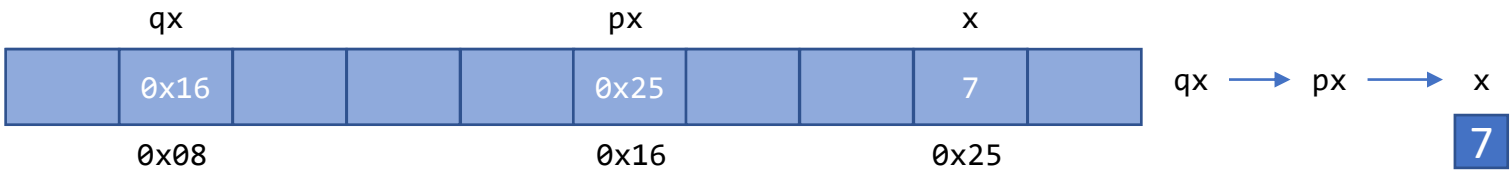
Por referencia



```
int *r;   ⇒   int *r = NULL;
```

1.3 Apuntadores a apuntadores

Viernes 19 Agosto 2022



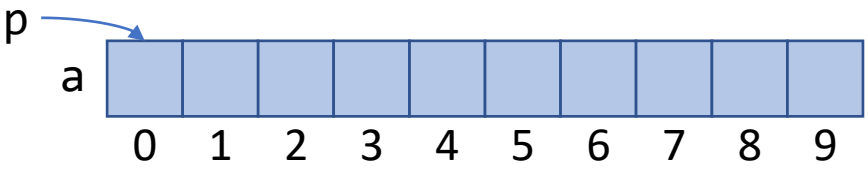

```c
#include <stdio.h>
int main(){
    int x = 7; //entero
    int *px = &x; //apuntador
    int **qx = &px; //apuntador a apunt
    printf("%p\n", px);  //output: 0x25
    printf("%d\n", *px); //output: 7
    printf("%p\n", &x);  //output: 0x25
    printf("%p\n", &px); //output: 0x16
    printf("%p\n", &qx); //output: 0x08
    printf("%p\n", qx);  //output: 0x16
    printf("%p\n", *qx); //output: 0x25
    return 0;
}
```

* = indirección = Obtener el valor almacenado en una dirección.
& = dirección = Obtener la dirección de una variable.

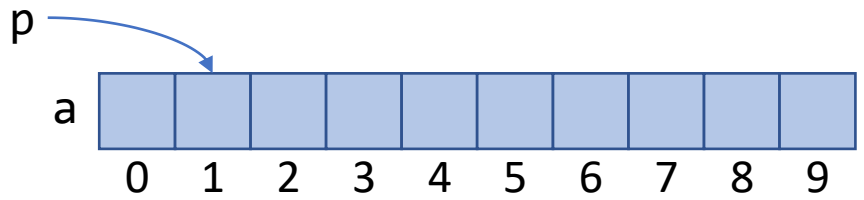
Los apuntadores a apuntadores se usan para matrices dinámicas.

1.4 Apuntadores y arreglos

```
int a[10];
int *p = a; ⇒ int *p = &a[0];
```

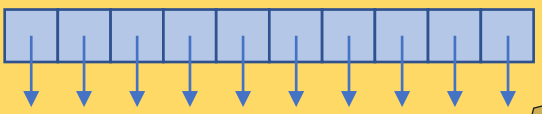



```
p = p + 1;
```



Arreglo de apuntadores

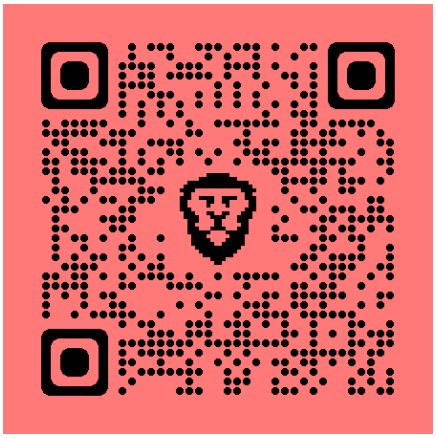
```
int *b[10];
```



Arreglo de apuntadores apuntando a NULL

```
for(int i = 0; i < 10; i++){   ⇔   int *b[10] = {NULL};
    b[i] = NULL;
}
```

Tarea #0: Resumen del video “Pointers”



Los punteros es uno de los temas más difíciles, sim embargo son de mucha utilidad, como por ejemplo al pasar datos a una función, modificarlos y luego regresarlos, cosa que no podría hacerse de otra manera.
Cuando pasamos un dato por valor a una función, lo que estamos pasando es en realidad una copia, sin embargo, cuando pasamos el dato por referencia haciendo el uso de punteros, lo que hacemos es darle acceso a la memoria en donde esta almacenado dicho valor, haciendo que todos los cambios que realicemos en la función, le sucedan al valor mismo.
Cada archivo de nuestra computadora esta almacenado en el disco duro, o en un disco de estado solido, según sea el caso. Esos discos son solo de almacenamiento, por lo tanto, no podemos trabajar directamente ahí, la manipulación de los datos únicamente puede hacerse en la memoria RAM, por lo que tenemos que movernos ahí. La memoria RAM es como un arreglo enorme. La memoria RAM es Memoria de Acceso Aleatorio, y una vez que apagamos la computadora, todos los datos de la memoria RAM se destruyen.

Retomando lo que dijimos hace un momento que la memoria RAM es un arreglo enorme dividido en celdas de bytes (dependiendo del tamaño del tipo de dato), así como los arreglos tienen índices para cada valor guardado, la memoria tiene una dirección para cada dato almacenado.
Las variables de tipo str o cadena de texto, necesitan llevar el `'\0'` para poder saber donde terminan. Una vez aclarado el tema de la memoria, lo más importante de recordar sobre los punteros es, que son solo direcciones de memoria.

Tipo de dato	Tamaño (en bytes)
int	4
char	1
float	4
double	8
long long	8
char*	4 u 8

Entonces, un puntero es un tipo que dato que guarda una dirección, y el tipo únicamente nos describe el dato guardado en esa dirección de memoria.
Al saber la dirección de la memoria en donde esta guardada una variable, podemos ir directamente a esa ubicación en la memoria y manipular el dato, es por eso que no es necesario hacer una copia cuando enviamos un valor por referencia en una función.
El puntero más simple en C es el puntero `NULL`, que como su nombre o indica, no apunta a nada (lo cual en realidad puede ser muy útil).Siempre que declaramos un puntero, y no le asignamos una dirección, debemos hacer `NULL` el valor de ese puntero (son buenas practicas de programación).Podemos extraer la dirección de una variable con el operador `&` (como en los ejemplos anteriores).
El propósito principal de un puntero es permitirnos modificar o inspeccionar la ubicación a la que apunta.
Si tenemos un puntero pc, entonces `*pc` es el dato que vive en la dirección de memoria guardada en pc.
Usado en contexto, * es el operador que nos permite acceder al dato almacenado en la dirección, dicho de otro modo, no nos sirve únicamente saber la dirección, también debemos ir allí, y el operador * nos permite hacerlo.
Por lo tanto, cuando tenemos un puntero que apunta a `NULL`, es decir a nada, e intentamos ir a esa dirección con el operador * nos aparecerá el error segmentation fault.
Es por esto la importancia de declarar todos los apuntadores a `NULL` cuando no se les asigna inmediatamente una dirección significativa, ya que si no lo hacemos, nuestro apuntador podría apuntar a cualquier espacio de memoria y manipularlo accidentalmente.

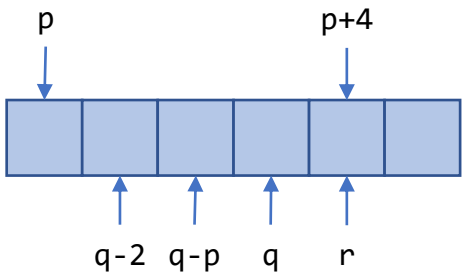
1.5 Aritmética de apuntadores

Lunes 22 Agosto 2022

Operador	Nombre
*	Indirección
&	Dirección
->	Apunta a
+	Suma
-	Resta
==	Igualdad
!=	Desigualdad
>	Comparación de direcciones
>=	
<	
<=	
(tipo de dato)	Cast ó conversión explicita

Operaciones aritméticas sobre apuntadores

1. Sumar un entero a un apuntador.
2. Restar un entero a un apuntador.
3. Restar dos apuntadores.
4. Comparaciones.




```
p < q  True
p == q False
q >= p True
```

```
q == r + 1 False
q + 1 == r True
```

Ejemplo 1:

```c
#include <stdio.h>
int main(){
    int a[] = {28, 41, 7};
    int *pi = a;
    printf("%d\n", *pi); //output: 28
    pi += 1;
    printf("%d\n", *pi); //output: 41
    pi++;
    printf("%d\n", *pi); //output: 7
    ++pi;
    printf("%d\n", *pi); //output: ???
    return 0;
}
```

Ejemplo 2:

```c
#include <stdio.h>
int main(){
    short s;
    short *ps = &s;
    char c;
    char *pc = &c;
    printf("%p\n", ps); //output: 0x7ffcca0af8c6
    ps = ps + 1;
    printf("%p\n", ps); //output: 0x7ffcca0af8c8
    printf("%p\n", pc); //output: 0x7ffcca0af8c5
    pc = pc + 1;
    printf("%p\n", pc); //output: 0x7ffcca0af8c6
    return 0;
}
```

short = 2 bytes, char = 1 byte

Martes 23 Agosto 2022

Ejemplo 3: Función strlen

```c
int strlen(char *s){
    char *p = s;
    while(*p != '\0'){
        p++;
    }
    return p-s;
}
```

Ejemplo 4: Función strcpy

```c
void strcpy(char *s, char *t){
    while((*s = *t) != '\0'){
        s++;
        t++;
    }
    return;
}
```

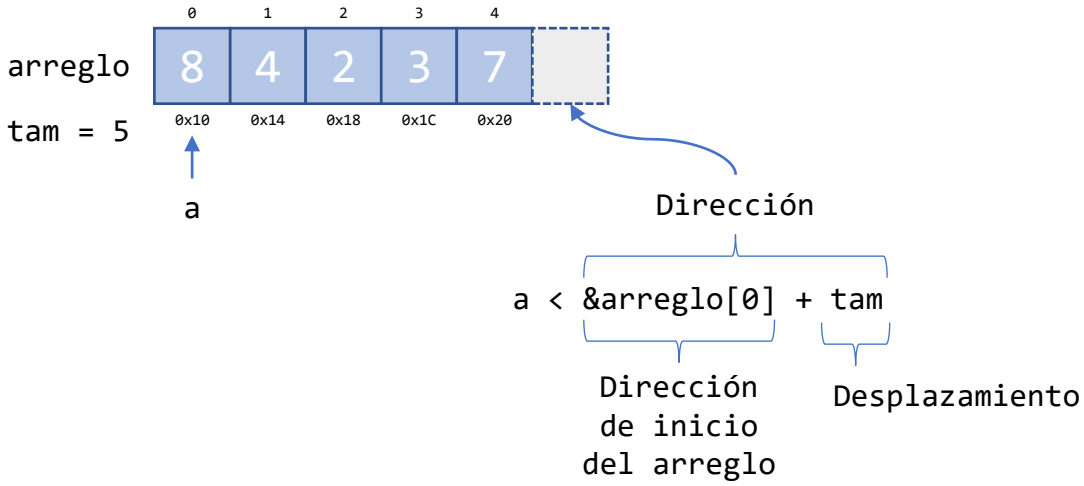
Ejemplo 5: Función strcmp

```c
int strcmp(char *s, char *t){
    for(;*s == *t; s++, t++){
        if(*s == '\0'){
            return 0;
        }
    }
    return *s - *t;
}
```

Problema

Obtener la suma de los elementos de un arreglo de enteros de un tamaño determinado, usando aritmética de apuntadores.

```
int arreglo[5] = {8 4, 2, 3, 7};
```



Con ciclo for

```c
#include <stdio.h>
int suma(int, int []);
int main(){
    int tam = 5;
    int arreglo[5] = {8, 4, 2, 3, 7};
    int p = suma(tam, arreglo);
    printf("%d\n", p);
    return 0;
}

int suma(int tam, int arreglo[tam]){
    int sum = 0;
    int *a = arreglo;
    for(int *a = arreglo; a - arreglo < tam; a++){
        sum += *a;
    }
    return sum;
}
```

Con ciclo while (más elegante)

```c
#include <stdio.h>
int suma(int, int []);
int main(){
    int tam = 5;
    int arreglo[5] =
    {8, 4, 2, 3, 7};
    int p = suma(tam, arreglo);
    printf("%d\n", p);
    return 0;
}

int suma(int tam, int arreglo[tam]){
    int sum = 0;
    int *a = arreglo;
    int *b = &arreglo[tam];
    while(a < b){
        sum += *a++;
    }
    return sum;
}
```

```
arreglo ⇒ &arreglo[0]

&arreglo[tam] ⇒ too-far pointer
                no forma parte
                del arreglo
```

Tarea #1: Aritmética de apuntadores

Instructions

Given any two lists A and B, determine if:

- List A is equal to list B; or
- List A contains list B (A is a superlist of B); or
- List A is contained by list B (A is a sublist of B); or
- None of the above is true, thus lists A and B are unequal

Specifically, list A is equal to list B if both lists have the same values in the same order.
List A is a superlist of B if A contains a sub-sequence of values equal to B.
List A is a sublist of B if B contains a sub-sequence of values equal to A.

Examples:

- If A = [] and B = [] (both lists are empty), then A and B are equal
- If A = [1, 2, 3] and B = [], then A is a superlist of B
- If A = []` and B = [1, 2, 3], then A is a sublist of B
- If A = [1, 2, 3] and B = [1, 2, 3, 4, 5], then A is a sublist of B
- If A = [3, 4, 5] and B = [1, 2, 3, 4, 5], then A is a sublist of B
- If A = [3, 4] and B = [1, 2, 3, 4, 5], then A is a sublist of B
- If A = [1, 2, 3] and B = [1, 2, 3], then A and B are equal
- If A = [1, 2, 3, 4, 5] and B = [2, 3, 4], then A is a superlist of B
- If A = [1, 2, 4] and B = [1, 2, 3, 4, 5], then A and B are unequal
- If A = [1, 2, 3] and B = [1, 3, 2], then A and B are unequal

sublist.h

```c
#ifndef SUBLIST_H
#define SUBLIST_H

#include <stddef.h>

typedef enum { EQUAL, UNEQUAL, SUBLIST, SUPERLIST } comparison_result_t;

comparison_result_t check_lists(int *list_to_compare_begin, int *list_to_compare_end,
                                int *base_list_begin, int *base_list_end);

#endif
```

test_sublist.c

```c
#include "test-framework/unity.h"
#include "sublist.h"
#define ELEMENT_COUNT(array) (sizeof(array) / sizeof(array[0]))
void setUp(void){
}

void tearDown(void){
}

static void test_empty_lists(void){
   TEST_IGNORE();   // delete this line to run test
   TEST_ASSERT_EQUAL(EQUAL, check_lists(NULL, NULL, NULL, NULL));
}

static void test_empty_list_within_non_empty_list(void){
   TEST_IGNORE();
   int base_list[] = { 1, 2, 3 };

   TEST_ASSERT_EQUAL(SUBLIST,
                     check_lists(NULL, NULL, base_list, base_list+ELEMENT_COUNT(base_list)));
}

static void test_non_empty_list_contains_empty_list(void){
   TEST_IGNORE();
   int list_to_compare[] = { 1, 2, 3 };

   TEST_ASSERT_EQUAL(SUPERLIST, check_lists(list_to_compare,
                                            list_to_compare+ELEMENT_COUNT(list_to_compare),
                                            NULL, NULL));
}

static void test_list_equals_itself(void){
   TEST_IGNORE();
   int list_to_compare[] = { 1, 2, 3 };
   int base_list[] = { 1, 2, 3 };

   TEST_ASSERT_EQUAL(EQUAL, check_lists(list_to_compare,
                                        list_to_compare+ELEMENT_COUNT(list_to_compare),
                                        base_list,
                                        base_list+ELEMENT_COUNT(base_list)));
}

static void test_different_lists(void){
   TEST_IGNORE();
   int list_to_compare[] = { 1, 2, 3 };
   int base_list[] = { 2, 3, 4 };

   TEST_ASSERT_EQUAL(UNEQUAL, check_lists(list_to_compare,
                                        list_to_compare+ELEMENT_COUNT(list_to_compare),
                                        base_list,
                                        base_list+ELEMENT_COUNT(base_list)));
}
```

```
static void test_false_start(void){
   TEST_IGNORE();
   int list_to_compare[] = { 1, 2, 5 };
   int base_list[] = { 0, 1, 2, 3, 1, 2, 5, 6 };

   TEST_ASSERT_EQUAL(SUBLIST, check_lists(list_to_compare,
                                          list_to_compare+ELEMENT_COUNT(list_to_compare),
                                          base_list,
                                          base_list+ELEMENT_COUNT(base_list)));
}

static void test_consecutive(void){
   TEST_IGNORE();
   int list_to_compare[] = { 1, 1, 2 };
   int base_list[] = { 0, 1, 1, 1, 2, 1, 2 };

   TEST_ASSERT_EQUAL(SUBLIST, check_lists(list_to_compare,
                                          list_to_compare+ELEMENT_COUNT(list_to_compare),
                                          base_list,
                                          base_list+ELEMENT_COUNT(base_list)));
}

static void test_sublist_at_start(void){
   TEST_IGNORE();
   int list_to_compare[] = { 0, 1, 2 };
   int base_list[] = { 0, 1, 2, 3, 4, 5 };

   TEST_ASSERT_EQUAL(SUBLIST, check_lists(list_to_compare,
                                          list_to_compare+ELEMENT_COUNT(list_to_compare),
                                          base_list,
                                          base_list+ELEMENT_COUNT(base_list)));
}

static void test_sublist_at_middle(void){
   TEST_IGNORE();
   int list_to_compare[] = { 2, 3, 4 };
   int base_list[] = { 0, 1, 2, 3, 4, 5 };

   TEST_ASSERT_EQUAL(SUBLIST, check_lists(list_to_compare,
                                          list_to_compare+ELEMENT_COUNT(list_to_compare),
                                          base_list,
                                          base_list+ELEMENT_COUNT(base_list)));
}

static void test_sublist_at_end(void){
   TEST_IGNORE();
   int list_to_compare[] = { 3, 4, 5 };
   int base_list[] = { 0, 1, 2, 3, 4, 5 };

   TEST_ASSERT_EQUAL(SUBLIST, check_lists(list_to_compare,
                                          list_to_compare+ELEMENT_COUNT(list_to_compare),
                                          base_list,
                                          base_list+ELEMENT_COUNT(base_list)));
}
```

```
static void test_at_start_of_superlist(void){
   TEST_IGNORE();
   int list_to_compare[] = { 0, 1, 2, 3, 4, 5 };
   int base_list[] = { 0, 1, 2 };

   TEST_ASSERT_EQUAL(SUPERLIST, check_lists(list_to_compare,
                                            list_to_compare+ELEMENT_COUNT(list_to_compare),
                                            base_list,
                                            base_list+ELEMENT_COUNT(base_list)));
}

static void test_in_middle_of_superlist(void){
   TEST_IGNORE();
   int list_to_compare[] = { 0, 1, 2, 3, 4, 5 };
   int base_list[] = { 2, 3 };

   TEST_ASSERT_EQUAL(SUPERLIST, check_lists(list_to_compare,
                                            list_to_compare+ELEMENT_COUNT(list_to_compare),
                                            base_list,
                                            base_list+ELEMENT_COUNT(base_list)));
}

static void test_at_end_of_superlist(void){
   TEST_IGNORE();
   int list_to_compare[] = { 0, 1, 2, 3, 4, 5 };
   int base_list[] = { 3, 4, 5 };

   TEST_ASSERT_EQUAL(SUPERLIST, check_lists(list_to_compare,
                                            list_to_compare+ELEMENT_COUNT(list_to_compare),
                                            base_list,
                                            base_list+ELEMENT_COUNT(base_list)));
}

static void test_first_list_missing_element_from_second_list(void){
   TEST_IGNORE();
   int list_to_compare[] = { 1, 3 };
   int base_list[] = { 1, 2, 3 };

   TEST_ASSERT_EQUAL(UNEQUAL, check_lists(list_to_compare,
                                            list_to_compare+ELEMENT_COUNT(list_to_compare),
                                            base_list,
                                            base_list+ELEMENT_COUNT(base_list)));
}

static void test_second_list_missing_element_from_first_list(void){
   TEST_IGNORE();
   int list_to_compare[] = { 1, 2, 3 };
   int base_list[] = { 1, 3 };

   TEST_ASSERT_EQUAL(UNEQUAL, check_lists(list_to_compare,
                                            list_to_compare+ELEMENT_COUNT(list_to_compare),
                                            base_list,
                                            base_list+ELEMENT_COUNT(base_list)));
}
```

```c
static void test_first_list_missing_additional_digits_from_second_list(void){
   TEST_IGNORE();
   int list_to_compare[] = { 1, 2 };
   int base_list[] = { 1, 22 };

   TEST_ASSERT_EQUAL(UNEQUAL, check_lists(list_to_compare,
                                          list_to_compare+ELEMENT_COUNT(list_to_compare),
                                          base_list,
                                          base_list+ELEMENT_COUNT(base_list)));
}

static void test_order_matters_to_a_list(void){
   TEST_IGNORE();
   int list_to_compare[] = { 1, 2, 3 };
   int base_list[] = { 3, 2, 1 };

   TEST_ASSERT_EQUAL(UNEQUAL, check_lists(list_to_compare,
                                          list_to_compare+ELEMENT_COUNT(list_to_compare),
                                          base_list,
                                          base_list+ELEMENT_COUNT(base_list)));
}

static void test_same_digits_but_different_numbers(void){
   TEST_IGNORE();
   int list_to_compare[] = { 1, 0, 1 };
   int base_list[] = { 10, 1 };

   TEST_ASSERT_EQUAL(UNEQUAL, check_lists(list_to_compare,
                                          list_to_compare+ELEMENT_COUNT(list_to_compare),
                                          base_list,
                                          base_list+ELEMENT_COUNT(base_list)));
}

static void test_different_signs(void){
   TEST_IGNORE();
   int list_to_compare[] = { 1, 2, 3 };
   int base_list[] = { 1, -2, 3 };

   TEST_ASSERT_EQUAL(UNEQUAL, check_lists(list_to_compare,
                                          list_to_compare+ELEMENT_COUNT(list_to_compare),
                                          base_list,
                                          base_list+ELEMENT_COUNT(base_list)));
}

int main(void){
   UnityBegin("test_sublist.c");
   RUN_TEST(test_empty_lists);
   RUN_TEST(test_empty_list_within_non_empty_list);
   RUN_TEST(test_non_empty_list_contains_empty_list);
   RUN_TEST(test_list_equals_itself);
   RUN_TEST(test_different_lists);
   RUN_TEST(test_false_start);
   RUN_TEST(test_consecutive);
   RUN_TEST(test_sublist_at_start);
   RUN_TEST(test_sublist_at_middle);
   RUN_TEST(test_sublist_at_end);
   RUN_TEST(test_at_start_of_superlist);
   RUN_TEST(test_in_middle_of_superlist);
   RUN_TEST(test_at_end_of_superlist);
   RUN_TEST(test_first_list_missing_element_from_second_list);
   RUN_TEST(test_second_list_missing_element_from_first_list);
   RUN_TEST(test_first_list_missing_additional_digits_from_second_list);
   RUN_TEST(test_order_matters_to_a_list);
   RUN_TEST(test_same_digits_but_different_numbers);
   RUN_TEST(test_different_signs);
   return UnityEnd();
}
```

sublist.c

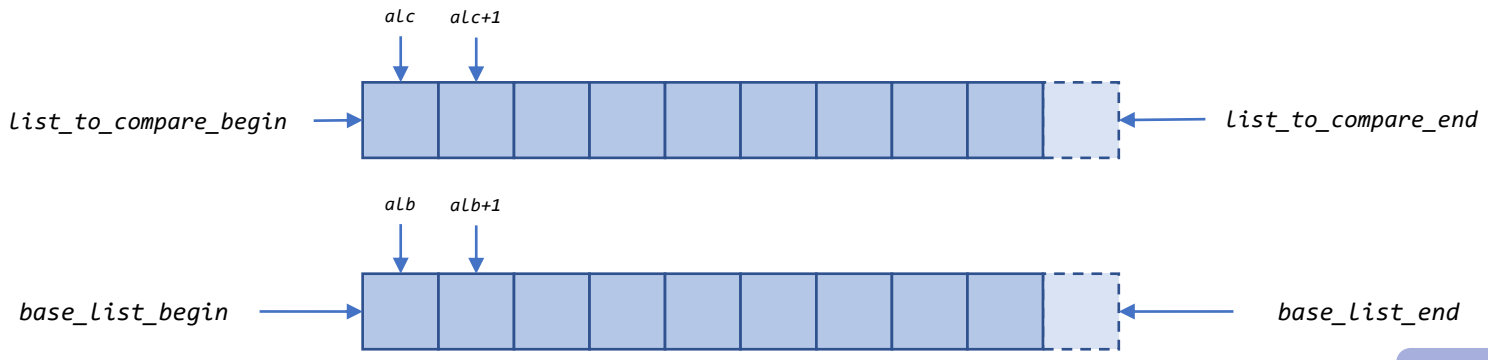
```c
#include "sublist.h"
comparison_result_t check_lists(int *list_to_compare_begin, int *list_to_compare_end,
                                int *base_list_begin, int *base_list_end){
    //Test 01
    //Primero verificamos que no esten vacios los dos
    if(list_to_compare_begin == NULL && base_list_begin == NULL){
        return EQUAL;
    }
    //Test 02
    //Revisamos si la lista a comparar esta vacia, pero otra no
    else if(base_list_begin != NULL && list_to_compare_begin == NULL){
        return SUBLIST;
    }
    //Test 03
    //Revisamos si la lista base esta vacia, pero la segunda no
    else if(list_to_compare_begin != NULL && base_list_begin == NULL){
        return SUPERLIST;
    }
    //Cuando las dos listas no estan vacias
    else{
        //Cuando son del mismo tamaño
        if(list_to_compare_end - list_to_compare_begin == base_list_end - base_list_begin){
            int *alc = list_to_compare_begin;
            int *alb = base_list_begin;
            //Compara uno por uno
            while(alc < list_to_compare_end){
                if(*alc == *alb){
                    alc++;
                    alb++;
                }
                //Si hay por lo menos uno diferente
                //Test 05, 16, 17, 19
                else{
                    return UNEQUAL;
                }
            }
            //Si el ciclo termina sin diferencias
            //Test 04
            return EQUAL;
        }
        //Cuando la lista base es mas grande
        else if(base_list_end - base_list_begin > list_to_compare_end - list_to_compare_begin){
            int *alc = list_to_compare_begin;
            int *alb = base_list_begin;
            int t = 0;
            int *tmp = NULL;
            while(alb < base_list_end){
                if(*alc == *alb){
                    alc++;
                    t++;
                    if(t == 1){
                        tmp = alb;
                    }
                    else if(t == list_to_compare_end - list_to_compare_begin){
```

```
55                          return SUBLIST;
56                      }
57                      alb++;
58                  }
59                  else{
60                      alc = list_to_compare_begin;
61                      if(t == 0){
62                          alb++;
63                      }
64                      else{
65                          t = 0;
66                          alb = tmp + 1;
67                      }
68                  }
69              }
70              return UNEQUAL;
71          }
72          //Cuando la lista a comparar es más grande
73          else{
74              int *alc = list_to_compare_begin;
75              int *alb = base_list_begin;
76              int t = 0;
77              int *tmp = NULL;
78              while(alc < list_to_compare_end){
79                  if(*alc == *alb){
80                      alb++;
81                      t++;
82                      if(t == 1){
83                          tmp = alc;
84                      }
85                      else if(t == base_list_end - base_list_begin){
86                          return SUPERLIST;
87                      }
88                      alc++;
89                  }
90                  else{
91                      alb = base_list_begin;
92                      if(t == 0){
93                          alc++;
94                      }
95                      else{
96                          t = 0;
97                          alc = tmp + 1;
98                      }
99                  }
100             }
101             return UNEQUAL;
102         }
103     }
104 }
```

1. Explique su algoritmo para determinar que dos arreglos son iguales.

```
5       //Primero verificamos que no esten vacios los dos
6       if(list_to_compare_begin == NULL && base_list_begin == NULL){
7           return EQUAL;
8       }
19      //Cuando las dos listas no estan vacias
20      else{
21          //Cuando son del mismo tamaño
22          if(list_to_compare_end - list_to_compare_begin == base_list_end - base_list_begin){
23              int *alc = list_to_compare_begin;
24              int *alb = base_list_begin;
25              //Compara uno por uno
26              while(alc < list_to_compare_end){
27                  if(*alc == *alb){
28                      alc++;
29                      alb++;
30                  }
31                  //Si hay por lo menos uno diferente
32                  //Test 05, 16, 17, 19
33                  else{
34                      return UNEQUAL;
35                  }
36              }
37              //Si el ciclo termina sin diferencias
38              //Test 04
39              return EQUAL;
40          }
```

Primero pongo un caso base para cuando los dos arreglos están vacíos (líneas de código de la 6 a la 8), si los apuntadores del inicio apuntan a `NULL`, quiere decir que no hay elementos en ninguno y son iguales.
Después, para cuando los arreglos no están vacíos y son del mismo tamaño (líneas de código de la 22 a la 40), primero confirmo eso con un condicional `if`, en el que utilizo aritmética de apuntadores restando el final con el principio de cada arreglo y luego los comparo. Una vez hecho eso, creo dos apuntadores, donde cada uno apunta al principio de cada arreglo, esto para no mover de lugar los originales y para escribir menos. Luego doy inicio a un ciclo `while` que se va a repetir desde el principio del arreglo a comparar (podía ser cualquiera, ya que son del mismo tamaño), hasta que el apuntador quede en donde termina el mismo arreglo.
Dentro del ciclo `while` hay un condicional `if` en donde comparo si lo que hay en el principio del arreglo base, es lo mismo que hay en el arreglo a comparar, en caso de que sea verdad, le sumo uno a cada apuntador para comparar los siguientes, en caso de que haya una diferencia, la función regresa `UNEQUAL` y por ende termina la función, en caso de que nunca haya uno diferente, el ciclo `while` termina sin entrar en el `else`, y regresa `EQUAL`.



2. Explique su algoritmo para determinar que un arreglo es subarreglo de otro.

```
41          //Cuando la lista base es mas grande
42          else if(base_list_end - base_list_begin > list_to_compare_end - list_to_compare_begin){
43              int *alc = list_to_compare_begin;
44              int *alb = base_list_begin;
45              int t = 0;
46              int *tmp = NULL;
47              while(alb < base_list_end){
48                  if(*alc == *alb){
49                      alc++;
50                      t++;
51                      if(t == 1){
52                          tmp = alb;
53                      }
54                      else if(t == list_to_compare_end - list_to_compare_begin){
55                          return SUBLIST;
56                      }
57                      alb++;
58                  }
59                  else{
60                      alc = list_to_compare_begin;
61                      if(t == 0){
62                          alb++;
63                      }
64                      else{
65                          t = 0;
66                          alb = tmp + 1;
67                      }
68                  }
69              }
70              return UNEQUAL;
71          }
```

Lo primero es comprobar que el arreglo base sea más grande, eso lo hice con aritmética de apuntadores (línea 42). Aquí lo que hago es un ciclo para recorrer todo el arreglo base para buscar que estén todos los números del arreglo a comparar, así mismo, debo corroborar que estén en el mismo orden, sin importar en que parte del arreglo base, pero que vayan seguidos y en el mismo orden, entonces cada vez que encuentro al primero en coincidir, guardo esa dirección en un apuntador temporal, esto por si el orden se rompe, y así poder volver a comenzar una casilla después de esa. También puse un contador t, el cual me decía si los encontraba a todos, porque podía darse el caso de que encontrara solo a la mitad en el mismo orden, y al final, entonces eso me permitía saber si fueron todos, en el momento en que los encontraba a todos, regresaba `SUBLIST`, y en caso contrario, `UNEQUAL`.

3. Explique su algoritmo para determinar que un arreglo es superarreglo de otro.

```
72          //Cuando la lista a comparar es más grande
73          else{
74              int *alc = list_to_compare_begin;
75              int *alb = base_list_begin;
76              int t = 0;
77              int *tmp = NULL;
78              while(alc < list_to_compare_end){
79                  if(*alc == *alb){
80                      alb++;
81                      t++;
82                      if(t == 1){
83                          tmp = alc;
84                      }
85                      else if(t == base_list_end - base_list_begin){
86                          return SUPERLIST;
87                      }
88                      alc++;
89                  }
90                  else{
91                      alb = base_list_begin;
92                      if(t == 0){
93                          alc++;
94                      }
95                      else{
96                          t = 0;
97                          alc = tmp + 1;
98                      }
99                  }
100             }
101             return UNEQUAL;
102         }
```

De la línea 73 a la 102, esta el algoritmo para una superlista, que es el caso cuando el arreglo a comparar es más grande que el arreglo base, básicamente es el mismo código que el de la sublista, con la diferencia de que cambio de lugar las variables del arreglo base con las del arreglo a comparar, y viceversa.

4. ¿Qué es una enumeración (enum) en el lenguaje C?

Es definir un conjunto de cadenas de texto, en donde cada cadena de texto equivale a un valor entero (int), que en este caso, será el índice en el que fue colocado en enum.

5. ¿Cuál es el nombre de la prueba (test) que le resultó más difícil pasar?

Pues en sí no estuvieron difíciles, pero digamos que las que me tomaron más tiempo en pasar, fueron las de sublistas y superlistas, esto fue porque leí mal las indicaciones y tuve que volver a escribirlas porque al principio solo checaba que estuvieran todos los valores, no que estuvieran en el mismo orden y todos seguidos.

6. ¿Hubo alguna prueba (test) que no pudo pasar? ¿Cuál fue? ¿Qué errores obtuvo?

No, si las pude pasar todas.

7. ¿Aplicó la refactorización en su código? Explique de qué manera lo hizo

Sí, al principio resolví todo dentro de esa misma función, entonces creé más funciones para separar el código, dejando únicamente los casos base en la función original, eso también ayudo a no repetir el código de superlista y sublista, a continuación pongo cada función.

Prototipos de funciones y función principal

```c
#include "sublist.h"

comparison_result_t iguales(int *alc, int *alb, int *blc);

comparison_result_t tipo_arr(int *alc, int *alb, int *blc, int *blb);

comparison_result_t check_lists(int *list_to_compare_begin, int *list_to_compare_end,
                                int *base_list_begin, int *base_list_end){
    if(list_to_compare_begin == NULL && base_list_begin == NULL){
        return EQUAL;
    }
    else if(base_list_begin != NULL && list_to_compare_begin == NULL){
        return SUBLIST;
    }
    else if(list_to_compare_begin != NULL && base_list_begin == NULL){
        return SUPERLIST;
    }
    else{
        if(list_to_compare_end - list_to_compare_begin == base_list_end - base_list_begin){
            return iguales(list_to_compare_begin, base_list_begin, list_to_compare_end);
        }
        else if(base_list_end - base_list_begin > list_to_compare_end - list_to_compare_begin){
            return tipo_arr(list_to_compare_begin, base_list_begin, list_to_compare_end, base_list_end);
        }
        else{
            if(tipo_arr(base_list_begin, list_to_compare_begin, base_list_end, list_to_compare_end) == 1){
                return UNEQUAL;
            }
            else{
                return SUPERLIST;
            }
        }
    }
}
```

Función para comprobar si son iguales

```c
comparison_result_t iguales(int *alc, int *alb, int *blc){
    while(alc < blc){
        if(*alc == *alb){
            alc++;
            alb++;
        }
        else{
            return UNEQUAL;
        }
    }
    return EQUAL;
}
```

Función para saber si es sublista o superlista

```c
comparison_result_t tipo_arr(int *alc, int *alb, int *blc, int *blb){
    int *beg = alc;
    int *beg2 = blc;
    int t = 0;
    int *tmp = NULL;
    while(alb < blb){
        if(*alc == *alb){
            alc++;
            t++;
            if(t == 1){
                tmp = alb;
            }
            else if(t == beg2 - beg){
                return SUBLIST;
            }
            alb++;
        }
        else{
            alc =  beg;
            if(t == 0){
                alb++;
            }
            else{
                t = 0;
                alb = tmp + 1;
            }
        }
    }
    return UNEQUAL;
}
```

8. ¿Qué hizo particularmente bien en esta tarea?

Aplicar correctamente aritmética de apuntadores en mi código, así como la implementación de apuntadores en funciones.

9. ¿Qué pudo haber hecho mejor en la tarea?

Tomarme el tiempo suficiente para leer a detalle los requerimientos de la tarea, para no tener que volver a reescrbir mi código.

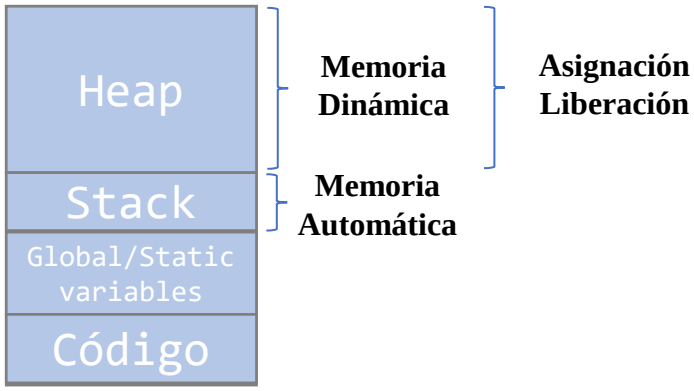
10. ¿Qué conocimientos nuevos adquirió?

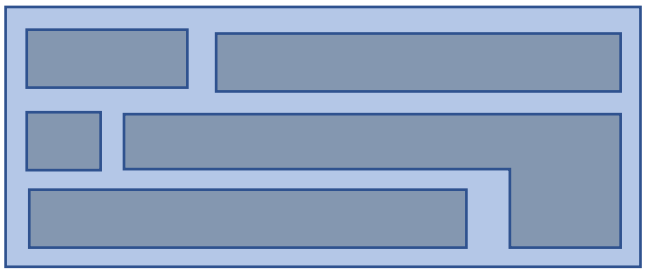
Primeramente, el `enum`, porque no lo conocía, en segunda, esta tarea me ayudo a terminar de entender la aritmética de apuntadores.

1.6 Gestión de memoria dinámica

Lunes 29 Agosto 2022

Memoria





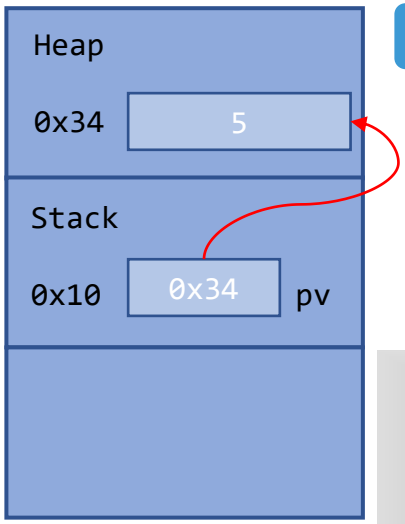
Fragmentación

Asignación

```
void *malloc(size_t);
Regresa un apuntador con la dirección del primer elemento del bloque.
Ejemplo:
int *pv = (int *)malloc(sizeof(int));
*pv = 5;
printf("%d\n", *pv); //output: 5
```

size_t es un tipo de dato unsigned int. En este caso especifico la cantidad de bytes por asignar.

void * regresa un apuntador con la dirección del primer elemento del bloque, y es void para poder usarlo en todo apuntador genérico.



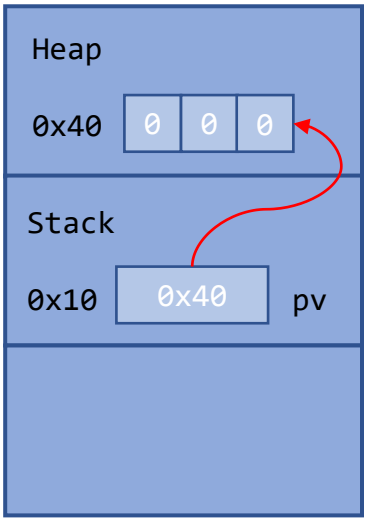
Liberación

```
void free(void *);
free(pv);
printf("%d\n", *pv); //output: random number
```

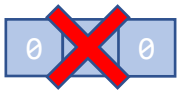
free() devuelve la memoria al sistema, ya no te pertenece

Para poder utilizar malloc() y free(), se debe agregar la librería stdlib.h

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(){
4      int *pv = (int *)malloc(3*sizeof(int));
5      *pv = 0;
6      *(pv + 1) = 0;
7      *(pv + 2) = 0;
8      free(pv);
9      return 0;
10 }
```



free(pv);

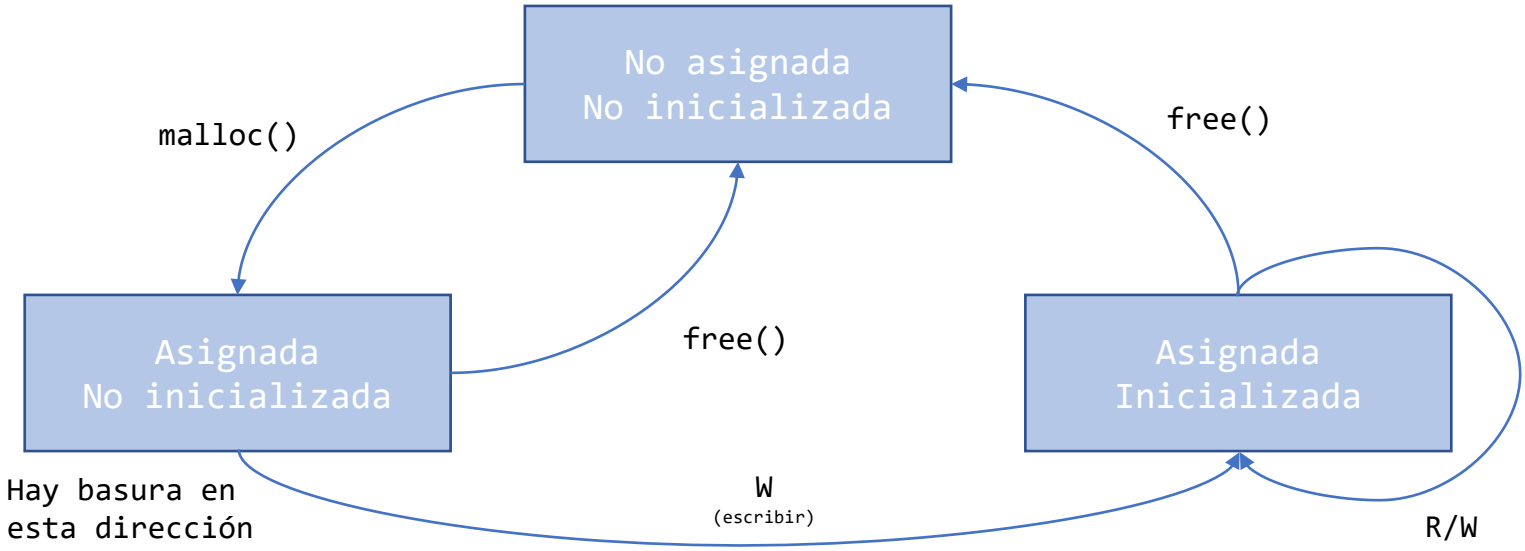


¿Cómo enterarnos si la asignación de memoria falla?

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    int *pv = NULL;
    pv = (int *)malloc(sizeof(int));
    if(pv == NULL){
        printf("Error: no hay memoria disponible");
        exit(EXIT_FAILURE);
    }
    /*codigo*/
    return 0;
}
```

1.7 Estados de memoria dinámica

Martes 30 Agosto 2022



1.8 Problemas en el manejo de memoria dinámica

Fragmentación de memoria, fuera de nuestro alcance, no hay un bloque contiguo lo suficientemente grande.

1) Fugas de memoria (memory leaks)

```c
char *bytes;
while(1){
    bytes = (char *)malloc(10);
}
```

Un ciclo infinito que en algún punto creará una fuga de memoria porque no hay memoria infinita.

```c
double *d = NULL;
d = (double *)malloc(100);
double pi = 3.1416;
d = &pi;
free(d);
```

Al hacer eso, perdemos la dirección de memoria en donde teníamos guardado los 100 de memoria.
Lo correcto es hacer otro apuntador (en este caso f) en donde guardemos la dirección antes de cambiar el apuntador.

```c
double *d = NULL;
d = (double *)malloc(100);
double pi = 3.1416;
double *f = d;
d = &pi;
free(f);
```

Por cada malloc() asigna un free()

2) Wild pounters (apuntadores no inicializados)
Apuntadores que no apuntan a nada.

Cada puntero se inicializa con un NULL

```c
int *p = NULL;
```

3) Dangling pointers (apuntadores a una dirección de memoria que ya fue liberada)

Asignar NULL al apuntador después de free()

```c
int *p = NULL;
p = (int *)malloc(sizeof(int));
free(p);
p = NULL;
```

1.9 Arreglos dinámicos

Jueves 01 Septiembre 2022

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
int *crea_arreglo(int);
int main(){
    int n;
    scanf("%d", &n);
    int *a = crea_arreglo(n);
    for(int i = 0; i < n; i++){
        scanf("%d", &a[i]);
    }
    for(int j = 0; j < n; j++){
        printf("%d ", a[j]);
    }
    printf("\n");
    free(a);
    return 0;
}
int *crea_arreglo(int tam){
    int *arreglo = NULL;
    arreglo = (int *)malloc(tam * sizeof(int));
    assert(arreglo != NULL);
    return arreglo;
}
```

Las líneas 9 y 12 las podemos cambiar, o bien, cambiar los dos ciclos for, tal como se muestra a continuación y exactamente lo mismo:

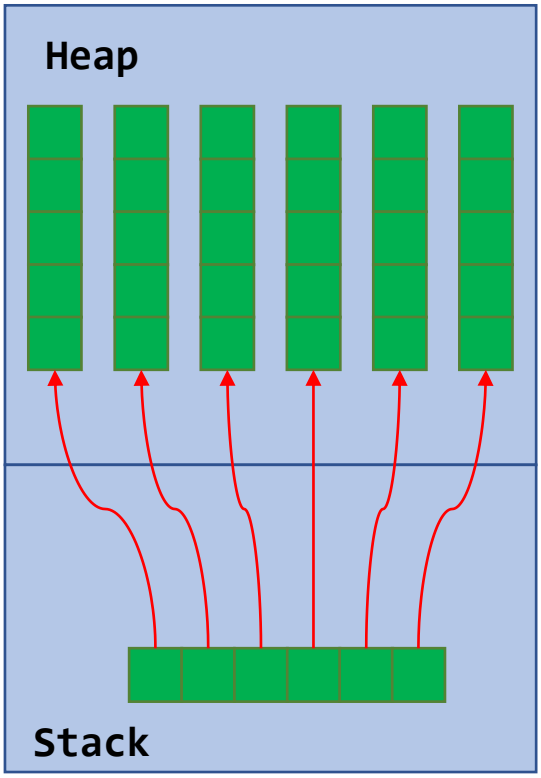
```c
9           scanf("%d", a + i);
12          printf("%d ", *(a + j));
```

```c
8       for(int *pi = a; pi < a + n; pi++){
9           scanf("%d", pi);
10      }
11      for(int *pj = a; pj < a + n; pj++){
12          printf("%d\n", *pj);
13      }
```

1.10 Matriz dinámica

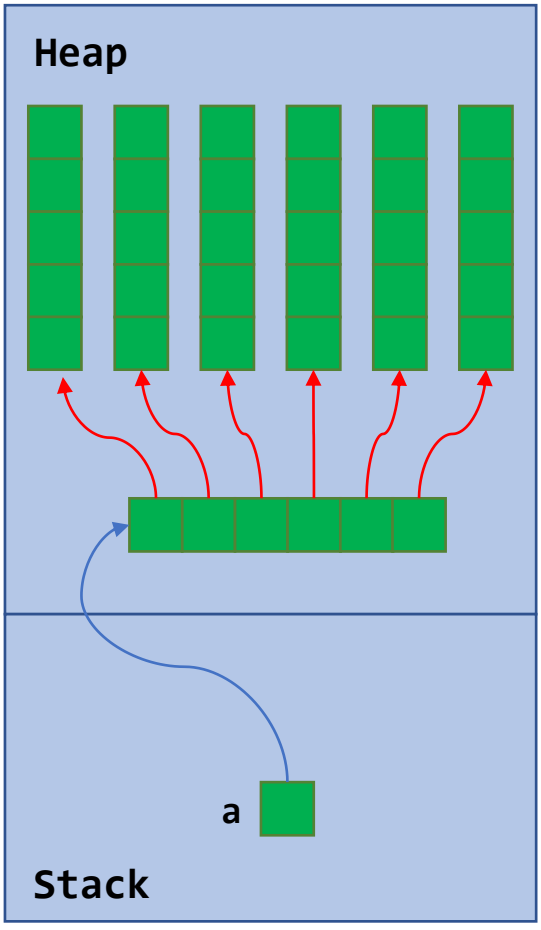
Viernes 02 Septiembre 2022

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
int main(){
    int *a[6] = {NULL};
    for(int i = 0; i < 6; i++){
        a[i] = (int *)malloc(5 * sizeof(int));
        assert(a[i] != NULL);
    }
    for(int j = 0; j < 6; j++){
        free(a[j]);
    }
    return 0;
}
```



También podemos liberar la matriz de esta forma:

```c
    int **p = a;
    for(int j = 0; j < 6; j++){
        free(*(p + j));
    }
    return 0;
}
```




```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
int main(){
    int **a = NULL;
    a = (int **)malloc(6 * sizeof(int *));
    assert(a != NULL);
    for(int i = 0; i < 6; i++){
        a[i] = (int *)malloc(5 * sizeof(int));
        assert(a[i] != NULL);
    }
    int **p = a;
    for(int j = 0; j < 6; j++){
        free(*(p + j));
    }
    free(a);
    return 0;
}
```

Implementación de una matriz dinámica con funciones

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
int **crea_matriz(int, int);
void llenar_matriz(int **, int, int);
void imprimir_matriz(int **, int, int);
void liberar_matriz(int **);
int main(){
    int r, c;
    scanf("%d", &r);
    scanf("%d", &c);
    int **a = crea_matriz(r, c);
    llenar_matriz(a, r, c);
    imprimir_matriz(a, r, c);
    liberar_matriz(a);
    return 0;
}

int **crea_matriz(int r, int c){
    int **a = NULL;
    a = (int **)malloc(c * sizeof(int *));
    assert(a != NULL);
    for(int i = 0; i < c; i++){
        a[i] = (int *)malloc(r * sizeof(int));
        assert(a[i] != NULL);
    }
    return a;
}

void llenar_matriz(int **matriz, int r, int c){
    int **a = matriz;
    for(int i = 0; i < c; i++){
        for(int j = 0; j < r; j++){
            scanf("%d", &a[i][j]);
        }
    }
}

void imprimir_matriz(int **matriz, int r, int c){
    int **a = matriz;
    for(int i = 0; i < c; i++){
        for(int j = 0; j < r; j++){
            printf("%d ", a[i][j]);
        }
        printf("\n");
    }
}

void liberar_matriz(int **matriz){
    int **p = matriz;
    for(int j = 0; j < 6; j++){
        free(*(p + j));
    }
    free(matriz);
}
```

Tarea #2: Arreglos Dinámicos

Minesweeper

If you need help running the tests or submitting your code, check out HELP.md.

Instructions

Add the mine counts to a completed Minesweeper board.

Minesweeper is a popular game where the user has to find the mines using
numeric hints that indicate how many mines are directly adjacent
(horizontally, vertically, diagonally) to a square.

In this exercise you have to create some code that counts the number of
mines adjacent to a given empty square and replaces that square with the
count.

The board is a rectangle composed of blank space (' ') characters. A mine
is represented by an asterisk ('*') character.

If a given space has no adjacent mines at all, leave that square blank.

Examples

For example, you may receive a 5 x 4 board like this (empty spaces are
represented here with the '·' character for display on screen):

```
.*.*.
..*..
..*..
.....
```

And your code will transform it into this:

```
1*3*1
13*31
·2*2·
·111·
```

minesweeper.h

```h
#ifndef MINESWEEPER_H
#define MINESWEEPER_H
#include <stddef.h>

char **annotate(const char **minefield, const size_t rows);
void free_annotation(char **annotation, const size_t rows);

#endif
```

test_minesweeper.c

```c
#include "test-framework/unity.h"
#include "minesweeper.h"
#include <stdlib.h>
#define ARRAY_SIZE(arr) (sizeof(arr) / sizeof(arr[0]))
void setUp(void){
}

void tearDown(void){
}

static void test_annotate_no_rows(void){
   TEST_IGNORE();   // delete this line to run test
   const char **minefield = NULL;
   const size_t rows = 0;
   char **actual = annotate(minefield, rows);
   TEST_ASSERT_NULL(actual);
}

static void test_annotate_no_columns(void){
   TEST_IGNORE();
   const char *minefield[] = { "" };
   const char *expected[] = { "" };
   const size_t rows = ARRAY_SIZE(expected);
   char **actual = annotate(minefield, rows);
   TEST_ASSERT_EQUAL_STRING_ARRAY(expected, actual, rows);
   free_annotation(actual, rows);
}

static void test_annotate_no_mines(void){
   TEST_IGNORE();
   const char *minefield[] = {
      // clang-format off
      "   ",
      "   ",
      "   "
      // clang-format on
   };
   const char *expected[] = {
      // clang-format off
      "   ",
      "   ",
      "   "
      // clang-format on
   };
   const size_t rows = ARRAY_SIZE(expected);
   char **actual = annotate(minefield, rows);
   TEST_ASSERT_EQUAL_STRING_ARRAY(expected, actual, rows);
   free_annotation(actual, rows);
}
```

```c
static void test_annotate_minefield_with_only_mines(void){
   TEST_IGNORE();
   const char *minefield[] = {
      // clang-format off
      "***",
      "***",
      "***"
      // clang-format on
   };
   const char *expected[] = {
      // clang-format off
      "***",
      "***",
      "***"
      // clang-format on
   };
   const size_t rows = ARRAY_SIZE(expected);
   char **actual = annotate(minefield, rows);
   TEST_ASSERT_EQUAL_STRING_ARRAY(expected, actual, rows);
   free_annotation(actual, rows);
}

static void test_annotate_mine_surrounded_by_spaces(void){
   TEST_IGNORE();
   const char *minefield[] = {
      // clang-format off
      "   ",
      " * ",
      "   "
      // clang-format on
   };
   const char *expected[] = {
      // clang-format off
      "111",
      "1*1",
      "111"
      // clang-format on
   };
   const size_t rows = ARRAY_SIZE(expected);
   char **actual = annotate(minefield, rows);
   TEST_ASSERT_EQUAL_STRING_ARRAY(expected, actual, rows);
   free_annotation(actual, rows);
}

static void test_annotate_space_surrounded_by_mines(void){
   TEST_IGNORE();
   const char *minefield[] = {
      // clang-format off
      "***",
      "* *",
      "***"
      // clang-format on
   };
```

```
    const char *expected[] = {
        // clang-format off
        "***",
        "*8*",
        "***"
        // clang-format on
    };
    const size_t rows = ARRAY_SIZE(expected);
    char **actual = annotate(minefield, rows);
    TEST_ASSERT_EQUAL_STRING_ARRAY(expected, actual, rows);
    free_annotation(actual, rows);
}

static void test_annotate_horizontal_line(void){
    TEST_IGNORE();
    const char *minefield[] = {
        // clang-format off
        " * * "
        // clang-format on
    };
    const char *expected[] = {
        // clang-format off
        "1*2*1"
        // clang-format on
    };
    const size_t rows = ARRAY_SIZE(expected);
    char **actual = annotate(minefield, rows);
    TEST_ASSERT_EQUAL_STRING_ARRAY(expected, actual, rows);
    free_annotation(actual, rows);
}

static void test_annotate_horizontal_line_mines_at_edges(void){
    TEST_IGNORE();
    const char *minefield[] = {
        // clang-format off
        "*   *"
        // clang-format on
    };
    const char *expected[] = {
        // clang-format off
        "*1 1*"
        // clang-format on
    };
    const size_t rows = ARRAY_SIZE(expected);
    char **actual = annotate(minefield, rows);
    TEST_ASSERT_EQUAL_STRING_ARRAY(expected, actual, rows);
    free_annotation(actual, rows);
}

static void test_annotate_vertical_line(void){
    TEST_IGNORE();
    const char *minefield[] = {
        // clang-format off
        " ",
        "*",
        " ",
        "*",
        " "
        // clang-format on
    };
```

```
    const char *expected[] = {
        // clang-format off
        "1",
        "*",
        "2",
        "*",
        "1"
        // clang-format on
    };
    const size_t rows = ARRAY_SIZE(expected);
    char **actual = annotate(minefield, rows);
    TEST_ASSERT_EQUAL_STRING_ARRAY(expected, actual, rows);
    free_annotation(actual, rows);
}

static void test_annotate_vertical_line_mines_at_edges(void){
    TEST_IGNORE();
    const char *minefield[] = {
        // clang-format off
        "*",
        " ",
        " ",
        " ",
        "*"
        // clang-format on
    };
    const char *expected[] = {
        // clang-format off
        "*",
        "1",
        " ",
        "1",
        "*"
        // clang-format on
    };
    const size_t rows = ARRAY_SIZE(expected);
    char **actual = annotate(minefield, rows);
    TEST_ASSERT_EQUAL_STRING_ARRAY(expected, actual, rows);
    free_annotation(actual, rows);
}

static void test_annotate_cross(void){
    TEST_IGNORE();
    const char *minefield[] = {
        // clang-format off
        "  *  ",
        "  *  ",
        "*****",
        "  *  ",
        "  *  "
        // clang-format on
    };
    const char *expected[] = {
        // clang-format off
        " 2*2 ",
        "25*52",
        "*****",
        "25*52",
        " 2*2 "
        // clang-format on
    };
    const size_t rows = ARRAY_SIZE(expected);
    char **actual = annotate(minefield, rows);
    TEST_ASSERT_EQUAL_STRING_ARRAY(expected, actual, rows);
    free_annotation(actual, rows);
}
```

```c
static void test_annotate_large_minefield(void){
   TEST_IGNORE();
   const char *minefield[] = {
      // clang-format off
      " *  * ",
      "  *   ",
      "    * ",
      "   * *",
      " *  * ",
      "      "
      // clang-format on
   };
   const char *expected[] = {
      // clang-format off
      "1*22*1",
      "12*322",
      " 123*2",
      "112*4*",
      "1*22*2",
      "111111"
      // clang-format on
   };
   const size_t rows = ARRAY_SIZE(expected);
   char **actual = annotate(minefield, rows);
   TEST_ASSERT_EQUAL_STRING_ARRAY(expected, actual, rows);
   free_annotation(actual, rows);
}

int main(void){
   UnityBegin("test_minesweeper.c");
   RUN_TEST(test_annotate_no_rows);
   RUN_TEST(test_annotate_no_columns);
   RUN_TEST(test_annotate_no_mines);
   RUN_TEST(test_annotate_minefield_with_only_mines);
   RUN_TEST(test_annotate_mine_surrounded_by_spaces);
   RUN_TEST(test_annotate_space_surrounded_by_mines);
   RUN_TEST(test_annotate_horizontal_line);
   RUN_TEST(test_annotate_horizontal_line_mines_at_edges);
   RUN_TEST(test_annotate_vertical_line);
   RUN_TEST(test_annotate_vertical_line_mines_at_edges);
   RUN_TEST(test_annotate_cross);
   RUN_TEST(test_annotate_large_minefield);
   return UnityEnd();
}
```

minesweeper.c

```c
#include "minesweeper.h"
#include <string.h>
#include <stdlib.h>
#include <assert.h>

char **annotate(const char **minefield, const size_t rows){
    //Caso base cuando no existe la matriz
    if(minefield == NULL){
        return NULL;
    }
    int c = strlen(minefield[0]);
    int cont = 0;
    int r = rows;
    //Crear la matriz
    char **a = NULL;
    a = (char **)malloc(r * sizeof(char *));
    assert(a != NULL);
    for(int i = 0; i < r; i++){
        a[i] = (char *)malloc((c + 1) * sizeof(char));
        assert(a[i] != NULL);
    }
    //Copiando la matriz para ya tener asteriscos y espacios
    for(int i = 0; i < r; i++){
        for(int j = 0; j < c + 1; j++){
            if(j != c){
                a[i][j] = minefield[i][j];
            }
            else{
                a[i][j] = '\0';
            }
        }
    }
    //Caso general para todas las matrices
    for(int i = 0; i < r; i++){
        for(int j = 0; j < c; j++){
            //Si no es un asterisco
            if(a[i][j] != '*'){
                //Fijamos los limites para revisar
                int r_begin, r_end, c_begin, c_end;
                //Si estamos al principio de un renglon
                if(i == 0){
                    r_begin = 0;
                }
                else{
                    r_begin = i - 1;
                }
                //Si estamos al final de un renglon
                if(i == r - 1){
                    r_end = r - 1;
                }
```

```
51                  else{
52                      r_end = i + 1;
53                  }
54                  //Si estamos en la primer columna
55                  if(j == 0){
56                      c_begin = 0;
57                  }
58                  else{
59                      c_begin = j - 1;
60                  }
61                  //Si estamos en la ultima columna
62                  if(j == c - 1){
63                      c_end = c - 1;
64                  }
65                  else{
66                      c_end = j + 1;
67                  }
68                  //Revisamos si los de alrededor son asteriscos
69                  for(int k = r_begin; k < r_end + 1; k++){
70                      for(int l = c_begin; l <= c_end; l++){
71                          if(a[k][l] == '*'){
72                              cont++;
73                          }
74                      }
75                  }
76                  //Si hay por lo menos un asterisco cerca
77                  if(cont != 0){
78                      a[i][j] = cont + '0';
79                      cont = 0;
80                  }
81              }
82          }
83      }
84      return a;
85  }
86
87  void free_annotation(char **annotation, const size_t rows){
88      char **p = annotation;
89      int r = rows;
90      for(int i = 0; i < r; i++){
91          free(*(p + i));
92      }
93      free(annotation);
94  }
```

1. Explica tus algoritmos para crear y liberar la memoria dinámica utilizada.

```
14      //Crear la matriz
15      char **a = NULL;
16      a = (char **)malloc(r * sizeof(char *));
17      assert(a != NULL);
18      for(int i = 0; i < r; i++){
19          a[i] = (char *)malloc((c + 1) * sizeof(char));
20          assert(a[i] != NULL);
21      }


87  void free_annotation(char **annotation, const size_t rows){
88      char **p = annotation;
89      int r = rows;
90      for(int i = 0; i < r; i++){
91          free(*(p + i));
92      }
93      free(annotation);
94  }
```

Para la creación de la memoria dinámica (líneas de la 14 a la 21), y la liberación de memoria (líneas de la 87 a la 94) utilicé el código aprendido el viernes 02 de septiembre, en donde primero es crear un apuntador que apunta a otro apuntador, primero, ya sabemos cuantos renglones hay, entonces creo un arreglo dinámico de apuntadores del mismo tamaño de los renglones, luego cada casilla de memoria (cada uno de los renglones), apunta a otra cadena de memoria del tamaño de cada renglón + 1 (el más uno es para poder el final de cadena `'\0'`) el tamaño de cada renglón lo obtuve en la línea 11 con la función `strlen()`. Así mismo, aplico el `assert()` para comprobar que hay memoria suficiente.
Para la liberación de la memoria, también aplico el código visto en clase el día 02, al liberar primero cada uno de los renglones, y posteriormente liberar la matriz de renglones.

2. Explica tu algoritmo para determinar cuántas minas hay alrededor de cada celda.

```
34      for(int i = 0; i < r; i++){
35          for(int j = 0; j < c; j++){
36              //Si no es un asterisco
37              if(a[i][j] != '*'){
38                  //Fijamos los limites para revisar
39                  int r_begin, r_end, c_begin, c_end;
40                  //Si estamos al principio de un renglon
41                  if(i == 0){
42                      r_begin = 0;
43                  }
44                  else{
45                      r_begin = i - 1;
46                  }
47                  //Si estamos al final de un renglon
48                  if(i == r - 1){
49                      r_end = r - 1;
```

```
50                  }
51                  else{
52                      r_end = i + 1;
53                  }
54                  //Si estamos en la primer columna
55                  if(j == 0){
56                      c_begin = 0;
57                  }
58                  else{
59                      c_begin = j - 1;
60                  }
61                  //Si estamos en la ultima columna
62                  if(j == c - 1){
63                      c_end = c - 1;
64                  }
65                  else{
66                      c_end = j + 1;
67                  }
68                  //Revisamos si los de alrededor son asteriscos
69                  for(int k = r_begin; k < r_end + 1; k++){
70                      for(int l = c_begin; l <= c_end; l++){
71                          if(a[k][l] == '*'){
72                              cont++;
73                          }
74                      }
75                  }
76                  //Si hay por lo menos un asterisco cerca
77                  if(cont != 0){
78                      a[i][j] = cont + '0';
79                      cont = 0;
80                  }
81              }
82          }
83      }
```

Primero hago dos cliclos `for` para hacer el recorrido por toda la matriz y compruebo que estoy en una casilla en blanco, al llegar a una, hago varios condicioales para saber si estoy en alguna de las orillas de la matriz o no, para saber que de donde a donde debo checar, y de este moso, no acceder a memoria que no me pertenece.
Luego de fijar los limites a revisar, hago otros dos ciclos for para revisar todas las casillas de alrededor en busca de asteriscos, en caso de encontrar una, aumento en uno mi contador de minas, y al final pongo ese numero en forma de char, y reinicio mi contador de bombas para la siguiente casilla en blanco.

3. ¿Qué es apuntador constante (const) en el lenguaje C? ¿En cuáles situaciones se recomienda utilizarlo?

Es cuando el valor al que apunta el apuntador va a ser constante

```
const int *p; / / El contenido apunta al puntero es constante
int const *P; / / El contenido apunta al puntero es constante.
int * const page; / / puntero es constante
const int * const p; // El contenido apunta al puntero y el puntero es constante.
```

Se recomienda utilizarlo cuando solo vamos a leer los datos y no a modificarlos.

4. ¿Cuál es el nombre de la prueba (test) que te resultó más difícil pasar? ¿Por qué consideras que fue difícil?

Ninguna.

5. ¿Hubo alguna prueba (test) que no se pudo pasar? ¿Cuál fue? ¿Qué errores obtuviste?

No.

6. Inventa una nueva prueba y agrégala al conjunto de pruebas del proyecto. Agrega el código fuente y explicación de tu prueba en el reporte.

```c
static void test_esquinas(void){
   //TEST_IGNORE();
   const char *minefield[] = {
      // clang-format off
      "* *",
      "   ",
      "* *"
      // clang-format on
   };
   const char *expected[] = {
      // clang-format off
      "*2*",
      "242",
      "*2*"
      // clang-format on
   };
   const size_t rows = ARRAY_SIZE(expected);
   char **actual = annotate(minefield, rows);
   TEST_ASSERT_EQUAL_STRING_ARRAY(expected, actual, rows);
   free_annotation(actual, rows);
}

```

```c
static void test_one_void(void){
   //TEST_IGNORE();
   const char *minefield[] = { " " };
   const char *expected[] = { " " };
   const size_t rows = ARRAY_SIZE(expected);
   char **actual = annotate(minefield, rows);
   TEST_ASSERT_EQUAL_STRING_ARRAY(expected, actual, rows);
   free_annotation(actual, rows);
}

static void test_one_bomb(void){
   //TEST_IGNORE();
   const char *minefield[] = { "*" };
   const char *expected[] = { "*" };
   const size_t rows = ARRAY_SIZE(expected);
   char **actual = annotate(minefield, rows);
   TEST_ASSERT_EQUAL_STRING_ARRAY(expected, actual, rows);
   free_annotation(actual, rows);
}
```

Agregué 3 pruebas, la primera es cuando únicamente las esquinas tienen bombas, la segunda cuando solo hay una casilla en blanco, y la tercera cuando solo hay una bomba.

7. ¿Aplicaste la refactorización en tu código? Explica de qué manera lo hiciste

```c
                //Fijamos los limites para revisar
                int r_begin = (i == 0) ? 0: i - 1;
                int r_end = (i == r - 1) ? r - 1 : i + 1;
                int c_begin = (j == 0) ? 0: j - 1;
                int c_end = (j == c - 1) ? c - 1 : j + 1;
```

Para no ocupar tantas líneas en los if else para fijar los limites, lo cambie por esa otra estructura.

8. ¿Qué hiciste particularmente bien en esta tarea?

Aplicar los conocimientos de los arreglos dinámicos para la creación y liberación de memoria.

9. ¿Qué pudiste haber hecho mejor en esta tarea?

Honestamente, no lo sé.

10. ¿Qué nuevos conocimientos y experiencias adquiriste con esta tarea?

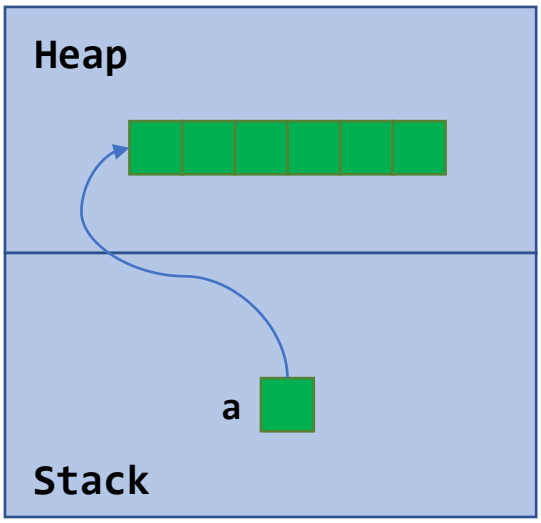
A utilizar el ? y los : en vez de if y else.

1.11 Apuntadores por “referencia”

Lunes 05 Septiembre 2022

V1

```c
int *crea_arreglo(int tam){
    int *arreglo = NULL;
    arreglo = (int *)malloc(tam * sizeof(int));
    assert(arreglo != NULL);
    return arreglo;
}
```



V2

```c
void crea_arreglo(int **arreglo, int tam){
    *arreglo = (int *)malloc(tam * sizeof(int));
    assert(*arreglo != NULL);
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
void crea_arreglo(int **, int);
int main(){
    int *arreglo = NULL;
    crea_arreglo(&arreglo, 10);
    for(int i = 0; i < 10; i++){
        scanf("%d", arreglo + i);
    }
    for(int i = 0; i < 10; i++){
        printf("%d", *(arreglo + i));
    }
    free(arreglo);
    return 0;
}
void crea_arreglo(int **arreglo, int tam){
    *arreglo = (int *)malloc(tam * sizeof(int));
    assert(*arreglo != NULL);
}
```

1.12 realloc

Martes 06 Septiembre 2022

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    int *p = (int *)malloc(4 * sizeof(int));
    p = (int *)realloc(p, 6 * sizeof(int));
    p = (int *)realloc(p, 2 * sizeof(int));
    free(p);
    return 0;
}
```

La función realloc es para cambiar el tamaño de la memoria almacenada.

1.13 Pila dinámica

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
typedef struct{
    int *arreglo;
    int tope;
    int max;
}pila_dinamica;

void inicializa_pila(pila_dinamica *);
void push(pila_dinamica *, int);
void pop(pila_dinamica *);
void imprimir_pila(pila_dinamica *);

int main(){
    pila_dinamica pila;
    inicializa_pila(&pila);
    push(&pila, 2);
    push(&pila, 34);
    push(&pila, 146);
    imprimir_pila(&pila);
    pop(&pila);
    push(&pila, 2);
    imprimir_pila(&pila);
    free(pila.arreglo);
    return 0;
}

void inicializa_pila(pila_dinamica *p){
    p->arreglo = NULL;
    p->tope = p->max = 0;
}

void push(pila_dinamica *p, int dato){
    if(p->tope == p->max){
        p->max = p->max == 0 ? 1 : p->max * 2;
        int *q = (int *)realloc(p->arreglo, p->max * sizeof(int));
        assert(q != NULL);
        p->arreglo = q;
    }
    p->arreglo[p->tope++] = dato;
}

void pop(pila_dinamica *p){
    if(p->tope > 0){
        int *q = (int *)realloc(p->arreglo, (p->tope - 1) * sizeof(int));
        assert(q != NULL);
        p->arreglo = q;
        p->tope--;
    }
}

void imprimir_pila(pila_dinamica *p){
    if(p->tope != 0){
        for(int i = 0; i < p->tope; i++){
            printf("%d ", p->arreglo[i]);
        }
        printf("\n");
    }
    else{
        printf("La pila esta vacia\n");
    }
}
```

1.14 Estructuras dinámicas

Jueves 08 Septiembre 2022

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

typedef struct{
    char nombre[50];
    int edad;
}persona_t;

typedef struct{
    char *nombre;
    int edad;
}dpersona_t;

int main(){
    persona_t persona;
    printf("Nombre: ");
    scanf("%[^\n]", persona.nombre);
    printf("Edad: ");
    scanf("%d", &persona.edad);
    persona_t persona2 = persona;
    printf("Nombre: %s\n", persona2.nombre);
    printf("Edad: %d\n", persona2.edad);
    dpersona_t persona_d;
    char nombre[50];
    char espacio;
    scanf("%c", &espacio);
    scanf("%[^\n]", nombre);
    persona_d.nombre = (char *)malloc(strlen(nombre) + 1);
    strcpy(persona_d.nombre, nombre);
    printf("Nombre dinamico: %s\n", persona_d.nombre);
    persona_t personas[10];
    dpersona_t *personas_d = NULL;
    personas_d = (dpersona_t *)malloc(10 * sizeof(dpersona_t));
    return 0;
}
```

1.15 Apuntadores genéricos

Viernes 09 Septiembre 2022

void * = apuntador genérico porque puede apuntar a cualquier cosa

No se puede usar indirección sobre void * (se debe hacer cast)

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

typedef struct{
    char nombre[50];
    int edad;
}persona_t;

int main(){
    int a = 100;
    float b = 3.1416;
    persona_t c;
    strcpy(c.nombre, "Hugo Esparza");
    c.edad = 26;
    void *p = NULL;
    p = &a;
    printf("%d\n", *(int *)p);
    p = &b;
    printf("%f\n", *(float *)p);
    p = &c;
    printf("%s\n", ((persona_t *)p)->nombre);
    return 0;
}
```

p->nombre ⇒ ((persona_t *)p)->nombre (*p).nombre ⇒ (*(persona_t *)p).nombre

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

int main(){
    void *p;
    p = malloc(10);
    printf("Dame el valor del entero: ");
    scanf("%d", (int *)p);
    printf("El entero es: %d\n", *(int *)p);
    printf("Dame el valor del flotante: ");
    scanf("%f", (float *)p);
    printf("El flotante es: %f\n", *(float *)p);
    return 0;
}
```

Trabajo en clase: “Arreglo dinámico”

Lunes 12 Septiembre 2022

Escribir un programa para escribir/leer datos en un arreglo dinámico por medio de un apuntador genérico. El usuario especificará el tamaño del arreglo y el tipo de dato seleccionado una de las siguientes opciones: 1 = int, 2 = char, 3 = float, 4 = alumno con clave (int) y nombre (char *). Se debe reservar memoria para el arreglo de acuerdo al tamaño y tipo de dato especificado por el usuario (todos los datos que contiene el arreglo serán del mismo tipo). Después de elegir el tipo de dato el usuario podrá escribir información en el arreglo e imprimir su contenido para verificar que la información se guardó correctamente. En cualquiera de los casos el arreglo se accederá por medio de una misma variable (void *). El programa terminará hasta que el usuario elija la opción 0 para salir, por lo que se deberá liberar la memoria antes de elegir un nuevo tipo de dato. Utilice notación/aritmética de apuntadores para acceder a los elementos del arreglo.

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>
#include <stdbool.h>

typedef enum{
    SALIR,
    ENTERO,
    LETRA,
    FLOTANTE,
    ALUMNO
}tipo_t;

typedef struct{
    int clave;
    char *nombre;
}alumno_t;

tipo_t seleccionar_tipo();
void crea_arreglo(void **, size_t, tipo_t);
int menu();
void escribir(void **, size_t, tipo_t);
void imprimir(void **, size_t, tipo_t);
void liberar_arreglo(void **, size_t, tipo_t);

int main(){
    void *arreglo = NULL;
    size_t tam;
    bool flag = true;
    int opt, o = 0;
    tipo_t tipo;
    do{
```

```
34          printf("Ingrese valor del tamaño: ");
35          scanf("%lu", &tam);
36          tipo = seleccionar_tipo();
37          crea_arreglo(&arreglo, tam, tipo);
38          do{
39              opt = menu();
40              o = 0;
41              if(opt == 3){
42                  o = 1;
43              }
44              else if(opt == 0){
45                  flag = false;
46                  o = 1;
47              }
48              else if(opt == 1){
49                  escribir(&arreglo, tam, tipo);
50              }
51              else if(opt == 2){
52                  imprimir(&arreglo, tam, tipo);
53              }
54              else{
55                  puts("opcion incorrecta");
56              }
57          }while(o != 1);
58          liberar_arreglo(&arreglo, tam, tipo);
59      }while(flag);
60      puts("Saliendo del programa...");
61      return 0;
62  }
63
64  tipo_t seleccionar_tipo(){
65      tipo_t t;
66      int b = 1;
67      do{
68          puts("Elige un tipo");
69          puts("1. int");
70          puts("2. char");
71          puts("3. float");
72          puts("4. Alumno");
73          printf("Ingresa numero de tipo: ");
74          scanf("%u", &t);
75          if(t == ENTERO){
76              return ENTERO;
77          }
78          else if(t == LETRA){
79              return LETRA;
```

```
80          }
81          else if(t == FLOTANTE){
82              return FLOTANTE;
83          }
84          else if(t == ALUMNO){
85              return ALUMNO;
86          }
87          else{
88              puts("Opcion no valida, vuelva a escoger");
89          }
90      }while(b != 0);
91  }
92
93  void crea_arreglo(void **arreglo, size_t tam, tipo_t tipo){
94      if(tipo == ENTERO){
95          *arreglo = malloc(tam * sizeof(int));
96          assert(*arreglo != NULL);
97      }
98      if(tipo == LETRA){
99          *arreglo = malloc(tam * sizeof(char));
100         assert(*arreglo != NULL);
101     }
102     if(tipo == FLOTANTE){
103         *arreglo = malloc(tam * sizeof(float));
104         assert(*arreglo != NULL);
105     }
106     if(tipo == ALUMNO){
107         *arreglo = malloc(tam * sizeof(alumno_t));
108         assert(*arreglo != NULL);
109     }
110 }
111
112 int menu(){
113     int op;
114     puts("Que desea hacer?");
115     puts("0. Salir");
116     puts("1. Escribir datos en el arreglo");
117     puts("2. Imprimir el arreglo");
118     puts("3. Cambiar tipo de dato (debera volver a crear la matriz)");
119     printf("Ingrese el numero de la opción: ");
120     scanf("%d", &op);
121     return op;
122 }
123
124 void escribir(void **a, size_t tam, tipo_t tipo){
125     if(tipo == ENTERO){
```

```
126             for(size_t i = 0; i < tam; i++){
127                 scanf("%d", ((int *)(*a) + i));
128             }
129         }
130         if(tipo == LETRA){
131             char s;
132             for(size_t i = 0; i < tam; i++){
133                 scanf("%c", &s);
134                 scanf("%c", ((char *)(*a) + i));
135             }
136         }
137         if(tipo == FLOTANTE){
138             for(size_t i = 0; i < tam; i++){
139                 scanf("%f", ((float *)(*a) + i));
140             }
141         }
142         if(tipo == ALUMNO){
143             char nombre[50], s;
144             int clave;
145             for(size_t i = 0; i < tam; i++){
146                 printf("Ingrese clave: ");
147                 scanf("%d", &((alumno_t *)(*a) + i)->clave);
148                 scanf("%c", &s);
149                 printf("Ingrese nombre: ");
150                 scanf("%s", nombre);
151                 ((alumno_t *)(*a) + i)->nombre = malloc(strlen(nombre) + 1);
152                 assert(((alumno_t *)(*a) + i)->nombre != NULL);
153                 strcpy(((alumno_t *)(*a) + i)->nombre, nombre);
154             }
155         }
156     }
157
158     void imprimir(void **a, size_t tam, tipo_t tipo){
159         if(tipo == ENTERO){
160             for(size_t i = 0; i < tam; i++){
161                 printf("%d ", *((int *)(*a) + i));
162             }
163         }
164         if(tipo == LETRA){
165             for(size_t i = 0; i < tam; i++){
166                 printf("%c ", *((char *)(*a) + i));
167             }
168         }
169         if(tipo == FLOTANTE){
170             for(size_t i = 0; i < tam; i++){
171                 printf("%f ", *((float *)(*a) + i));
```

```
172             }
173         }
174         if(tipo == ALUMNO){
175             char nombre[50];
176             for(size_t i = 0; i < tam; i++){
177                 printf("Nombre: %s  Clave: %d", ((alumno_t *)(*a) + i)->nombre,
    ((alumno_t *)(*a) + i)->clave);
178                 printf("\n");
179             }
180         }
181         printf("\n");
182     }
183
184     void liberar_arreglo(void **a, size_t tam, tipo_t tipo){
185         if(tipo != ALUMNO){
186             free(*a);
187         }
188         else{
189             for(size_t i = 0; i < tam; i++){
190                 free(((alumno_t *)(*a) + i)->nombre);
191             }
192             free(*a);
193         }
194     }
```

Tarea #3: Arreglos Redimensionables

```
Grade School

If you need help running the tests or submitting your code, check out HELP.md.

Instructions

Given students' names along with the grade that they are in, create a roster
for the school.

In the end, you should be able to:

- Add a student's name to the roster for a grade
  - "Add Jim to grade 2."
  - "OK."
- Get a list of all students enrolled in a grade
  - "Which students are in grade 2?"
  - "We've only got Jim just now."
- Get a sorted list of all students in all grades.  Grades should sort
  as 1, 2, 3, etc., and students within a grade should be sorted
  alphabetically by name.
  - "Who all is enrolled in school right now?"
  - "Let me think. We have
  Anna, Barb, and Charlie in grade 1,
  Alex, Peter, and Zoe in grade 2
  and Jim in grade 5.
  So the answer is: Anna, Barb, Charlie, Alex, Peter, Zoe and Jim"

Note that all our students only have one name  (It's a small town, what
do you want?) and each student cannot be added more than once to a grade or the
roster.
In fact, when a test attempts to add the same student more than once, your
implementation should indicate that this is incorrect.
```

grade_school.h

```c
#ifndef GRADE_SCHOOL_H
#define GRADE_SCHOOL_H

#include <stddef.h>
#include <stdint.h>
#include <stdbool.h>
```

```c
typedef struct {
   uint8_t grade;
   char *name;
} student_t;

typedef struct {
   size_t count;
   size_t max_students;
   student_t *students;
} roster_t;

void init_roster(roster_t *roster);
void free_roster(roster_t *roster);
bool add_student(roster_t *roster, const char *name, uint8_t grade);
roster_t get_grade(const roster_t *roster, uint8_t desired_grade);

#endif
```

test_grade_school.c

```c
#include "test-framework/unity.h"
#include "grade_school.h"

void setUp(void){
}

void tearDown(void){
}

static void check_roster_names(roster_t expected, roster_t actual){
   for (size_t i = 0; i < expected.count; ++i)
      TEST_ASSERT_EQUAL_STRING(expected.students[i].name,
                               actual.students[i].name);
}

static void check_rosters(roster_t expected, roster_t actual){
   TEST_ASSERT_EQUAL_size_t_MESSAGE(expected.count, actual.count,
                                    "Incorrect number of students");
   TEST_ASSERT_EQUAL_size_t_MESSAGE(expected.max_students, actual.max_students,
                                    "Incorrect maximum space for students");
   check_roster_names(expected, actual);
}

static void test_roster_is_empty_when_no_student_added(void){
   TEST_IGNORE();   // delete this line to run test
   roster_t actual;
   init_roster(&actual);
   TEST_ASSERT_EQUAL(0, actual.count);
   TEST_ASSERT_EQUAL(0, actual.max_students);
   TEST_ASSERT_NULL(actual.students);
   free_roster(&actual);
}
```

```c
static void test_add_student(void){
   TEST_IGNORE();   // delete this line to run test
   roster_t actual;
   init_roster(&actual);
   TEST_ASSERT_TRUE(add_student(&actual, "Aimee", 2));
   free_roster(&actual);
}

static void test_student_added_to_roster(void){
   TEST_IGNORE();
   student_t expected_students[] = { { 2, "Aimee" } };
   roster_t expected = { 1, 1, expected_students};
   roster_t actual;
   init_roster(&actual);
   add_student(&actual, "Aimee", 2);
   check_rosters(expected, actual);
   free_roster(&actual);
}

static void test_adding_multiple_students_in_same_grade_in_roster(void){
   TEST_IGNORE();
   roster_t actual;
   init_roster(&actual);
   TEST_ASSERT_TRUE(add_student(&actual, "Blair", 2));
   TEST_ASSERT_TRUE(add_student(&actual, "James", 2));
   TEST_ASSERT_TRUE(add_student(&actual, "Paul", 2));
   free_roster(&actual);
}

static void test_multiple_students_in_same_grade_are_added_to_roster(void){
   TEST_IGNORE();
   student_t expected_students[] = {
         { 2, "Blair" },
         { 2, "James" },
         { 2, "Paul" }
   };
   roster_t expected = { 3, 4, expected_students };
   roster_t actual;
   init_roster(&actual);
   add_student(&actual, "Blair", 2);
   add_student(&actual, "James", 2);
   add_student(&actual, "Paul", 2);
   check_rosters(expected, actual);
   free_roster(&actual);
}

static void test_cannot_add_student_to_same_grade_more_than_once(void){
   TEST_IGNORE();
   roster_t actual;
   init_roster(&actual);
   TEST_ASSERT_TRUE(add_student(&actual, "Blair", 2));
   TEST_ASSERT_TRUE(add_student(&actual, "James", 2));
   TEST_ASSERT_FALSE(add_student(&actual, "James", 2));
   TEST_ASSERT_TRUE(add_student(&actual, "Paul", 2));
   free_roster(&actual);
}
```

```c
static void test_student_not_added_to_same_grade_in_roster_more_than_once(void){
   TEST_IGNORE();
   student_t expected_students[] = {
      { 2, "Blair" },
      { 2, "James" },
      { 2, "Paul" }
   };
   roster_t expected = { 3, 4, expected_students};
   roster_t actual;
   init_roster(&actual);
   add_student(&actual, "Blair", 2);
   add_student(&actual, "James", 2);
   add_student(&actual, "James", 2);
   add_student(&actual, "Paul", 2);
   check_rosters(expected, actual);
   free_roster(&actual);
}

static void test_adding_students_in_multiple_grades(void){
   TEST_IGNORE();
   roster_t actual;
   init_roster(&actual);
   TEST_ASSERT_TRUE(add_student(&actual, "Chelsea", 3));
   TEST_ASSERT_TRUE(add_student(&actual, "Logan", 7));
   free_roster(&actual);
}

static void test_students_in_multiple_grades_are_added_to_roster(void){
   TEST_IGNORE();
   student_t expected_students[] = {
      { 3, "Chelsea" },
      { 7, "Logan" }
   };
   roster_t expected = {2, 2, expected_students};
   roster_t actual;
   init_roster(&actual);
   add_student(&actual, "Chelsea", 3);
   add_student(&actual, "Logan", 7);
   check_rosters(expected, actual);
   free_roster(&actual);
}

static void test_cannot_add_same_student_to_multiple_grades_in_roster(void){
   TEST_IGNORE();
   roster_t actual;
   init_roster(&actual);
   TEST_ASSERT_TRUE(add_student(&actual, "Blair", 2));
   TEST_ASSERT_TRUE(add_student(&actual, "James", 2));
   TEST_ASSERT_FALSE(add_student(&actual, "James", 3));
   TEST_ASSERT_TRUE(add_student(&actual, "Paul", 3));
   free_roster(&actual);
}
```

```c
static void test_student_not_added_to_multiple_grades_in_roster(void){
   TEST_IGNORE();
   student_t expected_students[] = {
      { 2, "Blair" },
      { 2, "James" },
      { 3, "Paul" }
   };
   roster_t expected = { 3, 4, expected_students};
   roster_t actual;
   init_roster(&actual);
   add_student(&actual, "Blair", 2);
   add_student(&actual, "James", 2);
   add_student(&actual, "James", 3);
   add_student(&actual, "Paul", 3);
   check_rosters(expected, actual);
   free_roster(&actual);
}

static void test_students_are_sorted_by_grades_in_roster(void){
   TEST_IGNORE();
   student_t expected_students[] = {
      { 1, "Anna" },
      { 2, "Peter" },
      { 3, "Jim" }
   };
   roster_t expected = { 3, 4, expected_students};
   roster_t actual;
   init_roster(&actual);
   add_student(&actual, "Jim", 3);
   add_student(&actual, "Peter", 2);
   add_student(&actual, "Anna", 1);
   check_rosters(expected, actual);
   free_roster(&actual);
}

static void test_students_are_sorted_by_name_in_roster(void){
   TEST_IGNORE();
   student_t expected_students[] = {
      { 2, "Alex" },
      { 2, "Peter" },
      { 2, "Zoe" }
   };
   roster_t expected = { 3, 4, expected_students};
   roster_t actual;
   init_roster(&actual);
   add_student(&actual, "Peter", 2);
   add_student(&actual, "Zoe", 2);
   add_student(&actual, "Alex", 2);
   check_rosters(expected, actual);
   free_roster(&actual);
}
```

```c
static void test_students_are_sorted_by_grades_and_then_by_names_in_roster(void){
    TEST_IGNORE();
    student_t expected_students[] = {
        { 1, "Anna" },
        { 1, "Barb" },
        { 1, "Charlie" },
        { 2, "Alex" },
        { 2, "Peter" },
        { 2, "Zoe" },
        { 3, "Jim" }
    };
    roster_t expected = { 7, 8, expected_students};
    roster_t actual;
    init_roster(&actual);
    add_student(&actual, "Peter", 2);
    add_student(&actual, "Anna", 1);
    add_student(&actual, "Barb", 1);
    add_student(&actual, "Zoe", 2);
    add_student(&actual, "Alex", 2);
    add_student(&actual, "Jim", 3);
    add_student(&actual, "Charlie", 1);
    check_rosters(expected, actual);
    free_roster(&actual);
}

static void test_grade_empty_if_no_students_in_roster(void){
    TEST_IGNORE();
    uint8_t desired_grade = 1;
    roster_t roster;
    init_roster(&roster);
    roster_t actual = get_grade(&roster, desired_grade);
    TEST_ASSERT_EQUAL(0, actual.count);
    free_roster(&roster);
    free_roster(&actual);
}

static void test_grade_empty_if_no_students_in_grade(void){
    TEST_IGNORE();
    uint8_t desired_grade = 1;
    roster_t roster;
    init_roster(&roster);
    add_student(&roster, "Peter", 2);
    add_student(&roster, "Zoe", 2);
    add_student(&roster, "Alex", 2);
    add_student(&roster, "Jim", 3);
    roster_t actual = get_grade(&roster, desired_grade);
    TEST_ASSERT_EQUAL(0, actual.count);
    free_roster(&roster);
    free_roster(&actual);
}
```

```c
static void test_student_not_added_to_same_grade_more_than_once(void){
   TEST_IGNORE();
   uint8_t desired_grade = 2;
   student_t expected_students[] = {
      { 2, "Blair" },
      { 2, "James" },
      { 2, "Paul" }
   };
   roster_t expected = { 3, 4, expected_students};
   roster_t roster;
   init_roster(&roster);
   add_student(&roster, "Blair", 2);
   add_student(&roster, "James", 2);
   add_student(&roster, "James", 2);
   add_student(&roster, "Paul", 2);
   roster_t actual = get_grade(&roster, desired_grade);
   check_rosters(expected, actual);
   free_roster(&roster);
   free_roster(&actual);
}

static void test_student_not_added_to_multiple_grades(void){
   TEST_IGNORE();
   uint8_t desired_grade = 2;
   student_t expected_students[] = {
      { 2, "Blair" },
      { 2, "James" }
   };
   roster_t expected = {2, 2, expected_students};
   roster_t roster;
   init_roster(&roster);
   add_student(&roster, "Blair", 2);
   add_student(&roster, "James", 2);
   add_student(&roster, "James", 3);
   add_student(&roster, "Paul", 3);
   roster_t actual = get_grade(&roster, desired_grade);
   check_rosters(expected, actual);
   free_roster(&roster);
   free_roster(&actual);
}

static void test_student_not_added_to_other_grade_for_multiple_grades(void){
   TEST_IGNORE();
   uint8_t desired_grade = 3;
   student_t expected_students[] = {
      { 3, "Paul" }
   };
   roster_t expected = { 1, 1, expected_students};
   roster_t roster;
   init_roster(&roster);
   add_student(&roster, "Blair", 2);
   add_student(&roster, "James", 2);
   add_student(&roster, "James", 3);
   add_student(&roster, "Paul", 3);
   roster_t actual = get_grade(&roster, desired_grade);
   check_rosters(expected, actual);
   free_roster(&roster);
   free_roster(&actual);
}
```

```c
static void test_students_are_sorted_by_name_in_grade(void){
   TEST_IGNORE();
   uint8_t desired_grade = 5;
   student_t expected_students[] = {
      { 5, "Bradley" },
      { 5, "Franklin" }
   };
   roster_t expected = {2, 2, expected_students};
   roster_t roster;
   init_roster(&roster);
   add_student(&roster, "Franklin", 5);
   add_student(&roster, "Bradley", 5);
   add_student(&roster, "Jeff", 1);
   roster_t actual = get_grade(&roster, desired_grade);
   check_rosters(expected, actual);
   free_roster(&roster);
   free_roster(&actual);
}

int main(void){
   UnityBegin("test_grade_school.c");
   RUN_TEST(test_roster_is_empty_when_no_student_added);
   RUN_TEST(test_add_student);
   RUN_TEST(test_student_added_to_roster);
   RUN_TEST(test_adding_multiple_students_in_same_grade_in_roster);
   RUN_TEST(test_multiple_students_in_same_grade_are_added_to_roster);
   RUN_TEST(test_cannot_add_student_to_same_grade_more_than_once);
   RUN_TEST(test_student_not_added_to_same_grade_in_roster_more_than_once);
   RUN_TEST(test_adding_students_in_multiple_grades);
   RUN_TEST(test_students_in_multiple_grades_are_added_to_roster);
   RUN_TEST(test_cannot_add_same_student_to_multiple_grades_in_roster);
   RUN_TEST(test_student_not_added_to_multiple_grades_in_roster);
   RUN_TEST(test_students_are_sorted_by_grades_in_roster);
   RUN_TEST(test_students_are_sorted_by_name_in_roster);
   RUN_TEST(test_students_are_sorted_by_grades_and_then_by_names_in_roster);
   RUN_TEST(test_grade_empty_if_no_students_in_roster);
   RUN_TEST(test_grade_empty_if_no_students_in_grade);
   RUN_TEST(test_student_not_added_to_same_grade_more_than_once);
   RUN_TEST(test_student_not_added_to_multiple_grades);
   RUN_TEST(test_student_not_added_to_other_grade_for_multiple_grades);
   RUN_TEST(test_students_are_sorted_by_name_in_grade);
   return UnityEnd();
}
```

grade_school.c

```c
#include "grade_school.h"
#include <stdlib.h>
#include <assert.h>
#include <string.h>
void sort_by_grade(roster_t *);
void sort_by_name(roster_t *);
void init_roster(roster_t *roster){
    roster->count = 0;
    roster->max_students = 0;
    roster->students = NULL;
}

void free_roster(roster_t *roster){
    for(size_t i = 0; i < roster->count; i++){
        free(roster->students[i].name);
    }
    free(roster->students);
}

bool add_student(roster_t *roster, const char *name, uint8_t grade){
    if(roster->count == roster->max_students){
        if(roster->max_students == 0){
            roster->max_students = 1;
            student_t *r = (student_t *)realloc(roster->students, roster->max_students * sizeof(student_t));
            assert(r != NULL);
            roster->students = r;
            roster->students[roster->count].grade = grade;
            roster->students[roster->count].name = malloc(strlen(name) + 1);
            assert(roster->students[roster->count].name != NULL);
            strcpy(roster->students[roster->count].name, name);
            roster->count++;
            return true;
        }
        else{
            for(size_t i = 0; i < roster->count; i++){
                if(strcmp(roster->students[i].name, name) == 0){
                    return false;
                }
            }
            roster->max_students = roster->max_students * 2;
            student_t *r = (student_t *)realloc(roster->students, roster->max_students * sizeof(student_t));
```

```
42              assert(r != NULL);
43              roster->students = r;
44              roster->students[roster->count].grade = grade;
45              roster->students[roster->count].name = malloc(strlen(name) + 1);
46              assert(roster->students[roster->count].name != NULL);
47              strcpy(roster->students[roster->count].name, name);
48              roster->count++;
49              sort_by_grade(roster);
50              sort_by_name(roster);
51              return true;
52          }
53      }
54      else{
55          for(size_t i = 0; i < roster->count; i++){
56              if(strcmp(roster->students[i].name, name) == 0){
57                  return false;
58              }
59          }
60          roster->students[roster->count].grade = grade;
61          roster->students[roster->count].name = malloc(strlen(name) + 1);
62          assert(roster->students[roster->count].name != NULL);
63          strcpy(roster->students[roster->count].name, name);
64          roster->count++;
65          sort_by_grade(roster);
66          sort_by_name(roster);
67          return true;
68      }
69  }
70
71  void sort_by_grade(roster_t *roster){
72      student_t tmp;
73      for(size_t i = 0; i < roster->count; i++){
74          for(size_t j = i + 1; j < roster->count; j++){
75              if(roster->students[i].grade > roster->students[j].grade){
76                  tmp = roster->students[i];
77                  roster->students[i] = roster->students[j];
78                  roster->students[j] = tmp;
79                  i = 0;
80              }
81          }
82      }
83  }
84
85  void sort_by_name(roster_t *roster){
86      student_t tmp;
87      for(size_t i = 0; i < roster->count; i++){
```

```
88          for(size_t j = i + 1; j < roster->count; j++){
89              if(roster->students[i].grade == roster->students[j].grade){
90                  if(strcmp(roster->students[i].name, roster-
    >students[j].name) > 0){
91                      tmp = roster->students[j];
92                      roster->students[j] = roster->students[i];
93                      roster->students[i] = tmp;
94                  }
95              }
96          }
97      }
98  }
99
100 roster_t get_grade(const roster_t *roster, uint8_t desired_grade){
101     roster_t answer;
102     init_roster(&answer);
103     for(size_t i = 0; i < roster->count; i++){
104         if(roster->students[i].grade == desired_grade){
105             add_student(&answer, roster->students[i].name, desired_grade);
106         }
107     }
108     return answer;
109 }
```

1. Explica tu algoritmo para agregar un estudiante a una lista. ¿Cómo crece tu lista dinámicamente? ¿Cómo determinas de que no puede haber dos estudiantes en la lista que tengan el mismo nombre? ¿Cómo determinas que un mismo estudiante no puede estar en dos grados distintos?

```
20  bool add_student(roster_t *roster, const char *name, uint8_t grade){
21      if(roster->count == roster->max_students){
22          if(roster->max_students == 0){
23              roster->max_students = 1;
24              student_t *r = (student_t *)realloc(roster->students, roster->max_students * sizeof(student_t));
25              assert(r != NULL);
26              roster->students = r;
27              roster->students[roster->count].grade = grade;
28              roster->students[roster->count].name = malloc(strlen(name) + 1);
29              assert(roster->students[roster->count].name != NULL);
30              strcpy(roster->students[roster->count].name, name);
31              roster->count++;
32              return true;
33          }
34          else{
35              for(size_t i = 0; i < roster->count; i++){
36                  if(strcmp(roster->students[i].name, name) == 0){
37                      return false;
38                  }
39              }
40              roster->max_students = roster->max_students * 2;
41              student_t *r = (student_t *)realloc(roster->students, roster->max_students * sizeof(student_t));
42              assert(r != NULL);
43              roster->students = r;
44              roster->students[roster->count].grade = grade;
45              roster->students[roster->count].name = malloc(strlen(name) + 1);
46              assert(roster->students[roster->count].name != NULL);
47              strcpy(roster->students[roster->count].name, name);
48              roster->count++;
49              sort_by_grade(roster);
50              sort_by_name(roster);
51              return true;
52          }
53      }
54      else{
55          for(size_t i = 0; i < roster->count; i++){
56              if(strcmp(roster->students[i].name, name) == 0){
57                  return false;
58              }
59          }
60          roster->students[roster->count].grade = grade;
61          roster->students[roster->count].name = malloc(strlen(name) + 1);
62          assert(roster->students[roster->count].name != NULL);
63          strcpy(roster->students[roster->count].name, name);
64          roster->count++;
65          sort_by_grade(roster);
66          sort_by_name(roster);
67          return true;
68      }
69  }
```

Primero reviso si el último elemento de la lista ya alcanzo al tamaño máximo, de ser así, hay dos opciones, la primera, que este vacía, en ese caso el tamaño máximo cambia de cero a uno, en caso contario, el tamaño maño máximo aumenta al doble, y se utiliza `realloc()` para cambiar el tamaño. En caso de que no, simplemente agrego el elemento. Para revisar que no haya dos estudiantes repetidos, solo necesito hacer una búsqueda de todos los alumnos, así no importa si son del mismo grupo o no.

2. Explica tu algoritmo para obtener la lista de estudiantes de un grado específico. ¿La lista que se obtiene como resultado de este algoritmo es dinámica? ¿Cómo determinas su tamaño?

```
100  roster_t get_grade(const roster_t *roster, uint8_t desired_grade){
101      roster_t answer;
102      init_roster(&answer);
103      for(size_t i = 0; i < roster->count; i++){
104          if(roster->students[i].grade == desired_grade){
105              add_student(&answer, roster->students[i].name, desired_grade);
106          }
107      }
108      return answer;
109  }
```

Lo que hago es crear otra lista llamada `answer`, la inicializo con `init_roster()`, y luego hago un ciclo `for` que recorre todos los elementos de la lista, cada vez que encuentra uno con el mismo grado, lo agrega a la lista `answer` con la función `add_student()`, al utilizar esa función, la lista crece dinámicamente, por lo que no necesito determinar su tamaño.

3. ¿Qué algoritmo de ordenamiento utilizaste para esta tarea? ¿Cómo aplicaste el criterio de ordenar primero por grado y luego por nombre?

```
71  void sort_by_grade(roster_t *roster){
72      student_t tmp;
73      for(size_t i = 0; i < roster->count; i++){
74          for(size_t j = i + 1; j < roster->count; j++){
75              if(roster->students[i].grade > roster->students[j].grade){
76                  tmp = roster->students[i];
77                  roster->students[i] = roster->students[j];
78                  roster->students[j] = tmp;
79                  i = 0;
80              }
81          }
82      }
83  }
```

Utilicé el método de ordenamiento `burbuja()` en el que va de forma iterativa uno por uno ordenándolos por grado, utilicé este método porque los `tests` no contenían muchos alumnos, por lo cual no hay mucha diferencia entre utilizar este o algún algoritmo más rápido como `quicksort()`, el método burbuja aprendido en clase suele utilizar otra función llamada `swap()`, pero decidí implementarlo ahí mismo.
Para el ordenamiento por nombre primero reviso si los elementos que trato de ordenar (el actual y el siguiente) son del mismo grado, ya que de lo contrario no tiene sentido, después reviso con la función `strcmp()` si es mayor a `0`, en cuyo caso los cambio de lugar.

```c
void sort_by_name(roster_t *roster){
    student_t tmp;
    for(size_t i = 0; i < roster->count; i++){
        for(size_t j = i + 1; j < roster->count; j++){
            if(roster->students[i].grade == roster->students[j].grade){
                if(strcmp(roster->students[i].name, roster->students[j].name) > 0){
                    tmp = roster->students[j];
                    roster->students[j] = roster->students[i];
                    roster->students[i] = tmp;
                }
            }
        }
    }
}
```

4. Imagina por un momento que se te pide agregar una nueva funcionalidad para dar de baja estudiantes. Explica detalladamente los cambios que consideres son necesarios para poder eliminar estudiantes de la lista dinámica.

```c
void remove_student(roster_t *roster, const char *name){
    for(size_t i = 0; i < roster->count; i++){
        if(strcmp(roster->students[i].name, name) == 0){
            for(size_t j = i; j < roster->count - 1; j++){
                student_t tmp = roster->students[j];
                roster->students[j] = roster->students[j + 1];
                free(tmp.name);
            }
            roster->count--;
            return;
        }
    }
}
```

Para eliminar un alumno, podemos hacer una función `remove_student()` a la que le pasemos como parámetros la lista y el nombre del alumno que queremos eliminar, el nombre puede ser `const`, ya que, no solo es de lectura.
Una vez entro de la función inicio una ciclo `for` para recorrer todos los elementos (ya que no hay alumnos repetidos, será el primero que encontremos), con la función `strcmp()` revisamos si ya lo encontramos, unas vez que lo encontramos, primero lo guardamos en una estructura temporal, luego hacemos que se recorran todos de lugar, y hacemos el `free()` del nombre del temporal para no perderlo. También es importante agregar el prototipo de la función al archivo .h porque si la agregamos al archivo .c nos aparecerá un `warning` por ser llamado desde ahí.

5. ¿Cuál es el nombre de la prueba (test) que te resultó más difícil pasar? ¿Por qué consideras que fue difícil? Al finalizar la tarea, ¿cuántas pruebas tuviste que ignorar? ¿Cuáles fueron? ¿Qué errores obtuviste?

Ninguna, todas fueron sencillas de pasar, todas las pruebas fueron pasadas.

6. Inventa una nueva prueba y agrégala al conjunto de pruebas del proyecto. Agrega el código fuente y explicación de tu prueba en el reporte.

```c
static void test_delete_student(void){
    //TEST_IGNORE();
    uint8_t desired_grade = 5;
    student_t expected_students[] = {
        { 5, "Andres" },
        { 5, "Franklin" },
        { 5, "Hugo" }
    };
    roster_t expected = {3, 4, expected_students};
    roster_t roster;
    init_roster(&roster);
    add_student(&roster, "Franklin", 5);
    add_student(&roster, "Bradley", 5);
    add_student(&roster, "Jeff", 1);
    remove_student(&roster, "Bradley");
    add_student(&roster, "Hugo", 5);
    add_student(&roster, "Andres", 5);
    roster_t actual = get_grade(&roster, desired_grade);
    check_rosters(expected, actual);
    free_roster(&roster);
    free_roster(&actual);
}
```

En este test decidí incorporar mi función para eliminar, agrego varios estudiantes, elimino a uno, y después sigo agregando. El `test` paso la prueba utilizando `make memcheck`.

7. ¿Aplicaste la refactorización en tu código? Explica de qué manera lo hiciste

Sí, me di cuenta que mi función para agregar alumnos era muy larga y repetitiva, así que la simplifique.

```c
22  bool add_student(roster_t *roster, const char *name, uint8_t grade){
23      for(size_t i = 0; i < roster->count; i++){
24          if(strcmp(roster->students[i].name, name) == 0){
25              return false;
26          }
27      }
28      if(roster->count == roster->max_students){
29          if(roster->max_students == 0){
30              roster->max_students = 1;
31          }
32          else{
33              roster->max_students = roster->max_students * 2;
34          }
```

```
35          student_t *r = NULL;
36          r = (student_t *)realloc(roster->students, roster->max_students *
37  sizeof(student_t));
38          assert(r != NULL);
39          roster->students = r;
40      }
41      roster->students[roster->count].grade = grade;
42      roster->students[roster->count].name = malloc(strlen(name) + 1);
43      assert(roster->students[roster->count].name != NULL);
44      strcpy(roster->students[roster->count].name, name);
45      roster->count++;
46      sort_by_grade(roster);
47      sort_by_name(roster);
48      return true;
49  }
```

8. ¿Qué hiciste particularmente bien en esta tarea?

Aplicar los conocimientos adquiridos en clase de arreglos dinámicos utilizando `realloc()`, así como los conocimientos adquiridos el semestre pasado de ordenamiento.

9. ¿Qué pudiste haber hecho mejor en esta tarea?

Tal vez hay una manera más fácil de simplificar la función para agregar estudiantes.

10. ¿Qué nuevos conocimientos y experiencias adquiriste con esta tarea?

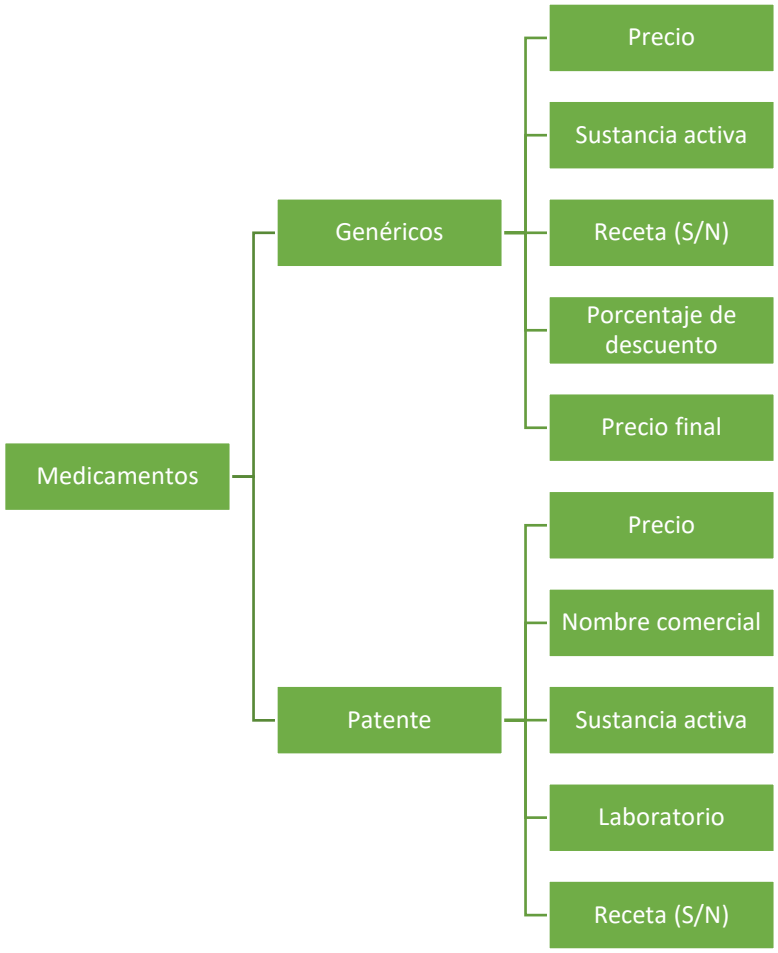
El utilizar los métodos de ordenamiento en punteros, aunque siento que más que aprender, esta tarea fue para poder terminar de entender el tema de arreglos dinámicos de manera practica.

Trabajo en clase: “Farmacia con medicamentos”

Martes 13 Septiembre 2022

En una farmacia se venden 2 tipos de medicamentos: genéricos y de patente, Los datos de los tipos genéricos son: el precio, sustancia activa, si requiere receta médica ("s" o "n"), porcentaje de descuento (si no hay se asigna un cero) y precio final. Los datos para los medicamentos de patente son: precio, nombre comercial (por ejemplo "aspirina"), sustancia activa (por ejemplo "ácido acetilsalicilico"), laboratorio (por ejemplo "Bayer") y si requiere receta médica ("s" o "n").

a. Diseñe las estructuras de datos necesarias para poder almacenar la información de varios medicamentos en un arreglo dinámico de estructuras, en el cual se puedan almacenar datos tanto de medicamentos genéricos como de medicamentos de patente.




```c
typedef enum{
    GENERICO,
    PATENTE
}tipo_medicamento_t;

typedef struct{
    float precio;
    char *sustancia_activa;
    char requiere_receta;
    void *info_adicional;
    tipo_medicamento_t tipo;
}medicamento_t;

typedef struct{
    float porcentaje_descuento;
    float precio_final;
}generico_t;

typedef struct{
    char *nombre_comercial;
    char *laboratorio;
}patente_t;
```

b. Escriba una función para reservar memoria para el arreglo de medicamentos. La función recibe dos parámetros: el tamaño del arreglo y un apuntador por referencia donde se asignará la memoria dinámica para el arreglo.

```c
medicamento_t *crea_arreglo(size_t tam){
    medicamento_t *arreglo = NULL;
    arreglo = (medicamento_t *)malloc(tam * sizeof(medicamento_t));
    assert(arreglo != NULL);
}
```

c. Escribir una función para capturar la información medicamentos genéricos y/o de patente. La función recibe dos parámetros: un apuntador al arreglo de medicamentos y un entero con el tamaño del arreglo. Utilice notación/aritmética de apuntadores para acceder a los elementos del arreglo.

```
1  void captura_medicamentos(medicamento_t *arreglo, size_t tam){
2      char s[40];
3      for(size_t i = 0; i < tam; i++){
4          scanf("%f", &arreglo[i].precio);
5          scanf("%u", &arreglo[i].tipo);
6          scanf("\n%c", &arreglo[i].requiere_receta);
7          scanf("\n%[^\n]", s);
8          arreglo[i].sustancia_activa = malloc(strlen(s) + 1);
9          assert(arreglo[i].sustancia_activa != NULL);
10         strcpy(arreglo[i].sustancia_activa, s);
11         if(arreglo[i].tipo == GENERICO){
12             arreglo[i].info_adicional = malloc(sizeof(generico_t));
13             scanf("%f", &((generico_t *)(arreglo[i].info_adicional))-
   >porcentaje_descuento);
14             float descuento = (arreglo[i].precio) * ((generico_t
   *)(arreglo[i].info_adicional))->porcentaje_descuento;
15             ((generico_t *)(arreglo[i].info_adicional))->precio_final =
   arreglo[i].precio - descuento;
16         }
17         else{
18             char nc[50];
19             char lab[50];
20             arreglo[i].info_adicional = malloc(sizeof(patente_t));
21             scanf("\n%[^\n]", nc);
22             scanf("\n%[^\n]", lab);
23             ((patente_t *)(arreglo[i].info_adicional))->laboratorio =
   malloc(strlen(lab) + 1);
24             assert(((patente_t *)(arreglo[i].info_adicional))->laboratorio
   != NULL);
25             strcpy(((patente_t *)(arreglo[i].info_adicional))->laboratorio,
   lab);
26             ((patente_t *)(arreglo[i].info_adicional))->nombre_comercial =
   malloc(strlen(nc) + 1);
27             assert(((patente_t *)(arreglo[i].info_adicional))-
   >nombre_comercial != NULL);
28             strcpy(((patente_t *)(arreglo[i].info_adicional))-
   >nombre_comercial, nc);
29         }
30     }
31 }
```

Primer Examen Parcial — Jueves 15 Septiembre 2022

Problema 1 (25 puntos)

Escribir una función para obtener los elementos de la diagonal principal de una matriz dinámica. La función recibe como parámetros el apuntador a una matriz dinámica de NxN números enteros, previamente creada y rellenada con datos, y regresa un arreglo dinámico con el contenido de la diagonal principal. **Utilice aritmética de apuntadores.**

Por ejemplo, si la matriz dinámica contiene la siguiente información:

1	2	3
4	5	6
7	8	9

Entonces regresa un arreglo dinámico que contiene:

1	5	9

Solución

Asumiendo que también nos dan N como un parámetro, por ejemplo, en el ejemplo sería N = 3

```c
int *diagonal_Principal(int **matriz, int N){
    int *diagonal = NULL;
    crea_arreglo(&diagonal, N);
    int j = 0;
    for(int i = 0; i < N; i++){
        *(diagonal + j) = *(*(matriz + i) + j);
        j = i + 1;
    }
    return diagonal;
}

void crea_arreglo(int **arreglo, int tam){
    *arreglo = (int *)malloc(tam * sizeof(int));
    assert(*arreglo != NULL);
}
```

Problema 2

Se requiere implementar un prototipo de una pequeña parte de una aplicación de red social, la cual mostrará publicaciones que aparecerán en pantalla cuando el usuario abra la página principal. Existen dos tipos de publicaciones en la red social: publicaciones de texto (mensajes) y publicaciones fotográficas. Las publicaciones de texto contienen el nombre del autor y el texto del mensaje. Las publicaciones fotográficas contienen el nombre del autor, el número de personas a las que les ha gustado la publicación y el tamaño del archivo de la imagen que será visualizada.

a. **(10 puntos)** Diseñe y escriba el código de las estructuras requeridas para almacenar las publicaciones de la red social en una estructura de datos dinámica.

```c
#define MAX_TEXTO 30

typedef enum{
    MENSAJE,
    FOTO
}tipo_post_t;

typedef struct{
    char username[MAX_TEXTO];
    tipo_post_t tipo;
    void *info;
}post_t;

typedef struct{
    char mensaje[MAX_TEXTO];
}mensaje_t;

typedef struct{
    int tam_archivo;
    int likes;
}foto_t;
```

b. **(10 puntos)** Completar la siguiente función para inicializar la estructura dinámica de publicaciones a un tamaño determinado. **Conteste únicamente sobre la hoja del examen**.

```c
void init_posts(post_t **arreglo, int tam){
    *arreglo = (post_t *)malloc(tam * sizeof(post_t));
    assert(*arreglo != NULL);
}
```

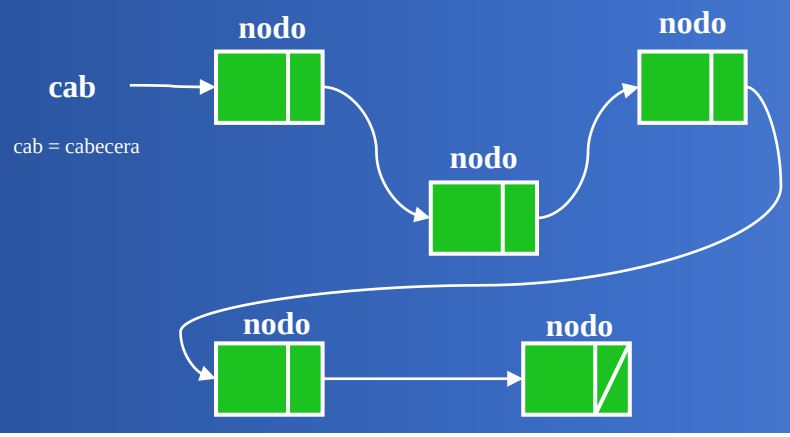
c. **(40 puntos)** Completar la siguiente función leer la información dentro de la estructura dinámica de publicaciones. **Conteste únicamente sobre la hoja del examen y use aritmética de apuntadores**.

```c
void read_posts(post_t *posts, int tam){
    for(post_t *p = posts; p < posts + tam; p++){
        printf("Username: ");
        scanf("%s", p->username);
        printf("Post type (0 = text, 1 = photo): ");
        scanf("%u", &p->tipo);
        if(p->tipo == MENSAJE){
            p->info = malloc(sizeof(mensaje_t));
            printf("Message text: ");
            scanf("\n%[^\n]", ((mensaje_t *)p->info)->mensaje);
        }   else if(p->tipo == FOTO){
                p->info = malloc(sizeof(foto_t));
                printf("File size: ");
                scanf("%d", &((foto_t *)p->info)->tam_archivo);
                printf("Likes count: ");
                scanf("%d", &((foto_t *)p->info)->likes);
        }
    }
}
```

d. **(15 puntos)** Completar la siguiente función para obtener la cantidad de “me gusta” de la publicación más popular. Si ninguna publicación tiene “me gusta” regresa un valor especial -1. **Conteste únicamente sobre la hoja del examen y use aritmética de apuntadores**.

```
int most_popular_count(post_t *posts, int n){
    int result = -1;
    for(post_t *p = posts; p < posts + n; p++){
        if(p->tipo == FOTO){
            int likes = ((foto_t *)p->info)->likes;
            if(likes > 0 && likes > result){
                result = likes;}
        }
    }
    return result;
}
```

2. LISTAS ENLAZADAS

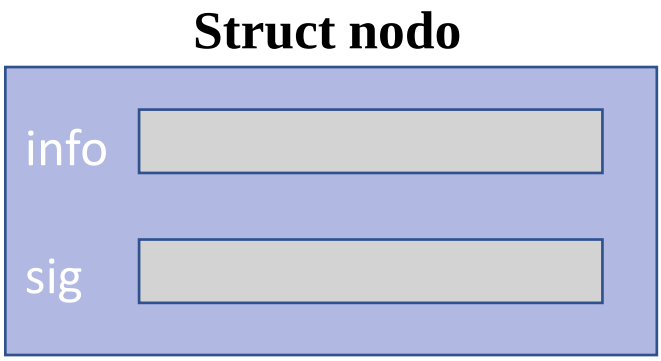


2.1 Listas simples

Objetivo: Ser capaz de diseñar diversos tipos de listas enlazadas y programar las principales operaciones para su manipulación.

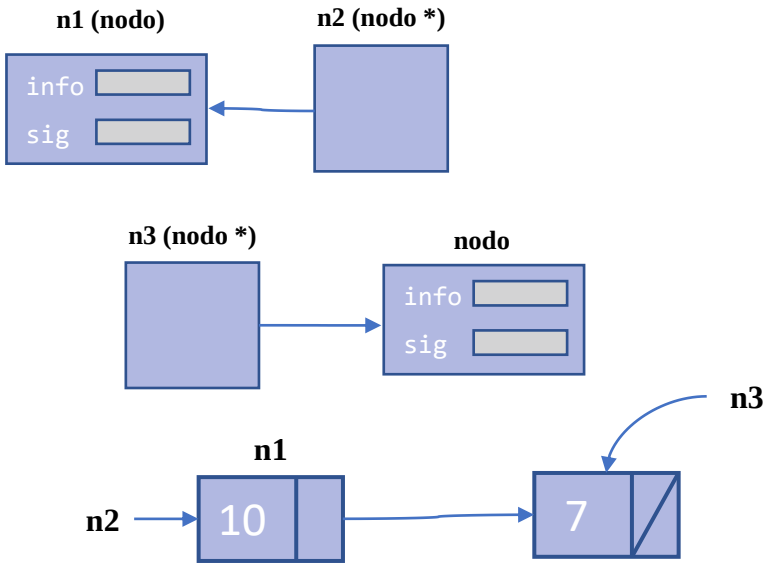
En una lista podemos guardar cualquier cosa, para este ejemplo de lista simple, haremos una lista de números enteros, cada casilla es un nodo, y para cada nodo hacemos una estructura con los datos que almacenara cada nodo.

2.1.1 Definir la estructura del nodo




```c
struct nodo{
    int info;
    struct nodo *sig;
};
typedef struct nodo nodo_t;
```

```c
int main(){
    nodo_t n1;
    nodo_t *n2, *n3;
    n1.info = 5;
    n1.sig = NULL;
    n2 = &n1;
    n2->info = 10;
    printf("%d", n1.info);
    n3 = (nodo_t *)malloc(sizeof(nodo_t));
    assert(n3 != NULL);
    n3->info = 7;
    n3->sig = NULL;
    n2->sig = n3;
    return 0;
}
```



Output

10

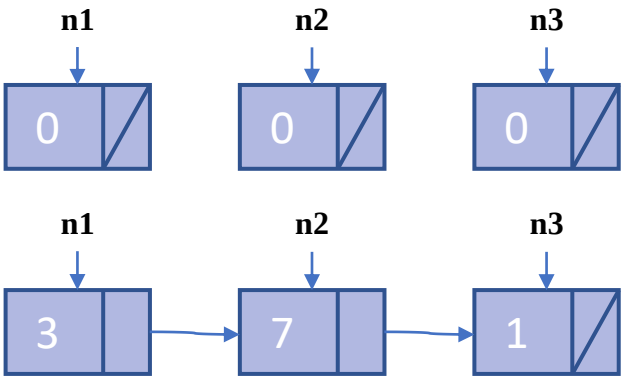
2.1.2 Funciones de listas simples

Función para crear un nodo

```c
nodo_t *crea_nodo(){
    nodo_t *nodo = NULL;
    nodo = (nodo_t*)malloc(sizeof(nodo_t));
    assert(nodo != NULL);
    nodo->info = 0;
    nodo->sig = NULL;
    return nodo;
}
```

Ejemplo Burdo

```c
int main(){
    nodo_t *n1 = crea_nodo();
    nodo_t *n2 = crea_nodo();
    nodo_t *n3 = crea_nodo();
    n1->info = 3;
    n2->info = 7;
    n3->info = 1;
    n1->sig = n2;
    n2->sig = n3;
    printf("%d ", n1->info);            //output: 3
    printf("%d ", n1->sig->info);       //output: 3
    printf("%d", n1->sig->sig->info);   //output: 3
    free(n1);
    free(n2);
    free(n3);
    return 0;
}
```



Función int main() con menú de opciones

Martes 20 Septiembre 2022

```c
int main(){
    nodo_t *lista = NULL;
    int opcion = 0, num = 0;
    do{
        opcion = selecciona_opcion();
        switch(opcion){
            case 0: puts("Saliendo del programa...");
                break;
            case 1: printf("Valor del numero: ");
                scanf("%d", &num);
                insertar_inicio(&lista, num);
                break;
            case 2: printf("Valor del numero: ");
                scanf("%d", &num);
                insertar_final(&lista, num);
                break;
            case 3: eliminar_inicio(&lista);
                break;
            case 4: eliminar_final(&lista);
                break;
            case 5: imprimir_lista(lista);
                break;
            case 6: printf("Valor del numero: ");
                scanf("%d", &num);
                elimina_nodo(&lista, num);
                break;
            case 7: liberar_lista(&lista);
                break;
            case 8: printf("Valor del numero: ");
                scanf("%d", &num);
                printf("%p\n", buscar_dato(lista, num));
                break;
            case 9: printf("%d\n", tam_lista(lista));
                break;
            default: puts("Opcion no valida");
                break;
        }
    }while(opcion != 0);
    liberar_lista(&lista);
    return 0;
}
```

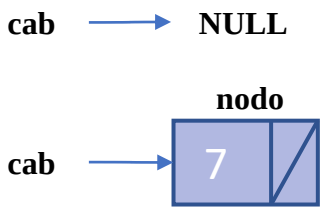
Función para seleccionar una opción

```c
int selecciona_opcion(){
    puts("Selecciona una opcion");
    puts("0. Salir");
    puts("1. Insertar inicio");
    puts("2. Insertar al final");
    puts("3. Eliminar al inicio");
    puts("4. Eliminar al final");
    puts("5. Imprimir toda la lista");
    puts("6. Eliminar nodo arbitrario");
    puts("7. Eliminar/Liberar toda la lista");
    puts("8. Buscar dato");
    puts("9. Imprimir tamaño de la lista");
    int opcion;
    scanf("%d", &opcion);
    return opcion;
}
```

puts() funciona como printf, pero es para imprimir en pantalla puro texto, y ya imprime el salto de línea sin necesidad de ponerlo.

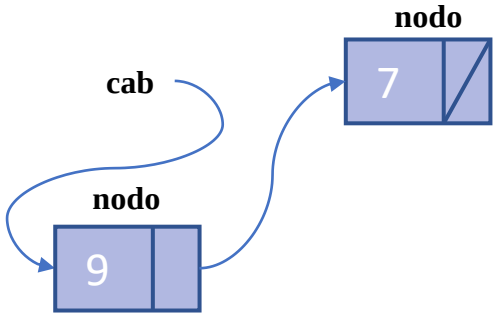
Función para insertar al inicio de la lista

```c
void insertar_inicio(nodo_t **cab, int dato){
    nodo_t *nodo = crea_nodo();
    nodo->info = dato;
    nodo->sig = *cab;
    *cab = nodo;
}
```



Ejemplo:

```c
int main(){
    nodo_t *lista = NULL;
    insertar_inicio(&lista, 7);
    insertar_inicio(&lista, 9);
    return 0;
}
```



Función para imprimir la lista (iterativo)

```c
void imprimir_lista(const nodo_t *cab){
    const nodo_t *it = cab;
    while(it != NULL){
        printf("%d ", it->info);
        it = it->sig;
    }
}
```

Función para imprimir la lista (recursivo)

```c
void imprimir_lista(const nodo_t *cab){
    if(cab == NULL){
        return;
    }
    printf("%d ", cab->info);
    imprimir_lista(cab->sig);
}
```

```
//Si lo intercambiamos, se imprime al revés.
```

Función para buscar un dato (iterativo)

```c
nodo_t *buscar_dato(const nodo_t *cab, int dato){
    for(const nodo_t *it = cab; it != NULL; it = it->sig){
        if(it->info == dato){
            return (nodo_t *)it;
        }
    }
    return NULL;
}
```

Función para liberar lista (iterativo)

```c
void liberar_lista(nodo_t **cab){
    nodo_t *it = *cab;
    while(it != NULL){
        nodo_t *tmp = it;
        it = it->sig;
        free(tmp);
    }
    *cab = NULL;
}
```

Función para liberar lista (recursivo)

```c
void liberar_lista(nodo_t **cab){
    if(*cab == NULL){
        return;
    }
    else{
        liberar_lista(&((*cab)->sig));
        free(*cab);
    }
    *cab = NULL;
}
```

Función para insertar al final de la lista (iterativo)

Jueves 22 Septiembre 2022

```c
void insertar_final(nodo_t **cab, int dato){
    nodo_t *nodo = crea_nodo();
    nodo->info = dato;
    if(*cab == NULL){
        *cab = nodo;
    }
    else{
        nodo_t *ultimo = *cab;
        while(ultimo->sig != NULL){
            ultimo = ultimo->sig;
        }
        ultimo->sig = nodo;
    }
}
```

Función para insertar al final de la lista (recursivo)

```c
void insertar_final(nodo_t **cab, int dato){
    nodo_t *nodo = crea_nodo();
    nodo->info = dato;
    if(*cab == NULL){
        *cab = nodo;
        return;
    }
    else{
        insertar_final(&(*cab)->sig, dato);
    }
}
```

Función para eliminar al inicio de la lista

```c
void eliminar_inicio(nodo_t **cab){
    nodo_t *tmp;
    if(*cab != NULL){
        tmp = *cab;
        *cab = (*cab)->sig;
        free(tmp);
    }
}
```

omegaUp — Ejercicio de OmegaUp: 14867. Listas enlazadas — Viernes 23 Septiembre 2022

Descripción

Dada una lista de enteros A inicialmente vacía, imprimir el contenido de A después de realizar N de las siguientes operaciones:

Operación	Explicación
push_front v	Agrega v al inicio de A
pop_front	Elimina el primer elemento de A
push_back v	Agrega v al final de A
pop_back	Elimina el último elemento de A
erase v	Elimina el primer elemento de A con valor igual a v
Clear	Elimina todos los elementos de A

Entrada

La primer línea contiene un entero N $(1 \leq N \leq 100)$, el número de operaciones a realizar. Las siguientes N líneas contienen las operaciones a realizar.

Salida

Una sola linea con el contenido final de A, cada elemento separado por un espacio.

Ejemplo 1

Entrada	Salida
6 push_back 3 push_front 7 clear push_front 5 erase 2 push_front 9	9 5

Ejemplo 2

Entrada	Salida
9 push_front 1 push_back 2 push_back 5 push_back 1 push_front 2 erase 1 push_front 6 push_back 3 erase 2	6 2 5 1 3

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

struct nodo{
    int info;
    struct nodo *sig;
};
typedef struct nodo nodo_t;

void insertar_inicio(nodo_t **, int);
void insertar_final(nodo_t **, int);
void imprimir_lista(const nodo_t *);
void eliminar_inicio(nodo_t **);
void eliminar_final(nodo_t **);
void elimina_nodo(nodo_t **, int);
void liberar_lista(nodo_t **);
nodo_t *crea_nodo();

int main(){
    nodo_t *lista = NULL;
    int n, x;
    char comando[12];
    scanf("%d", &n);
    for(int i = 0; i < n; i++){
        scanf("\n%s", comando);
        if(strcmp(comando, "push_front") == 0){
            scanf("%d", &x);
            insertar_inicio(&lista, x);
        }
        else if(strcmp(comando, "pop_front") == 0){
            eliminar_inicio(&lista);
        }
        else if(strcmp(comando, "push_back") == 0){
            scanf("%d", &x);
            insertar_final(&lista, x);
        }
        else if(strcmp(comando, "pop_back") == 0){
            eliminar_final(&lista);
        }
        else if(strcmp(comando, "erase") == 0){
            scanf("%d", &x);
            elimina_nodo(&lista, x);
        }
        else{
```

```
47              liberar_lista(&lista);
48          }
49      }
50      imprimir_lista(lista);
51      liberar_lista(&lista);
52      return 0;
53  }
54
55  nodo_t *crea_nodo(){
56      nodo_t *nodo = NULL;
57      nodo = (nodo_t*)malloc(sizeof(nodo_t));
58      assert(nodo != NULL);
59      nodo->info = 0;
60      nodo->sig = NULL;
61      return nodo;
62  }
63
64  void insertar_inicio(nodo_t **cab, int dato){
65      nodo_t *nodo = crea_nodo();
66      nodo->info = dato;
67      nodo->sig = *cab;
68      *cab = nodo;
69  }
70
71  void insertar_final(nodo_t **cab, int dato){
72      nodo_t *nodo = crea_nodo();
73      nodo->info = dato;
74      if(*cab == NULL){
75          *cab = nodo;
76      }
77      else{
78          nodo_t *ultimo = *cab;
79          while(ultimo->sig != NULL){
80              ultimo = ultimo->sig;
81          }
82          ultimo->sig = nodo;
83      }
84  }
85
86  void imprimir_lista(const nodo_t *cab){
87      const nodo_t *it = cab;
88      while(it != NULL){
89          printf("%d ", it->info);
90          it = it->sig;
91      }
92      printf("\n");
```

```
93  }
94
95  void eliminar_inicio(nodo_t **cab){
96      nodo_t *tmp;
97      if(*cab != NULL){
98          tmp = *cab;
99          *cab = (*cab)->sig;
100         free(tmp);
101     }
102 }
103
104 void liberar_lista(nodo_t **cab){
105     nodo_t *it = *cab;
106     while(it != NULL){
107         nodo_t *tmp = it;
108         it = it->sig;
109         free(tmp);
110     }
111     *cab = NULL;
112 }
113
114 void eliminar_final(nodo_t **cab){
115     if(*cab == NULL){
116         return;
117     }
118     else if((*cab)->sig == NULL){
119         nodo_t *tmp = *cab;
120         free(tmp);
121         *cab = NULL;
122     }
123     else{
124         nodo_t *ultimo = *cab;
125         nodo_t *penultimo = NULL;
126         while(ultimo->sig != NULL){
127             penultimo = ultimo;
128             ultimo = ultimo->sig;
129         }
130         free(ultimo);
131         penultimo->sig = NULL;
132     }
133 }
134
135 void elimina_nodo(nodo_t **cab, int x){
136     if(*cab == NULL){
137         return;
138     }
```

```
139       else if((*cab)->info == x){
140           nodo_t *tmp = *cab;
141           *cab = (*cab)->sig;
142           free(tmp);
143           return;
144       }
145       else{
146           nodo_t *prev = *cab, *it = (*cab)->sig;
147           while(it != NULL && it->info != x){
148               prev = it;
149               it = it->sig;
150           }
151           if(it != NULL){
152               prev->sig = it->sig;
153               free(it);
154           }
155       }
156   }
```

Función para eliminar al final de la lista (iterativo) Lunes 26 Septiembre 2022

```
1   void eliminar_final(nodo_t **cab){
2       if(*cab == NULL){
3           return;
4       }
5       else if((*cab)->sig == NULL){
6           nodo_t *tmp = *cab;
7           free(tmp);
8           *cab = NULL;
9       }
10      else{
11          nodo_t *ultimo = *cab;
12          nodo_t *penultimo = NULL;
13          while(ultimo->sig != NULL){
14              penultimo = ultimo;
15              ultimo = ultimo->sig;
16          }
17          free(ultimo);
18          penultimo->sig = NULL;
19      }
20  }
```

Función para eliminar al final de la lista (recursivo)

```c
void eliminar_final(nodo_t **cab){
    if(*cab == NULL){
        return;
    }
    else if((*cab)->sig == NULL){
        nodo_t *tmp = *cab;
        free(tmp);
        *cab = NULL;
        return;
    }
    else{
        eliminar_final(&(*cab)->sig);
    }
}
```

Función para eliminar nodo arbitrario (iterativo)

Martes 27 Septiembre 2022

```c
void elimina_nodo(nodo_t **cab, int x){
    if(*cab == NULL){
        return;
    }
    else if((*cab)->info == x){
        nodo_t *tmp = *cab;
        *cab = (*cab)->sig;
        free(tmp);
        return;
    }
    else{
        nodo_t *prev = *cab, *it = (*cab)->sig;
        while(it != NULL && it->info != x){
            prev = it;
            it = it->sig;
        }
        if(it != NULL){
            prev->sig = it->sig;
            free(it);
        }
    }
```

Función para contar el número de elementos (iterativo)

Jueves 29 Septiembre 2022

```c
int tam_lista(const nodo_t *cab){
    int cont = 0;
    const nodo_t *it = cab;
    while(it != NULL){
        cont++;
        it = it->sig;
    }
    return cont;
}
```

Función para contar el número de elementos (recursivo)

```c
int tam_lista(const nodo_t *cab){
    if(cab == NULL){
        return 0;
    }
    const nodo_t *resto = cab->sig;
    int tam_resto = tam_lista(resto);
    return 1 + tam_resto;
}
```

Función para eliminar nodo arbitrario (recursivo)

```c
void elimina_nodo(nodo_t **cab, int x){
    if(*cab == NULL){
        return;
    }
    else if((*cab)->info == x){
        nodo_t *tmp = *cab;
        *cab = (*cab)->sig;
        free(tmp);
        return;
    }
    else{
        elimina_nodo(&(*cab)->sig, x);
    }
}
```

Función para buscar un dato (recursivo)

```
1  nodo_t *buscar_dato(const nodo_t *cab, int dato){
2      if(cab == NULL){
3          return NULL;
4      }
5      else if(cab->info == dato){
6          return cab;
7      }
8      else{
9          buscar_dato(&(*cab)->sig, dato);
10     }
11 }
```

Tarea #4: Listas Simples Enlazadas

Instructions

Implement a singly linked list.

Like an array, a linked list is a simple linear data structure. Several common data types can be implemented using linked lists, like queues, stacks, and associative arrays.

A linked list is a collection of data elements called *nodes*. In a *singly linked list* each node holds a value and a link to the next node.

You will write an implementation of a singly linked list. Implement a Node to hold a value and pointers to the next node. Then implement a List which offers an array-like interface for adding and removing items:

* push_back (*insert value at back*);
* pop_back (*remove value at back*);
* push_front (*insert value at front*);
* pop_front (*remove value at front*);
* front (*gets the value at front*);
* back (*gets the value at back*);

To keep your implementation simple, the tests will not cover error conditions. Specifically: pop_back, pop_front, back, front will never be called on an empty list.

linked_list.h

```c
#ifndef LINKED_LIST_H
#define LINKED_LIST_H

#include <stddef.h>
#include <stdbool.h>

typedef int list_data_t;
typedef struct list_node list_node_t;

struct list_node {
   list_data_t data;
   struct list_node *next;
};

// initialize the list
void list_init(list_node_t **head);
```

```
// destroys an entire list
void list_destroy(list_node_t **head);

// checks if list is empty
bool list_is_empty(const list_node_t *head);

// counts the items on a list
size_t list_count(const list_node_t *head);

// inserts item at back of a list
void list_push_back(list_node_t **head, list_data_t item_data);

// returns the item from the back of a list
list_data_t list_back(const list_node_t *head);

// removes item from back of a list
void list_pop_back(list_node_t **head);

// inserts item at front of a list
void list_push_front(list_node_t **head, list_data_t item_data);

// returns the item from the front of a list
list_data_t list_front(const list_node_t *head);

// removes item from front of a list
void list_pop_front(list_node_t **head);

// deletes a node that holds the matching data
void list_delete(list_node_t **head, list_data_t data);


#endif
```

test_linked_list.c

```
#include <stddef.h>
#include "test-framework/unity.h"
#include "linked_list.h"

static list_node_t *list;

void setUp(void){
   list_init(&list);
}

void tearDown(void){
   list_destroy(&list);
}
```

```c
static void test_empty_list(void){
   TEST_IGNORE();
   TEST_ASSERT_TRUE(list_is_empty(list));
}

static void test_count_an_empty_list(void){
   TEST_IGNORE();
   TEST_ASSERT_EQUAL(0, list_count(list));
}

static void test_push_front_and_get_first_element_from_the_list(void){
   TEST_IGNORE();   // delete this line to run test
   list_push_front(&list, 7);
   TEST_ASSERT_EQUAL(7, list_front(list));
}

static void test_push_back_and_get_last_element_from_the_list(void){
   TEST_IGNORE();   // delete this line to run test
   list_push_back(&list, 7);
   TEST_ASSERT_EQUAL(7, list_back(list));
}

static void test_add_remove_at_the_end_of_the_list(void){
   TEST_IGNORE();   // delete this line to run test
   list_push_back(&list, 11);
   list_push_back(&list, 13);
   TEST_ASSERT_EQUAL(13, list_back(list));
   list_pop_back(&list);
   TEST_ASSERT_EQUAL(11, list_back(list));
}

static void test_pop_front_gets_an_element_from_the_list(void){
   TEST_IGNORE();
   list_push_back(&list, 17);
   TEST_ASSERT_EQUAL(17, list_front(list));
}

static void test_pop_front_gets_first_element_from_the_list(void){
   TEST_IGNORE();
   list_push_back(&list, 23);
   list_push_back(&list, 5);
   TEST_ASSERT_EQUAL(23, list_front(list));
   list_pop_front(&list);
   TEST_ASSERT_EQUAL(5, list_front(list));
}

static void test_push_front_adds_element_at_start_of_the_list(void){
   TEST_IGNORE();
   list_push_front(&list, 23);
   list_push_front(&list, 5);
   TEST_ASSERT_EQUAL(5, list_front(list));
   list_pop_front(&list);
   TEST_ASSERT_EQUAL(23, list_front(list));
}

static void test_push_pop_front_back_can_be_used_in_any_order(void){
   TEST_IGNORE();
```

```
    list_push_back(&list, 1);
    list_push_back(&list, 2);
    TEST_ASSERT_EQUAL(2, list_back(list));
    list_pop_back(&list);
    list_push_back(&list, 3);
    TEST_ASSERT_EQUAL(1, list_front(list));
    list_pop_front(&list);
    list_push_front(&list, 4);
    list_push_back(&list, 5);
    TEST_ASSERT_EQUAL(4, list_front(list));
    list_pop_front(&list);
    TEST_ASSERT_EQUAL(5, list_back(list));
    list_pop_back(&list);
    TEST_ASSERT_EQUAL(3, list_front(list));
}

static void test_count_a_list_with_items(void){
    TEST_IGNORE();
    list_push_back(&list, 37);
    list_push_back(&list, 1);
    TEST_ASSERT_EQUAL(2, list_count(list));
}

static void test_count_is_correct_after_mutation(void){
    TEST_IGNORE();
    list_push_back(&list, 31);
    TEST_ASSERT_EQUAL(1, list_count(list));
    list_push_front(&list, 43);
    TEST_ASSERT_EQUAL(2, list_count(list));
    list_pop_front(&list);
    TEST_ASSERT_EQUAL(1, list_count(list));
    list_pop_back(&list);
    TEST_ASSERT_EQUAL(0, list_count(list));
}

static void test_pop_back_to_empty_does_not_break_the_list(void){
    TEST_IGNORE();
    list_push_back(&list, 41);
    list_push_back(&list, 59);
    list_pop_back(&list);
    list_pop_back(&list);
    list_push_back(&list, 47);
    TEST_ASSERT_EQUAL(1, list_count(list));
    TEST_ASSERT_EQUAL(47, list_back(list));
}

static void test_pop_front_to_empty_does_not_break_the_list(void){
  TEST_IGNORE();
    list_push_back(&list, 41);
    list_push_back(&list, 59);
    list_pop_front(&list);
    list_pop_front(&list);
    list_push_back(&list, 47);
    TEST_ASSERT_EQUAL(1, list_count(list));
    TEST_ASSERT_EQUAL(47, list_front(list));
}
```

```
static void test_deletes_the_only_element(void){
   TEST_IGNORE();
   list_push_back(&list, 61);
   list_delete(&list, 61);
   TEST_ASSERT_EQUAL(0, list_count(list));
}

static void test_deletes_the_element_with_the_specified_value_from_the_list(void){
   TEST_IGNORE();
   list_push_back(&list, 71);
   list_push_back(&list, 83);
   list_push_back(&list, 79);
   list_delete(&list, 83);
   TEST_ASSERT_EQUAL(2, list_count(list));
   TEST_ASSERT_EQUAL(79, list_back(list));
   list_pop_back(&list);
   TEST_ASSERT_EQUAL(71, list_front(list));
}

static void test_deletes_the_element_with_the_specified_value_from_the_list_reassigns_tail(void){
   TEST_IGNORE();
   list_push_back(&list, 71);
   list_push_back(&list, 83);
   list_push_back(&list, 79);
   list_delete(&list, 83);
   TEST_ASSERT_EQUAL(2, list_count(list));
   TEST_ASSERT_EQUAL(79, list_back(list));
   list_pop_back(&list);
   TEST_ASSERT_EQUAL(71, list_back(list));
}

static void
test_deletes_the_element_with_the_specified_value_from_the_list_reassigns_head(void){
   TEST_IGNORE();
   list_push_back(&list, 71);
   list_push_back(&list, 83);
   list_push_back(&list, 79);
   list_delete(&list, 83);
   TEST_ASSERT_EQUAL(2, list_count(list));
   TEST_ASSERT_EQUAL(71, list_front(list));
   list_pop_front(&list);
   TEST_ASSERT_EQUAL(79, list_front(list));
}

static void test_deletes_the_first_of_two_elements(void){
   TEST_IGNORE();
   list_push_back(&list, 97);
   list_push_back(&list, 101);
   list_delete(&list, 97);
   TEST_ASSERT_EQUAL(1, list_count(list));
   TEST_ASSERT_EQUAL(101, list_back(list));
}

static void test_deletes_the_second_of_two_elements(void){
   TEST_IGNORE();
   list_push_back(&list, 97);
```

```
    list_push_back(&list, 101);
    list_delete(&list, 101);
    TEST_ASSERT_EQUAL(1, list_count(list));
    TEST_ASSERT_EQUAL(97, list_back(list));
}

static void
test_delete_does_not_modify_the_list_if_the_element_is_not_found(void){
    TEST_IGNORE();
    list_push_back(&list, 89);
    list_delete(&list, 103);
    TEST_ASSERT_EQUAL(1, list_count(list));
}

static void test_deletes_only_the_first_occurrence(void){
    TEST_IGNORE();
    list_push_back(&list, 73);
    list_push_back(&list, 9);
    list_push_back(&list, 9);
    list_push_back(&list, 107);
    list_delete(&list, 9);
    TEST_ASSERT_EQUAL(3, list_count(list));
    TEST_ASSERT_EQUAL(107, list_back(list));
    list_pop_back(&list);
    TEST_ASSERT_EQUAL(9, list_back(list));
    list_pop_back(&list);
    TEST_ASSERT_EQUAL(73, list_back(list));
}

int main(void){
    UnityBegin("test_linked_list.c");
    RUN_TEST(test_empty_list);
    RUN_TEST(test_count_an_empty_list);
    RUN_TEST(test_push_front_and_get_first_element_from_the_list);
    RUN_TEST(test_push_back_and_get_last_element_from_the_list);
    RUN_TEST(test_add_remove_at_the_end_of_the_list);
    RUN_TEST(test_pop_front_gets_an_element_from_the_list);
    RUN_TEST(test_pop_front_gets_first_element_from_the_list);
    RUN_TEST(test_push_front_adds_element_at_start_of_the_list);
    RUN_TEST(test_push_pop_front_back_can_be_used_in_any_order);
    RUN_TEST(test_count_a_list_with_items);
    RUN_TEST(test_count_is_correct_after_mutation);
    RUN_TEST(test_pop_back_to_empty_does_not_break_the_list);
    RUN_TEST(test_pop_front_to_empty_does_not_break_the_list);
    RUN_TEST(test_deletes_the_only_element);
    RUN_TEST(test_deletes_the_element_with_the_specified_value_from_the_list);
    RUN_TEST(test_deletes_the_element_with_the_specified_value_from_the_list_reassigns_tail);
    RUN_TEST(test_deletes_the_element_with_the_specified_value_from_the_list_reassigns_head);
    RUN_TEST(test_deletes_the_first_of_two_elements);
    RUN_TEST(test_deletes_the_second_of_two_elements);
    RUN_TEST(test_delete_does_not_modify_the_list_if_the_element_is_not_found);
    RUN_TEST(test_deletes_only_the_first_occurrence);
    return UnityEnd();
}
```

linked_list.c

```c
#include "linked_list.h"
#include <stdlib.h>

void list_init(list_node_t **head){
    *head = NULL;
}

void list_destroy(list_node_t **head){
    list_node_t *it = *head;
    while(it != NULL){
        list_node_t *tmp = it;
        it = it->next;
        free(tmp);
    }
    *head = NULL;
}

bool list_is_empty(const list_node_t *head){
    if(head == NULL){
        return true;
    }
    else{
        return false;
    }
}

size_t list_count(const list_node_t *head){
    int cont = 1;
    if(head == NULL){
        return 0;
    }
    else{
        const list_node_t *last = head;
        while(last->next != NULL){
            last = last->next;
            cont++;
        }
        return cont;
    }
}

void list_push_front(list_node_t **head, list_data_t item_data){
    list_node_t *node = NULL;
```

```
44      node = (list_node_t *)malloc(sizeof(list_node_t));
45      node->data = item_data;
46      node->next = *head;
47      *head = node;
48  }
49
50  list_data_t list_front(const list_node_t *head){
51      return head->data;
52  }
53
54  void list_push_back(list_node_t **head, list_data_t item_data){
55      list_node_t *node = NULL;
56      node = (list_node_t *)malloc(sizeof(list_node_t));
57      node->data = item_data;
58      node->next = NULL;
59      if(*head == NULL){
60          *head = node;
61      }
62      else{
63          list_node_t *last = *head;
64          while(last->next != NULL){
65              last = last->next;
66          }
67          last->next = node;
68      }
69  }
70
71  list_data_t list_back(const list_node_t *head){
72      const list_node_t *last = head;
73      while(last->next != NULL){
74          last = last->next;
75      }
76      return last->data;
77  }
78
79  void list_pop_back(list_node_t **head){
80      if((*head)->next == NULL){
81          list_node_t *tmp = *head;
82          free(tmp);
83          *head = NULL;
84      }
85      else{
86          list_node_t *last = *head;
87          list_node_t *plast = NULL;
88          while(last->next != NULL){
89              plast = last;
```

```
90              last = last->next;
91          }
92          free(last);
93          plast->next = NULL;
94      }
95  }
96
97  void list_pop_front(list_node_t **head){
98      list_node_t *tmp = NULL;
99      if(*head != NULL){
100         tmp = *head;
101         *head = (*head)->next;
102         free(tmp);
103     }
104 }
105
106 void list_delete(list_node_t **head, list_data_t data){
107     if(*head == NULL){
108         return;
109     }
110     else if((*head)->data == data){
111         list_node_t *tmp = *head;
112         *head = (*head)->next;
113         free(tmp);
114         return;
115     }
116     else{
117         list_node_t *prev = *head, *it = (*head)->next;
118         while(it != NULL && it->data != data){
119             prev = it;
120             it = it->next;
121         }
122         if(it != NULL){
123             prev->next = it->next;
124             free(it);
125         }
126     }
127 }
```

1. Algoritmo recursivo para eliminar al final de la lista

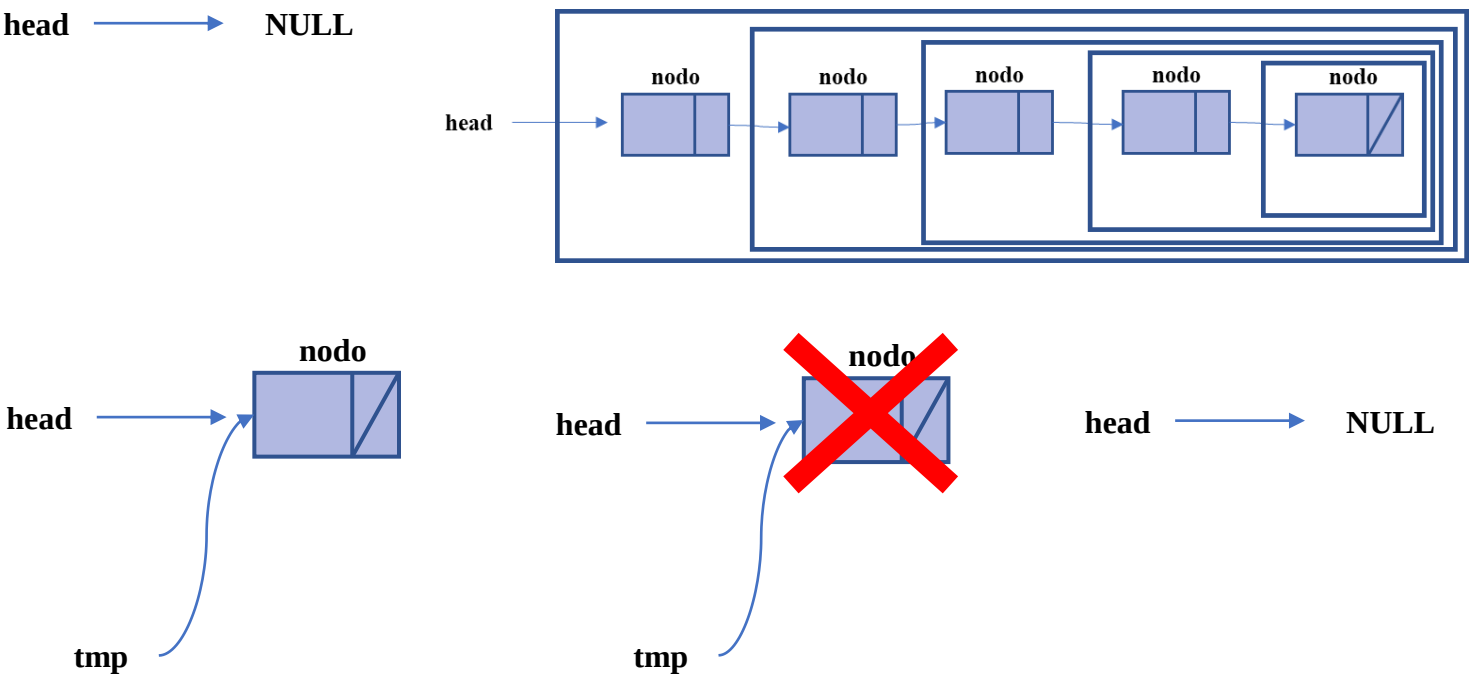
```
1  void list_pop_back(list_node_t **head){
2      if(*head == NULL){
3          return;
4      }
5      else if((*head)->next == NULL){
6          list_node_t *tmp = *head;
7          free(tmp);
8          *head = NULL;
9          return;
10     }
11     list_pop_back(&(*head)->next);
12 }
```

Explicación del algoritmo recursivo

Primero pongo una excepción en la que si la cabecera es igual a NULL, regrese la función sin hacer nada, esto por costumbre para evitar la posibilidad de que se use cuando la lista este vacía.

En la línea 5 pongo el caso base en el que se va a detener la función y va a dejar de llamarse a si misma, que es cuando el siguiente de la cabecera sea NULL, ya que eso significaría que ese sería el último elemento, al llegar a ese elemento, primero creo un nodo temporal para poder liberarlo sin perder la dirección, y así apuntarlo a NULL, que sería la nueva dirección a la que apunta el nodo anterior.

En la línea 11 pongo lo que hace la función si no se cumple el caso base, que básicamente es volver a llamar a la función enviando la dirección que hay en cabecera siguiente.

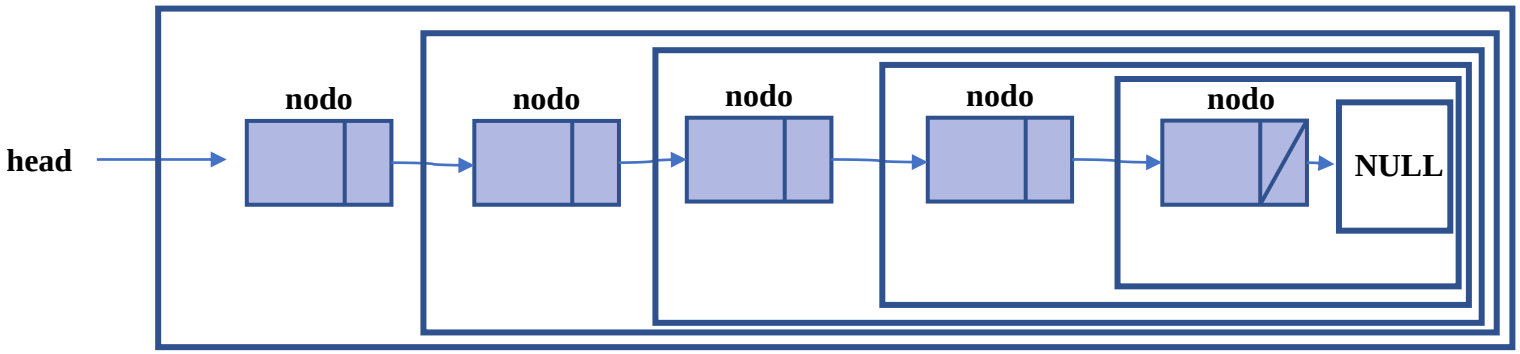


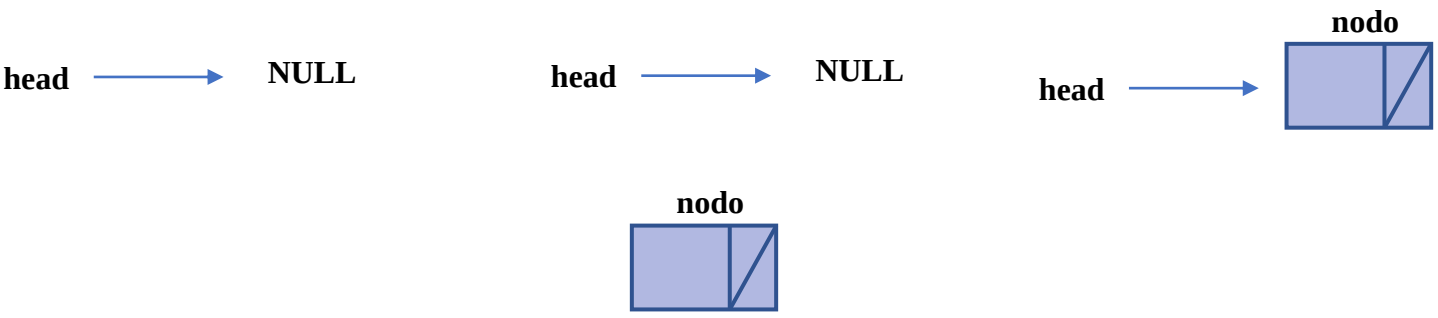
2. Algoritmo recursivo para insertar un nodo al final de la lista

```
1  void list_push_back(list_node_t **head, list_data_t item_data){
2      if(*head == NULL){
3          list_node_t *node = NULL;
4          node = (list_node_t *)malloc(sizeof(list_node_t));
5          node->data = item_data;
6          node->next = NULL;
7          *head = node;
8          return;
9      }
10     list_push_back(&(*head)->next, item_data);
11 }
```

Explicación del algoritmo recursivo

El algoritmo se va a estar llamando así mismo mandándose la dirección que tiene en cabecera siguiente (como se ve en la línea 10), hasta que se cumpla el caso base o condición en la que la cabecera sea NULL, puesto que lla llego al final, que es en donde vamos a insertar el nodo, creo el nodo, y posteriormente hago que la cabecera (la cabecera que ha sido llamada varias veces haciéndola avanzar) apunte al nodo en vez de a NULL.



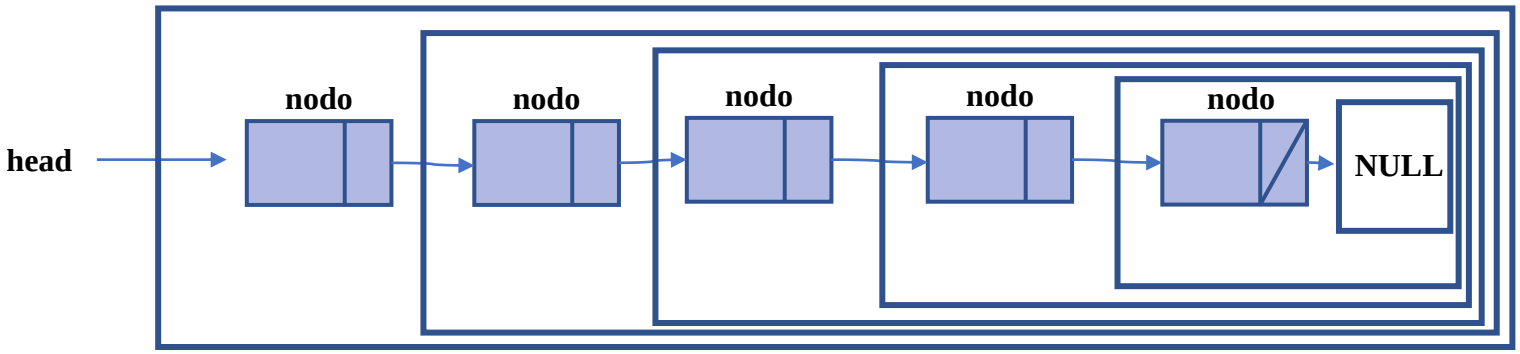


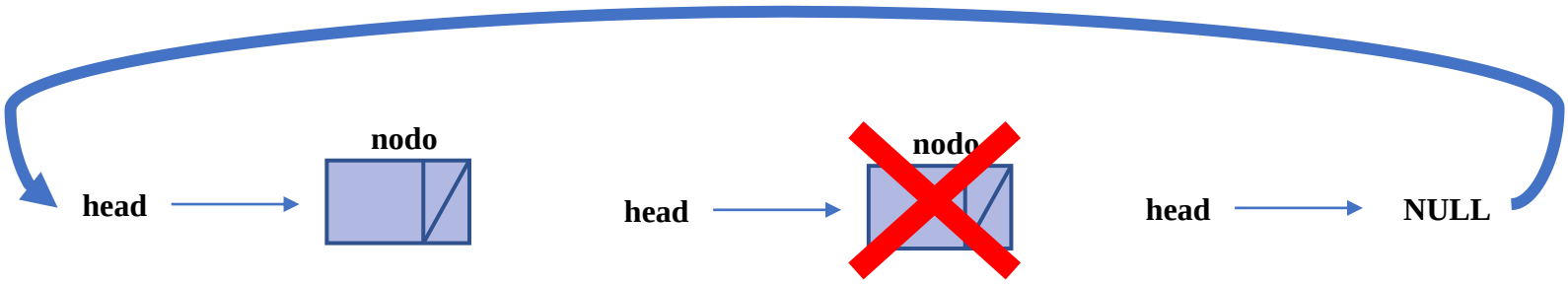
3. Algoritmo recursivo para eliminar todos los nodos de la lista

```c
void list_destroy(list_node_t **head){
    if(*head == NULL){
        return;
    }
    else{
        list_destroy(&((*head)->next));
        free(*head);
    }
    *head = NULL;
}
```

Explicación del algoritmo recursivo

El algoritmo se va a estar llamando así mismo hasta que llegue a NULL, una vez hecho eso, va a comenzar a liberar el nodo que haya en lo que apunta la nueva cabecera, que será el del final, y como lo acaba de liberar, ahora el nodo anterior apunta a NULL, y comienza a liberar todos los nodos.





4. Las funciones para acceder al valor del primer/último elemento de la lista hacen la suposición de que la lista no está vacía

a) ¿Qué ocurrirá si se llaman estas funciones para una lista que está vacía?

Segmentation Fault.
Debido a que la función va a intentar acceder al campo dato de la cabecera, pero si la lista esta vacía, va a intentar acceder a un dato en una dirección NULL, y ahí no hay nada.

b) ¿Qué deberían regresar estas funciones en caso de una lista vacía?

Debido a que las funciones son de tipo entero, y aunque en la descripción del ejercicio no especifica si la lista es únicamente de enteros positivos, todos los enteros de los tests son positivos, así que por practicidad, lo más fácil sería regresar un valor representativo como -1.

c) ¿Qué cambios deberías realizar en tu código para manejar estas situaciones?

```c
list_data_t list_front(const list_node_t *head){
    if(head == NULL){
        return -1;
    }
    else{
        return head->data;
    }
}
```

```c
list_data_t list_back(const list_node_t *head){
    if(head == NULL){
        return -1;
    }
    else{
        const list_node_t *last = head;
        while(last->next != NULL){
            last = last->next;
        }
        return last->data;
    }
}
```

5. ¿Cuál es el nombre de la prueba (test) que te resultó más difícil pasar?

Pues en realidad ninguna prueba se me hizo difícil, pero, si tuviera que elegir alguna, la de eliminar al final de la lista.

¿Por qué consideras que fue difícil?

Porque las demás funciones las hicimos en clases, esa la hice yo, y aunque a simple vista parecía funcionar, no tome en cuenta el caso en donde la lista tiene un solo elemento en la versión iterativa.

Al finalizar la tarea, ¿cuántas pruebas tuviste que ignorar?

Ninguna, si pasó todas y cada una de las pruebas.
Errores que llegué a tener, fue el segmentation fault, pero se pudo arreglar.

6. Inventa una nueva prueba y agrégala

```c
static void test_list_front_without_elements_in_the_list(void)
{
    //TEST_IGNORE();
    TEST_ASSERT_EQUAL(-1, list_front(list));
    TEST_ASSERT_EQUAL(-1, list_back(list));
}

int main(void)
{
    UnityBegin("test_linked_list.c");

    RUN_TEST(test_list_front_without_elements_in_the_list);

    return UnityEnd();
}
```

En base a las modificaciones del punto 4, agregue un test que corroborara el funcionamiento del código.
Esto fue agregado con otro git commit.

7. ¿Aplicaste la refactorización en tu código? Explica de qué manera lo hiciste

Cambiando la función que elimina un nodo arbitrario de manera recursiva, tal y como se muestra en el código de abajo, la función se llama así misma cambiando la cabecera a cabecera siguiente.
El if es en caso de que la cabecera sea igual a cero, lo cual ocurrirá si la lista esta vacía, o termina el recorrido recursivo sin encontrar el nodo con el valor buscado.
En el if else libera el nodo y hace el enlace entre el nodo anterior con el nodo siguiente, lo mismo aplica cuando solo hay un nodo, ya que lo que haría, sería apuntar la cabecera a NULL;

```c
void list_delete(list_node_t **head, list_data_t data){
    if(*head == NULL){
        return;
    }
    else if((*head)->data == data){
        list_node_t *tmp = *head;
        *head = (*head)->next;
        free(tmp);
        return;
    }
    list_delete(&((*head)->next), data);
}
```

8. ¿Qué hiciste particularmente bien en esta tarea?

Aplicar los conocimientos adquiridos en clase, generalmente no soy bueno entendiendo la recursividad, pero después de hacer la primer función recursiva, se me facilito mucho hacer recursivas las demás funciones.

9. ¿Qué pudiste haber hecho mejor en esta tarea?

Pensar en un mejor test de prueba para agregar, y pensar en más formas de refactorizar mi código.

10. ¿Qué nuevos conocimientos y experiencias adquiriste con esta tarea?

1. Cómo volver recursivas este tipo de funciones.
2. Considerar todos los casos base, ya que a pesar de creer que una función ya funciona, hay casos en los que no, y generaba un segmentation fault.
3. A utilizar codespaces desde visual studio code en vez del navegador.

2.2 Pilas, colas y conjuntos

Lunes 03 Octubre 2022

2.2.1 Pilas con listas

Estructuras

```c
struct nodo{
    int info;
    struct nodo *sig;
};
typedef struct nodo nodo_t;
```

```c
struct pila{
    nodo_t *tope;
    int tam;
};
typedef struct pila pila_t;
```

Función para inicializar la pila

```c
void inicializar(pila_t *pila){
    pila->tope = NULL;
    pila->tam = 0;
}
```

Funciones para saber si esta vacía o llena

```c
bool esta_vacia(const pila_t *p){
    if(p->tam == 0){
        return true;
    }
    return false;
}

bool esta_llena(const pila_t *p){
    return false;
}
```

Función para saber el tamaño de la pila

```c
int tam(const pila_t *p){
    return p->tam;
}
```

Función para apilar

```c
void apilar(pila_t *p, int dato){
    nodo_t *nodo = crea_nodo();
    nodo->sig = p->tope;
    nodo->info = dato;
    p->tope = nodo;
    (p->tam)++;
}
```

Función para desapilar

```c
int desapilar(pila_t *p){
    nodo_t *tmp = p->tope;
    if(p->tam > 0){
        p->tope = p->tope->sig;
        int dato = tmp->info;
        free(tmp);
        (p->tam)--;
        return dato;
    }
}
```

Función para liberar la pila

```c
void liberar(pila_t *p){
    for(nodo_t *it = p->tope; it != NULL; it = it->sig){
        nodo_t *tmp = it;
        it = it->sig;
        free(tmp);
    }
}
```

Ejemplo en int main()

```c
int main(){
    pila_t pila;
    inicializar(&pila);
    apilar(&pila, 1);
    apilar(&pila, 2);
    printf("%d\n", desapilar(&pila)); //output: 2
    printf("%d\n", desapilar(&pila)); //output: 1
    liberar(&pila);
    return 0;
}
```

2.2.2 Colas con listas

Martes 04 Octubre 2022

Estructuras

```c
struct nodo{
    int info;
    struct nodo *sig;
};
typedef struct nodo nodo_t;
```

```c
typedef struct{
    nodo_t *frente;
    nodo_t *atras;
    int tam;
}cola_t;
```

Función para inicializar la cola

```c
void inicializa_cola(cola_t *c){
    c->frente = NULL;
    c->atras = NULL;
    c->tam = 0;
}
```

Función para encolar

```c
void encolar(cola_t *c, int dato){
    nodo_t *nodo = crea_nodo();
    nodo->info = dato;
    nodo->sig = NULL;
    if(c->atras != NULL){
        c->atras->sig = nodo;
        c->atras = nodo;
    }
    else{
        c->atras = c->frente = nodo;
    }
    c->tam++;
}
```

Función para desencolar

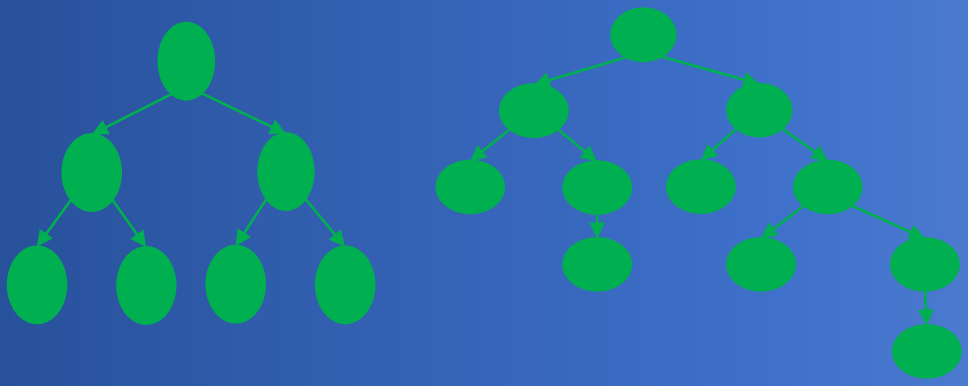
```c
int desencolar(cola_t *c){
    if(c->frente != NULL){
        int dato = c->frente->info;
        nodo_t *tmp = c->frente;
        c->frente = c->frente->sig;
        free(tmp);
        if(c->frente == NULL){
            c->atras = NULL;
        }
        c->tam--;
        return dato;
    }
    else{
        return -1;
    }
}
```

2.2.3 Conjuntos con listas

3. GRAFOS

Viernes 04 Noviembre 2022

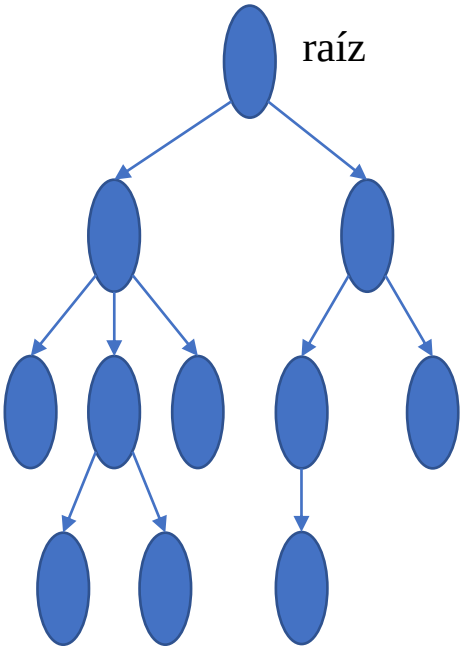
4. ÁRBOLES BINARIOS



4.1 Conceptos básicos de árboles

Objetivo: Diseñar y programar las estructuras y las operaciones básicas para la manipulación de grafos.

Lunes 07 Noviembre 2022

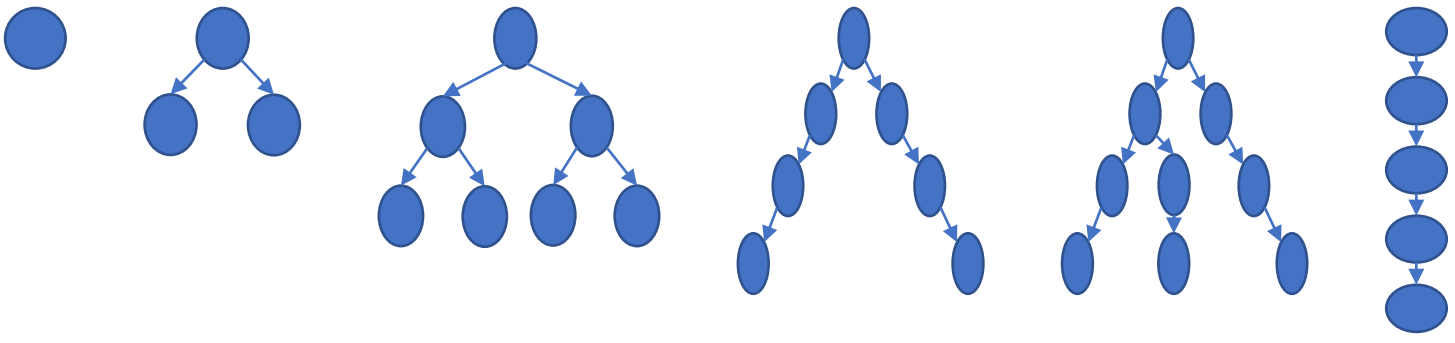


Terminología	
Raíz	Nodo principal, cabecera.
Hijos	Nodos izquierdo y derecho de un nodo.
Padre	Nodo con hijos.
Hermanos	Nodos que tienen el mismo padre.
Antecesor y descendiente	Si podemos ir desde el nodo A hacia el nodo B, entonces A es un antecesor de B, y B es un descendiente de A.
Hoja	Nodos sin hijos.
Nodo interno	Nodos que no son hojas.
Camino	Secuencia de aristas desde un nodo antecesor hasta un nodo descendiente.

Definiciones y aplicaciones

Un **árbol binario** es un árbol en el cual cada nodo puede tener a lo máximo 2 hijos.

Ejemplos:



Aplicaciones de los árboles:

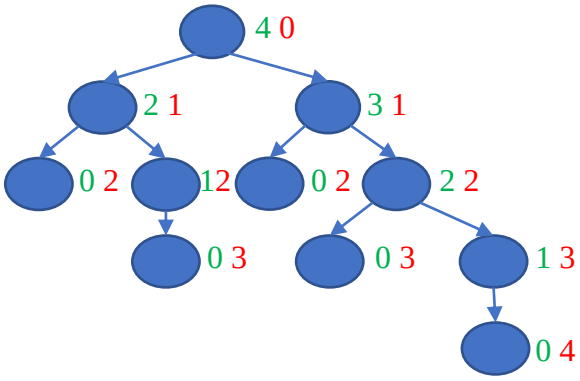
- Información con estructura jerárquica.
- Navegación web y redes.
- Bases de datos (árboles B y B+).
- Son la base para las estructuras eficientes como los conjuntos (sets).

Árbol binario completo

Todos los niveles excepto posiblemente el último están llenos y todos los nodos están lo más posible a la izquierda.

Altura de un nodo

Martes 08 Noviembre 2022



Altura de un nodo x

Es el número de aristas en el camino más largo desde x hasta una hoja. Las hojas tienen altura cero.
La altura de un árbol es la altura de la raíz (en este caso es 4).

La profundidad de un nodo x

Es la longitud del camino desde la raíz hasta el nodo x (cantidad de aristas). La profundidad de la raíz es cero.
Los nodos que tienen la misma profundidad están en un mismo nivel.
En un árbol binario perfecto, todos los niveles están completamente llenos.

El número máximo de nodos en el nivel i es 2^i. Un árbol perfecto de altura h tiene exactamente $2^{n+1} - 1$ nodos.

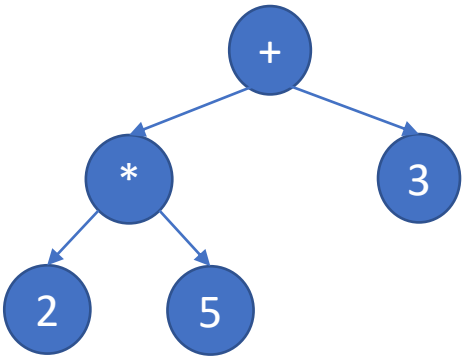
¿Cuál es la altura de un árbol binario perfecto que tiene N nodos?

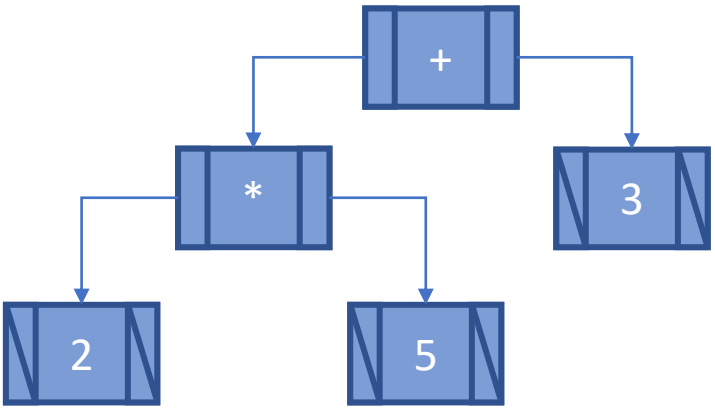
$$2^{n+1} - 1 = N \Rightarrow 2^{n+1} = N + 1 \qquad \log a^b = b \log a \qquad \log_x x = 1$$

$$\log_2\left(2^{h+1}\right) = \log_2(N+1) \Rightarrow (h+1)\log_2 2 = \log_2(N+1) \Rightarrow h = \log_2(N+1) - 1$$

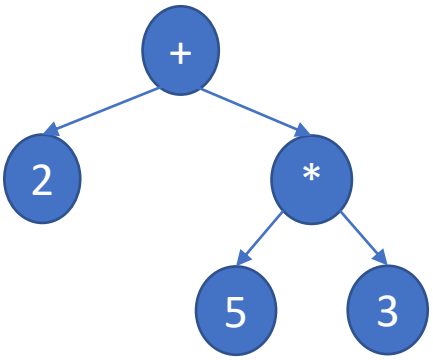
4.2 Árboles binarios de expresiones

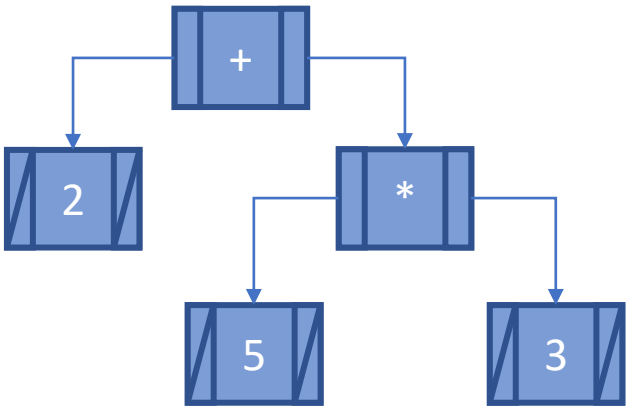
Jueves 10 Noviembre 2022



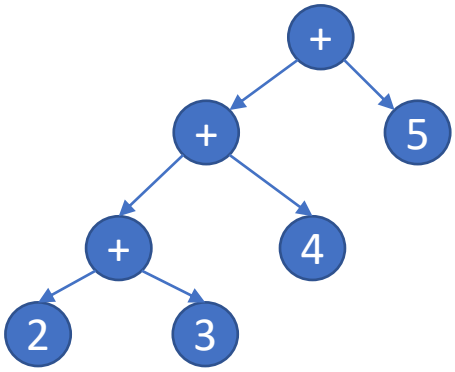


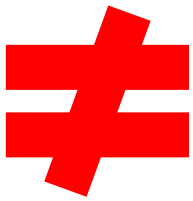
$$2 * 5 + 3 \Rightarrow 13$$

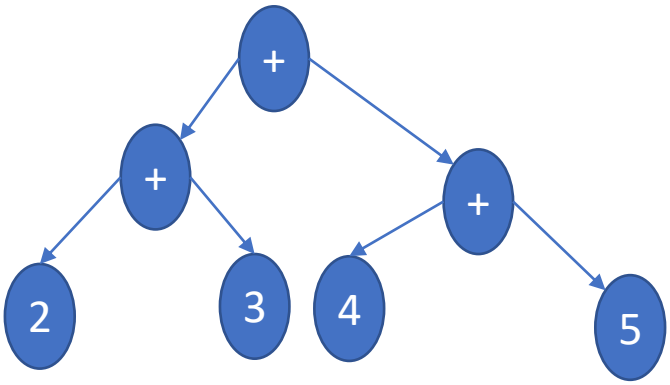




$$2 * (5 + 3) \Rightarrow 16$$







$$2 + 3 + 4 + 5 \Rightarrow 14$$

$$(2 + 3) + (4 + 5) \Rightarrow 14$$

Aunque para nosotros sea la misma operación matemática, son dos árboles completamente diferentes por los paréntesis.

4.2.1 Definir la estructura del nodo

```c
struct nodo{
    int info;
    struct nodo *izq;
    struct nodo *der;
};
typedef struct nodo nodo_t;
```

5. ÁRBOLES MULTICAMINOS