TEMARIO PROPEDEUTICO DE PROGRAMACIÓN

- Manejo de sintaxis en Lenguaje C o en Java o Python
- Tipos, operadores y expresiones.
- Control de flujo.
- apuntador y arreglos.
- Entrada y salida.
- Operaciones Vectores, matrices, arreglos y cadenas.
- Conectividad en la red y hacia la nube
- Solución de problemas
- Casos de estudio
- Conexion con Sockets
- Conexion con Web Services
- Despliegue en Consola y Web

Bibliografía

- Gonnella, G. (2022). TextFormats: Simplifying the definition and parsing of text formats in bioinformatics. *PloS one*, *17*(5), e0268910.
- Arnold, Ken, James Gosling and David Holmes (2005). The Java Programming Language, 3rd edition. NJ: Prentice Hall.
- Balagurusamy, E. (2007). Programming with JAVA: A Primer, 3rd edition. New Delhi: Tata McGraw-Hill.
- CM96
- Peter Coad and Mark Mayfield, Java Design: Building Better Apps and Applets, Yourdon Press, 1996.
- CH97
- Gary Cornell and Cay S. Horstmann, Core Java, second ed., SunSoft Press, 1997.
- ELW98
- Robert Eckstein and Marc Loy and Dave Wood, Java Swing, O'Reilly, 1998.
- Ceder, N. (2018). *The quick Python book*. Simon and Schuster.
- Kuhlman, D. (2009). *A python book: Beginning python, advanced python, and python exercises* (pp. 1-227). Lutz: Dave Kuhlman.

GUIA DE ESTUDIO

Los siguientes problemas requieren que conozcas los operandos de manipulación de apuntadors, cómo se sitúan los datos en memoria y el concepto de indirección. Escribe las definiciones de tipos, declaraciones y fragmentos de código en un archivo de texto plano en tu entorno de desarrollo y que compruebes los resultados mediante su compilación y ejecución. Para los ejercicios se suponen los siguientes tamaños de los tipos de datos básicos:

Tipo	Tamaño (bytes)
char, unsigned char	1
short int, unsigned short int	2
int, unsigned int, long int, unsigned long int	4
float	4
double, long double	8
apuntador de cualquier tipo	4

Suponte que se definen las siguientes estructuras de datos para guardar la información sobre las celdas con las que tiene posibilidad de conexión un teléfono móvil:

```
#define SIZE 100
/* Información sobre la celda */
struct informacion_celda
{
   char nombre[SIZE];              /* Nombre de la celda */
   unsigned int identificador;     /* Número identificador */
   float calidad_senal;            /* Calidad de la señal (entre 0 y 100) */
   struct informacion_operador *ptr_operador; /* apuntador a una segunda estructura
*/
};

/* Información sobre el operador */
struct informacion_operador
{
   char nombre[SIZE];          /* Cadena de texto con el nombre */
   unsigned int prioridad;     /* Prioridad de conexión */
   unsigned int ultima_comprob; /* Fecha de la última comprobación */
};
```

Responde a las siguientes preguntas:

1. ¿Qué tamaño en bytes ocupa una variable de tipo struct informacion_celda en memoria?
2. Las siguientes dos líneas declaran dos variables. ¿Cuál de ellas ocupa más espacio en memoria?

```
struct informacion_celda a;
```

3. struct informacion_celda *b;
4. ¿Qué tamaño tienen las siguientes variables?

```
struct informacion_celda *ptr1, *ptr2;
```

5. struct informacion_operador *i1, *i2;
6. Si una variable de tipo struct informacion_celda está almacenada en la posición de memoria 100, ¿qué dirección tienen cada uno de sus campos?

7. Si una variable de tipo struct informacion_celda * está almacenada en la posición de memoria 100, ¿qué dirección tiene cada uno de sus campos?
8. ¿Qué cambios debes hacer en las definiciones de la parte izquierda para que sean equivalentes a las descripciones de la parte derecha?

struct informacion_celda c; struct informacion_celda **c_ptr;	// variable de tipo estructura informacion_celda // apuntador a estructura informacion_celda;

9. ¿Se pueden hacer las siguientes asignaciones? ¿Qué declara exactamente la linea 3?

```
1  struct informacion_celda c;
2  struct informacion_celda *c_ptr = &c;
3  struct informacion_celda d;
4  struct informacion_celda *d_ptr = c_ptr;
```

10. Considera la siguiente declaración y asignación:

```
1  struct informacion_celda c;
2  struct informacion_celda *c_ptr;
3
4  c_ptr = *c;
```

11. ¿Es correcta? Y si lo es, ¿Qué contiene la variable c_ptr (no se pregunta por lo que apunta, sino su contenido)?
12. Si se declara una variable como “struct informacion_celda c;”, ¿qué tipo de datos es el que devuelve la expresión “&c.ptr_operador”?
13. Dado el siguiente código:

```
1   struct pack3
2   {
3       int a;
4   };
5   struct pack2
6   {
7       int b;
8       struct pack3 *next;
9   };
10  struct pack1
11  {
12      int c;
13      struct pack2 *next;
14  };
15
16  struct pack1 data1, *data_ptr;
17  struct pack2 data2;
18  struct pack3 data3;
19
20  data1.c = 30;
21  data2.b = 20;
22  data3.a = 10;
23  dataPtr = &data1;
24  data1.next = &data2;
25  data2.next = &data3;
```

14. Decide si las siguientes expresiones son correctas y en caso de que lo sean escribe a que datos se acceden.

Expresión	Correcta	Valor
data1.c		
data_ptr->c		
data_ptr.c		
data1.next->b		
data_ptr->next->b		
data_ptr.next.b		
data_ptr->next.b		
(*(data_ptr->next)).b		
data1.next->next->a		

data_ptr->next->next.a		
data_ptr->next->next->a		
data_ptr->next->a		
data_ptr->next->next->b		

15. Supongamos que se escriben las siguientes declaraciones y asignaciones en un programa:

```
1  info_celda c;
2  info_celda_ptr c_ptr = &c;
3  info_operador op;
4  info_operador_ptr op_ptr = &op;
```

16. La estructura “c” contiene el campo “ptr_operador” precisamente para almacenar la información relativa al operador. ¿Qué expresión hay que usar en el código para guardar la información del operador “op” como parte de la estructura “c”? Teniendo en cuenta los valores que se asignan en las declaraciones, escribe cuatro versiones equivalentes de esta expresión (utiliza “c”, “c_ptr”, “op” y “op_ptr”).
17. Supón ahora que la aplicación en la que se usan estas estructuras necesita almacenar la información para un máximo de 10 celdas. ¿Qué estructura de datos definirías?

Escribe un bucle con la variable declarada en el ejercicio anterior que asigne al campo ptr_operador el valor vacío.

18. La información sobre las celdas que se almacena en la estructura del ejercicio anterior la debe utilizar la aplicación para recordar cuál de ellas es la más próxima. Esta información puede cambiar a lo largo del tiempo. ¿Qué tipo de datos sugieres para almacenar esta información? Ofrece dos alternativas.
19. Se dispone de una estructura de tipo “info_celda c” que a su vez, en el campo “ptr_operador” tiene un apuntador a una estructura “info_operador cr”. ¿Qué tamaño tiene la estructura “c”? ¿Qué tamaño total ocupa la información incluyendo la información sobre el operador?
20. Escribe el cuerpo de la siguiente función:
21. void fill_in(info_celda_ptr dato, unsigned int id, float sq, info_operador_ptr op_ptr)
22. que asigna el valor de los parámetros “id”, “sq” y “op_ptr” a los campos “identificador”, “calidad_senal” y “ptr_operador” respectivamente de la estructura apuntada por el parámetro “dato”.
    ¿Cómo explicas que esta función asigne valores a unos campos y no devuelva resultado?
23. Considera las dos versiones del siguiente programa:

#include <stdio.h>
struct package
{
  int q;
};
 	
void set_value(struct package data, int value)
{
  data.q = value;
}
 	
int main()
{
  struct package p;
  p.q = 10;
  set_value(p, 20);
  printf("Value = %d\n", p.q);
  return 0;
}
	#include <stdio.h>
	struct package
	{
	  int q;
	};
	
	void set_value(struct package *d_ptr, int value)
	{
	  d_ptr->q = value;
	}
	
	int main()
	{
	  struct package p;
	  p.q = 10;
	  set_value(&p, 20);
	  printf("Value = %d\n", p.q);
	  return 0;
	}

24. La versión 1 del programa imprime el valor 10 por pantalla, y la versión 2 imprime el valor 20. Explica por qué.
    Sugerencia
    Puedes copiar y pegar el código en un archivo en tu entorno de desarrollo y verificar que los dos programas se comportan tal y como se dice.
25. ¿Qué cantidad de memoria ocupan estas dos estructuras? ¿Cuál es su diferencia?

```
info_celda t[SIZE];
```

26. cell_info *t[SIZE];
27. Una aplicación de gestión de fotografías en tu móvil tiene definido el catálogo de fotos de la siguiente forma:

```
#define SIZE_NAME

struct picture_info
{
  char name[SIZE_NAME];
  int date_time;
```

28. } pictures[SIZE];
29. ¿Qué tamaño tiene esta estructura de datos? La aplicación necesita crear una segunda tabla del mismo número de elementos, pero en lugar de tener los datos de las fotos quiere tener los apuntadors a los datos de las fotos. En otras palabras, es una tabla con idéntico número de elementos que la anterior, pero sus elementos no son estructuras sino apuntadores a las correspondientes estructuras de la tabla pictures. Escribe la declaración y el código para rellenar esa tabla.

30. Se dispone de la siguiente definición de datos:

```
#define SIZE 4
struct coordinates
{
   int latitude;
   int longitude;
} places[SIZE];

places[0].latitude  = 200;
places[0].longitude = 300;
places[1].latitude  = 400;
places[1].longitude = 100;
places[2].latitude  = 100;
places[2].longitude = 400;
places[3].latitude  = 300;
places[3].longitude = 200;
```

La tabla almacena cuatro puntos obtenidos del GPS de tu móvil, cada uno de ellos con su latitud y longitud que son números enteros. Hay una aplicación que ha obtenido estos puntos en este orden, pero necesita acceder a los datos de tres formas distintas. La primera es en el orden en que han sido obtenidos, y por tanto, tal y como está la tabla. El segundo es ordenados crecientemente por latitud y el tercero ordenados crecientemente por longitud. El acceso a estos datos en base a estos tres órdenes es continuo. Una primera solución podría ser reordenar los elementos de la tabla cada vez que se cambia de orden (por ejemplo, llega una petición de datos ordenados por latitud, pero la anterior se ordenaron por longitud, y entonces se reordena la tabla). Pero es extremadamente ineficiente. ¿Por qué? ¿Crees que utilizando apuntadores puedes ofrecer una alternativa más eficiente que evite reordenar los datos continuamente?

Realice un programa en C:

1. Que rellene un arreglo con los 100 primeros números enteros y los muestre en pantalla en orden ascendente.

2. Que rellene un arreglo con los 100 primeros números enteros y los muestre en pantalla en orden descendente.

EJEMPLO:

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

int main(void)
{
        int x,tabla[100];
```

```
        for (x=1;x<=100;x++)
        {
      tabla[x]=x;
   }

        for (x=100;x>=1;x--)
        {
      printf("%d\n",tabla[x]);
   }

   system("PAUSE");
   return 0;
}
```

3. Que rellene un arreglo con los números primos comprendidos entre 1 y 100 y los muestre en pantalla en orden ascendente.

4. Que rellene un arreglo con los números pares comprendidos entre 1 y 100 y los muestre en pantalla en orden ascendente.

5. Que rellene un arreglo con los números impares comprendidos entre 1 y 100 y los muestre en pantalla en orden ascendente.

6. Que lea 10 números por teclado, los almacene en un arreglo y muestre la suma, resta, multiplicación y división de todos.

7. Que lea 10 números por teclado, los almacene en un arreglo y los ordene de forma ascendente.

8. Que lea 10 números por teclado, 5 para un arreglo y 5 para otro arreglo distinto. Mostrar los 10 números en pantalla mediante un solo arreglo.

9. Que lea 5 números por teclado, los copie a otro arreglo multiplicados por 2 y muestre el segundo arreglo.

10. Que lea 5 números por teclado, los copie a otro arreglo multiplicados por 2 y los muestre todos ordenados usando un tercer arreglo.

11. Que rellene un arreglo con los 100 primeros números pares y muestre su suma.

12. Que lea 10 números por teclado, los almacene en un arreglo y muestre la media.

13. Que mediante un arreglo almacene números tanto positivos como negativos y los muestre ordenados.

14. Que rellene un arreglo con 20 números y luego busque un número concreto.