# ALGORITMOS Y ESTRUCTURAS DE DATOS

## Guía 1. Repaso General de Programación 1: Punteros a void, puntero a función.

**Para Repasar fuera de clase (ejercicios adicionales para hacer a discreción)**

**Mini-examen :**

1. ¿Cuál es la utilidad de los punteros a void?

2. Si queremos apuntar a un puntero a void, ¿por qué es necesario un cast?

3. ¿Qué información necesita el compilador para desreferenciar un puntero?

4. ¿Puede un casting incorrecto 'colgar' un programa? ¿Cómo y por qué?

5. ¿Por qué se puede desreferenciar sólo una vez un puntero de tipo void **? ¿Por qué no se puede desreferenciar dos veces (a menos que usemos casting)?

6. Para investigar: ¿qué es una tabla de saltos ("jump table")? ¿De qué tipo son los elementos de esta tabla? ¿En qué aplicaciones pueden ser útiles?

7. Para investigar: ¿cómo optimizan muchos compiladores una sentencia switch?

8. Para el siguiente código fuente:

```c
#include <stdio.h>
int main()
{
	int x;
	float y;
	unsigned short int array[] = {1, 2, 3};
	void *pV;
	unsigned char *pUC;
	int i;

	pV = &x;
	*((int *) pV) = 12;
	printf("pV as int:%d\n", *((int *) pV));

	pV = &y;
	*((float *) pV) = 123.24f;
	printf("pV as float:%f\n", *((float *) pV));

	pV = array;
	pUC = (unsigned char *) pV;
	for (i=0; i <sizeof(array) / sizeof(unsigned char); i++)
	{
		printf("pUC %d:%.2X\n", i, *pUC);
		pUC++;
	}
	return 0;
}
```

Se pide:

a. Crear un proyecto en C (en un archivo de extensión .c), compilar y ejecutar.

b. Eliminar el puntero pUC, para que su función sea reemplazada por pV, y comprobar el funcionamiento.

9. Para el siguiente código en C:

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

// Callback functions
void doFile()
{
	printf("File menu selected.\n");
}

void doEdit()
{
	printf("Edit menu selected.\n");
}

void doView()
{
	printf("View menu selected.\n");
}

// Type definitions
typedef void (*MenuCallback)();

void runMenu(char *title, char **label, MenuCallback *callback, int size)
{
	int isMenuRunning = 1;
	int i;
	unsigned int selectedItem;

	while(isMenuRunning)
	{
		// Print menu
		printf("%s\n", title);
		for (i= 0; i < size; i++)
			printf("%d: %s\n", i + 1, label[i]);

		// Get option
		char buffer[16];
		printf("> ");
		fgets(buffer, sizeof(buffer), stdin);

		// Call callback
		selectedItem = atoi(buffer) - 1;

		if (selectedItem > size)
			printf("Invalid entry.\n");
		else if (callback[selectedItem])
			callback[selectedItem]();
		else
			isMenuRunning = 0;
		printf("\n");
	}
}
```

```
int main()
{
	char *menuLabel[] = {"File",
	                     "Edit",
	                     "View",
	                     "Quit"};

	void (*menuCallback[])(void) ={ doFile,
	                                doEdit,
	                                doView,
	                                NULL};
	runMenu("Main Menu", menuLabel, menuCallback,
	          sizeof(menuCallback) / sizeof(MenuCallback));

	return 0;
}
```

Se pide analizar el código, crear un proyecto, compilar y ejecutar.

10. Explicar por qué el siguiente código en C no compila. Corregir el código.

```
#include <stdio.h>

int main(void)
{
	char a = 'h';
	void *p;

	p = &a;
	printf("a:%c\n", *p);
	return 0;
}
```

11. Suponiendo que trabajamos con una plataforma little-endian con enteros (int) de 4 bytes, ¿cuál es la salida a consola del siguiente programa? Explicar los resultados.

```
#include <stdio.h>
int main(void)
{
	int a = 0, b = 1, c = 2;
	void *p;

	a = 256;
	b = 257;
	c = 258;

	p = &a;
	ch1 = *((char *) p);
	p = &b;
	ch2 = *((char *) p);
	p = &c;
	ch3 = *((char *) p);
	printf("a:%c b:%c c:%c\n", ch1 + '0', ch2 + '0', ch3 + '0');
	return 0;
}
```

12. ¿Qué sucede si apuntamos a un puntero con un cast incorrecto? ¿Qué ocurre cuando se lee y escribe a través de ese puntero? Explicar el funcionamiento del siguiente código:

```c
#include <stdio.h>

int main(void)
{
	int a = 257;
	void *p;

	printf("a:%d\n", a);
	p = &a;
	*((char *) p) = 5;
	printf("a:%d\n", a);

	return 0;
}
```

13. Explicar por qué el siguiente código en C no compila. Corregir el código. Detallar en una tabla los valores de las variables a medida que ejecuta el programa.

```c
#include <stdio.h>

int main(void)
{
	int a = 5;
	void *p = &a;
	void **pp = &p;

	printf("pp:%p\n", pp);
	printf("*pp:%p\n", *pp);
	printf("**pp:%d", **pp);
	return 0;
}
```

14. Realizar una función que intercambie dos punteros de cualquier tipo (usando punteros a void).

15. Implementar una calculadora simple. El programa deberá solicitar dos operandos por consola:

Operando 1:
Operando 2:

Luego deberá mostrar un menú con operaciones matemáticas, y solicitar al usuario una de ellas:

(+) Sumar (-) Restar (*) Multiplicar (/) Dividir

Operación:

Finalmente deberá mostrar el resultado obtenido.

a. Empezar con un esquema del programa en papel: decidir qué módulos son necesarios, de qué manera se puede implementar la selección de las operaciones (¿quizás con un arreglo de punteros a funciones?), qué reciben y devuelven las funciones, qué variables son necesarias. Recordar el estatuto de prácticas: programar con simpleza, escribir código expresivo, dividir en módulos de interfaz gráfica y de operaciones, comentar el código, funciones simples y concisas con un único objetivo.

b. Implementar el programa. Algunos programadores crean el esqueleto del proyecto empezando por los comentarios de las funciones, y recién después escriben código.

c. Añadir las operaciones lógicas AND, OR y XOR al menú anterior.

d. ¿Qué ventaja tienen los arreglos de punteros a funciones frente a los bloques switch? Pista: ¿es posible implementar menús dinámicos con punteros a funciones (menús que cambian en función de cierta condición exterior)? ¿Es esto posible con bloques switch?

e. ¿Si quisiéramos crear sub-menús, ¿podríamos reutilizar la función que implementa un menú? ¿De qué manera?

**Para entregar (take home):**

16. Se pide diseñar una *librería* que procese los argumentos recibidos por línea de comando usando un *callback*. El diseño debe cumplir con las siguientes pautas:

    a. Debe poder procesar *opciones*. Una opción consiste de dos argumentos: el primero comienza con un guion ("-") y se denomina clave, y el segundo se denomina valor. Un ejemplo con dos opciones (la clave *maxclients* con valor *4*, y la clave *path* con valor *C:\WEB) es:* webserver -maxclients 4 -path C:\WEB

    b. Debe poder procesar *parámetros*, que es un argumento aislado que no comienza con guion. Un ejemplo con una opción y dos parámetros es:
    copyfile -isverify 1 archivo1.c archivo2.c

    c. El *parser* de la línea de comando debe cumplir con este prototipo:

```
typedef int (*pCallback) (char *, char*, void *);

int parseCmdLine(int argc, char *argv[], pCallback p, void
*userData);
```

    d. Para que el usuario de parseCmdLine() pueda usarla debe definir antes una función de callback. Esta función es llamada cada vez que parseCmdLine()

encuentra una opción o un parámetro. La función de callback deberá devolver 1 si la interpretación de las opciones o parámetros fuera correcta, y 0 para indicar que el procesamiento debe detenerse porque se encontró una opción o parámetro inválido. El prototipo es:

```
int parseCallback(char *key, char *value, void *userData);
```

e. El parser se ejecuta con la función parseCmdLine(). Cada vez que el parser encuentre una opción, deberá llamar al callback pasando en *key* el valor de la clave encontrada, y en *value* el valor encontrado. Cada vez que encuentre una parámetro, deberá pasar en *key* el valor NULL, y en *value* el valor del parámetro encontrado. Si el procesamiento de todas las opciones y parámetros fue exitoso y no se encontraron errores de forma (ver casos límites abajo), deberá devolver en su nombre la suma de la cantidad de opciones y parámetros encontrados. De lo contrario (-1).

f. Se recomienda usar typedef para los punteros a función.

g. Se debe realizar y entregar un banco de pruebas para probar el módulo a casos límite (¿qué es un caso límite?). Un ejemplo: ¿qué ocurre si enviamos una opción con clave pero sin valor? Lamparita: debemos pensar en todas las situaciones posibles y saber cómo tratarlas. Debatir en clase todos los posibles casos límites ¿Cómo se debe comportar parseCmdLine()) en dichos casos?