REPASO PRÁCTICO
SEGUNDO PARCIAL PROGRAMACIÓN I

Temas Evaluados

1. Multiarchivo.
2. Makefile.
3. Punteros Avanzados.
4. Tipos de Datos Avanzados.
5. Herramientas de Desarrollo.
6. Librerías.
7. CallBacks.
8. Recursividad.
9. Argumentos de Variables.
10. Software Testing.
11. Drivers Users Space.
12. Threads.

Autores

- Domínguez.
- Pugliese.

SEGUNDO PARCIAL - 2022 1C

Ejercicio 1: Escribir una función que realiza el cambio de 2 elementos de cualquier tipo (pero del mismo tipo entre sí). El prototipo debe ser *void gswap (void*p1 , void*p2 , unsigned int size)* donde p1 y p2 son punteros al primer y segundo elemento respectivamente y size es el tamaño del elemento en bytes.

```
void gswap ( void*p1 , void*p2 , unsigned int size )  // Prototipo, devuelve void y los punteros son los indicados.

void gswap ( void*p1 , void*p2 , unsigned int size )
{
      if ( p1==NULL || p2==NULL || size=0 )     // Chequeo que no haya error en los punteros o en el tamaño indicado.
      {
            return NULL;      // Acá se puede agregar un printf que avise al usuario en caso de error.
      }

      char aux;
      unsigned int i;

      for ( i=0 ; i<size ; i++ )    // Crea un for para intercambiar los elementos
      {
            aux = ( (char*) p1 )[i];      // Como tengo un puntero a void, tengo que desreferenciarlo a char para intercambiar
                                          // bit a bit, y realiza el cambio “i” veces ya que i es el tamaño del elemento.
            ( (char*) p1 )[i] = ( (char*) p2 )[i]
            ( (char*) p2 )[i] = aux;      // Realizo el cambio para no perder la información.
      }
}
```

Ejercicio 2: El siguiente programa utiliza una función de argumentos variables, que imprime el valor de los parámetros pudiendo especificar el tipo. Completar el código para que funcione de manera correcta.

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

void ShowVar ( char types[] , ... );  // Los 3 puntos van siempre

int main (void)
{
        ShowVar ("fcsd" , 32.4f , 'a' , "Testing String" , 4);
        return 0;
}

void ShowVar ( char types[] , ... )
{
        va_list v;              // Esto es para crear la lista de nombre v.
        int i;
        va_start(v, types);

        for ( i=0 ; types[i] != '\0' ; ++i )
        {
                switch ( types[i] )
                {
                        case 'd':
                                printf ("%d\n" , va_arg( v , int ) ); // En el va_arg tengo que indicar el nombre de la lista y el arg
                                break;
                        case 'f':
                                printf ("%f\n" , va_arg( v , double) );      // En el va_arg tengo que indicar el nombre de la lista y el arg
                                break;
                        case 'c':
                                printf ("%c\n" , va_arg( v , int) );  // En el va_arg tengo que indicar el nombre de la lista y el arg
                                break;
                        case 's':
                                printf ("%s\n" , va_arg( v , char*) );  // En el va_arg tengo que indicar el nombre de la lista y el arg
                                break;
                }
        }
        va_end(v);
}
```

Ejercicio 3: Sea una estructura para implementar una lista dinámica de elementos, cada elemento llamado nodo almacena un valor numérico y un puntero al próximo nodo, con el último nodo apuntando a NULL. Escribir la declaración de la estructura del nodo y colocarle el alias nodo_t. La estructura posee 2 campos: value con el valor numérico de tipo double y pNext con el puntero tipo puntero a struct nodo.

FORMA NÚMERO 1

```
typedef struct NODO_TAG{ // TAG para poder ref
   double value;
   struct NODO_TAG * pnext; // Pun a la struct
} nodo_t;
```

FORMA NÚMERO 2

```
typedef struct NODO_TAG nodo_t;
struct NODO_TAG{
    double value;
    struct NODO_TAG * pnext;
};
```

FORMA NÚMERO 3

```
typedef struct NODO_TAG nodo_t;
struct NODO_TAG{
   double value;
   nodo_t * pnext;
};
```

EJERCICIO 5: Sea un programa multiarchivo compuesto por file1.c y file2.c Dentro de file2.c y antes de las funciones (es decir, fuera de todas ellas) se encuentra definida una variable mediante la línea " static int magic; ". ¿Existe alguna forma de modificar el valor de magic desde una función escrita en file2.c? Justificar su rta.

file1.c	file2.c

static int magic;	void fun2a (void)
	{
	magic = 5;
	}

---

file1.c	file2.c	
int magic;	extern magic;	// Las modificaciones son por un lado quitar el static de para que pueda ser
	void fun2a (void)	// visible por el resto de archivos, y luego colocar el prototipo en file2

	{
	magic = 5;
	}	

Ejercicio 4: Considerando la estructura del ejercicio anterior, escribir la función *nodo_t * listAddLast (nodo_t * list , double nVal)* Que crea un nuevo nodo en el heap y lo agrega al final de la lista, donde list es el puntero al primer nodo (list vale NULL en una lista vacía) y nVal es el valor numérico del nuevo nodo La función devuelve un puntero al primer nodo o NULL si encontró algún problema.

```
#include <stdlib.h>
nodo_t * listAddLast ( nodo_t* list , double nVal )
{
   nodo_t* pNodo = malloc (sizeof (nodo_t) );      // Crea el puntero pNodo de tipo de dato nodo_t y reserva el nodo en memoria
   if (pNodo == NULL)   // Válido si falló malloc
   {
       return NULL;
   }
   pNodo->value = nVal;       // Accedo al campo value de pNodo, y almaceno el valor
   pNodo->pNext = NULL;       // Accedo al campo pNext y almaceno NULL

   if (list == NULL)   // List es el punt al primer nodo, si es NULL la lista estaba vacía
   {
       list = pNodo;    // Lista apunta a pNodo (list debe apuntar al primer nodo)
   }

   else     // De no ser así, es decir que la lista no sea vacía
   {
       nodo_t * plast = list;       // Creo el puntero pLast y lo iguala a list

       while (pLast->pNext != NULL) // Accedo a pNext, mientras que sea distinto a NULL, no estoy en el último, entonces avanzo
       {
           pLast = pLast->pNext;    // Avanzo de nodo
       }
       pLast->pNext = pNodo;  // Si ya estoy en el final, hago que el último de la lista apunte al recién creado.
   }
   return list;
}
```

RECUPERATORIO SEGUNDO PARCIAL - 2021 2C

Ejercicio 1: Sea un programa para simular el Juego de la Vida (Game of Life) de John Conway. Se pide escribir la función que cree e inicialice un mundo. La función recibe el alto y ancho del mundo a crear, lo crea, lo llena aleatoriamente con valores DEAD y ALIVE (probabilidad 50-50) y devuelve un puntero al mundo creado para poder utilizar en el programa. En caso que haya algún problema debe devolver NULL.

```
#include <stdio.h>
#include <stdlib.h> // Librería para el uso de malloc
#include <time.h> // Librería para el uso de valores random

int* mundo_init (int height, int width); // Prototipo de la función
srand (time(NULL) ); // Genera la semilla para obtener valores aleatorios (Al inicio del programa)

int* mundo_init( int height , int width)
{
	if ( height < 1 || width < 1 )
	{
		return NULL; // En el caso de solicitar una matriz nula o negativa devuelve null, se puede avisar con printf al usuario.
	}
	int * inicio; // Puntero para devolver al terminar
	int * point = malloc (height * width * sizeof(int)); // Reservo en el heap el lo que ocupará la matriz
	if ( point == NULL)
	{
		return point; //en el caso de que falle malloc, me devolverá un puntero a NULL, por lo que point valdrá NULL y evitaré manejarme con este puntero
	}

	int contador_height, contador_width; // Contadores para recorrer la matriz
	inicio = point; // Guardo en inicio la posición del puntero para devolverlo

	for(contador_height = 0 ; contador_height < height ; contador_height++)
	{
		for(contador_width = 0 ; contador_width < width ; contador_width++)
		{
			char random = rand()%2; // rand() devuelve valor aleatorio y el % 2 solo puede ser 1 (n° imp.) o 0,(n° par.)
			*point = random;	// Almaceno el valor aleatorio en la posición ubicada
			point++;		// Voy a la siguiente posición
		}
	}
	return inicio;
}
```

Ejercicio 2: Sea un programa para trabajar con estadísticas de futbolistas. Para gestionar la información, el programa almacena la siguiente información de un futbolista: 1. Nombre y apellido. 2. Clubes en los jugó y rango de fechas en cada uno (año inicial y final). Se pide:
a. Declarar dicha estructura y asignar el alias futbolista_t.
b. Escribir la función futJugaronJuntxs, que recibe la información de dos futbolistas (mediante punteros a futbolista_t) y devuelve si alguna vez jugaron juntxs en un mismo equipo. Nota: para permitir que futbolistas que hayan jugado en distinta cantidad de clubes se debe utilizar un arreglo de largo fijo NMAX (constante previamente definida) y especificar un "terminador" que indique el fin de la lista.

```
#include<stdio.h>
#define NMAX 10 //número máximo de clubes en los que habrá alguna vez jugado un futbolista

typedef struct
{
        char nombre[30];          // Nombre del futbolista.
        char* club [NMAX];        // El terminador de clubes es el puntero a NULL
        int fechainicio[NMAX];    // En el caso de que tenga más de un equipo, el puntero al club número
        int fecharetiro[NMAX];    // Las Fechas de inicio y de retiro están en AÑOS, en formato YYYY
}futbolista_t;

int futJugaronJuntxs(futbolista_t* pl1, futbolista_t* pl2); //devuelve un 1 si jugaron juntos
int strcomp(char*p1,char*p2);                               //compara 2 strings y devuelve un 1 si son idénticos, devuelve un 0 si no lo son

int futJugaronJuntxs(futbolista_t* pl1, futbolista_t* pl2)
{
        int contp1,contp2;        // crea un contador que me permitirá ir entre los clubes
        futbolista_t* aux;        // este puntero auxiliar me permitirá comparar entre ambos jugadores
        int estado = 0;           // esta variable la utilizaré para indicar que tienen un club en común pero quizás no hayan jugado en las mismas fechas, será igual a
                                  // 1 si tiene un club en común ó un 0 sinó
        for(contp1 = 0 ; contp1 < NMAX && pl1->club[contp1]!= NULL ; contp1++)
        {
                for(contp2 = 0 ; contp2 < NMAX && pl2->club[contp2] != NULL ; contp2++)
                {
                        estado = strcomp(pl1->club[contp1] , pl2->club[contp2]);

                        if (estado == 1) // si el estado es igual a 1, entonces hay un club en común

                        {
                                if(( pl1->fechainicio[contp1] <= pl2->fechainicio[contp2]) && (pl1->fecharetiro[contp1] >= pl2->fechainicio[contp2]))
                                {
                                        return 1;
                                }
                                else if (( pl1->fechainicio[contp1] >= pl2->fechainicio[contp2]) && (pl1->fecharetiro[contp1] <= pl2->fecharetiro[contp2] ))
                                {
                                        return 1;
                                }
                        }
                }
        }
        return 0;
}
```

```
int strcomp(char*p1,char*p2)
{
        int contador,size1,size2;         // creo un contador y una variable llamada size1 y size 2 que permitan ver el tamaño de ambos strings
        char*paux;                        // creo un puntero auxiliar que me permitirá contar el tamaño de un strng
        paux = p1;

        for(size1 = 0 ; *paux != '\0' ; )
        {
                paux++;
                size1++;
        }

        paux = p2;

        for(size2 = 0 ; *paux != '\0' ; )
        {
                paux++;
                size2++;
        }

        if(size1 != size2)
        {
                return 0;
        }

        for(contador = 0 ; *pl1 == *pl2 ; contador++)
        {
                pl1++;   //Estas dos líneas podrían estar junto a contador++
                pl2++;

                if(contador == size1)
                {
                        return 1;
                }
        }
        return 0;
}
```

Ejercicio 3: Un programa multiarchivo tiene 2 archivos fuentes (file1.c y file2.c). Al comienzo de file1.c, fuera de toda función, tiene textualmente la misma expresión. Hay 3 casos para dicha expresión:
a) static int var;
b) extern int var;
c) const int var;

a) static permite guardar la variable de forma estática y privada. Por lo que si estuviese dentro de una función, cada vez que sea llamada la función su valor se guardaría. Privada ya que si file2.c intenta modificarla, arrojará un error de linkedición. Se guarda en memoria dinámica.
b) extern permite "avisar" al compilador que esta variable está definida desde otro archivo y permite acceder al valor, por lo que, sí está definida como "int var;" en file2.c, será una variable pública y no arrojará error de linkedición, a menos que esta no se encuentre allí o que por error este escrito "int vaar;" en ese caso si será error de linkedicion
c) const permite evitar la modificación de esa variable, pero si permite acceder a su valor, siendo esta una variable pública, por lo que file2.c si puede acceder a su valor. Será error de ejecución si se intenta hacer un "var++;"

Ejercicio 4:
a) Para programar con lenguaje C se tiene librerías estándares y no estándares. Y en la cursada se mencionó que, de ser posible, conviene utilizar las estándares. Explicar porqué.
b) Algunos algoritmos pueden resolverse mediante funciones iterativas y recursivas. Explicar 2 ventajas y 2 desventajas de cada una.
c) Para poder almacenar el código fuente se explicó el uso de la herramienta GitHub. Explicar 2 ventajas con respecto a los servicios de alojamiento de archivos, como gDrive y Dropbox.

a) Las ventajas de las librerías estándares son que tenemos garantías de que van a funcionar, sabiendo sus capacidades y limitaciones, en internet hay muchas páginas que explican su funcionamiento, están incluídas en los sistemas operativos por lo que no es necesario descargarlas, sabemos que funcionan para distintas arquitecturas de procesador, además su prototipo es estándar.
b) La desventaja de usar funciones recursivas es que al ser una función va llenando el stack por los llamados que hace, es decir es más lento, además que suele ser más difícil de encontrar la forma recursiva de encontrar la manera recursiva de hacer las cosas. Las ventajas de la recursividad es que es un código más compacto/pequeño, y por otro lado permite resolver problemas cuyo algoritmo de forma iterativa sería extremadamente complejo/imposible.
c) GitHub permite ver el historial de archivos y modificaciones realizadas a través del tiempo, además de ver las diferencias en el código en una plataforma amigable

SEGUNDO PARCIAL - 2021 2C

Ejercicio 1: Escribir un programa que ordene los argumentos recibidos por línea de comando, tomando como criterio la cantidad de caracteres que tiene (de menos a más), y los imprima en consola. Se debe utilizar la función de la librería estandar qsort.
Por ejemplo, si el programa (llamado ordeno) es ejecutado con el comando:
ubuntu:~$ ./ordeno hola Paralelo 12345 N El programa debe imprimir: N hola 12345 Paralelo

---

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

typedef struct
{
        int tamaño;
        char *arreglo;

}estructura_t;

int length (char *);
int comparar (const void *pa, const void *pb);

int main(int argc, char *argv[])
{
        int largo[20];
        estructura_t* mat_estructuras = calloc((argc-1), sizeof(estructura_t));
        if(argc == 1)
                printf("No se dieron argumentos\n");
        else
        {
                for(int i = 0; i < (argc-1); ++i)
                {
                        largo[i] = length(argv[i+1]);
                        mat_estructuras[i].tamaño = largo[i];
                        mat_estructuras[i].arreglo = argv[i+1];

                }
                qsort(mat_estructuras, (argc-1), sizeof(estructura_t), comparar);
                for(int j = 0; j < (argc-1); ++j)
                        printf("%s\n",mat_estructuras[j].arreglo);
        }
        free(mat_estructuras);
        return 0;
}
```

```c
int length (char *array)
{
        int contador = 0;
        while(array[contador] != '\0')
        {
                contador++;

        }
        return contador;
}

int comparar (const void *pa, const void *pb)
{
        const estructura_t* p1 = pa;
        const estructura_t* p2 = pb;

        return (p1 -> tamaño)-(p2 -> tamaño);
}
```

EJERCICIO 2: Sea un módulo para hacer operaciones con los bits de los bytes.
Se pide: a) Escribir el alias byte_t, que permite acceder mediante campos a un byte completo, al nibble alto, al nibble bajo y a cada uno de los bits.
b) Escribir una función que intercambie el nibble alto y bajo de un byte. Su prototipo debe ser void swap_nibbles(byte_t * byte).

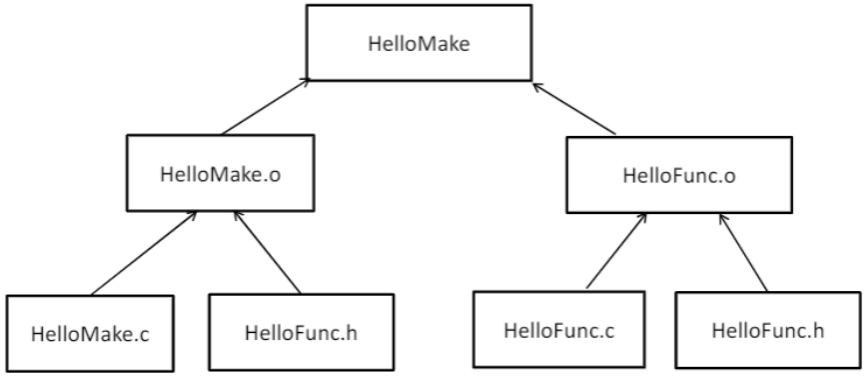
```c
#include <stdio.h>
#include <stdint.h> // Indico

typedef struct NIBBLES_T   // Crea la estructura
{
	uint8_t lo : 4;     // lo son los primeros 4
	uint8_t hi : 4;     // hi son los últimos 4
}nibbles_t;

typedef struct BITS_T      // Crea la estructura
{
	uint8_t B0 : 1;     // Indico a cada bit
	uint8_t B1 : 1;     // del byte por dif
	uint8_t B2 : 1;
	uint8_t B3 : 1;
	uint8_t B4 : 1;
	uint8_t B5 : 1;
	uint8_t B6 : 1;
	uint8_t B7 : 1;
}bits_t;

typedef union BYTE_T       // Crea la unión
{
    uint8_t byte;          // Indica que va a ser un byte
    bits_t bit;            // Va a estar compuesta de un bit_t
    nibbles_t nibble;      // y un nibble_t
}byte_t;
```

```c
void swap_nibbles( byte_t * byte ); // recibe un ptr tipo byte_t

int main (void)
{
	byte_t b1 = {.byte = 0x15};

	printf("%1x\n", b1.nibble.hi);
	printf("%1x\n", b1.nibble.lo);

	swap_nibbles(&b1);

	printf("%1x\n", b1.nibble.hi);
	printf("%1x\n", b1.nibble.lo);

	return 0;
}

void swap_nibbles( byte_t * byte)
{
    uint8_t temp;    // Creo la variable temp
    temp = byte->nibble.lo; // guardo en temp el nibble lo
    byte->nibble.lo = byte->nibble.hi; // guardo en lo el nibble hi
    byte->nibble.hi = temp; // guardo en hi el nibble lo
}
```

Ejercicio 3:

a) Explicar detalladamente las etapas de compilación y linkear: qué realiza cada una, cuáles son sus diferencias y cuáles sus archivos de entrada y salida. Dar un ejemplo concreto de error de compilación y otro de linkedición.
b) Explicar qué es el "árbol de dependencias" en makefile.

a) La compilación convierte archivos .c y .h en ejecutables, su salida es código assembler, es decir archivos .s. La linkedición une varios archivos para crear un ejecutable,recibe códigos objeto y genera ejecutable .exe.
b) Élárbol de dependencias es un diagrama en el cual se muestra los archivos que este genera, estando el ejecutable encabezando el mismo, y continuado por sus .o de los cuales se indica los .c y .h que lo componen.



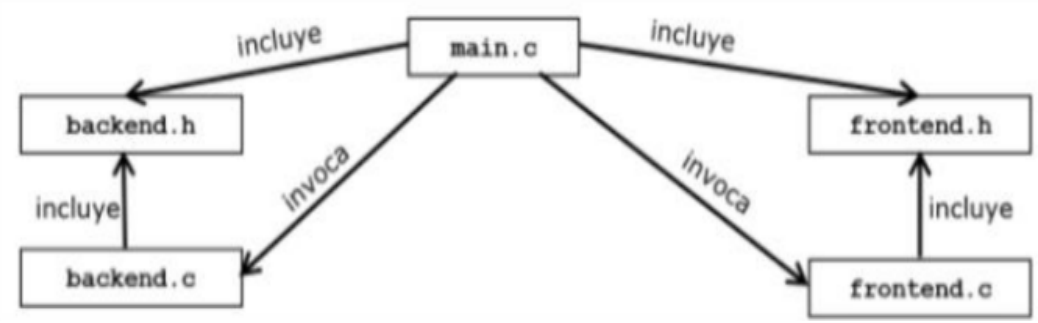
Árbol de dependencias

Ejercicio 4: Suponer que se está trabajando en un microcontrolador MSP430 (arquitectura de 16 bits, little-endian) y que la variable mat está ubicada comenzando en la dirección 0x2000. Para el siguiente código se pide:
a) Completar las expresiones para que realicen lo indicado en su comentario.
b) Indicar que imprime el siguiente programa. Justificar brevemente la respuesta.

```
#include <stdio.h>
int main (void)
{
	int mat[3][2] = {{1, 2}, {3, 4}, {5, 6}};
	char * ptr;

	// PARTE a)
	ptr = (char*) & mat [2][1] ; // utilizando mat, hacer que ptr apunte al último byte de mat
	* (( ptr = (char*) & (mat[0][0]) )) = -4; // utilizando ptr, se le asigna -4 al primer elemento de mat

	// PARTE b)
	printf("%p\n", mat+1);		// Apunta al siguiente elemento, es decir al 3, imprime 0x2004
	printf("%p\n", mat[2]-1);	// Apunta al 4, imprime 0x2006
	printf("%p\n", &mat+1);		// Apunta al primer byte fuera de la matriz, imprime 0x200C
	printf("%p\n", *mat+1);		// Apunta al 2, imprime 0x2002

	return 0;
}
```

RECUPERATORIO SEGUNDO PARCIAL - 2021 1C

Ejercicio 1: Sea el siguiente árbol de dependencias de un programa multiarchivo, se pide escribir el makefile de manera que genera los archivos objetos intermedios (.o) y el ejecutable tpf. Además escribir el comando para ejecutar el makefile suponiendo que el archivo se llama makefileTPF.mak



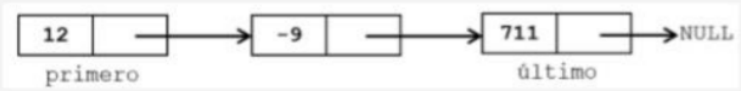

```
tpf: main.o frontend.o backend.o
	gcc main.o frontend.o backend.o -o tpf
main.o: main.c backend.h frontend.h
	gcc main.c -c
backend.o: backend.c backend.h
	gcc backend.c -c
frontend.o: frontend.c frontend.h
	gcc frontend.c -c


makefile -f makefileTPF.mak
```

Ejercicio 2: Explicar detalladamente dos de los distintos modos de acceso a la función fopen. Para cada caso indicar si el archivo no existe, que ocurre con el archivo y que devuelve la función, si el archivo no existe, que ocurre con el archivo.

Una de las formas de acceso del fopen es el modo de solo lectura. Este modo solo permite leer el documento, no se lo puede editar, si el archivo no existe devuelve un puntero a Null. Si el archivo existe entonces se puede leer y pasa un puntero al primer elemento del documento, PERO siempre hay que usar fclose para cerrarlo.
Otra forma de acceso de fopen es el modo escritura el cual permite editar el documento devolviendo un puntero al primer elemento, si el archivo no existe fopen crea uno vacío.

Ejercicio 3

Sea una estructura para implementar una lista dinámica de elementos. Cada elemento, llamado nodo, tiene un valor y un puntero al próximo nodo. El último nodo apunta a `NULL`.
Un ejemplo de una lista de 3 nodos sería:



a) Escribir la declaración de la estructura del nodo y colocarle el alias `nodo_t`. La estructura posee 2 campos: campo `data` (tipo `int`) y campo `pNext` (tipo puntero a la estructura nodo).
b) Escribir la función `int listAdd(nodo_t* first, int newData)` que agrega un nodo al final de la lista; donde `first` es un puntero al primer nodo (en una lista vacía first vale `NULL`) y `newData` es el nuevo dato a agregar. Debe devolver si pudo agregar el nodo a la lista correctamente.

```c
#include <stdio.h>
#include <stdlib.h>  // Incluye la librería para malloc y NULL

typedef struct NODO  // Crea la estructura autorreferenciada
{
	int data;     // Campo Data de tipo int
	struct NODO* pNext;  // Campo pNext que es de tipo puntero a la estructura misma
} nodo_t;

int listAdd (nodo_t* first, int newData); // Prototipo, devuelve un int, y recibe un puntero al primer nodo y el int a agregar

int listAdd(nodo_t* first, int newData)
{
	if (first->pNext == NULL)    // En el caso de estar en el último nodo (pNext apuntando a NULL) crea el próximo
	{
		nodo_t * pNewItem = malloc(sizeof(nodo_t));      // Con el puntero pNewItem guarda el nodo en el heap

		if (pNewItem  == NULL)      // Si falló el malloc devuelve NULL, además podría avisarse por consola
		{
			return 0;
		}

		(*pNewItem).data = newData; // Desreferencia el puntero, accede al campo data, y almacena el valor indicado
		(*pNewItem).pNext = NULL;          // Idem, accede al campo pNext, y almacena NULL
		first->pNext = &(*pNewItem);// Indica que el campo pNext de first apunte al nodo recientemente creado
		return 1;                          // Devuelve un 1 si pudo crear el nodo correctamente
	}
	return listAdd(first + 1, newData);// En el caso de no estar en el último nodo, vuelve a chequear pero en el nodo siguiente
}
```

Ejercicio 4: Completar la siguiente línea de código indicada para que el programa imprima 34, sin que haya problemas de compilación, suponer que se trabaja en ARM (32 bits little endian)

```
#include <stdio.h>

int main (void)
{
    int var = 0x12345678;
    char * pbyte;

    pbyte = (char*)&var +2;   // Lo castea a tipo puntero a char, y luego le asigna lo que está en esa posición de memoria.

    printf ("%X\n", *pbyte);
    return 0;
}
```

Ejercicio 5: Idéntico a ejercicio de Argumentos Variables en el 2P 1C 2022.

SEGUNDO PARCIAL - 2021 1C

EJERCICIO 1: Se le solicita a un alumno de Programación 1 que escriba la función gswap que realiza el intercambio de dos elementos de cualquier tipo (pero del mismo tipo entre sí). El prototipo es "void gswap (void* p1, void* p2, int size)" donde p1 y p2 son punteros al 1er y 2do elemento respectivamente y size es el tamaño del elemento en bytes. Mencione errores y re-escriba.

```
void gswap (void* p1, void* p2, int size)
{
	char *aux;
	int i;
	for(pA=p1, pB=p2, i=0; i<=size; i++)
	{
		*aux = *p1;
		*p1 = *p2;
		*p2 = *aux;
	}
}
```

RTA Errores: nunca crea a pA y a pB, como es un puntero aux nunca guarda el valor al que apunta p1 sino que simplemente aux apunta a la misma dirección de memoria que el otro puntero. Lo mismo pasa en el resto de las líneas, estas nunca guardan el contenido sino que simplemente cambian de dirección a la que apuntan, es por esto que la función no cumple con lo pedido

```
#include <stdio.h>
void gswap (void *p1, void *p2, int size); //prototipo
```

Sigue abajo la función

```
void gswap (void *p1, void *p2, int size)
{
	switch (size) // Size es un int el cual indica el tamaño de los elementos, por eso hago un switch
	{
		case 1: // Si size vale 1, es porque se trata de un char, hago el gswap casteando los punteros
		{
			char aux = *(char *)p1;    // Almacena en el chau auxiliar lo ubicado en la pos a la que apunta p1
			*(char *)p1 = *(char *)p2; // Almacena en p1 lo ubicado en p2
			*(char *)p2 = aux;         // Almacena en p2 lo que previamente estaba en p1
			break;
		}

		case 4: // Si size vale 4, intercambio de a ints
		{
			int aux = *(int *)p1;
			*(int *)p1 = *(int *)p2;
			*(int *)p2 = aux;
			break;
		}

		case 8: // Si vale 8, de a doubles
		{
			double aux = *(double*)p1;
			*(double*)p1 = *(double*)p2;
			*(double*)p2 = aux;
			break;
		}

		default: // Por último, si no es ninguno de los casos mencionados anteriormente es porque hubo un error
		{
			printf ("ERROR - tamaño de dato ingresado no es válido");
			break;
		}
	}
}
```

EJERCICIO 2: Sea un driver para la utilización del mouse con lenguaje C, el cual contiene 3 funciones:

1. mouse_init (): la cual inicializa el mouse, indicando si lo pudo hacer correctamente.
2. mouse_readStatus (): lee la posición del mouse en coordenadas y si hay algún botón presionado.
3. mouse registerCallback (): registra la función callback a llamar cada vez que se realiza un clic, indicando si lo pudo hacer correctamente. Dicha función callback recibe las coordenadas donde se hizo el click y si fue izquierdo o derecho.

Se pide escribir mouse.h considerando las buenas prácticas y eligiendo correctamente los argumentos de los prototipos y crear un alias al tipo ptr a callback.

```
#ifndef MOUSE_H// Guarda
#define MOUSE_H


int mouse_init (void)  // Devuelve un 0 o 1 dependiendo de si funcionó correctamente, no recibe nada
void mouse_readStatus (int x , int y , char click)   // No devuelve nada, recibe las coordenadas y que tipo de click fue
int mouse registerCallback ( void (*p2f) int x , int y , char click ) // Devuelve 0 o 1, recibe un puntero a fun, función la cual lee el mouse

typedef void (*p2f)(int , int , char) // alias al tipo ptr a callback

#endif // MOUSE_H
```

EJERCICIO 3: En el lenguaje C se puede crear una matriz de cantidad variable y seguir utilizando la notación mat[i][j], para esto se crea dinámicamente los arreglos fila_i y otro arreglo que apunte al primer elemento de cada fila. Escribir la función dinmat que crea dinámicamente la matriz, recibiendo el tamaño de la misma y devolviendo un puntero a la matriz creada.

```
include <stdlib.h>
double** dinmat ( int n, int m)
{ //acá se podría meter una validación de los int m y n
        double ** mat malloc ( n*sizeof (double* ) );
        if (mat != NULL)
        {
                int i;
                for (i=0 ; i<n ; ++i )
                {
                        mat[i] = malloc (m*sizeof(double));
                        if (mat[i] == NULL)
                        {
                                for (i ; i>=0 ; i--)
                                {
                                        free (mat[i]);
                                }
                                free(mat);
                                return NULL;
                        }
                }
        }
        return mat;
}
```

EJERCICIO 4: Se desea escribir una función que reciba un arreglo de n notas y devuelva la nota máxima, mínima, promedio y mediana, explicar detalladamente 2 métodos distintos para poder devolver los 4 valores calculados, no se permiten ni variables globales ni imprimir el resultado, escribir el prototipo de la función en ambos casos

1. Una forma de devolver el resultado puede ser almacenar los datos en la memoria dinámica, y devolver un puntero al primer elemento de estos datos, para que el usuario pueda acceder libremente. El prototipo sería *double fun (double notas[])

2. Otra forma sería crear un struct con 4 campos donde cada uno de ellos es la nota solicitada, y la función devuelve la estructura en cuestión. Siendo su prototipo notas_t fun (double notas[])

EJERCICIO 5: Indicar que imprime en consola la siguiente función si se invoca con 7. Justificar la respuesta

```
void fun(int n)
{
	if (n < 3)
	{
		putchar('0' + n);
	}

	else
	{
		fun(n-1);
		fun(n-3);
	}
}
```

RTA: QOPQQOQOP

Explicación: La función es llamada con 7, como es mayor que 3 hago f6, idem f5, idem f4, idem f3, estoy en f2, imprime **Q**. En f3 además de estar en f2 estoy en f0, imprime **O**, luego en f4 tambíen inicia f1, imprime **P**, ahora paso a f5, donde por un lado ya imprime f4, ahora estoy en f2, imprime **Q**, paso a f6, ya hice el ciclo de f5 ahora hago el de f3, el cual imprime **QO**, y por último paso a f7, donde me queda únicamente f4 que imprime **QOP**

SEGUNDO PARCIAL - 2020 2C

EJERCICIO 1: Escribir una función bsort, que cumple la misma función que qsort (ordenamiento de un arreglo de elementos genéricos con criterio genérico) pero usando burbujeo como algoritmo de ordenamiento.
QSORT: void qsort (void* base, size_t num, size_t size, int (*comparar)(const void*, cont void*));

```
#include <stdio.h>
#include <stdlib.h>
void bsort (void* base, size_t num, size_t size, int (*compar)(const void*, const void*)); // Prototipo de la función

void bsort (void* base, size_t num, size_t size, int (*compar)(const void*, const void*))
{
	void *pEscritura = (char*)base;
	int i, j, k;
	int accion; // Variable que va a guardar lo que devuelve la función "comparar".
	char temp;
	for(i = 0; i <= num; i++) // Recorre todos los números de adelante para atrás.
	{
		for(j = 0; j < (num-1-i); j++) // Para pasar por todos los numeros de atras para adelante
		{
			accion = compar(pEscritura[j*size],pEscritura[(j+1) * size]); // Compara dos elementos y ve cual es más grande
			if(accion > 0) // Solo en este caso se deben intercambiar de posición los elementos
			{
				for(k = 0; k < size; k++) // Cambia byte por byte
				{
					temp = pEscritura[j*size + k];
					pEscritura[j * size + k] = pEscritura[(j+1) * size + k];
					pEscritura[(j+1) * size + k] = temp;
				}
			}
		}
	}
}
```

EJERCICIO 2: Sea un módulo de software llamado displayu_rgb para manejar un display matricial a color (similar a una pantalla de computadora, con píxeles a color). El mismo tiene las funciones públicas:

```
void dispInit(??);      // Inicia el display
void dispClear(??);     // Borra el buffer
void dispWrite(??);     // Configura un pixel del buffer con un color
void dispFlip(??);      // Envía el buffer al display
```

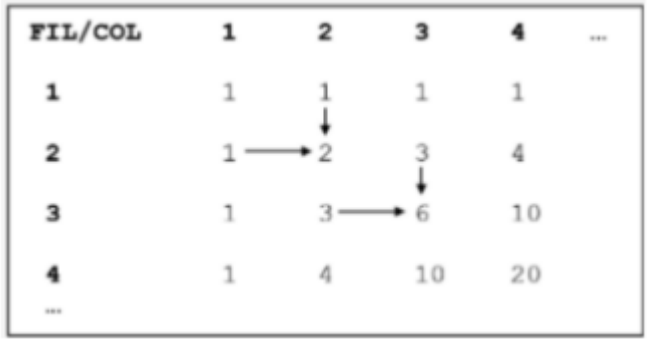
Escribir el archivo header del módulo (display_rgb.h) completando los prototipos de las funciones y considerando las buenas prácticas de multiarchivo. No es necesario escribir disp

RTA

```
#ifndef _DISPLAY_RGB_H_             // Para que no se vuelva a incluir en caso de incluirlo en más de un archivo
#define _DISPLAY_RGB_H_

#define ANCHO 255
#define ALTO 255
#define MAX_COLOR 255

void dispInit(char[][] display);    // Inicia el display, recibiendo una matriz de char[ALTO][ANCHO]
void dispClear(char[][] display);   // Borra el buffer
void dispWrite(int eje_x, int eje_y, char color); // Configura un pixel del buffer con un color (ejes = coordenada a escribir)
void dispFlip(char[][] display);    // Envía el buffer al display

#endif //_DISPLAY_RGB_H_
```

EJERCICIO 3: Escribir de forma recursiva una función que devuelva el elemento (i, j) de una matriz tartaglia.

- La función recibe dos parámetros: fila i y columna j del elemento. Comienza en 1 y no tiene límite máximo.
- La función debe devolver el valor del elemento (i, j) de una matriz tartaglia, o 0 si los índices son inválidos.
- La matriz de tartaglia contiene un número 1 en la 1ra fila y columna. Los elementos restantes se obtienen de sumar 2 elementos de la fila y columna anterior.




```
int matTartaglia (int i, int j);

int matTartaglia (int i, int j)
{
	if (i <= 0 || j <= 0)
	{
		printf("ERROR - índice de la matriz inválido");
		return 0;
	}

	else if (i == 1 || j == 1) // Caso base
	{
		return 1;
	}

	else
	{
		return (matTartaglia(i-1,j) + matTartaglia(i, j-1)); // Suma de los 2 elementos de la fila y columna anterior.
	}
}
```

EJERCICIO 4: Explicar similitudes y diferencias, ventajas y desventajas entre crear una variable en stack o en el heap. Dar un ejemplo concreto de donde sea conveniente cada uno.

RTA

La primera diferencia que podemos encontrar en cuanto al lugar de creación de una variable es la persistencia y la visibilidad de la misma. Si la variable es creada en el heap, está es visible y modificable desde cualquier línea del código hasta que se realice la función "free" con la variable. Por otro lado, si esta se crea en el stack, la visibilidad de la misma es local, solo tiene visibilidad y persistencia en el bloque o la función en la que se encuentra.

Las principales similitudes que encontramos entre crear variables en el heap vs en el stack son que ambos son utilizados para almacenar variables, constituyendo un segmento de la memoria. Por otro lado ambos cambian durante la ejecución del programa y existen en la RAM.

Algunas desventajas del uso del heap es que el programador debe reservar el espacio de memoria en el que las variables pueden ser almacenadas por lo que él mismo es limitado. Esto no pasa con el stack debido a que las variables creadas en el stack se almacenan en la memoria RAM.

Como fue mencionado, para utilizar el heap se le debe indicar cuánto espacio de memoria reservar y este se debe liberar una vez se termine de usar, de lo contrario este quedará reservado hasta que se apague la computadora, para evitar los “Memory leaks”. Esto no pasa con el stack ya que esto es automático.

Conviene crear una variable en el heap si se desea que esta sea global a nivel archivo pero que no persista durante toda la ejecución, de esta manera, en cualquier momento se puede hacer un "free" de la misma y esta dejará de existir. Por otra parte, es conveniente crear una variable en el stack en la mayoría de las ocasiones ya que C trabaja de forma eficiente y controlada con su persistencia. Un ejemplo de esto son los "for loops".

EJERCICIO 5: Sea un programa que trabaja con árboles genealógicos. Para gestionar la información de una persona, el programa almacena la siguiente información por persona:

- Nombre y apellido de la persona (string)
- Fecha de nacimiento (arreglo de enteros formato [dd,mm,aaaa])
- Madres/padres (arreglo de punteros a personas)
- Hermanos/as (arreglos de punteros a personas)
- Hijos/as (arreglos de punteros a personas)

Se pide:
a. Declarar dicha estructura y asignarle el alias persona_t
b. Escribir la función geniCantSobris que recibe la información de una persona (mediante un puntero a persona_t) y devuelve la cantidad de sobrinas/os que tiene esa persona.

SEGUNDO PARCIAL - 2020 1C

EJERCICIO 1: Explicar el uso de las palabras reservadas static y const en el siguiente ejemplo ¿Qué cambiaría si no estuvieran?

```
static char str[] = "COVID-19";
char* const p = str+5;

int main ()
{
.....
}
```

RTA

STATIC: Para definir un arreglo de char. Esto hace que el arreglo global sea visible únicamente para este archivo. Al estar esta palabra si se quiere usar este mismo arreglo en otro archivo sale error. Si no estuviese la palabra static se podría usar desde otro archivo utilizando la palabra extern.

CONST: No se puede modificar el lugar al que apunta, es un puntero constante que apunta a un char. Si se puede modificar el contenido de lo que está apuntando.
Sin esta palabra se podría modificar la dirección de memoria a la cual apunta el puntero p, se podría mover a través del puntero por el arreglo de char.

EJERCICIO 2: Escribir una función que cree una matriz identidad de nxn double. Prototipo: double* eye (unsigned int n), debe crear la matriz en el heap, completarla con los valores de la matriz identidad y devolver un puntero al primer elemento de la matriz. En caso de haber un problema debe devolver NULL.

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

double * eye (unsigned int n);

double * eye (unsigned int n) // Función pedida por el ejercicio.
{
	double * puntero = malloc((n*n) *sizeof(double)); // Malloc(reservar bytes) → malloc(cant. de elementos * tipo de elemento);
	if (puntero == NULL) // En el caso de que haya un error
	{
		printf("ERROR - No hay suficiente espacio en la memoria heap.\n");
		return puntero; // Devuelve el puntero (en este caso NULL)
	}

	int i,j;
	double*inicio = puntero; // Puntero al inicio de la matriz

	for (i=0; i<n; i++) // Pasa por todas las filas
	{
		for (j=0; j<n; j++, puntero++) // Pasa por todas las columnas
		{
			if (i == j) // Si es un elemento de la diagonal va un 1
			{
				*puntero = 1;
			}

			else // Sino va un 0
			{
				*puntero = 0;
			}
		}
	}

return inicio;
}
```

EJERCICIO 3: Se tiene un programa para trabajar en el HC11 con registros de 16 bits de manera genérica que no funciona. Mencionar los errores y corregir el programa para que funcione haciendo lo que dicen los comentarios. Obs: HC11 = plataforma big endian, arquitectura 16 bits.

RTA //programa corregido

```
#include <stdio.h>  // No <<stdio.h>>
#include <stdint.h> // Faltaba

typedef struct // Primero hay que hacer la estructura de los 8 bits A y B
{
      uint8_t lo; // Crea un campo de la estructura para trabajar con la parte menos significativa
      uint8_t hi; // Crea un campo de la estructura para trabajar con la parte más significativa
} bytes_t;

typedef struct // Después se unen
{
      bytes_t word;
} word_t;

typedef union
{
      word_t uword;
      bytes_t ubyte;
}reg_t;

int main(void)
{
      reg_t r1 = {.uword.word.0x1234}; // Inicializo con un valor que entra en los 16 bits de la variable, antes decía 0x1234
      reg_t r2 = {.ubyte = {0x56, 0x78}};

      printf("%X \n", r2.uword.word); // Imprime 7856
      printf("%X \n", r1.ubyte.lo);   // Imprime 34, la primera parte, es decir la “lo” de AB.
      printf("%X \n", r1.ubyte.hi);   // Imprime 12.

      return 0;
}
```

EJERCICIO 4: Escribir el makefile de un programa multiarchivo acorde al diagrama que hay en la foto que genere los archivos objetos intermedios y un ejecutable llamado cajero.

RTA

```
cajero: main.o display.o
      gcc -Wall main.o display.o -o cajero      #dependencias del main
main.o: display.h mensajes.h main.c
      gcc -Wall -c main.c           #genera el main.c
display.o: display.h display.c
      gcc -Wall -c dislpay.c        #genera el display.c
```

EJERCICIO 5: Explicar en profundidad el concepto de Integración Contínua

RTA

Integración Contínua es cuando los desarrolladores suben su código al repositorio compartido para que, al hacer un push, el servidor ejecute pruebas y rechace inmediatamente las modificaciones si estas no salen positivas.

SEGUNDO PARCIAL - 2019 1C

Ejercicio 1: Explicar qué es el argument promotion.

Es cuando los argumentos son promovidos automáticamente al utilizar argumentos variables, los char y short int son promovidos a int, y los floats son promovidos a double.

Ejercicio 6: Dado un makefile indicar que se imprime en consola si

- **a.** Se ejecuta por primera vez.
- **b.** Si se ejecuta después de hacer cambios en front.c
- **c.** Después de modificar front.h

SEGUNDO PARCIAL - 2018 1C

Ejercicio 1: Se pide implementar la función gswap() con el prototipo "void gswap(void *a, void* b, unsigned int size)" cuya misión es intercambiar los datos genéricos apuntados por los punteros. Dónde size es el tamaño de los datos en bytes

```
#include <stdio.h>

void gswap(void * a, void * b, unsigned int size); // Prototipo

void gswap(void * a, void * b, unsigned int size)
{
	char aux; // Variable para poder intercambiar datos sin perder el valor original de ninguno
	int byte; // Para poder trabajar con datos de cualquier tamaño

	for(byte = 0; byte < size; byte++)
	{
		aux = * ( ( char * ) a ); // Guarda el valor de lo que apunta el puntero para no perderlo

		*  ( ( char * ) a ) = * ( ( char * ) b ); // Ahora el puntero a contiene lo que apuntaba el puntero b

		*  ( ( char * ) b ) = aux; // El puntero b contiene el valor original al que apuntaba puntero a
	}
}
```

Ejercicio 2: Crear un puerto para un micro de 8 bits llamada "porta" y
a. se pueda acceder al byte completo, a cada nibble que compone el byte (MSN y LSN) y a cada bit de forma independiente
b. Escribir un programa que ponga en 1 los bits 0 y 2 de porta y que ponga en 0xA el MSN del porta
A continuación se deberá imprimir: El porta completo (byte) y cada nibble por separado

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

typedef struct
{
	uint8_t B0 : 1;
	uint8_t B1 : 1;
	uint8_t B2 : 1;
	uint8_t B3 : 1;
	uint8_t B4 : 1;
	uint8_t B5 : 1;
	uint8_t B6 : 1;
	uint8_t B7 : 1;
}bits_t;

typedef struct
{
	uint8_t MSB : 4; //cada nibble ocupa 4 bits
	uint8_t LSB : 4;
}nibbles_t;

typedef union
{
	bits_t bits;   // Para acceder a cada bit en particular
	nibbles_t nibbles; // Para acceder a cada nibble
	uint8_t puerto; // Puerto completo
}port_t;

int main (void)
{
	port_t porta; // Crea una variable de tipo port_t
	porta.bits.B0 = 1; // Prende ese bit
	porta.bits.B2 = 1; // Idem
	porta.nibbles.MSB = 0XA; // Le asigna ese valor al nibble
	printf("%08x\n",porta.puerto); // Imprime el puerto entero
	printf("%04x\n",porta.nibbles.LSB); // Imprime el LSN
	printf("%04x\n",porta.nibbles.MSB); // Imprime el MSN
	return 0;
}
```

SEGUNDO PARCIAL - 2015 2C

Ejercicio 2: Indicar la salida del programa

```
#include <stdio.h>

int main(int argc, const char * argv[])
{
    union trennen {     // trennen es el tag de la unión
        short int num;      // un campo dentro de la unión vier

      struct by4in4 {     // otro campo de la unión
            unsigned a1 : 4;
            unsigned a2 : 4;
            unsigned a3 : 4;
            unsigned a4 : 4;
        } vier;
    };
    union trennen valu;     // Crea la variable valu tipo de dato trennen
    valu.num=130;   // Le asigna un 130 a num

    printf( "\n %u-%u-%u-%u\n", valu.vier.a4 , valu.vier.a3 , valu.vier.a2 , valu.vier.a1 );
}
```

IMPRIME 0-0-8-2

Ejercicio 3: Cuál es la salida del siguiente código? asumir que el programa corre sobre una plataforma little endian y que 0x31='1' 0x41='A' 0x42='B'

```
#include <stdio.h>

int main()
{
    void *vp;   // Crea un puntero a void
    char ch=0x31, *cp="JACK"; // Crea la variable ch, le asigna 31 hexa, y crea el puntero a char, y lo apunta a un string de la flash
    int j=0x4142;

    vp=&ch;
    printf("%c - ", *(char*)vp);    // Imprime 1

    vp=&j;
    printf("%c - ", *(int*)vp);     // Imprime B

    vp=cp;
    printf("%s\n", (char*)vp+2);    // Imprime CK

    return 0;
}
```

Ejercicio 5: Escribir un programa que imprime los arg. de la línea de comandos, si no hay debe imprimir EMPTY

```
int main ( int argc , char * argv [] )
{
      int i;
      if ( argc == 1 )
      {
            printf (“EMPTY\n”);
      }
      for ( i=1 ; i<argc ; i++ )
      {
            printf (“%s” , argv[i]);
      }
}
```

Ejercicio 6: Dado el siguiente código cuyo propósito es ordenar el arreglo dado en orden ascendente se pide completar los argumentos de qsort() y escribir la función asc() así como su prototipo.

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

int compar (const void* p1, const void* p2);

void qsort(void* base, size_t nume, size_t size, int(*compar)(const void*, const void*));

int arr[] = {10,9,8,1,2,3,5}; // 7 elementos

int main (void)
{
	qsort (arr, 7, sizeof(arr[0]), compar);

	return 0;
}

int compar (const void* p1, const void* p2)
{
	if (*(int*)p1 > *(int*)p2)
	{
		return 1;
	}

	else if (*(int*)p1 < *(int*)p2)
	{
		return -1;
	}

	else
	{
		return 0;
	}
}
```

TRABAJO PRÁCTICO NÚMERO 13

Ejercicio 1: Escribir una función que cree una matriz identidad de nxn double. El prot. debe ser double* eye (unsigned int n), debe crear la matriz en el heap, completarla con los valores de la matriz identidad y devolver un puntero al primer elemento de la matriz. En caso de problema debe devolver NULL. Pensar cómo quedan ordenados en memoria los elementos de una matriz.

```
double* eye (unsigned int n) // La función devuelve un puntero a double, y recibe n.
{
      double *mat = (double*) malloc ( n * n * sizeof(double) ); // Crea el puntero mat para poder acceder al heap ya
                        // Que es la única forma de hacerlo. Y reservo n*n (el tamaño de la matriz) por el tipo de dato.

      if ( mat == NULL ) // Chequea que el puntero no sea nulo. Y de ser así, devuelvo el el mismo puntero, es decir
      {                   // El puntero nulo, tal y como indica la consigna. Imprimiendo un mensaje de error.
            printf("Error");
            return mat;
      }

      int i,j;                // Índices para recorrer la matriz
      double *p = mat;        // Puntero al inicio de la matriz

      for (i=0; i<n; i++) // Recorre la matriz
      {
            for (j=0; j<n; j++, p++) // Recorro las filas
            {
                  if (i == j)       // Recorro las columnas
                  {
                        *puntero = 1;     // Si estoy en un elemento de la identidad, relleno con unos
                  }

                  else
                  {
                        *puntero = 0;     // De lo contrario, ceros.
                  }
            }

      }

      return p; // Devuelve el puntero al primer elemento como pide la consigna
}
```

Ejercicio 2: Escribir la salida del siguiente programa multiarchivo. En cada printf indicar con una flecha a cuál instancia de la variable var hace referencia.

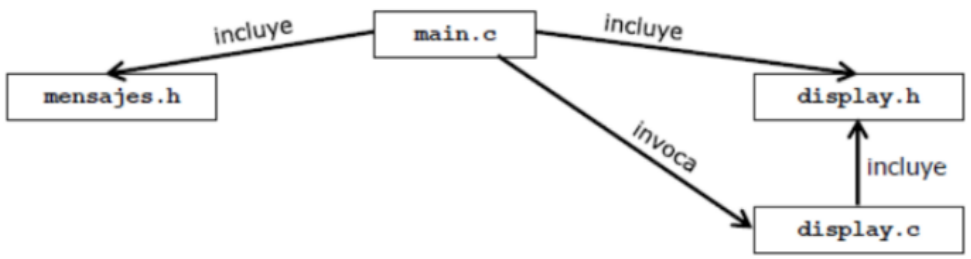
```c
#include <stdio.h>

void fun1(void);
void fun2(void);
int var;

int main(void)
{
	{
		int var=2;
		printf("S1: %d\n", var); // Imprime el 2 de arriba
		fun1(); // Voy a fun 1
	}
	var++; // Toma la variable 4, incrementa
	printf("S2: %d\n", var); // Imprime el 5
	fun1(); // Va a fun1, incrementa el 9 y lo imprime
	fun2(); // fun2, imprimo el 5 que es 4 incrementada
	return(0);
}

void fun1(void)
{
	static int var = 8;
	printf("S3: %d\n", ++var); // Imprimo 8 incrementado = 9
}
```

```c
#include <stdio.h>

int var = 4;

void fun2(void)
{
	printf("S4: %d\n", var);
}
```

RTA:

```
S1: 2
S3: 9
S2: 5
S3: 10
S4: 5
```

EJERCICIO 3: Escribir el makefile de un programa multiarchivo, acorde al siguiente diagrama, de manera que genere los archivos objetos intermedios y el archivo ejecutable, cuyo nombre debe ser cajero.



RTA

```
cajero: main.o display.o
	gcc -Wall -g main.o display.o -o cajero

main.o: main.c display.h mensajes.h
	gcc -Wall -g -c main.c

display.o: display.c mensajes.h
	gcc -Wall -g -o display.c
```

EJERCICIO 4: Escribir de manera recursiva una función que invierta la posición de los elementos de un arreglo de double de n elementos, para todo n>0. El prototipo debe ser *void flipArr (double arr[], int n)*.

```
void flipArr (double arr[], int n) //Inicia la función recursiva
{
	double aux;		//Crea una variable auxiliar tipo double
	if(n > 1)		//Chequea que n sea mayor que 1 (que no sea el caso base para de ser así salir)
	{
		aux = arr[0];			// Guarda en aux lo ubicado la posición inicial del arreglo.
		arr[0] = arr[n-1];		// Guarda en la posición inicial el valor de la posición final.
		arr[n-1] = aux;			// Guarda en el último elemento lo que estaba en el primero.
		flipArr(arr+1,n-2);		// Vuelve a invocar la función, con el segundo elemento, y con un n igual a n-2 ya que al
	}					// intercalar el inicial con el último, se trabajaron con 2 de los n elementos.
}
```

Ejercicio 5: Sea un programa de un menú de acceso mediante teclado, que se implementa con una lookup table, permite vincular el ingreso de una tecla con una acción. Para el mismo se pide:

a) Declarar una estructura, con su alias menu_t, que posea los campos tecla (tipo char) y action (tipo puntero a función que no recibe nada y devuelve un int).

b) Definir e inicializar un arreglo llamado menu_list de tipo menu_t de 2 elementos: el primer elemento es el carácter asterisco con la función borrar y el segundo es el carácter numeral (hashtag) con la función aceptar. Suponer que ya se encuentra escrita la definición y el prototipo de ambas funciones.

```
typedef struct{         // Define la estructura
{
      char tecla;       // Primer campo
      int ( *action ) (void); // Segundo campo
}menu_t; // Cierra la estructura con el nombre indicado

int main (void)
{
      menu_t menu_list [] = { {'*',borrar} , {'#',aceptar} };     // Creo una variable llamada menu_list de tipo de dato matriz de
      return 0;                                                   // elementos menu_t, donde el primer elemento es * - Borrar y el
}                                                                 // segundo es # - aceptar
```

Ejercicio 6: El siguiente es un programa para imprimir en pantalla una “mano de dados” (valor aleatorio de 5 dados cada vez que se invoca). Indicar si el código funciona correctamente o no, justificando su respuesta. En caso de que funcione incorrectamente, corregir el código y reescribirlo.

```
#include <stdio.h>

int main(void)
{
	int dado, i;
	for (i=1 ; i<=5 ; ++i)
	printf("%d\n", rand()%6 + 1);
	return 0;
}
```

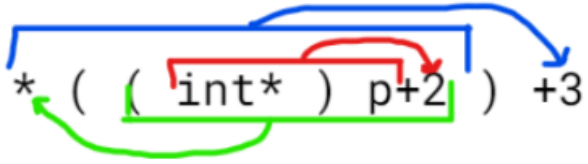
RTA

```
#include <time.h> // Estas librerías no estaban incluídas
#include <stdlib.h>
#include <stdio.h> // Mal incluida

int main(void)
{
	srand ( time(NULL) ); // Faltaba crear la semilla
	int i; // int dado sobra

	for (i=1 ; i<=5 ; ++i) // Imprime 5 dados
	{
		printf("%d\n", rand()%6 + 1); // +1 pq no hay dado 0
	}

return 0;
}
```

Ejercicio 7: Indicar que indica la salida del siguiente programa

```
#include <stdio.h>

void muestra(void* p);

int main(void)
{
      int mat[3][5] = { {0, 1, 2, 3, 4},{-5, -6, -7, -8, -9},{5, 6, 7, 8, 9} };
      muestra(mat+1+1/2);
      return 0;
}

void muestra(void* p)
{
      printf("%d\n", * ( ( int* ) p+2 ) +3 );
}
```

Razonamiento: De la invocación de muestra,se invoca a mat+1 por los casteos, y al ser arreglo de matrices con mat+1 se va al segundo elemento de mat, es decir a la segunda matriz que comienza con -5. Ahora con p+2 avanzo 2 ints, es decir me ubico en -7. Y ahora por la precendencia de los paréntesis, a dicho valor se le suma 3, es decir de -7+3 da -4 como resultado.



RTA

Imprime -4