



Práctica Final de Puntos

Fundamentos de programación

Reporte de práctica

Alumno: Ian Gutiérrez Segura



Para esta práctica final se implementará todo el conocimiento abarcado en el transcurso del semestre. Se ahondará en los temas más cruciales y de mayor presencia en el proyecto, se definirán conceptos, funciones y temas de matemáticas y geometría que fueron necesarios para satisfacer los requerimientos del extenso código que se realizó.

Este proyecto ha sido codificado en el lenguaje de programación C: desarrollado en los laboratorios Bell por AT&T entre 1969 y 1973 por Dennis Ritchie, de propósito general, compilado y de nivel medio bajo que ofrece como ventaja la economía de expresión, control de flujo y estructuras de datos modernos, así como un vasto conjunto de operadores. Para su implementación y ejecución, se utilizará la plataforma en línea GDB o GNU, que es el depurador estándar para el compilador GNU. Es un depurador portable que se puede utilizar en varias plataformas Unix y funciona para varios lenguajes de programación como C, C++ y Fortran. GDB fue escrito por Richard Stallman en 1986 y es un software libre distribuido bajo la licencia GPL.

Volviendo a la práctica, el reporte evaluará y profundizará punto a punto cada especificación en el documento de instrucciones de la práctica final para asegurar un mayor entendimiento de la problemática planteada y la forma en la que fue resuelto.

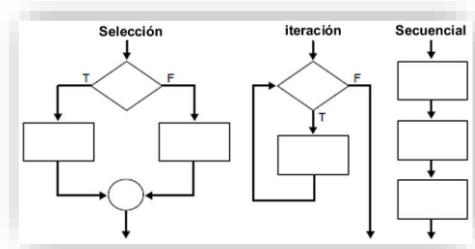
- El programa debe permitir el manejo de un menú interactivo con el usuario con las siguientes opciones:
 - Menú
 - 1.- Genera lista de puntos aleatoriamente
 - 2.- Genera lista de puntos manualmente
 - 3.- Obtener puntos de archivo
 - 4.- Suma puntos
 - 5.- Resta puntos
 - 6.- Multiplica puntos
 - 7.- Divide puntos
 - 8.- Determina distancia entre puntos
 - 9.- Determina la menor distancia entre puntos
 - 10.- Determinar el punto vecino más cercano
 - 11.- Imprime detalles de puntos
 - 12.- Grafica puntos (opcional)
 - 13.- Salir
 - (uso de funciones)
 - (uso de estructuras de control y repetición)

En el primer punto de esta práctica se solicita crear un menú de opciones que le permita al usuario realizar todas las operaciones y funciones que el programa puede ofrecer.

Durante las clases, se especificó que el programa debía ofrecer un control total al usuario; que fuera intuitivo, sin dejar que el usuario se pierda o confunda.

En las instrucciones igualmente se puede observar que los temas a desarrollar en este primer inciso son el uso de funciones de cualquier tipo y estructuras de control y repetición.

Hay que recordar que las estructuras de control son un conjunto de reglas que permiten controlar el flujo de ejecución de las instrucciones de un algoritmo. Estas están divididas en tres distintas categorías: secuencial, selección y repetición.



Por otro lado, las funciones son segmentos de código fuera de la función principal (externas) que dividen tareas grandes de computación en varias más pequeñas que pueden ser implementadas en cualquier momento desde cualquier parte del código (globales).

Estas funciones requieren tres partes esenciales para definirlas: tipo de dato de retorno, nombre de la función y, de ser el caso, parámetros de entrada. Estos parámetros pueden ser por copia (asignar un duplicado de una variable existente) o por referencia (asignar la dirección de memoria de la variable original).

El “ciclo” de una función es: definir la función (preferentemente bajo la función principal), hacer su prototipo (arriba de la función principal) e implementarse (en cualquier función que sea necesaria, incluso en la misma definición de la función, a eso se le conoce como una función recursiva directa).

Otro punto importe es la implementación de la librería o utilería stdio.h que contiene declaraciones para funciones relacionadas con entrada y salida estándar, como printf y scanf que son las principales funciones que se utilizaran para mostrar al usuario el menú y para recibir la opción seleccionada por él.

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

Con estas herramientas se tiene lo necesario para hacer el menú solicitado, sin embargo, el menú que se codificó sufrió ciertas variaciones al original: En lugar de crear un único menú con todas las opciones y validar en que momentos que opción puede el usuario seleccionar. Se crearon varios sub-menús, que permitieran una interacción un poco más llevadera.

Para ellos se utilizaron barias “banderas”, variables con el objetivo de marcar en que sub-menú se encuentra el usuario y que opciones ha seleccionado en que menú.

```
//Valores de la variable ronda:  
//0 -> Estamos en el menu_principal  
//1 -> Estamos en el menu_secundario  
//2 -> Estamos en algun otro menu  
int ronda = 0;  
//Valores de la variable opc y opc2:  
//opc -> opciones del menu principal y secundario  
//opc2 -> opciones de los otros menus  
int opc, opc2;
```

```
void menu_principal(int *opc){  
    do{  
        printf("\nMenú Principal\n");  
        printf("1.- Genera lista de puntos aleatoriamente\n");  
        printf("2.- Genera lista de puntos manualmente\n");  
        printf("3.- Obtener puntos de archivo\n");  
        printf("4.- Salir\n");  
        printf("Opción: ");  
        scanf("%d",opc);  
        _fpurge(stdin);  
    }while(*opc!=1&&*opc!=2&&*opc!=3&&*opc!=4);  
    return;  
}
```

Primero tenemos el menú principal que da la bienvenida al usuario. Este menú pregunta de qué forma se creará la lista de puntos: Aleatoriamente, manualmente o con un archivo ya existente. Repitiéndose indefinidamente en caso de que seleccionemos una opción inexistente.

Igualmente, el menú, al igual que todos los demás, cuanta con la opción de salir del programa.

Como se puede observar, esta función recibe parámetros por referencia, lo que implica utilizar punteros. Un puntero es una variable que contiene la dirección de memoria de otra variable. La definición de un puntero es igual a la de una variable, con la distinción de usa el operador de dirección (*) al inicio del nombre del puntero. Si se quiere cambiar la dirección de memoria a la que apunta, simplemente se iguala el nombre con la nueva dirección de memoria (recordemos que una forma de obtener la dirección de memoria de una variable es a través del símbolo de direccionamiento &_nombre_de_variable). En caso de querer acceder al contenido, se usa el operador de dirección.

El siguiente menú, una vez creados nuestros puntos, es el menú secundario:

Las operaciones que ofrece el programa han sido clasificadas en tres distintas categorías: Detalles, Operaciones y Distancias de puntos.

```
void menu_secundario(int *opc){  
    do{  
        printf("\nSegundo Menú\n");  
        printf("1.- Detalles de puntos\n");  
        printf("2.- Operaciones entre puntos\n");  
        printf("3.- Distancias entre puntos\n");  
        printf("4.- Regresar(Se perderán los datos guardados)\n");  
        printf("5.- Salir\n");  
        printf("Opción: ");  
        scanf("%d",opc);  
        _fpurge(stdin);  
    }while(*opc!=1&&*opc!=2&&*opc!=3&&*opc!=4&&*opc!=5);  
    return;  
}
```

La cuarta opción permite al usuario regresar al menú principal, con la advertencia de que los puntos y datos guardados anteriormente serán borrados del programa, aunque, los archivos se mantendrán sin ninguna variación siempre y cuando no se vuelvan a crear nuevos puntos y no se hayan movido los archivos de la ruta en la que fueron creados, en tal caso, los archivos se reescribirán con la nueva información.

```
void menu_detalles(int *opc){
    do{
        printf("\nMenú de Detalles\n");
        printf("1.- Detalles de un punto\n");
        printf("2.- Detalles de todos los puntos\n");
        printf("3.- Regresar\n");
        printf("4.- Salir\n");
        printf("Opción: ");
        scanf("%d",opc);
        _fpurge(stdin);
    }while(*opc!=1&&*opc!=2&&*opc!=3&&*opc!=4);
    return;
}

void menu_operaciones(int *opc){
    do{
        printf("\nMenú de Operaciones\n");
        printf("1.- Suma puntos\n");
        printf("2.- Resta puntos\n");
        printf("3.- Multiplica puntos\n");
        printf("4.- Divide puntos\n");
        printf("5.- Regresar\n");
        printf("6.- Salir\n");
        printf("Opción: ");
        scanf("%d",opc);
        _fpurge(stdin);
    }while(*opc!=1&&*opc!=2&&*opc!=3&&*opc!=4&&*opc!=5);
    return;
}

void menu_distancia(int *opc){
    do{
        printf("\nMenú de Distancia\n");
        printf("1.- Determina distancia entre puntos\n");
        printf("2.- Determina la menor distancia entre puntos\n");
        printf("3.- Determinar el punto vecino más cercano\n");
        printf("4.- Regresar\n");
        printf("5.- Salir\n");
        printf("Opción: ");
        scanf("%d",opc);
        _fpurge(stdin);
    }while(*opc!=1&&*opc!=2&&*opc!=3&&*opc!=4&&*opc!=5);
    return;
}
```

Con esto se cubre el primer punto de la práctica.

Para el segundo punto de la práctica, se especifican los detalles que integran a cada punto.

Para crear cada punto, se utilizarán los datos estructurados, estos son colecciones o agrupaciones de una o más variables que pueden ser de distintos tipos de datos que se referencian bajo un solo nombre como una unidad para manejarlas de una forma más conveniente y organizada.

Cabe remarcar que, para acceder a los miembros dentro de una variable de estructura, se utiliza el operador de acceso “.” que conecta al nombre de la estructura con el nombre del miembro. Sin embargo, en caso de que se quiera acceder a los miembros de un puntero de estructura, se utiliza el operador de acceso “->” que conecta al nombre de la estructura con el nombre del miembro.

Se recomienda definir las estructuras entre las librerías y los prototipos de las funciones.

```
struct PUNTO{
    int IdentificadorDePunto;
    float valorEnX;
    float valorEnY;
    float modulo;
    float angulo;
    int IdentificadorDeVecino;
};
```

En este caso, el nombre que se le ha asignado a esta estructura es “PUNTO”. Los miembros se han mantenido de la misma forma en la que fueron especificados, excepto por el ultimo atributo que servirá en un futuro para identificar el punto más cercano.

Siguiente: Menú de Detalles.

Este menú presenta las opciones de obtener los detalles de un punto el particular que el usuario desee, los detalles de todos los puntos almacenados, regresar al menú secundario o salir del programa.

Menú de operaciones.

Muestra las operaciones aritméticas que se pueden realizar entre los puntos almacenados: sumar, restar, multiplicar y dividir.

Siempre permitiendo regresar al menú secundario o salir.

Menú de Distancia.

En este sub-menú están contenidas las funciones: distancia entre dos puntos en concreto, la menor distancia entre dos puntos de todos los almacenados y obtener el punto más cercano al punto que el usuario escoga.

- Cada punto es una representación bidimensional que debe de contener:
 - IdentificadorDePunto como entero
 - valorEnX como flotante
 - valorEnY como flotante
 - modulo como flotante
 - ángulo como flotante
- (uso de funciones)
(uso de datos estructurados “structs”)

- Para la generación de puntos de forma manual o aleatoria, primero se recibirá del usuario la cantidad de puntos a crear y luego los valores valorEnX y valorEnY de cada punto (en un rango de -200 a 200), por lo que módulo y ángulo deberá de calcularse por cada punto, además de guardar dicho contenido en un archivo de texto con el nombre compendioDePuntos.txt con el siguiente formato:

IdentificadorDePunto	valorEnX	valorEnY	módulo	ángulo
Int	float	float	float	float

coordenadas en el eje “x” y “y” de cada punto. Aclarando que tanto el identificado, módulo y ángulo, deberán calcularse de forma autónoma sin la intervención del usuario. Agregando que, al finalizar la obtención de puntos, se creará un archivo que contenga la lista de puntos creados.

En clase, se hizo la distinción de que el programa, después de preguntar la cantidad de puntos a crear y de que se creen los puntos, debe permitir al usuario volver a crear más punto mientras él lo desee.

Para este punto se utilizarán varios conocimientos: calcular el módulo y ángulo de un punto, implementación de memoria dinámica y manejo de archivos.

¿Cómo se calcula el módulo? Se calcula de igual forma que la distancia entre puntos, con la distinción que un punto es el punto del que queremos calcular su módulo y el otro punto es el origen.

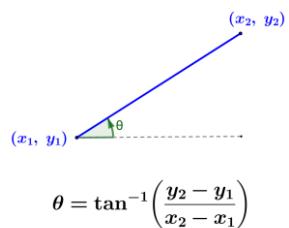
¿Cómo se calcula la distancia entre puntos? Para calcular la longitud del segmento que define al vector con origen en el punto uno y que se extiende hasta las coordenadas del punto dos. Se debe sacar la distancia en el eje x: desde “x1” hasta “x2”, y en el eje y: desde “y1” hasta “y2”; después, ambas distancias se elevan al cuadrado y se suman; al resultado obtenido le sacamos raíz cuadrada y ese será el valor de la distancia entre los puntos.

```
float distancia_entre_puntos(float ax,float ay,float bx,float by){
    //obtenemos La distancia en el eje_x y en el eje_y
    float distancia_x = ax-bx;
    float distancia_y = ay-by;
    //La distancia entre puntos es La raíz de La distancia en "x" al cuadrado más La de "y" al cuadrado
    float distancia = sqrt(pow(distancia_x,2)+pow(distancia_y,2));
    return distancia;
}
```

¿Cómo se calcula el ángulo de un punto? Para calcular el ángulo de cada punto, existe una ecuación que declara que la tangente del ángulo será igual a la distancia desde cero hasta la coordenada “y” del punto (que es igual a “y”), sobre la distancia desde cero hasta “x” (que es igual a “x”). Despejando: el ángulo es igual a la arco-tangente o tangente inversa de “y” sobre “x”.

Sin embargo, aquí se presenta un pequeño inconveniente, y es que, si se llega a dar el caso de que “x” es igual a cero, se nos presentará un error por dividir sobre cero, por lo que tendremos que validar que en esos casos el ángulo sea de cero, noventa o doscientos setenta grados, dependiendo del signo y magnitud de la coordenada en “y”.

Después se precisa que en el caso de que el usuario seleccione la opción uno o dos del menú principal, el usuario ingresará la cantidad de puntos que desea crear y el programa le solicitará uno a uno las



$$\theta = \tan^{-1}\left(\frac{y_2 - y_1}{x_2 - x_1}\right)$$

Por otro lado, existe otro detalle a resolver: la función atan() nos dará un valor expresado en radianes, por lo que, deberemos tener clara la siguiente relación de unidades para convertir el ángulo a grados: 1rad = 180°/pi.

```
float angulo_de_puntos(float ax,float ay){  
    //Existen algunas excepciones a la formula que debemos evaluar  
    //Como son los 0°, 90° y 270°  
    if(ax==0){  
        if(ay<0) return 270;  
        else if(ay==0) return 0;  
        else return 90;  
    }  
    //La tangente del angulo es igual a cateto opuesto sobre cateto adyacente  
    //Por lo que dejaremos al angulo de esta formula  
    float angulo_radianes = atan(ay/ax);  
    //1rad = 180/pi  
    float angulo_grados = angulo_radianes*180/3.14159265358979323846/*valor de pi*/;  
    //Ahora, tenemos que identificar el cuadrante en el que se encuentra para que el angulo en grados  
    //este bien representado en coordenadas polares con angulo positivo  
    if(ax < 0)angulo_grados = 180+angulo_grados;  
    else if(ay < 0)angulo_grados = 360+angulo_grados;  
  
    return angulo_grados;  
}
```

Otro punto importante es que el ángulo puede ser positivo o negativo, sin embargo, para resolver las futuras operaciones, se deben volver positivos todos los ángulos. Para esto se requerirá cierto conocimiento sobre los cuadrantes del plano cartesiano.

Para utilizar las funciones sqrt(), pow() y atan(), se requiere de la utilería math.h, la cual contiene declaraciones para funciones matemáticas.

```
#include <math.h>
```

Para manejar memoria dinamica se utilizaran las funciones malloc, calloc, realloc, free.

void *malloc(size_t t). Regresa un puntero de tipo “void”, que apunta al bloque de memoria dinámica reservado para un objeto de tamaño “t” en bytes o “NULL” si la solicitud no se puede satisfacer.

void *calloc(size_t n, size_t t). Regresa un puntero de tipo “void”, que apunta al bloque de memoria dinámica reservado para “n” cantidad de objetos de tamaño “t” en bytes o “NULL” si la solicitud no se puede satisfacer.

void *realloc(void *p, size_t t). Modifica, reasigna o cambia el tamaño “t”, en bytes, de un bloque de memoria dinámica previamente establecido “*p”.

void free(void *p). o Para evitar fugas de memoria y asegurar un uso eficiente de los recursos del sistema, free permite liberar o desasigna el bloque de memoria al que apunta “*p”. Si p es NULL, no hace nada.

Se deberá agregar al código la utilería stdlib.h, donde se declaran funciones de asignación de memoria.

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

Para el manejo de archivos únicamente será necesaria la utilería stdio.h la cual cuenta con las funciones requeridas para su aplicación.

El primer paso para crear un archivo es declarar una variable de tipo FILE que opere como el apuntador a la estructura del archivo.

El segundo paso será abrir un archivo con la función fopen(), esta determina el modo de apertura, el tipo de contenido del archivo y establece la vía de comunicación mediante su alias correspondiente.

```
FILE *alias;  
alias = fopen("nombre.txt","modo_apertura");
```

Los modos de apertura de archivos de texto y binario son:

Modo de apertura (archivos de texto)	Modo de apertura (archivos binarios)	Operación
“r”	“rb”	Apertura en modo de sólo lectura. El archivo debe existir.
“w”	“wb”	Apertura en modo de sólo escritura. Si el archivo existe, se reescribirá (pierde el contenido anterior). Si el archivo no existe, lo crea.
“a”	“ab”	Apertura en modo de agregar. Si el archivo existe, los datos se agregan al final del archivo, en caso contrario, el archivo se crea.
“r+”	“rb+”	Apertura en modo de lectura/escritura. El archivo debe existir.
“w+”	“wb+”	Apertura en modo de lectura/escritura. Si el archivo existe, se reescribirá (pierde el contenido anterior).
“a+”	“ab+”	Apertura en modo de lectura/agregar. Si el archivo no existe lo crea.

Por otro lado, cualquier archivo abierto dentro del programa, también deberá ser cerrado al finalizar su uso. Para ello, existen dos funciones: `fclose()` y `fcloseall()`. La diferencia entre estas radica en el hecho de que si se usa `fclose()` entonces será necesario indicar el alias del archivo que se desea cerrar. Por el contrario, la función `fcloseall()` cierra todos los archivos abiertos dentro del programa.

Ahora bien, para poder manipular su contenido existen varias funciones, sin embargo, para fines de esta práctica solo se profundizarán en algunas de ellas.

Nombre	Función
<code>fopen()</code>	Abre un archivo.
<code>fclose()</code>	Cierra un archivo.
<code>fgets()</code>	Lee una cadena de un archivo.
<code>fputs()</code>	Escribe una cadena en un archivo
<code>fseek()</code>	Busca un byte específico de un archivo.
<code>fprintf()</code>	Escribe una salida con formato en el archivo.
<code>fscanf()</code>	Lee una entrada con formato desde el archivo.
<code>feof()</code>	Devuelve cierto si se llega al final del archivo.
<code>ferror()</code>	Devuelve cierto si se produce un error.
<code>rewind()</code>	Coloca el localizador de posición del archivo al principio del mismo.
<code>remove()</code>	Borra un archivo.
<code>fflush()</code>	Vacia un archivo.

Función `rewind()`. Se usa esta función para colocar el puntero del archivo al principio de un archivo abierto. Basta con enviar el alias del archivo como argumento.

Función `fprintf()`. Esta función sirve para dar formato e imprimir una serie de caracteres y valores en la salida de stream. Para `fprintf`, el format argumento tiene la misma sintaxis que tiene en `printf`, por lo que la forma de utilizarlo es: `fprintf(FILE *stream, const char *format [, argument]...);`

Función `fscanf()`. La función `fscanf()` lee datos a partir de la posición actual de stream en las ubicaciones que argument proporciona (de haberlas). Cada argument debe ser la dirección de memoria una variable de un tipo que se corresponda con un especificador de tipo en format (igual que en `scanf()`). `int fscanf(FILE *stream, const char *format [, argument]...);`

Función feof(). Se usa feof() para determinar si se ha llegado al final de un archivo. En dicho caso, se devuelve un valor diferente de cero y cero en caso contrario. Para invocarlo es necesario colocar el alias del archivo abierto como argumento.

- En caso de elegir la opción de Obtener puntos de archivo, se deberá de llenar el arreglo estructurado de la lectura de un archivo de texto con el siguiente formato:

IdentificadorDePunto	valorEnX	valorEnY	modulo	ángulo
Int	float	float	float	float

dinamica y el calculo del modulo y ángulo que serviran para comparar los valores con los datos del archivo y modificarlos en caso de que sean incorrectos.

En adición, el profesor brindó un archivo demo:

El cual sirve de referencia para conocer ciertos parametros importantes para la creación y lectura de archivo, como la separación de un espacio entre elementos, el orden, etc.

Para el siguiente punto, que esta muy relacionado con el anterior. Se utilizará de igual forma el manejo de archivos, memoria

memoria

demo: Bloc de notas				
Archivo	Edición	Formato	Ver	Ayuda
1 52.36 0 52.36 0				
2 0 25.15 25.15 90				
3 -12.5 0 12.5 180				
4 0 -36.5 36.5 -90				
5 8.69 10.5 13.62 50.3				
6 -18.6 9.3 20.79 153.43				
7 -6.25 -5.36 8.23 -139.38				
8 15.8 -12.3 20.02 -37.90				
9 0 0 0 0				

Continuando con los requisitos de la practica:

Si el usuario ingresa al menú de operaciones entre puntos, dichas operaciones podran ser realizadas entre más de un par de puntos, según lo aclarado por el profesor.

Estas operaciones basicas son:

- Suma: La cual se representa como la suma de todas las coordenadas en "x" y, por separado, la suma de todas las coordenadas en "y", de los puntos implicados en la operación.

$$A+B+\dots+Z = (x_1+x_2+\dots+x_n) + i(y_1+y_2+\dots+y_n)$$
- Resta: Esta se representa restando al primer punto la suma de coordenadas en "x" de los demás puntos y la suma de coordenadas en "y" de los demás puntos.

$$A-B-\dots-Z = (x_1-x_2-\dots-x_n) + i(y_1-y_2-\dots-y_n)$$
- Multiplicación: En este caso se multiplican los modulos de los puntos y se suman los ángulos.

$$AB\dots Z = r_1 * r_2 * \dots * r_n \angle \alpha_1^\circ + \alpha_2^\circ + \dots + \alpha_n^\circ$$
- División: Aquí se dividen los modulos de los puntos y se restan los ángulos.

$$A/B/\dots/Z = r_1/r_2/\dots/r_n \angle \alpha_1^\circ - \alpha_2^\circ - \dots - \alpha_n^\circ$$

Para las ultimas operaciones existe la posibilidad de que los ángulos finales del resultado sean mayores a 360° o menores a 0° , por lo que se realizo una función que nos permitiera ajustar dichos ángulos a valores positivos dentro del rango previamente establecido.

```
float validar_angulo(float angulo){
    //Restamos o sumamos, dependera del caso, hasta que el angulo este entre el 0° y Los 360°
    while(angulo <= 0){
        angulo += 360;
    }
    while(angulo >= 360){
        angulo -= 360;
    }
    return angulo;
}
```

- El programa debe de permitir imprimir todos los puntos definidos o algún punto en particular.
(uso de funciones)
(uso de apuntadores)
(uso de apuntadores a datos estructurados)

Si el usuario se dirige al menú de detalles, podrá observar los detalles inherentes a un punto en específico a través de su id o el de todos los puntos que conforman el arreglo.

Desde que se crean los puntos y el usuario queda satisfecho con la cantidad de puntos en el arreglo, automáticamente después, se genera el archivo con la lista de puntos, el archivo de todas las distancias entre puntos y el archivo de los vecinos cercanos a cada punto.

- El programa debe de permitir determinar e imprimir la menor distancia existente entre 2 puntos del total de las comparativas de las distancias entre puntos y dichos resultados debe de guardarse en un archivo de texto con el nombre resultadosDeDistancias.txt.

Al crear el archivo de todas las distancias entre puntos se calcula, durante el proceso, la distancia menor y los puntos implicados.

Cuando el usuario ingrese al menú de distancias y seleccione la opción de la menor distancia entre 2 puntos, el resultado pre-calculado se mostrará.

- El programa debe de permitir determinar e imprimir el punto vecino más cercano de un punto dado y dicho resultado debe de guardarse en un archivo de texto con el nombre vecinoMasCercano.txt.

De igual forma que el punto anterior:

Al crear el archivo de los vecinos cercanos a cada punto, se guarda

dentro de los atributos de cada punto, el id del punto vecino. Por lo que, cuando el usuario ingrese al menú de distancias entre puntos y seleccione el vecino más cercano, el resultado pre-calculado se mostrará.

Una vez explicado cada punto, es hora de mostrar el código completo, junto con los comentarios de cada parte:

```
1 //Gutiérrez Segura Ian 1BV2
2
3 //-----Utilerias
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <math.h>
7 #include <time.h>
8
9 //-----Estructuras
10 struct PUNTO{
11     int IdentificadorDePunto;
12     float valorEnX;
13     float valorEnY;
14     float modulo;
15     float angulo;
16     int IdentificadorDeVecino;
17 };
18 struct DISTANCIA_MENOR{
19     int IdentificadorDePunto1;
20     int IdentificadorDePunto2;
21     float distancia;
22 };
23
24 //-----Prototipos
25 void menu_principal(int* );
26 void menu_secundario(int* );
27 void menu_detalles(int* );
28 void menu_operaciones(int* );
29 void menu_distancia(int* );
30 void llenar_PUNTOS(unsigned int,struct PUNTO*,unsigned int*,int);
31 float distancia_entre_puntos(float,float,float,float);
32 float angulo_de_puntos(float,float);
33 float validar_angulo(float);
34 void distancia_menor_vecino_cercano(struct PUNTO*,unsigned int,int*,struct DISTANCIA_MENOR* );
35
36 //-----Funcion Principal
37 int main(){
38     //Valores de la variable ronda:
39     //0 -> Estamos en el menu_principal
40     //1 -> Estamos en el menu_secundario
41     //2 -> Estamos en algun otro menu
42     int ronda = 0;
43     //Valores de la variable opc y opc2:
44     //opc -> opciones del menu principal y secundario
45     //opc2 -> opciones de los otros menus
46     int opc, opc2;
47     //IdentificadorDePunto y cantidad total de puntos
48     unsigned int id = 1, c_total = 0;
49     //Arreglo de puntos
50     struct PUNTO *PUNTOS;
51     //Guarda Los datos de la distancia menor
52     struct DISTANCIA_MENOR d_menor;
53     for();{
54         //----MENU
55         if(ronda == 0){//0 -> Estamos en el menu_principal
56             menu_principal(&opc);
57             //---Lista aleatoria o manual
58             if(opc==1 || opc==2){
59                 // c -> puntos que se quieren agregar
60                 unsigned int c;
61                 //Asignamos bloque de memoria
62                 PUNTOS = (struct PUNTO*)malloc(0);
63                 if(PUNTOS!=NULL){//Todo bien, seguimos:
64                     do{
65                         printf("\nCuantos puntos deseas crear?(0 si ya no deseas más puntos) ");
66                         scanf("%d",&c);
67                         _fpurge(stdin);
68                         if(c>0){
```

```

69 //Reajustamos el bloque de memoria reservado
70 PUNTOS = (struct PUNTO*)realloc(PUNTOS,(c+c_total)*sizeof(struct PUNTO));
71 if(PUNTOS!=NULL){//Todo bien, seguimos:
72     //Llevamos el puntero hasta Los Lugares que aún no se Llenan
73     PUNTOS += c_total;
74     //Agregamos Los nuevos puntos
75     llenar_PUNTOS(c, PUNTOS, &id, opc);
76     //Regresamos el puntero al primer lugar
77     PUNTOS -= c_total;
78     //Actualizamos La variable "c_total"
79     c_total += c;
80
81     //Pasamos a La siguiente ronda
82     ronda=1;
83
84 }//Si hay error, repetimos menu.
85 else printf("\nError al asignar el bloque de memoria dinamica\n");
86 }
87 }while(c!=0);
88 //Abrimos el archivo en le que guardaremos Los datos
89 FILE *f = fopen("compendioDePuntos.txt","w");
90 if(f!=NULL){
91     //Guardamos Los detalles de cada punto
92     for(id = 0; id < c_total; id++){
93         fprintf(f,"%d %f %f %f\n",PUNTOS->IdentificadorDePunto,PUNTOS->valorEnX,
94             PUNTOS->valorEnY,PUNTOS->modulo,PUNTOS->angulo);
95         PUNTOS++;
96     }
97     //Regresamos el puntero al primer lugar
98     PUNTOS -= c_total;
99     //Cerramos el archivo
100    fclose(f);
101 }//Si hay error, repetimos menu.
102 else printf("\nError al abrir archivo\n");
103 //Si hay error, repetimos menu.
104 else printf("\nError al asignar el bloque de memoria dinamica\n");
105 }
106 //----Lista archivo
107 else if(opc==3){
108     //Pedimos el nombre del archivo a Leer
109     char nom[21];
110     printf("\nDame el nombre del archivo a leer: ");
111     scanf("%s",nom);
112     _fpurge(stdin);
113
114     //Abrimos el archivo
115     FILE *f = fopen(nom,"r");
116     if(f!=NULL){
117         //Asignamos bloque de memoria
118         PUNTOS = (struct PUNTO*)malloc(0);
119         if(PUNTOS!=NULL){//Todo bien, seguimos:
120             //Mientras no lleguemos al final del archivo
121             while(!feof(f)){
122                 c_total++;
123                 //Reajustamos el bloque de memoria reservado
124                 PUNTOS = (struct PUNTO*)realloc(PUNTOS, c_total*sizeof(struct PUNTO));
125                 if(PUNTOS!=NULL{//Todo bien, seguimos:
126                     //Llevamos el puntero hasta los Lugares que aún no se Llenan
127                     PUNTOS += c_total-1;
128                     //Obtenemos datos y guardamos
129                     fscanf(f,"%d %f %f %f\n",&PUNTOS->IdentificadorDePunto,&PUNTOS->valorEnX,
130                         &PUNTOS->valorEnY,&PUNTOS->modulo,&PUNTOS->angulo);
131                     PUNTOS->IdentificadorDePunto = id++;
132                     //Validamos que los puntos no superen el rango de -200 a 200
133                     if(PUNTOS->valorEnX>200) PUNTOS->valorEnX=200;
134                     else if(PUNTOS->valorEnX<-200) PUNTOS->valorEnX=-200;
135                     if(PUNTOS->valorEnY>200) PUNTOS->valorEnY=200;
136                     else if(PUNTOS->valorEnY<-200) PUNTOS->valorEnY=-200;

```

```

136 //Obtenemos el modulo y angulo con los decimales extendidos y valores positivos
137 PUNTOS->modulo = distancia_entre_puntos(PUNTOS->valorEnX,PUNTOS->valorEnY,0,0);
138 PUNTOS->angulo = angulo_de_puntos(PUNTOS->valorEnX,PUNTOS->valorEnY);
139 //Regresamos el puntero al inicio
140 PUNTOS -= c_total-1;
141
142 //Pasamos a la siguiente ronda
143 ronda=1;
144
145 }else{
146     printf("\nError al asignar el bloque de memoria dinamica\n");
147 }
148 }
149 //Si hay error, repetimos menu.
150 else printf("\nError al asignar el bloque de memoria dinamica\n");
151 fclose(f);
152 //Si hay error, repetimos menu.
153 else printf("\nError al abrir archivo\n");
154 }
155 //----SALIR
156 else if(opc==4){
157     //Liberamos el bloque de memoria dinamica
158     free(PUNTOS);
159     return 0;
160 }
161 //Creamos el archivo de La distancia menor entre puntos y de vecinos cercanos
162 distancia_menor_vecino_cercano(PUNTOS,c_total,&ronda,&d_menor);
163 }
164 else if(ronda == 1){//1 -> Estamos en el menu_secundario
165     menu_secundario(&opc);
166     //Pasamos al siguiente menu
167     ronda = 2;
168 }
169 else if(ronda == 2){//2 -> Estamos en algun otro menu
170     //----Menu de Detalles
171     if(opc==1){
172         menu_detalles(&opc2);
173         //---Un solo punto
174         if(opc2==1){
175             //Obtenemos el id del punto a buscar
176             do{
177                 printf("\nCual es el identificador del punto? ");
178                 scanf("%d",&id);
179                 _fpurge(stdin);
180             }while(id<1 || id>c_total);
181             //Hacemos que el puntero señale el punto dentro del bloque de memoria
182             PUNTOS+=id-1;
183             //Imprimimos Los datos
184             printf("\nIdentificador: %d",PUNTOS->IdentificadorDePunto);
185             printf("\nValor en x: %f",PUNTOS->valorEnX);
186             printf("\nValor en y: %f",PUNTOS->valorEnY);
187             printf("\nModulo: %f",PUNTOS->modulo);
188             printf("\nAngulo: %f\n",PUNTOS->angulo);
189             //Regresamos el puntero al inicio del bloque de memoria
190             PUNTOS-=id-1;
191         }
192         //---Todos los puntos
193     else if(opc2==2){
194         //Recorremos todos los espacios del bloque de memoria con aritmetica de punteros
195         for(int i = 0; i<c_total;i++){
196             //Imprimimos la info de cada punto
197             printf("\nIdentificador: %d",PUNTOS->IdentificadorDePunto);
198             printf("\nValor en x: %f",PUNTOS->valorEnX);
199             printf("\nValor en y: %f",PUNTOS->valorEnY);
200             printf("\nModulo: %f",PUNTOS->modulo);
201             printf("\nAngulo: %f\n",PUNTOS->angulo);
202             PUNTOS++;
203         }

```

```

204 //Regresamos el puntero al inicio del bloque
205 PUNTOS=&_total;
206 }
207 //Pasamos a La ronda anterior
208 else if(opc2==3) ronda=1;
209 //---SALIR
210 else if(opc2==4){
211 //Liberamos el bloque de memoria dinamica
212 free(PUNTOS);
213 return 0;
214 }
215 }
216 //----Menu de Operaciones
217 else if(opc==2){
218 menu_operaciones(&opc2);
219 //---Suma o Resta o Multiplica o Divide
220 if(opc2==1||opc2==2||opc2==3||opc2==4){
221 //Variable que almacena la cantidad de puntos a operar
222 int _c;
223 do{
224 printf("\nDe cuantos puntos será la operación? ");
225 scanf("%d",&_c);
226 __fpurge(stdin);
227 }while(_c<1);
228 //Variables que guardan Los resultados de La operacion suma y resta
229 float res_en_x = 0, res_en_y = 0;
230 //Variables que guardan Los resultados de La operacion multiplica y divide
231 float modulo = 1, angulo = 0;
232 //Por la cantidad de puntos que operaremos, haremos:
233 for(int i = 0; i < _c; i++){
234 //Pedimos el identificador del punto (i+1)
235 do{
236 printf("\nCual es el identificador del punto %d? ",i+1);
237 scanf("%d",&id);
238 __fpurge(stdin);
239 }while(id<1 || id>_c_total);
240 //Hacemos que el puntero seleccione al punto especificado
241 PUNTOS+=id-1;
242 //Si la operacion es una suma
243 if(opc2==1){
244 //Sumamos en X y Y
245 res_en_x += PUNTOS->valorEnX;
246 res_en_y += PUNTOS->valorEnY;
247 }else if(opc2==2){
248 //Si la operacion es una resta
249 if(i==0){
250 //El primer punto será positivo
251 res_en_x = PUNTOS->valorEnX;
252 res_en_y = PUNTOS->valorEnY;
253 }else{
254 //Los demás puntos serán negativos
255 res_en_x -= PUNTOS->valorEnX;
256 res_en_y -= PUNTOS->valorEnY;
257 }
258 }else if(opc2==3){
259 //Si la operacion es una multiplicación
260 //Se multiplican los modulos u se suman los angulos
261 modulo *= PUNTOS->modulo;
262 angulo += PUNTOS->angulo;
263 //Validamos que el angulo no supere Los 360° o sea menor a 0°
264 angulo = validar_angulo(angulo);
265 }else if(opc2==4){
266 //Si la operacion es una division
267 if(i==0){
268 //El primer modulo será un dividendo
269 modulo = PUNTOS->modulo;
270 //El primer angulo será el positivo
271 angulo = PUNTOS->angulo;
272 }else{

```

```

273 //Los demás modulos serán los divisores
274 modulo /= PUNTOS->modulo;
275 //Los demás angulos serán los negativos
276 angulo -= PUNTOS->angulo;
277 //Validamos que el angulo no supere los 360° o sea menor a 0°
278 angulo = validar_angulo(angulo);
279 }
280 }
281 //Imprimimos los puntos que forman parte de la operación
282 printf("\nIdentificador: %d",PUNTOS->IdentificadorDePunto);
283 printf("\nValor en x: %f",PUNTOS->valorEnX);
284 printf("\nValor en y: %f",PUNTOS->valorEnY);
285 printf("\nModulo: %f",PUNTOS->modulo);
286 printf("\nAngulo: %f\n",PUNTOS->angulo);
287 //Regresamos el puntero al inicio
288 PUNTOS-=id-1;
289 }
290 //Imprimimos resultados dependiendo de la operación
291 if(opc2==1||opc2==2)
292 printf("\nEl resultado de la operación es: (%f)+i(%f)\n",res_en_x,res_en_y);
293 if(opc2==3||opc2==4)
294 printf("\nEl resultado de la operación es: %f %f\n",modulo,angulo);
295 }

296 //Pasamos a la ronda anterior
297 else if(opc2==5){ronda=1;}
298 //---SALIR
299 else if(opc2==6){
300 free(PUNTOS);
301 return 0;
302 }
303 }
304 //---Distancias
305 else if(opc==3){
306 menu_distancia(&opc2);
307 //---Entre dos puntos
308 if(opc2==1){
309 //Pedimos el primer punto
310 do{
311 printf("\nCual es el identificador del punto 1? ");
312 scanf("%d",&id);
313 _fpurge(stdin);
314 }while(id<1 || id>c_total);
315 //Hacemos que el puntero señale el punto ingresado
316 PUNTOS+=id-1;
317 //Mostramos Los datos del punto
318 printf("\nIdentificador: %d",PUNTOS->IdentificadorDePunto);

319 printf("\nValor en x: %f",PUNTOS->valorEnX);
320 printf("\nValor en y: %f",PUNTOS->valorEnY);
321 printf("\nModulo: %f",PUNTOS->modulo);
322 printf("\nAngulo: %f\n",PUNTOS->angulo);
323 //Obtenemos sus coordenadas
324 float ax = PUNTOS->valorEnX;
325 float ay = PUNTOS->valorEnY;
326 //Regresamos el puntero al inicio
327 PUNTOS-=id-1;
328 //Repetimos el proceso con el segundo punto
329 do{
330 printf("\nCual es el identificador del punto 2? ");
331 scanf("%d",&id);
332 _fpurge(stdin);
333 }while(id<1 || id>c_total);
334 PUNTOS+=id-1;
335 printf("\nIdentificador: %d",PUNTOS->IdentificadorDePunto);
336 printf("\nValor en x: %f",PUNTOS->valorEnX);
337 printf("\nValor en y: %f",PUNTOS->valorEnY);
338 printf("\nModulo: %f",PUNTOS->modulo);
339 printf("\nAngulo: %f\n",PUNTOS->angulo);
340 float bx = PUNTOS->valorEnX;
341 float by = PUNTOS->valorEnY;

```

```

342         PUNTOS->id-1;
343         //Obtenemos la distancia entre los dos puntos
344         float d = distancia_entre_puntos(ax,ay,bx,by);
345         //Imprimimos resultado
346         printf("\nLa distancia entre estos puntos es de: %f\n",d);
347     }
348     //---La menor distancia
349     else if(opc2==2){
350         //Imprimimos la distancia menor
351         printf("\nLa distancia menor (%f) es entre el punto %d y %d\n",
352             d_menor.distancia,d_menor.IdentificadorDePunto1,d_menor.IdentificadorDePunto2);
353         //Damos info de los dos puntos que cumplieron con dicha medición
354         PUNTOS+=d_menor.IdentificadorDePunto1-1;
355         printf("\nIdentificador: %d",PUNTOS->IdentificadorDePunto);
356         printf("\nValor en x: %f",PUNTOS->valorEnX);
357         printf("\nValor en y: %f",PUNTOS->valorEnY);
358         printf("\nModulo: %f",PUNTOS->modulo);
359         printf("\nAngulo: %f\n",PUNTOS->angulo);
360         PUNTOS+=d_menor.IdentificadorDePunto1-1;
361         PUNTOS+=d_menor.IdentificadorDePunto2-1;
362         printf("\nIdentificador: %d",PUNTOS->IdentificadorDePunto);
363         printf("\nValor en x: %f",PUNTOS->valorEnX);
364         printf("\nValor en y: %f",PUNTOS->valorEnY);
365         printf("\nModulo: %f",PUNTOS->modulo);
366         printf("\nAngulo: %f\n",PUNTOS->angulo);
367         PUNTOS+=d_menor.IdentificadorDePunto2-1;
368     }
369     //---Punto vecino
370     else if(opc2==3){
371         //Pedimos el punto a evaluar
372         do{
373             printf("\nCual es el identificador del punto? ");
374             scanf("%d",&id);
375             _fpurge(stdin);
376             }while(id<1 || id>c_total);
377         //Imprimimos el punto seleccionado
378         PUNTOS-=id-1;
379         printf("\nIdentificador: %d",PUNTOS->IdentificadorDePunto);
380         printf("\nValor en x: %f",PUNTOS->valorEnX);
381         printf("\nValor en y: %f",PUNTOS->valorEnY);
382         printf("\nModulo: %f",PUNTOS->modulo);
383         printf("\nAngulo: %f\n",PUNTOS->angulo);
384         //Obtenemos el id. del punto vecino
385         int id_vecino = PUNTOS->IdentificadorDeVecino;
386         //Obtenemos las coordenadas
387         float ax = PUNTOS->valorEnX;
388         float ay = PUNTOS->valorEnY;
389         //Regresamos el puntero al inicio
390         PUNTOS-=id-1;
391         //Imprimimos el punto vecino
392         PUNTOS+=id_vecino-1;
393         printf("\nIdentificador: %d",PUNTOS->IdentificadorDePunto);
394         printf("\nValor en x: %f",PUNTOS->valorEnX);
395         printf("\nValor en y: %f",PUNTOS->valorEnY);
396         printf("\nModulo: %f",PUNTOS->modulo);
397         printf("\nAngulo: %f\n",PUNTOS->angulo);
398         //Obtenemos las coordenadas
399         float bx = PUNTOS->valorEnX;
400         float by = PUNTOS->valorEnY;
401         //Regresamos el puntero al inicio
402         PUNTOS-=id_vecino-1;
403         //Obtenemos La distancia:
404         float menor = distancia_entre_puntos(ax,ay,bx,by);
405         //Imprimimos en pantalla el resultado
406         printf("\nEl vecino mas cercano del punto %d es el punto %d con %f\n",id,id_vecino,menor);
407     }
408     //Pasamos a La ronda anterior
409     else if(opc2==4){ronda=1;}
410     //---SALIR

```

```

411     }
412     //Liberamos Los bloques de memoria reservados
413     free(PUNTOS);
414     return 0;
415   }
416 }
417 //----Regresar
418 else if(opc==4){
419   //Regresamos Las variables a Los valores iniciales
420   id = 1;
421   c_total = 0;
422   //Liberamos Los bloques de memoria reservados
423   free(PUNTOS);
424   //Pasamos a La ronda anterior
425   ronda=0;
426 }
427 //----SALIR
428 else if(opc==5){
429   //Liberamos Los bloques de memoria reservados
430   free(PUNTOS);
431   return 0;
432 }
433 }

434 }

435 //-----Funciones
436 void menu_principal(int *opc){
437   do{
438     printf("\nMenú Principal\n");
439     printf("1.- Genera lista de puntos aleatoriamente\n");
440     printf("2.- Genera lista de puntos manualmente\n");
441     printf("3.- Obtener puntos de archivo\n");
442     printf("4.- Salir\n");
443     printf("Opción: ");
444     scanf("%d",opc);
445     _fpurge(stdin);
446   }while(*opc!=1&&opc!=2&&opc!=3&&opc!=4);
447   return;
448 }
449 void menu_secundario(int *opc){
450   do{
451     printf("\nSegundo Menú\n");
452     printf("1.- Detalles de puntos\n");
453     printf("2.- Operaciones entre puntos\n");
454     printf("3.- Distancias entre puntos\n");
455     printf("4.- Regresar(Se perderan los datos guardados)\n");
456     printf("5.- Salir\n");
457     printf("Opción: ");
458     scanf("%d",opc);
459     _fpurge(stdin);
460   }while(*opc!=1&&opc!=2&&opc!=3&&opc!=4&&opc!=5);
461   return;
462 }
463 void menu_detalles(int *opc){
464   do{
465     printf("\nMenú de Detalles\n");
466     printf("1.- Detalles de un punto\n");
467     printf("2.- Detalles de todos los puntos\n");
468     printf("3.- Regresar\n");
469     printf("4.- Salir\n");
470     printf("Opción: ");
471     scanf("%d",opc);
472     _fpurge(stdin);
473   }while(*opc!=1&&opc!=2&&opc!=3&&opc!=4);
474   return;
475 }
476 void menu_operaciones(int *opc){
477   do{

```

```

480     printf("\nMenú de Operaciones\n");
481     printf("1.- Suma puntos\n");
482     printf("2.- Resta puntos\n");
483     printf("3.- Multiplica puntos\n");
484     printf("4.- Divide puntos\n");
485     printf("5.- Regresar\n");
486     printf("6.- Salir\n");
487     printf("Opción: ");
488     scanf("%d",opc);
489     __fpurge(stdin);
490 }while(*opc!=1&*opc!=2&*opc!=3&*opc!=4&*opc!=5&*opc!=6);
491     return;
492 }
493 void menu_distancia(int *opc){
494 do{
495     printf("\nMenú de Distancia\n");
496     printf("1.- Determina distancia entre puntos\n");
497     printf("2.- Determina la menor distancia entre puntos\n");
498     printf("3.- Determinar el punto vecino más cercano\n");
499     printf("4.- Regresar\n");
500     printf("5.- Salir\n");
501     printf("Opción: ");
502     scanf("%d",opc);
503     __fpurge(stdin);
504 }while(*opc!=1&*opc!=2&*opc!=3&*opc!=4&*opc!=5);
505     return;
506 }
507 void llenar_PUNTOS(unsigned int c, struct PUNTO *PUNTOS, unsigned int *id, int mod){
508 //Llenamos el arreglo con aritmética de punteros
509 srand (time(NULL));
510 for(int i = 0; i < c; i++){
511
512     //Guardamos el id
513     PUNTOS -> IdentificadorDePunto = *id;
514     //Imprimimos el id
515     printf("\nIdentificador: %d\n",*id);
516     //Le aumentamos uno
517     *id+=1;
518
519     if(mod==1){//Llenado aleatorio
520         //Números aleatorios entre el -200 al 200, con decimales
521         PUNTOS -> valorEnX = ((rand()%399)-200)+(float)(rand()%1000)/1000;
522         PUNTOS -> valorEnY = ((rand()%399)-200)+(float)(rand()%1000)/1000;
523         //Imprimimos las coordenadas
524         printf("Valor en x: %f",PUNTOS -> valorEnX);
525         printf("\nValor en y: %f\n",PUNTOS -> valorEnY);
526     }
527     else if(mod==2){//Llenado manual
528         //Pedimos coordena x entre -200 y 200
529         do{
530             printf("Ingresa el valor en x entre -200 y 200: ");
531             scanf("%f",& PUNTOS -> valorEnX);
532             __fpurge(stdin);
533         }while(PUNTOS->valorEnX>200||PUNTOS->valorEnX<-200);
534         //Pedimos coordena y entre -200 y 200
535         do{
536             printf("Ingresa el valor en y entre -200 y 200: ");
537             scanf("%f",& PUNTOS -> valorEnY);
538             __fpurge(stdin);
539         }while(PUNTOS->valorEnY>200||PUNTOS->valorEnY<-200);
540     }
541     //Calculamos y guardamos el modulo del punto
542     PUNTOS->modulo = distancia_entre_puntos(PUNTOS->valorEnX,PUNTOS->valorEnY,0,0);
543     //Calculamos y guardamos el angulo del punto
544     PUNTOS->angulo = angulo_de_puntos(PUNTOS->valorEnX,PUNTOS->valorEnY);
545
546     //Imprimimos los valores
547     printf("Modulo: %f", PUNTOS->modulo);
548     printf("\nAngulo: %f\n",PUNTOS->angulo);

```

```

549     //Pasamos al siguiente punto
550     PUNTOS++;
551 }
552 return;
553 }
554 }
555 float distancia_entre_puntos(float ax,float ay,float bx,float by){
556 //Obtenemos la distancia en el eje_x y en el eje_y
557 float distancia_x = ax-bx;
558 float distancia_y = ay-by;
559 //La distancia entre puntos es la raiz de la distancia en "x" al cuadrado mas la de "y" al cuadrado
560 float distancia = sqrt(pow(distancia_x,2)+pow(distancia_y,2));
561
562 return distancia;
563 }
564 float angulo_de_puntos(float ax,float ay){
565 //Existen algunas excepciones a la formula que debemos evaluar
566 //Como son los 0°, 90° y 270°
567 if(ax==0){
568     if(ay<0) return 270;
569     else if(ay==0) return 0;
570     else return 90;
571 }
572 //La tangente del angulo es igual a cateto opuesto sobre cateto adyacente
573 //Por lo que dejaremos al angulo de esta formula
574 float angulo_radianes = atan(ay/ax);
575 //1rad = 180/pi
576 float angulo_grados = angulo_radianes*180/3.14159265358979323846/*valor de pi*/;
577 //Ahora, tenemos que identificar el cuadrante en el que se encuentra para que el angulo en grados
578 //este bien representado en coordenadas polares con angulo positivo
579 if(ax < 0)angulo_grados = 180+angulo_grados;
580 else if(ay<0)angulo_grados = 360+angulo_grados;
581
582 return angulo_grados;
583 }
584 float validar_angulo(float angulo){
585 //Restamos o sumamos, dependera del caso, hasta que el angulo este entre el 0° y los 360°
586 while(angulo <= 0){
587     angulo += 360;
588 }
589 while(angulo >= 360){
590     angulo -= 360;
591 }
592 return angulo;
593 }
594 void distancia_menor_vecino_cercano(struct PUNTO *PUNTOS,unsigned int c_total,int *ronda,struct DISTANCIA_MENOR *menor){
595 if(c_total==0){
596     //Repetimos ronda
597     *ronda = 0;
598     return;
599 }
600 //----Distancia Menor
601 //Abrimos nuestro archivo
602 FILE *f = fopen("resultadosDeDistancias.txt","w");
603 //----Vecino cercano
604 //Abrimos nuestro archivo
605 FILE *f2 = fopen("vecinoMasCercano.txt","w");
606 if(f!=NULL && f2!=NULL){
607     //menor -> Guardara la menor distancia
608     //menor2 -> Guardara la menor distancia entre vecinos
609     float menor, menor2;
610     //Guardaran el identificador del primer y segundo punto
611     int id1, id11;
612     int id2, id22;
613     //Titulo
614     fprintf(f,"\tDistancias entre puntos\n\n");
615     //Titulo
616     fprintf(f2,"\tVecino cercano entre puntos\n\n");
617     //Ponemos menor a un valor muy grande

```

```

618     menor = 1000;
619     //Pasamos por cada punto en el bloque de memoria
620     for(int i = 0; i<c_total; i++){
621         //Ponemos menor2 a un valor muy grande
622         menor2 = 1000;
623         //Guardamos el id del primer punto
624         id11 = i+1;
625         //Obtenemos las coordenadas del punto
626         PUNTOS+=i;
627         float ax = PUNTOS->valorEnX;
628         float ay = PUNTOS->valorEnY;
629         PUNTOS-=i;
630         if(i!=c_total-1){
631             //Pasamos por cada punto en el bloque de memoria a partir del que se tiene en i
632             for(int j = i+1; j<c_total; j++){
633                 //Obtenemos las coordenadas del punto
634                 PUNTOS+=j;
635                 float bx = PUNTOS->valorEnX;
636                 float by = PUNTOS->valorEnY;
637                 PUNTOS-=j;
638                 //Calculamos la distancia entre ambos puntos
639                 float d = distancia_entre_puntos(ax,ay,bx,by);
640                 //veremos si d es menor a la que teniamos guardada
641                 if(menor > d){
642                     //En tal caso, actualizamos los datos
643                     menor = d;
644                     id1 = i+1;
645                     id2 = j+1;
646                 }
647                 //Imprimimos en el archivo la distancia y los puntos de cada comparación
648                 fprintf(f,"Punto1: %d\tPunto2: %d\tDistancia: %f\n",i+1,j+1,d);
649             }
650         }
651         //Volvemos a pasamos por cada punto en el bloque de memoria
652         for(int j = 0; j<c_total; j++){
653             //Mientras no sea el mismo punto
654             if(i!=j){
655                 //Obtenemos las coordenadas del punto
656                 PUNTOS+=j;
657                 float bx = PUNTOS->valorEnX;
658                 float by = PUNTOS->valorEnY;
659                 PUNTOS-=j;
660                 //Calculamos la distancia entre ambos puntos
661                 float d = distancia_entre_puntos(ax,ay,bx,by);
662
663                 //Veremos si d es menor a la que teniamos guardada
664                 if(menor2 > d){
665                     //En tal caso, actualizamos los datos
666                     menor2 = d;
667                     //Guardamos el id del segundo punto
668                     id22 = j+1;
669                 }
670             }
671         }
672         //Imprimimos en el archivo la distancia menor y los puntos de cada comparación
673         fprintf(f2,"Punto1: %d\tPunto2: %d\tDistancia: %f\n",id11,id22,menor2);
674         PUNTOS+=i;
675         PUNTOS->IdentificadorDeVecino = id22;
676         PUNTOS-=i;
677     }
678     //Guardo los datos importantes de la distancia menor entre puntos
679     d_menor -> IdentificadorDePunto1 = id1;
680     d_menor -> IdentificadorDePunto2 = id2;
681     d_menor -> distancia = menor;
682     //Cerramos los archivos
683     fclose(f);
684     fclose(f2);
685     //Pasaremos a la siguiente ronda
686     *ronda = 1;
687     return;
688 } //En caso de un problema
689 printf("\nHubo un problema al crear los archivos.\n");
690 //Repetimos ronda
691 *ronda = 0;
692 return;
693 }

```

Casos de Uso:

Para cualquier menú, si se llega a escribir cualquier cosa que no sea una de las opciones presentadas, el menú se repetirá:

```
Menú Principal
1.- Genera lista de puntos aleatoriamente
2.- Genera lista de puntos manualmente
3.- Obtener puntos de archivo
4.- Salir
Opción: 5

Menú Principal
1.- Genera lista de puntos aleatoriamente
2.- Genera lista de puntos manualmente
3.- Obtener puntos de archivo
4.- Salir
Opción: 0

Menú Principal
1.- Genera lista de puntos aleatoriamente
2.- Genera lista de puntos manualmente
3.- Obtener puntos de archivo
4.- Salir
Opción: a
```

De cualquier manera, se generan los puntos apropiadamente:

Forma aleatoria:

```
Menú Principal
1.- Genera lista de puntos aleatoriamente
2.- Genera lista de puntos manualmente
3.- Obtener puntos de archivo
4.- Salir
Opción: 1

Cuantos puntos deseas crear?(0 si ya no deseas más puntos) 2
Identificador: 1
Valor en x: -197.399002
Valor en y: 121.676003
Modulo: 231.886642
Angulo: 148.350479

Identificador: 2
Valor en x: 611.443997
Valor en y: 183.031006
Modulo: 200.211624
Angulo: 66.090630

Cuantos puntos deseas crear?(0 si ya no deseas más puntos) 0
```

Regresamos:

```
Segundo Menú
1.- Detalles de puntos
2.- Operaciones entre puntos
3.- Distancias entre puntos
4.- Regresar(Se perderán los datos guardados)
5.- Salir
Opción: 4
```

Forma manual:

```
Menú Principal
1.- Genera lista de puntos aleatoriamente
2.- Genera lista de puntos manualmente
3.- Obtener puntos de archivo
4.- Salir
Opción: 2

Cuantos puntos deseas crear?(0 si ya no deseas más puntos) 2
Identificador: 1
Ingresa el valor en x entre -200 y 200: -201
Ingresa el valor en x entre -200 y 200: -200
Ingresa el valor en y entre -200 y 200: 201
Ingresa el valor en y entre -200 y 200: 200
Modulo: 282.842712
Angulo: 135.000000

Identificador: 2
Ingresa el valor en x entre -200 y 200: 0
Ingresa el valor en y entre -200 y 200: 0
Modulo: 0.000000
Angulo: 0.000000

Cuantos puntos deseas crear?(0 si ya no deseas más puntos) 0
```

Regresamos:

Leyendo el archivo de los puntos anteriores y mostrando detalles:

```
Segundo Menú
1.- Detalles de puntos
2.- Operaciones entre puntos
3.- Distancias entre puntos
4.- Regresar(Se perderán los datos guardados)
5.- Salir
Opción: 4
```

```
Menú Principal
1.- Genera lista de puntos aleatoriamente
2.- Genera lista de puntos manualmente
3.- Obtener puntos de archivo
4.- Salir
Opción: 3

Dame el nombre del archivo a leer: compendioDePuntos.txt
segundo Menú
1.- Detalles de puntos
2.- Operaciones entre puntos
3.- Distancias entre puntos
4.- Regresar(Se perderán los datos guardados)
5.- Salir
Opción: 1

Menú de Detalles
1.- Detalles de un punto
2.- Detalles de todos los puntos
3.- Regresar
4.- Salir
Opción: 2
```

```
Identificador: 1
Valor en x: -200.000000
Valor en y: 200.000000
Modulo: 282.842712
Angulo: 135.000000

Identificador: 2
Valor en x: 0.000000
Valor en y: 0.000000
Modulo: 0.000000
Angulo: 0.000000

Menú de Detalles
1.- Detalles de un punto
2.- Detalles de todos los puntos
3.- Regresar
4.- Salir
Opción: 1
```

Generación de archivos al concluir la creación de puntos:

Al inicio:

```
main.c
main.c: In function 'main':
main.c:67:25: warning: implicit declaration of function 'fpurge'
    67 |             fpurge(stdin);
                  ^~~~~~
Menú Principal
1.- Genera lista de puntos aleatoriamente
2.- Genera lista de puntos manualmente
3.- Obtener puntos de archivo
4.- Salir
Opción: 1
```

Después de crear puntos y salir del programa:

```
main.c  compendioDePunt...  resultadosDeDist...  vecinoMasCercan...
1  Vecino cercano entre puntos
input
Menú Principal
1.- Genera lista de puntos aleatoriamente
2.- Genera lista de puntos manualmente
3.- Obtener puntos de archivo
4.- Salir
Opción: 1

Cuantos puntos deseas crear?(0 si ya no deseas más puntos) 2

Identificador: 1
Valor en x: 100.846001
Valor en y: -142.533005
Modulo: 174.601181
Angulo: 305.280457

Identificador: 2
Valor en x: 93.974998
Valor en y: -194.542999
Modulo: 216.051559
Angulo: 295.783112

Cuantos puntos deseas crear?(0 si ya no deseas más puntos) 0
```

```
Segundo Menú
1.- Detalles de puntos
2.- Operaciones entre puntos
3.- Distancias entre puntos
4.- Regresar(Se perderán los datos guardados)
5.- Salir
Opción: 5

...Program finished with exit code 0
Press ENTER to exit console.
```

Archivos:

```
main.c  compendioDePunt...  resultadosDeDist...  vecinoMasCercan...
1  180.846001 -142.533005 174.601181 305.280457
2  2 93.974998 -194.542999 216.051559 295.783112
3
```

```
main.c  compendioDePunt...  resultadosDeDist...  vecinoMasCercan...
1  Distancias entre puntos
2
3  Punto1: 1  Punto2: 2  Distancia: 52.461891
4
5
```

```
main.c  compendioDePunt...  resultadosDeDist...  vecinoMasCercan...
1  Vecino cercano entre puntos
2
3  Punto1: 1  Punto2: 2  Distancia: 52.461891
4  Punto1: 2  Punto2: 1  Distancia: 52.461891
5
```

Validación al generar puntos (No se pueden crear –n puntos, ni solo un punto):

```
Menú Principal
1.- Genera lista de puntos aleatoriamente
2.- Genera lista de puntos manualmente
3.- Obtener puntos de archivo
4.- Salir
Opción: 1

Cuantos puntos deseas crear?(0 si ya no deseas más puntos) -1
No es aceptable su respuesta.

Cuantos puntos deseas crear?(0 si ya no deseas más puntos) 0

Menú Principal
1.- Genera lista de puntos aleatoriamente
2.- Genera lista de puntos manualmente
3.- Obtener puntos de archivo
4.- Salir
Opción: 1

Cuantos puntos deseas crear?(0 si ya no deseas más puntos) 1
No es aceptable su respuesta.
```

```
Cuantos puntos deseas crear?(0 si ya no deseas más puntos) 2

Identificador: 1
Valor en x: -180.468002
Valor en y: 72.663002
Modulo: 194.547195
Angulo: 158.069461

Identificador: 2
Valor en x: 93.917999
Valor en y: -176.358002
Modulo: 199.806747
Angulo: 298.037170

Cuantos puntos deseas crear?(0 si ya no deseas más puntos) 1

Identificador: 3
Valor en x: 127.000000
Valor en y: -129.338997
Modulo: 149.745371
Angulo: 121.993698

Cuantos puntos deseas crear?(0 si ya no deseas más puntos) 0
```

```
Segundo Menú
1.- Detalles de puntos
2.- Operaciones entre puntos
3.- Distancias entre puntos
4.- Regresar(Se perderán los datos guardados)
5.- Salir
Opción: 5

...Program finished with exit code 0
Press ENTER to exit console.
```

Módulos y angulos correctos:

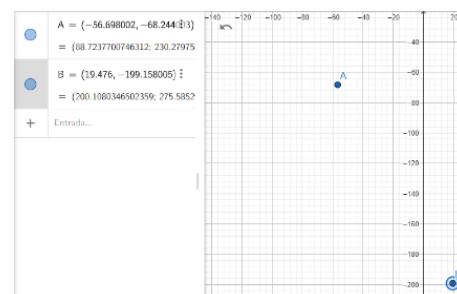
```
Menú Principal
1.- Genera lista de puntos aleatoriamente
2.- Genera lista de puntos manualmente
3.- Obtener puntos de archivo
4.- Salir
Opción: 1

Cuantos puntos deseas crear?(0 si ya no deseas más puntos) 2

Identificador: 1
Valor en x: -56.698002
Valor en y: -68.244003
Modulo: 88.723770
Angulo: 230.279755

Identificador: 2
Valor en x: 19.476000
Valor en y: -199.158005
Modulo: 200.108032
Angulo: 275.585297

Cuantos puntos deseas crear?(0 si ya no deseas más puntos) 0
```



Módulos, angulos “especiales” ($0^\circ, 90^\circ, 180^\circ, 270^\circ$) y distancias:

```

Menú Principal
1.- Genera lista de puntos aleatoriamente
2.- Crea una lista de puntos manualmente
3.- Obtener puntos de archivo
4.- Salir
Opción: 2

Cuantos puntos deseas crear?(0 si ya no deseas más puntos) 5

Identificador: 1
Ingresa el valor en x entre -200 y 200: 0
Ingresa el valor en y entre -200 y 200: 5
Modulo: 5.000000
Angulo: 90.000000

Identificador: 2
Ingresa el valor en x entre -200 y 200: -5
Ingresa el valor en y entre -200 y 200: 0
Modulo: 5.000000
Angulo: 180.000000

Identificador: 3
Ingresa el valor en x entre -200 y 200: 5
Ingresa el valor en y entre -200 y 200: 0
Modulo: 5.000000
Angulo: 0.000000

Identificador: 4
Ingresa el valor en x entre -200 y 200: 0
Ingresa el valor en y entre -200 y 200: -5
Modulo: 5.000000
Angulo: 270.000000

Identificador: 5
Ingresa el valor en x entre -200 y 200: 5
Ingresa el valor en y entre -200 y 200: 5
Modulo: 7.071068
Angulo: 45.000000

Cuantos puntos deseas crear?(0 si ya no deseas más puntos) 0

```

```

Segundo Menú
1.- Detalles de puntos
2.- Operaciones entre puntos
3.- Distancias entre puntos
4.- Regresar(Se perderan los datos guardados)
5.- Salir
Opción: 3

Menú de Distancia
1.- Determina distancia entre puntos
2.- Determina la menor distancia entre puntos
3.- Determinar el punto vecino más cercano
4.- Regresar
5.- Salir
Opción: 2

La distancia menor (5.000000) es entre el punto 1 y 5

Identificador: 1
Valor en x: 0.000000
Valor en y: 5.000000
Modulo: 5.000000
Angulo: 90.000000

Identificador: 5
Valor en x: 5.000000
Valor en y: 5.000000
Modulo: 7.071068
Angulo: 45.000000

Menú de Distancia
1.- Determina distancia entre puntos
2.- Determina la menor distancia entre puntos
3.- Determinar el punto vecino más cercano
4.- Regresar
5.- Salir
Opción: 3

Cual es el identificador del punto? 1

Identificador: 1
Valor en x: 0.000000
Valor en y: 5.000000
Modulo: 5.000000
Angulo: 90.000000

Identificador: 5
Valor en x: 5.000000
Valor en y: 5.000000
Modulo: 7.071068
Angulo: 45.000000

El vecino mas cercano del punto 1 es el punto 5 con 5.000000

Menú de Distancia
1.- Determina distancia entre puntos
2.- Determina la menor distancia entre puntos
3.- Determinar el punto vecino más cercano
4.- Regresar
5.- Salir
Opción: 1

Cual es el identificador del punto 1? 1

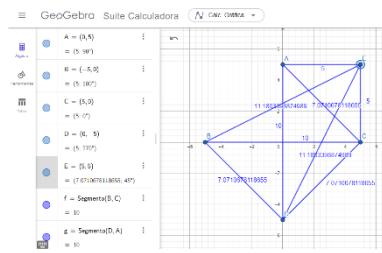
Identificador: 1
Valor en x: 0.000000
Valor en y: 5.000000
Modulo: 5.000000
Angulo: 90.000000

Cual es el identificador del punto 2? 5

Identificador: 5
Valor en x: 5.000000
Valor en y: 5.000000
Modulo: 7.071068
Angulo: 45.000000

La distancia entre estos puntos es de: 5.000000

```



Como se puede ver en Geogebra, la distancia minima es entre el punto 1 y 5 o 3 y 5, y como se puede apreciar en los resultados del programa, los valores que nos arrojó fue de la primer distancia menor que encontró (1,5)

No se pueden calcular distancias, operaciones o imprimir detalles de puntos que no existen:

Creamos 10 puntos aleatorios:

```
Menú Principal
1.- Genera lista de puntos aleatoriamente
2.- Genera lista de puntos manualmente
3.- Obtener puntos de archivo
4.- Salir
Opción: 1

Cuantos puntos deseas crear?(0 si ya no deseas más puntos) 10
```

Pedimos distancias de puntos no validos:

```
Menú de Detalles
1.- Detalles de un punto
2.- Detalles de todos los puntos
3.- Regresar
4.- Salir
Opción: 3

Segundo Menú
1.- Detalles de puntos
2.- Operaciones entre puntos
3.- Distancias entre puntos
4.- Regresar(Se perderan los datos guardados)
5.- Salir
Opción: 3

Menú de Distancia
1.- Determina distancia entre puntos
2.- Determina la menor distancia entre puntos
3.- Determinar el punto vecino más cercano
4.- Regresar
5.- Salir
Opción: 3
Cual es el identificador del punto? 0
Cual es el identificador del punto? 11
Cual es el identificador del punto? e
Cual es el identificador del punto? 1
Identificador: 1
Valor en x: 140.759995
Valor en y: 46.650002
Modulo: 148.288910
Angulo: 18.335981

Identificador: 9
Valor en x: 184.328995
Valor en y: 14.830000
Modulo: 184.924606
Angulo: 4.599765

El vecino mas cercano del punto 1 es el punto 9 con 53.951553
```

Pedimos detalles de puntos no validos:

```
Segundo Menú
1.- Detalles de puntos
2.- Operaciones entre puntos
3.- Distancias entre puntos
4.- Regresar(Se perderan los datos guardados)
5.- Salir
Opción: 1
```

```
Menú de Detalles
1.- Detalles de un punto
2.- Detalles de todos los puntos
3.- Regresar
4.- Salir
Opción: 1

Cual es el identificador del punto? 0
Cual es el identificador del punto? 11
Cual es el identificador del punto? e
Cual es el identificador del punto? 1
Identificador: 1
Valor en x: 140.759995
Valor en y: 46.650002
Modulo: 148.288910
Angulo: 18.335981
```

Conclusiones:

C es un lenguaje de programación que abarca una amplia gama de conceptos y procesos, permitiéndonos resolver una gran cantidad de problemáticas de distintas índoles. El lenguaje C es de propósito general y estructurado; a través de funciones, punteros, estructuras, memoria dinámica y manejo de archivos, se pueden codificar diversos programas, incluido el propuesto en esta práctica.

La versatilidad de C radica en su capacidad para abordar desafíos de programación de manera eficiente y efectiva. Su enfoque estructurado facilita la creación de programas organizados y comprensibles, lo que resulta fundamental en el desarrollo de software robusto y mantenable. Las funciones en C permiten modularizar el código, mejorando la legibilidad y fomentando la reutilización del mismo.

El uso de punteros en C proporciona un control preciso sobre la memoria, lo que permite una gestión eficiente de recursos. Esto se traduce en la capacidad de optimizar el rendimiento de los programas, especialmente en situaciones donde la eficiencia es crucial. Además, la manipulación de estructuras brinda una manera efectiva de organizar y gestionar datos complejos, facilitando la implementación de algoritmos avanzados.

La asignación dinámica de memoria es otra característica distintiva de C que permite adaptarse a las necesidades cambiantes durante la ejecución del programa. Esta flexibilidad es esencial para resolver problemas dinámicos y escalar aplicaciones a medida que los requisitos evolucionan.

El manejo de archivos en C proporciona la capacidad de interactuar con sistemas de almacenamiento, facilitando la entrada y salida de datos. Esto es fundamental en situaciones donde la persistencia de información es crucial, como en el caso de bases de datos o la manipulación de archivos de configuración.

En adición, la resolución de esta práctica ha sido en extremo beneficiosa, brindando habilidades relacionadas a la eficiencia y optimización en diversos campos, enriqueciendo la experiencia cotidiana al facilitar tareas, tomar decisiones informadas y mejorar la calidad del trabajo. La versatilidad y poder de C en este contexto hacen de la programación en este lenguaje una habilidad valiosa y relevante en el panorama actual.

Como conclusión, el lenguaje de programación C se erige como una herramienta indispensable en el mundo del desarrollo de software debido a su versatilidad, eficiencia y capacidad para abordar una amplia variedad de desafíos. La combinación de sus características estructuradas, el uso de punteros, la manipulación de estructuras, la asignación dinámica de memoria y el manejo de archivos ofrece a los programadores un conjunto robusto de herramientas para crear soluciones sólidas y adaptables. En este sentido, la práctica realizada demuestra cómo C puede aplicarse con éxito para resolver problemas concretos, sentando las bases para el dominio y la aplicación efectiva de este lenguaje en proyectos más amplios y complejos.

Referencias:

- BRIAN W, KERNIGHAN DENNIS M y RITCHIEEL. (1991). LENGUAJE DE PROGRAMACION C. https://frrq.cvg.utn.edu.ar/pluginfile.php/13741/mod_resource/content/0/El-lenguaje-de-programacion-C-2-ed-kernighan-amp-ritchie.pdf
- Enrique Vicente Bonet Esteban. (s.f.). Lenguaje C. <https://informatica.uv.es/estguia/ATD/apuntes/laboratorio/Lenguaje-C.pdf>
- GCC, the GNU Compiler Collection - GNU Project. (n.d.). <https://gcc.gnu.org/>
- Raichman. (Febrero de 2016). Geometría analítica para ciencias e ingenierías. https://bdigital.uncu.edu.ar/objetos_digitales/7224/librogeoing.pdf
- López, E. (2020, December 3). Prototipo de una función en C++ - C++ esencial [Video]. LinkedIn. <https://es.linkedin.com/learning/c-plus-plus-esencial/prototipo-de-una-funcion-en-c-plus-plus>
- Oscar Parada. (2021, July 6). Programa en Lenguaje C, Agenda Telefónica [Video]. YouTube. <https://www.youtube.com/watch?v=9gaokCVP97I>
- Conversiones de tipo (casting) en C. (2012, May 15). <https://programacionnerd.blogspot.com/2012/05/c-conversiones-de-tipo-casting-en-c.html>
- GeeksforGeeks. (2023, February 2). C malloc. <https://www.geeksforgeeks.org/cpp-malloc/>
- Bruno López Takeyas, M.C. (s.f.). Manejo de Archivos en Lenguaje C++. https://nlaredo.tecnm.mx/takeyas/Apuntes/Administracion_Archivos/Apuntes/Manejo%20de%20Archivos%20en%20Lenguaje%20C/Manejo%20de%20Archivos%20en%20Lenguaje%20C.pdf
- TylerMSFT. (2023, December 10). fscanf, _fscanf_l, fscanf, _fwscanf, _fwscanf_l. Microsoft Learn. <https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/fscanf-fscanf-l-fwscanf-fwscanf-l?view=msvc-170>
- TylerMSFT. (2023, February 4). fprintf, _fprintf_l, fwprintf, _fwprintf_l. Microsoft Learn. <https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/fprintf-fprintf-l-fwprintf-fwprintf-l?view=msvc-170>
- datos.gob.es. (2022, May 3). 11 librerías para crear visualizaciones de datos. datos.gob.es. <https://datos.gob.es/es/blog/11-librerias-para-crear-visualizaciones-de-datos>
- Studocu. (n.d.). C historia - C: EL ORIGEN C es el lenguaje de programación de propósito general asociado, de modo - Studocu. <https://www.studocu.com/es-mx/document/instituto-mexico/fundamentos-de-programacion/c-historia/8917285>
- McGraw-Hill Interamericana de España SL. (n.d.). https://www.mhe.es/ceo_datos.php?tipo=1_01_B&isbn=8448198441&sub_materia=98&materia=21&nivel=&comunidad=&ciclo=&portal=&letrero=&cabecera=
- Documentación oficial del estándar ANSI C (ISO/IEC 9899).