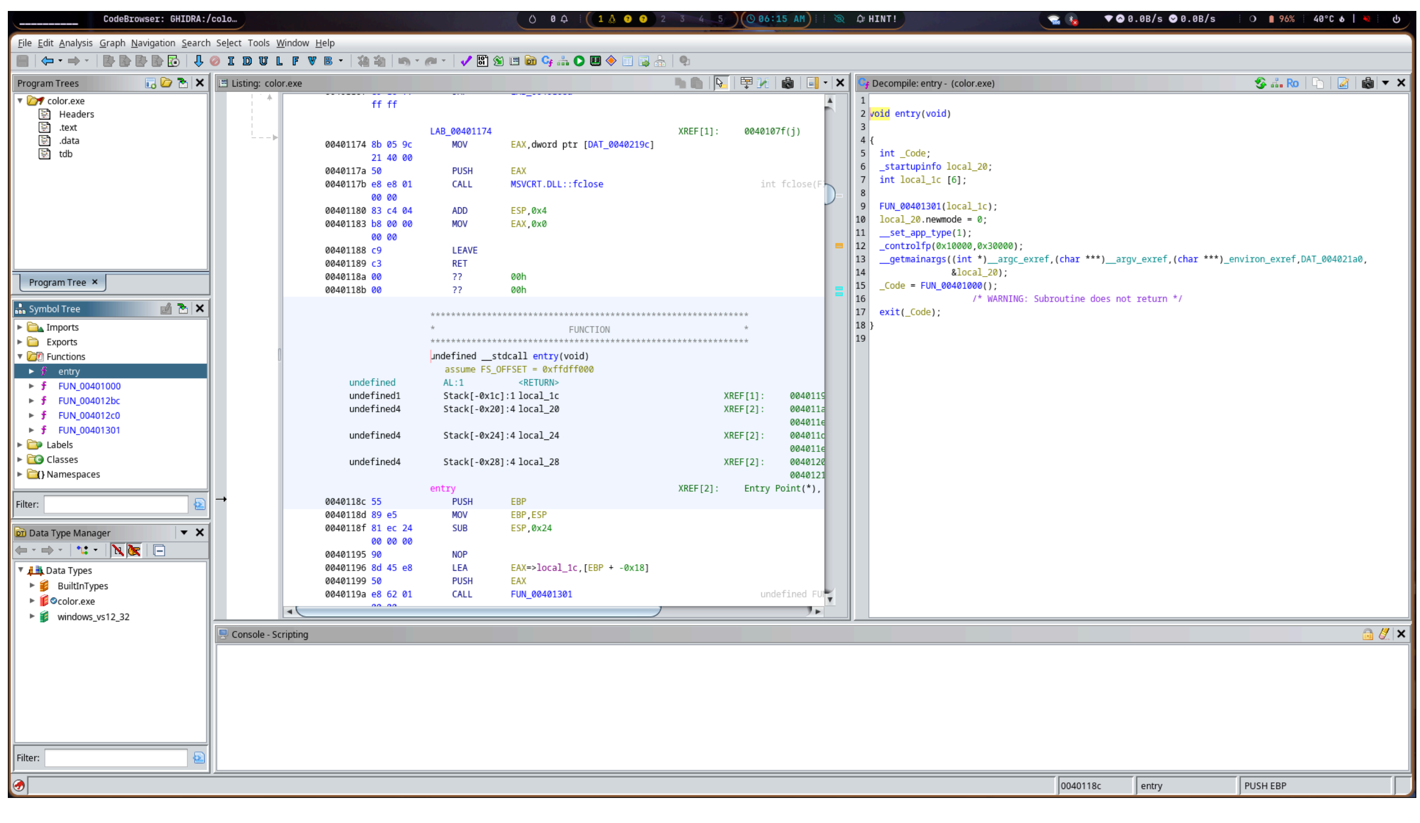
GHIDRA - PRACTICA 1

Integrantes:

- Danilo Romualdo Cruz
- Kevin Jacobo Hernadez
- Anayeli Rodriguez Zepeda

Apertura del GHIDRA

- En esta seccion primero ejecutamos nuestro GHIDRA con el exe proporcionario por el docente



Decompile: entry - (color.exe)

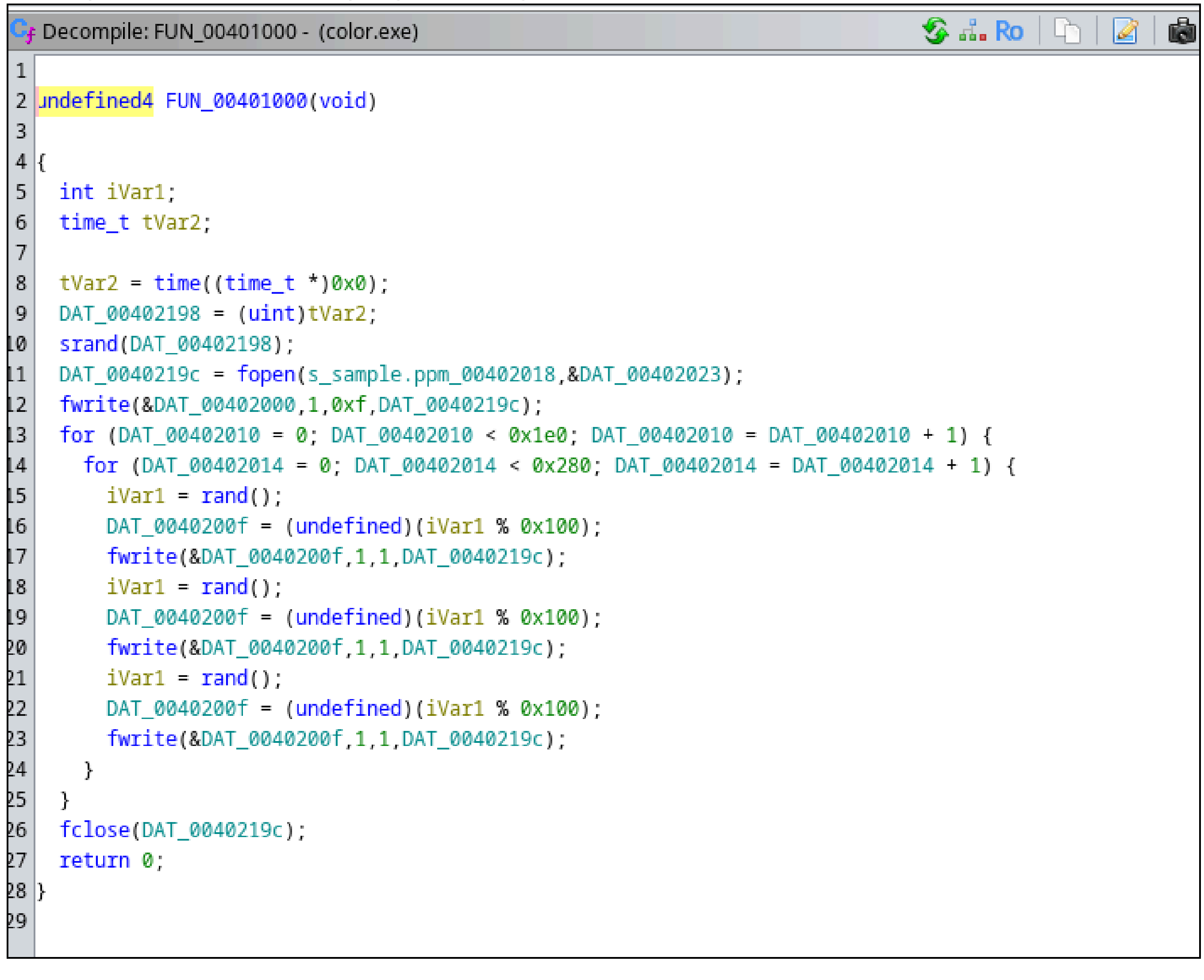
```
void entry(void)

{
  int _Code;
  _startupinfo local_20;
  int local_1c [6];

  FUN_00401301(local_1c);
  local_20.newmode = 0;
  __set_app_type(1);
  _controlfp(0x10000,0x30000);
  __getmainargs((int *)__argc_exref,(char ***)__argv_exref,(char ***)_environ_exref,DAT_004021a0,
                &local_20);
  _Code = FUN_00401000();
                    /* WARNING: Subroutine does not return */
  exit(_Code);
}
```

2. - Buscar las funciones mas relevantes.

- Hay mucha funciones que realmente puede servir o no ya que ghidra solo nos da una interpretacion del codigo en C entonces despues de haber vizualizado las funciones que nos aparecen en su editor.
- Notamos que la funcion FUN_00401000 es la que tiene un contexto en codigo mas explicito de lo que esta haciendo el programa original



Decompile: FUN_00401000 - (color.exe)

```
undefined4 FUN_00401000(void)

{
  int iVar1;
  time_t tVar2;

  tVar2 = time((time_t *)0x0);
  DAT_00402198 = (uint)tVar2;
  srand(DAT_00402198);
  DAT_0040219c = fopen(s_sample.ppm_00402018,&DAT_00402023);
  fwrite(&DAT_00402000,1,0xf,DAT_0040219c);
  for (DAT_00402010 = 0; DAT_00402010 < 0x1e0; DAT_00402010 = DAT_00402010 + 1) {
    for (DAT_00402014 = 0; DAT_00402014 < 0x280; DAT_00402014 = DAT_00402014 + 1) {
      iVar1 = rand();
      DAT_0040200f = (undefined)(iVar1 % 0x100);
      fwrite(&DAT_0040200f,1,1,DAT_0040219c);
      iVar1 = rand();
      DAT_0040200f = (undefined)(iVar1 % 0x100);
      fwrite(&DAT_0040200f,1,1,DAT_0040219c);
      iVar1 = rand();
      DAT_0040200f = (undefined)(iVar1 % 0x100);
      fwrite(&DAT_0040200f,1,1,DAT_0040219c);
    }
  }
  fclose(DAT_0040219c);
  return 0;
}
```

Funciones Adicionales

- Esta secion de la funcion no se ve tan relevante

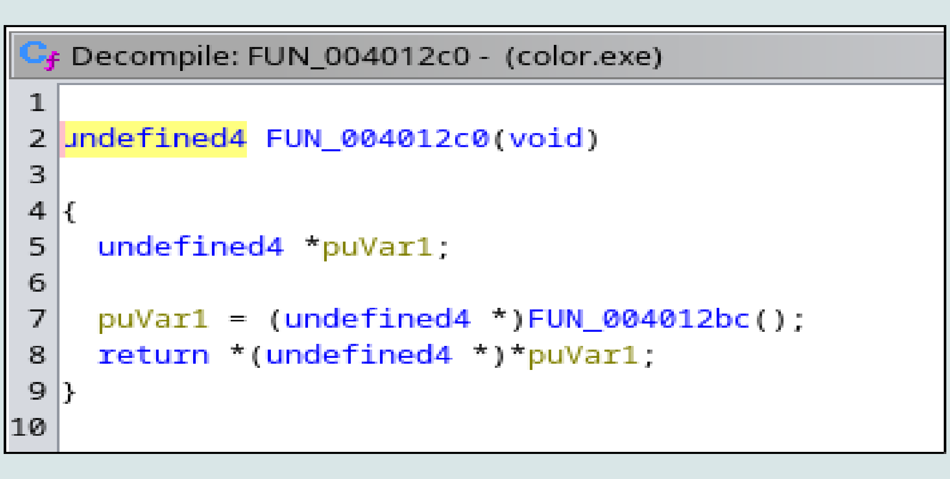
Otras funciones adicionales

Decompile: FUN_004012bc - (color.exe)

```
undefined4 FUN_004012bc(void)

{
  int unaff_EBP;

  return *(undefined4 *)(unaff_EBP + -0x14);
}
```



Decompile: FUN_004012c0 - (color.exe)

```
undefined4 FUN_004012c0(void)

{
  undefined4 *puVar1;

  puVar1 = (undefined4 *)FUN_004012bc();
  return *(undefined4 *)*puVar1;
}
```

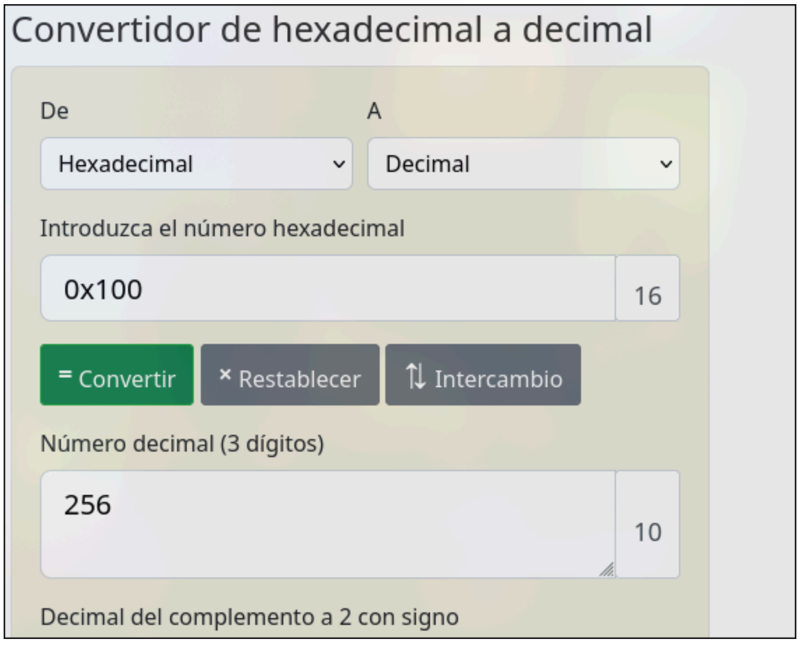
Decompile: FUN_00401301 - (color.exe)

```
void __cdecl FUN_00401301(int *param_1)

{
  *param_1 = (int)&stack0x00000008;
  param_1[1] = 0;
  param_1[2] = (int)ExceptionList;
  param_1[3] = (int)&LAB_004012fc;
  param_1[4] = (int)&DAT_004012f0;
  param_1[5] = 0;
  ExceptionList = param_1 + 2;
  return;
}
```

Conversion de Hexadecimales - Programa en C

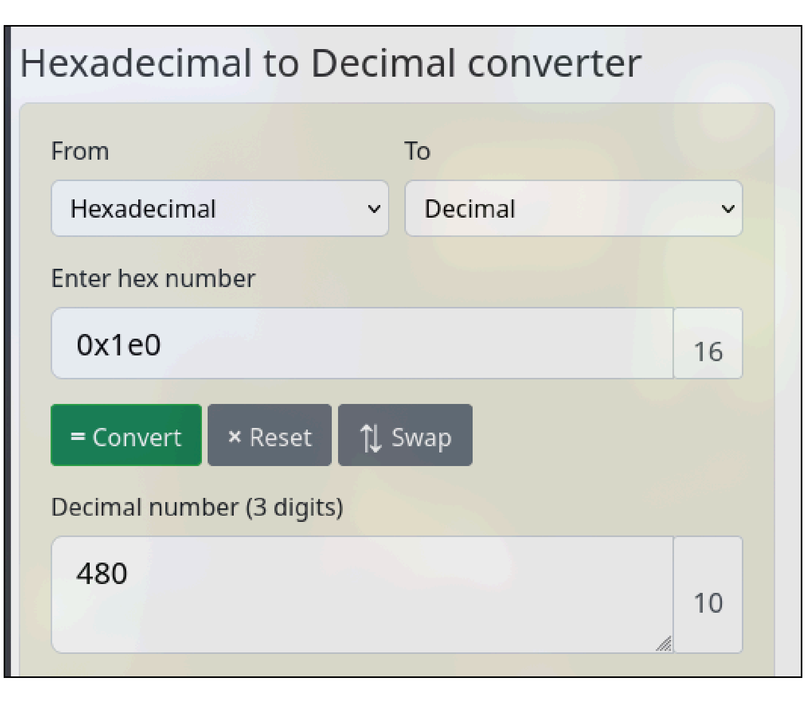
Nota: Hacemos la transformacion de los valores hexadecimales que nos aparecen en el GHIDRA por valores decimales para nuesto codigo en C



Convertidor de hexadecimal a decimal

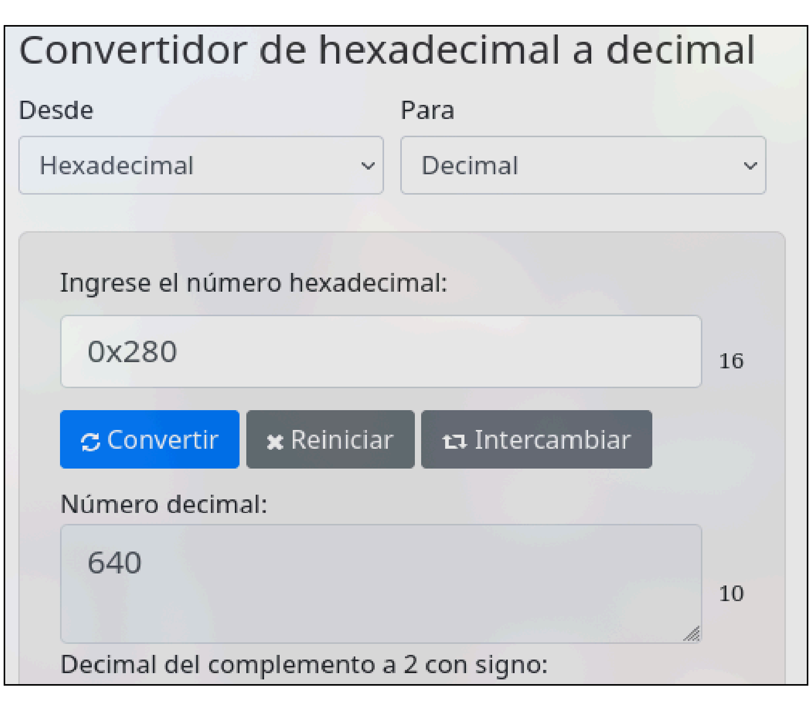
De: Hexadecimal
A: Decimal
Introduzca el número hexadecimal: 0x100 (16)
Número decimal (3 dígitos): 256 (10)
Decimal del complemento a 2 con signo

- Conversion de la siguiente seccion del modelo 0x1e0.
  Nota: Estas convesiones las pase en una pagina web nomas para rectificar.



Hexadecimal to Decimal converter

From: Hexadecimal
To: Decimal
Enter hex number: 0x1e0 (16)
Decimal number (3 digits): 480 (10)



Convertidor de hexadecimal a decimal

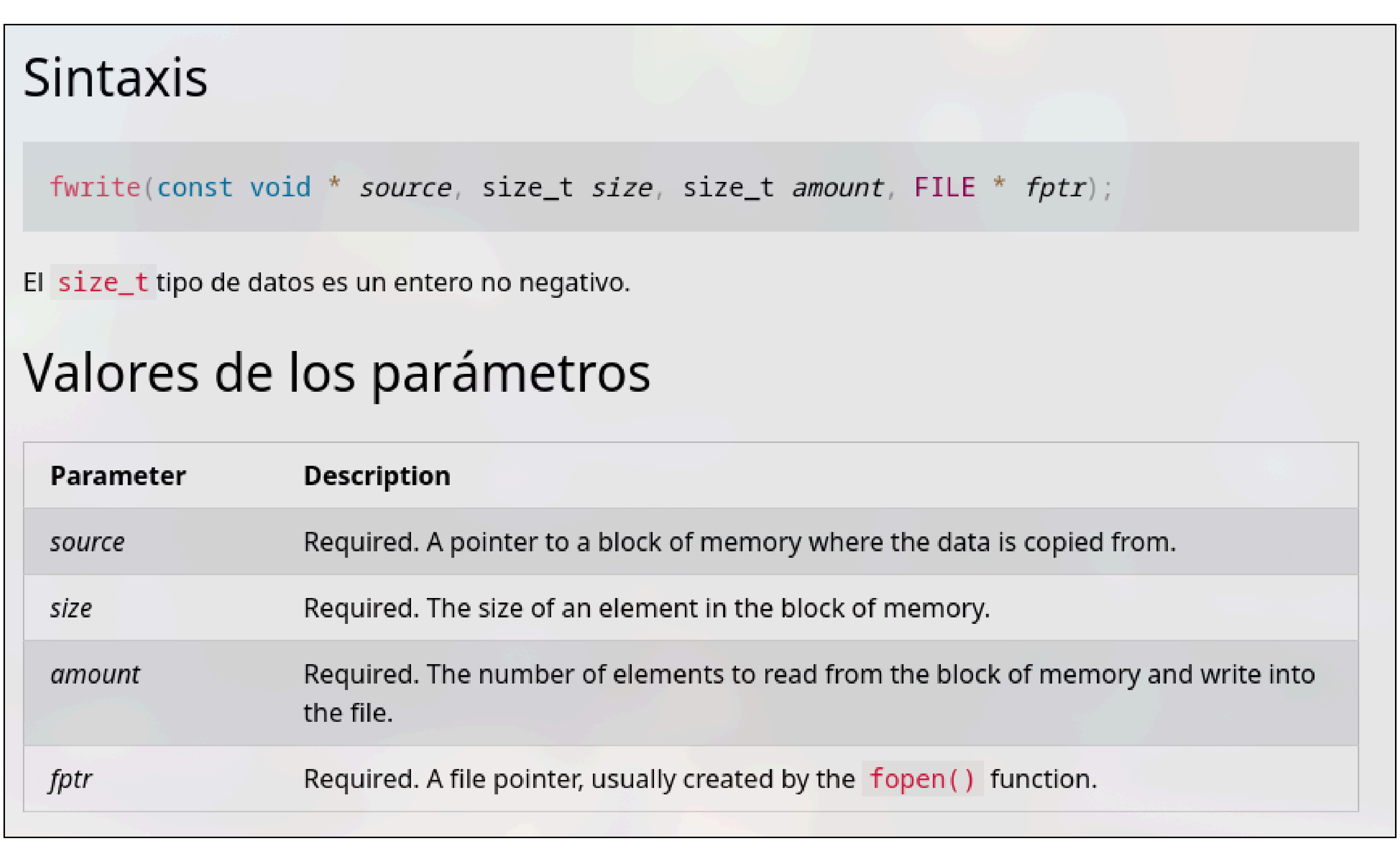
Desde: Hexadecimal
Para: Decimal
Ingrese el número hexadecimal: 0x280 (16)
Número decimal: 640 (10)
Decimal del complemento a 2 con signo:

OBSERBACIONES

Podemos notar que los valores Hexadecimales que nos esta arrojando parecidos a las dimesiones del archivo original asi que podemos deducir que esas son las que debemos de ocupar en nuestro archivo clonado

Investigacion de algunas estructuras de C

- En la FUN_00401000 se encuentra una seccion de codigo que decidimos investiga para entender que elementos recibiria del usuario o para que utilidad tiene en especifico



Sintaxis

```
fwrite(const void * source, size_t size, size_t amount, FILE * fptr);
```

El `size_t` tipo de datos es un entero no negativo.

Valores de los parámetros

Parameter	Description
source	Required. A pointer to a block of memory where the data is copied from.
size	Required. The size of an element in the block of memory.
amount	Required. The number of elements to read from the block of memory and write into the file.
fptr	Required. A file pointer, usually created by the `fopen()` function.

Estructura del Codigo en C

- Procedamos a la elaboracion de la logica del codigo de la creacion de laimagen aleatoria.

```
    // Doble bucle anidado para generar imagen 640x480
    for (int y = 0; y < 480; y++) {           // 0x1e0
        for (int x = 0; x < 640; x++) {       // 0x280
            // Generar R
            iVar1 = rand();
            byte = (unsigned char)(iVar1 % 256);
            fwrite(&byte, 1, 1, file);

            // Generar G
            iVar1 = rand();
            byte = (unsigned char)(iVar1 % 256);
            fwrite(&byte, 1, 1, file);

            // Generar B
            iVar1 = rand();
            byte = (unsigned char)(iVar1 % 256);
            fwrite(&byte, 1, 1, file);
        }
    }
```

Nota: En vase a la transcripcion del codigo que nos da ghidra y la conversion de los valores hexadecimales a decimal que decidimos transformalos para una mejor legibilidad notamos.

- Los ciclos corresponden a la formacion de una matriz por eso necesitamos los valores decimales para identificar las dimeciones que se toman para su creacion.
- se asignan valores aleatorios a las variables que corresponde a un valor sobre el modulo de 256 tal vez en esta parte se refiera a como los programas interpretan las imagenes con valores del 0 al 255
- Estas caracteristicas son las que le dan forma a la imagen aleatoria es la estructura pricipal del codigo

---

Codigo Final.

Nota: Esto ya es la recopilacion de toda la transformacion del codigo a C incluyendo dependecias asi como la declaracion de variables esto de manera general ya que lo mas importante era la logica que hacia la imagen.

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

int FUN_00401000(void);

int main() {
    return FUN_00401000();
}

int FUN_00401000(void) {
    int iVar1;
    time_t tVar2;
    FILE *file;
    unsigned char byte;

    // Obtener tiempo actual y usarlo como semilla
    tVar2 = time(NULL);
    srand((unsigned int)tVar2);

    // Abrir archivo en modo binario para escritura
    file = fopen("s_sample4.ppm", "wb");

    // Escribir cabecera PPM (formato P6)
    fwrite("P6\n640 480\n255\n", 1, 15, file);  // 0xF bytes

    // Doble bucle anidado para generar imagen 640x480
    for (int y = 0; y < 480; y++) {           // 0x1e0
        for (int x = 0; x < 640; x++) {       // 0x280
            // Generar R
            iVar1 = rand();
            byte = (unsigned char)(iVar1 % 256);
            fwrite(&byte, 1, 1, file);

            // Generar G
            iVar1 = rand();
            byte = (unsigned char)(iVar1 % 256);
            fwrite(&byte, 1, 1, file);

            // Generar B
            iVar1 = rand();
            byte = (unsigned char)(iVar1 % 256);
            fwrite(&byte, 1, 1, file);
        }
    }

    // Cerrar archivo
    fclose(file);
```