



Universidad Abierta y a Distancia
de México

División de Ciencias Exactas, Ingeniería y Tecnología
Desarrollo de software

Semestre: Primer semestre.

Unidad didáctica: 01 – Fundamentos de programación.

Unidad de aprendizaje: Unidad 2. Introducción al lenguaje C.

Actividad: Act.3 Aplicación de estructuras de control.

Nombre del estudiante: Orlando Antonio Maturano Pizaña

Matrícula: ES251107915

Grupo: DS-DFPR-2501-B1-013

Figura académica: JOSÉ MANUEL NORIEGA BARRERA

Fecha de entrega: 26 de febrero de 2025

Estado de México, a 24 de febrero del 2025.

Diseño: DL-CPL

ÍNDICE

INTRODUCCIÓN	1
DESARROLLO DE LA ACTIVIDAD	1
Estructuras selectivas	1
Estructuras repetitivas	2
Estructuras anidadas	3
Ánalisis del problema	4
Pseudocódigo	4
Prueba de escritorio	8
Desarrollo del código en lenguaje C	10
CONCLUSIÓN	14
CITAS DE AUTOR	14
REFERENCIAS BIBLIOGRÁFICAS	15

INTRODUCCIÓN

Las estructuras de control son esenciales en programación, ya que permiten dirigir el flujo de ejecución de un programa mediante decisiones y repeticiones. Estas estructuras surgieron como respuesta a la necesidad de resolver problemas complejos en los que los algoritmos deben tomar distintos caminos según los requerimientos de cada programa específico y repetir acciones determinadas hasta cumplir un objetivo. Su importancia radica en que posibilitan la creación de programas dinámicos eficientes y adaptables a diferentes escenarios. En el lenguaje C, estas estructuras se basan en los principios de programación estructurada, lo que favorece la claridad, legibilidad y mantenibilidad del código.

El objetivo de esta actividad es comprender y aplicar estructuras selectivas, repetitivas y anidadas para aprender a diseñar estructuras algorítmicas eficientes. Se abordarán las estructuras if, switch (selectivas), for, while, do-while (repetitivas) y su combinación en estructuras anidadas, exemplificando su uso en el lenguaje C. La actividad se limita al enfoque de la programación con estructuras de control.

DESARROLLO DE LA ACTIVIDAD

Estructuras selectivas

Las estructuras selectivas son construcciones que permiten tomar decisiones durante la ejecución de un programa. Esto significa que, en función del resultado de una o varias

condiciones lógicas, el programa puede ejecutar un bloque de código u otro. Las dos formas principales de estructuras selectivas en lenguaje C son:

- La instrucción if con sus variantes if-else y else-if, que evalúan la condición, y si esta resulta verdadera, ejecuta el bloque del código asociado; de lo contrario puede ejecutar otro bloque definiendo un else.
- La instrucción switch, que evalúa el valor de una expresión y lo compara con varios casos (case), ejecutando el bloque de código correspondiente al caso que coincida, o un bloque por defecto (default) si no hay coincidencia.

Estas estructuras funcionan evaluando expresiones booleanas (donde cero se considera falso y cualquier valor distinto a cero se considera verdadero) y en función del resultado, dirigen el flujo de ejecución a diferentes secciones del código.

A continuación se presenta un ejemplo sustraído de The C Programming Language (2nd ed.) de Kernighan, B. W., & Ritchie, D. M. que representa el uso de una estructura if-else para clasificar caracteres (distinguiendo dígitos, espacios en blanco y otros):

```
#include <stdio.h>

main( ) // Cuenta dígitos, espacios en blanco y otros caracteres

{
    int c, i, nwhite, nother, ndigit[10];

    nwhite = nother = 0;
    for (i = 0; i < 10; ++i)
        ndigit[i] = 0;

    while ((c = getchar()) != EOF)
        if (c >= '0' && c <= '9')
            ++ndigit[c - '0'];
        else if (c == ' ' || c == '\t' || c == '\n')
            ++nwhite;
        else
            ++nother;

    printf("dígitos = ");
    for (i = 0; i < 10; ++i)
        printf(" %d", ndigit[i]);
    printf(", espacios blancos = %d, otros = %d\n", nwhite, nother);
}
```

Estructuras repetitivas

Las estructuras repetitivas o bucles, permiten ejecutar repetidamente un bloque de código mientras se cumpla una condición determinada. Cada uno de los tres bucles principales tiene sus particularidades y aplicaciones:

El bucle for se utiliza cuando se conoce de antemano el número de iteraciones. Permite inicializar un contador, evaluar una condición y actualizar el contador en una sola línea, lo que lo hace ideal para recorrer arreglos o realizar operaciones de conteo.

El bucle while evalúa la condición antes de cada iteración, por lo que si la condición no se cumple desde el inicio, el bloque de código no se ejecuta en absoluto. Es útil en situaciones donde el número de repeticiones depende de condiciones que pueden cambiar dinámicamente durante la ejecución.

El bucle do-while, similar al while, evalúa la condición al final de cada ciclo. Esto garantiza que el bloque de código se ejecute al menos una vez, siendo especialmente adecuado para escenarios donde se requiere una acción inicial antes de comprobar si se debe continuar repitiendo.

Estas estructuras son fundamentales para controlar el flujo de ejecución y adaptar la repetición de tareas según las necesidades del programa.

Estructuras anidadas

Las estructuras anidadas permiten colocar una construcción de control (como if, for o while dentro del cuerpo de otra, lo que posibilita la resolución de problemas complejos mediante la evaluación de múltiples condiciones o la ejecución de subprocessos repetitivos dentro de otros. Por ejemplo, al procesar la entrada del usuario, es común usar un bucle while que recorra cada carácter y en su interior emplear sentencias if-else para clasificar el carácter según su tipo (por ejemplo, si es un espacio un salto de línea o parte de una palabra). De esta forma se combinan dos o más estructuras para lograr un flujo de ejecución más detallado y controlado.

A continuación se presenta un ejemplo extraído de The C Programming Language (2nd ed.) de Kernighan, B. W., & Ritchie, D. M., en el que se cuenta el número de líneas palabras y caracteres de la entrada, utilizando estructuras anidadas (un if dentro de un while y un if-else anidado):

```
#include <stdio.h>
#define IN 1 /* dentro de una palabra */
#define OUT 0 /* fuera de una palabra */

main( ) // Cuenta líneas, palabras y caracteres de la entrada
{
    int c, nl, nw, nc, state;
    state = OUT;
    nl = nw = nc = 0;
    while ((c = getchar()) != EOF) {
        ++nc;
        if (c == '\n')
            if (state == OUT)
                ++nl;
            else
                ++nw;
        state = (c == ' ' || c == '\t' || c == '\r') ? OUT : IN;
    }
}
```

```

    ++nl;
    if (c == ' ' || c == '\n' || c == '\t')
        state = OUT;
    else if (state == OUT) {
        state = IN;
        ++nw;
    }
}
printf("%d %d %d\n", nl, nw, nc);
}

```

Análisis del problema:

El programa debe presentar las opciones de registrar desecho, imprimir boleta y salir mediante un menú interactivo. Al seleccionar “registrar desecho” el programa debe solicitar el nombre del desecho, la categoría de clasificación y el número de piezas. Para que el programa sea útil debe tener una opción para registrar un nuevo desecho y regresar al menú principal.

Al imprimir la boleta de información se deberán imprimir en pantalla los desechos registrados con nombre, categoría y número de piezas, lo ideal sería poder guardar la boleta de información pero solo me centraré en como se describe el programa en el caso de estudio. Al final de la impresión de la boleta de información en pantalla se deberá mostrar el menú principal de nuevo para poder registrar un nuevo desecho para que se añada a la boleta de información con todos los desechos anteriores, también para poder imprimir la boleta y para seleccionar la opción salir la cual finalizará el procesamiento del programa.

Las variables identificadas son:

Nombre del desecho, categoría, número de piezas y opciones del menú.

Pseudocódigo:

Se usarán las siguientes abreviaciones con letras para el pseudocódigo:

N E: Nombre del estudiante.
M: Matrícula.
G: Grupo.
N D: Nombre del desecho.
C: Clasificación.
R: Reciclable.
P: Peligroso.
N R: No reciclable.
NPi: Número de piezas.

Orlando Antonio Maturano Pizana
Matrícula: ES251107915

Inicio // Primero se deben imprimir mis datos:

Imprimir "N E: Orlando Antonio Maturano Pizana."

Imprimir "M; ES251107915"

Imprimir "G; DS-DFPR-2509-B1-013"

// Despues imprimir las opciones.

Imprimir "1. Registrar Desecho"

Imprimir "2. Imprimir Boleta"

Imprimir "3. Salir"

// Funcionamiento de las opciones.

Opción 1:

Pedir ND

Leer ND

Pedir CL

Leer CL

Pedir Pi

Leer Pi

Imprimir "Registro exitoso."

Opción 2:

Imprimir "Boleta de informacion"

Imprimir "ND"

Imprimir "CL"

Orlando Antonio Maturano Pizaniq
ES251107915

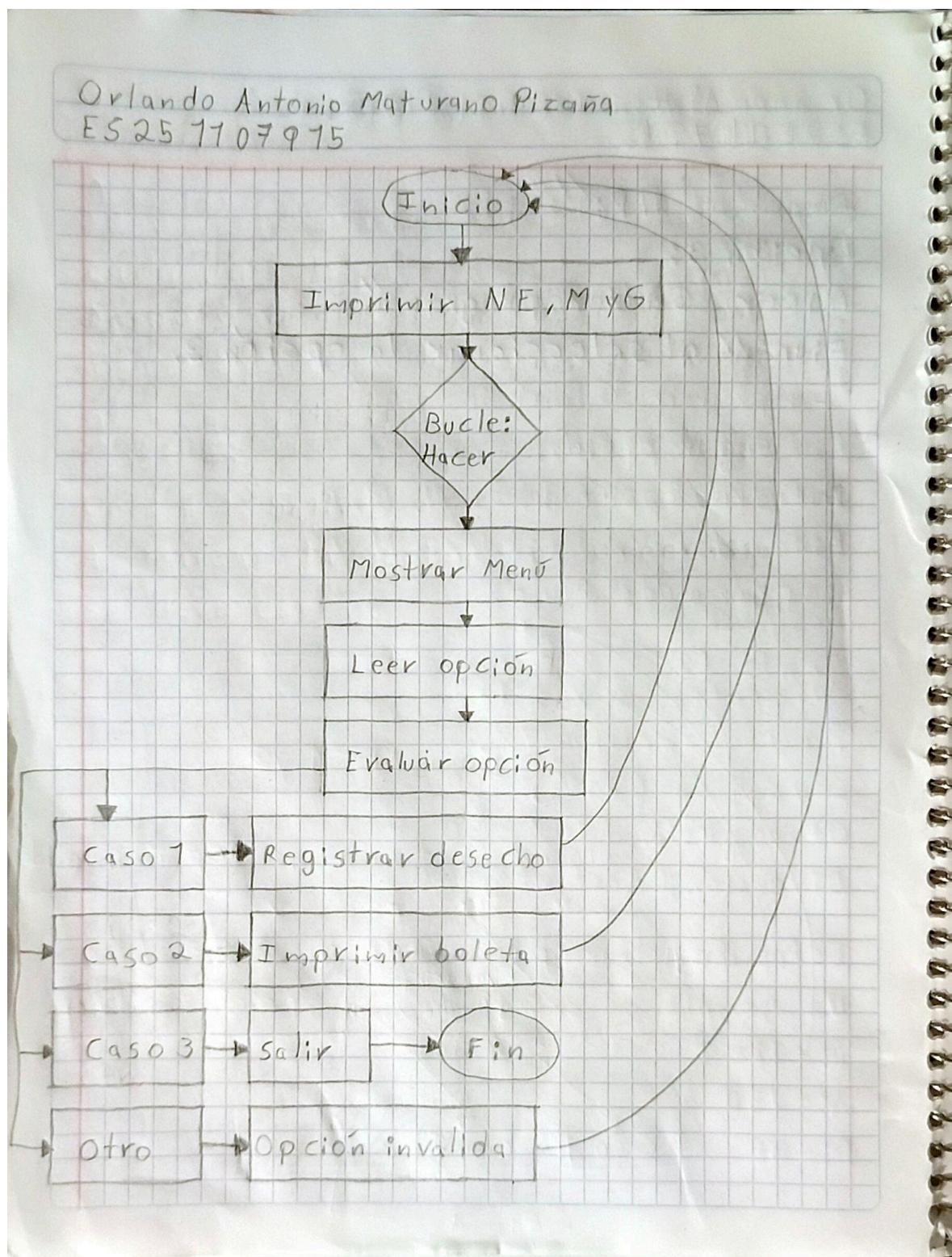
Imprimir "NP;"

Opción 3:

Hacer "Cerrar programa"

Fin // al seleccionar la opción 3.

Diagrama de flujo



Prueba de escritorio

Orlando Antonio Maturano Pizná
ES251107915

Paso 1

Salida esperada

Nombre: Orlando Antonio Maturano Pizná

Matrícula: ES251107915

Grupo: DS-DFPR-2501-B1-013

Paso 2

Salida: Seleccione una opción:

1. Registrar desecho

2. Imprimir boleta

3. Salir

Opción:

Paso 3

Opción 1 Registrar desecho

1 Ingresar nombre

2 Ingresar categoría

3 Ingresar la cantidad de piezas

4 Menú principal

Opción 2:

Imprimir boleta con los datos de la opción 1, se deben imprimir todos los datos sin errores.

Orlando Antonio Maturano Pizarn
ES251107915

Se deben imprimir todos los datos de cada desecho.

Después de imprimir los datos debe aparecer la opción de volver al menú principal.

Paso 4:

Opción 3: Salir

Al marcar la opción 3, el programa se deve cerrar.

Desarrollo del código en lenguaje C

```

Gestor de desechos.c
1 //include <stdio.h>
2 //include <stdlib.h>
3 //include <string.h>
4 #include <locale.h> // Necesaria para configuración regional en sistemas UNIX.
5
6 //Estructura para almacenar los datos de cada desecho electrónico.
7 struct Desecho {
8     char nombre[50]; // Almacena el nombre de cada desecho.
9     char categoria[20]; // Clasificación: Reciclable, Peligroso, No reciclable.
10    int piezas; // Cantidad de unidades del desecho.
11 };
12
13 // Función para leer cadenas de texto de forma segura.
14 void leerCadena(char* buffer, int tamaño) {
15     fgets(buffer, tamaño, stdin); // Lee la entrada del usuario.
16     buffer[strcspn(buffer, "\n")] = '\0'; // Elimina el salto de línea final.
17 }
18
19 int main() {
20     // ===== CONFIGURACIÓN INICIAL =====
21     #ifdef _WIN32
22         system("chcp 65001 > nul"); // Habilita UTF-8 en Windows
23     #else
24         setlocale(LC_ALL, "en_US.UTF-8"); // Configuración regional para UTF-8 en Mac OS, Linux y Solaris.
25     #endif
26
27     // ===== DATOS DEL ESTUDIANTE =====
28     printf("Nombre: Orlando Antonio Maturano Pizáña\n");
29     printf("Matrícula: ES251107915\n");
30     printf("Grupo: DS-DFPR-2501-B1-013\n\n");
31
32     // ===== VARIABLES GLOBALES =====
33     struct Desecho *registros = NULL; // Función dinámica para registros indefinidos.
34     int total = 0; // Contador de desechos registrados.
35     int opcion; // Almacena la opción seleccionada del menú principal.
36     char respuesta[10]; // Buffer para respuestas de Sí/No.
37
38     // ===== BUCLE PRINCIPAL =====
39     do {
40         // ---- Menú de opciones ----
41         printf("\n== MENU PRINCIPAL ==\n");
42         printf("1. Registrar desecho\n");
43         printf("2. Imprimir boleta\n");
44
45         leerCadena(respuesta, 10); // Lee la opción como cadena.
46         opcion = atoi(respuesta); // Convierte a número.
47
48         // ---- Manejo de opciones ----
49         switch(opcion) {
50             //===== REGISTRO DE DESECHOS =====
51             case 1:
52                 do {
53                     // Redimensionar memoria para nuevo registro
54                     struct Desecho *temp = realloc(registros, (total + 1) * sizeof(struct Desecho));
55                     if (!temp) {
56                         printf("\nError: Memoria insuficiente\n");
57                         free(registros);
58                         exit(1);
59                     }
60                     registros = temp;
61
62                     // Captura del nombre:
63                     printf("Nombre del desecho: ");
64                     leerCadena(registros[total].nombre, 50);
65
66                     // Captura de categoría:
67                     printf("Categoría (reciclable/peligroso/no reciclable): ");
68                     leerCadena(registros[total].categoria, 20);
69
70                     // Captura de cantidad de piezas:
71                     printf("Número de piezas: ");
72                     leerCadena(respuesta, 10);
73                     registros[total].piezas = atoi(respuesta);
74
75                     total++; // Incrementa el contador de registros.
76                     printf("\nDesecho registrado.\n");
77                     leerCadena(respuesta, 10);
78                 } while(respuesta[0] == 's' || respuesta[0] == 'S');
79                 break;
80
81             //===== IMPRESIÓN DE BOLETA =====
82             case 2:
83
84         }
85     }
86
87     // Gestor de desechos.c" 106L, 4426B escritos

```

```

Gestor de desechos.c > main()
44     printf("3. Salir\n");
45     printf("Seleccione una opción: ");
46
47     leerCadena(respuesta, 10); // Lee la opción como cadena.
48     opcion = atoi(respuesta); // Convierte a número.
49
50     //---- Manejo de opciones ----
51     switch(opcion) {
52         //===== REGISTRO DE DESECHOS =====
53         case 1:
54             do {
55                 // Redimensionar memoria para nuevo registro
56                 struct Desecho *temp = realloc(registros, (total + 1) * sizeof(struct Desecho));
57                 if (!temp) {
58                     printf("\nError: Memoria insuficiente\n");
59                     free(registros);
60                     exit(1);
61                 }
62                 registros = temp;
63
64                 // Captura del nombre:
65                 printf("Nombre del desecho: ");
66                 leerCadena(registros[total].nombre, 50);
67
68                 // Captura de categoría:
69                 printf("Categoría (reciclable/peligroso/no reciclable): ");
70                 leerCadena(registros[total].categoria, 20);
71
72                 // Captura de cantidad de piezas:
73                 printf("Número de piezas: ");
74                 leerCadena(respuesta, 10);
75                 registros[total].piezas = atoi(respuesta);
76
77                 total++; // Incrementa el contador de registros.
78                 printf("\nDesecho registrado.\n");
79                 leerCadena(respuesta, 10);
80             } while(respuesta[0] == 's' || respuesta[0] == 'S');
81             break;
82
83         //===== IMPRESIÓN DE BOLETA =====
84         case 2:
85
86     }
87
88     // Gestor de desechos.c" 106L, 4426B escritos

```

```

Administrator: C:\WINDOWS > + ^
Gestor de desechos.c *
64     printf("\n==== REGISTRO ===\n");
65
66     // Captura del nombre:
67     printf("Nombre del desecho: ");
68     leerCadena(&registros[total].nombre, 50);
69
70     // Captura de categoría:
71     printf("Categoría (recicitable/peligroso/no recicable): ");
72     leerCadena(&registros[total].categoria, 20);
73
74     // Captura de cantidad de piezas:
75     printf("Número de piezas: ");
76     leerCadena(respuesta, 10);
77     registros[total].piezas = atoi(respuesta);
78
79     total++; // Incrementa el contador de registros.
80     printf("\nDesecho registrado.\n");
81     leerCadena(respuesta, 10);
82 } while(respuesta[0] == 's' || respuesta[0] == 'S');
83 break;
84
85 // ===== IMPRESIÓN DE BOLETA =====
86 case 2:
87     printf("\n==== BOLETA DE INFORMACIÓN\n");
88     for(int i = 0; i < total; i++) { // Recorre todos los registros.
89         printf("\nDesecho %d:\n", i+1);
90         printf("Nombre: %s\n", registros[i].nombre);
91         printf("Categoría: %s\n", registros[i].categoria);
92         printf("Piezas: %d\n", registros[i].piezas);
93     }
94     break;
95
96 // ===== SALIR DEL PROGRAMA =====
97 case 3:
98     printf("\nSaliendo del sistema...\n");
99 }
100 } while(opcion != 3);
101
102 // ===== LIMPIEZA FINAL =====
103 free(registros); // Para liberar memoria asignada dinámicamente.
104 return 0;
105 }

"Gestor de desechos.c" 106L, 4426B escritos
[clangd] - * 2 ① c 106:1 Final/106

```

Al compilarlo con me aparecieron algunos errores:

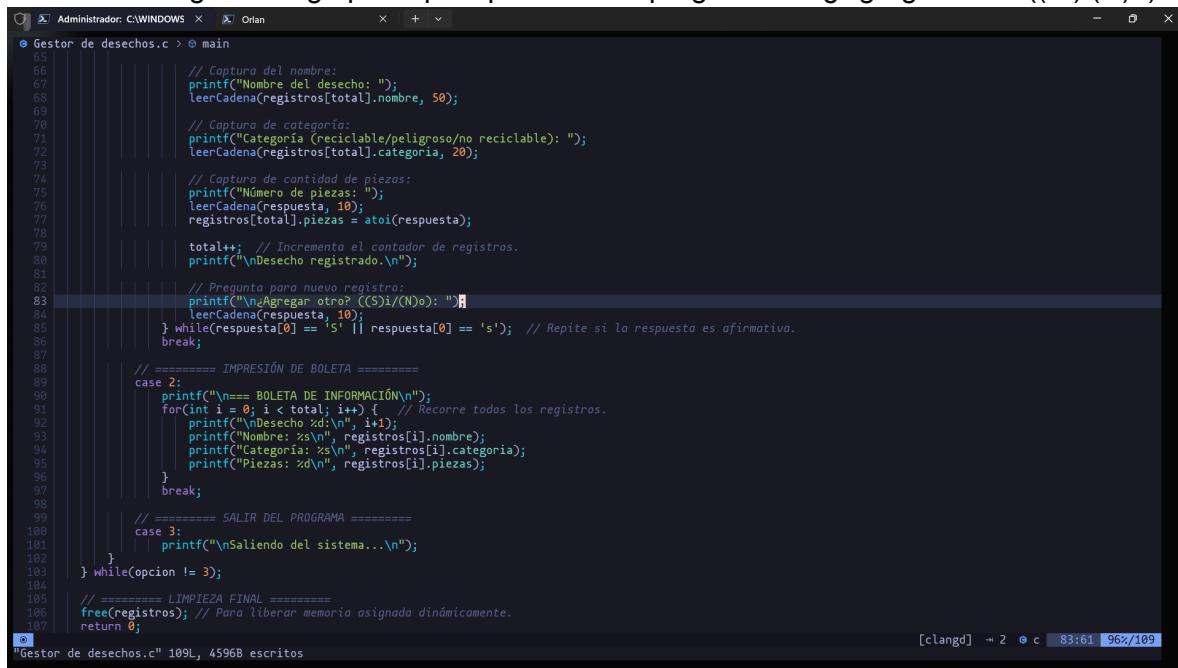
```

Administrator: C:\WINDOWS > Orlan
PowerShell 7.5.0
Loading personal and system profiles took 6754ms.
> clang "Gestor de desechos.c" -o Gestordedeschos.exe
Gestor de desechos.c:9:19: error: expected ';' at end of declaration list
  9 |     char categoria{20};      // Clasificación: Reciclable, Peligroso, No reciclable.
   |     ^
Gestor de desechos.c:72:32: error: incompatible integer to pointer conversion passing 'char' to parameter of type 'char *'; take the address with & [-Wint-conversion]
 72 |             leerCadena(&registros[total].categoria, 20);
   |             ^
Gestor de desechos.c:14:23: note: passing argument to parameter 'buffer' here
 14 |     void leerCadena(char* buffer, int tamaño) {
   |             ^
Gestor de desechos.c:91:48: warning: format specifies type 'char *' but the argument has type 'char' [-Wformat]
 91 |             printf("Categoria: %s\n", registros[i].categoria);
   |             ^
   |             ^
1 warning and 2 errors generated.
< Orlan on Wednesday at 2:54 PM
> { ~ home } *
0.318s ⚡ ⚡ MEM: 67% (5/7GB)

```

Por alguna razón el fallo era el comentario: // Clasificación: Reciclable, Peligroso, no reciclable, de la estructura para almacenar los datos de cada desecho; Tuve que escribir las clasificaciones del comentario empezando por minúsculas. Tal vez fué un error de configuración del editor de código que es neovim con la configuración de lunar vim.

También corregí el código para que apareciera la pregunta de: ¿Aregar otro? ((Si)/(No)):



```

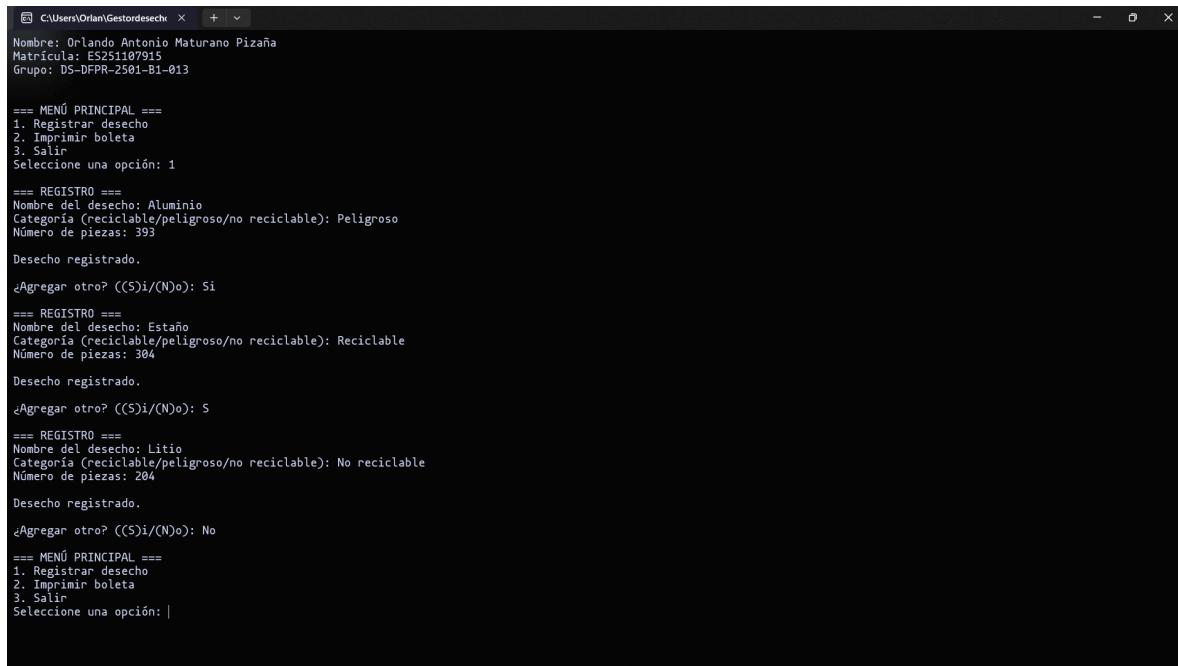
Administrator: C:\WINDOWS > Orian
Gestor de desechos.c > main
65     // Captura del nombre:
66     printf("Nombre del desecho: ");
67     leerCadena(registros[total].nombre, 50);
68
69     // Captura de categoría:
70     printf("Categoría (recicitable/peligroso/no recicitable): ");
71     leerCadena(registros[total].categoria, 20);
72
73     // Captura de cantidad de piezas:
74     printf("Número de piezas: ");
75     leerCadena(respuesta, 10);
76     registros[total].piezas = atoi(respuesta);
77
78     total++; // Incrementa el contador de registros.
79     printf("\nDesecho registrado.\n");
80
81     // Pregunta para nuevo registro:
82     printf("\n¿Aregar otro? ((S)i/(N)o): ");
83     leerCadena(respuesta, 10);
84 } while(respuesta[0] == 's' || respuesta[0] == 'S'); // Repite si la respuesta es afirmativa.
85 break;
86
87 // ===== IMPRESIÓN DE BOLETA =====
88 case 2:
89     printf("\n==== BOLETA DE INFORMACIÓN\n");
90     for(int i = 0; i < total; i++) { // Recorre todos los registros.
91         printf("\nDesecho %d:\n", i+1);
92         printf("Nombre: %s\n", registros[i].nombre);
93         printf("Categoría: %s\n", registros[i].categoria);
94         printf("Piezas: %d\n", registros[i].piezas);
95     }
96     break;
97
98 // ===== SALIR DEL PROGRAMA =====
99 case 3:
100    printf("\nSaliendo del sistema...\n");
101 }
102
103 } while(opcion != 3);
104
105 // ===== LIMPIEZA FINAL =====
106 free(registros); // Para liberar memoria asignada dinámicamente.
107 return 0;

```

Gestor de desechos.c" 109L, 4596B escritos

[clangd] -w 2 o c 83:61 96%:109

Después de las correcciones el programa es funcional:



```

C:\Users\Orian\Gestordesechos > +
Nombre: Orlando Antonio Maturano Pizaña
Matrícula: ES251107915
Grupo: DS-DFPR-2501-B1-013

== MENÚ PRINCIPAL ==
1. Registrar desecho
2. Imprimir boleta
3. Salir
Seleccione una opción: 1

== REGISTRO ==
Nombre del desecho: Aluminio
Categoría (recicitable/peligroso/no recicitable): Peligroso
Número de piezas: 393

Desecho registrado.

¿Aregar otro? ((S)i/(N)o): Si

== REGISTRO ==
Nombre del desecho: Estaño
Categoría (recicitable/peligroso/no recicitable): Recicitable
Número de piezas: 304

Desecho registrado.

¿Aregar otro? ((S)i/(N)o): S

== REGISTRO ==
Nombre del desecho: Litio
Categoría (recicitable/peligroso/no recicitable): No recicitable
Número de piezas: 204

Desecho registrado.

¿Aregar otro? ((S)i/(N)o): No

== MENÚ PRINCIPAL ==
1. Registrar desecho
2. Imprimir boleta
3. Salir
Seleccione una opción: |

```

```
C:\Users\Orlan\Gestordesech... + 

== REGISTRO ==
Nombre del desecho: Estaño
Categoría (recicitable/peligroso/no recicable): Recicitable
Número de piezas: 304
Desecho registrado.
¿Agregar otro? ((S)i/(N)o): S
== REGISTRO ==
Nombre del desecho: Litio
Categoría (recicitable/peligroso/no recicable): No recicitable
Número de piezas: 204
Desecho registrado.
¿Agregar otro? ((S)i/(N)o): No
== MENÚ PRINCIPAL ==
1. Registrar desecho
2. Imprimir boleta
3. Salir
Seleccione una opción: 2
== BOLETA DE INFORMACIÓN
Desecho 1:
Nombre: Aluminio
Categoría: Peligroso
Piezas: 393
Desecho 2:
Nombre: Estaño
Categoría: Recicitable
Piezas: 304
Desecho 3:
Nombre: Litio
Categoría: No recicitable
Piezas: 204
== MENÚ PRINCIPAL ==
1. Registrar desecho
2. Imprimir boleta
3. Salir
Seleccione una opción: |
```

Al indicar que no se quieren registrar más desechos, el programa regresa al menú principal, después se puede volver a seleccionar la opción 1., y el nuevo desecho aparece en la lista de la boleta de información con todos los demás desechos registrados.

```
C:\Users\Orlan\Gestordesech... + 

== MENÚ PRINCIPAL ==
1. Registrar desecho
2. Imprimir boleta
3. Salir
Seleccione una opción: 1
== REGISTRO ==
Nombre del desecho: Aluminio
Categoría (recicitable/peligroso/no recicable): Peligroso
Número de piezas: 392
Desecho registrado.
¿Agregar otro? ((S)i/(N)o): Si
== REGISTRO ==
Nombre del desecho: Estaño
Categoría (recicitable/peligroso/no recicable): Recicitable
Número de piezas: 394
Desecho registrado.
¿Agregar otro? ((S)i/(N)o): No
== MENÚ PRINCIPAL ==
1. Registrar desecho
2. Imprimir boleta
3. Salir
Seleccione una opción: 1
== REGISTRO ==
Nombre del desecho: Mercurio
Categoría (recicitable/peligroso/no recicable): No recicitable
Número de piezas: 393
Desecho registrado.
¿Agregar otro? ((S)i/(N)o): N
== MENÚ PRINCIPAL ==
1. Registrar desecho
2. Imprimir boleta
3. Salir
Seleccione una opción: |
```

Al seleccionar la opción 3. Salir se cierra el programa.

CONCLUSIÓN

El uso estratégico de estructuras de control fue fundamental en la construcción de este prototipo de programa. Mediante un bucle do-while se garantizó que el menú principal se repitiera hasta que el usuario eligiera la opción de salir, asegurando una interacción continua. La estructura switch-case permitió organizar de manera clara y eficiente las tres opciones del sistema, facilitando la escalabilidad del código y evitando complejidades innecesarias.

Dentro del (case 1), se implementó un segundo bucle do-while para permitir múltiples registros consecutivos, demostrando como el anidamiento de estructuras optimiza flujos repetitivos en contextos específicos. Además el uso de condiciones implícitas en el switch (como la validación de opciones inválidas en default) aseguró un manejo adecuado de entradas inesperadas, evitando errores críticos.

Un desafío clave fue equilibrar la lógica de las estructuras para mantener la cohesión del programa. Por ejemplo, la correcta inicialización de variables antes de los bucles y el uso preciso de break en el switch fueron esenciales para evitar comportamientos no deseados, como bucles infinitos o saltos incorrectos entre opciones.

Esta actividad refuerza la relevancia de las estructuras de control en el desarrollo del código. Al automatizar procesos como la clasificación de desechos electrónicos mediante flujos estructurados, se logra precisión y eficiencia.

CITAS DE AUTOR

«C proporciona las construcciones fundamentales de control de flujo que se requieren en programas bien estructurados: agrupación de proposiciones, toma de decisiones (if-else), selección de un caso entre un conjunto de ellos (switch), iteración con la condición de paro en la parte superior (while, for) o en la parte inferior (do), y terminación prematura de ciclos (break).» (Kernighan & Ritchie, 1988, p. 2)

«Cualquier algoritmo puede ser escrito utilizando tres tipos de instrucciones, conocidas como estructuras de control (Kernighan y Ritchie, 1986), las cuales son: secuenciales (cuando se ejecutan una tras otra), selectivas (cuando se ejecutan dependiendo de una condición) y repetitivas (que se ejecutan varias veces en función de una condición).» (Universidad Abierta y a Distancia de México, s.f., p. 41)

«Las estructuras de control en C se dividen en secuenciales, condicionales y repetitivas, lo que permite al programador definir de forma precisa el flujo de ejecución del algoritmo y, en consecuencia, resolver problemas complejos de manera ordenada.» (Ceballos, 2008, p. 35)

REFERENCIAS BIBLIOGRÁFICAS

ISO. (2024). Information technology — Programming languages — C, De:
<https://www.open-std.org/jtc1/sc22/wg14/www/docs/n3220.pdf>

Javier Ceballos Sierra. (2019). C(C++) Curso de programación 5ed. De:
<https://es.scribd.com/doc/9838927/Ceballos-C-C-Curso-de-programacion-5Ed>

Joyanes, L. (2005). Programación en C. Algoritmos, estructuras de datos y objetos. McGraw-Hill, De:
<https://intprog.wordpress.com/wp-content/uploads/2013/08/programacion-en-c-metodologia-algoritmos-y-estructura-de-datos-editorial-mcgraw-hill1.pdf>

Kernighan, B. W., & Ritchie, D. M. (1988). The C Programming Language (2nd ed.). De:

Inglés:
https://www.cimat.mx/ciencia_para_jovenes/bachillerato/libros/%5BKernighan-Ritchie%5DThe_C_Programming_Language.pdf

Español:
https://frrq.cvg.utn.edu.ar/pluginfile.php/13741/mod_resource/content/0/El-lenguaje-de-programacion-C-2-ed-kernighan-amp-ritchie.pdf

UnADM. (s.f.) Unidad 2. Introducción al lenguaje C. De:
https://dmd.unadmexico.mx/contenidos/DCEIT/Compartidas/FPR/U2/descargables/FPR_U_2_Contento.pdf