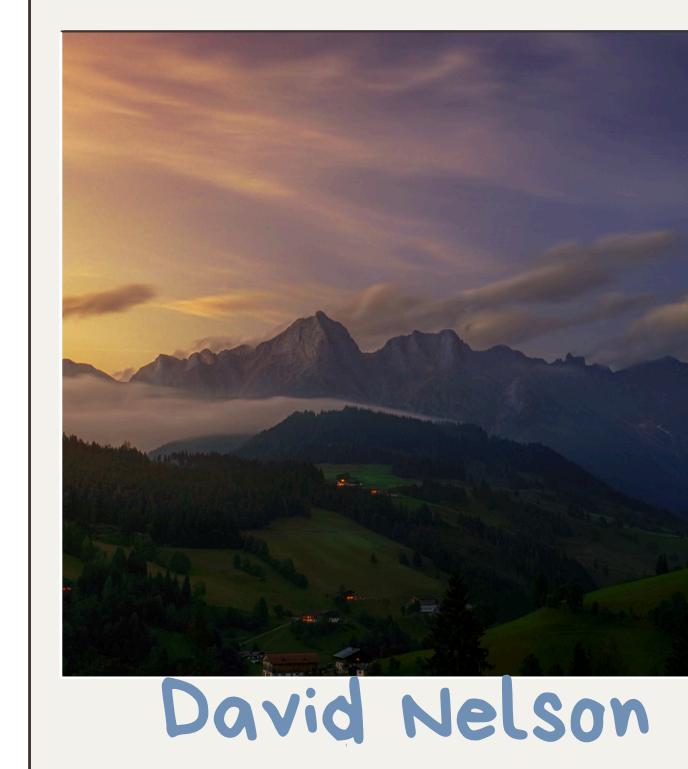


PILA - PLATOS

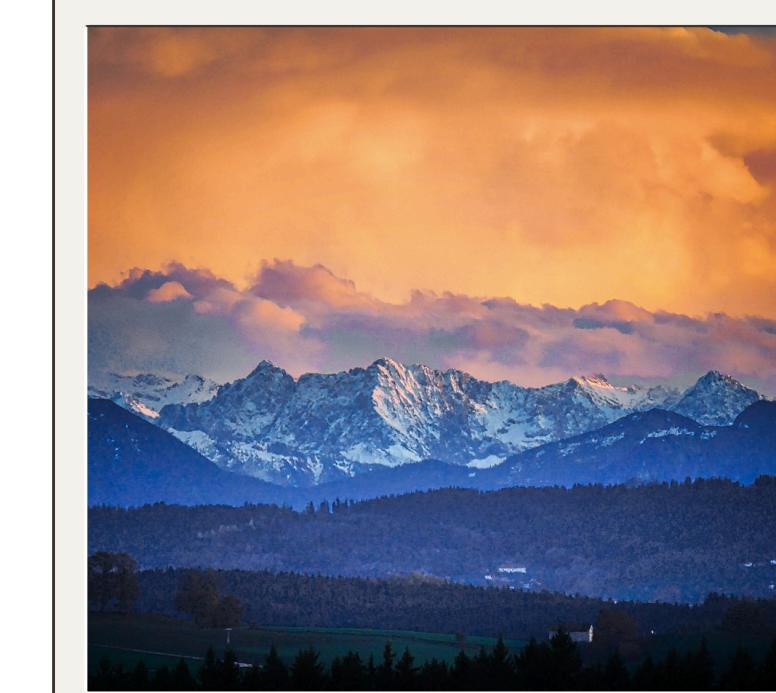
INTEGRANTES



Benavente lozada
Ariana Yaneliz



David Nelson
Chambi Lorocachi



Waldermar Emil
Roque Sullca

INDICE

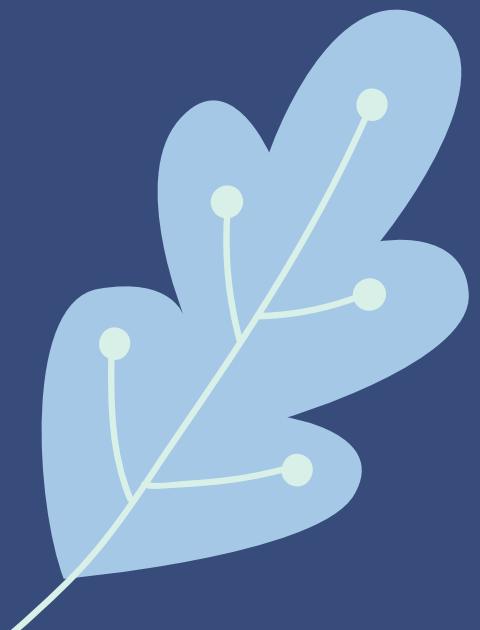
1. Análisis del Problema
2. Diseño de la Solución
3. Solución Final

ANÁLISIS DEL PROBLEMA

1.1 Descripción del Problema

El sistema busca resolver dos objetivos principales:

- Gestión de información mediante una pila: Administrar inserciones, eliminaciones y consultas sobre una colección de datos (en este caso, "platos") usando una estructura LIFO (Last In, First Out).
- Resolución de un problema matemático básico: Calcular el factorial de un número entero no negativo. Para ello, se utiliza una pila de enteros que permite almacenar y procesar los factores de manera ordenada.



1.2 Requerimientos del Sistema

Requerimientos

- **Ingreso de elementos en la pila:** Permitir insertar nuevos datos (platos) o números para el cálculo.
- **Eliminación de elementos:** Eliminar el elemento en la parte superior de la pila.
- **Consulta del elemento superior:** Visualizar el elemento que se encuentra en la parte superior de la pila.
- **Verificación de estado:** Determinar si la pila está vacía.
- **Visualización completa:** Mostrar todos los elementos almacenados.
- **Cálculo de factorial:** Permitir al usuario ingresar un número y calcular su factorial utilizando la pila.
- **Menú interactivo:** Contar con un menú que permita acceder a todas estas funcionalidades de manera intuitiva.

1.3 Estructuras de Datos Propuestas

- Pila mediante Nodos (para platos): Se utiliza una pila implementada con nodos enlazados. Cada nodo contiene un dato (el nombre de un plato) y un puntero al siguiente nodo.
- Pila de Enteros (para el factorial): Se utiliza una pila similar para almacenar los números enteros (factores) que se usarán para calcular el factorial.

1.4 Justificación de la Elección

- Adecuación al Problema: La estructura de pila es idónea para operaciones de tipo LIFO, lo que la hace perfecta para gestionar datos donde la inserción y eliminación deben ocurrir en un extremo específico (la "parte superior").
- Eficiencia: Las operaciones en una pila (push, pop, peek) se realizan en tiempo constante, lo cual es fundamental para aplicaciones que requieran respuesta rápida.

Flexibilidad: La implementación con nodos enlazados permite un manejo dinámico de la memoria, evitando límites fijos en la cantidad de datos que se pueden manejar

DISEÑO DE LA SOLUCIÓN

2.1 Descripción de las Estructuras de Datos y Operaciones

- **Estructura de Pila (nodos):** Cada nodo contiene el dato y un puntero que apunta al siguiente nodo. Se mantiene un puntero denominado "tope" para indicar el último elemento agregado.
- **Operaciones Definidas:**
 - **Ingreso (push):** Agregar un nuevo nodo en la parte superior.
 - **Eliminación (pop):** Remover el nodo en la parte superior.
 - **Consulta (peek):** Mostrar el elemento en la parte superior sin eliminarlo.
 - **Verificación:** Comprobar si la pila está vacía.
 - **Visualización completa y vaciado:** Listar todos los elementos o eliminar todos de la pila.

2.2 Algoritmos Principales

- Pseudocódigo para Agregar un Proceso (Insertar Elemento)

Algoritmo AgregarProceso(Dato):

 Crear un nuevo nodo

 nodo.dato ← Dato

 nodo.siguiente ← tope

 tope ← nodo

 Incrementar el contador de elementos (tamaño)

FinAlgoritm

Pseudocódigo para Cambiar el Estado del Proceso (Ejemplo: Marcar un elemento como procesado o realizar pop)

Algoritmo

CambiarEstadoProceso:

 Si tope es NULL entonces

 Imprimir "La pila está vacía"

 Sino

 Nodo temporal ← tope

 Actualizar tope a

 tope.siguiente

 Liberar memoria del nodo

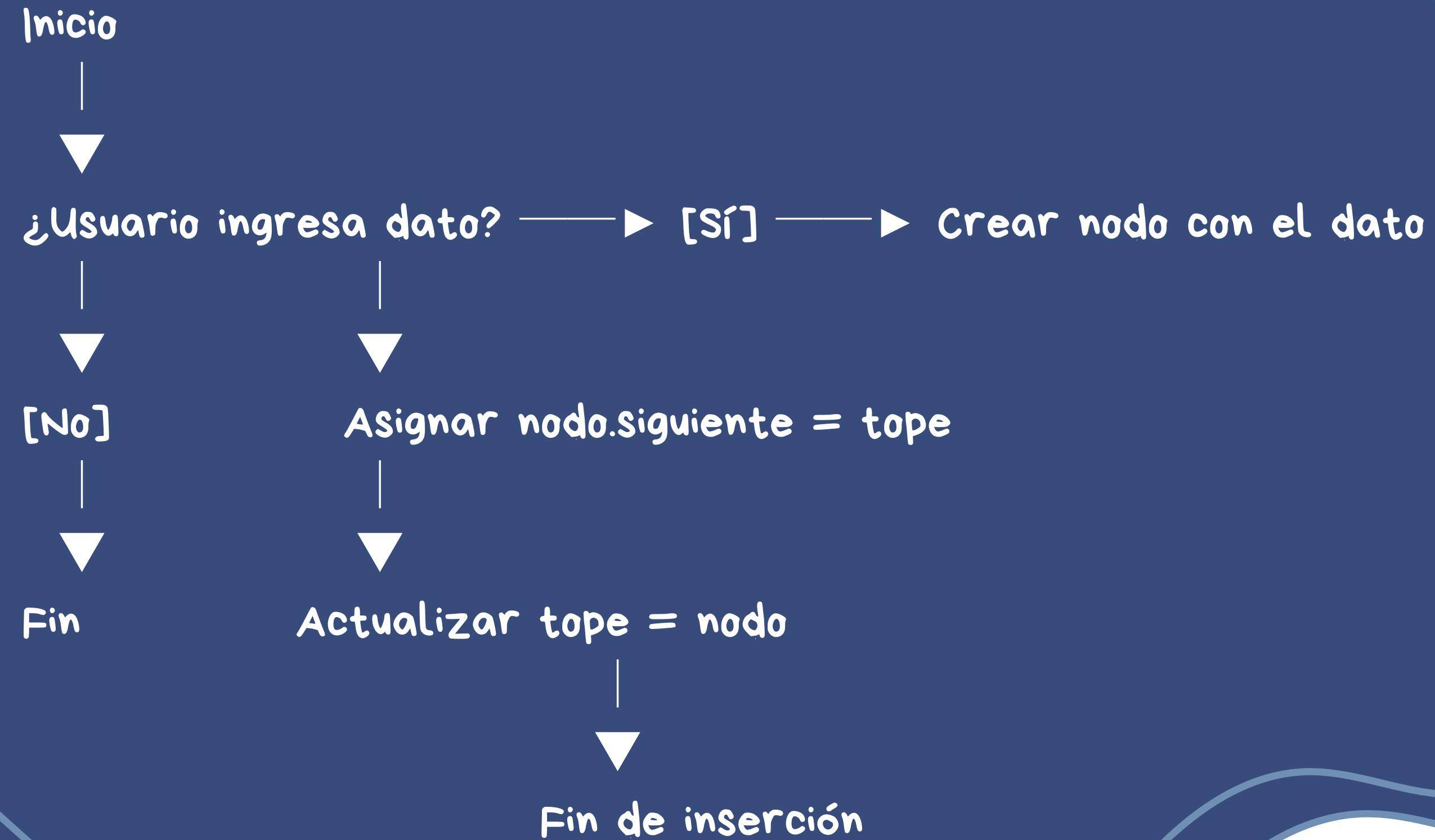
temporal

 Decrementar el contador de elementos (tamaño)

 FinSi

FinAlgoritmo

2.3 Diagramas de Flujo



2.4 Justificación del Diseño

- Ventajas y Eficiencia:
 - La pila permite realizar operaciones de inserción y eliminación en tiempo $O(1)$, lo que garantiza rapidez.
 - El uso de nodos enlazados permite una asignación dinámica de memoria, evitando límites fijos.
- Claridad y Mantenimiento:
 - La modularidad permite separar la lógica de gestión de platos y la resolución del problema matemático, facilitando pruebas y futuras modificaciones.

SOLUCIÓN FINAL

```
1 #include <iostream>
2 #include <string>
3 #include <locale>
4 using namespace std;
5 struct Nodo {
6     string plato;
7     Nodo* siguiente;
8 };
9
10 class Pila {
11 private:
12     Nodo* tope;
13     int tamano;
14 public:
15     Pila() {
16         tope = NULL;
17         tamano = 0;
18     }
19 }
```

#INCLUDE <Iostream>: ESTA LIBRERÍA SE UTILIZA PARA LAS OPERACIONES DE ENTRADA Y SALIDA, COMO USAR COUT PARA MOSTRAR MENSAJES EN LA CONSOLA.

#INCLUDE <String>: PERMITE UTILIZAR LA CLASE STRING, LO CUAL ES ÚTIL CUANDO SE TRABAJA CON TEXTOS (EN ESTE CASO, PARA ALMACENAR EL NOMBRE DE UN PLATO).

#INCLUDE <Locale>: SE INCLUYE PARA CONFIGURAR LA LOCALIZACIÓN DEL PROGRAMA, LO QUE PUEDE AYUDAR A MANEJAR LA SALIDA DE CARACTERES ESPECIALES O ACENTUADOS.

USING NAMESPACE STD: ESTO EVITA TENER QUE ESCRIBIR STD: ANTES DE CADA FUNCIÓN O CLASE QUE PERTENECE AL ESPACIO DE NOMBRES ESTÁNDAR, SIMPLIFICANDO EL CÓDIGO.

- STRUCT NODO: SE DEFINE UNA ESTRUCTURA LLAMADA NODO QUE ACTÚA COMO EL BLOQUE BÁSICO O “NODO” DE LA PILA.
- STRING PLATO: ALMACENA EL DATO DEL NODO, QUE EN ESTE CASO ES EL NOMBRE DEL PLATO.
- NODO* SIGUIENTE: ES UN PUNTERO QUE APUNTARÁ AL SIGUIENTE NODO DE LA PILA. ESTA ES LA FORMA DE ENLAZAR LOS NODOS (ESTRUCTURA DE LISTA ENLAZADA).
- NODO* TOPE: ES UN PUNTERO QUE INDICA EL ELEMENTO QUE ESTÁ EN LA PARTE SUPERIOR DE LA PILA. INICIALMENTE, CUANDO LA PILA ESTÁ VACÍA, ESTE PUNTERO ES NULL.
- INT TAMANO: UN ENTERO QUE LLEVA LA CUENTA DE CUÁNTOS NODOS HAY EN LA PILA. SE INCREMENTA O DECREMENTA CONFORME SE INSERTAN O ELIMINAN ELEMENTOS.
- TOPE = NULL: INDICA QUE LA PILA ESTÁ VACÍA AL INICIO.
- TAMANO = 0: INICIALIZA EL TAMAÑO DE LA PILA EN 0, YA QUE AÚN NO SE HAN INSERTADO ELEMENTOS.

- **CREAR UN NUEVO NODO:** SE RESERVA MEMORIA PARA UN NUEVO NODO MEDIANTE `NEW NODO()` Y SE ALMACENA EN EL PUNTERO `NUEVONODO`.
- **ASIGNAR EL DATO:** SE ASIGNA EL VALOR DEL PARÁMETRO `PLATO` AL CAMPO `PLATO` DEL NODO: `NUEVONODO->PLATO = PLATO;`
- **ENLAZAR EL NODO:** LA LÍNEA `NUEVONODO->SIGUIENTE = TOPE;` HACE QUE EL NUEVO NODO APUNTE A LO QUE ACTUALMENTE ES EL TOPE DE LA PILA. ESTO PRESERVA EL ENLACE CON LOS ELEMENTOS PREVIAMENTE INSERTADOS.
- **ACTUALIZAR EL TOPE:** AL ASIGNAR `TOPE = NUEVONODO;` SE CONVIERTA EL NUEVO NODO EN LA PARTE SUPERIOR DE LA PILA.
- **ACTUALIZAR EL TAMAÑO:** SE INCREMENTA LA VARIABLE `TAMANO` PARA REFLEJAR QUE SE HA AÑADIDO UN ELEMENTO.
- **MENSAJE INFORMATIVO:** SE MUESTRA EN PANTALLA UN MENSAJE INDICANDO QUE EL PLATO HA SIDO INSERTADO.
- **VERIFICAR SI LA PILA ESTÁ VACÍA:** CON `IF (TOPE == NULL)` SE COMPRUEBA QUE NO EXISTAN ELEMENTOS (EN CUYO CASO SE MUESTRA UN MENSAJE INFORMATIVO).
- **GUARDAR EL NODO SUPERIOR:** SE UTILIZA UN PUNTERO AUXILIAR `TEMP` PARA ALMACENAR EL NODO QUE SE VA A ELIMINAR (EL NODO EN EL `TOPE`).
- **MOSTRAR QUÉ SE ELIMINA:** SE IMPRIME EN PANTALLA EL PLATO QUE SE ESTÁ ELIMINANDO.
- **ACTUALIZAR EL TOPE:** `TOPE = TOPE->SIGUIENTE;` ACTUALIZA EL PUNTERO DE LA PARTE SUPERIOR, HACIENDO QUE EL SIGUIENTE NODO SE CONVIERTA EN EL NUEVO `TOPE`.
- **LIBERAR MEMORIA:** SE ELIMINA EL NODO PREVIAMENTE SUPERIOR CON `DELETE TEMP;` PARA EVITAR FUGAS DE MEMORIA.
- **DECREMENTAR EL TAMAÑO:** SE REDUCE EL CONTADOR `TAMANO` EN 1, YA QUE SE HA ELIMINADO UN ELEMENTO.

```
void ingresoP(string plato) {  
    Nodo* nuevoNodo = new Nodo();  
    nuevoNodo->plato = plato;  
    nuevoNodo->siguiente = tope;  
    tope = nuevoNodo;  
    tamano++;  
    cout << "El plato " << plato << " se ha insertado." << endl;  
}  
  
void EliminarP() {  
    if (tope == NULL) {  
        cout << "La pila de platos está vacía." << endl;  
    } else {  
        Nodo* temp = tope;  
        cout << "El plato " << tope->plato << " se ha eliminado." << endl;  
        tope = tope->siguiente;  
        delete temp;  
        tamano--;  
    }  
}
```

```
41     string UltimoE() {
42         if (tope != NULL) {
43             return tope->plato;
44         } else {
45             return "Pila vacía.";
46         }
47     }
48
49     bool Vacia() {
50         return tope == NULL;
51     }
```

- SI LA PILA NO ESTÁ VACÍA (TOPE != NULL), RETORNA EL DATO DEL NODO SUPERIOR (TOPE->PLATO).
- SI LA PILA ESTÁ VACÍA, RETORNA EL MENSAJE "PILA VACÍA".
- TOPE ES UN PUNTERO QUE APUNTA AL ÚLTIMO NODO INSERTADO EN LA PILA.
- LA EXPRESIÓN TOPE == NULL EVALÚA SI EL PUNTERO TOPE ES NULL.
- CUANDO TOPE ES NULL, SIGNIFICA QUE NO HAY NINGÚN NODO EN LA PILA, O Dicho de otro modo, la pila está vacía.
- LA FUNCIÓN RETORNA EL VALOR BOOLEANO :
- RETORNA TRUE SI LA PILA ESTÁ VACÍA (PORQUE TOPE ES NULL).
- RETORNA FALSE SI HAY AL MENOS UN ELEMENTO EN LA PILA (PORQUE TOPE APUNTA A UN NODO).

```

void MostrarP() {
    if (Vacio()) {
        cout << "La pila de platos está vacía." << endl;
    } else {
        cout << "Lista de platos:" << endl;
        Nodo* temp = tope;
        while (temp != NULL) {
            cout << temp->plato << " ";
            temp = temp->siguiente;
        }
        cout << endl;
    }
}

int TamanoPila() {
    return tamano;
}

void VaciarPila() {
    while (!Vacio()) {
        EliminarP();
    }
    cout << "Todos los elementos han sido eliminados." << endl;
}

```

- **CHEQUEA SI LA PILA ESTÁ VACÍA:** SE UTILIZA LA FUNCIÓN VACIO(), QUE RETORNA TRUE SI TOPE ES NULL.
- **SI LA PILA ESTÁ VACÍA, SE IMPRIME UN MENSAJE INFORMANDO "LA PILA DE PLATOS ESTÁ VACÍA".**
- **RECORRE LA PILA:** SI LA PILA CONTIENE ELEMENTOS (ES DECIR, TOPE NO ES NULL), EL MÉTODO:
IMPRIME UN ENCABEZADO "LISTA DE PLATOS".
- DECLARA UN PUNTERO AUXILIAR TEMP QUE INICIALMENTE APUNTA AL ELEMENTO SUPERIOR (TOPE).
- USA UN BUCLE WHILE (TEMP != NULL) PARA RECORRER LA LISTA ENLAZADA. EN CADA ITERACIÓN:
IMPRIME EL CONTENIDO DE TEMP->PLATO (EL DATO DEL NODO).
- ACTUALIZA TEMP PARA APUNTAR AL SIGUIENTE NODO (TEMP = TEMP->SIGUIENTE).
- LA VARIABLE TAMANO SE ACTUALIZA EN CADA OPERACIÓN DE INSERCIÓN Y ELIMINACIÓN.
- SIMPLEMENTE SE RETORNA EL VALOR DE TAMANO, LO QUE PERMITE CONOCER CUÁNTOS NODOS (ELEMENTOS) SE ENCUENTRAN EN LA PILA EN ESE MOMENTO.
- **BUCLE DE ELIMINACIÓN:** SE UTILIZA UN BUCLE WHILE (!VACIO()) QUE CONTINÚA MIENTRAS LA PILA NO ESTÉ VACÍA. DENTRO DEL BUCLE:
SE LLAMA A LA FUNCIÓN ELIMINARP(), QUE ELIMINA EL ELEMENTO EN LA PARTE SUPERIOR DE LA PILA.
- **MENSAJE FINAL:** DESPUÉS DE VACIAR LA PILA, SE MUESTRA UN MENSAJE INDICANDO QUE TODOS LOS ELEMENTOS HAN SIDO ELIMINADOS.

```

78 struct NodoInt {
79     int valor;
80     NodoInt* siguiente;
81 };
82
83 class PilaInt {
84 private:
85     NodoInt* tope;
86 public:
87     PilaInt() {
88         tope = NULL;
89     }
90     void push(int valor) {
91         NodoInt* nuevo = new NodoInt();
92         nuevo->valor = valor;
93         nuevo->siguiente = tope;
94         tope = nuevo;
95     }

```

- **INT VALOR:** ALMACENA EL VALOR ENTERO QUE SE DESEA GUARDAR EN EL NODO.
- **NODOINT* SIGUIENTE:** ES UN PUNTERO QUE APUNTA AL SIGUIENTE NODO EN LA PILA, PERMITIENDO ENLAZAR LOS NODOS DE FORMA SECUENCIAL.
- **PRIVATE:** SOLO ACCESIBLE DENTRO DE LA CLASE PILAINT
- **TOPE:** ES UN PUNTERO QUE SIEMPRE APUNTA AL NODO QUE SE ENCUENTRA EN LA PARTE SUPERIOR DE LA PILA. INICIALMENTE SE ESTABLECE EN NULL YA QUE LA PILA ESTÁ VACÍA.
- **INICIALIZA LA PILA:** ASIGNANDO QUE EL PUNTERO TOPE SEA NULL, LO QUE INDICA QUE LA PILA NO CONTIENE ELEMENTOS.
- **CREACIÓN DEL NUEVO NODO:** SE RESERVA MEMORIA PARA UN NUEVO NODO DE TIPO NODOINT MEDIANTE NEW NODOINT() Y SE LO ASIGNA AL PUNTERO NUEVO.
- **ASIGNACIÓN DEL VALOR:** EL VALOR QUE SE DESEA INSERTAR SE ASIGNA AL CAMPO VALOR DEL NUEVO NODO: NUEVO->VALOR = VALOR;
- **ENLACE DEL NODO:** SE CONFIGURA NUEVO->SIGUIENTE PARA QUE APUNTE AL NODO QUE ACTUALMENTE ES EL TOPE DE LA PILA. ESTO PRESERVA EL ENLACE CON LOS ELEMENTOS PREVIAMENTE INSERTADOS: NUEVO->SIGUIENTE = TOPE;
- **ACTUALIZACIÓN DEL TOPE:** FINALMENTE, EL PUNTERO TOPE SE ACTUALIZA PARA QUE AHORA APUNTE AL NUEVO NODO, COLOCÁNDOLO EN LA PARTE SUPERIOR DE LA PILA: TOPE = NUEVO;

```

int pop() {
    if (tope == NULL) {
        cout << "La pila de enteros está vacía." << endl;
        return 1; // Valor por defecto en caso de error
    } else {
        int val = tope->valor;
        NodoInt* temp = tope;
        tope = tope->siguiente;
        delete temp;
        return val;
    }
}
bool empty() {
    return tope == NULL;
}

```

- **VERIFICACIÓN DE VACÍO:** SE COMPRUEBA SI LA PILA ESTÁ VACÍA MEDIANTE IF (TOPE == NULL).
- SI TOPE ES NULL, SE IMPRIME UN MENSAJE INDICANDO QUE LA PILA ESTÁ VACÍA Y SE RETORNA UN VALOR POR DEFECTO (EN ESTE CASO, 1, COMO SEÑAL DE ERROR).
- **EXTRACCIÓN DEL ELEMENTO SUPERIOR:**
- SI LA PILA NO ESTÁ VACÍA, SE ALMACENA EN LA VARIABLE VAL EL VALOR DEL NODO EN EL TOPE: INT VAL = TOPE->VALOR;
- **ACTUALIZACIÓN DEL TOPE:**
- SE GUARDA EL NODO A ELIMINAR EN UN PUNTERO TEMPORAL TEMP.
- SE ACTUALIZA EL PUNTERO TOPE PARA QUE APUNTE AL SIGUIENTE NODO DE LA PILA (TOPE = TOPE->SIGUIENTE).
- **LIBERACIÓN DE MEMORIA:**
- SE UTILIZA DELETE TEMP: PARA LIBERAR LA MEMORIA OCUPADA POR EL NODO QUE ESTABA EN EL TOPE.
- **RETORNO DEL VALOR:**
- FINALMENTE, SE RETORNA EL VALOR EXTRAÍDO DEL NODO.
- LA FUNCIÓN EVALÚA LA CONDICIÓN TOPE == NULL.
- SI TOPE ES NULL, SIGNIFICA QUE NO HAY ELEMENTOS EN LA PILA, RETORNANDO TRUE.
- SI TOPE NO ES NULL, LA PILA CONTIENE UNO O MÁS ELEMENTOS, RETORNANDO FALSE.

```

113 void resolverFactorial() {
114     int n;
115     cout << "\n--- Resolver Problema Matemático: Calcular Factorial ---" << endl;
116     cout << "Problema: Calcular el factorial de un número entero n (n!)." << endl;
117     cout << "Definición: n! = n * (n-1) * (n-2) * ... * 1, siendo 0! = 1." << endl;
118     cout << "Ingrese un número entero no negativo: ";
119     cin >> n;
120
121     // Verificación básica e interpretación del input.
122     if (n < 0) {
123         cout << "\nInterpretación: El número ingresado (" << n << ") es negativo." << endl;
124         cout << "El factorial está definido solo para enteros no negativos." << endl;
125         return;
126     }
127
128     cout << "\nInterpretación: Ha ingresado n = " << n << ". Procederemos a calcular " << n << "!" << endl;
129     cout << "Paso 1: La pila se utilizará para almacenar los números del 1 hasta " << n << "." << endl;
130
131     PilaInt pila;
132     for (int i = 1; i <= n; i++) {
133         pila.push(i);
134         cout << "Interpretación: Se agrega el factor " << i << " a la pila." << endl;
135     }
136

```

- **DECLARACIÓN DE VARIABLE:** SE DECLARA LA VARIABLE N PARA ALMACENAR EL NÚMERO CUYO FACTORIAL SE DESEA CALCULAR.
- **MENSAJES INFORMATIVOS:** SE IMPRIMEN VARIAS LÍNEAS QUE EXPLICAN EL PROBLEMA:
 - SE PRESENTA UN TÍTULO INTRODUCTORIO.
 - SE DEFINE QUÉ ES EL FACTORIAL Y SE REMARCA QUE 0! ES IGUAL A 1.
 - SE SOLICITA AL USUARIO QUE INGRESE UN NÚMERO ENTERO NO NEGATIVO.
 - **CHEQUEO DE NEGATIVIDAD:** SE COMPRUEBA SI EL NÚMERO ES NEGATIVO.
 - **SIN N ES MENOR QUE 0, SE INFORMA AL USUARIO QUE EL FACTORIAL SÓLO ESTÁ DEFINIDO PARA ENTEROS NO NEGATIVOS.**
 - **SE USA RETURN:** PARA SALIR DE LA FUNCIÓN SIN CONTINUAR EL CÁLCULO, YA QUE LA ENTRADA NO ES VÁLIDA PARA EL PROBLEMA.
- **MENSAJE DE CONFIRMACIÓN:** SE NOTIFICA AL USUARIO QUE EL NÚMERO INGRESADO ES CORRECTO Y SE ANUNCIA EL INICIO DEL CÁLCULO.
- **USO DE LA PILA PARA ALMACENAR FACTORES:** SE DECLARA UN OBJETO PILA DE LA CLASE PILAINT. LUEGO, MEDIANTE UN BUCLE FOR QUE VA DESDE 1 HASTA N, SE:
 - **INSERTA CADA NÚMERO (FACTOR) EN LA PILA:** CON EL MÉTODO PUSH(), CADA ENTERO SE COLOCA EN LA PARTE SUPERIOR DE LA PILA.
 - **IMPRIME INFORMES DE PROGRESO:** CADA VEZ QUE SE AGREGA UN NÚMERO, SE MUESTRA UN MENSAJE INTERPRETATIVO QUE INDICA EL FACTOR QUE SE ACABA DE INSERTAR.

```

5
6     long long resultado = 1;
7     cout << "\nPaso 2: Se extraen los factores de la pila para multiplicarlos:" << endl;
8     while (!pila.empty()) {
9         int factor = pila.pop();
10        cout << "Interpretación: Se extrae el factor " << factor << " de la pila." << endl;
11        resultado *= factor;
12        cout << "Interpretación: Resultado intermedio = " << resultado << endl;
13    }
14
15    cout << "\nInterpretación: El factorial de " << n << " (n!) es: " << resultado << endl;
16}

```

- **INICIALIZACIÓN DEL ACUMULADOR:** LA VARIABLE RESULTADO SE INICIALIZA EN 1. ESTE VALOR SERVIRÁ PARA MULTIPLICAR CADA FACTOR EXTRAÍDO.
- **EXTRACCIÓN Y MULTIPLICACIÓN:**
- **SE USA UN BUCLE WHILE COMBINADO CON EL MÉTODO EMPTY() PARA CONTINUAR EXTRAYENDO ELEMENTOS HASTA QUE LA PILA ESTÉ VACÍA.**
- **EN CADA ITERACIÓN:**
- **EXTRACTO DEL FACTOR:** SE LLAMA A PILA.POP(), QUE SACA EL ELEMENTO EN LA PARTE SUPERIOR DE LA PILA.
- **MENSAJE DE PROGRESO:** SE INFORMA QUÉ FACTOR SE HA EXTRAÍDO.
- **CÁLCULO INTERMEDIO:** EL FACTOR EXTRAÍDO SE MULTIPLICA CON EL VALOR ACUMULADO EN RESULTADO PARA ACTUALIZARLO.
- **SE IMPRIME EL RESULTADO INTERMEDIO PARA MOSTRAR EL AVANCE DEL CÁLCULO.**
- **PRESENTACIÓN DEL RESULTADO:** UNA VEZ QUE SE HAN EXTRAÍDO Y MULTIPLICADO TODOS LOS FACTORES, SE IMPRIME EN PANTALLA EL RESULTADO FINAL DEL CÁLCULO DEL FACTORIAL.

```
148 int main() {
149     setlocale(LC_CTYPE, "Spanish");
150
151     Pila pilaPlatos;
152     int opcion;
153     string plato;
154
155     do {
156         cout << "\n==== MENÚ PRINCIPAL ===" << endl;
157         cout << "1. Insertar plato en la pila" << endl;
158         cout << "2. Eliminar plato de la pila" << endl;
159         cout << "3. Ver plato superior" << endl;
160         cout << "4. Verificar si la pila de platos está vacía" << endl;
161         cout << "5. Mostrar todos los platos" << endl;
162         cout << "6. Mostrar tamaño de la pila de platos" << endl;
163         cout << "7. Eliminar todos los platos de la pila" << endl;
164         cout << "8. Resolver problema matemático: Calcular factorial" << endl;
165         cout << "9. Salir" << endl;
166         cout << "Ingrese una opción: ";
167         cin >> opcion;
168     }
```

- **PILA PILAPLATOS:** SE CREA UN OBJETO PILAPLATOS DE LA CLASE PILA, QUE GESTIONARÁ LOS PLATOS MEDIANTE LA ESTRUCTURA LIFO (LAST IN, FIRST OUT).
- **INT OPCION:** VARIABLE QUE ALMACENARÁ LA OPCIÓN QUE EL USUARIO SELECCIONE EN EL MENÚ.
- **STRING PLATO:** VARIABLE PARA ALMACENAR EL NOMBRE DEL PLATO QUE EL USUARIO QUIERA INSERTAR EN LA PILA.
- SE MUESTRA UN MENÚ DE OPCIONES EN PANTALLA QUE PERMITE AL USUARIO ELEGIR ENTRE INSERTAR O ELIMINAR PLATOS, CONSULTAR EL TOPE, VERIFICAR LA PILA, MOSTRAR EL TAMAÑO Y VACIARLA, RESOLVER EL PROBLEMA DEL FACTORIAL O SALIR DEL PROGRAMA.

```

switch (opcion) {
    case 1:
        cout << "Ingrese el nombre del plato: ";
        cin >> plato;
        pilaPlatos.ingresoP(plato);
        break;
    case 2:
        pilaPlatos.EliminarP();
        break;
    case 3:
        cout << "Plato superior: " << pilaPlatos.UltimoE() << endl;
        break;
    case 4:
        cout << (pilaPlatos.Vacio() ? "La pila de platos está vacía." : "La pila de platos tiene elem
        break;
    case 5:
        pilaPlatos.MostrarP();
        break;
    case 6:
        cout << "Tamaño de la pila de platos: " << pilaPlatos.TamanoPila() << endl;
        break;
    case 7:
        pilaPlatos.VaciarPila();
        break;
    case 8:
        resolverFactorial();
        break;
    case 9:
        cout << "Saliendo del programa..." << endl;
        break;
    default:
        cout << "Opción no válida. Intente nuevamente." << endl;
}
} while (opcion != 9);

return 0;

```

- EL BUCLE DO-WHILE SE EJECUTA REPETIDAMENTE MIENTRAS LA OPCIÓN ESCOGIDA POR EL USUARIO NO SEA 9. CUANDO EL USUARIO INGRESA 9, EL PROGRAMA SALE DEL BUCLE Y FINALIZA SU EJECUCIÓN.

CASE 1: PIDE QUE EL USUARIO INGRESE EL NOMBRE DE UN PLATO Y LUEGO LO INSERTA EN LA PILA USANDO PILAPLATOS.INGRESOP(PLATO).

CASE 2: LLAMA A PILAPLATOS.ELIMINARP() PARA ELIMINAR EL PLATO QUE SE ENCUENTRA EN LA PARTE SUPERIOR DE LA PILA.

CASE 3: MUESTRA EL PLATO QUE ESTÁ EN LA CIMA DE LA PILA MEDIANTE PILAPLATOS.ULTIMOE().

CASE 4: VERIFICA SI LA PILA ESTÁ VACÍA LLAMANDO A PILAPLATOS.VACIO(). UTILIZA EL OPERADOR TERNARIO PARA IMPRIMIR UN MENSAJE ADECUADO ("LA PILA DE PLATOS ESTÁ VACÍA." O "LA PILA DE PLATOS TIENE ELEMENTOS.").

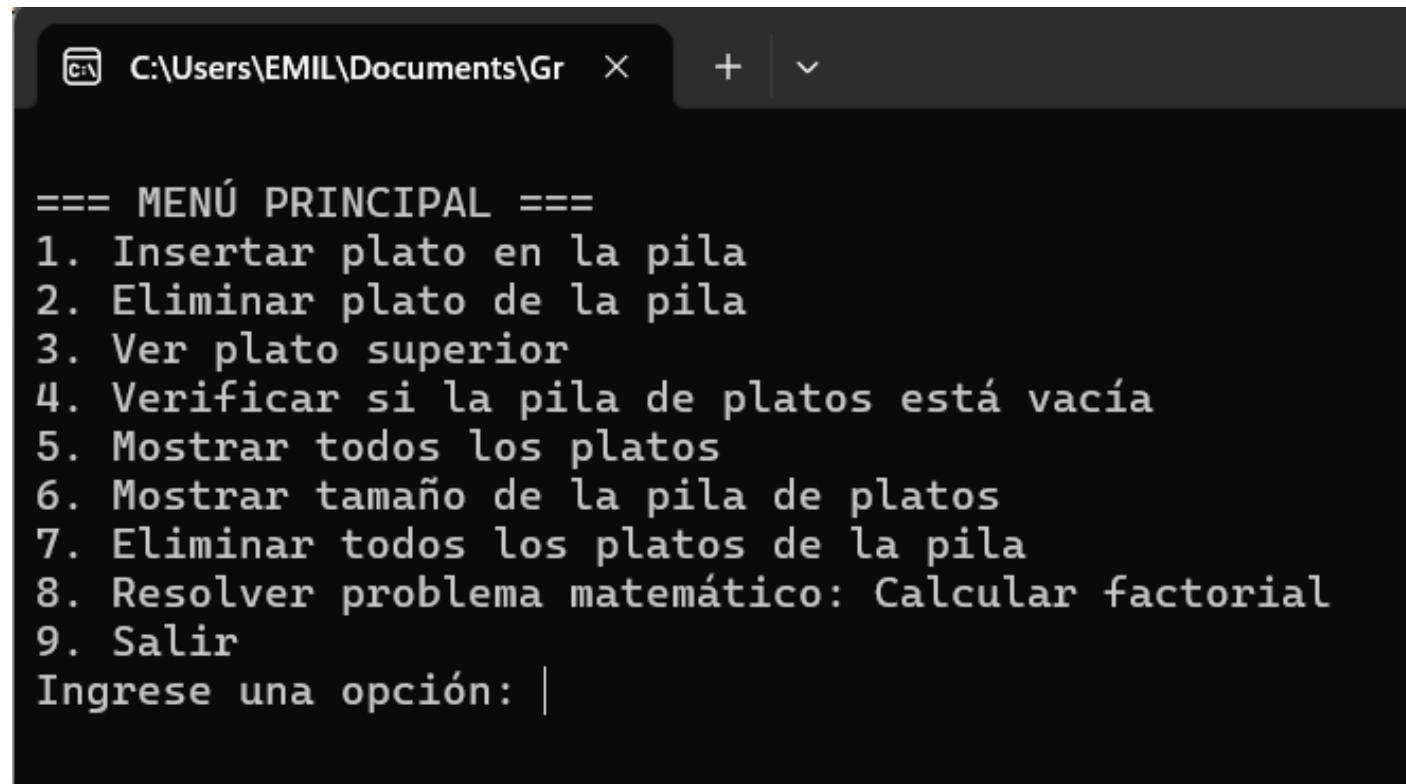
CASE 5: MUESTRA TODOS LOS PLATOS ALMACENADOS EN LA PILA CON PILAPLATOS.MOSTRARP().

CASE 6: IMPRIME EL TAMAÑO ACTUAL DE LA PILA UTILIZANDO PILAPLATOS.TAMANOPILA().

CASE 7: VACÍA COMPLETAMENTE LA PILA MEDIANTE LA LLAMADA PILAPLATOS.VACIARPILA().

- **CASE 8:** LLAMA A LA FUNCIÓN RESOLVERFACTORIAL(), QUE CALCULA EL FACTORIAL DE UN NÚMERO USANDO UNA PILA DE ENTEROS.
- **CASE 9:** MUESTRA UN MENSAJE DE SALIDA Y FINALIZA EL PROGRAMA.
- **DEFAULT:** SI EL USUARIO INGRESA UNA OPCIÓN INVÁLIDA, SE MUESTRA UN MENSAJE INDICÁNDOLE.

CAPTURAS DE PANTALLA



== MENÚ PRINCIPAL ==
1. Insertar plato en la pila
2. Eliminar plato de la pila
3. Ver plato superior
4. Verificar si la pila de platos está vacía
5. Mostrar todos los platos
6. Mostrar tamaño de la pila de platos
7. Eliminar todos los platos de la pila
8. Resolver problema matemático: Calcular factorial
9. Salir
Ingresé una opción: |

```
== MENÚ PRINCIPAL ==  
1. Insertar plato en la pila  
2. Eliminar plato de la pila  
3. Ver plato superior  
4. Verificar si la pila de platos está vacía  
5. Mostrar todos los platos  
6. Mostrar tamaño de la pila de platos  
7. Eliminar todos los platos de la pila  
8. Resolver problema matemático: Calcular factorial  
9. Salir  
Ingresé una opción: 1  
Ingresé el nombre del plato: te  
El plato te se ha insertado.
```

CAPTURAS DE PANTALLA

== MENÚ PRINCIPAL ==

1. Insertar plato en la pila
2. Eliminar plato de la pila
3. Ver plato superior
4. Verificar si la pila de platos está vacía
5. Mostrar todos los platos
6. Mostrar tamaño de la pila de platos
7. Eliminar todos los platos de la pila
8. Resolver problema matemático: Calcular factorial
9. Salir

Ingresé una opción: 2

El plato te se ha eliminado.

CAPTURAS DE PANTALLA

```
== MENÚ PRINCIPAL ==
1. Insertar plato en la pila
2. Eliminar plato de la pila
3. Ver plato superior
4. Verificar si la pila de platos está vacía
5. Mostrar todos los platos
6. Mostrar tamaño de la pila de platos
7. Eliminar todos los platos de la pila
8. Resolver problema matemático: Calcular factorial
9. Salir
```

Ingrese una opción: 8

--- Resolver Problema Matemático: Calcular Factorial ---

Problema: Calcular el factorial de un número entero n ($n!$).

Definición: $n! = n * (n-1) * (n-2) * \dots * 1$, siendo $0! = 1$.

Ingrese un número entero no negativo: 5

Interpretación: Ha ingresado $n = 5$. Procederemos a calcular $5!$

Paso 1: La pila se utilizará para almacenar los números del 1 hasta 5.

Interpretación: Se agrega el factor 1 a la pila.

Interpretación: Se agrega el factor 2 a la pila.

Interpretación: Se agrega el factor 3 a la pila.

Interpretación: Se agrega el factor 4 a la pila.

Interpretación: Se agrega el factor 5 a la pila.

Paso 2: Se extraen los factores de la pila para multiplicarlos:

Interpretación: Se extrae el factor 5 de la pila.

Interpretación: Resultado intermedio = 5

Interpretación: Se extrae el factor 4 de la pila.

Interpretación: Resultado intermedio = 20

Interpretación: Se extrae el factor 3 de la pila.

Interpretación: Resultado intermedio = 60

Interpretación: Se extrae el factor 2 de la pila.

Interpretación: Resultado intermedio = 120

Interpretación: Se extrae el factor 1 de la pila.

Interpretación: Resultado intermedio = 120

Interpretación: El factorial de 5 ($n!$) es: 120



Gracias