CUCEI

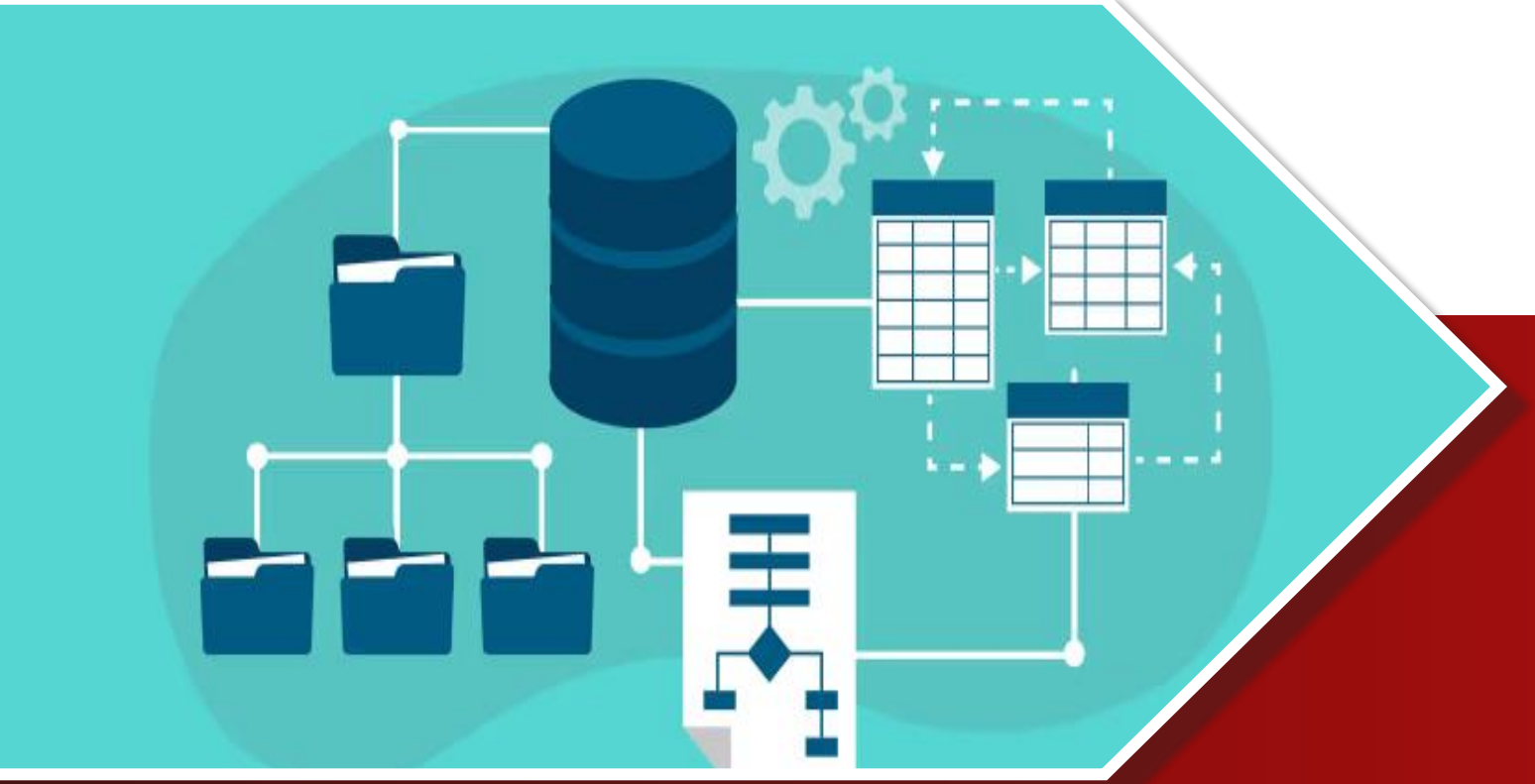
Actividad de Aprendizaje 05

Aplicación de Pilas y Colas

Centro Universitario de Ciencias Exactas e Ingenierías

Materia: Estructuras de Datos

Clave: V0731 ***Sección***: D02



Alumno: Mariscal Rodríguez Omar Jesús

Código: 220858478

Profesor: Gutiérrez Hernández Alfredo

Fecha: 14 de Septiembre de 2025

UNIVERSIDAD DE GUADALAJARA

PIENSA Y TRABAJA

Contenido

Test de Autoevaluación

Autoevaluación			
Concepto	***Sí***	***No***	***Acumulado***
Bajé el trabajo de internet o alguien me lo pasó (aunque sea de forma parcial)	***-100 pts***	***0 pts***	***0***
Incluí el código fuente ***en formato de texto (sólo si funciona cumpliendo todos los requerimientos)***	***+25 pts***	***0 pts***	***25***
Incluí las ***impresiones de pantalla (sólo si funciona cumpliendo todos los requerimientos)***	***+25 pts***	***0 pts***	***25***
Incluí una ***portada*** que identifica mi trabajo (nombre, código, materia, fecha, título)	***+25 pts***	***0 pts***	***25***
Incluí una ***descripción y conclusiones*** de mi trabajo	***+25 pts***	***0 pts***	***25***
		Suma:	***100***

Introducción y Abordaje del Problema

El objetivo de esta actividad radico en realizar una aplicación de pilas y colas, en un problema relacionado a transformar de notación infija (como es la aritmética tradicional) a una notación postfija que no requiera de que los operadores posean una precedencia intrínseca.

Planeación del Problema

Este problema de transformación de notaciones lo habíamos visto en clase a papel, y, de hecho, teníamos con nosotros ya el algoritmo (no en código) para realizar este cambio de notación. Analizando el algoritmo y sus pasos, notamos que el algoritmo requiere de tres contenedores de información (o datos en este caso), uno para la entrada de la notación infija, otra para la notación posfija y además, un contenedor auxiliar para almacenar temporalmente los operadores; y analizando esta serie de pasos notamos que debemos tratarlas como si fueran dos colas para la entrada y salida y una pila para almacenar los operadores; de esta manera, notamos como es que las pilas y colas pueden ser utilidad y no casarnos definitivamente con una lista convencional, sin mencionar que todas las llamadas a funciones se basan en una pila.

Conociendo el algoritmo y los requerimientos, a partir de aquí planearemos nuestro programa y las clases que intervienen. Primeramente, Cola (Queue) y Pila (Stack), y usaremos una clase que utilice estas para la transformación, la llamaremos NotationTransformer, y tendrá un método para capturar una notación infija y uno para transformarla a notación posfija. Como usaremos una cola para las transformaciones, haremos un método estático para convertir una cadena en una Cola normalizada (sin espacios), de esta manera, podremos capturar el dato de manera convencional y pasarlo a una cola normalizada con este método.

También, programaremos una función boolena isValidExpresion para determinar si la expresión que se ingresó es una de notación infija bien escrita para evitar que el algoritmo trabaje con casos imprevisibles que lleven a un funcionamiento incorrecto; esta función tendrá que evaluar que por ejemplo, no haya dos operadores u operandos seguidos (algo como a++b o ab+c), que cada paréntesis de apertura tenga su paréntesis de cierre o que se inicie con un operando o paréntesis de apertura y no con un operando.

Si todo sale correcto, pasaremos esta función a un atributo que la almacene y con la que trabajará posteriormente, si se intenta guardar una expresión que no es válida, lanzaremos una excepción personalizada para que el programa cierre y no provocar comportamientos indefinidos; esto, a su vez servirá en el menú para que mediante con un try catch, poder hacerle saber al usuario que la notación es incorrecta y que lo intente nuevamente.

De ahí, tendremos más métodos auxiliares, por ejemplo, necesitamos tener precedencia con los operadores, así que haremos un método llamado getPrecedence que reciba un carácter y que determine un valor, a más precedencia como ^, más valor, y un método estilo ToString para pasar a cadena de caracteres.

Para darle un poco más de funcionalidad al programa y que no tenga solo una función, vamos a ponerle al menú principal una Lista por referencia donde almacenaremos el historial de transformaciones, para en el mismo programa poder ver las transformaciones que hemos hecho en su totalidad (también ayudará para poder presentar el trabajo y los ejemplos de manera más detallada), podremos también eliminar este historial. El funcionamiento detrás no tiene gran ciencia, solo, cada que hagamos una transformación la guardaremos en formato de cadena de caracteres en la Lista (tanto su expresión original como su expresión en notación posfija), dándole una nueva funcionalidad al programa. Aprovechando, reciclaré un par de funciones del menú del trabajo anterior, como para capturar enteros, crear los header's y demás funciones útiles que nos servirán.

Programación

Una vez con nuestro proceso de abstracción, pasamos a programar de las clases más simples primero, NotationTransformer, y programamos en un inicio el método statico stringToQueue, revisé que funcionara correctamente y quitara los espacios, las funciones útiles como isValidExpresion, la cuál la hice mediante una serie de variables que contemplan varias cosas, por ejemplo, existe un contador de paréntesis, cada vez que se abre uno se le suma +1 y cada vez que hay un paréntesis de cierre se resta -1, y al final, si el resultado es ¡= de 0, quiere decir que hay un desbalance, por lo que regresamos false, o existe una variable booleana que determina si se espera un operador o un operando, esto es, cada vez que en una expresión infija se coloca un operador, inmediatamente después de debe ingresar un operando o un paréntesis, esta variable controla que no haya

operandos u operadores seguidos. Con este tipo de variables controlamos que haya una sintaxis válida.

La Precedencia no fue más que un switch case donde agrupamos las de igual valor y cada operador le toca un valor (también tenemos una función llamada isOperator, donde regresa true si se le pasa un operador considerado como +-/*^).

Con la función principal, de transformación, seguí el algoritmo que trabajamos en clase, traduciéndolo en lenguaje C++ con las herramientas que se me da, con algo de prueba y error para depurar errores o descuidos, quedó correcta sin mayores complicaciones. Y la implementación del menú funciona como la actividad anterior, una función de mainMenu y una función que maneja el switch por aparte para dividir la responsabilidad, funciones específicas para cada operación que queramos usar de la aplicación y manejo de excepciones del transformador.

Código Fuente

Carpeta Include

list.hpp

```
#ifndef __LIST_H__
#define __LIST_H__

#include <fstream>
#include <iomanip>
#include <iostream>
#include <sstream>

#include "ownexceptions.hpp"

template <class T, int ARRAYSIZE = 1024>
class List {
 private:
  T data[ARRAYSIZE];
  int last;

  void copyAll(const List<T, ARRAYSIZE>&);

 public:
  List<T, ARRAYSIZE>();
  List<T, ARRAYSIZE>(const List<T, ARRAYSIZE>&);

  bool isEmpty();
  bool isFull();
  void insertElement(const T&, const int&);
  void deleteData(const int&);
  T* retrieve(const int&);

  // Getter's
  int getFirstPosition() const;
  int getLastPosition() const;

  int getPrevPosition(const int&) const;
  int getNextPosition(const int&) const;

  std::string toString() const;

  void deleteAll();

  bool isValidPosition(const int&) const;

  List<T, ARRAYSIZE> operator=(const List<T, ARRAYSIZE>&);
```

```
  template <class X>
  friend std::ostream& operator<<(std::ostream&, const List<X>&);
  template <class X>
  friend std::istream& operator>>(std::istream&, List<X>&);
};

template <class T, int ARRAYSIZE>
List<T, ARRAYSIZE>::List() : last(-1) {}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::copyAll(const List<T, ARRAYSIZE>& other) {
  for (int i = 0; i < other.last; i++)
    this->data[i] = other.data[i];
  this->last = other.last;
}

template <class T, int ARRAYSIZE>
bool List<T, ARRAYSIZE>::isValidPosition(const int& position) const {
  return !(position > last || position < 0);
}

template <class T, int ARRAYSIZE>
List<T, ARRAYSIZE>::List(const List<T, ARRAYSIZE>& other) {
  copyAll(other);
}

template <class T, int ARRAYSIZE>
bool List<T, ARRAYSIZE>::isEmpty() {
  return this->last == -1;
}

template <class T, int ARRAYSIZE>
bool List<T, ARRAYSIZE>::isFull() {
  return this->last == (ARRAYSIZE - 1);
}

// Inserción en el Punto de Interés
template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::insertElement(const T& newData, const int&
position) {
  if (isFull())
    throw DataContainersExceptions::MemoryDeficiency(
        "Lista Llena, InsertElement(List)");

  if (!isValidPosition(position) && position != last + 1)
    throw DataContainersExceptions::InvalidPosition(
        "Posicion Invalida, InsertElement(List)");
```

```
    for (int i = last; i >= position; i--)
      this->data[i + 1] = this->data[i];
    this->data[position] = newData;
    last++;
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::deleteData(const int& position) {
    if (!isValidPosition(position))
      throw DataContainersExceptions::InvalidPosition(
          "Poscion Invalida, delteData(List)");

    for (int i = position; i < last; i++)
      this->data[i] = this->data[i + 1];
    last--;
}

template <class T, int ARRAYSIZE>
T* List<T, ARRAYSIZE>::retrieve(const int& position) {
    if (!isValidPosition(position))
      throw DataContainersExceptions::InvalidPosition(
          "Posicion Invalida, retrieve(List)");
    return &data[position];
}

template <class T, int ARRAYSIZE>
int List<T, ARRAYSIZE>::getFirstPosition() const {
    return isEmpty() ? -1 : 0;
}

template <class T, int ARRAYSIZE>
int List<T, ARRAYSIZE>::getLastPosition() const {
    return this->last;
}

template <class T, int ARRAYSIZE>
int List<T, ARRAYSIZE>::getPrevPosition(const int& position) const {
    return (!isValidPosition(position) || position == 0) ? -1 : (position -
1);
}

template <class T, int ARRAYSIZE>
int List<T, ARRAYSIZE>::getNextPosition(const int& position) const {
    return (!isValidPosition(position) || position == last) ? -1 :
(position + 1);
}

template <class T, int ARRAYSIZE>
```

```
std::string List<T, ARRAYSIZE>::toString() const {
  if (this->last <0 )
    return "";

  std::ostringstream oss;
  size_t max_left_width = 0;

  for (int i = 0; i <= last; i++) {
    size_t arrow_pos = data[i].find(" -> ");
    if (arrow_pos != std::string::npos) {
      max_left_width = std::max(max_left_width, arrow_pos);
    }
  }

  for (int i = 0; i <= last; i++) {
    size_t arrow_pos = data[i].find(" -> ");

    if (arrow_pos != std::string::npos) {
      std::string left_part = data[i].substr(0, arrow_pos);
      std::string right_part = data[i].substr(arrow_pos + 4);
      oss << std::right << std::setw(max_left_width) << left_part << " ->
"
          << right_part << std::endl;
    } else {
      oss << data[i] << std::endl;
    }
  }

  return oss.str();
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::deleteAll() {
  this->last = -1;
}

template <class T, int ARRAYSIZE>
List<T, ARRAYSIZE> List<T, ARRAYSIZE>::operator=(
    const List<T, ARRAYSIZE>& other) {
  copyAll(other);

  return *this;
}

template <class X>
std::ostream& operator<<(std::ostream& os, const List<X>& list) {
  int i = 0;
  while (i <= list.last)
```

```
    os << list.data[i++] << "," << std::endl;

  return os;
}

template <class X>
std::istream& operator>>(std::istream& is, List<X>& list) {
  X obj;
  std::string aux;

  try {
    while (is >> obj) {
      if (!list.isFull())
        list.data[++list.last] = obj;
    }
  } catch (const std::invalid_argument& ex) {
  }
  return is;
}

#endif  // __LIST_H__
```

notationtransformer.hpp

```
#ifndef __NOTATIONTRANSFORMER_H__
#define __NOTATIONTRANSFORMER_H__

#include <string>

#include "queue.hpp"
#include "stack.hpp"

#include "ownexceptions.hpp"

class NotationTransformer {
 private:
  Queue<char>& infixQueue;
  Queue<char>& postfixQueue;
  Stack<char>& operatorStack;

  bool isOperator(const char&) const;
  int getPrecedence(const char&) const;
  bool isValidExpresion(const Queue<char>&);

 public:
  NotationTransformer();
  NotationTransformer(const NotationTransformer&);
  NotationTransformer(const Queue<char>&);

  void setInflixQueue(const Queue<char>&);
  void transformToPosfija();

  static Queue<char> stringToQueue(const std::string&);
  std::string postFixToString() const;

  NotationTransformer& operator=(const NotationTransformer&);
};

#endif  // __NOTATIONTRANSFORMER_H__
```

notationtransformermenu.hpp

```
#ifndef __NOTATIONTRANSFORMERMENU_H__
#define __NOTATIONTRANSFORMERMENU_H__

#include <iomanip>
#include <iostream>
#include <sstream>

#include "list.hpp"
#include "notationtransformer.hpp"
#include "ownexceptions.hpp"

class NotationTransformerMenu {
 private:
  NotationTransformer& transformer;
  List<std::string>& record;

  int readInteger(std::string, const int&, const int&);
  std::string windowHeader(const int&, const std::string&);
  void enterToContinue();
  bool handleOption(const std::string&);

  void makeTranformation();
  void showRecord();
  void deleteRecords();
  void exitScreen();

 public:
  NotationTransformerMenu();
  NotationTransformerMenu(NotationTransformer&, List<std::string>&);
  NotationTransformerMenu(const NotationTransformerMenu&);

  void mainMenu();
};

#endif  // __NOTATIONTRANSFORMERMENU_H__
```

ownexceptions.hpp

```
#ifndef __OWNEXCEPTIONS_H__
#define __OWNEXCEPTIONS_H__

#include <stdexcept>
#include <string>

namespace DataContainersExceptions {
class MemoryDeficiency : public std::runtime_error {
 public:
  explicit MemoryDeficiency(const std::string& msg = "Insuficiencia de
Memoria")
      : std::runtime_error(msg) {}
};

class MemoryOverflow : public std::runtime_error {
 public:
  explicit MemoryOverflow(const std::string& msg = "Desbordamiento de
Memoria")
      : std::runtime_error(msg) {}
};

class InvalidPosition : public std::runtime_error {
 public:
  explicit InvalidPosition(
      const std::string& msg = "La posicion Ingresada es Invalida")
      : std::runtime_error(msg) {}
};
}  // namespace DataContainersExceptions

namespace NotationTransformerExceptions {
class InvalidExpresion : public std::runtime_error {
 public:
  explicit InvalidExpresion(
      const std::string& msg = "La Expresion No es Valida")
      : std::runtime_error(msg) {}
};

class NotOperator : public std::runtime_error {
 public:
  explicit NotOperator(
      const std::string& msg = "No es un operador para asignarle
precedencia.")
      : std::runtime_error(msg) {}
};

class NotInfixData : public std::runtime_error {
 public:
```

```
    explicit NotInfixData(
        const std::string& msg = "No se ha capturado la notacion infija.")
        : std::runtime_error(msg) {}
};
}  // namespace NotationTransformerExceptions

namespace MenuExceptions {
class InvalidOption : public std::runtime_error {
 public:
  explicit InvalidOption(
        const std::string& msg = "La opcion ingresada esta fuera de rango")
        : runtime_error(msg) {}
};
}  // namespace MenuExceptions

#endif  // __OWNEXCEPTIONS_H__
```

queque.hpp

```
#ifndef __QUEUE_H__
#define __QUEUE_H__

#include "ownexceptions.hpp"

using namespace std;

// Definición

template <class T, int ARRAYSIZE = 1024>
class Queue {
 private:
  T data[ARRAYSIZE];
  int frontPos = 0;
  int endPos = ARRAYSIZE - 1;

  void copyAll(const Queue<T, ARRAYSIZE>&);

 public:
  Queue();
  Queue(const Queue<T, ARRAYSIZE>&);

  bool isEmpty() const;
  bool isFull() const;

  void enqueue(const T&);
  T dequeue();

  T& getTop();

  Queue<T, ARRAYSIZE>& operator=(const Queue<T, ARRAYSIZE>&);
};

// Implementación

template <class T, int ARRAYSIZE>
void Queue<T, ARRAYSIZE>::copyAll(const Queue<T, ARRAYSIZE>& other) {
  for (int i = other.frontPos; i <= other.endPos; i++) {
    this->data[i] = other.data[i];
  }
  this->frontPos = other.frontPos;
  this->endPos = other.endPos;
}

template <class T, int ARRAYSIZE>
Queue<T, ARRAYSIZE>::Queue() {}
```

```
template <class T, int ARRAYSIZE>
Queue<T, ARRAYSIZE>::Queue(const Queue<T, ARRAYSIZE>& other) {
  copyAll(other);
}

template <class T, int ARRAYSIZE>
bool Queue<T, ARRAYSIZE>::isEmpty() const {
  return (this->frontPos == this->endPos + 1) ||
         (this->frontPos == 0 && this->endPos == ARRAYSIZE - 1);
}

template <class T, int ARRAYSIZE>
bool Queue<T, ARRAYSIZE>::isFull() const {
  return (this->frontPos == this->endPos + 2) ||
         (this->frontPos == 0 && this->endPos == ARRAYSIZE - 2) ||
         (this->frontPos == 1 && this->endPos == ARRAYSIZE - 1);
}

template <class T, int ARRAYSIZE>
void Queue<T, ARRAYSIZE>::enqueue(const T& data) {
  if (this->isFull())
    throw(DataContainersExceptions::MemoryOverflow());

  this->endPos++;
  if (this->endPos == ARRAYSIZE) {
    this->endPos = 0;
  }

  this->data[this->endPos] = data;
}

template <class T, int ARRAYSIZE>
T Queue<T, ARRAYSIZE>::dequeue() {
  if (this->isEmpty())
    throw(DataContainersExceptions::MemoryDeficiency());

  T result(this->data[this->frontPos]);

  if (++this->frontPos == ARRAYSIZE) {
    this->frontPos = 0;
  }

  return T(result);
}

template <class T, int ARRAYSIZE>
T& Queue<T, ARRAYSIZE>::getTop() {
  if (this->isEmpty())
```

```
    throw DataContainersExceptions::MemoryDeficiency();

  return this->data[this->frontPos];
}

template <class T, int ARRAYSIZE>
Queue<T, ARRAYSIZE>& Queue<T, ARRAYSIZE>::operator=(
    const Queue<T, ARRAYSIZE>& other) {
  copyAll(other);
  return *this;
}

#endif  // __QUEUE_H__
```

stack.hpp

```
#ifndef __STACK_H__
#define __STACK_H__

// Definición

#include "ownexceptions.hpp"

template <class T, int ARRAYSIZE = 1024>
class Stack {
 private:
  T data[ARRAYSIZE];
  int top;

  void copyAll(const Stack<T, ARRAYSIZE>&);

 public:
  Stack();
  Stack(const Stack<T, ARRAYSIZE>&);

  bool isEmpty();
  bool isFull();

  void push(const T&);
  T pop();

  T& getTop();

  Stack<T, ARRAYSIZE>& operator=(const Stack<T, ARRAYSIZE>&);
};

// Implementación

template <class T, int ARRAYSIZE>
void Stack<T, ARRAYSIZE>::copyAll(const Stack<T, ARRAYSIZE>& other) {
  int i = 0;

  while (i <= other.top) {
    this->data[i++] = other.data[i];
  }

  this->top = other.top;
}

template <class T, int ARRAYSIZE>
Stack<T, ARRAYSIZE>::Stack() : top(-1) {}

template <class T, int ARRAYSIZE>
```

```
Stack<T, ARRAYSIZE>::Stack(const Stack<T, ARRAYSIZE>& other) {
  copyAll(other);
}

template <class T, int ARRAYSIZE>
bool Stack<T, ARRAYSIZE>::isEmpty() {
  return this->top == -1;
}

template <class T, int ARRAYSIZE>
bool Stack<T, ARRAYSIZE>::isFull() {
  return this->top == (ARRAYSIZE - 1);
}

template <class T, int ARRAYSIZE>
void Stack<T, ARRAYSIZE>::push(const T& data) {
  if (isFull()) {
    throw DataContainersExceptions::MemoryOverflow();
  }

  this->data[++this->top] = data;
}

template <class T, int ARRAYSIZE>
T Stack<T, ARRAYSIZE>::pop() {
  if (isEmpty()) {
    throw DataContainersExceptions::MemoryDeficiency();
  }

  return this->data[this->top--];
}

template <class T, int ARRAYSIZE>
T& Stack<T, ARRAYSIZE>::getTop() {
  if (isEmpty()) {
    throw DataContainersExceptions::MemoryDeficiency();
  }
  return this->data[this->top];
}

template <class T, int ARRAYSIZE>
Stack<T, ARRAYSIZE>& Stack<T, ARRAYSIZE>::operator=(
    const Stack<T, ARRAYSIZE>& other) {
  this->copyAll(other);
  return *this;
}

#endif  // __STACK_H__
```

Carpeta src

main.cpp

```
#include "notationtransformermenu.hpp"
int main() {
  NotationTransformerMenu menu;

  return 0;
}
```

notationtransformer.cpp

```cpp
#include "notationtransformer.hpp"
#include <iostream>

// Operadores permitidos
bool NotationTransformer::isOperator(const char& caracter) const {
  return caracter == '+' || caracter == '-' || caracter == '*' ||
         caracter == '/' || caracter == '^';
}

// Darle un valor numérico para la comparación
int NotationTransformer::getPrecedence(const char& caracter) const {
  switch (caracter) {
    case '^':
      return 3;
    case '*':
    case '/':
      return 2;
    case '+':
    case '-':
      return 1;
    default:
      return 0;
  }
}

bool NotationTransformer::isValidExpresion(const Queue<char>&
infixExpresion) {
  // Verificar si una notación Infija es válida
  if (infixExpresion.isEmpty())
    return false;

  Queue<char> q = infixExpresion;

  // Esta variable determinará si se espera un operador o un operando a
lo largo
  // de las validaciones
  bool expectedOperand = true;
  // El balace de paréntesis, como debe haber un número par, sumaremos +1
por
  // cada ( y restaremos -1 por cada ). Tiene que estar en 0's a final
  int parenBalance = 0;
  // Número de operandos
  int operands = 0;
  // Número de Operadores
  int operators = 0;
  // Caracter anterior para hacer comparativas
  char prev = '\0';
```

```
  // Si esta vacía
  while (!q.isEmpty()) {
    char c = q.dequeue();

    // Si es una Variable o un dígito
    if (std::isalnum(static_cast<unsigned char>(c))) {
      operands++;
      // Si se inicia con un paréntesis de cierre es invalida
      if (prev == ')')
        return false;
      expectedOperand = false;  // Ahora esperamos un operador no un
operando
    }

    else if (c == '(') {
      parenBalance++;                       // Balance de Paréntesis
      if (!expectedOperand && prev != '(')  // Si se ingresa un operador
        return false;
      expectedOperand = true;
    }

    else if (c == ')') {
      if (parenBalance == 0)  // Si no se han agregado paréntesis de
apertura
        return false;
      if (expectedOperand)  // Si sigue inmediatamente después de un
operador
        return false;
      parenBalance--;  // Balanceamos los paréntesis
      expectedOperand = false;
    }

    else if (this->isOperator(c)) {  // Si es un operador
      if (expectedOperand)           // Si se juntan dos operando
        return false;
      operators++;
      expectedOperand = true;
    }

    else {
      return false;  // Cualquier otra entrada será inválida
    }

    prev = c;
  }

  // Validaciones Varias
```

```
   if (parenBalance != 0)
     return false;
   if (expectedOperand)
     return false;
   if (operands == 0)
     return false;
   if (operands != operators + 1)
     return false;
   return true;
}

NotationTransformer::NotationTransformer()
    : postfixQueue(*new Queue<char>),
      infixQueue(*new Queue<char>),
      operatorStack(*new Stack<char>) {}

NotationTransformer::NotationTransformer(const NotationTransformer&
other)
    : infixQueue(other.infixQueue),
      postfixQueue(*new Queue<char>),
      operatorStack(*new Stack<char>) {}

NotationTransformer::NotationTransformer(const Queue<char>& iQ)
    : postfixQueue(*new Queue<char>),
      infixQueue(*new Queue<char>),
      operatorStack(*new Stack<char>) {
  setInflixQueue(iQ);
}

void NotationTransformer::setInflixQueue(const Queue<char>&
infixNotation) {
  if (this->isValidExpresion(infixNotation))  // Si es válida
    this->infixQueue = infixNotation;
  else  // Excepción si no es válida
    throw NotationTransformerExceptions::InvalidExpresion();
}

void NotationTransformer::transformToPosfija() {
  // Verificar que hay una cola de entrada.
  // En el código nunca debería haber un
  if (infixQueue.isEmpty())
    throw NotationTransformerExceptions::NotInfixData();

  char caracter;
  // Vacías si tiene algo el resultado
  while (!postfixQueue.isEmpty())
    postfixQueue.dequeue();
```

```
 // Operar sobre la cola de entrada en Infija
  while (!infixQueue.isEmpty()) {
    caracter = infixQueue.dequeue();  // Capturamos el primer caracter

    if (caracter == '(') {
      operatorStack.push(caracter);  // Agregamos si es paréntesis de
apertura
      continue;
    }

    if (this->isOperator(caracter)) {  // Si es operador, vaciamos los
que sean
                                       // de mayor o igual valor
      while (!operatorStack.isEmpty() &&
             (this->getPrecedence(caracter) <=
              this->getPrecedence(operatorStack.getTop())) &&
             operatorStack.getTop() != ')') {
        postfixQueue.enqueue(operatorStack.pop());
      }
      operatorStack.push(caracter);  // Guardamos el operador
      continue;
    } else if (caracter ==
               ')') {  // Paréntesis de cierre saca todo de la pila hasta
                       // encontrarse con un paréntesis de apertura
      while (operatorStack.getTop() != '(') {
        postfixQueue.enqueue(operatorStack.pop());
      }
      operatorStack.pop();  // Se saca el paréntesis de apertura
encontrado
      continue;
    }
    postfixQueue.enqueue(
        caracter);  // Si no es ni operador ni paréntesis, se considera
operando
  }

  while (!operatorStack.isEmpty())  // Vacíamos al resultado el resto de
                                    // operadores en la pila
    postfixQueue.enqueue(operatorStack.pop());
}

Queue<char> NotationTransformer::stringToQueue(const std::string& s) {
  Queue<char> q;

  for (int i = 0; i < s.size(); ++i) {
    char c = s[i];
    if (c == ' ')  // Ignorar Espacios
      continue;
```

```
      q.enqueue(c);
    }
    return q;
}

std::string NotationTransformer::postFixToString() const {
    Queue<char> postfix = this->postfixQueue;
    string result;
    while (!postfix.isEmpty())
      result += postfix.dequeue();
    return result;
}

NotationTransformer& NotationTransformer::operator=(
      const NotationTransformer& other) {
    this->infixQueue = other.infixQueue;
    this->postfixQueue = other.postfixQueue;
    this->operatorStack = other.operatorStack;

    return *this;
}
```

notationtransfomermenu.cpp

```
#include "notationtransformermenu.hpp"

using namespace std;

int NotationTransformerMenu::readInteger(string oss,
                                         const int& lowerLimit,
                                         const int& upperLimit) {
  string aux("");
  int result;
  while (true) {
    try {
      system("CLS");
      cout << oss;
      getline(cin, aux);
      result = stoi(aux);

      if (result > upperLimit || result < lowerLimit)
        throw MenuExceptions::InvalidOption("Numero Fuera de Rango");
      break;
    } catch (std::invalid_argument) {
      system("CLS");
      cout << "Entrada invalida" << endl;
      cout << "Intente nuevamente" << endl;
      system("PAUSE");
    } catch (MenuExceptions::InvalidOption msg) {
      system("CLS");
      cout << msg.what() << endl;
      system("PAUSE");
    }
  }

  return result;
}

string NotationTransformerMenu::windowHeader(const int& widthBorder,
                                             const string& prompt) {
  ostringstream oss;

  oss << left << setfill('=') << setw(widthBorder) << "" << endl;
  oss << setfill(' ');

  // Título de Ventana
  oss << setw(widthBorder / 2 - (prompt.size() / 2)) << "| " << prompt
      << setw((widthBorder / 2) - (prompt.size() / 2)) << "" << "|" <<
endl;
  oss << setfill('-') << setw(widthBorder) << "" << endl;
  oss << setfill(' ');
```

```
  return oss.str();
}

void NotationTransformerMenu::enterToContinue() {
  cout << "[Enter] Para Continuar..." << endl;
  getchar();
}

bool NotationTransformerMenu::handleOption(const std::string& promt) {
  std::string response;

  system("CLS");
  std::cout << promt;
  std::getline(std::cin, response);

  if (response.empty())
    return true;

  // Hacer las letras mayúsculas
  char option =
      static_cast<char>(std::toupper(static_cast<unsigned
char>(response[0])));

  switch (option) {
    case 'T':
      makeTranformation();
      break;
    case 'R':
      showRecord();
      break;
    case 'E':
      deleteRecords();
      break;
    case 'S':
      exitScreen();
      return false;
      break;

    default:
      system("CLS");
      std::cout << "Comando invalido\nIntentelo nuevamente.\n";
      system("PAUSE");
      break;
  }  // switch

  return true;
}
```

```
void NotationTransformerMenu::makeTranformation() {
  string infixNotation;
  Queue<char> queueNotation;

  // Bucle perpetuo hasta que se ingresa una expresión válida
  while (true) {
    try {
      system("CLS");
      cout << windowHeader(50, "Transformar de Notacion Infija a
PostFija");
      cout << "Ingrese la operacion en notacion infija: ";
      getline(cin, infixNotation);
      queueNotation = this->transformer.stringToQueue(infixNotation);
      this->transformer.setInflixQueue(queueNotation);
      break;  // Si no hay excepción salimos del bucle.
    } catch (const NotationTransformerExceptions::InvalidExpresion&
                 ex) {  // Manejo de la Excepción Personalizada
      cout << "La expresion no es valida." << endl;
      cout << "Recuerde que no se pueden tener dos operadores juntos
(3++5)"
           << endl;
      cout << "Parentesis que no se cierran. No se puede empezar con ^
entre "
              "otras cosas (Si puede poner espacios)"
           << endl
           << endl;
      enterToContinue();
    }
  }
  // Convertir y mostrar la Expresión
  this->transformer.transformToPosfija();
  cout << "La conversion es: " << this->transformer.postFixToString() <<
endl;
  // Agregarla al Historial
  this->record.insertElement(
      infixNotation + "   ->   " + this->transformer.postFixToString(),
      this->record.getLastPosition() + 1);
  cout << "Conversion agregada al historial." << endl << endl;

  enterToContinue();
}

void NotationTransformerMenu::showRecord() {
  ostringstream oss;
  system("CLS");
  oss << windowHeader(100, "Historial de Tranformaciones");
```

```
    if (this->record.isEmpty()) {
      oss << "Aun no se tienen registros de transformaciones" << endl;
      oss << "Regresando al Menu Principal..." << endl;
    } else
      oss << this->record.toString() << endl;

    cout << oss.str();
    enterToContinue();
}

void NotationTransformerMenu::deleteRecords() {
    system("CLS");
    ostringstream oss;
    oss << windowHeader(50, "Eliminar Historial");

    if (this->record.isEmpty()) {
      oss << "No se tienen registros de conversiones aun." << endl;
      oss << "Regresando al Menu Principal." << endl;
      cout << oss.str();
      enterToContinue();
      return;
    }

    int response;
    oss << "Esta seguro que desea eliminar el historial de
transformaciones? (1. "
           "Si / 2. No): ";

    response = readInteger(oss.str(), 1, 2);
    system("CLS");

    if (response == 1) {
      this->record.deleteAll();
      oss << endl << "Historial Eliminado Con Exito!" << endl;
    } else {
      oss << endl << "Operacion Cancelada" << endl;
    }
    cout << oss.str();
    enterToContinue();
}

void NotationTransformerMenu::exitScreen() {
    system("CLS");
    ostringstream oss;
    oss << windowHeader(50, "Saliendo del Programa");
    oss << "Programa hecho por: Mariscal Rodriguez Omar Jesus" << endl;
    oss << "Profesor: Gutierrez Hernandez Alfredo" << endl << endl;
    oss << "Tenga un buen dia :D" << endl;
```

```
  cout << oss.str();
  enterToContinue();
}

NotationTransformerMenu::NotationTransformerMenu()
    : transformer(*new NotationTransformer), record(*new
List<std::string>) {
  mainMenu();
}

NotationTransformerMenu::NotationTransformerMenu(NotationTransformer& t,
                                                 List<string>& r)
    : transformer(t), record(r) {
  mainMenu();
}

NotationTransformerMenu::NotationTransformerMenu(
    const NotationTransformerMenu& other)
    : transformer(other.transformer), record(other.record) {
  mainMenu();
}

void NotationTransformerMenu::mainMenu() {
  ostringstream oss;
  bool op = true;
  oss << windowHeader(100, "Tranformador de Notacion Infija a Postfija");
  oss << "Opciones: [T]ransformar una Notacion \n"
      << "          [R]evisar el Historial de Transformaciones\n"
      << "          [E]liminar el Historial de Transformaciones\n"
      << "          [S]alir del Programa\n";
  oss << "Eleccion: ";

  while (op) {
    system("CLS");
    op = handleOption(oss.str());
  }
}
```

Ejecución del Programa

A continuación de muestran capturas de la ejecución del programa, comenzando por el menú principal:

```
================================================================================================
|                          Tranformador de Notacion Infija a Postfija                          |
------------------------------------------------------------------------------------------------
Opciones: [T]ransformar una Notacion
          [R]evisar el Historial de Transformaciones
          [E]liminar el Historial de Transformaciones
          [S]alir del Programa
Eleccion:
```

Tenemos opciones para transformar una notación (la principal del programa), para revisar el historial, eliminarlo y salir. Cualquier entrada diferente mandará la siguiente pantalla:

```
Comando invalido
Intentelo nuevamente.
Presione una tecla para continuar . . .
```

Si intentamos ingresar a una de las opciones que necesitan el historial como querer revisarlo o eliminarlo, nos aparecerá el siguiente mensaje:

```
================================================================================================
|                                  Historial de Tranformaciones                                |
------------------------------------------------------------------------------------------------
Aun no se tienen registros de transformaciones
Regresando al Menu Principal...
[Enter] Para Continuar...
```

Entonces, comencemos por transformar una notación:

```
==================================================
|    Transformar de Notacion Infija a PostFija    |
--------------------------------------------------
Ingrese la operacion en notacion infija:
```

Nos salta esta ventana donde se nos pide una expresión en notación infija, y si ponemos algo inválido como A++B

```
==================================================
|    Transformar de Notacion Infija a PostFija    |
--------------------------------------------------
Ingrese la operacion en notacion infija: A++B
La expresion no es valida.
Recuerde que no se pueden tener dos operadores juntos (3++5)
Parentesis que no se cierran. No se puede empezar con ^ entre otras cosas (Si puede poner espacios)

[Enter] Para Continuar...
```

Se nos avisa de unas primeras condiciones y nos pide volver a intentarlo, y una vez ingresemos una válida como (a+b)/c:

```
================================================
|    Transformar de Notacion Infija a PostFija     |
------------------------------------------------
Ingrese la operacion en notacion infija: (a+b)/c
La conversion es: ab+c/
Conversion agregada al historial.

[Enter] Para Continuar...
```

Se nos da la conversión y nos avisa que se agrega al historial.

Para poner a prueba el programa, vamos a más de 60 operaciones en notación infija para que se transformen a posfija.

Con todos estos registros, entremos a la opción [R] para revisar el historial de transformaciones:

```
================================================================================================
|                                Historial de Tranformaciones                                  |
------------------------------------------------------------------------------------------------
                                    (a+b)/c   ->   ab+c/
                                      a + b   ->   ab+
                                ( a + b ) * c   ->   ab+c*
                            a + b * c - d / e   ->   abc*+de/-
                        ( a + b ) * ( c - d ) / e   ->   ab+cd-*e/
                                    x ^ y ^ z   ->   xy^z^
                            ( x ^ ( y ^ z ) ) + w   ->   xyz^^w+
                        ( a + b + c + d + e ) * f   ->   ab+c+d+e+f*
                    a + ( b - c ) * ( d + e ) / f   ->   abc-de+*f/+
                ( ( a + b ) * c - d ) / ( e + f )   ->   ab+c*d-ef+/
                        n * ( m + ( k - j ) ) / p   ->   nmkj-+*p/
                        a * b + c * d - e / f + g   ->   ab*cd*+ef/-g+
            ( a + ( b * c ) ) - ( d / ( e + f ) )   ->   abc*+def+/-
            ( ( ( a + b ) * ( c + d ) ) - e ) / f   ->   ab+cd+*e-f/
                            p + q * r - s ^ t / u   ->   pqr*+st^u/-
                        ( p + q ) * ( r - s ) ^ t / u   ->   pq+rs-t^*u/
                            a + b + c + d + e + f   ->   ab+c+d+e+f+
                    ( a - b ) * ( c + d ) - e / f + g   ->   ab-cd+*ef/-g+
                        ( 3 + 4 ) * 5 - 6 / ( 7 + 8 )   ->   34+5*678+/-
                            1 + 2 * ( 3 - 4 ) / 5   ->   1234-*5/+
                        ( 9 - 2 ) / ( 3 + 4 ) + 3 ^ 2   ->   92-34+/32^+
                    ( ( 1 + 2 ) * ( 3 + 4 ) ) ^ 2 - 5   ->   12+34+*2^5-
                        a * b ^ c + d / e - f + g * h   ->   abc^*de/+f-gh*+
                        ( a + b ) * c - d ^ ( e + f )   ->   ab+c*def+^-
                    ( ( a + b ) ^ c ) / ( d - e ) + f   ->   ab+c^de-/f+
                        x + ( y - z ) * ( p + q ) / r   ->   xyz-pq+*r/+
                    ( x + y ) * ( z - p ) + q / r - s   ->   xy+zp-*qr/+s-
                            a ^ b + c ^ d - e ^ f + g   ->   ab^cd^+ef^-g+
                    ( a ^ ( b + c ) ) * ( d - e ) + f   ->   abc+^de-*f+
                a + ( b * ( c + ( d - e ) ) ) - f   ->   abcde-+*+f-
            ( ( a + b ) + ( c + d ) + ( e + f ) ) / g   ->   ab+cd++ef++g/
            ( ( ( a + b ) * c ) + d ) / ( e - f ) + g   ->   ab+c*d+ef-/g+
                            1 + 2 * ( 3 - 4 ) / 5   ->   1234-*5/+
                        ( 2 + 3 ) ^ ( 4 - 1 ) * 5 - 6   ->   23+41-^5*6-
                            a / b / c + d * e - f ^ g   ->   ab/c/de*+fg^-
                    ( a / ( b + c ) ) * ( d - e ) + f   ->   abc+/de-*f+
            ( ( a + b ) * ( c - d ) ) + ( e ^ f ) - g   ->   ab+cd-*ef^+g-
                        m + n + o + p + q + r + s   ->   mn+o+p+q+r+s+
```

Tenemos el registro de todas las transformaciones hechas, de notación infija a posfija, en este punto, podemos regresar al menú principal para probar la opción [E] de eliminar el registro:

```
================================================
|              Eliminar Historial               |
------------------------------------------------
Esta seguro que desea eliminar el historial de transformaciones? (1. Si / 2. No):
```

Se nos pide una confirmación (que no admite caracteres diferentes a 1 o 2) y al confirmarlo:

```
================================================
|              Eliminar Historial              |
------------------------------------------------
Esta seguro que desea eliminar el historial de transformaciones? (1. Si / 2. No):
Historial Eliminado Con Exito!
[Enter] Para Continuar...
```

Regresaremos al menú principal, pero al intentar acceder al historial volverá a estar bloqueado por falta de registros.

```
================================================================================================
|                                Historial de Tranformaciones                                  |
------------------------------------------------------------------------------------------------
Aun no se tienen registros de transformaciones
Regresando al Menu Principal...
[Enter] Para Continuar...
```

Finalmente, el botón [S] para salir nos muestra el siguiente mensaje:

```
================================================
|              Saliendo del Programa           |
------------------------------------------------
Programa hecho por: Mariscal Rodriguez Omar Jesus
Profesor: Gutierrez Hernandez Alfredo

Tenga un buen dia :D
[Enter] Para Continuar...
```

Y se cierra la ejecución del programa.

Conclusiones

Trabajar en esta aplicación de las pilas y colas fue bastante más interesante de lo que creía; al principio pecaba de no valorar suficiente la utilidad de una pila y una cola, pensaba que hacía más difícil el acceso y procesamiento de los datos ahí contenidos, pero es cierto que pueden llegar a ser sumamente convenientes en ciertas situaciones, son intuitivas y dan paso a realizar muchos programas con algoritmos compatibles con alguno de ellos. La implementación estática de la cola también fue interesantísima de conocer, imaginarla como una cola circular sin realmente serlo sino tratarla como tal, y sacrificar un espacio en la memoria con tal de no hacer corrimientos para que a la hora de trabajar con cantidades más grandes de datos no subir tanto el costo computacional.

Está claro que aún me falta por imaginar y experimentar con este par de estructuras de datos para verlas viables en muchas más situaciones de mi trabajo de abstracción previo a la programación tecleando, considerando también lo relevantes que son para la arquitectura de software moderno, pero este es un gran avance; un algoritmo que utiliza ambas para llegar a su objetivo final es grandioso. Los modelos de pila y cola permiten organizar trabajos y plantear soluciones más eficientes en costo computacional y en comprensión de funcionamiento, espero pronto aprender mucho más de estos y más modelos para ampliar mi conocimiento, experiencia y ser un poco mejor cada día. Estoy ansioso de ello.