UNIVERSIDAD NACIONAL DEL CALLAO

FACULTAD DE INGENIERÍA INDUSTRIAL Y DE SISTEMAS

UNIDAD DE INVESTIGACIÓN

INFORME FINAL DEL PROYECTO DE INVESTIGACIÓN

"DISEÑO DE SOFTWARE PARA EL USO DE ESTRUCTURAS DINÁMICAS CON LENGUAJE DE PROGRAMACIÓN C++"

AUTOR: MG. BERTILA LIDUVINA GARCIA DIAZ

PERIODO DE EJECUCÍON: Del 01/06/2020 al 31/05/2021

Resolución de aprobación N° R.R. N° 293-2020-R.- Callao 16/06/2020

Callao, 2021

Dedicatoria

Con gratitud a mi Padres, Esposo, Ana Lucía, Sobrinos y Hermanos.

Agradecimiento

Mi agradecimiento a Dios Todopoderoso, que me permitió finalizar esta investigación. Agradezco a todas las personas que colaboraron directa o indirectamente para la realización de esta investigación.

A mis alumnos del curso de Programación Estructurada, que coincidieron en la necesidad de profundizar el tema a nivel de programación. Al Dr. Teodomiro Pérez Cabrel y al Mg. Manuel Gallardo Otero, por iniciar este tema con un enfoque Algorítmico, durante la Maestría en Computación e Informática en la UNMSM.

INDICE

página

INDICE DE FIGURAS

RESUMEN

Esta investigación está referida a la Programación Dinámica, que utiliza el concepto de punteros, listas enlazadas para poder representar también a las pilas y colas. El objetivo es determinar el efecto de un nuevo enfoque de la Programación Dinámica en la programación informática de este tipo de estructuras. La Metodología de esta investigación es cualitativa e inductiva, se buscó generar teoría, conforme se investigó el tema y se conoció más el Marco Teórico. En lo referente a los resultados se demostró que el uso de punteros y listas enlazadas en la programación dinámica, contribuyen significativamente en la viabilidad de su programación informática, disminuyendo su complejidad. En conclusión, es posible estandarizar el tratamiento a nivel de programación de las estructuras dinámicas lineales utilizando una representación única, mediante nodos de listas enlazadas que tienen diferente tipo de comportamiento. En esta investigación se ha realizado un enfoque algorítmico y también de programación, que espero inspiren a los alumnos de programación a investigar y conocer estas nuevas estructuras que son utilizadas en inteligencia artificial, creación de sistemas operativos y software de alto nivel.

Palabras Clave: Programación Informática, Programación Dinámica, Punteros, Listas Enlazadas, Pilas, Colas

ABSTRACT

This research is related to Dynamic Programming, which uses the concept of pointers, linked lists to also represent stacks and queues. The objective is to determine the effect of a new approach to Dynamic Programming in the computer programming of this type of structure. The Methodology of this research is qualitative and inductive, it was sought to generate theory, as the subject was investigated and the Theoretical Framework was better known. Regarding the results, it was demonstrated that the use of pointers and linked lists in dynamic programming contribute significantly to the viability of its computer programming, reducing its complexity. In conclusion, it is possible to standardize the treatment at the programming level of linear dynamic structures using a single representation, through linked list nodes that have different types of behavior. In this research, an algorithmic and programming approach has been carried out, which I hope will inspire programming students to investigate and learn about these new structures that are used in artificial intelligence, creation of operating systems and high-level software.

Keywords: Computer Programming, Dynamic Programming, Pointers, Linked Lists, Stacks, Queues

INTRODUCCIÓN

La presente investigación que pongo a su consideración corresponde al campo de la programación de lenguajes de programación.

Dentro de la Ingeniería de Sistemas, los cursos orientados a la programación corresponden a los saberes más importantes de esta carrera tecnológica, ya que para la programación de software se requiere el dominio de lenguajes de programación como: C++, Visual Basic, Power Builder y otros. Esta carrera tiene dos aristas bien definidas que son el hardware y el software, sobre las cuales recae el avance de esta carrera, que muchos la definen como matemática aplicada. El hardware permite la creación de sistemas operativos como: Linux, Windows que son los encargados de administrar los recursos de la computadora como: procesadores, memoria, impresoras, diferentes dispositivos. Todos estos recursos están en constante evolución, así vemos que cada cierto tiempo los procesadores mejoran, actualmente estamos en Intel CORE i7, y las memorias mínimas ahora de 16 GB en adelante, igualmente aumenta la capacidad de almacenamiento que ahora se expresan en terabytes.

El software también está en constante evolución, en términos de programación procedimental hay dos tipos de estructuras: estáticas y dinámicas. Las estructuras estáticas nacieron con el origen de la computadora y manejan conceptos como: sentencias de control, funciones, arreglos. Posteriormente surgieron las estructuras dinámicas, necesarias para la creación de inteligencia artificial y mejora en el diseño de los sistemas operativos. Sin embargo las estructuras dinámicas no se enseñan a nivel superior, tal vez por su complejidad. Esta carencia de entendimiento de estas estructuras ha sido el motivo de realizar una investigación al respecto.

CAPÍTULO I: PLANTEAMIENTO DEL PROBLEMA

1.1 Descripción de la realidad problemática

El presente trabajo de investigación “Diseño de software para el uso de estructuras dinámicas con lenguaje de Programación C++.” Trata de cubrir información tecnológica faltante acerca de las estructuras dinámicas, que son la base para crear Inteligencia Artificial, Sistemas Operativos y tecnología de última generación. No olvidemos que las estructuras dinámicas han sido las estructuras más recientes, que aparecieron luego de las estructuras estáticas como: arreglos, registros, archivos y cadenas; y que han propiciado el gran avance tecnológico que hoy tenemos; como por ejemplo el control de colas en el spooling de una impresora, mejoramiento del Hardware de una computadora, Avances en inteligencia artificial, Avance de los Sistemas Expertos y otros.

Existe un vacío en el desarrollo del software acerca del tratamiento de estructuras dinámicas, como Listas, Pilas, Colas, Arboles.

El presente trabajo de investigación “Diseño de software para el uso de estructuras dinámicas con lenguaje de Programación C++.” Cubre información tecnológica faltante acerca de las estructuras dinámicas, revisando los antecedentes del tema, nos dice que no hay investigaciones al respecto y asimismo manifiestan que estas usan una programación compleja. Estas estructuras se consideran en la programación de los Syllabus de los cursos de Algoritmos y de Programación, donde son tratadas superficialmente en el mejor de los casos, de ahí mi interés en su investigación por ser técnicas que se utilizan en programación científica.

1.2 Formulación del problema

Problema General

¿Cómo el lenguaje C++ contribuye en el diseño del software de las Estructuras Dinámicas, que son la base para crear Inteligencia Artificial, Sistemas operativos, Teoría de Lenguajes?

Problemas Específicos

¿Cómo el lenguaje C++ contribuye en el diseño del software de la Estructura Dinámica Listas Enlazadas?

¿Cómo el lenguaje C++ contribuye en el diseño del software de la Estructura Dinámica Pilas?

¿Cómo el lenguaje C++ contribuye en el diseño del software de la Estructura Dinámica Colas?

1.3 Objetivos

Objetivo General

Determinar si el lenguaje C++ contribuye en el diseño de software de las Estructuras Dinámicas, que son la base para crear Inteligencia Artificial, Sistemas operativos, Teoría de Lenguajes.

Objetivos Específicos

Determinar si el lenguaje C++ contribuye en el Diseño de los Algoritmos de las Estructuras Dinámicas, que son la base para crear Inteligencia Artificial, Sistemas operativos, Teoría de Lenguajes.

Determinar si el lenguaje C++ contribuye en la programación de las Estructuras Dinámicas, que son la base para crear Inteligencia Artificial, Sistemas operativos, Teoría de Lenguajes.

1.4 Limitantes de la Investigación

Teórico

Se revisó textos y Revistas Científicas indizadas, y existe escasa información acerca de la Programación Dinámica. Una limitación realmente porque las fuentes bibliográficas no permiten al investigador profundizar en el tema.

Temporal

Como el estudio tiene un principio y fin menor a un año el efecto se puede visualizar los resultados con rapidez.

Espacial

Como la investigación es de tipo teórico, la investigación no tiene esta limitante.

CAPITULO II: MARCO TEÓRICO

2.1 Antecedentes

Realizaremos una breve historia del Lenguaje de Programación C++, este lenguaje ha ido evolucionando en diversas versiones a través del tiempo, desde el típico C, Turbo C, hasta llegar al Dev C++, que es el lenguaje preferido de los estudiantes de Ingeniería para programar. C++ es un lenguaje utilizado básicamente en el diseño de Sistemas Operativos, Compiladores, Bases de Datos, dónde se requiera rigor de código. Esto es subrayado por (Ayala, Aguilar, Zarco, & Gómez, 2016) quién manifiesta que C está ligado a los sistemas operativos, es así que el Sistema operativo Unix, fue desarrollado en C. Ritchie creó el Lenguaje C para escribir Unix. Fue el lenguaje adecuado para ese tiempo y ha liderado desde entonces la programación de sistemas, reduciendo tiempos de desarrollo en diferentes sistemas operativos, y otorgaron licencias de Unix a precios bajos, a las universidades; también es usado para escribir procesadores de texto, bases de datos. En 1990 se estandarizó y se llamó C90.

El presente trabajo de investigación “Diseño de software para el uso de estructuras dinámicas con lenguaje de Programación C++.” Examina y estudia las estructuras dinámicas, que pueden ser Listas enlazadas, Pilas, Colas y árboles.

Los antecedentes de la teoría de colas, se remontan a Agner Krarup Erlang, Lonborg (Dinamarca), (1878 –1929). Fue un matemático, estadístico e ingeniero danés quién en 1917 publicó su artículo titulado: "Soluciones a problemas importantes de la teoría de probabilidades aplicada a centrales automáticas de conmutación telefónica". También los trabajos de Erlang se constituyen en la base de la actual "teoría de colas". Se le considera el Inventor de la ingeniería de tráfico telefónico. Posteriormente Kendall perfeccionó el modelo de colas.

Figura N° 2.1

EJEMPLOS DE COLAS

Fuente: internet

2.2 Marco

2.2.1 Teórico

Toda variable que interviene en un programa, es asignada a una dirección de memoria para que el programa pueda ser ejecutado. En ese aspecto internamente el sistema operativo ubica las direcciones libres; asimismo se reserva memoria para que se almacenen los datos, llamada zona de datos, el puntero es un tipo de datos que almacena direcciones de memoria de diferente tipo; existen lenguajes que permiten la gestión de memoria dinámica y el uso de punteros como el C++, Dev C++, lo cual condiciona a un mayor nivel de abstracción o tipo abstracto de datos e implementan mecanismos que posibiliten tipos de datos no implementados (Martinez, 2006).

Los punteros son variables que contienen una dirección de memoria, la cual puede corresponder a la dirección de otra variable, de una función, de una estructura, de un objeto. Un puntero se representa de la siguiente manera:

Tipo de dato * puntero

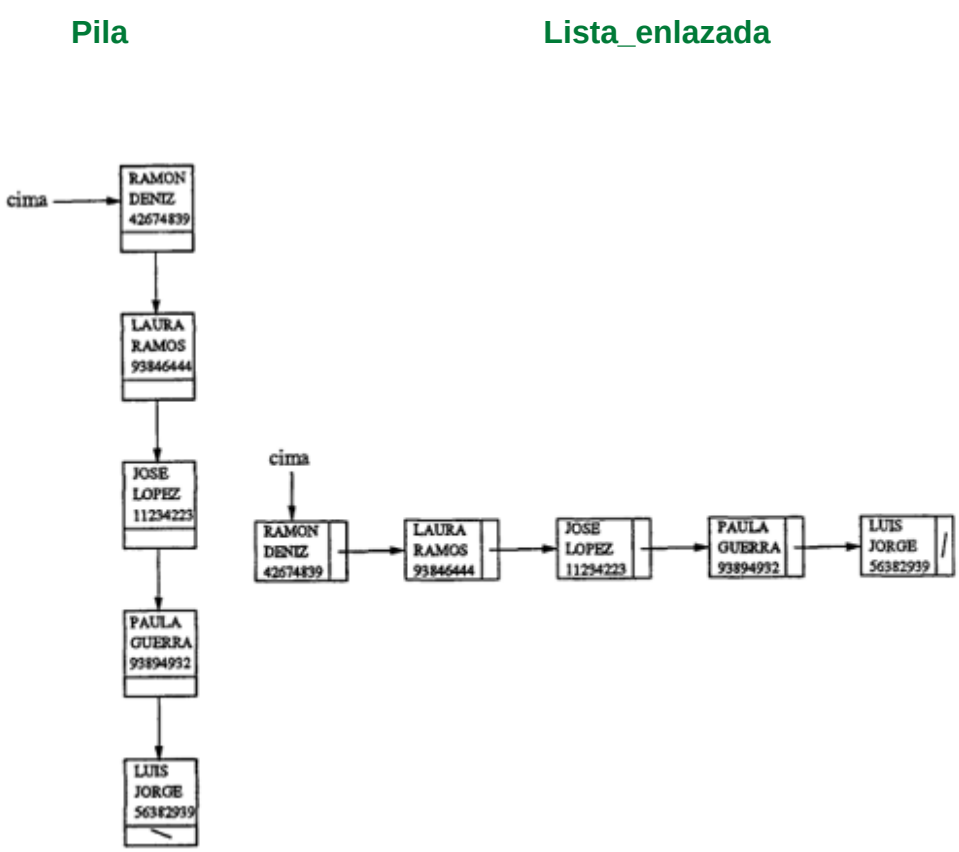
En este sentido punteros utiliza los operadores & y *, donde el operador & indica la dirección de un objeto y el operador * permite acceder al contenido de una dirección de memoria; el verdadero sentido del uso de punteros es reservar memoria para tiempo de ejecución (Bellot, y otros, 2001). El Lenguaje C++ accede a direcciones de memoria de manera dinámica, lo que significa que está en condiciones de asignar memoria en tiempo de ejecución, para lo cual debe

utilizar las librerías <stdlib.h> que gestiona memoria dinámica y <alloc.h> para asignación dinámica de memoria (Ayala, Aguilar, Zarco, & Gómez, 2016).

La pila es en realidad una lista enlazada, donde la inserción y la eliminación se realizan por la cima de la pila. En ese sentido (Gallardo & Pérez, 1995) opinan que la pila tiene la estructura de un Lista enlazada y que las pilas almacenan información temporal y lo que importa son los estados de la pila.

Figura N° 2.2.

ANALOGÍA ENTRE UNA PILA Y UNA LISTA ENLAZADA



Fuente: internet

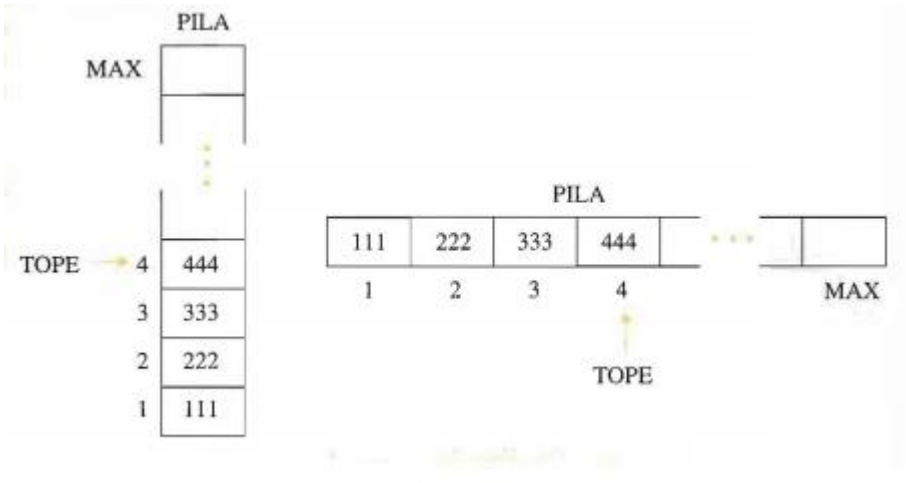
Según (Cairo & Guardati, 2005) las operaciones básicas de las pilas son básicamente dos:

Push: inserta la data en el tope de la pila

Pop: remueve la data del tope de la pila

Figura N° 2.3

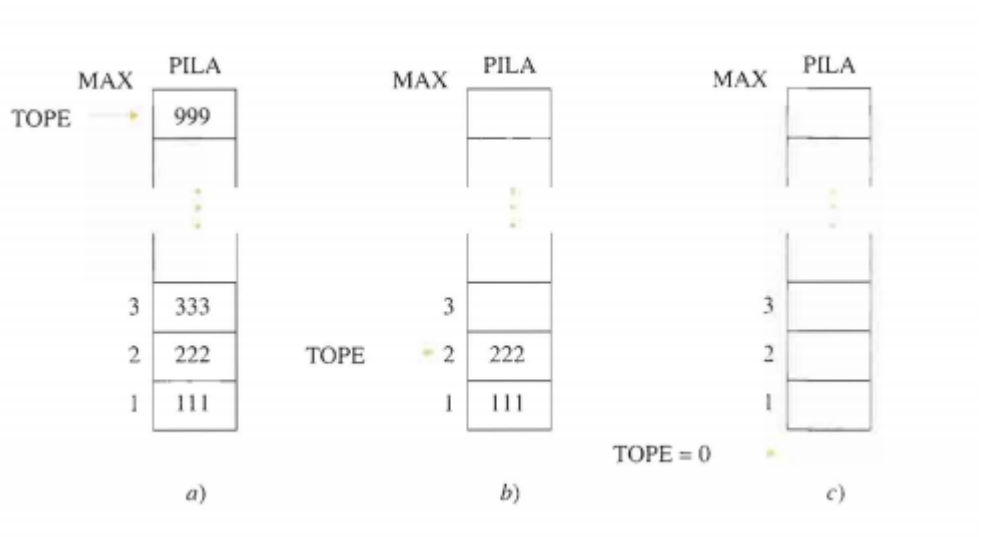
REPRESENTACIÓN GRÁFICA DE UNA PILA



Fuente: (Cairo & Guardati, 2005)

Figura N° 2.4

EJEMPLOS DE A) PILA LLENA, B) PILA CON ALGUNOS ELEMENTOS Y C) PILA VACÍA



Fuente: (Cairo & Guardati, 2005)

Una pila puede estar vacía o llena.

- UNDERFLOW: Si un programa intenta sacar un elemento de una pila vacía, se producirá un error, debido a que esa operación es imposible; esta situación se denomina desbordamiento negativo.

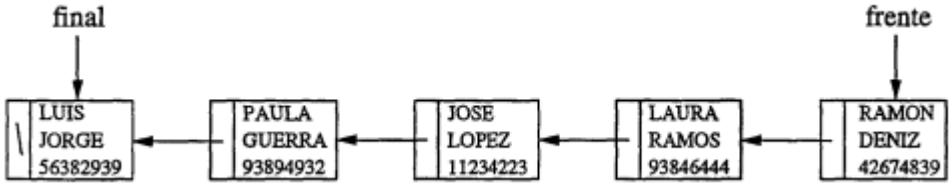
- OVERFLOW: Si un programa intenta poner un elemento en una pila llena se produce un error, una excepción, de desbordamiento o rebosamiento.

La lista enlazada se le denomina lineal porque es una secuencia lineal y dinámica de datos, es dinámica porque permite el uso de memoria dinámica (Cairo & Guardati, 2005).

Una cola es un tipo de lista lineal, en la cual las eliminaciones se realizan al principio de las colas y las inserciones se realizan en el otro extremo. Ejemplo una cola para subir al bus, cola para atención en el Banco. Una cola es realmente una lista enlazada, que difiere de estas listas en cuanto a su comportamiento y tipo de operaciones definidas (Jimenéz & Sánchez, 2002).

Figura N° 2.5

REPRESENTACIÓN GRÁFICA DE UNA COLA



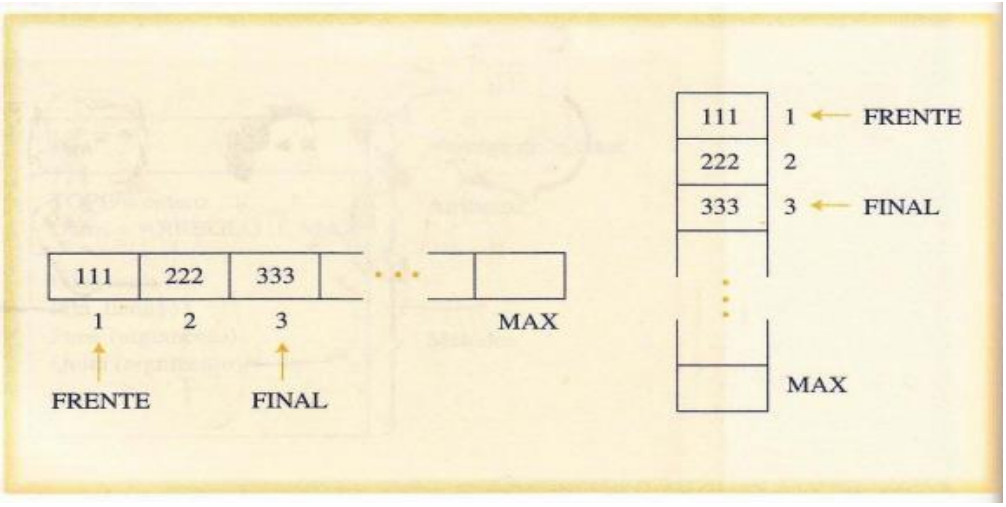
.Fuente: (Ojeda, Martel, Ramírez, & Quintana, 1997)

Son estructuras de datos de tipo FIFO (First in - First out). El comportamiento de las colas se caracteriza porque el primer elemento en ingresar es el primero en salir. Una cola almacena elementos en una lista, donde la inserción se realiza por el final y la eliminación se efectúa por el frente de la lista (Joyanes & Zahonero, Estructura de Datos en C++, 2007).

La cola es utilizado a veces como un buffer, como un depósito donde se pueden almacenar objetos que están en espera de ser procesados por la computadora, es un subtipo de la lista enlazada (Bottazi, y otros).

Figura N° 2.6

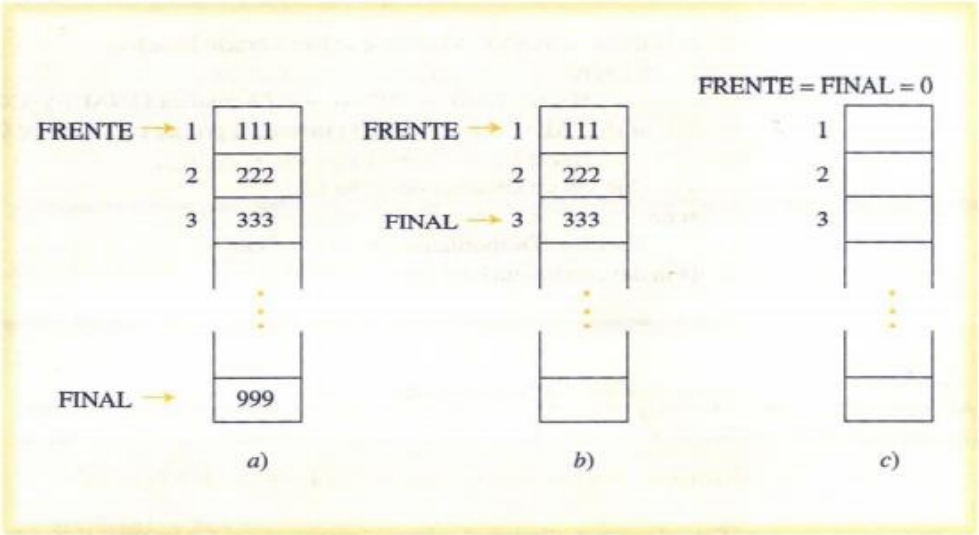
REPRESENTACIÓN GRÁFICA DE UNA COLA, MEDIANTE ARREGLOS



Fuente: (Cairo & Guardati, 2005)

Figura N° 2.7

EJEMPLOS DE A) COLA LLENA, B) COLA CON ALGUNOS ELEMENTOS Y C) COLA VACÍA



. Fuente: (Cairo & Guardati, 2005)

2.2.2 Conceptual

En la programación dinámica es posible variar el contenido de la variable en tiempo de ejecución y también variar su tamaño.

En la asignación dinámica de memoria el sistema operativo asigna espacios libres para ubicar un elemento. Y enlazan las distintas partes de la estructura dinámica. Se conserva la dirección de memoria del primer nodo. El tamaño máximo de almacenamiento dinámico va a depender de la memoria RAM de la computadora y debe ser consultada antes de pedir espacio de memoria. La ventaja de usar tipos dinámicos de datos es que se puede pedir posiciones de memoria, conforme se van necesitando liberarlas cuando no se necesiten (Cairo & Guardati, 2005).

La programación dinámica lineal, utiliza el concepto de punteros y define un nodo (cada elemento de la estructura dinámica) de la siguiente manera:

```
Struct Nodo {
        Int dato; // tipo de dato entero
        Nodo *siguiente; //puntero del siguiente nodo
        }
```

Las listas enlazadas son estructuras dinámicas de datos que a través de punteros, permite que se pueda adquirir posiciones de memoria a medida que se necesitan y liberarlas cuando ya no se requieren, solucionando el problema de memoria. Al respecto la lista es un conjunto de datos del mismo tipo donde cada elemento debe tener un predecesor y un sucesor, debe hacer un primero y un último elemento que no tiene sucesor, estas listas permiten que los datos ocupen solo la cantidad requerida en cada instante, y su diseño permite que se almacenen en forma no contigua; el problema se reduce a cómo almacenar la dirección del sucesor de cada nodo, para lo cual se hace necesario el conocimiento de punteros, existe un puntero que indica la posición del primer elemento de la lista, que si debe ser guardado en una variable de tipo estática, pero el resto de elementos de la lista serán dinámicas (Bellot, y otros, 2001). Una lista enlazada está formada por 2 partes: un valor o contenido y un puntero al siguiente nodo (Joyanes & Zahonero, 2007).

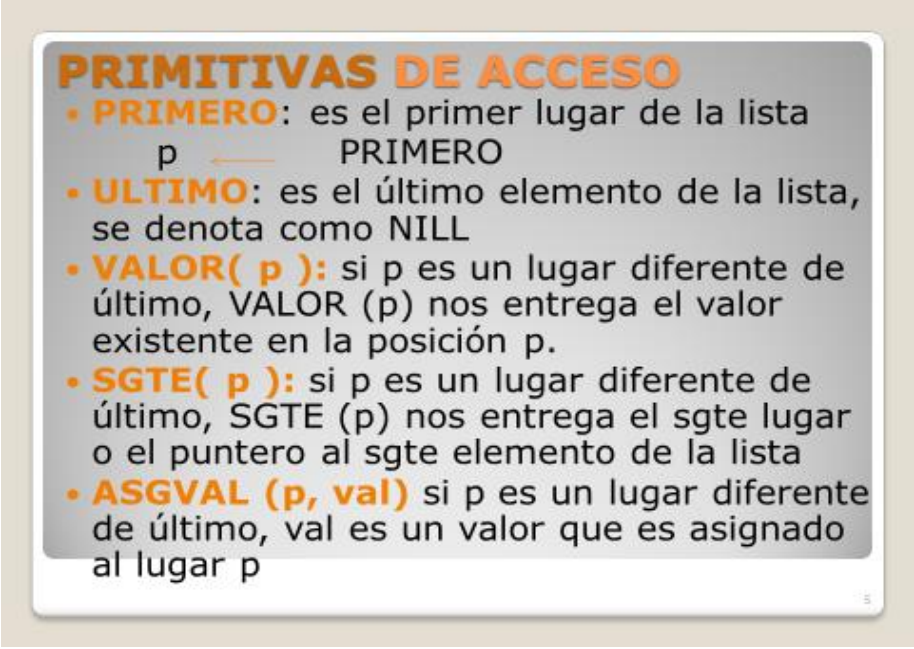
Figura N° 2.8

EJEMPLO DE LISTA ENLAZADA

Fuente: Autor

Figura N° 2.9

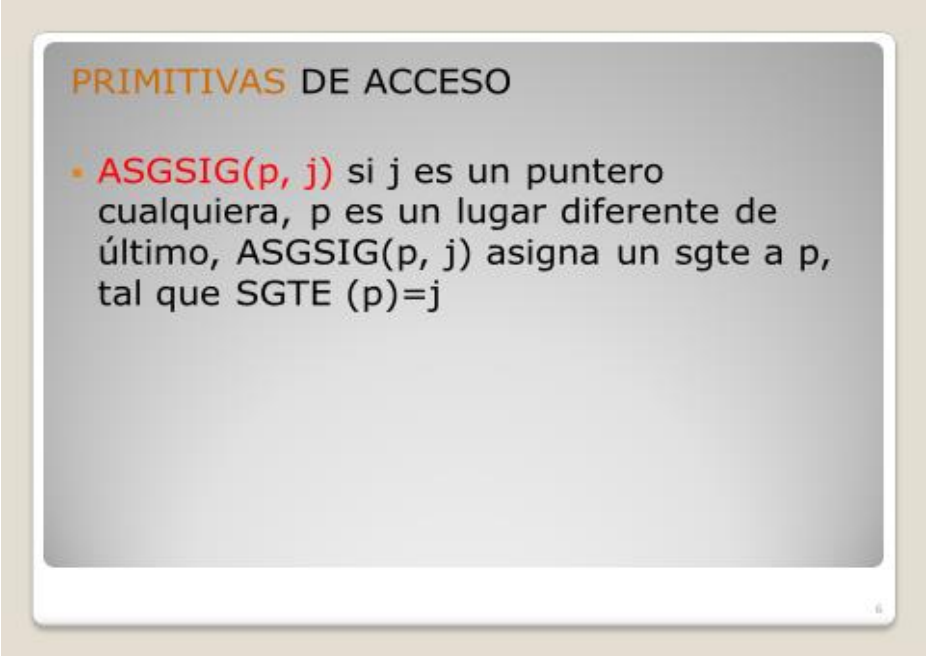
PRIMITIVAS DE ACCESO DE LISTA ENLAZADA



Fuente: Autor

Figura N° 2.10

PRIMITIVAS DE ACCESO DE LISTA ENLAZADA

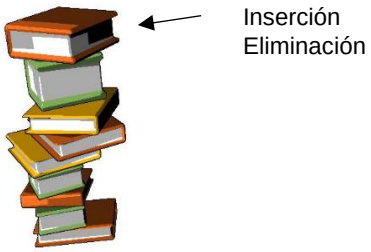


Fuente: Autor

Una pila es un tipo de lista lineal, en la cual la inserción y borrado de nuevos elementos se efectúa sólo por un extremo llamado tope o cima. Como por ejemplo una pila de libros.

Figura N° 2. 11

EJEMPLO DE PILA



Fuente: Autor

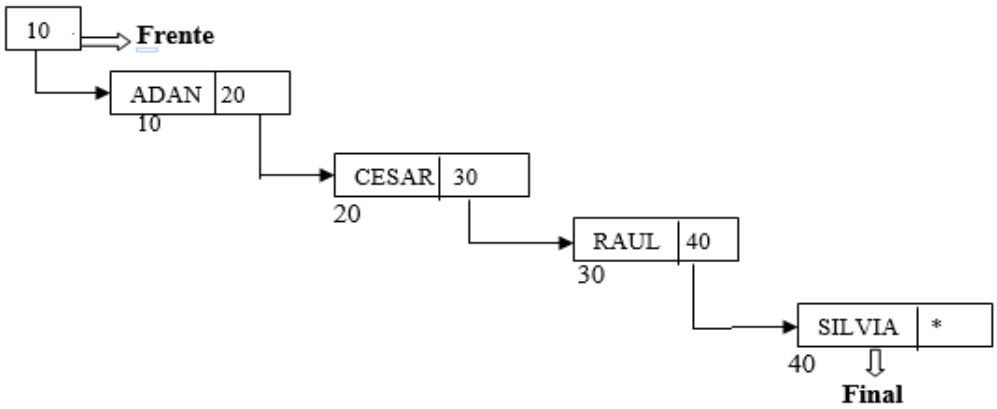
Son estructuras utilizadas muy a menudo como herramientas de programación de tipo LIFO (Last in-First out). El comportamiento de las pilas se caracteriza porque el último elemento en ingresar es el primero en salir. Los

elementos solo pueden eliminarse en el orden inverso al que se insertan. Permiten el acceso solo a un elemento a la vez: el último elemento insertado. La mayoría de los procesadores utilizan una arquitectura basada en pilas, esta es una de las aplicaciones de las pilas. Las pilas son utilizadas en el diseño de compiladores, sistemas operativos, programas de aplicación, etc. En ese aspecto (Joyanes, Rodriguez, & Fernández, Fundamentos de Programación, 1996) subraya que la principal utilidad de una pila es para recuperar una serie de elementos en orden inverso del que se introdujeron. Otra aplicación interesante es la evaluación de expresiones algebraicas mediante pilas.

Se prefiere almacenar la cola en una estructura dinámica, como lo es la lista enlazada, ya que tiene más posibilidades de crecimiento, necesitando más memoria para guardar el enlace y permite que la cola puede crecer, su única limitación es la memoria libre. La otra forma de representación de una cola son los arreglos, que son estructuras estáticas, y tienen limitaciones. Las colas son utilizadas por sistemas del mundo real: En las computadoras conectadas a una impresora, mediante el uso del spooling, para imprimir los trabajos por orden de llegada, también son utilizadas en programas de simulación, colas en organización de viajes, colas en las ventanillas de los Bancos. Etc. En ese aspecto también es usada en el campo informático en la cola de procesamiento de un sistema operativo, en las colas de mensajes o en el intercambio de información de la computadora con otros elementos, unión de 2 colas y formar una nueva sin alterar las originales (Jimenéz & Sánchez, 2002).

Figura N° 2.12

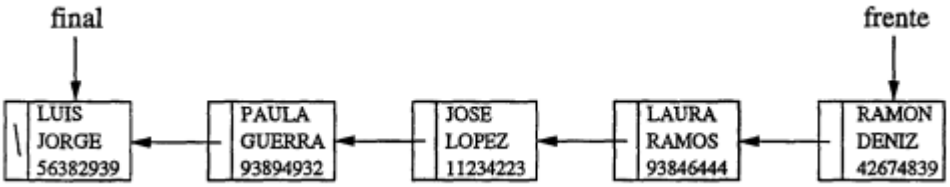
REPRESENTACIÓN GRÁFICA DE UNA COLA, MEDIANTE LISTAS



Fuente: Elaboración propia.

Figura N° 2.13

REPRESENTACIÓN GRÁFICA DE UNA COLA



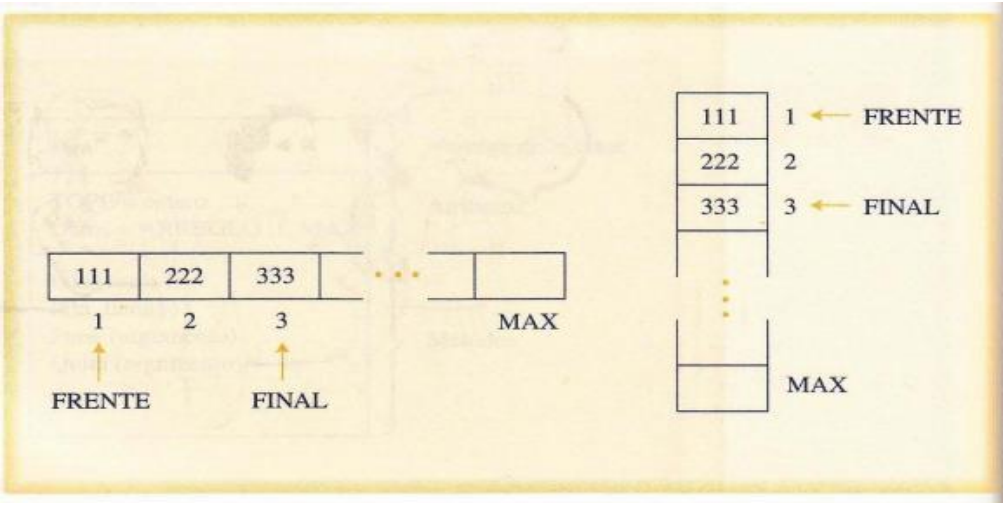
Fuente: (Ojeda, Martel, Ramírez, & Quintana, 1997)

Son estructuras de datos de tipo FIFO (First in - First out). El comportamiento de las colas se caracteriza porque el primer elemento en ingresar es el primero en salir. Una cola almacena elementos en una lista, donde la inserción se realiza por el final y la eliminación se efectúa por el frente de la lista (Joyanes & Zahonero, Estructura de Datos en C++, 2007). Se prefiere almacenar la cola en una estructura dinámica, como lo es la lista enlazada, ya que tiene más posibilidades de crecimiento, necesitando más memoria para guardar el enlace y permite que la cola puede crecer, su única limitación es la memoria libre. La otra forma de representación de una cola son los arreglos, que son estructuras estáticas, y tienen limitaciones.

La cola es utilizado a veces como un buffer, como un depósito donde se pueden almacenar objetos que están en espera de ser procesados por la computadora, es un subtipo de la lista enlazada (Bottazi, y otros). Las colas son utilizadas por sistemas del mundo real: En las computadoras conectadas a una impresora, mediante el uso del spooling, para imprimir los trabajos por orden de llegada, también son utilizadas en programas de simulación, colas en organización de viajes, colas en las ventanillas de los Bancos. Etc. En ese aspecto también es usada en el campo informático en la cola de procesamiento de un sistema operativo, en las colas de mensajes o en el intercambio de información de la computadora con otros elementos, unión de 2 colas y formar una nueva sin alterar las originales (Jimenéz & Sánchez, 2002).

Figura N° 2.14

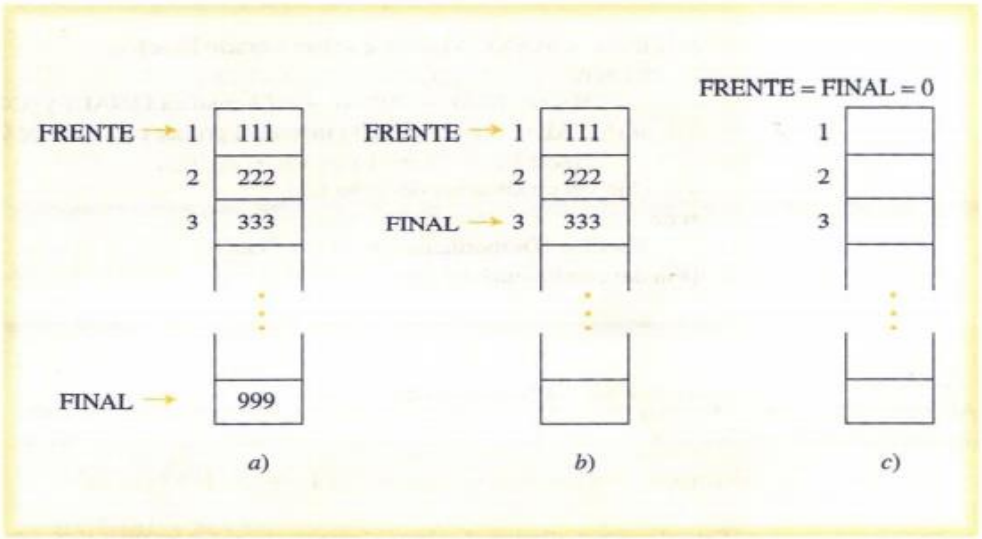
REPRESENTACIÓN GRÁFICA DE UNA COLA, MEDIANTE ARREGLOS



Fuente: (Cairo & Guardati, 2005)

Figura N° 2.15.

EJEMPLOS DE A) COLA LLENA, B) COLA CON ALGUNOS ELEMENTOS Y C) COLA VACÍA



Fuente: (Cairo & Guardati, 2005)

Las primitivas de acceso a colas son:

Frente: [1er elemento] o inicio de la cola

Final : [último elemento] o fin de la cola

X: elemento a insertar o eliminar

MAX: es el máximo nro de elementos

Las operaciones a realizar son:

- Insertar un elemento
- Eliminar un elemento
- Cola vacía
- Cola llena

2.3 Definición de términos básicos

Programación Estática.- Es un estilo de programación de computadoras que resuelve los algoritmos de manera imperativa y mandatoria de manera secuencial, en donde se asigna memoria estática.

Programación Dinámica.- Es un Estilo de programación de computadoras que busca asignar y liberar memoria en cualquier momento de la ejecución de un programa, por lo tanto asigna memoria dinámica.

Punteros.- Es el medio más efectivo para programar en la programación dinámica, ya que los punteros representan la dirección de memoria de todo tipo de variables: enteros, reales, cadenas, arreglos, funciones, clases. Etc.

Listas Enlazadas.- es una estructura dinámica lineal, que mediante punteros las representan, tienen un campo de valor y un campo tipo puntero que asigna el siguiente nodo de la lista enlazada.

Pilas.- Es una estructura dinámica lineal, que tiene un comportamiento tipo LIFO (last in first out) y que es representada por una lista enlazada de comportamiento especial.

Colas.- Es una estructura dinámica lineal, que tiene un comportamiento tipo FIF (first in first out) y que es representada por una lista enlazada de comportamiento especial.

CAPITULO III: HIPÓTESIS Y VARIABLES

HIPÓTESIS

El lenguaje C++ contribuye en el diseño de software de las Estructuras Dinámicas, que son la base para crear Inteligencia Artificial, Sistemas operativos, Teoría de Lenguajes.

VARIABLES

Lenguaje de Programación C++

Indicadores:

Antecedentes de C++

Conceptos de C++

Punteros

Estructuras Dinámicas

Indicadores:

Listas Enlazadas

Pilas

Colas

Diseño de software

Indicadores:

Análisis y especificación del problema.

Diseño de una solución.

Solución Algorítmica

Implementación (codificación).

Pruebas, ejecución, corrección y depuración.

Documentación.

Mantenimiento y evaluación.

.

CAPITULO IV: DISEÑO METODOLÓGICO

4.1 Tipo y diseño de la investigación

El tipo de investigación es cualitativa e inductiva, se busca generar teoría, conforme se investigue el tema y se conozca más el Marco Teórico. De la perspectiva teórica depende lo que estudia la metodología cualitativa, el modo en que lo estudia, y en que se interpreta lo estudiado (Taylor & Bodgan, 2015). Los métodos cualitativos no han sido muy exquisitos y los investigadores cualitativos son flexibles en lo referente al modo en que intentan conducir sus estudios. El investigador es un artífice. Es ideal para investigar problemas sociales y también programas informáticos.

Un marco Conceptual explica los principales aspectos que serán objeto de estudio en una investigación cualitativa. También indican la posición del investigador en el estudio cualitativo y como escribir la versión final del informe. En este aspecto el investigador analiza datos para formar temas o categorías.

Por otro lado, Ballén, Pulido & Zuñiga (2007), Báez & De Tudela (2009) y Bisquerra (2009) (citados por (Escudero & Cortez, 2018)) manifiestan que las investigaciones cualitativa tienen las siguientes características: son inductivas, holísticas, interactivas, reflexivas, rigurosas.

4.2 Método de investigación

La investigación cualitativa registra de forma narrativa los hechos que son objetos de nuestro estudio, fundamentalmente a través de la observación del fenómeno (Escudero & Cortez, 2018). La metodología de la investigación cualitativa se centraliza en los procesos activos que nacen de la experiencia del investigador (Escudero & Cortez, 2018). La investigación cualitativa generalmente utiliza técnicas como: la observación de los fenómenos para su estudio.

Se siguió el siguiente procedimiento: se dividió el tema en 3 partes: tratamiento de listas enlazadas, pilas y colas. Pilas, que son estructuras dinámicas que se pueden representar como arreglos o como punteros. Se ha elegido la representación en base a punteros, porque representa más

objetivamente la asignación de memoria dinámica. Donde es posible asignar espacio en memoria en tiempo de ejecución de manera no limitada.

En cuanto a Colas, que son estructuras dinámicas que se pueden representar como arreglos o como punteros. Se eligió la representación en base a punteros, porque representa de manera práctica la asignación de memoria dinámica y tiene un tipo de programación más amigable. Donde es posible asignar espacio en memoria en tiempo de ejecución de manera no limitada. Se trata de obtener una Metodología que sea asequible y didáctica, que permita enfrentar el reto de la programación de estructuras dinámicas y sus aplicaciones. En ese sentido, uno de los puntos álgidos del Lenguaje C es el manejo de punteros para aquellos que estudian C, un buen conocimiento de punteros maximiza mejores competencias en programación (Ayala, Aguilar, Zarco, & Gómez, 2016).

Los materiales utilizados fueron: una Laptop LENOVO, con procesador INTEL 5i, 12 GB de Memoria RAM, Lenguaje de programación DEV C++. El error potencial del método utilizado es mínimo, ya que son programas informáticos que ejecutan o no ejecutan las instrucciones.

4.3 Población y Muestra

No compete a una investigación cualitativa.

4.4 Lugar de estudio y periodo desarrollado

El lugar de estudio es la Facultad de Ingeniería Industrial y de Sistemas de la Universidad Nacional del Callao. Durante los semestres 2020-I y 2020-II.

4.5 Técnicas e instrumentos para la recolección de la información

Las técnicas e instrumentos para la recolección de la información son: la Observación, análisis del Marco Teórico y Conceptual, procesamiento de la información en el software DEV C++.

Diseño del software

LISTA ENLAZADA

Es necesario las Primitivas de acceso para este tipo de estructura dinámica lineal, que permitirán el manejo de dichas listas, se presentan una serie de casos,

como: Recorrido de una lista, conteo de número de ocurrencias de un elemento dado, búsqueda de valores máximos y mínimos, inserción y eliminación de un elemento (Gallardo & Pérez, 1995).

Algoritmo N° 1: Recorrido de una lista

```
Inicio
      p <- PRIMERO
      Repetir mientras p <> NIL
      Hacer
            Escribir valor (p)
            p<- SGTE (p)
      F_hacer
Fin
```

Algoritmo N° 2: Calculo del número de elementos de una lista:

```
Inicio
      N<- 0
      p <- PRIMERO
      Repetir mientras p <> NIL
      Hacer
        N <- N+1
        p<- SGTE (p)
      F_hacer
      Escribir “N° de elementos es: ”, N
Fin
```

Algoritmo N° 3: Calculo del número de ocurrencias de un elemento dado en una lista:

```
Numocur (VAL, PRIMERO, num)
//VAL es el elemento dado

Inicio
      Leer VAL
      p <- PRIMERO
      num <- 0
      Repetir mientras p <> NIL
      Hacer
        Si valor (p) = VAL entonces
              num = num +1
        f_si
```

```
        p<- SGTE (p)
    F_hacer
    Escribir “N° de ocurrencias de VAL es: ”, num
Fin
```

Algoritmo N° 4: busca un elemento con información X en una lista desordenada formada por enteros. PRIMERO es el puntero al primer nodo de la lista

```
Inicio
    Leer X
    p <- PRIMERO
    Repetir mientras (p <> NIL y VALOR (p) <> X)
    Hacer
            p<- SGTE (p)
    F_hacer
    Si (p=NILL) entonces
            Escribir “El elemento X no fue encontrado”
    Sino
            Escribir “El elemento se encuentra en la lista, en el puntero P”
    F_si
Fin
```

Algoritmo N° 5: busca un elemento con información X en una lista ordenada formada por enteros. PRIMERO es el puntero al primer nodo de la lista

```
Inicio
    Leer X
    p <- PRIMERO
    Repetir mientras (p <> NIL y VALOR(p) < X)
    Hacer
            p<- SGTE (p)
    F_hacer
    Si (p=NILL) entonces
            Escribir “El elemento X no fue encontrado”
    Sino
            Escribir “El elemento se encuentra en la lista, en el puntero p”
    F_si
Fin
```

Algoritmo N° 6: inserta un elemento, al inicio de una lista enlazada

```
Insertar (q, PRIMERO, dato)
// q es el nodo a insertar
// dato es el valor del nodo q
// No se considera el caso de lista vacía
Inicio
        Crea q
        Asgvalor (q, dato)
        Asgsgte (q, p)
        Primero <- q
Fin
```

Algoritmo N° 7: inserta un elemento, al final de una lista enlazada

```
Insertar (q, PRIMERO, dato)
// q es el nodo a insertar
// dato es el valor del nodo
// No se considera el caso de lista vacía

Inicio
      p <- PRIMERO
      Repetir mientras p <> NIL
      Hacer
            p<- SGTE (p)
      F_hacer
      Crea q
      Asgvalor (q, dato)
      Asgsgte (q, NILL)
      Asgsgte(p,q)
Fin
```

Algoritmo N° 8: elimina el primer nodo de una lista enlazada

```
Inicio
      p <- PRIMERO
      si  (p <> NIL && sgte(p) <> NIL) entonces
            p<- SGTE (p)
      sino
            asgsgte(p, NIL)
      F_si
      Quita( p)
      Fin
```

Algoritmo N° 9: que modifica un elemento con información X por la información Y, en una lista desordenada, formada por enteros, PRIMERO es el puntero al primer nodo de la lista

```
Inicio
Leer X,Y
p <- PRIMERO
Repetir mientras (p <> NIL && valor(p) <> X)
Hacer
        p <- sgte(p)
f_hacer
si (p=NIL) entonces
        escribir “X no está en la lista
sino
        asgvalor( p, Y)
fin_si
```

Algoritmo N° 10: Lista enlazada en la que los datos de cada elemento contienen un entero. Determinar si la lista está ordenada.

```
Inicio
      p <- PRIMERO
      Repetir mientras sgte(p) <> NIL y valor (p) <= valor (sgte(p))
      Hacer
            p<- SGTE (p)
      F_hacer
      Si sgte(p) = NIL entonces
            Escribir “la lista está ordenada”
      Sino
            Escribir “la lista está desordenada”
       F_si
Fin
```

PILAS

En este tipo de estructura dinámica, se tomará como representación los punteros. ya que la pila se puede representar como una lista enlazada, donde la inserción y la eliminación se realizan a través de la cima de la pila, que vendría a ser el último nodo insertado en la lista enlazada. En ese sentido (Joyanes & Zahonero, ESTRUCTURA DE DATOS EN C++, 2007) subraya que una pila es un conjunto de elementos que están ordenados a los que solamente se accede

por un extremo de la pila, también llamada cima, como por ejemplo: una pila de platos, una pila de monedas.

Básicamente poseen dos operaciones primarias:

Añadir Datos en la pila

Eliminar elementos en la pila

ALGORITMOS

Algoritmo N° 11: Añadir Datos en la pila

Para Insertar elementos en la pila, se sigue el siguiente algoritmo:

Inicio

Crear el espacio en memoria para almacenar un nodo
Cargar el valor en el nodo
Asignar el puntero pila dentro del nodo (siguiente)
Asignar el nuevo nodo a la pila

Fin

Algoritmo N° 12: Eliminar elementos en la pila

Para quitar elementos en la pila, se sigue el siguiente algoritmo:

Inicio

Crear una variable *aux de tipo nodo
Asignar el n a aux ->dato;
Pasar pila al siguiente nodo
Eliminar aux

Fin

COLAS

La programación dinámica lineal, utiliza el concepto de punteros y define un nodo (cada elemento de la estructura dinámica) de la siguiente manera:

```
Struct Nodo {
        Int dato; // tipo de dato entero
        Nodo *siguiente; //puntero del siguiente nodo
        }
```

Una cola es un tipo de lista lineal, en la cual las eliminaciones se realizan al principio de las colas y las inserciones se realizan en el otro extremo. Ejemplo una cola para subir al bus, cola para atención en el Banco. Una cola es realmente

una lista enlazada, que difiere de estas listas en cuanto a su comportamiento y tipo de operaciones definidas (Jimenéz & Sánchez, 2002).

Tienen 2 operaciones básicas:

Insertar un elemento

Eliminar un elemento

Algoritmo N° 13: Insertar un elemento

1. Crear espacio en una memoria para almacenar un nodo.
2. Asignar ese nuevo nodo al dato que queremos insertar.
3. Asignar

Algoritmo N° 14: Eliminar un elemento

1. Obtener el valor del nodo
2. Crear un nodo aux y asignarle el frente de la cola
3. Eliminar el nodo del frente de la cola

4.6 Análisis y procesamiento de datos

Punteros

Las estructuras dinámicas utilizan el concepto de puntero, para asignar memoria dinámicamente. Es necesario para manipular estos tipos de estructura conocimientos de direcciones de memoria, punteros, estructuras con arreglos.

Programa N° 1: Muestra la dirección y el valor de q

SinNombre1 puntero11.cpp

```
1  // pd7-01 Muestra la dirección y el valor de q
2  #include<iostream>
3  #include<conio.h>
4  #include<stdlib.h>
5  using namespace std;
6  int main()
7  {
8  system("color 7C");
9  int a=10,*p,*q;
10 p=&a;
11 q=p;
12 cout<<"\n En la direccion "<<q<<" esta "<<*q;
13 getch();
14 return 0;
15 }
16
```

Ejecución

```
D:\LABOR\Programaci¾n\ProyectoEstDinamicas2020\

En la direccion 0x6ffdfc esta 10
```

Programa N° 2: Empleo de los operadores & y *

Lista_enlazada3.cpp [*] puntero3.cpp

```
1  //Empleo de los operadores & y *
2  //punte3.cpp
3  #include <iostream>
4  using namespace std;
5   int main()
6   {
7      system("color 7C");
8      int a; //a es unentero
9      int *aPtr; //  aPtr es un apuntador a un entero;
10     a=7;
11     aPtr = &a, //aPtr se establece a la direccion e a;
12
13     cout<<"La direccion de a es "<<&a
14          <<"\nEl valor de aPtr es "<<aPtr;
15     cout<<"\nEl valor de a es "<<a
16          <<"\nEl valor de *aPtr es "<<*aPtr;
17
18     cout<<"\n\nMostrando que * y & son inversos "
19          <<"entre si.\n&*aPtr = "<< &*aPtr
20          <<"\n*&aPtr = "<<*&aPtr<<endl;
21     system("pause");
22     return 0;
23
24   }
```

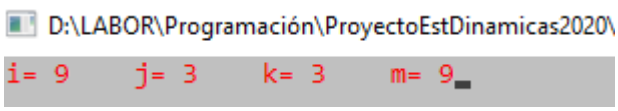
Ejecución

```
D:\LABOR\Programaci¾n\ProyectoEstDinamicas2020\prograr
La direccion de a es 0x6ffe0c
El valor de aPtr es 0x6ffe0c
El valor de a es 7
El valor de *aPtr es 7

Mostrando que * y & son inversos entre si.
&*aPtr = 0x6ffe0c
*&aPtr = 0x6ffe0c
Presione una tecla para continuar . . .
```

Programa N° 3: Aritmética de Punteros

```
// :mostrar el uso de punteros-  punte5
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
system("color 7C");
 int i,j,k,m;
int *p;
j=3;
p=&j; //p almacena la direccion de j;
k=*p; //k recibe el valor almacenado en p, es decir 3
i=*p * j;   // i=3*3
m=*p * *p; //m=3*3
cout<<" i= "<<i<<"\t"<<" j= "<<j<<"\t"<<" k= "<<k<<"\t"<<" m= "<<m;
getch();
return 0;
}
```

Ejecución



ESTRUCTURAS

Estructuras, usando punteros.

Programa N° 4: Define una estructura y accesa al contenido a través del puntero

// Define la struct Alumno y una variable de tipo puntero asignándole la dirección de alumno.

```
struct Alumno{
    char nombre[30];
    int edad;
}alumno, *puntero_alumno =  &alumno;
```

// El nombre y edad son ingresados usando punteros

```
void ingresarDatos() {
    cout<<"Digite su nombre: ";
    cin.getline(puntero_alumno->nombre,30,'\n');
    cout<<"Digite su edad: ";
    cin>> puntero_alumno -> edad;
    }
```

// Muestra el contenido de alumno a través del puntero

```
22  void mostrar(Alumno * puntero_alumno) {
23      cout<<"\nNombre: "<<puntero_alumno -> nombre<<endl;
24      cout<<"\nEdad: "<<puntero_alumno -> edad<<endl;}
25
```

estructuras.cpp

```
1   #include<iostream>
2   #include <conio.h>
3   using namespace std;
4   struct Alumno{
5       char nombre[30];
6       int edad;
7   }alumno, *puntero_alumno =  &alumno;
8   void ingresarDatos();
9   void mostrar(Alumno *);
10  int main(){
11      ingresarDatos();
12      mostrar(puntero_alumno);
13      getch();
14      return 0;
15  }
16  void ingresarDatos() {
17      cout<<"Digite su nombre: ";
18      cin.getline(puntero_alumno->nombre,30,'\n');
19      cout<<"Digite su edad: ";
20      cin>> puntero_alumno -> edad;
21      }
22  void mostrar(Alumno * puntero_alumno) {
23      cout<<"\nNombre: "<<puntero_alumno -> nombre<<endl;
24      cout<<"\nEdad: "<<puntero_alumno -> edad<<endl;}
25
26
```

Resultado de la compilación | Depurar | Ver Resultados | Cerrar

```
Compilation results...
--------
Errors: 0
Warnings: 0
Output Filename: D:\LABOR\Programación\ProyectoEstDinamicas2020\
Output Size: 1.83289623260498 MiB
Compilation Time: 1.92s
```

S

Ejecución

```
D:\LABOR\Programaci¾n\ProyectoEstDinamicas2020\
Digite su nombre: Ana María
Digite su edad: 15

Nombre: Ana María

Edad: 15
```

LISTAS ENLAZADAS

Definiremos algunos conceptos que coadyuven al procesamiento de la información, se ha utilizado el lenguaje DEV C++.

A. Declaración del nodo

Cada elemento, que llamaremos «nodo» estará definido por un struct (o registro) así:

```
struct nodo{
    int dato;
    nodo *siguiente;
};
```

Hay dos elementos en el registro:

- Un elemento llamado “dato” de tipo entero.
- Un puntero “siguiente” al próximo nodo.

B. Lista vacía

Definimos la lista inicialmente vacía así:

```
nodo *lista = NULL;
```

C. Adición de elementos

- **Por la cabeza del nodo**

Reserva memoria para crear el nodo:	**new nodo( );**
En el campo dato se inserta n:	**nuevo_nodo ->dato = n;**
El campo puntero apunta a la lista:	**nuevo_nodo ->siguiente = lista;**
Al nodo creado se le pondrá el nombre lista:	**lista = nuevo_nodo;**

```
void insertar_lista(nodo *&lista,int n){
    nodo *nuevo_nodo = new nodo();
    nuevo_nodo->dato = n;
    nuevo_nodo->siguiente = lista;
    lista = nuevo_nodo;
}
```

- **Por la cola del nodo**

```
void insertar_lista(nodo *&lista,int n){
    nodo *nuevo_nodo = new nodo();
    nodo *aux;
    nuevo_nodo->dato = n;
    nuevo_nodo->siguiente = NULL;

    if(lista == NULL){
        lista = nuevo_nodo;
    }
    else{
        aux = lista;

        while(aux->siguiente != NULL){
            aux = aux->siguiente;
        }

        aux->siguiente = nuevo_nodo;
    }
}
```

Secuencia gráfica

Búsqueda en listas enlazadas

```
void buscar_lista(nodo *lista,int n){
    bool band = false;
    nodo *actual = new nodo();
    actual = lista;

    while(actual != NULL){
        if(actual->dato == n){
            band = true;
        }

        actual = actual->siguiente;
    }

    if(band == true){
        cout<<n<<" SI fue encontrado en lista";
    }
    else{
        cout<<n<<" NO fue encontrado en lista";
    }
}
```

D. Eliminación de un nodo

```
void eliminar_nodo(nodo *&lista,int n){
    if(lista != NULL){
        nodo *aux = lista;
        nodo *anterior = NULL;

        while((aux != NULL) && (aux->dato != n)){
            anterior = aux;
            aux = aux->siguiente;
        }

        if(aux == NULL){
            cout<<"Elemento no encontrado";
        }
        else if(anterior == NULL){
            lista = lista->siguiente;
            delete aux;
        }
        else{
            anterior->siguiente = aux->siguiente;
            delete aux;
        }
    }
}
```

E. Mostrar lista

Se realizar el recorrido de la lista y mostrando cada elemento hasta que el ultimo puntero apunte a NULL.

```
void mostrar_lista(nodo *lista){
    nodo *actual = new nodo();
    actual = lista;

    while(actual != NULL){
        cout<<actual->dato<<" -> ";
        actual = actual->siguiente;
    }
}
```

F. Vaciar lista

Esta función solo elimina un elemento de la lista, por eso se hace mediante un while mientras la lista no esté vacía.

```
void eliminar_lista(nodo *&lista,int &n){
    nodo *aux = lista;
    n = aux->dato;
    lista = aux->siguiente;
    delete aux;
}
```

Programa N° 5: Crea una lista enlazada y la muestra

// Define la estructura nodo: dato y puntero siguiente.

```
5  struct nodo{
6      int dato;
7      nodo *siguiente;
8  };
```

//Envía el contenido del nodo y del dato, crea dos nuevos nodos, asigna a nuevo_nodo -> dato =n y al nuevo_nodo -> siguiente = Null

```
37 void insertar_lista(nodo *&lista,int n){
38     nodo *nuevo_nodo = new nodo(),*aux;
39     nuevo_nodo->dato = n;
40     nuevo_nodo->siguiente = NULL;
```

// si la lista está vacía se le asigna el nuevo nodo caso contrario se guarda en el nodo aux, la lista. Mientras aux->siguiente no sea nulo en aux se asigna el aux -> siguiente. Cuando llega a null, al aux-> siguiente se le asigna el nuevo nodo

```
42     if(lista == NULL){
43         lista = nuevo_nodo;
44     }
45     else{
46         aux = lista;
47
48         while(aux->siguiente != NULL){
49             aux = aux->siguiente;
50         }
51         aux->siguiente = nuevo_nodo;
```

// Envía el puntero lista de tipo nodo y crea un nuevo nodo que apunta a actual. Mientras actual no sea null, muestra el contenido del dato a través del puntero y avanza al siguiente nodo.

```
54 void mostrar_lista(nodo *lista){
55     nodo *actual = new nodo();
56     actual = lista;
57     while(actual != NULL){
58         cout<<actual->dato<<" -> ";
59         actual = actual->siguiente; }
60 }
```

Programa N° 6: Crea una lista enlazada y la muestra

```
1  //crea una lista enlazada y la muestra
2  #include<iostream>
3  #include<conio.h>
4  using namespace std;
5  struct nodo{
6      int dato;
7      nodo *siguiente;
8  };
9
10 void insertar_lista(nodo *&lista,int);
11 void mostrar_lista(nodo *);
12 bool band = false;
13 int main(){
14     nodo *lista = NULL;
15     char resp;
16     int elemento,valor;
17     do{
18         do{
19             cout<<"Ingrese un elemento positivo: ";
20             cin>>elemento;
21
22             if(elemento <= 0){
23                 cout<<"El elemento es incorrecto\n"<<endl;
24             }
25         }while(elemento <= 0);
26         insertar_lista(lista,elemento);
27         cout<<"Desea ingresar otro elemento? (s/n): ";
28         cin>>resp;
29         cout<<endl;
30     }while(resp == 's');
```

```
31      //Muestra la lista enlazada
32      cout<<"elementos de la lista actual:"<<endl;
33      mostrar_lista(lista);
34      getch();
35      return 0;
36  }
37  void insertar_lista(nodo *&lista,int n){
38      nodo *nuevo_nodo = new nodo(),*aux;
39      nuevo_nodo->dato = n;
40      nuevo_nodo->siguiente = NULL;
41
42      if(lista == NULL){
43          lista = nuevo_nodo;
44      }
45      else{
46          aux = lista;
47
48          while(aux->siguiente != NULL){
49              aux = aux->siguiente;
50          }
51          aux->siguiente = nuevo_nodo;
52      }
53  }
54  void mostrar_lista(nodo *lista){
55      nodo *actual = new nodo();
56      actual = lista;
57      while(actual != NULL){
58          cout<<actual->dato<<" -> ";
59          actual = actual->siguiente; }
60  }
```

Ejecución

```
D:\LABOR\Programaci¾n\ProyectoEstDinamicas2020\programaC++
Ingrese un elemento positivo: 2
Desea ingresar otro elemento? (s/n): s

Ingrese un elemento positivo: 5
Desea ingresar otro elemento? (s/n): s

Ingrese un elemento positivo: 1
Desea ingresar otro elemento? (s/n): n

elementos de la lista actual:
2 -> 5 -> 1 ->
```

Programa N° 7: Crea una lista enlazada y determina si la lista esta ordenada

```
#include<iostream>
    #include<stdlib.h>
using namespace std;

struct Nodo{
    int dato;
    Nodo *siguiente;
};

void InsertarLista(Nodo *&lista);
void Orden(Nodo *lista);

int main(){
    system("color 7C");
    Nodo *lista = NULL;

    InsertarLista(lista);
    Orden(lista);

    system("pause");
    return 0;
}

void InsertarLista(Nodo *&lista){
    char resp;

    do{
        Nodo *nuevo_nodo = new Nodo(),*aux1 = lista,*aux2;

        cout<<"\nIngrese un numero: ";cin>>nuevo_nodo->dato;
```

```
30          cout<<"\nIngrese un numero: ";cin>>nuevo_nodo->dato;
31          while(aux1!=NULL){
32              aux2 = aux1;
33              aux1 = aux1->siguiente;
34          }
35          if(lista == aux1){
36              lista = nuevo_nodo;
37          }
38          else{
39              aux2->siguiente = nuevo_nodo;
40          }
41          nuevo_nodo->siguiente = aux1;
42
43          cout<<nuevo_nodo->dato<<" fue ingresado a la lista correctamente."<<endl;
44          cout<<"Desea agregar otro elemento? S/N: ";cin>>resp;
45      }while(toupper(resp) == 'S');
46  }
47
48  void Orden(Nodo *lista){
49      Nodo *aux1 = lista, *aux2 = lista->siguiente;
50      bool band = true;
51
52      while(aux2!=NULL){
53          if((aux1->dato) < (aux2->dato) ){
54              aux1 = aux2;
55              aux2 = aux2->siguiente;
56          }
57          else{
58              band = false;break;
```

os | Resultado de la compilación | Depurar | Ver Resultados

```
59          }
60      }
61
62      if(band == true){
63          cout<<"\nLa lista esta ordenada ascendentemente.\n";
64      }
65      else{
66          cout<<"\nLa lista no esta ordenada.\n";
67      }
68  }
69
```

D:\LABOR\Programaci¾n\ProyectoEstDinamicas2020\e

```
Ingrese un numero:
2
2 fue ingresado a la lista correctamente
Desea agregar otro elemento? S/N: s

Ingrese un numero: 3
3 fue ingresado a la lista correctamente
Desea agregar otro elemento? S/N: s

Ingrese un numero: 5
5 fue ingresado a la lista correctamente
Desea agregar otro elemento? S/N: n

La lista esta ordenada ascendentemente.
Presione una tecla para continuar . . .
```

PILAS

Programa N° 8: inserta elementos en la pila

```
 2  #include<iostream>
 3  #include<conio.h>
 4  #include<stdlib.h>>
 5  using namespace std;
 6  struct Nodo{
 7      int dato;
 8      Nodo *siguiente;
 9  };
10  void insertar_pila(Nodo *&,int );
11
12  int main(){
13      system("color 7C");
14      Nodo *pila = NULL;
15      int n1,n2;
16      cout<<"Digite un numero:";
17      cin>>n1;
18      insertar_pila(pila,n1);
19      cout<<"\nDigite otro numero:";
20      cin>>n2;
21      insertar_pila(pila,n2);
22      getch();
23      return 0;
24      }
25      void insertar_pila(Nodo *&pila,int n)
26      {
27      Nodo *nuevo_nodo = new Nodo();
28      nuevo_nodo->dato = n;
29      nuevo_nodo->siguiente = pila;
30      pila=nuevo_nodo;
31      cout<<"\nElemento "<<n<<" agregado a PILA correctamente";};
```

Sel: 0 Lines: 33 Length: 659 Insertar Done parsing in 0.015 seconds

Al ejecutar:

```
D:\LABOR\Programaci¾n\ProyectoEstDinamicas2020\PILAS_DEVC++\InsertarPila.exe
Digite un numero:5

Elemento 5 agregado a PILA correctamente
Digite otro numero:3

Elemento 3 agregado a PILA correctamente
```

Programa N° 9: quitar elementos de una pila

[*] QuitarPila.cpp

```
1  //sacar elementos de una pila
2  #include<iostream>
3  #include<conio.h>
4  #include<stdlib.h>>
5  using namespace std;
6  struct Nodo{
7      int dato;
8      Nodo *siguiente;
9  };
10 void insertar_pila(Nodo *&,int );
11 void sacar_pila(Nodo *&,int &);
12 int main(){
13     system("color 7C");
14     Nodo *pila = NULL;
15     int dato;
16     cout<<"Digite un numero:";
17     cin>>dato;
18     insertar_pila(pila,dato);
19     cout<<"\nDigite otro numero:";
20     cin>>dato;
21     insertar_pila(pila,dato);
23     cout<<"\n\nSacando los elementos de la pila: ";
24     while(pila!= NULL){// mientras no sea el final de la pila
25         sacar_pila(pila,dato);
26         if (pila != NULL){
27             cout<<dato<<" , ";
28             }
29         else{
30             cout<<dato<<".";
31             }
32         }
33     getch();
34     return 0;
35     }
36     void insertar_pila(Nodo *&pila,int n)
37     {
38     Nodo *nuevo_nodo = new Nodo();
39     nuevo_nodo->dato = n;
40     nuevo_nodo->siguiente = pila;
41     pila=nuevo_nodo;
42     cout<<"Elemento "<<n<<" agregado a PILA correctamente";};
43     void sacar_pila(Nodo *&pila,int &n)
44     {
45     Nodo *aux = pila;
46     n = aux->dato;
47     pila=aux->siguiente;
48     delete aux;
49     };
```

Al ejecutar:

```
D:\LABOR\Programaci¾n\ProyectoEstDinamicas2020\PILAS\PILAS_DEVC++\QuitarPila.exe
Digite un numero:5
Elemento 5 agregado a PILA correctamente
Digite otro numero:10
Elemento 10 agregado a PILA correctamente

Sacando los elementos de la pila: 10 , 5.
--------------------------------
Process exited after 31.17 seconds with return value 0
Presione una tecla para continuar . . .
```

Programa N° 10: inserta y quita elementos de una pila, mientras el usuario lo desee

insertaQuita.cpp

```
1  //agregar elementos a una pila hasta que lo desee el usuario,
2  //mostrarlos, eliminarlos.
3  #include<iostream>
4  #include<conio.h>
5  #include<stdlib.h>>
6  using namespace std;
7  struct Nodo{
8      int dato;
9      Nodo *siguiente;
10 };
11 void insertar_pila(Nodo *&,int );
12 void sacar_pila(Nodo *&,int &);
13 int main(){
14     system("color 7C");
15     Nodo *pila = NULL;
16     int dato;
17     char rpt;
18     do{
19         cout<<"Digite un numero:";
20         cin>>dato;
21         insertar_pila(pila,dato);
22         cout<<"\nDesea agregar otro elemento a PILA(s/n): ";
23         cin>>rpt;
24     }while((rpt=='S')||(rpt=='s'));
```

```
26      cout<<"\n\nSacando los elementos de la pila: ";
27      while(pila!= NULL){// mientras no sea el final de la pila
28          sacar_pila(pila,dato);
29          if (pila != NULL){
30              cout<<dato<<" , ";
31              }
32          else{
33              cout<<dato<<".";
34              }
35          }
36      getch();
37      return 0;
38      }
39      void insertar_pila(Nodo *&pila,int n)
40      {
41      Nodo *nuevo_nodo = new Nodo();
42      nuevo_nodo->dato = n;
43      nuevo_nodo->siguiente = pila;
44      pila=nuevo_nodo;
45      cout<<"Elemento "<<n<<" agregado a PILA correctamente";};
46
47      void sacar_pila(Nodo *&pila,int &n)
48      {
49      Nodo *aux = pila;
50      n = aux->dato;
51      pila=aux->siguiente;
52      delete aux;
53      };
```

Al ejecutar:

```
D:\LABOR\Programaci¾n\ProyectoEstDinamicas2020\PILAS\PILAS_DEVC++\insertaQuit
Digite un numero:
1
Elemento 1 agregado a PILA correctamente
Desea agregar otro elemento a PILA(s/n): s
Digite un numero:3
Elemento 3 agregado a PILA correctamente
Desea agregar otro elemento a PILA(s/n): s
Digite un numero:5
Elemento 5 agregado a PILA correctamente
Desea agregar otro elemento a PILA(s/n): s
Digite un numero:7
Elemento 7 agregado a PILA correctamente
Desea agregar otro elemento a PILA(s/n): n


Sacando los elementos de la pila: 7 , 5 , 3 , 1.
```

Programa N° 11: implementación mediante estructuras dinámicas (menú)

```
1  //implementacion mediante estructuras dinamicas
2  #include <iostream>
3  #include <conio.h>
4  #include <stdlib.h>
5  using namespace std;
6
7  struct Pilas{
8      int dato;
9      Pilas *siguiente;
10 };
11
12 void menu(){
13
14     cout<<"  *********  IMPLEMENTACION DE PILAS EN C++  **********\n\n";
15     cout<<" 1. PONER                                  "<<endl;
16     cout<<" 2. SACAR                                  "<<endl;
17     cout<<" 3. MOSTRAR PILA                           "<<endl;
18     cout<<" 4. DESTRUIR PILA                          "<<endl;
19     cout<<" 5. SALIR                                  "<<endl;
20
21     cout<<"\n INGRESE UNA OPCION: ";
22 }
23
24 void push(Pilas *&pila,int n){
25         Pilas *nueva_pila = new Pilas();
26         nueva_pila->dato = n;
27         nueva_pila->siguiente = pila;
28         pila = nueva_pila;
29 }
30
```

```
31  int pop(Pilas *&pila){
32          int n;
33          Pilas *aux = pila;
34          n = aux->dato;
35          pila = aux->siguiente;
36          delete aux;
37          return n;
38  }
39
40  void mostrar_pila(Pilas *&pila){
41      Pilas *aux;
42      aux = pila;     // apunta al inicio de la lista
43
44      while( aux !=NULL )
45      {
46          cout<<"\t"<< aux->dato <<endl;
47          aux = aux->siguiente;
48      }
49  }
50
51  void destruir(Pilas *&pila){
52      Pilas *aux;
53
54      while( pila != NULL)
55      {
56          aux = pila;
57          pila = aux->siguiente;
58          delete(aux);
59      }
60  }
```

```
int main(){

    Pilas *pila = NULL;
    int opc;
    int n1;

    system("color 7C");

    do{
        system("cls");
        menu();
        cin>>opc;
        switch(opc)
        {
            case 1:
                cout<<"Ingrese un numero: ";
                cin>>n1;
                push(pila,n1);
                cout<<"\n El numero "<<n1<<" fue ingresado en la PILA";
                getch();
                break;
            case 2:
                n1 = pop(pila);
                cout<<"El numero "<<n1<<" fue retirado de la PILA";
                getch();
                break;
```

```
            case 3:
                cout << "\n\n MOSTRANDO PILA\n\n";
                if(pila!=NULL){
                   mostrar_pila(pila); }
                else{
                   cout<<"\n\n\tPila esta vacia..!"<<endl;}
                getch();
            break;


            case 4:
                destruir(pila);
                cout<<"\n\n\tLa Pila ha sido eliminada...\n\n";
                getch();
            break;

             case 5:
                exit (0);
                break;
        }
    }while(opc != 5);

    getch();
    return 0;

}
```

Al Ejecutar:

```
D:\LABOR\Programaci³⁄4n\ProyectoEstDinamicas2020\PILAS\PILAS_DEVC
 *********  IMPLEMENTACION DE PILAS EN C++  *******

1. PONER
2. SACAR
3. MOSTRAR PILA
4. DESTRUIR PILA
5. SALIR

 INGRESE UNA OPCION: 1
Ingrese un numero: 7

 El numero 7 fue ingresado en la PILA
```

```
 INGRESE UNA OPCION: 1
Ingrese un numero: 9

 El numero 9 fue ingresado en la PILA
```

```
 INGRESE UNA OPCION: 1
Ingrese un numero: 12

 El numero 12 fue ingresado en la PILA
```

```
 *********  IMPLEMENTACION DE PILAS EN C++  **

1. PONER
2. SACAR
3. MOSTRAR PILA
4. DESTRUIR PILA
5. SALIR

INGRESE UNA OPCION: 3


MOSTRANDO PILA

       12
       9
       7
```

D:\LABOR\Programaci³¼n\ProyectoEstDinamicas2020\PILAS

```
 *********  IMPLEMENTACION DE PILAS EN C++

1. PONER
2. SACAR
3. MOSTRAR PILA
4. DESTRUIR PILA
5. SALIR

INGRESE UNA OPCION: 2
El numero 12 fue retirado de la PILA
```

```
 *********  IMPLEMENTACION DE PILAS EN C++  *

1. PONER
2. SACAR
3. MOSTRAR PILA
4. DESTRUIR PILA
5. SALIR

INGRESE UNA OPCION: 3


MOSTRANDO PILA

      9
      7
```

D:\LABOR\Programaci³¼n\ProyectoEstDinamicas2020\PIL

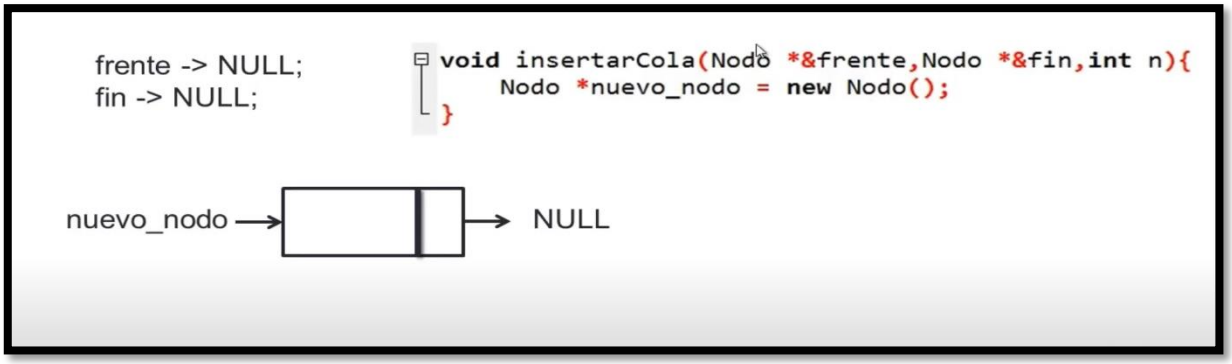
```
 *********  IMPLEMENTACION DE PILAS EN C+

1. PONER
2. SACAR
3. MOSTRAR PILA
4. DESTRUIR PILA
5. SALIR

INGRESE UNA OPCION: 4


       La Pila ha sido eliminada...
```

COLAS

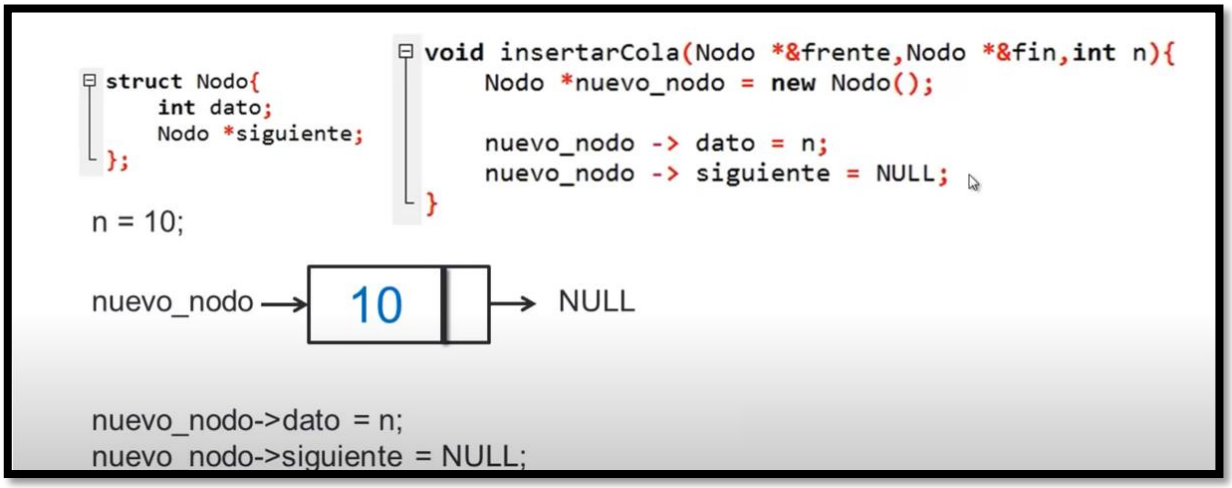
Programa N° 12: Pasos para insertar elementos en una Cola

Para insertar elementos en una cola, solo hay que seguir 3 pasos:

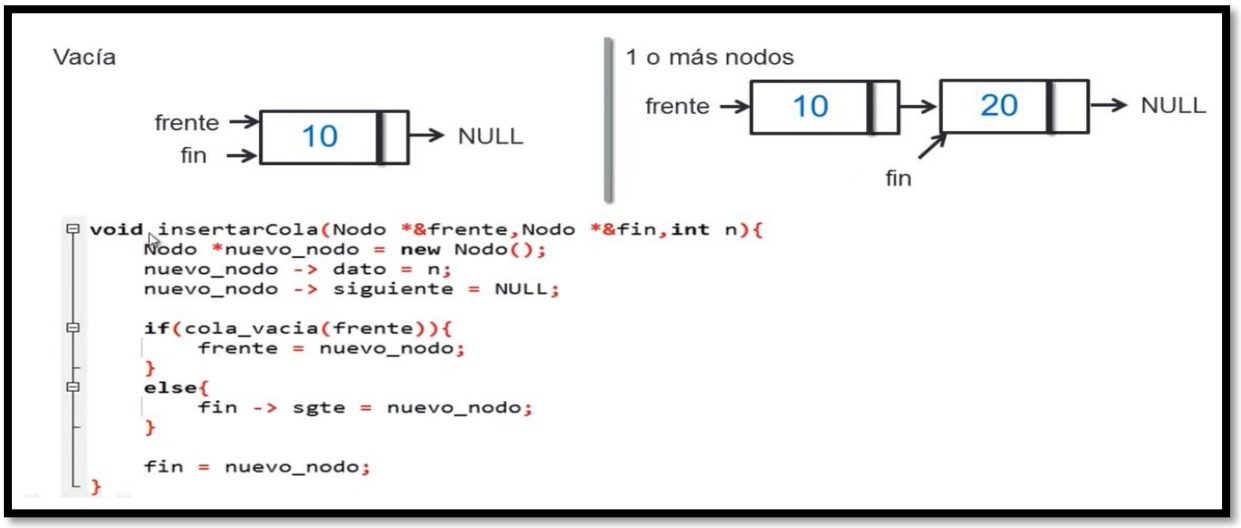
1- Crear espacio en una memoria para almacenar un nodo.



2- Asignar ese nuevo nodo al dato que queremos insertar.



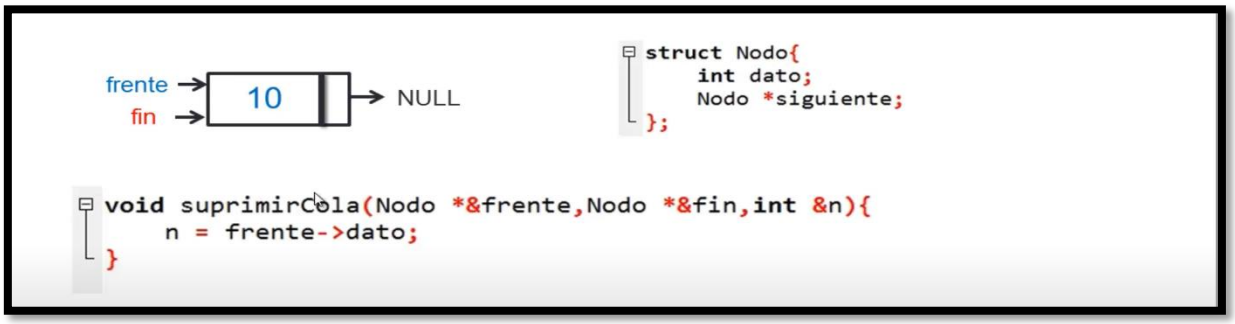
3 - Asignar




```
void insertarCola(Nodo *&frente,Nodo *&fin,int n){
    Nodo *nuevo_nodo = new Nodo();
    nuevo_nodo -> dato = n;
    nuevo_nodo -> siguiente = NULL;

    if(cola_vacia(frente)){
        frente = nuevo_nodo;
    }
    else{
        fin -> sgte = nuevo_nodo;
    }

    fin = nuevo_nodo;
}
```

Programa N° 13: Pasos para eliminar elementos en una Cola

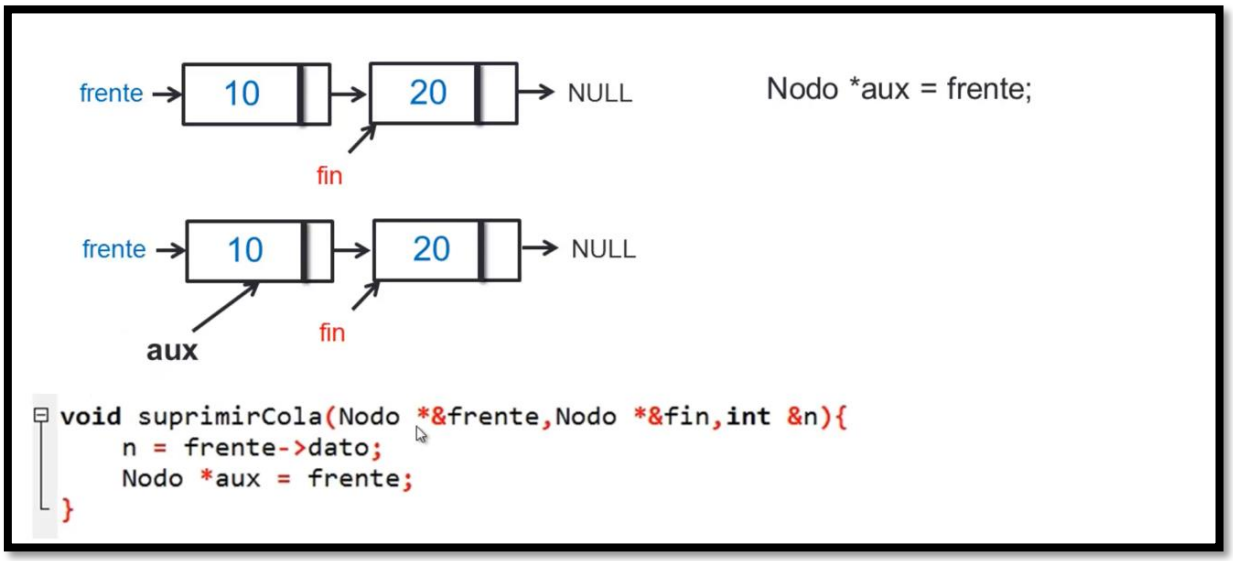
1-Obtener el valor del nodo.




```
struct Nodo{
    int dato;
    Nodo *siguiente;
};
```

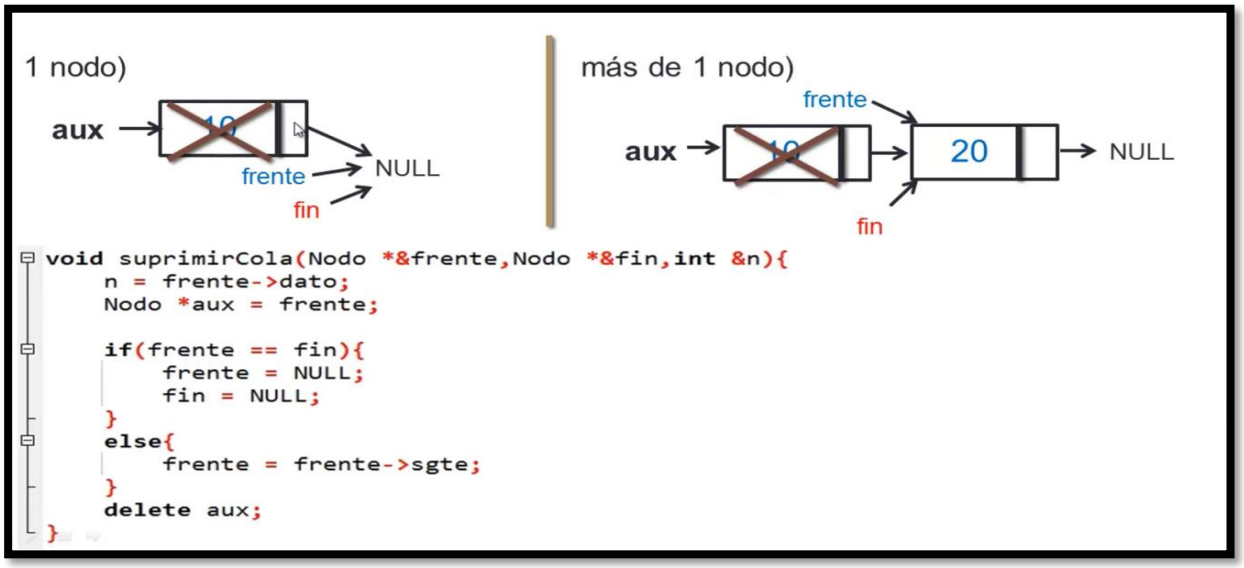
```
void suprimirCola(Nodo *&frente,Nodo *&fin,int &n){
    n = frente->dato;
}
```

2-Crear un nodo aux y asignarle el frente de la cola.




```
void suprimirCola(Nodo *&frente,Nodo *&fin,int &n){
    n = frente->dato;
    Nodo *aux = frente;
}
```

3-Eliminar el nodo del frente de la cola.




```
void suprimirCola(Nodo *&frente,Nodo *&fin,int &n){
    n = frente->dato;
    Nodo *aux = frente;

    if(frente == fin){
        frente = NULL;
        fin = NULL;
    }
    else{
        frente = frente->sgte;
    }
    delete aux;
}
```

Programa N° 14: inserta elementos en la cola

InsertarCola.cpp

```
1   //insertar cola
2   #include<iostream>
3   #include<conio.h>
4   #include<stdlib.h>>
5   using namespace std;
6   struct Nodo{
7       int dato;
8       Nodo *siguiente;
9   };
10  //prototipos
11  void insertar_Cola(Nodo *&, Nodo *&, int );
12  bool colaVacia(Nodo *);
13
14  int main(){
15      system("color 7C");
16      Nodo *frente = NULL;
17      Nodo *fin = NULL;
18
19      int dato;
20
21      cout<<"\nDigite un numero:";
22      cin>>dato;
23      insertar_Cola(frente,fin,dato);
24
25      cout<<"\nDigite un numero:";
26      cin>>dato;
27      insertar_Cola(frente,fin,dato);
28
29      cout<<"\nDigite un numero:";
30      cin>>dato;
31      insertar_Cola(frente,fin,dato);
32
33      getch();
34      return 0;
35      }
36      void insertar_Cola(Nodo *&frente, Nodo *&fin,int n)
37      {
38      Nodo *nuevo_nodo = new Nodo();
39      nuevo_nodo->dato = n;
40      nuevo_nodo->siguiente = NULL;
41
42      if(colaVacia(frente)){
43          frente = nuevo_nodo;
44      }
45      else {
46          fin->siguiente = nuevo_nodo;
47      }
48      fin = nuevo_nodo;
49      cout<<"Elemento "<<n<<" insertado a cola correctamente\n";
50      }
51
52      //función para determinar si la cola está vacía o no
53      bool colaVacia(Nodo *frente){
54          return (frente == NULL)? true: false;
55      }
```

Al ejecutar:

```
D:\LABOR\Programaci¾n\ProyectoEstDinamicas2020\COLAS\ColasDEVC++\InsertarCola.exe

Digite un numero:2
Elemento 2 insertado a cola correctamente

Digite un numero:3
Elemento 3 insertado a cola correctamente

Digite un numero:5
Elemento 5 insertado a cola correctamente
```

Programa N° 15: inserta elementos a la cola y luego los muestra en forma FIFO

Menu_cola.cpp

```
1   /* programa en Dev C++, utiliza colas en el sgte
2   menú:
3       1. Insertar nodos a una cola
4       2. Mostrar el comportamiento FIFO
5       3. Salir
6   */
7   #include<iostream>
8   #include<conio.h>
9   #include<stdlib.h>
10  using namespace std;
11  struct Nodo{
12      char dato;
13      Nodo *siguiente;
14  };
15  //prototipos
16  void menu();
17  void insertarCola(Nodo *&, Nodo *&, char);
18  bool cola_vacia(Nodo *);
19  void suprimirCola(Nodo *&, Nodo *&, char &);
20
21  int main(){
22      system("color 7C");
23      menu();
24      int dato;
25      getch();
26      return 0;
27      }
28
```

Menu_cola.cpp

```
29      void menu()
30      {
31      int opc;
32      char dato;
33
34      Nodo *frente = NULL;     //a frente se le asigna NULL
35      Nodo *fin = NULL;        //a fin se le asigna NULL
36
37      do{ //system("cls");
38          cout<<"\t. MENU:\n";
39          cout<<"1. Insertar nodos a una cola"<<endl;
40          cout<<"2. Mostrar el comportamiento FIFO"<<endl;
41          cout<<"3. Salir"<<endl;
42          cout<<"Opcion: "<<endl;
43          cin>>opc;
44          switch(opc) {
45              case 1: char rpt;
46                      do{
47                      cout<<"\nIngrese el caracter para agregar a la cola: ";
48                      cin>>dato;
49                      insertarCola(frente,fin,dato);
50                      cout<<"\nDesea agregar otro elemento a COLA(s/n):";
51                      cin>>rpt;
52                      }while((rpt=='s')||(rpt=='S'));
53                      break;
```

Menu_cola.cpp

```
54              case 2: cout<<"\nMostrando los elementos de la cola: ";
55                      while(frente != NULL){
56                          suprimirCola(frente, fin, dato);
57                          if(frente != NULL){
58                              cout<<dato<<" , ";
59                          }
60                          else{
61                              cout<<dato<<".";
62                          }
63                      }
64                      cout<<"\n";
65                      system("pause");
66                      break;
67              case 3: break;
68          }
69          //system("cls");
70      }while(opc != 3);
71  }
```

[*] Menu_cola.cpp

```
72  void insertarCola(Nodo *&frente, Nodo *&fin,char n)
73      {
74      Nodo *nuevo_nodo = new Nodo();  //crear un nuevo nodo
75      nuevo_nodo->dato = n;           //a dato se le asigna n
76      nuevo_nodo->siguiente = NULL;   //a siguiente se le asigna NULL
77
78      if(cola_vacia(frente)){
79          frente = nuevo_nodo;        //si cola está vacia el frente es el nuevo_nodo
80      }
81      else {
82          fin->siguiente = nuevo_nodo;//si la cola no está vacia el fin es el nuevo_nodo
83      }
84      fin = nuevo_nodo;
85      cout<<"Elemento "<<n<<" insertado a cola correctamente\n";
86      }
87
88      //función para determinar si la cola está vacía o no
89  bool cola_vacia(Nodo *frente){
90          return (frente == NULL)? true: false;
91      }
```

```
92      //Suprimir el elemento de la Cola por el frente
93  void suprimirCola(Nodo *&frente, Nodo *&fin, char &n)
94      {   n=frente ->dato;
95          Nodo *aux = frente;//en el nodo *aux guarda el frente
96
97          if(frente == fin)
98          {
99              frente = NULL;
100             fin = NULL;
101         }
102         else{//avanza al siguiente nodo
103             frente = frente -> siguiente;
104         }
105         delete aux;//elimina por el frente o inicio de cola
106     }
```

Al ejecutar:

```
        . MENU:
1. Insertar nodos a una cola
2. Mostrar el comportamiento FIFO
3. Salir
Opcion:
1

Ingrese el caracter para agregar a la cola: S
Elemento S insertado a cola correctamente

Desea agregar otro elemento a COLA(s/n):s

Ingrese el caracter para agregar a la cola: O
Elemento O insertado a cola correctamente

Desea agregar otro elemento a COLA(s/n):s

Ingrese el caracter para agregar a la cola: L
Elemento L insertado a cola correctamente

Desea agregar otro elemento a COLA(s/n):n
        . MENU:
1. Insertar nodos a una cola
2. Mostrar el comportamiento FIFO
3. Salir
Opcion:
2

Mostrando los elementos de la cola: S , O , L.
```

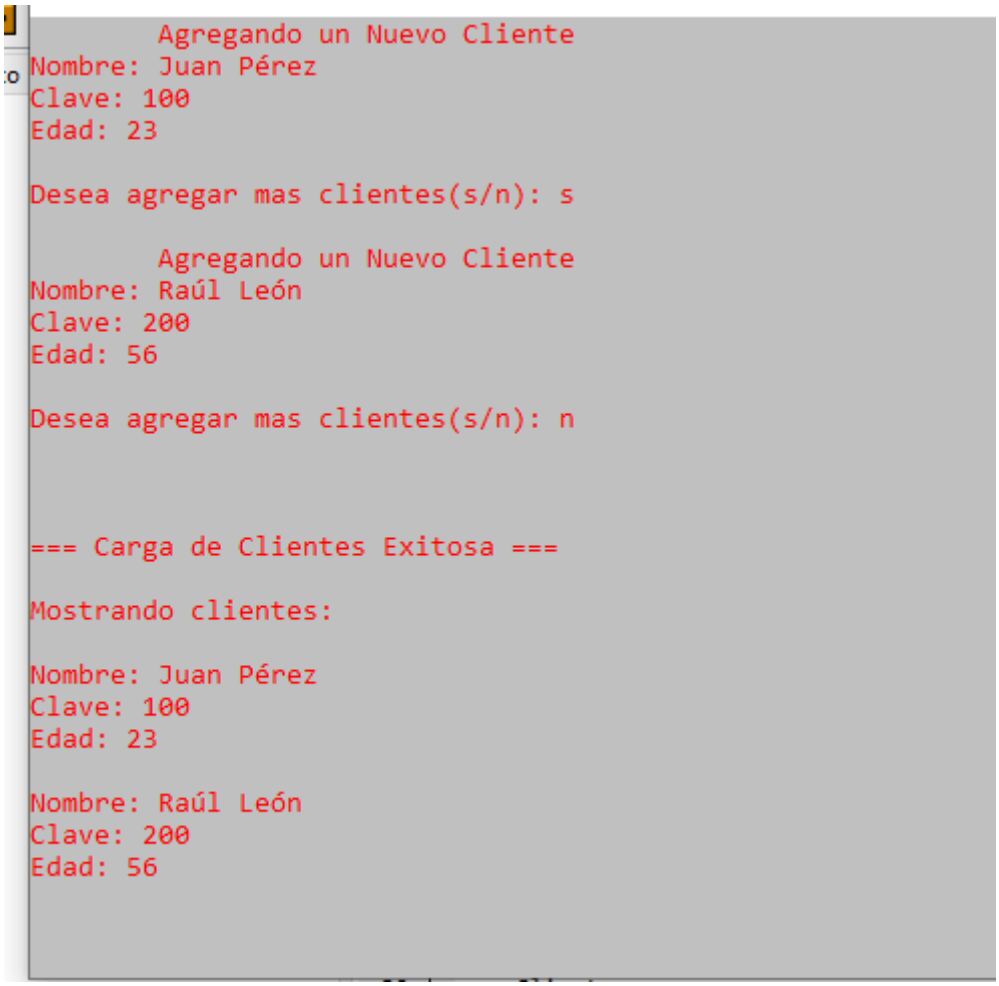
Programa N° 16: inserta elementos a la cola de un banco y luego muestra el comportamiento FIFO

cola-banco.cpp

```cpp
/*Ejemplo: Hacer un programa que guarde datos de clientes de un banco,
los almacene en una cola, y muestre el comportamiento FIFO.*/

#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;

struct Cliente{
    char nombre[30];
    char clave[10];
    int edad;
};

struct Nodo{
    Cliente c;
    Nodo *siguiente;
};

//Prototipos de Funciones
void cargar_cliente(Cliente &);
void insertarCola(Nodo *&,Nodo *&,Cliente);
bool cola_vacia(Nodo *);
void suprimirCola(Nodo *&,Nodo *&,Cliente &);

int main(){
    system("color 7C");
    Nodo *frente = NULL;
    Nodo *fin = NULL;
    Cliente c;
    char rpt;

    do{
        cargar_cliente(c); //Cargamos cliente
        insertarCola(frente,fin,c); //y luego lo agregamos a cola

        cout<<"Desea agregar mas clientes(s/n): ";
        cin>>rpt;
        cout<<"\n";
    }while(rpt == 'S' || rpt == 's');

    cout<<"\n\n=== Carga de Clientes Exitosa ===\n\n";

    cout<<"Mostrando clientes:\n\n";
    while(frente != NULL){//Vaciando la cola
        suprimirCola(frente,fin,c);
        //Mostrando todos los clientes agregados
        cout<<"Nombre: "<<c.nombre<<endl;
        cout<<"Clave: "<<c.clave<<endl;
        cout<<"Edad: "<<c.edad<<endl;
        cout<<"\n";
    }
```

```
53
54      getch();
55      return 0;
56  }
57
58  void cargar_cliente(Cliente &c){
59      fflush(stdin);
60      cout<<"\tAgregando un Nuevo Cliente"<<endl;
61      cout<<"Nombre: "; cin.getline(c.nombre,30,'\n');
62      cout<<"Clave: "; cin.getline(c.clave,10,'\n');
63      cout<<"Edad: "; cin>>c.edad;
64      cout<<"\n";
65  }
66
67  void insertarCola(Nodo *&frente,Nodo *&fin,Cliente c){
68      Nodo *nuevo_nodo = new Nodo();
69
70      nuevo_nodo->c = c;
71      nuevo_nodo->siguiente = NULL;
72
73      if(cola_vacia(frente)){
74          frente = nuevo_nodo;
75      }
76      else{
77          fin->siguiente = nuevo_nodo;
78      }
79
80      fin = nuevo_nodo;
81  }
82
83  bool cola_vacia(Nodo *frente){
84      return (frente == NULL)? true : false;
85  }
86
87  void suprimirCola(Nodo *&frente,Nodo *&fin,Cliente &c){
88      c = frente->c;
89      Nodo *aux = frente;
90
91      if(frente == fin){
92          frente = NULL;
93          fin = NULL;
94      }
95      else{
96          frente = frente->siguiente;
97      }
98
99      delete aux;
100 }
```

Al ejecutar:


Agregando un Nuevo Cliente
Nombre: Juan Pérez
Clave: 100
Edad: 23
Desea agregar mas clientes(s/n): s
Agregando un Nuevo Cliente
Nombre: Raúl León
Clave: 200
Edad: 56
Desea agregar mas clientes(s/n): n
=== Carga de Clientes Exitosa ===
Mostrando clientes:
Nombre: Juan Pérez
Clave: 100
Edad: 23
Nombre: Raúl León
Clave: 200
Edad: 56

Programa N° 17: menú de atención de un banco usando la estructura dinámica cola:

1. Agregar Cliente
2. Atender Cliente
3. Clientes Que Faltan Por Atender
4. Finalizar Atención

Menu_Banco.cpp

```
/* Menú de atención de un banco en una cola.
        1. AGREGAR CLIENTE
        2. ATENDER CLIENTE
        3. CLIENTES QUE FALTAN POR ATENDER
        4. FINALIZAR ATENCION
        5. SALIR              */

#include<conio.h>
#include<stdlib.h>
#include <iostream>
using namespace std;

/*                Estructura de los nodos de la cola
--------------------------------------------------------------------------*/
struct nodo{
    int nro;
    nodo *sgte;
};

/*                        Estructura de la cola
--------------------------------------------------------------------------*/
struct cola{
    nodo *delante;
    nodo *atras  ;
};

/*                        Encolar elemento (anadir cola)
--------------------------------------------------------------------------*/
void encolar( struct cola &q, int valor ){

    nodo *aux = new(nodo);

    aux->nro = valor;
    aux->sgte = NULL;

    if( q.delante == NULL)
        q.delante = aux;    // encola el primero elemento
    else
        (q.atras)->sgte = aux;

    q.atras = aux;         // puntero que siempre apunta al ultimo elemento

}

```

```
/*                          Desencolar elemento (eliminar  cola)
-------------------------------------------------------------------------*/
int desencolar( struct cola &q ){

     int num ;
     nodo *aux ;

     aux = q.delante;       // aux apunta al inicio de la cola
     num = aux->nro;
     q.delante = (q.delante)->sgte;

     delete(aux);           // libera memoria a donde apuntaba aux

     return num;
}

/*                          Mostrar Cola
-------------------------------------------------------------------------*/
void muestraCola( struct cola q ){

     struct nodo *aux;
     aux = q.delante;

     while( aux != NULL )       // mientras aux = q.delante sea diferente de NULL
     {
          cout<<"   "<< aux->nro ;
          aux = aux->sgte;
     }
}

/*              Eliminar todos los elementos de la Cola
-------------------------------------------------------------------------*/
void vaciaCola( struct cola &q){

     struct nodo *aux;

     while( q.delante != NULL){

          aux = q.delante;
          q.delante = aux->sgte;
          delete(aux);
     }

     q.delante = NULL;
     q.atras   = NULL;

}

/*                          Menu de opciones
-------------------------------------------------------------------------*/
void menu(){

    cout<<"\n\t ATENCION DEL BANCO \n\n";
    cout<<" 1. AGREGAR CLIENTE                          "<<endl;
    cout<<" 2. ATENDER CLIENTE                          "<<endl;
    cout<<" 3. CLIENTES QUE FALTAN POR ATENDER          "<<endl;
    cout<<" 4. FINALIZAR ATENCION                       "<<endl;
    cout<<" 5. SALIR                                    "<<endl;
    cout<<"\n INGRESE OPCION: ";
}
```

```
105
106 /*                          Funcion Principal
107 ------------------------------------------------------------------------------*/
108 int main(){
109
110     struct cola q;
111     q.delante = NULL;
112     q.atras   = NULL;
113
114     int dato;  // numero a encolar
115     int op;    // opcion del menu
116     int x ;    // numero que devuelve la funcon pop
117
118     do{
119         menu();
120         cin>> op;
121         switch(op)
122         {
123             case 1:
124                 cout<< "\n AGREGAR CLIENTE: "; cin>> dato;
125                 encolar( q, dato );
126                 cout<<"\n\n\t\tCLIENTE " << dato << " POR ATENDER...\n\n";
127                  break;
128             case 2:
129                  x = desencolar( q );
130                 cout<<"\n\n\t\tCLIENTE "<< x <<" ATENDIDO...\n\n";
131                 break;
132             case 3:
133                 cout << "\n\n CLIENTES QUE FALTAN POR ATENDER\n\n";
134                 if(q.delante!=NULL) muestraCola( q );
135                 else   cout<<"\n\n\tCOLA VACIA...!"<<endl;
136                 break;
137             case 4:
138                 vaciaCola( q );
139                 cout<<"\n\n\t\tBANCO ESTA POR CERRAR..\n\n";
140                 break;
141
142         }
143
144         cout<<endl<<endl;
145         system("pause");  system("cls");
146
147     }while(op!=5);
148
149     return 0;
150 }
```

Al ejecutar:

```
D:\LABOR\Programaci³⁄4n\ProyectoEstDinamicas2020\COLAS\Cola
        ATENCION DEL BANCO

1. AGREGAR CLIENTE
2. ATENDER CLIENTE
3. CLIENTES QUE FALTAN POR ATENDER
4. FINALIZAR ATENCION
5. SALIR

INGRESE OPCION: 1

AGREGAR CLIENTE: 1

                CLIENTE 1 POR ATENDER...


Presione una tecla para continuar . . .
```

```
        ATENCION DEL BANCO

1. AGREGAR CLIENTE
2. ATENDER CLIENTE
3. CLIENTES QUE FALTAN POR ATENDER
4. FINALIZAR ATENCION
5. SALIR

INGRESE OPCION: 1

AGREGAR CLIENTE: 2

                CLIENTE 2 POR ATENDER...


Presione una tecla para continuar . . .
```

```
        ATENCION DEL BANCO

1. AGREGAR CLIENTE
2. ATENDER CLIENTE
3. CLIENTES QUE FALTAN POR ATENDER
4. FINALIZAR ATENCION
5. SALIR

INGRESE OPCION: 1

AGREGAR CLIENTE: 3

                CLIENTE 3 POR ATENDER...


Presione una tecla para continuar . . .
```

```
        ATENCION DEL BANCO

1. AGREGAR CLIENTE
2. ATENDER CLIENTE
3. CLIENTES QUE FALTAN POR ATENDER
4. FINALIZAR ATENCION
5. SALIR

INGRESE OPCION: 3


CLIENTES QUE FALTAN POR ATENDER

  1   2   3

Presione una tecla para continuar . . .
```

```
        ATENCION DEL BANCO

1. AGREGAR CLIENTE
2. ATENDER CLIENTE
3. CLIENTES QUE FALTAN POR ATENDER
4. FINALIZAR ATENCION
5. SALIR

INGRESE OPCION: 2


               CLIENTE 1 ATENDIDO...


Presione una tecla para continuar . . .
```

```
        ATENCION DEL BANCO

1. AGREGAR CLIENTE
2. ATENDER CLIENTE
3. CLIENTES QUE FALTAN POR ATENDER
4. FINALIZAR ATENCION
5. SALIR

INGRESE OPCION: 3


CLIENTES QUE FALTAN POR ATENDER

  2   3

Presione una tecla para continuar . . .
```

CAPITULO V: RESULTADOS

5.3 Otro tipo de Resultados

En lo referente a las listas enlazadas se han analizado en forma algorítmica y en programación Dev C++ todos los componentes de una estructura dinámica, desde los temas de: punteros, punteros a una estructura, listas enlazadas, pilas, colas y árboles. Todo lo cual ha sido probado tanto en forma algorítmica como en programación.

En cuanto a pilas, se han analizado en forma algorítmica y en programación Dev C++ la estructura lineal Pila. Se ha elegido representar a la Pila como una lista enlazada, donde la inserción y la eliminación se realiza por la cima, todo lo cual ha sido probado tanto en forma algorítmica como en programación. Esta forma de representación permite el uso del concepto de memoria dinámica, ya que se está asignando memoria en tiempo de ejecución, que es uno de los objetivos de la programación dinámica.

Cuando la Pila se implementa en una lista enlazada, cada elemento de la pila forma un nodo de la lista; la lista crece o decrece según se añaden o se extraen elementos; ésta es una representación dinámica y no existe limitación en su tamaño.

En el tema de colas, se ha analizado en forma algorítmica y en programación Dev C++ la estructura lineal Cola. Se ha elegido representar a la Cola como una lista enlazada, donde la inserción se realiza por la cola y la eliminación se realiza por la cima, todo lo cual ha sido demostrado tanto en forma algorítmica como en programación Dev C++. Esta forma de representación permite el uso del concepto de memoria dinámica.

Cuando la cola se implementa en una lista enlazada, cada elemento de la cola forma un nodo de la lista; la lista crece o decrece según se añaden o se extraen elementos según el comportamiento FIFO de las colas. La escritura en el valor del nodo equivale a la Inserción del nodo, creándose un nuevo nodo y de la misma manera las lecturas se entienden como una eliminación del nodo leído, el frente avanza al siguiente nodo y se guarda en un nodo auxiliar antes de ser borrado. Asimismo se debe tener presente que es necesario definir un nodo Frente y un nodo Final.

Quedó demostrado que el uso del Lenguaje DEV C++, de punteros y listas enlazadas en la programación dinámica, contribuyen significativamente en la viabilidad de su programación informática, disminuyendo su complejidad.

CAPITULO VI: DISCUSIÓN DE RESULTADOS

Después de la revisión del Marco teórico, se observa que todas las estructuras dinámicas lineales son representadas como listas, adaptándose a la estructura que se desea programar, así si es una pila se puede representar como una lista enlazada donde su inserción y eliminación es por único punto llamado cima, asimismo las colas son también listas enlazadas, con la peculiaridad de que sus elementos se añaden por un extremo y se extraen por el otro extremo.

Los resultados en el tema de pilas, nos indican que es posible la programación en Dev C++ de las estructuras dinámicas, en este caso de las pilas, de una forma amigable y que esté al alcance de los alumnos de Ingeniería de Sistemas, y de esta manera puedan llegar a construir aplicativos que requieran este tipo de estructuras.

Los resultados en cuanto a colas, nos indican que es posible la programación en Dev C++ de la estructura dinámica Cola, de una forma didáctica y que sea comprensible por el alumno. y de esta manera puedan llegar a construir aplicativos con este tipo de estructuras. Es importante la creación de una metodología que permita la programación de las estructuras dinámicas y sean incorporadas al conocimiento de los estudiantes, por la poca información existente.

Otra forma de representación de las estructuras dinámicas son los arreglos, pero la hemos descartado, porque finalmente un arreglo utiliza memoria estática, es decir que es limitada y que debe ser asignada antes de que se ejecute el programa, este es el motivo por el cual hemos usado las listas enlazadas, que asignan memoria en tiempo de ejecución

CONCLUSIONES

1. Esta investigación me ha llevado a determinar la importancia de liberar memoria y reservarla para tiempo de ejecución.

Se puede estandarizar el tratamiento a nivel de programación de las estructuras dinámicas lineales utilizando una representación única, mediante nodos de listas enlazadas que tienen diferente tipo de comportamiento.

2. La bibliografía de este tema es escasa, asimismo hay muy pocos artículos científicos que investigan el tema, es un tema de naturaleza muy abstracta. Sabemos la importancia de las listas dinámicas en la construcción de sistemas operativos, inteligencia artificial y software de alto nivel; es programado en los syllabus de los cursos de algoritmos y programación de las diferentes carreras de Ingeniería que llevan cursos de programación, pero casi nadie cubre el tema.

3. La investigación que se está realizando es muy importante para el uso de Estructuras Dinámicas, hay nuevos avances en la programación de las asignaciones de memoria dinámica, que en la década pasada resultaba poco entendible, muy abstracta. Ahora se está tratando de encontrar otras formas de asignación dinámica más asequibles, que permitan estar al alcance de la comunidad informática.

RECOMENDACIONES

1.- Incorporar definiciones más simples y didácticas de punteros, nodos, listas enlazadas que permitan la manipulación más simple de las estructuras dinámicas lineales.

2.- Capacitar a los Docentes de los cursos que utilizan estas estructuras en su aplicación tanto a nivel algorítmico como de programación, ya que son necesarias como soportes teóricos en el avance de las ciencias y nuevas tecnologías, para que estos a su vez repliquen dichos conocimientos y experiencias a sus alumnos.

3.- Utilizar como forma de evaluación de estos temas un producto final, que pueda ser valorada con rúbricas y permita medir las competencias en dichas materias.

REFERENCIAS BIBLIOGRÁFICAS

Ayala, J., Aguilar, I., Zarco, A., & Gómez, H. (2016). *Memoria dinámica en el Lenguaje de Programación C* (primera ed.). México: Editorial Centro de estudios e investigaciones.

Bellot, A., Pascual, F., Largo, F., Domenech, M., Lizán, F., Ortín, F., . . . Cuerda, R. (2001). *Fundamentos de Programación. Vol. II. Lenguajes.*

Bottazi, C., Costarelli, S., D'Elia, J., Dalcin, L., Galizzi, D., Giménez, J., & ...&Romeo, L. (s.f.). *Algoritmos y Estructuras de Datos.* Universidad Nacional del Litoral-Facultad de Ingenieria y Ciencias Hídricas-Centro de Investigación de Métodos Computacionales. Obtenido de https://cimec.org.ar/~mstorti/aed/aednotes.pdf

Cairo, O., & Guardati, S. (2005). *Estructura de Datos* (Tercera ed.). México: Edi.McGraw Hill.

Escudero, C., & Cortez, L. (2018). *Técnicas y métodos cualitativos para la investigación científica - ISBN: 978-9942-24-092-7* (Primera edición en español ed.). MACHALA, Ecuador: © Editorial UNIVERSIDAD TÉCNICA DE MACHALA,.

Gallardo, M., & Pérez, T. (1995). *Estructura de Datos- un enfoque algoritmico* (primera ed.). Lima: Facultad de Ciencias Matemáticas de la UNMSM. impreso en la oficina de Publicaciones e impresiones.

Jimenéz, F., & Sánchez, G. (2002). *Algoritmos y Estructura de Datos. Implementaciones en Java.* (U. d.-D. Comunicaciones, Ed.) España.

Joyanes, L., & Zahonero, I. (2007). *Estructura de Datos en C++* (Primera ed.). Madrid, España: McGRAW-HILL/INTERAMERICANA DE ESPAÑA, S. A. U.

Joyanes, L., & Zahonero, I. (2007). *ESTRUCTURA DE DATOS EN C++* (Primera ed.). Madrid, España: McGRAW-HILL/INTERAMERICANA DE ESPAÑA, S. A. U.

Joyanes, L., Rodriguez, L., & Fernández, M. (1996). *Fundamentos de Programación* (primera ed.). Madrid: McGraw-Hill.

Martinez, V. (2006). Capítulo 6 - Estructuras de Datos Dinámicas. En V. Martinez.

Ojeda, C., Martel, E., Ramírez, C., & Quintana, M. (1997). *PROGRAMACIÓN Estructuras de Datos Dinámicas.* España: Universidad de Las Palmas de Gran Canaria. Departamento de Electrónica, Telemática y Automática.

Taylor, S., & Bodgan, R. (2015). *Introduction to qualitative reasearch methods: a guidebook and resourse.*

ANEXOS: **MATRIZ DE CONSISTENCIA**

"DISEÑO DE SOFTWARE PARA EL USO DE ESTRUCTURAS DINAMICAS CON LENGUAJE DE PROGRAMACION C++."

Formulación del problema	Objetivos	Hipótesis	Variables	Metodología
¿Cómo el lenguaje C++ contribuye en el diseño del software de las Estructuras Dinámicas, que son la base para crear Inteligencia Artificial, Sistemas operativos, Teoría de Lenguajes?	**Objetivo General:** Determinar si el lenguaje C++ contribuye en el diseño de software de las Estructuras Dinámicas, que son la base para crear Inteligencia Artificial, Sistemas operativos, Teoría de Lenguajes. **Objetivos Específicos:** Determinar si el lenguaje C++ contribuye en el Diseño de los Algoritmos de las Estructuras Dinámicas, que son la base para crear Inteligencia Artificial, Sistemas operativos, Teoría de Lenguajes. Determinar si el lenguaje C++ contribuye en la programación de las Estructuras Dinámicas, que son la base para crear Inteligencia Artificial, Sistemas operativos, Teoría de Lenguajes.	El lenguaje C++ contribuye en el diseño de software de las Estructuras Dinámicas, que son la base para crear Inteligencia Artificial, Sistemas operativos, Teoría de Lenguajes.	**Lenguaje de Programación C++** **Indicadores:** Antecedentes de C++ Conceptos de C++ Punteros **Estructuras Dinámicas** **Indicadores:** Listas Enlazadas Pilas Colas **Diseño de software** **Indicadores:** Análisis y especificación del problema. Diseño de una solución. Solución Algorítmica Implementación (codificación). Pruebas, ejecución, corrección y depuración. Documentación. Mantenimiento y evaluación.	**Nivel de la investigación:** Es de tipo tecnológica **Tipo de la investigación**: cualitativa **Método:** Método inductivo. **Diseño de la investigación:** Experimental **Universo:** Estructuras dinámicas **Muestra:** Por ser cualitativa no hay estadísticas.