Laboratorio 10

LISTAS

1. Competencias

1.1. Competencias del curso

Conoce, comprende e implementa programas usando punteros y POO del lenguaje de programación C++.

1.2. Competencia del laboratorio

Conoce, comprende e implementa programas usando punteros y POO del lenguaje de programación C++.

2. Equipos y Materiales

- Un computador.
- IDE para C++.
- Compilador para C++.

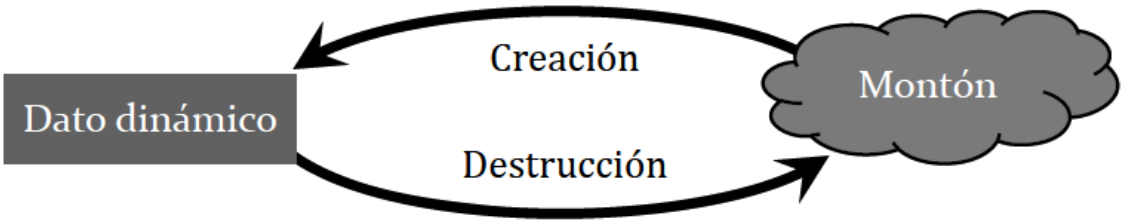
3. Marco Teórico

3.1. Memoria dinámica

Datos dinámicos

Se crean y se destruyen durante la ejecución del programa

Se les asigna memoria del montón:



¿Por qué utilizar memoria dinámica?

Almacén de memoria muy grande: datos o listas de datos que no caben en memoria principal pueden caber en el montón.

El programa ajusta el uso de la memoria a las necesidades de cada momento: ni le falta ni la desperdicia.

¿Cuándo se asigna memoria a los datos?

Datos globales: En memoria principal al comenzar la ejecución del programa. Existen durante toda la ejecución del programa.

Datos locales de un subprograma: En la pila al ejecutarse el subprograma. Existen sólo durante la ejecución de su subprograma.

Datos dinámicos: En el montón cuando el programa lo solicita. Existen a voluntad del programa.

3.2. Datos estáticos vs datos dinámicos

Datos estáticos

Datos declarados como de un tipo concreto:

int i;

Se acceden directamente a través del identificador:

cout << i;

Datos dinámicos

Datos accedidos a través de su dirección de memoria. Esa dirección de memoria debe estar en algún puntero. Los datos estáticos también se pueden acceder a través de punteros

```
int *p = &i;
```

3.2. Punteros y datos dinámicos

El operador new

new tipo: Reserva memoria del montón para una variable del tipo y devuelve la primera dirección de memoria utilizada, que debe ser asignada a un puntero

```
int *p; // Todavía sin una dirección válida
p = new int; // Ya tiene una dirección válida
*p = 12;
```

La variable dinámica se accede exclusivamente por punteros.

No tiene identificador asociado.

```
int i; // i es una variable estática
int *p1, *p2;
p1 = &i;    // Puntero que da acceso a la variable
            // estática i (accesible con i o con *p1)
p2 = new int; // Puntero que da acceso a una variable
            // dinámica (accesible sólo a través de p2)
```

Inicialización con el operador new

El operador new admite un valor inicial para el dato creado:

```
int *p;
p = new int(12);
```

Se crea la variable, de tipo int, y se inicializa con el valor 12

```
#include <iostream>
using namespace std;
#include "registro.h"
int main() {
    tRegistro reg;
    reg = nuevo();
    tRegistro *punt = new tRegistro(reg);
    mostrar(*punt);
    ...
```

El operador delete

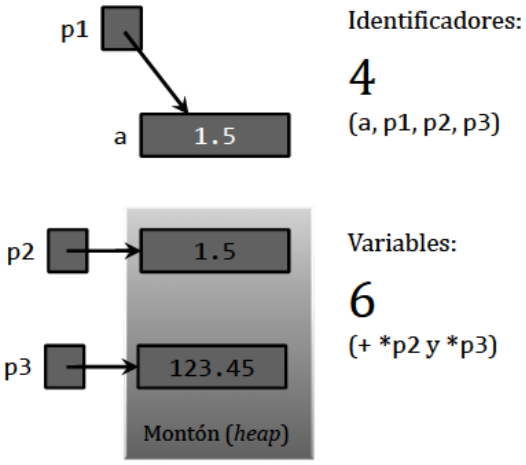
delete puntero: Devuelve al montón la memoria usada por la variable dinámica apuntada por puntero

```
int *p;
p = new int;
*p = 12;
...
delete p; // Ya no se necesita el entero apuntado por p
```

¡El puntero deja de contener una dirección válida!

Ejemplo de variables dinámicas

```
#include <iostream>
using namespace std;
int main() {
    double a = 1.5;
    double *p1, *p2, *p3;
    p1 = &a;
    p2 = new double;
    *p2 = *p1;
    p3 = new double;
    *p3 = 123.45;
    cout << *p1 << endl;
    cout << *p2 << endl;
    cout << *p3 << endl;
    delete p2;
    delete p3;
    return 0;
}
```



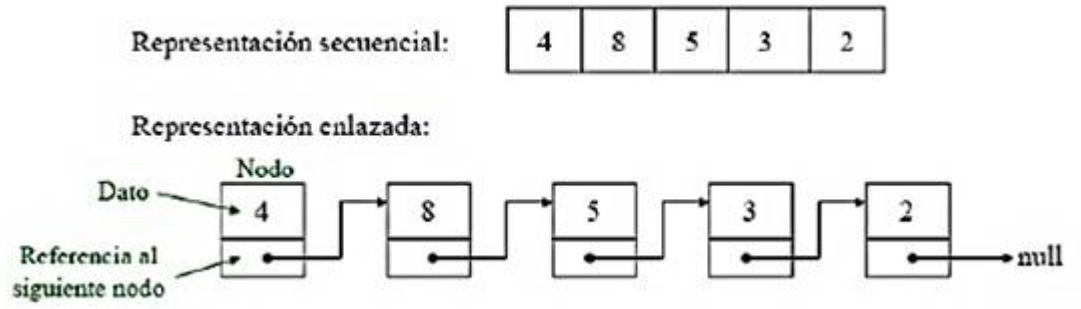
3.3. Listas enlazadas

Una lista lineal enlazada es un conjunto de elementos o objetos de cualquier tipo, originalmente vacía que durante la ejecución del programa va creciendo o decreciendo elementos según las necesidades previstas punto en una lista lineal cada elemento apunta al siguiente, es decir cada elemento tiene información de dónde está el siguiente, por este motivo también se llama lista enlazada.

La forma más simple de estructura dinámica es la lista enlazada. En esta forma los nodos se organizan de modo que cada uno apunta al siguiente, y el último no apunta a nada, es decir, el puntero del nodo vale NULL.

En las listas lineales existe un nodo especial: el primero. Normalmente diremos que nuestra lista es un puntero a ese primer nodo y llamaremos a ese nodo la cabeza de la lista. Eso es porque mediante ese único puntero podemos acceder a toda la lista.

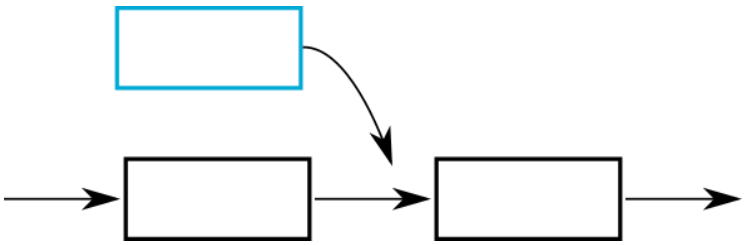
Cuando el puntero que usamos para acceder a la lista vale NULL le diremos que este está vacía.



4. Ejercicios

Resolver los siguientes ejercicios planteados:

1. Defina una lista enlazada que permita insertar elementos al final de todos los elementos que ya se hayan ingresado. Por el momento no es necesario preservar un orden, simplemente los elementos nuevos deben de ingresar como el último elemento.
2. Con la implementación de la lista enlazada anterior, desarrolle una función que permita ingresar los elementos al inicio de todos los demás elementos. Tendrá que modificar el comportamiento del puntero que tiene referencia al primer elemento para que sea redireccionado al nuevo elemento por ingresar.
3. Desarrolle una función que permita ingresar elementos en el medio de dos elementos de la lista enlazada, como se muestra en la siguiente imagen. Solicite que se ingrese una posición válida dentro de la lista y permita que el valor ingresado se pueda anexar en esa posición.



4. Elabore una función que permita eliminar el último elemento de una lista enlazada. (Evite copiar los elementos en una nueva lista para completar la eliminación del elemento)
5. Desarrolle una función que permita eliminar el primer elemento de una lista sin perder referencia de los demás elementos que ya se encuentran almacenados en la estructura .(Evite copiar los elementos en una nueva lista para completar la eliminación de los elementos)
6. Dado una posición válida dentro de la lista, permita al usuario eliminar un elemento de cualquier posición sin perder referencia de los demás elementos.
7. Desarrolle un algoritmo de ordenamiento que permita ordenar los elementos de forma ascendente y descendente de acuerdo a la elección del usuario. Se debe poder simular el ingreso de 10 mil elementos de forma aleatoria y ordenarlos en el menor tiempo posible (< 2 seg).

5. Entregables

Al final estudiante deberá:

1. Compactar el código elaborado y subirlo al aula virtual de trabajo. Agregue sus datos personales como comentario en cada archivo de código elaborado.
2. Elaborar un documento que incluya tanto el código como capturas de pantalla de la ejecución del programa. Este documento debe de estar en formato PDF.
3. El nombre del archivo (comprimido como el documento PDF), será su LAB10_GRUPO_A/B/C_CUI_1erNOMBRE_1erAPELLIDO.

   (Ejemplo: LAB10_GRUPO_A _2022123_PEDRO_VASQUEZ).
4. Debe remitir el documento ejecutable con el siguiente formato:

   LAB10_GRUPO_A/B/C_CUI_ EJECUTABLE_1erNOMBRE_1erAPELLIDO

   (Ejemplo: LAB10_GRUPO_A_EJECUTABLE_2022123_PEDRO_VASQUEZ).

En caso de encontrarse trabajos similares, los alumnos involucrados no tendrán evaluación y serán sujetos a sanción.