# Laboratorio 05

## Punteros

## 1. Competencias

### 1.1. Competencias del curso

Conoce, comprende e implementa programas usando funciones del lenguaje de programación C++.

### 1.2. Competencia del laboratorio

Conoce, comprende e implementa programas usando funciones del lenguaje de programación C++.

## 2. Equipos y Materiales

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

## 3. Marco Teórico

### 3.1. Introducción

Una de las características fundamentales del lenguaje C++ es el concepto de punteros. Los punteros permiten realizar tareas como:

- Acceder a elementos de arrayas
- Pasar argumentos a funciones cuando se necesita modificar un de sus parámetros sin retornar valores.
- Enviar listas de datos a las funciones
- Obtener memoria de forma directa al sistema
- Crear estructuras de datos como listas, arboles, heaps, entre otros.

## 3.2. Direcciones de memoria

Supongamos que tenemos una variable entera con un valor asignado

```
int suma = 100;
```

La dirección (o ubicación) de la variable es donde se encuentra almacenada en la memoria principal de la computadora. Podemos mostrar donde se encuentra con la siguiente sentencia:

```
std::cout << &suma << std::endl;
```

Cuando una variable es declarada, se le asigna una porción de memoria a la variable y esta dirección de la memoria asignada no cambiara a lo largo de la existencia de la variable.

## 3.3. Punteros como Variables

Podemos utilizar punteros como variables. Para declararlo, necesitamos usar el operador asterisco (*) junto a la declaración del tipo de dato para indicar que será un puntero.

```
double *p_decimal;
int *p_entero;
```

En un inicio, estos punteros están sin ningún tipo de dato asignado. Se puede asignar el valor de vacío (*nullptr*) para indicar que por el momento no se desea asignarle un espacio de memoria para trabajar

```
double *p_decimal = nullptr;
```

Con apoyo del operador *new* podemos asignar memoria de acuerdo al tipo de dato declarado en el puntero

```
p_entero = new int;
```

En este caso, el puntero tendrá acceso a una dirección de memoria lista para trabajar con un tipo entero (*int*). Sin esta asignación de memoria, el sistema no tendrá identificado donde encontrar la información apuntada por el puntero. Para acceder a este espacio, tenemos que referenciar el valor del puntero para poder modificarlo.

```
*p_entero = 45;
```

Podemos inspeccionar la dirección de la memoria asignada con el operador &. Si colocamos la siguiente sentencia obtendremos un valor hexadecimal de donde se encuentra físicamente la dirección con la que el puntero se encuentra trabajando actualmente:

```
std::cout << &p << std::endl;|
```

Una vez finalizado el uso de esta variable, podemos forzar su eliminación de la memoria con el operador *delete*. Este operador reclamará la memoria asignada para ser usada en otra actividad que solicite memoria adicional.

```
delete p_entero;
```

**Ejemplo de punteros con variables**

```
int main()
{
    double y, z;
    double *px;

    z = 5.6
    px = &z; //redireccionamos el puntero al valor de z
    y = *px + 1.2
    std::cout << y << std::endl;
}
```

**Modificación de valores de variables**

```
int main()
{
    double y;
    double *px;
    y = 1.0;
    px = &y;
    std::cout << "y = " << y << std::endl;
    *px = 5.9; // Modificacion de la variable y, no de *px
    std::cout << "y = " << y << std::endl;
}
```

## 3.4. Punteros como Vectores

Podemos utilizar los punteros como arrays de una dimensión. Para esto debemos de tratar a los punteros como listas de valores de un determinado tipo.

```
double *mivector;
mivector = new double[10];
```

Para poder acceder a los elementos del puntero, debemos de aplicar el mismo concepto de arrays.

```
#include <iostream>
int main(int argc, char* argv[])
{
        double* x;
        double* y;
        x = new double [10];
        y = new double [10];
        for (int i=0; I <10; i++)
        {
                x[i] = ((double)(i)); //Asignación de un dato al vector x
                y[i] = 2.0*x[i]; // Operación con un elemento de un puntero
        }
        delete[] x;
        delete[] y;
        return 0;
}
```

## 3.5. Matrices

Podemos trabajar con matrices que serán creadas de forma dinámica gracias a los punteros.

```
int filas  = 500, columnas = 40;
int **matriz;

matriz = new int*[filas];
for(int i=0 ; i<filas ; i++)
{
        Matriz[i] = new int[columnas];
}
```

Hay que resaltar que cada fila de la matriz es un array de punteros, los cuales pueden ser de longitud variable. Para poder liberar la memoria asignada a cada uno de los

punteros de las filas, hay que apoyarse en un For para lograr liberar la memoria de forma correcta.

```
for ( int i=0; i<filas ; i++)
{
        delete[] matriz[i];
}
delete[] matriz;
```

## 3.6. Puntero inteligente unique_ptr

Versiones posteriores de C++ (versión C++11 en adelante) proveen un nuevo tipo de punteros que permiten controlar de forma automática la liberación de la memoria asignada a un puntero.

Se procederá de forma automática a reclamar los recursos de memoria de un puntero una vez que ha finalizado su ámbito de ejecución.

```
#include <iostream>
#include <memory>

int main()
{
        std::unique_ptr<int> px(new int);
        *px = 6;

        std::cout << *px;
        return 0; //No es necesario llamar a delete
}
```

## 3.7. Punteros a funciones

Otro de los usos de punteros son los punteros a funciones. El caso más practico que se tiene es el pasar una función como parámetro para otra función.

```
#include <iostream>

int addition (int a, int b)
{ return (a+b); }
int subtraction (int a, int b)
{ return (a-b); }
```

```
int operation (int x, int y, int (*functocall)(int,int)) //Puntero a función como parámetro
{
        int g;
        g = (*functocall)(x,y); //Invocación a la función mediante el puntero
        return (g);
}
int main ()
{
  int m,n;
  int ( *pf )(int,int) = subtraction;

  m = operation (7, 5, addition); //Envió de una función como parámetro
  std::cout << m << std::endl;
  n = operation (20, m, pf ); //Envió de una función como parámetro a través de un puntero
  std::cout << n;
  return 0;
}
```

# 4. Ejercicios

Resolver los siguientes ejercicios planteados:

1. Asignar valores a dos variables enteras, intercambie estos valores almacenados usando solo punteros a enteros.
2. Cree dos vectores con valores flotantes y asígnele valores aleatorios, para esto deberá de asignar memoria a cada vector. Calcule el producto punto de vectores y muestre por pantalla. Una vez finalizado este proceso, retire la memoria asignada a los punteros. Repita este proceso de asignación y retiro de memoria dentro de un for de 1 000 000 veces.
3. Construya una lista enlazada simple utilizando solo punteros. Añada las funciones de insertar y eliminar un elemento. En la función insertar se debe asegurar que los números insertados estén en orden creciente. Simule el proceso con 10000 números aleatorios sin que el programa falle.
4. Construya una lista enlazada que almacene tanto números como cadenas de texto utilizando punteros void. Incluya una función de búsqueda de muestre un dato almacenado además del tipo de dato que se encuentra almacenado (int, float, char, …)
5. Implemente su propia función de concatenación de cadenas de texto especial (¡no es la función ordinaria de concatenación!), recibirá como parámetro dos punteros de

caracteres y tendrá que asignar el contenido del segundo puntero al INICIO del primer puntero. La función no retorna ningún valor.

6. Utilizando punteros a funciones, implemente las funciones:
    a. *void sumar (int a, int b);*
    b. *void restar (int a, int b);*
    c. *void multiplicar (int a, int b);*
    d. *void dividir (int a, int b);*

    Cree un vector de punteros a funciones e implemente un programa que permita la invocación de cada función, pero a través del puntero.

## 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 LAB03_GRUPO_A/B/C_CUI_1erNOMBRE_1erAPELLIDO.

    (Ejemplo: LAB03_GRUPO_A _2022123_PEDRO_VASQUEZ).
4. Debe remitir el documento ejecutable con el siguiente formato:

    LAB03_GRUPO_A/B/C_CUI_ EJECUTABLE_1erNOMBRE_1erAPELLIDO

    (Ejemplo: LAB03_GRUPO_A_EJECUTABLE_2022123_PEDRO_VASQUEZ).

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