Grupo:

- Esteban Llanos
- William Nova
- Juan José Fernández

Informe ClassWork # 3

Actividad 1:

```cpp
#include <iostream>

int main(){
    int numero = 42;

    std::cout << "Valor de la variable 'numero': " << numero << std::endl;

    std::cout << "Dirección de memoria de 'numero': " << &numero << std::endl;

    int* punteroNumero = &numero;
    *punteroNumero = 100;

    std::cout << "Nuevo valor de la variable 'numero': " << numero << std::endl;

    return 0;
}
```

En código muestra como se le asigna un nuevo valor a través de un puntero a la dirección de la variable 'numero' definida como un entero al principio del código.

Salidas o resultados del programa:

```
Valor de la cariable 'numero': 42
Dirección de memoria de 'numero': 0x7ffd4d0c044c
Nuevo valor de la variable 'numero': 100
```

Actividad 2:

```cpp
#include <iostream>

int main(){
    int numero = 42;
    int *punteroNumero = &numero;

    std::cout << "Valor de 'numero' a través del puntero: " << *punteroNumero << std::endl;

    *punteroNumero = 100;

    std::cout << "Nuevo valor de 'numero': " << numero << std::endl;

    int& referenciaNumero = numero;
    referenciaNumero = 200;

    std::cout << "Valor de 'numero' a través de la referencia: " << numero << std::endl;

    return 0;
}
```

Muestra cómo acceder y modificar el valor de una variable, 'numero', de dos maneras distintas: a través de un puntero que apunta a la dirección de memoria de la variable y mediante una referencia que se conecta directamente a la variable. Al final del programa, el valor de 'numero' es 200, lo que se imprime tanto a través del puntero como de la referencia.

Salidas o resultados del programa:

```
Valor de 'numero' a través del puntero: 42
Nuevo valor de 'numero': 100
Valor de 'numero' a través de la referencia: 200
```

Actividad 3:

```cpp
#include <iostream>

int main(){
    const int tamanoArray = 5;
    int numeros[tamanoArray] = {10,20,30,40,50};

    std::cout << "Contenido del array 'numeros': ";
    for (int i = 0; i < tamanoArray; i++)
    {
        std::cout << numeros[i] << " ";
    }
    std::cout << std::endl;

    int* punteroNumeros = numeros;

    punteroNumeros[1] = 99;

    std::cout << "Contenido del array 'numeros' despues de la modificacion: ";
    for (int i = 0; i < tamanoArray; i++)
    {
        std::cout << numeros[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Este código muestra la implementación de arreglos, punteros y cómo se pueden alterar elementos de un arreglo utilizando un puntero. Además, muestra cómo los cambios efectuados a través del puntero tienen un impacto en el arreglo original y cómo se pueden acceder y modificar elementos del arreglo mediante el uso de punteros.

Salidas o resultados del programa:

```
Contenido del array 'numeros': 10 20 30 40 50
Contenido del array 'numeros' despues de la modificacion: 10 99 30 40 50
```

Actividad 4:

```cpp
#include <iostream>

int main() {
    int** matrizDinamica;
    int filas = 3;
    int columnas = 4;

    matrizDinamica = new int*[filas];

    for (int i = 0; i < filas; i++){
        matrizDinamica[i] = new int[columnas];
    }

    int valor = 1;
    for (int i = 0; i < filas; i++){
        for (int j = 0; j < columnas; j++){
            matrizDinamica[i][j] = valor++;
        }
    }

    std::cout << "Matriz Dinámica:" << std::endl;
    for (int i = 0; i < filas; i++){
        for(int j = 0; j < columnas; j++){
            std::cout << matrizDinamica[i][j] << " ";
        }
        std::cout << std::endl;
    }

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

    return 0;
}
```

Este código muestra cómo crear, llenar, mostrar y liberar una matriz especial llamada "matriz dinámica". Comenzamos por definir las dimensiones de la matriz, luego llenamos la matriz con números y luego la mostramos en la pantalla. Finalmente, nos aseguramos de que la memoria utilizada se libere adecuadamente para evitar problemas.

Salidas o resultados del programa:

```
Matriz Dinámica:
1 2 3 4
5 6 7 8
9 10 11 12
```

Extra:

```cpp
#include <iostream>

void obtenerDireccionStack(){
    int stackVariable;
    std::cout << "Direccion del stackVariable: " << &stackVariable << std::endl;

}

int main(){
    obtenerDireccionStack();

    int* heapMemory = new int;
    std::cout << "Direccion de la memoria asignada en el heap: " << heapMemory << std::endl;

    std::cout << "Direccion de la funcion main: " << (void*)main << std::endl;

    delete heapMemory;

    return 0;
}
```

En este código se muestra cómo acceder a la dirección de memoria de una variable almacenada en el stack, la dirección de una asignación de memoria en el heap y también se presenta la dirección de la función 'main'. Se le da importancia a liberar la memoria asignada en el heap para evitar complicaciones en la gestión de la memoria.

Salidas o resultados del programa:

```
Direccion del stackVariable: 0x7fff3eb02934
Direccion de la memoria asignada en el heap: 0x55b81e2832c0
Direccion de la funcion main: 0x55b81cdbf298
```