Trabajo de clase

Descripción general

A continuación se definen los lineamientos de un actividad teórico-práctica para los estudiantes del curso sistemas operativos, en particular, el direccionamiento de memoria.

Objetivos

Durante el desarrollo de de las actividades se logra:

- Identificar la memoria física.
- Aplicar los conocimientos adquiridos en la programación en C++

Antes de empezar

- Verifique la instalación de docker y de la imagen GCC o PUJGCC

En este trabajo de clase, se explorará el concepto de direccionamiento de memoria en C++. A través de una serie de ejercicios, practicarás cómo trabajar con variables, punteros y direcciones de memoria para acceder y modificar datos en la memoria de la computadora.

Actividad No. 1

1. Cree un programa en C++, declare y asigne una variable entera (int)
2. Presente en pantalla la dirección de memoria de esa variable.
3. Modifique el valor de la variable indirectamente utilizando punteros
4. Presente nuevamente el valor de la variable y la dirección de memoria

Actividad No. 2

1. Cree un programa en C++, declare un puntero a una variable .
2. Utilice el puntero para acceder y modificar el valor de la variable.
3. Cree una referencia a la variable y utilicela para modificar el valor
4. Presente las direcciones de memoria del puntero y la referencia

Actividad No. 3

1. Cree un programa en C++, declare un array de números enteros (int)
2. Utilice punteros para acceder a los elementos del array y modificar su contenido.
3. Presente la dirección de memoria del array y del puntero

Actividad No. 4

1. Cree un programa que use la asignación dinámica de memoria con *new*
2. Cree una matriz de enteros dinámica de 2D.
3. Llenen la matriz con datos
4. Libere la memoria con delete cuando haya terminado.

Extra: Presente el stack, el heap y el code usando C++

Entrega: Aloje el código en un repositorio público de Github y envíe un informe de la implementación y las salidas a través del aula digital.

SOLUCIONARIO

Actividad No. 1

```cpp
#include <iostream>

int main() {
    int numero = 42; // Declaración y asignación de una variable entera

    // Imprimir el valor de la variable
    std::cout << "Valor de la variable 'numero': " << numero << std::endl;

    // Imprimir la dirección de memoria de la variable
    std::cout << "Dirección de memoria de 'numero': " << &numero << std::endl;

    // Utilizar un puntero para modificar el valor de la variable
    int* punteroNumero = &numero; // Declaración de un puntero que apunta a 'numero'
    *punteroNumero = 100; // Modificar el valor de 'numero' a través del puntero

    // Imprimir el nuevo valor de la variable
    std::cout << "Nuevo valor de la variable 'numero': " << numero << std::endl;

    return 0;
}

```

Actividad No. 2

```cpp
#include <iostream>

int main() {
    int numero = 42; // Declaración y asignación de una variable entera
    int* punteroNumero = &numero; // Declaración de un puntero que apunta a 'numero'

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

    // Modificar el valor de 'numero' a través del puntero
    *punteroNumero = 100; // Modificar el valor de 'numero' a través del puntero

    // Imprimir el nuevo valor de 'numero'
    std::cout << "Nuevo valor de 'numero': " << numero << std::endl;

    // Crear una referencia a 'numero'
    int& referenciaNumero = numero; // Declaración de una referencia
    referenciaNumero = 200; // Modificar el valor de 'numero' a través de la referencia

    // Imprimir el nuevo valor de 'numero' después de modificarlo con la referencia
    std::cout << "Nuevo valor de 'numero' a través de la referencia: " << numero << std::endl;

    return 0;
}

```

Actividad No. 3

```cpp
#include <iostream>

int main() {
    const int tamañoArray = 5;
    int numeros[tamañoArray] = {10, 20, 30, 40, 50}; // Declaración de un array de enteros

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

    // Declarar un puntero que apunta al inicio del array
    int* punteroNumeros = numeros;

    // Utilizar el puntero para acceder y modificar elementos del array
    punteroNumeros[1] = 99; // Modificar el segundo elemento del array

    // Imprimir el contenido del array después de la modificación
    std::cout << "Contenido del array 'numeros' después de la modificación: ";
    for (int i = 0; i < tamañoArray; i++) {
        std::cout << numeros[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}

```

Actividad No. 4

```cpp
#include <iostream>

int main() {
    int** matrizDinamica; // Declaración de un puntero a puntero para crear una matriz dinámica
    int filas = 3;       // Número de filas de la matriz
    int columnas = 4;    // Número de columnas de la matriz

    // Asignación de memoria para la matriz dinámica
    matrizDinamica = new int*[filas]; // Crear un array de punteros a int (filas)

    for (int i = 0; i < filas; i++) {
        matrizDinamica[i] = new int[columnas]; // Cada puntero apunta a un array de int (columnas)
    }

    // Inicialización de la matriz con valores
    int valor = 1;
    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
            matrizDinamica[i][j] = valor++;
        }
    }

    // Imprimir la matriz
    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;
    }

    // Liberar la memoria asignada
    for (int i = 0; i < filas; i++) {
        delete[] matrizDinamica[i]; // Liberar cada array de int (columnas)
    }
    delete[] matrizDinamica; // Liberar el array de punteros (filas)

    return 0;
}

```

Extra

```cpp
#include <iostream>

// Función para obtener la dirección del stack
void obtenerDireccionStack() {
    int stackVariable;
    std::cout << "Dirección del stackVariable: " << &stackVariable << std::endl;
}

int main() {
    // Llamar a la función para obtener la dirección del stack
    obtenerDireccionStack();

    // Usar new para asignar memoria en el heap
    int* heapMemory = new int;
    std::cout << "Dirección de la memoria asignada en el heap: " << heapMemory << std::endl;

    // Obtener la dirección de una función en el código (segmento de texto)
    std::cout << "Dirección de la función main: " << (void*)main << std::endl;

    // Liberar la memoria asignada en el heap
    delete heapMemory;

    return 0;
}

```