Pointers, Dyamic Memory, and the Rule of Three in C++

1 Pointers in C++

Pointers are variables that store memory addresses of other variables. They are used for dynamic memory management, passing large structures efficiently, building data structures like linked lists, and interacting with hardware.

A memory address is typically shown in hexadecimal, such as `0x7ffee4b6eabc`. Pointers allow us to access and manipulate the memory directly.

Example: Basic Pointer Usage

```
#include <iostream>
using namespace std;

int main() {
    int x = 10;
    int* ptr = &x;  // pointer to x
    cout << "Value of x: " << *ptr << endl;  // dereferencing
    cout << "Address of x: " << ptr << endl;  // prints address of
    x
    return 0;
}
```

2 Dynamic Arrays

Dynamic memory allocation allows creation of arrays whose size is determined at runtime. These arrays are allocated on the heap using the `new` operator. The `new` operator returns a pointer to the beginning of the block of memory.

This is in contrast to compile-time arrays, which have a fixed size known at compile time and are usually stored on the stack. Run-time allocation provides flexibility, especially when the required size is unknown beforehand.

Example: Dynamic Array

```
#include <iostream>
using namespace std;
```

```cpp
int main() {
    int size;
    cout << "Enter array size: ";
    cin >> size;

    int* arr = new int[size];  // dynamically allocated array on
    the heap
    for (int i = 0; i < size; ++i) {
        arr[i] = i * i;
    }

    for (int i = 0; i < size; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;

    delete[] arr;  // free memory
    return 0;
}
```

3 Rule of Three

The Rule of Three states that if a class needs a user-defined destructor, copy constructor, or copy assignment operator, it likely needs all three. This typically arises when a class manages resources like dynamic memory, file handles, or network connections.

- **Destructor**: Releases owned resources to avoid memory leaks.
- **Copy Constructor**: Creates a new object as a copy of an existing object.
- **Copy Assignment Operator**: Assigns values from one existing object to another, handling cleanup and deep copying.

Failing to define all three can result in shallow copies or double deletions, leading to undefined behavior.

Example: Rule of Three

```cpp
class MyVector {
private:
    int* data;
    size_t size;

public:
    // Constructor
    MyVector(size_t n) : size(n) {
        data = new int[size];
        for (size_t i = 0; i < size; ++i) {
            data[i] = 0;
        }
    }
```

```
    // Copy Constructor
    MyVector(const MyVector& other) : size(other.size) {
        data = new int[size];
        for (size_t i = 0; i < size; ++i) {
            data[i] = other.data[i];
        }
    }

    // Copy Assignment Operator
    MyVector& operator=(const MyVector& other) {
        if (this != &other) {
            delete[] data;
            size = other.size;
            data = new int[size];
            for (size_t i = 0; i < size; ++i) {
                data[i] = other.data[i];
            }
        }
        return *this;
    }

    // Destructor
    ~MyVector() {
        delete[] data;
    }

    // Access operator
    int& operator[](size_t index) {
        return data[index];
    }

    const int& operator[](size_t index) const {
        return data[index];
    }

    // Size accessor
    size_t getSize() const {
        return size;
    }
};

#include <iostream>
using namespace std;

int main() {
    // Create a vector of size 5
    MyVector v1(5);
    for (size_t i = 0; i < v1.getSize(); ++i) {
        v1[i] = static_cast<int>(i * 10);
    }

    // Copy constructor
    MyVector v2 = v1;

    // Modify v1
    v1[0] = 999;
```

```
    // Copy assignment
    MyVector v3(3);
    v3 = v2;

    // Print all vectors
    cout << "v1: ";
    for (size_t i = 0; i < v1.getSize(); ++i) {
        cout << v1[i] << " ";
    }
    cout << endl;

    cout << "v2: ";
    for (size_t i = 0; i < v2.getSize(); ++i) {
        cout << v2[i] << " ";
    }
    cout << endl;

    cout << "v3: ";
    for (size_t i = 0; i < v3.getSize(); ++i) {
        cout << v3[i] << " ";
    }
    cout << endl;

    return 0;
}
```