45 - The Arrow Operator in C++

terça-feira, 11 de março de 2025 07:16

```cpp
class Entity45
{
    public:
        int x;
    public:
        void Print() const {std::cout << "Hello!" << std::endl;}
};

class Scoedtr45
{
    private:
        Entity45* m_Obj;
    public:
        Scoedtr45(Entity45* entity) : m_Obj(entity)
        {

        }

        // automated the deletion of that pointer object with this wrapper?
        ~Scoedtr45()
        {
            delete m_Obj;
        }

        Entity45* operator->()
        {
            return m_Obj;
        }
};

struct Vector45
{
    float x, y, z;

};

int main()
{


    std::cout<< "" << std::endl;

    // Getting the ofset of that x
    // // used when serializing data
    // int offsetx = (int)&((Vector45*)nullptr)->x;
    // int offsety = (int)&((Vector45*)nullptr)->y;
    // int offsetz = (int)&((Vector45*)nullptr)->z;

    // std::cout << offsetx << std::endl;
    // std::cout << offsety << std::endl;
    // std::cout << offsetz << std::endl;

    Entity45 e45;
    e45.Print(); // Works fine

    // But if it was a pointer
    Entity45* e45_2 = &e45;
    // e45.Print(); // Can't use the point, we need to derreference that pointer
    Entity45& entity45 = *e45_2;
    entity45.Print(); // works fine after the derreference
    // But we could also do is:
    (*e45_2).Print();  // Direclty derreference it, it's ok and works fine but looks ugly
    e45_2->Print(); // This do the same thing, derreferencing the pointer and calling the function ( shortut instead of manually derraferencing it)

    // We have a class that will delete the obkect when it goes out of scope
    Scoedtr45 se45 = new Entity45();
    se45->Print(); // Error, in this case we need to implement the arrow overload ->



    std::cout<< "" << std::endl;
```