20220408-C++

1.过程描述

virtual和override的简单使用 C++ 复制代码

```cpp
class Printable
{
public:
    virtual string GetClassName() = 0;//pure virtual
};

class Entity:public Printable
{
public:
    virtual string GetName() {
        return "Entity";
    }
    string GetClassName() override
    {
        return "Entity";
    }
};

class Player: public Entity
{
private:
    string m_Name;
public:
    Player(const string& name)
        :m_Name(name){}
    string GetName() override {
        return m_Name;
    }
    string GetClassName() override
    {
        return "Player";
    }
};

void PrintName(Entity* entity)
{
    cout << entity->GetName() << endl;
}

void Print(Printable* obj)
{
    cout << obj->GetClassName() << endl;
}

int  main() {
```

```
46      Entity* e = new Entity();
47      PrintName(e);
48      Player* p = new Player("Hello");
49      PrintName(p);
50
51      Print(e);
52      Print(p);
53  }
```

数组的一点补充 C++ 复制代码

```
1  static const int demosize = 5;//注意必须static const
2  int demo[demosize];
3
4  array<int, 5> another;
```

string literal C++ 复制代码

```
1  string literal也成为字符串常量，以空字符"\0"结尾，由const char组成的字符数
   组.Read-only
2  //换行输出
3  const char* name = R"(line1
4  line2
5  line3
6  line4)";
7  std::cout << name << std::endl;
```

const的用法 C++ 复制代码

```cpp
const int demo = 90;

const int* a = new int;//can not modify the content of the pointer
//*a=2;this would fail
//a = &demo;this would work

int* const b = new int;//can change the content,but can not reassign the actual pointer to point to something else
//*b = 2;this would work
//b = &demo;this would fail

int const* c = new int;
//*c = 3;this would fail
//c = &demo;this would work

const int* const d = new int;
//changing the content or reassign the pointer would fail

------------------------------------------------------------
class Entity
{
private:
    int m_X, m_Y;
    mutable int var;
public:
    int GetX() const //这里的const表示该函数不能修改类成员的值.如果这里不加const,下面的函数将报错
    {
        return m_X;
        var = 3;//这是mutable的特性，能够在const中被修改
    }

};

void PrintEntity(const Entity& e)
{
    cout << e.GetX() << endl;
}
------------------------------------------------------------
mutable的另一种用法
int x = 8;
auto f = [=]() mutable//mutable在lambda中的应用
{
    x++;
    cout << x << endl;
```

```
};
f();
```

两种创建对象的方法 C++ 复制代码

```cpp
class Entity
{
private:
    string m_Name;
public:
    Entity():m_Name("Unknown"){}
    Entity(const string& name):m_Name(name){}
    const string& GetName() const {
        return m_Name;
    }

};

int  main() {
    Entity* e;
    //stack allocation
    {
        Entity entity("Cherno");
        e = &entity;
        cout << e->GetName() << endl;
    }//这种情况下，当代码执行到这个程序块结束时，所创建的对象也就被销毁了（stack allocation）
    //heap allocation
    {
        Entity* entity = new Entity("Cherno");// （heap allocation）更花时间，而且必须delete
        e = entity;
        cout << e->GetName() << endl;
    }
    delete e;

}
```

new C++ 复制代码

```
new主要是用来在heap上进行内存分配的，创建的是一个指针
Entity* e = new Entity();
// Entity* e = (Entity*)malloc(sizeof(Entity));这个不会call the constructor，不要这么做
delete e;
```

隐式转换及显式关键字 C++ 复制代码

```
class Entity
{
private:
    string m_Name;
    int m_Age;
public:
    Entity(const string name):m_Name(name),m_Age(-1) {}
    Entity(int age):m_Name("Unknown"),m_Age(age){}
    //explicit Entity(int age) :m_Name("Unknown"), m_Age(age) {}，如果加了explicit关键字则下面隐式转换都会报错

};

int  main() {
    Entity a("Cherno");
    Entity b(22);

    Entity c=22;//隐式转换
    PrintEntity(22);
    PrintEntity("Cherno");//would not work 因为Cherno不是string
    PrintEntity(string("Cherno"));//this would work
}
```

操作符重载 C++ 复制代码

```cpp
struct Vector2
{
    float x, y;
    Vector2(float x,float y)
        :x(x),y(y){}
    Vector2 Add(const Vector2& other) const
    {
        return Vector2(x + other.x, y + other.y);
    }

    Vector2 operator+(const Vector2& other)
    {
        return Add(other);
    }


    Vector2 Multiply(const Vector2& other) const
    {
        return Vector2(x * other.x, y * other.y);
    }

    Vector2 operator*(const Vector2& other)
    {
        return Multiply(other);
    }

};

ostream& operator << (ostream& stream, const Vector2& other)
{
    stream << other.x << ", " << other.y;
    return stream;
}

int  main() {
    Vector2 position(4.0f, 5.0f);
    Vector2 speed(0.5f, 1.5f);
    Vector2 powerup(1.1f, 1.1f);

    Vector2 result1 = position.Add(speed.Multiply(powerup));
    Vector2 result2 = position + speed * powerup;

    cout << result2 << endl;

}
```

2.结果输出

今天主要看了The Cherno的C++教程，很多概念之前看书已经有所了解，就当复习一遍。