

40 Implicit Conversion and the Explicit Keyword in C++

1. 隐式构造函数/隐式转换

implicit(隐式)的意思是不会明确地告诉你它要做什么，它有点像在某种情况下自动的工作。实际上C++允许编译器对代码进行一次隐式的转换。

如果我开始使用一种数据类型作为另一种类型来使用，在这两周类型之间就会有类型转换，C++允许隐式地转换，不需要用`cast`等做强制转换。

```
C++

class Entity
{
private:
    std::string m_Name;
    int m_Age;
public:
    Entity(const std::string& name)
        : m_Name(name), m_Age(-1){}
    Entity(int age)
        : m_Name("Unknown"), m_Age(age) {}

    const int& GetName() const { return m_Age; }
};

int main()
{
    Entity a("Cherno");
    // Entity a = "Cherno"; 不建议，上面的写法更清楚
    Entity b = Entity(22);
    // Entity b = 22;
    std::cout << a.GetName() << std::endl;
    std::cin.get();
}
```

这里为什么能把一个整数赋值给一个类，且这个类还有一个字符串成员name？

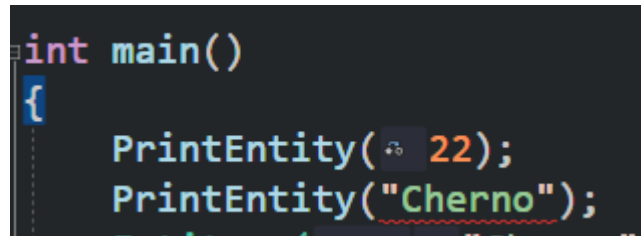
这就是隐式转换或者隐式构造函数。

它隐式地把22转换为一个Entity对象，构造出了一个Entity，因为Entity有一个构造函数，接受一个整形参数age，另一个接受一个字符串参数。

C++

```
void PrintEntity(const Entity& entity)
{
    // Printing
}

int main()
{
    PrintEntity(22);
    PrintEntity("Cherno");
}
```



```
int main()
{
    PrintEntity(22);
    PrintEntity("Cherno");
}
```

这里的参数是一个 `char array[7]`，而不是 `std::string`，见 [32 How Strings Work in C++ \(and how to use them\)](#)。为了让这里能工作，C++ 需要做两次转换，一次是用 `const char` 数组到 `string`，一次是从 `string` 到 `Entity`，但是只允许做一次隐式转换。

C++

```
PrintEntity(std::string("Cherno"));
PrintEntity(Entity("Cherno"));
```

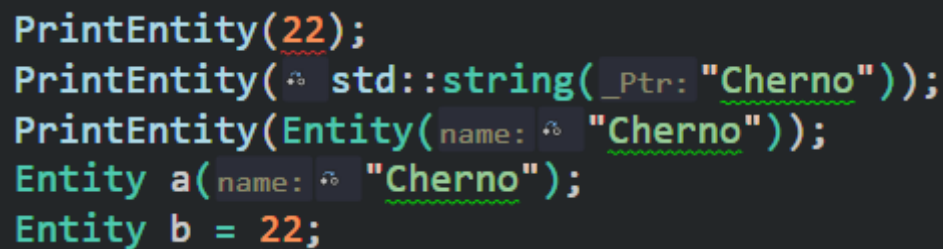
可以帮助你简化代码，不过就个人来说会尽量避免使用它，除了某些情况要用来简化代码，而不是通过构造函数来包裹。

2. explicit关键字

`explicit` 会禁用隐式转换，放在构造函数前面，意味着这个构造函数不会进行隐式转换。如果逆向用一个整数构造一个 `Entity` 对象，那你必须显式地调用这个构造函数。

C++

```
explicit Entity(int age)
: m_Name("Unknown"), m_Age(age) {}
```



```
PrintEntity(22);
PrintEntity(std::string(_Ptr: "Cherno"));
PrintEntity(Entity(name: "Cherno"));
Entity a(name: "Cherno");
Entity b = 22;
```

C++

```
Entity b(22);
Entity b = (Entity)22;
Entity b = Entity(22);
```

有时会在数学运算库的地方用到 `explicit`，因为不想把数字和向量进行比较，保证代码安全。不过不经常使用。低级封装可能会用到，防止偶然转换和性能问题或者bug。