20220427-书&C++

1.过程描述

1.1类中内联

1.2虚函数与内联

2.结果输出

1.过程描述

1.1类中内联

inline是一种用于实现的关键字，而不是用于声明的关键字。在定义处加inline关键字。编译器对inline函数的处理步骤：

- 将inline函数体复制到inline函数调用处
- 为所用inline函数中的局部变量分配内存空间
- 将inline函数的输入参数和返回值映射到调用方法的局部变量空间中
- 如果inline函数有多个返回点，将其转变为inline函数代码块末尾的分支（使用GOTO）

内联能提高函数效率，但并不是所有函数都适合定义为内联函数。内联是以代码膨胀为代价，仅仅省去了函数调用的开销，从而提高函数的执行效率

- 如果执行函数体内代码的时间相比于函数调用的开销较大，那效率的收获更少（如出现循环）
- 每一处内联函数的调用都要复制代码，将使程序的总代码量增大，消耗更多的内存空间

C++ 复制代码

```
头文件：
class A
{
    public:
        void f1(int x);
        void Foo(int x,int y)
        {};
}

实现文件
int Foo(int x,int y);
inline int Foo(int x,inty)
{
    return x+y
}
inline void A::f1(int x)
{

}
int main()
{
    cout<<Foo(1,2)<<endl;
}
```

1.2虚函数与内联

虚函数可以是内联函数，内联是可以修饰虚函数的，但是当虚函数表现多态性的时候不能内联。内联是在编译期建议编译器内联，而虚函数的多态性在运行期，编译器无法知道运行期调用哪个代码，因此虚函数表现为多态性时（运行期）不可以内联。

inline virtual唯一可以内联的时候是：编译器知道所调用的对象是哪个类，这只有在编译器具有实际对象而不是对象的指针或引用时才会发生

C++ | 复制代码

```cpp
#include <iostream>
using namespace std;
class Base
{
public:
    inline virtual void who()
    {
        cout << "I am Base.\n";
    }
    virtual ~Base(){}
};

class Derived :public Base
{
public:
    inline void who()
    {
        cout << "I am Derived.\n";
    }
};

int main()
{
    //此处的虚函数who是通过Base的具体对象b来调用的，编译期就能确定了，所以它可以是内联的，但最终是否内联取决于编译器
    Base b;
    b.who();
    //此处的虚函数是通过指针调用的，呈现多态性，需要在运行期间才能确定，所以不能为内联
    Base* ptr = new Derived();
    ptr->who();
    //因为Base有析构虚函数，所以delete时，会先调用派生类析构函数，再调用基类析构函数，防止内存泄露
    delete ptr;
    ptr = nullptr;

    system("pause");
    return 0;
}
```

2.结果输出

今天白天基本啥正事没干（只在下午copy了会spdlog的代码），到晚上才稍微看了会书跟温习了一点C++知识。不行的呀，明天无论如何开始看数据结构与算法的视频。