用多态的思想实现计算器

开发思想：开闭原则，对拓展开放，对修改关闭。

多态优点：利于后期拓展，结构性好，可读性强。

多态缺点：效率偏低，结构中增加了指针，复杂了一点。

如果是用之前的定义一个类，在类中定义各种计算函数，那么在需要拓展时，就得修改类中的某些函数，以达到拓展的目的。但是，开发中忌讳修改之前的代码，以免造成其他错误的产生。因此，要用多态的思想，拓展功能，避免修改之前的代码。

下面用一个计算器案例来说明多态思想的优势：

首先，我们定义了父类 abstractCalculator，在其中定义了虚函数getResult()。

之后，定义了子类加法计算器类 PlusCalculator，继承父类 abstractCalculator。同时，其中还定义了 函数 getResult()，计算两个数之和。

之后，功能扩展，要添加减法计算器时，直接增加一个减法计算器类，同样继承父类abstractCalculator，定义函数 getResult()，计算两数之差。

当我们调用子类的getResult时，它就通过多态指向了子类的getResult()，并实现相应的功能。

不需要修改之前的代码，直接添加一个子类，继承了父类的成员和函数，并通过多态将父类函数getReult()指向了子类自己的getResult()。

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

using namespace std;

class abstractCalculator

{

public:

virtual int getResult()

{

return 0;

}

void setv1(int v)

{

this->val1 = v;

}

void setv2(int v)

{

this->val2 = v;

}

public:

int val1;

int val2;

};

//加法计算器

class PlusCalculator : public abstractCalculator

{

public:

virtual int getResult()

{

return val1 + val2;

}

};

//减法计算器

class SubCalculator : public abstractCalculator

{

public:

virtual int getResult()

{

return val1 - val2;

}

};

void test()

{

abstractCalculator\* abc;

abc = new PlusCalculator;

abc->setv1(10);

abc->setv2(20);

cout << abc->getResult() << endl;

}

int main()

{

test();

return 0;

}