```
//构造函数，友元函数，成员函数的简单实现
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
#define MAXV 100

class Box
{
public:
	static int county;
	double len;
	double high;
	Box(double len, double high) :len(len), high(high) { county++; }  //构造函数函数体，要加{}函数体
	friend void write_width(Box& a, double width);		//成员函数不能由对象引用，且成员函数要调用对象应该用引用类型，不然是按值传递，无法改变对象的值
	void print_box()
	{
		cout << "len:" << this->len << "  high:" << this->high << "  width:" << this->width << endl;
	}
	void write_password(int password)
	{
		this->password = password;
	}
private:
	double width;
	int password;

};
int Box::county = 0;
class Bigbox :public Box
{
public:
	Bigbox(double l, double high, double width, int password) :Box(l, high), password(password) {};
private:
	int password;
};
void write_width(Box& a, double width)
{
	a.width = width;
}
int main()
{
	Box b(10.0, 5.0);
	write_width(b, 5.0);
	Bigbox a(1, 2, 3, 4);
	write_width(a, 5.12);
	a.print_box();
	cout << "TOTLE BOX NUM" << Box::county << endl;

	return 0;
}
```

```
//构造函数，友元函数，成员函数的简单实现
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
#define MAXV 100
class Box
{
public:
	static int county;
	double len;
	double high;
	Box(double len, double high) :len(len), high(high) { county++; }  //构造函数函数体，要加{}函数体
	friend void write_width(Box& a, double width);		//成员函数为void没有传回参数，因此不会改变对象的值，如果成员函数换做引用类型的话，则可以改变对象的值
	void print_box()
	{
		cout << "len:" << this->len << "  high:" << this->high << "  width:" << this->width << endl;
	}
	void write_password(int password)
	{
		this->password = password;
	}
private:
	double width;
	int password;

};
void write_width(Box& a, double width)
{
	a.width = width;
}


int Box::county = 0;


class Bigbox :public Box
{
public:
	Bigbox(double l, double high, double width, int password) :Box(l, high), password(password) {};
private:
	int password;
};

int main()
{
	Box b(10.0, 5.0);
	write_width(b, 5.0);
	Bigbox a(1, 2, 3, 4);
	write_width(a, 5.12);
	a.print_box();
	cout << "TOTLE BOX NUM" << Box::county << endl;

	return 0;
}
```

//特别注意,

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring> // 包含strcpy所需头文件
using namespace std;
#define MAXV 100
int main()
{
	char aaa[] = "good";
	const char* bbb = "hollo";

	char ddd[MAXV];
	strcpy(ddd, bbb);

	//char* ccc;  //错误的，char*必须被定义成const char *,即指向的数据不可被修改
	//strcpy(ccc, bbb);  //这里就报错，说指向的数据不能被修改
	cout << ccc << endl;

	return 0;
}
```

//后续

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring> // 包含strcpy所需头文件
using namespace std;
#define MAXV 100
int main()
{
	char aaa[] = "good";
	const char* bbb = "hollo";

	char ddd[MAXV];
	strcpy(ddd, bbb);		//数组指针类型通过strcpy修改
	bbb = "okay";			//const char* 类型得直接修改
	cout << bbb << endl;



	return 0;
}
```

//字符串字面量(常量指针)

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring> // 包含strcpy所需头文件
using namespace std;
#define MAXV 100
int main()
{
	char* aaa = (char*)"happy";			//双引号内的字符串为字符串字面量，是不可以被修改的，只能给其他字符串赋值用（强转为 char*后可以赋值给char*类型，但是实际上还是const char *类型，无法被赋值）   strcpy(aaa,"111")报错。
	char bbb[] = "good";
	char ccc[10];
	strcpy(bbb, aaa);

	//strcpy(aaa,"111");		//报错，因为此时aaa是一个指向const char * 类型的指针，所指为常量，不能被修改。
	aaa = (char*)"111";		//char * 类型可以直接赋值
	//bbb = (char*)"111";	//错误，数组名是指针常量，即char* const ptr ,不能对其赋值->赋值 意味着指向的改变

	strcpy(ccc, "111");



	strcpy(bbb, "111");
	cout << "ccc=" << ccc << endl;
	cout << "bbb=" << bbb << endl;

	return 0;
}
```

//数组地址是一常量，无法被修改

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring> // 包含strcpy所需头文件
using namespace std;
#define MAXV 100
int main()
{
	int a[10];
	int* pa = a;
	for (int i = 0; i < 10; i++)
	{
		*(pa+i) = i;			//利用一连串的地址来接受数组的数据


	for (int i = 0; i < 10; i++)
	{
		cout << *(pa + i) << endl;
	}
	int num = sizeof(a) / sizeof(int);
	cout << "num:" << num << endl; //输出数组内元素个数
	//*(a++) = 100;  //有误:数组地址是一个常量，是无法被修改的    // 但是可以写成*(a+1)=100;
	return 0;
}
```

//指针数组->用于存字符串数组(存每个字符串的首地址)

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring> // 包含strcpy所需头文件
using namespace std;
#define MAXV 100
int main()
{
	char* a[10];
	for (int i = 0; i < 3; i++)
	{
		a[i] = new char[MAXV];  //因为只存字符串的首地址，因此每个字符串得自己new出来，并将首地址赋给a[i]让其存储下来。
		cin.getline(a[i], 5);		//注意:getline函数第一个参数是首地址，第二个参数是字符串长度

	}
	cout << endl;
	for (int i = 0; i < 3; i++)
	{
		cout << (a[i]) << endl;;
	}
	//注意:a[i]的值为第i个字符串的首地址，*a[i]仍然是地址，为第i个字符串的第一个元素即a[i][0]的地址!**a[i]是a[i][0]的值！！！
	return 0;
}
```

重点，一定要再写一遍，老是犯各种错

//含指针成员的构造函数和复制构造函数（含有指针成员不能用默认构造函数，需要手动为字符数组分配新的地址空间，并进行赋值。如果直接a.name = "hallo"也是错的(没分配内存)）

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring> // 包含strcpy所需头文件
using namespace std;
#define MAXV 100
class Stu
{
public:
	char *name;
	int id;
	Stu(){};  //重载的不含参数的构造函数:注意写法:类名(){};中间没有冒号分隔！！！
	Stu(const char* name, int id) :id(id)
	{		this->name = new char[50];
		strcpy(this->name, name);

	}
	Stu(Stu &a) {
	id =  a.id;
	name = new char[strlen(a.name) + 1];
	strcpy(name, a.name);
	};

};

int main()
{
	Stu a("hallo",15);
	Stu b(a);
	Stu c;
	//c.name = "haha"; 错误！ name还未被分配内存，无法对其赋值!

	cout << "ID:" << b.id << "\tName:" << b.name << endl;

	return 0;
}
```

//双目运算符重载！！ 虚数加减法

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring> // 包含strcpy所需头文件
using namespace std;
#define MAXV 100
class Com
{
public:
	double i;
	double r;
	Com(double r, double i) :r(r), i(i) {};
	Com() { r = 0; i = 0; };//重构为不需要参数的构造函数
	Com operator +(Com& a)
	{
		return Com(r + a.r, i + a.i);
	}
	Com operator -(Com& a)
	{
		return Com(r - a.r, i - a.i);
	}
	Com operator +=(Com &a)    //特别注意
	{
		return Com(r + a.r, i + a.i);
	}
	void show()
	{
		cout << r << "+" << i << endl;
	}
};
int main()
{
	Com a(1, 1);
	Com b(5, 6);
	Com c = a + b;
	c.show();
	Com d;
	d = d += a;
	//因为重载双目运算符默认参数在操作符的两旁？？ d = d+=a;第
	一个d接收传回来的参数，第二个d为双目运算符的操作数。
	d.show();
	return 0;
}
```

//其他类的对象做本类的成员书上P253

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring> // 包含strcpy所需头文件
using namespace std;
#define MAXV 100
class Point
{
public:
	Point(int x = 0, int y = 0):x(x),y(y)
	{
		cout << "Point构造函数:(" << x << "," << y << ")"<<endl;
	}~Point() { cout << "Point析构函数:(" << x << "," << y << ")" << endl; }
	int getx() { return x; }
	int gety(){ return y; }
protected:
	int x, y;
};
class Rec
{
public:				//注意:这里构造函数调取Point类的构造函数进行初始化
	Rec(int x1, int y1, int x2, int y2) :p1(x1, y1), p2(x2, y2) { cout << "Rec 构造P1:(" << p1.getx() << ", " << p1.gety() << ")\n" << "\tP2:(" << p2.getx() << ", " << p2.gety() << ")" << endl; }
private:
	Point p1, p2;
	int color;
};
int main()
{
	Rec A(1, 2, 3, 4);   //初始化Rec的类A

	return 0;
```

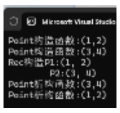
//对象指针做成员

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring> // 包含strcpy所需头文件
using namespace std;
#define MAXV 100

class Point {
public:
  Point(int x = 0, int y = 0) : x(x), y(y) {
    cout << "Point构造函数: (" << x << ", " << y << ")" << endl;
  }
  ~Point() {
    cout << "Point析构函数: (" << x << ", " << y << ")" << endl;
  }
  void display() const {
    cout << "Point: (" << x << ", " << y << ")" << endl;
  }
private:
  int x, y;
};

class Rec {
public:
  Rec(int x1, int y1, int x2, int y2) {
    // 动态分配 Point 对象，并让 p1 和 p2 指针指向它们
    p1 = new Point(x1, y1);
    p2 = new Point(x2, y2);
    cout << "Rec构造函数" << endl;
  }
  ~Rec() {			//一、派生类无法继承基类的析构函数，所以需要自己来写
private:
        Point p1, p2;
        int color;
};
int main()
{
        Rec A(1, 2, 3, 4);    //初始化Rec的类A

        return 0;
}
```



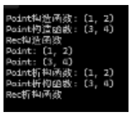
```
Point构造函数:(1,2)
Point构造函数:(3,4)
Rec构造P1:(1, 2)
        P2:(3, 4)
Point析构函数:(3,4)
Point析构函数:(1,2)
```

注意执行的次序！！！

```
public:
   Rec(int x1, int y1, int x2, int y2) {
      // 动态分配 Point 对象，并让 p1 和 p2 指针指向它们
      p1 = new Point(x1, y1);
      p2 = new Point(x2, y2);
      cout << "Rec构造函数" << endl;
   }
   ~Rec() {                    //一、派生类无法继承基类的析构函数，所以需要自己来写
      // 释放动态分配的内存              //！二、派生类的析构函数只负责派生类成员的释放！
      delete p1;
      delete p2;
      cout << "Rec析构函数" << endl;
   }
   void display() const {
      p1->display();
      p2->display();
   }
private:
   Point* p1; // 指向 Point 对象的指针
   Point* p2; // 指向 Point 对象的指针
};

int main() {
   Rec A(1, 2, 3, 4);
   A.display();

   return 0;
}
```



```
Point构造函数: (1, 2)
Point构造函数: (3, 4)
Rec构造函数
Point: (1, 2)
Point: (3, 4)
Point析构函数: (1, 2)
Point析构函数: (3, 4)
Rec析构函数
```

//重构等号使之可以充当带有指针成员的复制构造函数书上P273

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring> // 包含strcpy所需头文件
using namespace std;
#define MAXV 100

class stu
{
public:
        stu() {} //无参数构造函数

        stu(int id, const char* name, double score) :id(id), score(score)   //重载为需要穿参数的构造函数
        {
                this->name = new char[10];        //特别注意！这里是给this->name分配内存，而不是给name分配内存(我之前写成name = new char[10]就一直报错！！)
                strcpy(this->name, name);
        }


        stu(stu& a) :id(a.id), score(a.score)    //复制构造函数
        {
                this->name = new char[10];
                strcpy(this->name, a.name);
        }
        stu& operator =(stu&);
        ~stu()
        {
                delete name;
        }

        int id;
        char* name;
        double score;
        void display()
        {
                cout << "ID:" << this->id << "  name:" << this->name << "  score:" << this->score << endl;
        }
};
stu& stu::operator=(stu& a)     //重载运算符=，使之可以充当指针的对象的复制构造函数！//注意，返回值应为引用类型!!
{
        this->id = a.id;
        this->score = a.score;
        this->name = new char[10];
        strcpy(this->name, a.name);
        return *this;
}
int main()
{
        stu A(1, "haha", 3.5);
        A.display();
        stu B(A);      //利用复制构造函数
        B.display();
        stu C;
        C = A;                    //利用重构后的等号充当构造函数
        C.display();



        return 0;
}
```

模板：加强版的重载，允许使用不同类型的参数在同一个函数下操作：比如栈如果不用模板的话，那么char型，int型，float型都需要单独定义各自的栈，造成大量冗余。

//函数模板，书P282

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring> // 包含strcpy所需头文件
using namespace std;
#define MAXV 100

template<class T>          //需要在template中声明含有一种类型的参数：用T指代
T max_value(T x,T y,T z)          //其余都是一样的！
{
        T temp = x > y ? x : y;
        return temp > z ? temp : z;

}
int main()
{
        cout << max_value(12, 9, 10) << endl;
        cout << max_value('a', 'A', '9')<<endl;

        return 0;
}
```

//函数模板，传入两种不同参数的类型！！书P282

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring> // 包含strcpy所需头文件
using namespace std;
#define MAXV 100

template<class T, class P>    //需要在template中声明含有两种类型的参数：用T和P指代
T adds(T x,P y)   //这里的返回值为T：是第一个参数的类型
{
        return x + y;
}
int main()
{
        cout << adds(12, 9.5) << endl;    //返回值为第一个参数的类型（整形），返回值为21
        cout << adds(9.5, 12) << endl;    //返回值为第二个参数的类型（浮点），返回值为21.5

        return 0;
}
```

//类模板，传入两种不同参数的类型！！书P286

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring> // 包含strcpy所需头文件
using namespace std;
#define MAXV 100

template<class T1, class T2>      //类模板需要在template中声明含有两种类型的参数：用T1和T2指代
class Myclass
{
public:
        Myclass(T1 x, T2 y);
        void display()
        {
                cout << "x:" << x << "  y:" << y << endl;
        }
private:
        T1 x;
        T2 y;
};
template<class T1, class T2>      //类模板的构造函数，也需要在template中声明含有两种类型的参数：用T1和T2指代
Myclass<T1, T2>::Myclass(T1 x, T2 y) :x(x), y(y) { }            //注意成员函数类外实现时的写法
int main()
{
        Myclass<int, float> A(6, 5.4);                  //特别注意：类模板在实例化时要说明两个类型具体是哪两个<int,float>
        A.display();
        Myclass <char,const char*> B('x', "hello");         //这里只是浅拷贝，更改了指针的指向，但是并未给字符数组分配新的内存！！
        B.display();
        return 0;
}
```

有异常：try语句块内为可能出现异常的语句-》出现异常时throw将异常抛出-》catch捕获异常并进行操作！

//异常处理:检测是否有0作为除数的异常！！

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring> // 包含strcpy所需头文件
using namespace std;
#define MAXV 100

double divi(double a, double b)
{
        if (b == 0)throw"输入错误!除数不可以是零！"; //对于可能出现异常的语句，用throw将异常提交给catch,这里提交const char *给catch接收
```

//异常处理:用c++自带的异常类exception的派生类传递异常 该颜色部分为较前一段代码的改动部分！！

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
#include<exception>  //注意！！要include<exception>!
using namespace std;
#define MAXV 100

double divi(double a, double b)
{
        if (b == 0)throw runtime_error("输入错误!除数不可以是零！");  //exception的派生类
        return a / b;
}
int main()
{
        double x, y, z;
        while (cin>>x>>y)    //try语句块后面要紧接着写处理改异常的const语句块，两者之间不能再有其他语句！一个try可以有多个catch语句！但不能几个try后面接一个const语句
```

```
[illegible]
#include <iostream>
#include <cstring> //包含strcpy所需头文件
using namespace std;
#define MAXV 100

double divi(double a, double b)
{
	if (b == 0)throw"输入错误!除数不可以是零！ ";  //对于可能出现异常的语句，用throw将异常提交给catch,这里提交const char *给catch接收
	return a / b;
}
int main()
{
	double x, y, z;
	while (cin>>x>>y)     //try语句块后面要紧接着写处理改异常的const语句块，两者之间不能再有其他语句！一个try可以有多个catch语句！但不能几个try后面接一个const语句
	{
		try{z = divi(x, y);}  //try内写可能出现异常的语句块

		catch (const char* s)
		{
			cout << s << endl;
			cout << "输入一对新实数:";
			continue;
		}

		cout << "x/y=" << z << endl;


	}
	return 0;
}
```

```
double divi(double a, double b)
{
	if (b == 0)throw runtime_error("输入错误!除数不可以是零！ ");  //exception的派生类
	return a / b;
}
int main()
{
	double x, y, z;
	while (cin>>x>>y)     //try语句块后面要紧接着写处理改异常的const语句块，两者之间不能再有其他语句！一个try可以有多个catch语句！但不能几个try后面接一个const语句
	{
		try{z = divi(x, y);}   //try内写可能出现异常的语句块

		catch (runtime_error err)        //在 catch 块中使用 runtime_error err 会创建一个新的 runtime_error 对象 err，
		{                                //它是捕获的异常对象的副本。这意味着异常信息将被复制到 err，然后调用 err.what() 时，您将得到异常的详细信息。
			cout << err.what() << endl;
			cout << "输入一对新实数:";
			continue;
		}

		cout << "x/y=" << z << endl;


	}
	return 0;
}
```

is_member_function_pointer
runtime_error
_VCRUNTIME_DISABLED_WARNINGS
_VCRUNTIME_DISABLED_WARNING_4339
_VCRUNTIME_DISABLED_WARNING_4412
_VCRUNTIME_EXTRA_DISABLED_WARNINGS
_VCRUNTIME_H
_Xruntime_error
__EDG_RUNTIME_USES_NAMESPACES
class std::runtime_error
base of all runtime error exceptions

//二维数组值得注意的点！！！

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

void display(int a[][3], int i, int j)    //二维数组作为形参传入，第二维需要指名其大小
{
	printf("%d", a[i][j]);
}
int main()
{
	int a[2][3] = { {1,2,3},{4,5,6 } };
	display (a,1,2);                //实参这里只用传入数组地址即可
		return 0;
}
```

//通过友元函数来修改对象的值牛头人

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>
using namespace std;
class stu;   //声明类stu
class tech
{
public:
	void setgrade(stu& a,int b);
};
class stu
{
public:
	friend void tech::setgrade(stu& a,int b);
	stu(int grade) :grade(grade) {};
	void display() { cout << grade << endl; }
private:
	int grade;
};
void tech::setgrade(stu& a,int b)
{
	a.grade = b;
}
int main()
{
	stu a(50);
	tech haha;
	haha.setgrade(a,100);
	a.display();
	return 0;
}
```