1. 变量是某段内存人为起的名字，赋值的过程是将某个数据储存进内存，如

int a=10;申请某段整型内存，命名为a，将数据10存入内存名字为a的内存中

1. char a=99;

cout<<a<<endl;

显示的结果是c，因为c对应的ASCII编码是99.



使用C++风格的字符串时，一定要设置头文件



int h=10;

int i=10;

cout<<(h||i)<<endl;

//输出值为0或1，用于判断真假



if()括号中比较的两个数可以互换位置，但不能写一串字符，如if(5<a<10)

else if() 不能写在第一个。else可写在最后，后面不接()。但三者后面都要接{}

if与else if 的相同点：都能选择不同的内容。

if与else if 的不同点: 但是如果某个数满足if(a>10) 与if(a>5),则会将两个if都执行，而if与else if并列，则只会if与else if中的其中一个(谁先满足条件就先执行谁)



while() 括号里输入非零数字可使得while内部的语句无线循环



switch语句的逻辑: 计算表达式的值。 并逐个与其后的常量表达式值相比较，当表达式的值与某个常量表达式的值相等时， 即执行其后的语句，然后不再进行判断，继续执行后面所有case后的语句。如表达式的值与所有case后的常量表达式均不相同时，则执行default后的语句。

1. break跳出最近的一层***循环***。

如：while(1)

{

if(sex==1||sex==2)

{

cout<<”输入正确”<<endl;l

break;

}

}

其中，break跳出的是while循环而不是if语句。同理，***continue***也是一个原理。



do…while中，while()后面要写; 总之，如果后面没有{} 就需要;



continue作用是结束该层次的本次循环，继续该层次的下一次循环



数组arr[] 括号中的数表示元素个数，下标是从0开始计数。{}中的元素使用 , 隔开



cout<<arr; 输出的是数组arr[]={…} 所在的首地址（注意：数组也是一段内存的名字）

cout<<&arr[1]; 输出的是数组第二个元素的首地址

cout<<arr[1]; 输出的是数组中第二个元素的值

cout<<sizeof(arr); 输出的是整个数组arr对应的内存

cout<<sizeof(arr[1]) 输出的是数组中第二个元素对应的内存

在变量名（如&a）或数组名（如arr）前输入(int)可以将地址转换为十进制



数组名不可以修改，数组名是地址。如arr=100;是错误操作。

但是数组中的元素可以修改，如arr[0]=100;



if while 等内部创建的变量是不能被外部识别的，而外部创建的变量可以被内部识别。如：

int a=0;

if(a<10)

{

int b=a; a++;

}

cout<<b; 此处b不能被识别

cout<<a; 此处a可以被识别



int arr[行][列];

cout<<"二维数组占用的列数为："<<sizeof(arr[0])/sizeof(arr[0][0])<<endl;



指针的常见写法：int \*p=&a; int \*p; p=&a; \*p=a; sizeof(int \*) sizeof(p)=sizeof(\*p) int \*p=NULL; const int \*p;//解引用的内容不能变 int \* const p;//指针不能变



int \*p1=arr; //arr前不需要写\*,arr后也不需要写[]

int \*p2=&a; //a前必须要写&



int a=10,b=20;

const int\* p=&a;

p=&b//正确操作

\*p=15;//错误操作



const在结构体中主要运用于修饰地址传递中的参数。防止内容被修改。



使用引用是必须初始化引用，引用在设定后不能修改。错误操作：int& a=10;正确操作int n=10; int& a=n;

不要返回局部变量的引用和地址。

关于函数的返回：

返回变量：

int ret(int a)

{

int a=10;

return a;

}

main函数中：int Ret=ret(b);

返回指针：

int\* ret(int\* a)

{

int a=10; int \*p=&a;

return p;//或return &a;

}

main函数中：int\* p=ret(&A);

返回引用：

int& ret(int a)

{

int a=10;

return a;

}

main函数中：int Ret=ret(b);

传递引用：传递的始终是变量a，而不是引用&a。接收方始终要带上&。

void ret(int& a)

{

a=10;

}

main函数中：ret(b);

注：以上A和b都是整型变量。

**数组只能以指针的形式通过形参进行传递！**

传递数组的正确姿势：

main函数中： int arr[10]; func(arr); arr默认为数组首地址,只能这样传递。

接受函数：void func(int arr)

1. int \*p=arr; //注：arr是数组，但该表示方法正确。

变量前的&有两个意义：一是取地址，二是引用标志。



不要返回局部变量的引用。

函数的默认参数使用注意事项：参数列表中某个参数设定了值，那么对以后的参数必须设定值。函数定义和函数声明只能有一个有默认参数。默认参数与形参发生冲突时，以形参为准。



函数的占位参数也可以设定默认值。



函数重载的核心就是函数名相同，但不会产生歧义。



const可以作为函数重载的条件。（const修饰形参列表中的参数且参数必须是引用）

const int& a=10;合法操作int& a=10;非法操作



函数返回值不可以作为函数重载的条件。如：

int func(){return 0}

void funct(){}

两者冲突！

1. 类对象实例化的同时会有构造函数的调用。

值传递、将值返回到局部对象的时候、值拷贝的同时会有拷贝构造函数的调用。

给变量赋值的同时会有有参构造函数的调用。



构造函数按有无参数分为无参构造函数（默认构造函数）和有参构造函数。

按照类型分为普通构造函数和拷贝构造函数。



创建一个类，C++程序都会自动创建三个函数：默认构造函数、析构函数、默认拷贝构造函数。当人为添加有参构造函数，计算机会自动添加拷贝构造函数，省略默认构造函数。以此类推。



如果写了有参构造，编译器就不会提供默认构造。如果写了拷贝构造函数，编译器就不会提供构造函数。



构造函数的作用是为了给成员属性初始化（及给成员属性赋值）。



类对象：class person{string name}; person p;

p是person类对象。name是person的类成员



当自定义拷贝构造函数时，编译器就不会自动进行拷贝。要按照人为构造的拷贝构造函数中的内容执行操作。

1. 指针通过->访问 。普通变量和引用通过 . 访问
2. 当数据开辟到堆区时，运行后记得手动释放数据。这时如果出现拷贝的过程，编译器会自动进行浅拷贝，导致堆区数据重复释放。需在拷贝构造函数中进行深拷贝。
3. 数据创建于释放有“先入后出”的原理。
4. delete 的是指针。delete之后要将删除的指针指向空。
5. 如果Student是一个类，student是这个类的对象，m\_A是Student类中的int类型。那么student.m\_A是int类型。
6. 非静态成员变量不能在类外初始化，也不能通过类名访问。
7. 静态成员变量也是有访问权限的。
8. 静态成员函数只能访问静态成员变量，不能访问非静态成员变量。非静态成员函数既可以访问静态成员变量，也可以访问非静态成员变量。(清楚逻辑)
9. 静态成员函数可以通过类名和类对象进行访问。而非静态成员函数只能通过类对象进行访问，不能通过类名进行访问（清楚逻辑）。
10. 一个空类占1字节。一个空类中有一个非静态成员属性占4字节，两个占8字，以此类推。
11. this指针作为返回值时必须要返回引用（我也不彻底理解原理），为了对某个数据（案例中是Person的类对象p）持续操作，需要不断通过引用传入返回p并且返回p）。
12. 空指针访问类中的成员函数时只能访问不含类属性的成员函数，不能访问不含类属性的成员函数。
13. const修饰成员函数时，不能在成员函数中对成员属性进行修改，但是当成员属性前有mutable时，常函数也可以对它进行修改值。
14. 常对象只能访问常函数，不能访问非常函数，不能访问成员属性（但是可以访问静态成员属性）。普通对象既能访问常函数，也能访问非常函数，还能访问成员属性。（理解原理）但是成员属性前加mutable时常对象也可以访问。（与常函数的使用作类比）
15. 静态成员函数不能使用const修饰。因为const修饰的成员函数实际上是带有this指针的类成员函数。而静态成员函数作用于全局。
16. const int \*p; 与 int const \*p;都是常量指针。int \* const p;是指针常量。
17. const int a 和int const a都是一样的。
18. 重载加号时传出参数要用值，重载左移运算符时传入、传出参数必须用引用！（我也不知道原因）
19. 前置返回引用，后置返回值！
20. 重载赋值运算符时返回值必须是引用。（我也不知道原因）
21. 在重载关系运算符的函数中，必须判断this是否指向NULL，如果不，则需要先释放this指针指向的内容。（我也不知道原因）
22. bool 运算符：

bool operator==(Person &p)

{

if(m\_A==p.m\_A){return turn; }

else(return false)

}

if(p1==p2)

{

cout<<”相等”<<endl;

}

else

{

cout<<”不相等”<<endl;

}

1. 父类中的所有非静态成员属性都会被继承下去，私有权限的属性也会被继承，但是会被隐藏导致无法访问。
2. 初始非静态类属性时只能通过内部的构造函数实现，不能通过外部赋值实现。
3. 通过子类对象访问父类中的同名成员，需要加作用域，如：s.Base::m\_A;
4. 子类出现和父类同名静态成员函数，也会隐藏父类中所有同名成员函数（所有的与之发生重载的函数）。如果想访问父类中被隐藏的成员，需要加作用域。
5. 多态的条件：继承关系、子类重写父类的内容（多写和漏写都不行）。

多态的使用：父类指针指向子类对象。

1. 多态使用案例：将父类的“主内容”作为端口。将子类重写的“附内容”作为目标。通过父类指针指向不同的子类对象为不同的“钥匙”，通过父类的端口，访问子类的内容。
2. 有了纯虚析构或纯虚函数，这个类都是抽象类。
3. 虚析构和虚函数都是用在父类。