特殊性质

内存控制

1. 实际的delete操作：第一步，对delete的对象或delete的数组中的元素执行对应的析构函数；第二步，编译器调用名为operator delete或者operator delete[]的标准库函数释放内存空间。
2. 类内重载new和delete运算符时会隐式地声明为静态的，它们不能操纵类的任何数据成员
3. malloc函数接受一个表示待分配字节数的size_t，返回指向分配空间的指针或返回0表示分配失败。free函数接受一个void*，将相关内存返回给系统。调用free(0)没有任何意义。（两者存在与cstdlib头文件中）

```
//简单的new版本
void* operator new(size_t size){
    if(void* mem=malloc(size)){
        return mem;
    }
    else{
        throw bad_alloc();
    }
}
//简单的delete版本
void operator delete(void* mem) noexcept {
    free(mem);
}
```

运行时类型识别

1. 运行时类型识别RTTI的功能由两个运算符实现：
   - typeid运算符：用于返回表达式的类型
   - dynamic_cast运算符，用于将基类的指针或引用安全地转换成派生类的指针或引用

   当上面两个运算符用于某种类型的指针或引用且该类型含有虚函数时，运算符将使用指针或引用所绑定对象的动态类型。
2. dynamic_cats运算符的使用形式：

```
dynamic_cast<type*>(e)
dynamic_cast<type&>(e)
dynamic_cast<type&&>(e) //该形式下e不能是左值
```

其中type必须是一个**类类型**且通常情况下应该含有虚函数。上面的e的类型必须是type的公有派生类、或者是type的公有基类、或者是type的类型。若类型转换失败且转换目标是指针类型，则结果

为0；若是引用类型则抛出bad_cast异常。

3. 假设Base类至少含有一个虚函数且Derived是Base的公有派生类。若有一个指向Base的指针bp，则可以在运行时将其转换为指向Derived的指针：

```
if(Derived *dp=dynamic_cast<Derived*>(bp)){
    //使用dp执行Derived对象
}
else {  //bp指向一个Base对象
    //使用bp指向的Base对象
}
```

若bp指向的是Derived对象则上述的类型转换初始化dp并令其指向bp所指的Derived对象。此时if语句内部使用Derived操作的代码是安全的。否则类型转换结果为0。

4. 引用类型的dynamic_cast：因为不存在空引用，所以当对引用的类型转换失败时，程序抛出一个名为bad_cast的异常。

```
try{
    const Derived &d=dynamic_cast<const Derived&>(bp);
} catch (bad_cast) {
    //bad_cast定义在typeinfo中
}
```

5. 当运算对象不属于类类型或者是一个不包含任何虚函数的类时，typeid运算符指示的是运算对象的静态类型。而当运算对象是定义了至少一个虚函数的类的左值时，typeid的结果直到运行时才会求得。**typeid应该作用于对象，当其作用于指针时返回的结果是该指针的静态编译类型**

枚举类型

1. 枚举类型属于字面值常量类型，包括限定作用域的枚举类型：enum class声明，和不限定作用域的枚举类型：enum声明。默认情况下枚举值从0开始依次+1，但我们也可以为一个或几个枚举成员指定专门的值。**枚举成员是const**。
2. 默认情况下限定作用域的enum成员类型是int。对于不限定作用域的枚举类型来，其枚举成员不存在默认类型，只知道成员的潜在类型足够大，肯定能容纳枚举值。若指定了枚举成员的潜在类型则一旦某个枚举成员的值超出了该类型所能容纳的范围则将引发程序错误。

```
enum LongValue:long { val=255};
```

类成员指针

1. 成员指针指的是可以指向类的非静态成员的指针。一般情况下，指针指向的是一个对象，但是成员指针指示的是类的成员而非类的对象。

```
//声明一个指向 Screen类的const string成员 的指针
const string Screen::*pdata;
pdata=Screen::content;

Screen myScreen,*pScreen=&myScreen;
auto s=myScreen.*pdata; //获得myScreen对象的content成员
s=pScreen->*pdata;      //活动pScreen所指对象的content成员
```

union联合

1. union是一种特殊的类。一个union可以有多个数据成员，但是在任意时刻只有一个数据成员可以有值。当我们给union的某个成员赋值之后，其他成员就变成未定义的状态了。分配给一个union对象的存储空间至少要能容纳它的最大的数据成员。

固有的不可移植特性

1. 为了支持底层编程，C++定义了一些固有的不可移植特的特性。即因机器而异的特性，当我们将含有不可移植特性的程序从一台机器转移到另一台机器上时通常需要重新编写程序。
2. 位域：可以定义类的非静态数据成员，一个位域含有一定数量的二进制位，常用于向其他程序或硬件设备传递二进制数据。位域的类型必须是整型或者枚举类型。声明形式如下：

```
typedef unsigned int Bit;
Bit mode:2; //mode占2位
```

3. 若数据元素的值由程序直接控制之外的过程控制，例如程序可能包含一个由系统时钟顶是更新的变量。当对象的值可能在程序的控制或检测之外被改变时，应该将该对象声明为**volatile**，告诉编译器不应对这样的对象进行优化。
4. 链接指示：C++使用链接指示指出任意非C++函数所用的语言。想要把C++代码和其他语言编写的代码放在一起使用，要求我们必须有权访问其他语言的编译器，并且这个编译器与当前的C++编译器是兼容的。

```
//可能出现在C++头文件<cstring>中的链接指示

//单句链接指示
extern "C" size_t strlen(const char*);
//复合语句链接指示
extern "C" {
    int strcmp(const char*,const char*);
    char* strcat(char*,const char*);
}
```

链接指示使用关键字extern，后面的字面值常量指明了编写函数所用的语言。

5. 导出C++函数到其他语言：

```
//func()可以被C程序调用
extern "C" void func(){/****/};
```

链接指示的