# 纯虚函数和抽象类

Virtual 返回值 函数名（参数）=0

# 虚析构和纯虚析构

虚析构解决子类指针释放子类对象时不干净的问题

# 文件操作

Ofstream:写操作

Ifstream：读操作

Fstream：读写操作

# 写文件

1包含头文件

#include<fstream>

2流对象

Ofstream ofs；

3打开文件

Ofs.open(“文件路径”,“打开方式”);

4写入数据

ofs<<写入的数据；

5关闭文件

ofs.close();

# 读文件

1包含头文件

#include<fstream>

2创建流对象

ifstream ifs；

3打开文件，并判断是否打开成功

ifs.open(“文件路径”,“打开方式”);

4读取数据

第一种

Char buf[1024]={0};

while(ifs>>buf)

{

}

第二种

char buf[1024]={0};

while(ifs.getline(buf,sizeof(buf))

{

}

第三种

string buf;

while(getline(ifs,buf){}

第四章

char c;

while((c=ifs.get())!=EOF){}

5关闭文件

ifs.close();

# const限定符

## cosnt创建后值就不能改变，必须初始化。

## 只能在const类型的对象上执行不改变其内容的操作

## 默认状态下，const’对象仅仅在文件内有效，可以添加extern关键字来解决

# Const引用

## 不能将一个不同的引用他绑定到常量上

## 对常量的引用不能用来修改他所绑定的对象

1. int i = 42;

2. const int& r1 = i;/\*正确\*/

3. const int& r1 = r1 \* 2;/\*正确r3是一个常量引用\*/

4. int& r4 = r1 \* 2;/\*错误r4是一个普通话的引用\*/

## 临时量

1. double dval = 3.14;

2. const int& ri = dval;

3. /\* ri引用一个整数，dval却是一个双精度浮点数，

4. 编译器会将此代码变成下面的样子，ri绑定到了一个临时量的对象\*/

5. const int temp = dval;

6. const int &ri =temp;

## 常量引用仅仅对引用可参与的操作做出了限定，对引用的对象本身是不是常量则未作限定

## 顶层const指是指针本身是一个常量

## 底层const表示指针所指的对象是一个常量

1. int i = 0;

2. int\* const p1 = &i;/\*不能改变p1的值，这是一个顶层const；\*/

3. const int ci = 42;/\*不能改变ci的值，这是一个顶岑那个const\*/

4. const int\* p2 = &ci;/\*允许改变p2的值，这是一个顶层const\*/

5. const int\* const p3 = p2;/\*左边的是顶层指针，右边的是底层const\*/

6. const int& r = ci;/\*用于声明引用的const都是顶层const\*/

7.

# constexpr和常量表达式

常量表达式是不会改变把那个且在编译过程就能得到计算结果的表达式

C++11允许将变量声明为constexpr类型来由编译器来验证变量的值是否是一个常量表达式，声明为constexpr的变量一定是一个常量，而且必须用常量表达式初始化

# 指针和cosntexpr

1. const int\* p = nullptr;/\*p是一个只想常量的的指针\*/

2. constexpr int\* q = nullptr;/\*q是一个指向整数的常量指针，

3. cosntexpr把他所定义的对象置为顶层const\*/

4.

# 类型别名

1. typedef double wages;

2. typedef wages base, \* p;/\*base是double的同义词，p是double\*的同义词\*/

3. using SI = Sales\_item;

4.

# auto类型说明符

auto让编译器替我们去分析表达式所属的类型

1. auto& h = 42;/\*错误，不能为非常量引用绑定字面值，报错E0461 非常量引

2. 用的字面值必须为左值\*/

3. const auto& j = 42;/\*正确，可以为常量引用绑定字面值\*/

4.

# decltype类型指示符

希望从表达式的类型推断出要定义的变量的类型，但是不想用该表达式的值初始化变量就可以用decltype类型说明符，它的作用是选择并返回操作数的数据类型，在此过程中，编译器分析表达式并得到他的类型，却不实际计算表达式的值；

1. decltype(\*p) c;/\*错误，c是int&，必须初始化\*/

2. //decltype的结果类型与表达式形式密切相关，对于decltype，

3. //如果变量名加上一对括号则和和不加括号会有不同。

4. decltype((i)) d;/\*错误。d是int&，必须初始化\*/

5. decltype(i) e;/\*正确，e是一个未初始化的int\*/

decltype（（varible））的结果永远是引用

# 预处理器

确保头文件多次包含仍然能够安全工作的技术

1. #ifdef /\*当且仅当变量已定义时为真\*/

2. #ifndef /\*当且仅当变量未定义时为真\*/

3. #endif /\*一旦检查为真则执行后续操作知道#endif指令位置\*/

4.

# string

## 初始化是string

1. string s1; /\*默认初始化，s1是一个空串\*/

2. string s2(s1); /\*s2是s1的副本\*/

3. string s3 = s1;/\*等价于s2(s1),s2是s1的副本\*/

4. string s3("value");/\*s3是字面值“value”的副本，除了字面值最后的那个空字串外\*/

5. string s3 = "value";/\*等价于s3（“value”），s3是字面值“value”的副本\*/

6. string s4(8, 'c');/\*把s4初始化为由连续n个字符c组成的串\*/

7.

String上的操作

1. os << s /\*将s写道输出流os当中，返回os\*/

2. is >>s /\*从is中读取字符串赋值给s。字符串以空白分割，返回is\*/

3. getline(is,s) /\*从is中读取一行赋给s，返回is\*/

4. s.empty() /\*为空返回true，否则返回false\*/

5. s.size() /\*返回s中字符的个数\*/

6. s1+s2

7. s1=s2

8. s1 == s2

9. s1 != s2

10. <,<=,>,>=

11.

## String::size\_type类型

size函数返回的是一个string：：size\_type类型的值

1. auto len = s.size();/\* len的类型是stering::sizetype\*/

2.

# C++版本的c标准库头文件

C++标准库除了定义c++语言特有的功能外，也兼容了c语言的标准库，c语言的头文件形如name.h，c++则将这些文件命名为cname，去掉了.h后缀，而在文件名name之后加了字母c，合理的c表示这是一个属于c语言的标准库的头文件，因此cctype和ctype.h拓扑文件的内容是一样的的，只不过从命名规范上更符合c++语言的要求

在名为cname的头文件中定义的名字从属于命名空间std，而定义为.h的头文件中的则不同

# 标准库类型vector

## 定义和初始化

1. vector<T> v1 /\*空的vector，潜在类型是T类型，执行默认初始化\*/

2. vector<T> v2(v1) /\*是一个包含有v1所有元素的副本\*/

3. vector<T> v2=v1

4. vector<T> v3(n,val)

5. vector<T> v4(n) /\*v4包含了n个重复的执行了值初始化的对象\*/

6. vector<T> v5{a,b,c,d...}

7. vector<T> v5={a,b,c,d...}

8.

## vector支持的操作

1. v.empty();

2. v.size();

3. v.push\_back;

4. v[n];

5. v1 = v2;

6. v1 == v2;

7. v1 != v2;

# 迭代器

## 标准迭代器运算符

1. \*iter;

2. iter->mem;

3. ++iter;

4. --iter;

5. iter1 == iter2;

6. iter1 != iter2;

7.

begin和end返回的具体类型由对象是否是常量决定，如果对象是常量，begin和end返回cosnt\_iterator,如果不是则返回iterator；

迭代器运算

1. iter+n

2. iter-n

3. iter1+ =n

4. iter2-=n

5. iter1-iter2

6.

# C风格字符串

尽管c++支持c风格字符串，但是c++程序韩式不要使用他们，这是因为c风格字符串不仅使用起来不方便，而且极易引发程序漏洞，是诸多安全问题的根本原因。

C风格字符串不是一种类型，而是为了表达和使用字符串而新城的一种约定俗成的写法，字符数组以空格字符结束,字符串最后跟着一个空字符（’\0’）,一般利用指针来操作

## c标准库String函数

1. strlen(p)

2. strcmp(p1,p2)

3. strcat(p1,p2) /\*将p2附加到p1之后，返回p1\*/

4. strcpy(p1,p2) /\*讲p2拷贝到p1，返回p1\*/

5.

## C风格字符串比较的是指针而非字符串本身

If（ca1<ca2）比较的是地址

如果要比较内容则要调用strcmp函数

## 混用string对象和c风格字符串

string s(“Hello World”);

char \*str=s; //错误，不能用string对象初始化char\*

cosnt char \*str=c.c\_str(); //正确

## 使用数组初始化vector对象

1. int int\_rr[]={0,1,2,3,4,5,6,7,8,9};

2. vector<int> ivec(begin(int\_arr),end(int\_arr));

3. vector<int> subVec(int\_arr+1,int\_arr+4);

4.

# 多维数组

严格来说c++没有多维数组，通常所说的多维数组其实是数组的数组，紧急这一点

## 指针和多维数组

Int (\*p)[4]=ia;//p只想含有四个整数的数组；

p=&ia[2] //p指向ia的尾元素

# sizeof运算符

sizeof运算符返回一条表达式或一个类型名字所占的字节数，结果是一个size\_t类型的常量表达式

# 显示转换

## 命名的强制类型转换

形式是：cast-name<type>(expression);

cast-name是static\_cast , dynamic\_cast , cosnt\_cast和reinterpret\_cast中的一种

### static\_cast

任何具有明确定义的类型转换，只要不包含底层const，都可以使用static\_cast

### cosnt\_cast

cosnt\_cast只能改变运算对象的底层const

const char \*pc;

char \*p=const\_cast<cahr\*>(pc);

### reinterpret\_cast

## 旧的强制类型转换

type(expr) ;//函数形式的强制类型缓缓

(type)expr;//c语言风格的强制类型转换

# 避免强制类型转换

# goto语句

1. goto reai；

2. reai：

3.

# 内联函数