# 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，这是一个底层const\*/

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

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

## 直接初始化和拷贝初始化

String s5=”hiya”;//拷贝初始化

String s6(“hiya”);//直接初始化

String s7(10,’c’)//直接初始化

## 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.

1.

## 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：

# initializer\_list

如果函数的实参数量未知但是全部实参的类型都相同，我们可以使用initializer\_list类型的实参，initializer\_list是一个标注库类型，表示某种特定类型的值的数组

1. initializer\_list<T> lst;//默认初始化，T类型元素的空列表

2. initializer\_list<T> lst{a,b,c};/\*list的元素数量和初始值一样多，

3. lst的元素是对应初始值的副本，列表中的元素是const\*/

4. lst2(lst);lst2 = lst;/\*拷贝或者赋值一个initializer\_list对象不会拷贝

5. 列表中的元素，拷贝后，原始列表和副本共享元素\*/

6. lst.size();

7. lst.begin();//返回指向lst中首元素的指针

8. lst.end();//返回指向lst中尾元素的下一个位置的指针

# 函数重载

函数名字相同但是形参列表不同，我们称之为函数重载

## 重载与作用域

在内层作用域中声明名字，他将移仓外层作用域中声明的同名实体，在不同的作用域中无法重在函数名字

# 内联函数和cosntexpr函数

## 内联函数

内联机制用于优化规模较小，流程直接，频繁调用的函数，许多编译器不支持内联递归函数

1. inline cosnt string &

2. shortString(const string &s1,const string &s2)

3. {

4. retuen s1.size()<=s2.size()?s1:s2;

5. }

6.

## 内联说明只是向编译器发出一个请求，编译器可以忽略这个请求

## constexpr函数

能够用于常量表达式的函数，要遵循几项约定：函数返回值及所有形参的类型都必须是字面值类型，函数体中必须有一条return语句，允许返回hi并非常量，

1. constexpr int new\_sz() { return 66778899; }

2. constexpr int foo = new\_sz();//正确：foo是一个常量表达式

## 把内联函数和constexpr函数放在头文件内

# assert预处理宏

assert(expr)

对expr求值，如果表达式为假，则输出信息并且种植程序的执行，如果表达式为真，assert什么也不做

# NDEBUG预处理变量

assert的行为依赖于一个名为NDEBUG的预处理变量的状态，如果定义了NDEBUG则assert什么也不做，默认状态下没有定义NDEBUG，此时asssert将执行运行时检查

## 使用尾置返回类型

任何函数的定义都能够使用尾置返回，但是这种形式会与返回类型比较复杂的函数最有效

1. //func接受一个int类型的实参，返回一个指针，该指针指向含有十个整数的数组

2. auto func(int i) -> int(\*)[10];

# 函数指针

1. bool (\*pf)(const string&, const string&);//pf指向一个函数

2. pf = lengthCompare;//pf指向名为lengthCompare的函数

3. pf = &lengthCompare;//等价的赋值语句：取地址符是可选的

4. bool b1 = pf("heloo", "goodbye");//调用lengthCompare函数

5. bool b2 = (\*pf)("helo", "goodbye");//等价的调用

6. bool b3 = lengthCompare（"hello", "goodbye"）;//另外一个等价的调用

7.

能够为函数指针夫一个nullptr或者值为0的整形常量表达式，表示该指针没有指向任何一个函数

## 重载函数的指针

## 函数指针形参

和数组想想四，虽然不饿能定影函数类型的形参，但是形参是指向函数的指针

1. //第三个形参是函数类型，它会自动的转换成函数的指针

2. void useBigger(const string& s1, const string& s2, bool pf(const string&, const string&));

3. //等价的声明

4. void useBigger(const string& s1, const string& s2, bool （\*pf）(const string&, const string&));

我们可以直接把函数作为实参使用，自称是他会自动转换成指针

## 返回指向函数的指针

1. using F = int(int\*, int);//F是函数类型，不是指针

2. using PF = int(\*)(int\*, int);//PE是指针类型

3. PF f1(int);//正确：PF是指向函数的指针，F1返回指向函数的指针

4. F f1(int);//错误，F是函数类型，f1不能

5. F\* f1(int);//正确：显式的指定返回类型是指向函数的指针

也能够用下面的形式直接声明f1

int (\*f1(int))(int\*, int);

看到f1有形式参数列表，所以f1是个函数，f1前面有\*，所以f1返回的是一个指针，指针的本身也包含形参列表，因此指针指向函数，该函数的返回类型是int

处于完整性的考虑，有必要提醒读者我们可以使用为尾置返沪类型的方式声明一个返回函数指针的函数

auto f1(int)->int (\*)(int\*,int);

## 将auto和decltype用于函数指针类型

string::size\_type sumLength(const string&, const string&);

string::size\_type largerLength(const string&, const string&);

//根据器形参的取值，getFcn函数返回指向sumLength或者largeLength指针

decltype(sumLength)\* getfcn(const string&);

## std::function

std::function就是C++中用来代替C函数指针的

void print1(){

std::cout << "hello, print1" << std::endl;

}

void print2(){

std::cout << "hello, print2" << std::endl;

}

int main(int argc, char \*argv[])

{

std::function<void()> func(&print1);

func();

func = &print2;

func();

return 0;

}

输出结果

hello, print1

hello, print2

# 类

## =default的含义

在c++11新标准中，如果需要默认的行为，那么可以通过在参数列表后面写上=default来要求编译器生成构造函数，和其他函数一样，如果=default在类的内部，则默认构造函数是内联的，如果他在类的外部，则该成员默认情况下不是内联的

## 友元

类可以允许其他类或者函数访问它的非公有成员，方法是令其他类或者函数成为他的友元，如果类想啊吧一个函数作为它的友元，只需要增加一条以friend关键词开始的函数声明语句即可

类成员之间的友元关系

### 一般来说，最好在类定义开始或者结束前的位置集中声明友元。

## 令成员作为内联函数

## 可变数据成员（mutable data member）

有时候希望改变类的某个数据成员，即使在一个cnst成员函数内，可以通过在变量的生命中加入mutable关键字做到这一点

可变数据成员永远不会是

1. class Screen

2. {

3. private:

4. mutable size\_t access\_ctr;

5. public:

6. void some\_member() const;

7. };

8.

9. void Screen::some\_member()const

10. {

11. ++access\_ctr;

12. }

13.

友元声明和作用域

委托构造函数（cppprimer261）

## 默认构造函数

## 构造函数初始值列表

构造函数的初始值又是必不可少

1. class ConstRef {

2. public:

3. ConstRef(int ii);

4. private:

5. int i;

6. const int ci;

7. int& ri;

8. };

9. ConstRef::ConstRef(int ii) {

10. i = ii;

11. ci = ii;//错误不能给const赋值

12. ri = i;//ri没被初始化

13. }

14. ConstRef::ConstRef(int ii):i(ii),ci(ii),ri(i){}

15.

## 聚合类

聚合类使得用户可以直接访问其成员，并且具有特殊的初始化语法形式，当一个类满足如下条件的时候，我们说他是聚合的

# IO库

## 文件输入输出

1. char \*s={'a','b','c','\0'};

2. string ss = "woshireai";

3. fstream fstrm;//创建一个未绑定的文件流

4. fstream fstrm(s); /\*创建一个fstream, 并打开名为s的文件，

5. s可以是string类型，或者是一个指向c风格字符串的指针\*/

6. fstream fstrm(s, mode);//和前一个类似，但是是用置顶mode打开文件

7. fstrm.open(s); //打开名字为s的文件

8. fstrm.close();//关闭一个fstrm绑定的文件。返回void

9. fstrm.is\_open();//返回一个bool值，指出fstrm关联的文件是否成功打开而且尚未关闭

## string流

### 使用istringstream

1. #include<iostream>

2. #include<sstream>

3. #include<fstream>

4. #include <string>

5. #include <vector>

6. using namespace std;

7. int main() {

8. cout << "wo" << "shi" << unitbuf << "reai";

9. char s[] = {'6','7','8','9','\0'};

10. //string ss = "woshireai";

11. //fstream fstrm;//创建一个未绑定的文件流

12. //fstream fstrm(s); /\*创建一个fstream, 并打开名为s的文件，

13. // s可以是string类型，或者是一个指向c风格字符串的指针\*/

14. //fstream fstrm(s, mode);//和前一个类似，但是是用置顶mode打开文件

15. //fstrm.open(s); //打开名字为s的文件

16. //fstrm.close();//关闭一个fstrm绑定的文件。返回void

17. //fstrm.is\_open();//返回一个bool值，指出fstrm关联的文件是否成功打开而且尚未关闭

18. struct PersonInfo {

19. string name;

20. vector<string> phones;

21. };

22. string line, word;

23. vector<PersonInfo> people;

24.

25. while (getline(cin, line)) {

26. PersonInfo info;

27. istringstream record(line);

28. record >> info.name;

29. while (record>>word)

30. {

31. info.phones.push\_back(word);

32. }

33. people.push\_back(info);

34. }

35. for (size\_t i = 0; i < people.size(); i++)

36. {

37. cout << people[i].name;

38. for (int j = 0; j < people[i].phones.size(); j++)

39. cout << people[i].phones[j];

40. }

41. }

42.

## 顺序容器

vector //可变大小数组，支持快速随机访问，在尾部之外的位置插入或删除慢

deque //双端队列，支持快速随机访问，在头尾插入快

list //双向链表，只支持双向顺序访问，在list中任何位置进行插入删除都很快

forward\_list //单向链表，只支持单项顺序访问，在链表任何位置进行插入删除都很快

array //固定大小数组，支持快速随机访问，不能添加或者删除元素

string //和vector相似的容器，专门用于保存字符，随机访问快，在尾部插入删除，速度快

### 选择原则

除非有很好的理由选择其他容器，否则选择vector

程序有很多小元素，而且额外开销很重要，则不要是太阳list或者forward\_list

如果程序员要求随机访问元素，应使用vector或者deque

如果程序要在头尾位置插入或者删除元素，但不会在中间插入或删除则使用deque

## 容器库c++primer294

iterator //此容器类型的迭代类型

const\_iterator //可以读取元素，但是不能改变

size\_type //无符号整数类型，足够保存此种容器类型最大可能容器的大小

difference\_type //带符号整数类型，足够保存两个迭代器之间的距离

value\_type //元素类型

reference //元素的左值类型，与value\_type&含义相同

const\_reference //元素的const左值类型即cosnt value\_type&

//构造函数

C c; //默认构造函数

C c1(c2);//构造c2的拷贝c1

C c(b, e);//构造c,将迭代器b和e指定范围内的元素拷贝到c（array）不支持

C c(a,b,c,d,e)

# 迭代器

与容器一样，迭代器有着公共的接口，如果迭代器提供某个操作，那么所有提供相同操作的迭代器对这个操作的实现是一样样的

例如标准容器类型上的所有迭代容器都允许我们访问容器中的元素，而所有迭代器都是通过解引用运算符来实现这个操作的，类似的标准库容器的所有迭代器都定义了递增运算符，从当前元素移动到下一个元素

# 回调函数

