C++程序设计精要教程

华中科技大学

第5章 成员及成员指针

◆5.1 实例成员指针

- 运算符.*和->*均为双目运算符，优先级均为第14级，结合性自左向右
- .*的左操作数为类的实例(对象)，右操作数为指向实例成员的指针
- ->*的左操作数为对象指针，右操作数为指向该对象实例成员的指针。
- 实例成员指针是指向实例成员的指针，可分为实例数据成员指针和实例函数成员指针。
- 实例成员指针必须直接或间接同.*或->*左边的实例(对象)结合，以便访问该对象的实例数据成员或函数成员。
- 构造函数不能被显式调用，故不能有指向构造函数的实例成员指针。

第5章 成员及成员指针

◆5.1 实例成员指针

- 实例成员指针是成员相对于对象首地址的偏移，不是真正的代表地址的指针。
- 实例成员指针不能移动：
  - 实例数据成员的大小及类型不一定相同，移动后指向的内存可能是某个成员的一部分，或者跨越两个(或以上)成员的内存；
  - 即使移动前后指向的成员的类型正好相同，这两个成员的访问权限也有可能不同，移动后可能出现越权访问问题。
- 实例成员指针不能转换类型：
- 否则便可以通过类型转换，间接实现实例成员指针移动

第5章 成员及成员指针

【例5.2】本例说明普通成员指针不能移动。

```
#include <iostream>
using namespace std;
struct A{
   int i;     //公有的成员i
private:
   long j;
public:
   int f( ){ cout<<"F\n";   return 1;  }
private:
   void  g( ){ cout<<"Function g\n";  }
}a;
```

第5章 成员及成员指针

```
void main(void){
    int  A::*pi=&A::i;          //普通数据成员指针pi指向public成员A::i
    int(A::*pf)( )=&A::f;       //普通函数成员指针pf指向函数成员A::f
    long  x=a.*pi;              //等价于x=a.*(&A::i)=a.A::i=a.i
    x=(a.*pf)( );               //.*的优先级低，故用(a.*pf)
    pi++;                       //错误， pi不能移动，否则指向私有成员j
    pf+=1;                      //错误， pf不能移动
    x=(long) pi;                //错误，pi不能转换为长整型
    x=x+ sizeof(int)            //间接移动指针
    pi=(int A::*)x;             //错误，x不能转换为成员指针
}
```

第5章 成员及成员指针

◆5.2 const、volatile和mutable

- **const只读，volatile易变，mutable机动：**
- **const和volatile可以定义变量、类的数据成员、函数成员及普通函数的参数和返回类型。**
- **mutable只能用来定义类的实例数据成员。**
- **含const实例数据成员的类必须定义构造函数，且该实例数据成员必须在构造函数参数表之后、函数体之前初始化。**
- **含volatile、mutable实例数据成员的类则不一定需要定义构造函数。**

第5章 成员及成员指针

【例5. 3】定义导师类，允许改名但不允许改性别。

```
#include <string.h>
#include <iostream>
using namespace std;
class TUTOR{
    char            name[20];
    const           char  sex; //性别为只读成员
    int    salary;
public:
    TUTOR(const char *name, const TUTOR *t);
    TUTOR(const char *name, char gender, int salary);
    const char *getname( ) { return name; }
    char *setname(const char *name);
};
```

第5章 成员及成员指针

```
TUTOR::TUTOR(const char *n, const TUTOR *t): sex(t->sex){
   strcpy(name,n);    salary=t->salary;
}//只读成员sex必须在构造函数体之前初始化
TUTOR::TUTOR(const char *n, char g, int s): sex(g),sarlary(s){
   strcpy(name,n);
}//非只读成员sarlary可在函数体前初始化，也可在体内再次赋值
char *TUTOR::setname(const char*n){
   return strcpy(name, n); //注意：strcpy的返回值为name
}
void main(void){
   TUTOR  wang("wang", 'F', 2000);
   TUTOR  yang("yang", &wang);
   *wang.getname( )='W';   //错误:不能改wang.getname( )指的字符
   *yang.setname("Zang")='Y';
}
```

第5章 成员及成员指针

◆5.2 const、volatile和mutable

- 普通函数成员参数表后出现const或volatile，修饰隐含参数this指向的对象。出现const表示this指向的对象(其非静态数据成员)不能被函数修改，但可以修改this指向对象的非只读类型的静态数据成员。
- 构造或析构函数的this不能被说明为const或volatile的(即要构造或析构的对象应该能被修改，且状态要稳定不易变)。
- 对隐含参数的修饰还会会影响函数成员的重载：
- 普通对象应调用参数表后不带const和volatile的函数成员；
- const和volatile对象应分别调用参数表后出现const和volatile的函数成员，否则编译程序会对函数调用发出警告。

第5章 成员及成员指针

【例5.4】参数表后出现const和volatile。

```
#include <iostream>
class A{
   int a;  const  int  b; //b为const成员或y引用时，只能在构造函数初始化
public:
   int f( ){a++; //this类型为A * const  this, 指向的
         return a;//对象可修改(故其普通成员a可修改)，只读成员b不可改
   }
    int f( )const{//a++;          //this类型为const A * const  this，指向的对象
         return a;//不可改，其普通成员a不可改。同上, b不可改
   }
   int f( )volatile{//this类型为volatile A * const this，指向的对象可修
        a++;          //改，其普通成员a可修改。同上，只读成员b不可改
        return a;
   }
```

第5章 成员及成员指针

```
    int f( )const volatile{      //this类型为const volatile A* const this，
        //a++;          //不能修改普通成员a。同上，只读成员b不可改
        return a;
    }
     A(int x)  : b(x) { a=x; }
} x(3);                          //等价于A  x(3), x可修改,
const A y(6);                    // y、z不可改
const volatile A z(8);           // x、y、z由开工函数构造、收工函数析构
void main(void) {
    x.f( ); //普通对象x调用int f( ): 指向的对象可修改
    y.f( ); //只读对象y调用int f( )const:指向的对象不可修改
    z.f( ); //只读易变对象z调用int f( )const volatile
}
```

第5章 成员及成员指针

◆5.2 const、volatile和mutable

- 函数成员参数表后出现volatile，常表示调用该函数成员的对象是挥发对象，这通常意味着存在并发执行的进程。
- C++编译程序几乎都支持编写并发进程，编译时不对挥发对象作任何访问优化，即不利用寄存器存放中间计算结果，而是直接访问对象内存以便获得对象的最新值。
- 函数成员参数表后出现const时，不能修改调用对象的实例(即非静态)数据成员，但若该实例数据成员的存储类为mutable，则该数据成员就可以被修改。
- mutable说明实例数据成员为机动数据成员，该类成员不能用const、volatile或static修饰。

第5章 成员及成员指针

◆5.2 const、volatile和mutable

- 有址引用变量(&)只是被引用对象的别名，被引用对象自己负责构造和析构，该引用变量(逻辑上不分配内存的实体)不必构造和析构。
- 无址引用变量(&&)常用来引用缓存中的常量对象，该引用变量(逻辑上不分配缓存的实体)不必构造和析构。无址引用变量可为左值，但若同时用const定义则为传统右值。
- 如果A类型的有址引用变量r引用了new生成的(一定有址的）对象x，则应使用delete &r析构x，同时释放其所占内存。
- r.~A( )仅析构x而不释放其所占内存(由new分配)，造成内存泄漏。应该用delete &r;
- 引用变量必须在定义的同时初始化，引用参数则在调用函数时初始化。有址传统左值引用变量和参数必须用同类型的左值表达式初始化。

第5章 成员及成员指针

◆5.2 const、volatile和mutable

- **mutable仅用于说明实例数据成员为机动成员，不能用于静态数据成员的。**
- **所谓机动是指在整个对象为只读状态时，其每个成员理论上都是不可写的，但若某个成员是mutable成员，则该成员在此状态是可写的。**
- **例如，产品对象的信息在查询时应处于只读状态，但是其成员“查询次数”应在此状态可写，故可以定义为“机动”成员。**
- **保留字mutable还可用于定义Lambda表达式的参数列表是否允许在Lambda的表达式内修改捕获的外部的参数列表的值。**

第5章 成员及成员指针

◆5.2 const、volatile和mutable

```
class PRODUCT {
    char* name;                          //产品名称
    int  price;                          //产品价格
    int  quantity;                       //产品数量
    mutable int count;                   //产品查询次数
public:
    PRODUCT(const char* n, int m, int p);
    int buy(int money);
    void get(int& p, int& q)const;
    ~PRODUCT(void);
};
void PRODUCT::get(int &p, int &q)const{//const  PRODUCT *const this
    p=price;      q=quantity; //当前对象为const对象，故其成员不能被修改
    count++;                  //但count为mutable成员，可以修改
}
```

第5章 成员及成员指针

◆5.3 静态数据成员

- 静态数据成员是使用static说明或定义的类的数据成员。
- 静态数据成员通常在类的里面说明，在类的外面唯一定义一次。
- 静态数据成员一般用来描述类的总体信息，例如对象总个数。
- 实例数据成员可以定义默认值，但静态数据成员不能定义默认值。
- 静态数据成员在类中初始化只能定义为inline static、const static、constexpr static, const inline static类型(若为初始化，则在static之前必须出现inline, const, constexpr其中之一或其中若干修饰符，顺序不限)。
- 静态数据成员不管是否用inline、const说明，在所有代码文件只有一个副本。
- 函数中的局部类不能定义静态数据成员，容易造成生命期矛盾。
- 静态数据成员不能定义为位段成员。

第5章 成员及成员指针

◆5.3 静态数据成员

例5.9函数局部类不能定义静态数据成员，全局类可以定义inline或const静态数据成员。

```
int  x=3;
union  S {
   const  static int b=0;       //全局类中可用const定义同时初始化静态成员,必须用常量
   inline  static int c = x;    //全局类可用inline定义同时初始化静态成员，可用任意表达式
   inline  const static int d = x;  //可用任意表达式
};
void f(void){
   class  T{                    //定义函数中的局部类T
      int  c;                   //static int d;  //错误：函数中的局部类不能定义静态数据成员
   };
   T a;                         //局部自动变量a
   static T s;                  //局部静态变量s
}
void main( ){ f( ); f( ); }     //第一个函数调用f( )返回后a.d⇔s.d⇔T::d产生生成矛盾
```

第5章 成员及成员指针

◆5.4 静态函数成员

- 静态函数成员通常在类里以static说明或定义，它没有this参数。
- 有this的构造和析构函数、虚函数及纯虚函数都不能定义为静态函数成员。
- 静态函数成员一般用来访问类的总体信息，例如对象总个数。
- 静态函数成员可以重载、内联、定义默认值参数。
- 静态函数成员同实例成员的继承、访问规则没有没有太大区别。
- 静态函数成员的参数表后不能出现const、volatile、const volatile等修饰符。
- 静态函数成员的返回类型可以同时使用inline、const、volatile等修饰。

第5章 成员及成员指针

◆5.4 静态函数成员

```
class A{
    double  i;
    const static int  j=3;
public:
    static A& inc(A &);        //说明静态函数成员
    static A& dec(A &a){       //在内体内定义静态函数成员：默认使用inline
        a.i=a.i-A::j;
        return a;
     }
};
A& A::inc(A&a){                //不能定义static  A& A::inc(A&a)
    a.i+=A::j;                 //静态函数可访问静态数据成员
    return a;
}
```

第5章 成员及成员指针

◆5.5 静态成员指针

- 静态成员指针是指向类的静态成员的指针，包括静态数据成员指针和静态函数成员指针。
- 静态数据成员的存储单元为该类所有的对象共享，因此，通过该指针修改成员的值时会影响到所有对象该成员的值。
- 静态数据成员除了具有访问权限外，同普通变量没有本质区别；静态成员指针则和普通指针没有任何区别。
- 变量、数据成员、普通函数和函数成员的参数和返回值都可以定义成静态成员指针。

第5章 成员及成员指针

◆5.5 静态成员指针

【例5.15】定义群众类，使每个群众共享人数信息。

```
#include <iostream>
using namespace std;
class  CROWD{
   int    age;
   char   name[20];
public:
   static int number;
   static int getn( ) { return number;}
 CROWD(char *n, int a){
      strcpy(name,n);
      age=a;   number++;
   }
   ~CROWD( ) { number – –; }
};
int CROWD::number=0;
void main(void){
   int *d=&CROWD::number; //普通指针指向静态数据成员
   int (*f)( )=&CROWD::getn; //普通函数指针指向静态函数成员
   cout<<“\nCrowd number=”<<*d;
   //类CROWD无对象时访问静态成员
   CROWD zan("zan", 20);
   //d=&zan.number;  等价于如下
   d=&CROWD::number;
   cout<<"\nCrowd number="<<*d;
   CROWD tan("tan", 21);
   cout<<"\nCrowd number="<<(*f)( );
}
```

第5章 成员及成员指针

◆5.5 静态成员指针

- 静态成员指针与普通成员指针有很大区别。静态成员指针存放成员地址，普通成员指针存放成员偏移；静态成员指针可以移动，普通成员指针不能移动；静态成员指针可以强制转换类型，普通成员指针不能强制转换类型。

```
struct A{
    int a, *b;
    int A::*u;   int A::*A::*x;
    int A::**y; int *A::*z;
    static int c, A::*d;
};
int A::c=0;
int A::*A::d=&A::a;
void main(void){
    int i, A::**m;
    z.a=5;     z.u=&A::a;   i=z.*z.u;
    z.x=&A::u;      i=z.*(z.*z.x);
    m=&A::d;
    m=&z.u;         i=z.**m;
    z.y=&z.u;       i=z.**z.y;
    z.b=&z.a;
    z.z=&A::b;      i=*(z.*z.z);
}
```

第5章 成员及成员指针

◆5.6 联合的成员指针

- 函数中局部类不能定义静态数据成员，故函数中的局部联合也不能定义。
- 全局类中的联合或全局联合可以定义静态数据成员。
- 故静态数据成员指针一般指向全局类中的联合或全局联合的静态数据成员。
- 联合可以定义实例和静态函数成员，故也可以定义实例和静态函数成员指针。
- 联合的实例数据成员共享内存，因此，指向这些实例数据成员的指针存储的偏移量值实际上是相同的。