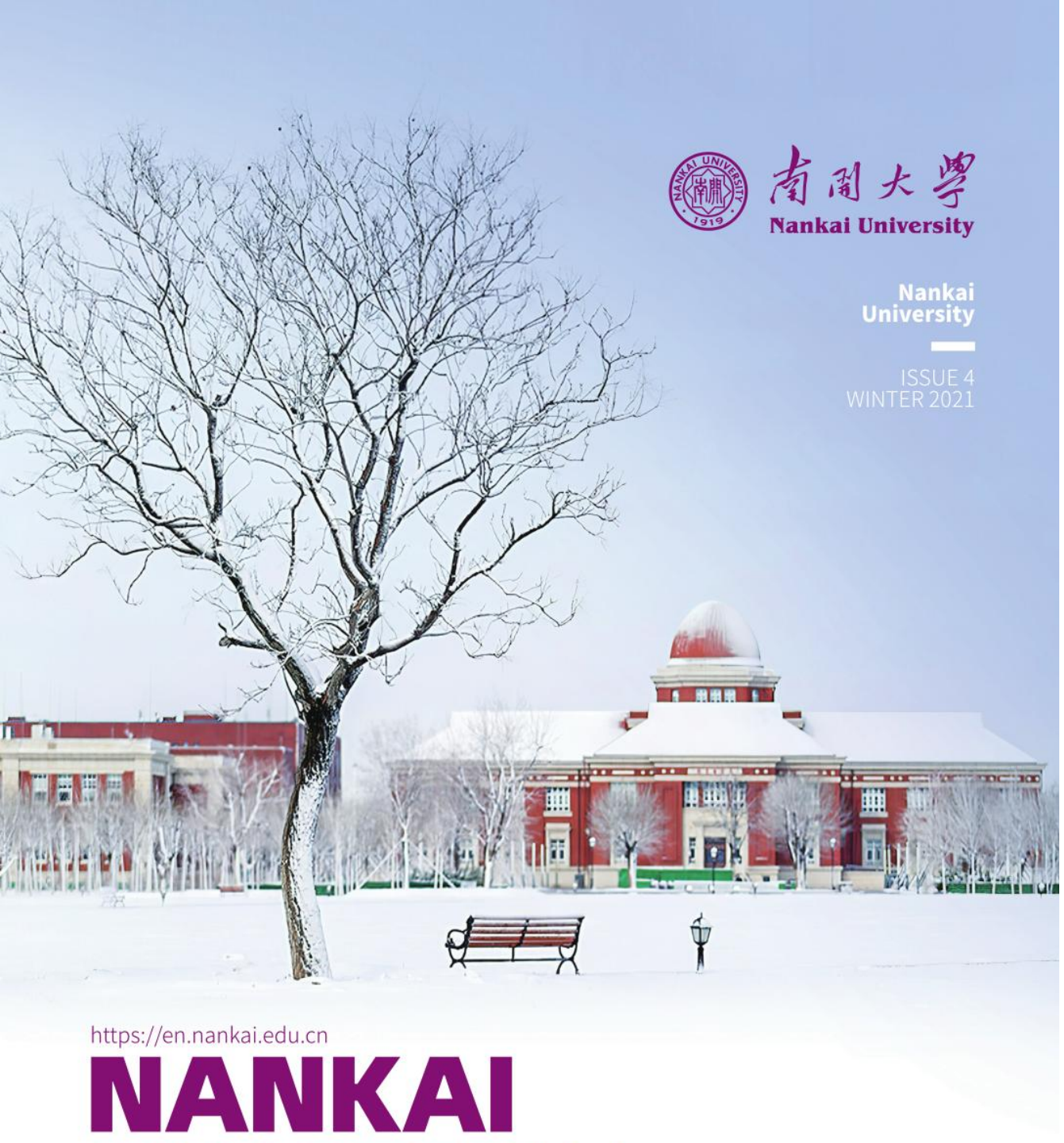

05 特殊类成员

龚成

cheng-gong@nankai.edu.cn

类的特殊成员

静态数据成员

静态函数成员

常量数据成员

常量函数成员

静态数据成员

- 类的静态成员为其所有对象所共享，不管有多少对象，静态成员都只有一份存于公用内存中。
- 类的静态数据成员：其数据将被该类的所有对象所共享（同一个类的不同对象的非静态数据成员所占据的存储空间是相互独立的）。

静态数据成员

- 注意：静态数据成员说明属于引用性说明，还必须在类外文件作用域中的某个地方对静态数据成员按如下格式进行定义性说明（且仅能说明一次）：

〈类型〉〈类名〉::〈静态数据成员〉 = 〈初值〉;

静态数据成员

- 类的静态数据成员在一个类中只有一份拷贝，不为某一个类对象所特有，对类的静态数据成员的访问**通常使用：**

  〈类名〉::〈静态数据成员名〉

- **也可通过“〈对象名〉.〈静态数据成员名〉”的方式来进行访问，但由于只有一份拷贝，其中的〈对象名〉也只是起到一个类名的作用而已。**

静态数据成员

- 设 float xcoord 是类 point 的静态数据成员，p、q 是 point 的对象，则在任何情况下：p.getx()和 q.getx() 将永远同时取相同的值（假设 getx 的功能是获取静态数据成员 xcoord 的值）。
- 如果 xcoord 是公有静态数据成员，那么p.xcoord、q.xcoord、point::xcoord实质上是相同的，指向同一个数据内存空间。

```
class Point{
public:
    Point(int x, int y):x(x),y(y){}
    int x, y;
    static float xcoord, ycoord;
};
Point::xcoord = 0;
Point::ycoord = 1;
```

```
int main(){
    Point p1, p2;
    cout<<Point::xcoord<<", "<<&Point::xcoord<<endl;
    cout<<Point::ycoord<<", "<<&Point::ycoord<<endl;
    cout<<p1.xcoord<<", "<<&p1.xcoord<<endl;
    cout<<p1.ycoord<<", "<<&p1.ycoord<<endl;
    cout<<p2.xcoord<<", "<<&p2.xcoord<<endl;
    cout<<p2.xcoord<<", "<<&p2.ycoord<<endl;
    return 0;
}
```

静态函数成员

- 函数成员被说明成静态的，同样与该类的不同对象无关。对类的静态函数成员的引用通常使用“<类名>::<静态函数成员调用>”的方式（当然也可通过“<对象名>.<静态函数成员调用>”的方式）。
- 类的静态函数成员与类的非静态函数成员的最大区别在于：类的静态函数成员没有this指针，从而无法处理不同调用者对象的各自数据成员值。
- 通常情况下，类的静态函数只处理类的静态数据成员值（它只隶属于类而不属于任一个特定对象）。若要访问类中的非静态成员时，必须借助对象名或指向对象的指针的函数参数。

静态函数成员

- 到目前为止，使用 static 可进行的五种说明有：**局部静态变量、全局静态变量、具有静态存储类别的函数、类的静态数据成员以及类的静态函数成员**。下面对它们稍做区别：
  - 局部静态变量是指在函数或块的内部说明的静态变量，它的**作用域仅局部于函数或块（出函数或块后则不可见），但它的“生命期”却与整个程序的执行期相同**。
  - 全局静态变量指的是在所有函数的外部说明的具有单文件级全局性的静态变量（相对多文件程序来说，其作用域也是局部的）。它的**作用域仅局部于单文件（在其他文件中不可见，从而不可在其他文件中被使用），但它的“生命期”却与整个程序的执行期相同**。
  - 具有静态存储类别的函数称为**静态函数（有时也称为内部函数）。这种函数只具有文件级作用域（也称单文件级作用域），即这样的函数只能在本文件的内部被调用，在其他文件中均不可见，从而在其他文件中不能调用这种函数**。
  - 类的静态数据成员是指由关键字 static 修饰的类中的数据成员。 **类的静态数据成员为该类的所有对象所共享。 对类的静态数据成员的访问通常使用“<类名>::<静态数据成员名>”的方式**。
  - 类的静态函数成员是指由关键字 static 修饰的类中的函数成员。**对类的静态函数成员的引用通常使用“<类名>::<静态函数成员调用>”的方式。类的静态函数成员没有this 指针，通常只在其中处理类的静态数据成员值**。

常量数据成员

- 常量数据成员，通过关键字 const 修饰的类中的数据成员。它不同于一般的符号常量，在成员说明时不能被赋值，而只能在对象被说明时通过构造函数的成员初始化列表的方式来赋初值。（因为只有创建对象时才能初始化）
- 一旦对象被创建，其常量数据成员的值就不允许被修改，任何类内外函数只可读取其值，但不可改变它。

```
class CC {
        int i;
        const int c1;              //私有的常量数据成员 c1
public:
        const int c2;              //公有的常量数据成员 c2
        CC(int a,int b):c1(a),c2(b){
        //成员初始化列表 c1(a)、c2(b)将实参 a 与 b 的值赋给 c1 和 c2
        i=c1;
        };
  ...
  };
```

```
CC cobj(4,7);
```

常量函数成员

- 类的函数成员也可以被说明为常量类型。**常量类型的函数成员只有权读取相应对象（即调用者对象* this）的内容，但无权修改它们**。
- 类的常量函数成员的说明格式如下：

  〈类型说明符〉〈函数名〉（〈参数表〉）const;

- 注意：**修饰符const 要加在函数说明的尾部**（放在首部表示对函数值的修饰），它是函数类型的一部分。在该函数的实现部分也要加 const 关键字。
- 当一个函数成员被 const 说明后，其**函数中出现的对该对象（即* this）的任何写入或修改都将被系统检查为出错**。

```
class CC {
  int me;
public:
    int  readme()const{return  me;}        //常量成员函数
```

常量函数成员

- 系统实现这一常量型函数成员禁止写入和修改对象内容的功能，是通过把常量型函数成员中隐含的 this指针说明为：const CC *const this;型的，其中前面的 const 指出 this 的内容不变（总是指向该对象），后面的 const 则指出* this 不可改变。
- 当程序员需要对对象做小的修改时，采用非正规的方法：

```
int readme() const { return me++; }
```

是非法的，因为 me++不被允许，但使用：

```
int readme() const { return((CC* )this)->me++; }
```

就不会出错。因为把 this 指针的类型强迫转换为 CC* 型。