面向对象程序设计
420420

对象的定义和使用

- 1、对象的定义
- 2、对象的动态建立和释放

26.1 对象的定义

- 定义一个类时，也就是定义了一个具体的数据类型。若要使用类，需要将类实例化，即定义该类的对象。

- 需要注意，我们之前也使用了“对象”一词，那里主要是指数据对象。

- 从现在起，“对象”一词专门表示类的实体。

- 1．先定义类类型再定义对象
- 有两种定义对象的形式：
- ①将类的名字直接用作类型名：

```
类名 对象名列表;
```

- ②指定关键字class或struct，后面跟类名（ 兼容C语言特色 ）：

```
class 类名 对象名列表;
或
struct 类名 对象名列表;
```

- 对象名列表是一个或多个对象的序列，各对象之间用逗号（，）分隔，最后必须用分号（；）结束，对象取名必须遵循标识符的命名规则。例如：

```
Point   a,b;   //C++特色定义对象
class   Point   x,y;   //兼容C语言特色定义对象
```

- 2. 定义类类型的同时定义对象
- 一般形式为：

```
class  类名  {  //类体
        成员列表
}  对象名列表;
```

- 例如：

```
class  Point  {  //类体
  public:     …     //公有的数据成员和成员函数
  private:    …     //私有的数据成员和成员函数
}  one  ,  two;  //对象列表
```

- 3. 直接定义对象
- 一般形式为：

```
class  {  //类体
          成员列表
}  对象名列表;
```

- 例如：

```
class  {     //无类名的类体
public:    … //公有的数据成员和成员函数
private:   … //私有的数据成员和成员函数
}  p1  ,  p2;  //对象列表
```

- 声明类型时不进行存储空间的分配。
- 当定义一个对象时，才为其分配存储空间。

- 有时希望在需要用到对象时才创建（create）对象，在不需要用该对象时就撤销（destroy）它，释放其所占的存储空间，从而提高存储空间的利用率。

- 利用new运算符可以动态地分配对象空间，delete运算符释放对象空间。

- 动态分配对象的一般形式为：

```
类名 * 对象指针变量;
对象指针变量 = new 类名;
```

- 例如：

```
Point *p; //定义指向Point对象的指针变量p
p = new Point; //动态分配Point对象
```

- 用new运算动态分配得到的对象是无名的，它返回一个指向新对象的指针的值，即分配得到是对象的内存单元的起始地址。程序通过这个地址可以间接访问这个对象，因此需要定义一个指向类的对象的指针变量来存放该地址。总结，用new建立的动态对象是通过指针来引用的。
- 在执行new运算时，如果内存不足，无法开辟所需的内存空间，C++编译器会返回一个0值指针。因此，只要检测返回值是否为0，就可以判断动态分配对象是否成功，只有指针有效时才能使用对象指针。

- 当不再需要使用由new建立的动态对象时，必须用delete运算予以撤销。例如：

```
delete   p;  //撤销p所指向的动态生成的对象
```

- 释放了p所指向的对象。此后程序不能再使用该对象。

- 请注意，new建立的动态对象不会自动被撤销，即使程序运行结束也是如此，必须人为使用delete撤销。

【例26.1】

```
#include<iostream>
using namespace std;
class Box
{ public:
    int width, length, height;
};
int main()
{
    Box * p=new Box;
    p->width=10; p->length=20; p->height=30;
    cout<<p->width<<"\t"<<p->length<<"\t"<<p->height<<endl;
    delete p;
    return 0;
}
```

对象的定义和使用

- 3、对象成员的引用
- 4、对象的赋值
- 5、对象、对象指针或对象引用作为函数的参数和返回值

- 访问对象中的成员可以有3种方法：

- ①通过对象名和对象成员引用运算符（.）访问对象中的成员；
- ②通过指向对象的指针和指针成员引用运算符（->）访问对象中的成员；
- ③通过对象的引用变量和对象成员引用运算符（.）访问对象中的成员；

- 1．通过对象名访问对象中的成员

- 访问对象中数据成员的一般形式为：

```
对象名.成员名
```

- 调用对象中成员函数的一般形式为：

```
对象名.成员函数（实参列表）
```

- 需要注意，从类外部只能访问类公有的成员。

- 例如：定义一个Data类：

```
class  Data  {  //Data类
public:
        int  data;                              //公有数据成员
        void  fun(int  a,int  b,int  d);  //公有成员函数
private:
        void  add(int  m)  {  data+=m;  } //私有成员函数
        int  x,y;                                  //私有数据成员
        ……
};
```

- 函数caller1

```
1 void   caller1()
2 {
3       Data   A,   B;  //定义对象
4       A.fun(1,2,3);     ?
5       A.data=100;       ?
6       A.add(5);         ?
7       A.x=A.y=1;        ?
8       B.data=101;  //正确，A.data和B.data是两个对象的数据成员，
9                    // 为不同的存储空间
10      B.fun(4,5,6);  //正确，A.fun和B.fun函数调用相同的代码，但
11                     // 作用不同的数据成员
12}
```

- 函数caller1

```
1 void   caller1()
2 {
3       Data   A,   B;   //定义对象
4       A.fun(1,2,3);  //正确，类外部可以访问类的public成员函数
5       A.data=100;  //正确，类外部可以访问类的public数据成员
6       A.add(5);      //错误，类外部不能访问类任何private的成员
7       A.x=A.y=1;    //错误，类外部不能访问类任何private的成员
8       B.data=101;  //正确，A.data和B.data是两个对象的数据成员，
9                          // 为不同的存储空间
10      B.fun(4,5,6);  //正确，A.fun和B.fun函数调用相同的代码，但
11                         // 作用不同的数据成员
  }
```

- 2．通过对象指针访问对象中的成员

- 访问对象中数据成员的一般形式为：

```
对象指针—>成员名
```

- 调用对象中成员函数的一般形式为：

```
对象指针—>成员函数（实参列表）
```

- 函数caller1

```
1  void  caller1()
2  {
3          Data  A,  *p,  *p1;  //定义对象指针变量
4          p1=&A;  //p1指向对象A
5          p1->data=100;  //正确，类外部可以访问类的public数据成员
6          p1->fun(1,2,3);  //正确，类外部可以访问类的public成员函数
7          p  =  new  Data;  //动态分配Data对象
8          p->data=100;  //正确，类外部可以访问类的public数据成员
9          p->fun(1,2,3);  //正确，类外部可以访问类的public成员函数
10         delete  p;  //撤销p所指向的Data对象
11 }
```

- 3．通过引用变量访问对象中的成员

- 访问对象中数据成员的一般形式为：

```
对象引用变量名.数据成员名
```

- 调用对象中成员函数的一般形式为：

```
对象引用变量名.成员函数(实参列表)
```

- 函数caller1

```
void   caller1()
{
        Data   A,   &r=A;   //定义对象引用变量
        r.data=100;   //正确，类外部可以访问类的public数据成员
        r.fun(1,2,3);   //正确，类外部可以访问类的public成员函数
}
```

26.4 对象的赋值

- 如果一个类定义了两个或多个对象，则这些同类的对象之间可以互相赋值。这里所指的对象的“值”是指对象中所有数据成员的值。

- 对象赋值的一般形式为：

```
对象名1=对象名2
```

- 注意能赋值的前提是：对象名1和对象名2必须属于同一个类。

- 关于对象赋值的说明：

- （1）对象的赋值只对其中的非静态数据成员赋值，而不对成员函数赋值。

- （2）如果对象的数据成员中包括动态分配资源的指针，按上述赋值的原理，赋值时只复制了指针值而没有复制指针所指向的内容。

26.5 对象、对象指针或对象引用作为函数的参数和返回值

- 函数的参数可以是对象、对象指针或对象引用。

- （1）当形参是对象时，实参要求是相同类的对象名，C++不能对类对象进行任何隐式类型转换。此时形参是实参对象的副本。
- 采用这样的值传递方式会增加函数调用在空间、时间上的开销，特别是当数据成员的长度很大时，开销会急剧增加。
- 实际编程中，传递对象时需要考虑类的规模带来的调用开销，如果开销很大时建议不用对象作为函数参数。

- （2）当形参是对象指针时，实参要求是同类对象的指针，C++不能对对象指针进行任何隐式类型转换。

- 函数调用时，无论类多大规模，传递的参数是一个地址值，其开销非常小。

- 采用地址传递方式，在函数中若按间接引用方式修改了形参对象本质上就是修改实参对象。因此，使用对象指针作为函数参数可以向主调函数传回变化后的对象。

- （3）当形参是对象引用时，实参要求是同类的对象，其功能与对象指针相似。

- 此时函数形参对象实际上是实参对象的别名，为同一个对象。在函数中若修改了形参对象本质上就是修改实参对象。因此，使用对象引用作为函数参数可以向主调函数传回变化后的对象。

26.5 对象、对象指针或对象引用作为函数的参数和返回值

【例26.2】

```
class  Data  {  //Data类
   public:
      int  data;
      void  fun(int  a,int  b,int  d);
   private:
    void  add(int  m)  { data+=m; }
      int  x,y;
      ……
}
```

```
1  #include  <iostream>
2  using  namespace  std;
3  void  func1(Data  a,Data  *p,Data  &r)
4  {
5      a.data=100;    p->data=200;    r.data=300;
6  }
7  int  main()
8  {
9      Data  A,  B,  C;
10     A.fun(1,2,3);  B.fun(4,5,6);  C.fun(7,8,9);
11     func1(A,&B,C);//将对象A、B的地址、对象C的引用传递到函数func1
12     return  0;
13 }
```

- 如果不希望在函数中修改实参对象的值，函数形参可以作const限定，例如：

```
void func2(Data a,const Data *p,const Data &r)
{
        a.data=100;
        p->data=200;  //错误，左值是const对象
        r.data=300;  //错误，左值是const对象
}
```

- 无需对对象形参作const限定，如上述代码中的形参a，因为即使在func2函数中修改了a，也不会影响实参对象。

26.5 对象、对象指针或对象引用作为函数的参数和返回值

- 函数返回值可以是对象、对象指针或对象引用。

- 函数返回对象时，将其内存单元的所有内容复制到一个临时对象中。因此函数返回对象时会增加调用开销。

- 函数返回对象指针或引用，本质上返回的是对象的地址而不是它的存储内容，因此不要返回局部对象的指针或引用，因为它在函数返回后是无效的。

例：

```
Data  Try1()
{    Data  a;  a.fun(1,2,3);
     return  a;  //可以返回局部对象，它被复制返回
}
Data*  Try2(Data  *p1,Data  *p2)
{          if  (p1->data  >  p2->data)  return  p1;
          return  p2;
}
Data&  Try3(Data  &r1,Data  &r2)
{          if  (r1.data  >  r2.data)  return  r1;
          return  r2;
}
```

```
void   caller()
{
      Data   A,   B,   C;
      A.fun(1,2,3);
      B.fun(4,5,6);
      C=Try1();
      Try2(&A,&B)->data=100;
      //等价于 (&B)->data=100;
      Try3(A,B).data=100;
      //等价于B.data=100;
}
```

26.5 对象、对象指针或对象引用作为函数的参数和返回值

例：

```
1     Data*  function()
2     {
3           Data  a;
4           Data  *p=&a;
5           return  p;   //返回局部对象a的指针p
6     }
7     void  caller()
8     {
9           Data  *p1;
10          p1=function();
11          p1->data=100;  //a已不存在，引用错误
12    }
```

```
class  Data  {  //Data类
   public:
      int  data;
      void  fun(int  a,int  b,int  d);
   private:
     void  add(int  m)  { data+=m; }
      int  x,y;
      ……
}
```