面向对象程序设计

运算符重载

2020 年 春

耿楠

计算机科学系
信息工程学院

西北农林科技大学
NORTHWEST A&F UNIVERSITY
中国 · 杨凌

- 使用一致接口 (uniform interface) 处理不同数据
  - 函数重载
  - 运算符重载（“+”）

```
3.14 + 0.0015 = 3.1415;
[1, 2, 3] + [4, 5, 6] = [1, 2, 3, 4, 5, 6];
[3 + 4i] + [1 + 5i] = [4 + 9i];
"coffee" + " tea" = "coffee tea";
```

  - 虚函数（继承与派生）

- 函数重载
  - 功能相同、数据不同 (类型或个数) 的同名函数
- 运算符重载
  - 同一个运算符作用于不同类型数据的操作
  - 运算符 $\Rightarrow$ 函数名

- 运算符重载是 C++ 的一大亮点
  - 函数调用更简洁、直观
  - 增加程序可读性 (readability)
- 实际应用
  - 复数操作：`(a+bi)+(c+di) = (a+c) + (b+d)i`
  - 字符串操作：`string1 == string2`
  - 向量操作：`(a1, ···, an)*s = (a1*s, ···, an*s)`
  - 矩阵乘法：`M = N*K`
  - ···

- 位置信息：纬度 (latitude) 和经度 (longitude)
- 经纬度决定地球上一个地点的精确位置

已知：

北京天安门：（39.907306, 116.391264）

求：

西南方向偏移（-5.642159, -8.323558）后的新位置？

解：

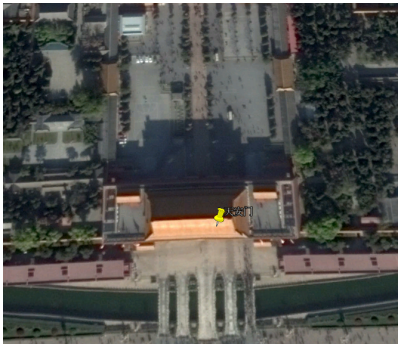
```
39.907306 + (- 5.642159)
```

```
116.391264 + (- 8.323558)
```

答案：

（34.265147 108.067706）⇒ 西北农林科技大学行政楼位置

- 位置信息：纬度 (latitude) 和经度 (longitude)
- 经纬度决定地球上一个地点的精确位置



⇒

- 位置信息：纬度 (latitude) 和经度 (longitude)
- 经纬度决定地球上一个地点的精确位置

```cpp
// 例 04-01-01: ex04-01-01.cpp
// 重载 + 操作符

#include <iostream>
#include <iomanip>
using namespace std;
class CLocation{
   double latitude, longitude;
public:
   CLocation(){}
   CLocation(double lt, double lg){
      latitude = lt;
      longitude = lg;
   }
   void show(){
      cout << setprecision(9) << latitude << " ";
      cout << longitude << endl;
   }
   CLocation operator+(CLocation op2);
};
CLocation CLocation::operator+(CLocation op2){
   CLocation temp;
   temp.latitude = op2.latitude + latitude;
   temp.longitude = op2.longitude + longitude;
   return temp;
}
```

+运算符的重载

```cpp
// 例 04-01-02: ex04-01-02.cpp
// 重载操作符的测试

int main()
{
   CLocation Tiananmen(39.907306, 116.391264);
   CLocation Offset(-5.642159, -8.323558);
   CLocation Yangling;

   Tiananmen.show();

   Yangling = Tiananmen + Offset;

   Yangling.show();
   return 0;
}
```

北京天安门:
(39.907306, 116.391264)

西南方向偏移:
(-5.642159, -8.323558)

西北农林科技大学
行政楼位置

- 可重载的运算符

+	-	*	/	%	^	&	|	~
!	,	=	<	>	<=	>=	++	--
<<	>>	==	!=	&&	||	+=	-=	/=
%=	^=	&=	|=	*=	<<=	>>=	[]	()
->	->*	new	new []	delete	delete []			

- 单目运算符和双目运算符
- 不可重载的运算符

.	.*	::	?:

重载规则

- 重载后运算符的优先级和结合性不变
- 运算符操作数的个数不能改变
- 不能重载 C++ 中不支持的运算符 (@、#、$ 等)
- 保持运算符的语义

- 重载为类的成员函数
  - 定义

```
返回类型 [类名::]operator 运算符 (形参表) {}
CLocation operator+(CLocation op2);
```

- 重载为类的非成员函数 (一般为友员函数)
  - 定义

```
friend 返回类型 operator 运算符 (形参表) {}
```

- 语法：返回类型 [类名::]operator 运算符（形参表）{}
  - 返回类型一般是一个对象
  - 可以省略一个形参

```
CLocation operator+(CLocation op2);
CLocation operator-(CLocation op2);
CLocation operator=(CLocation op2);
CLocation operator++();
```

注意：用成员函数实现 + 运算符重载，只有一个参数！

```
CLocation loc1(30,100), loc2(5,4);
loc1 = loc1 + loc2;
```

重载函数定义

```
// 例 04-02-01: ex04-02-01.cpp
// 重载 - , = , ++ 操作符

CLocation CLocation::operator-(CLocation op2){
   CLocation temp;
   temp.latitude = latitude - op2.latitude;
   temp.longitude = longitude - op2.longitude;
   return temp;
}
CLocation CLocation::operator=(CLocation op2){
   longitude = op2.longitude;
   latitude = op2.latitude;
   return *this;
}
CLocation CLocation::operator++(){
   longitude++;
   latitude++;
   return *this;
}
```

```
// 例 04-02-02: ex04-02-02.cpp
// 重载操作符的测试

int main(){
   CLocation ob1(39.907306,
                 116.391264);
   CLocation ob2, ob3;
   ob1.show();
   ob2 = ++ob1;
   ob2.show();
   // multiple assignment
   ob3 = ob2 = ob1;
   ob1 = ob1 - ob2;
   ob1.show();
   ob2.show();
   ob3.show();
   return 0;
}
```

利用*this指针返回运算后的对象

- 前缀运算符：++i, --i

```
CLocation operator++();
CLocation operator--();
```

- 后缀运算符：i++, i--

```
CLocation operator++(int x);
CLocation operator--(int x);
```

哑元参数，仅表示重载后缀运算符

前缀与后缀

```
// 例 04-03-01: ex04-03-01.cpp
// + 及 ++ 作为成员函数的重载

// Overload prefix ++ for CLocation.
CLocation CLocation::operator++()
{
   longitude++;
   latitude++;
   return *this;
}

// Overload postfix ++ for CLocation.
CLocation CLocation::operator++(int x)
{
   longitude++;
   latitude++;
   return CLocation(latitude - 1, longitude - 1);
}
```

返回运算后的对象

```
// 例 04-03-02: ex04-03-02.cpp
// 重载操作符的测试

int main()
{
   CLocation ob1(39.907306,
                 116.391264);
   CLocation ob2, ob3;

   ob2 = ++ob1;
   ob3 = ob2++;

   ob1.show();
   ob2.show();
   ob3.show();
}
```

- 语法：**friend** 返回类型 **operator** 运算符（形参表）{}
  - 友元函数没有 this 指针，需给出所有传递参数

```
friend CLocation operator+(CLocation op1,CLocation op2);
```

  - 若使用单目运算符且修改成员数据，需使用引用传递

```
friend CLocation operator++(CLocation &op1);
friend CLocation operator++(CLocation &op1, int x);
```

哑元参数表示后缀

重载为类的友元函数

```
// 例 04-04: ex04-04.cpp
// 操作符 + , ++ 的重载

CLocation operator+(CLocation op1, CLocation op2)
{
   CLocation temp;
   temp.longitude = op1.longitude + op2.longitude;
   temp.latitude = op1.latitude + op2.latitude;
   return temp;
}
CLocation operator++(CLocation &op1)
{
   op1.longitude++;
   op1.latitude++;
   return op1;
}
CLocation operator++(CLocation &op1, int x)
{
   return CLocation(op1.latitude++, op1.longitude++);
}
```

- 操作符左右为不同类型数据
  - CLocation loc1;

```
loc1 = loc1 * 1.01; ···(1)
loc1 = 1.01 * loc1; ···(2)
```

  - 如何实现?
    - 重载为类的成员函数无法实现!

- 操作符左右为不同类型数据

```
// 例 04-05-01: ex04-05-01.cpp
// 重载 * 操作符

friend CLocation operator*(CLocation op1,
                           double s);
friend CLocation operator*(double s,
                           CLocation op1);
CLocation operator*(CLocation op1, double s)
{
   CLocation temp;
   temp.longitude = s * op1.longitude;
   temp.latitude = s * op1.latitude;
   return temp;
}
CLocation operator*(double s, CLocation op1)
{
   CLocation temp;
   temp.longitude = s * op1.longitude;
   temp.latitude = s * op1.latitude;
   return temp;
}
```

```
// 例 04-05-02: ex04-05-02.cpp
// 重载 * 操作符的测试

int main()
{
   CLocation ob1(39.907306,
          116.391264);
   CLocation ob2, ob3;

   ob2 = ob1 * 1.01;
   ob3 = -1.01 * ob1;

   ob1.show();
   ob2.show();
   ob3.show();
}
```

- 一般单目运算符重载为类的成员函数，双目运算符重载为类的友元函数
- =，()，[]，->双目运算符不能重载为类的友元函数
- “>>”和“<<”只能重载为类的友元函数

- “>>” 和 “<<”

只能以友元函数方式重载

- “=”
- “[]”
- “()”
- “->”

只能以成员函数方式重载

- “new”、“delete”、“new[]” 和 “delete []”
- “,”

- 重载运算符 “>>” 和 “<<”

```cpp
// 例 04-06-01: ex04-06-01.cpp
// 重载输入输出操作符

class CLocation
{
   double latitude, longitude;
public:
   CLocation (double lt = 0, double lg = 0)
   {
      latitude = lt;
      longitude = lg;
   }
   friend ostream& operator<<( ostream& out , CLocation loc);
   friend istream& operator>>( istream& in , CLocation  & loc);
};
```

- 重载运算符 “>>” 和 “<<”

```cpp
// 例 04-06-02: ex04-06-02.cpp
// 重载输入输出操作符的实现及测试

ostream &operator<<(ostream &out, CLocation loc){
   out << loc.latitude << " " << loc.longitude << endl;
   return out;
}
istream& operator>>(istream& in, CLocation& loc){
   in >> loc.latitude >> loc.longitude;
   if(loc.latitude < -90  loc.latitude > 90)
      cout << "Error latitude input!" << endl;
   if(loc.longitude < -180  loc.longitude > 180)
      cout << "Error longitude input!" << endl;
   return in;
}
int main(){
   CLocation loc;
   cin >> loc ;
   cout << loc ;
   return 0;
}
```

- 重载运算符 “=”

```
ch_stack operator=(ch_stack & sObj)
{
  size = sObj.size;
  tp = sObj.tp;
  s = sObj.s;
}
```

浅拷贝

```
ch_stack operator=(ch_stack & sObj)
{
  if(s != NULL)
     free(s);
  size = sObj.size;
  tp = sObj.tp;
  s = NULL;
  if (size != 0)
  {
    s = new char[size];
    for (int i = 0; i <= tp; i++)
       s[i] = sObj.s[i];
  }
}
```

深拷贝

- 自赋值检测

```
ClassName &ClassName::operator=(ClassName &s)
{
  if (this == &s)
     return *this;

  ...
}
```

- 引用计数与浅拷贝
  - 深拷贝安全但某些情况下易浪费空间。
  - 缺点：增加复杂度。

- 重载运算符 “[]”
  - 防止数组越界

```
// 例 04-07-01: ex04-07-01.cpp
// 重载 [] 操作符

class atype{
   int a[3];
public:
   atype(int i, int j, int k){
      a[0] = i;
      a[1] = j;
      a[2] = k;
   }
   int &operator[](int i){
      if(i < 0  i > 2)
      {
         cout << "Boundary Error\n";
         exit(1);
      }
      return a[i];
   }
};
```

```
// 例 04-07-02: ex04-07-02.cpp
// 重载 [] 操作符的测试

int main()
{
   atype ob(1, 2, 3);
   cout << ob[1];
   cout << " ";
   ob[1] = 25;
   cout << ob[1];
   ob[3] = 44;
}
```

- 重载运算符 “()”
  - 自动执行和表达式中的使用
  - 函数对象

```
// 例 04-08-01: ex04-08-01.cpp
// + , () 运算符的重载

class loc
{
   double longitude, latitude;
public:
   ...
   loc operator+(loc op2);
   loc operator()(double i, double j)
   {
      longitude = j;
      latitude = i;
      return *this;
   }
};
```

```
// 例 04-08-02: ex04-08-02.cpp
// 运算符重载的测试

int main()
{
   loc ob1(10, 20), ob2(1, 1);
   ob1.show();
   ob1(7, 8);
   ob1.show();
   ob1 = ob2 + ob1(10, 10);
   ob1.show();
   return 0;
}
```

- 重载运算符 “->”(应用于安全指针)

```
ClassName* ClassName::operator->();
```

- 重载运算符 “new” 和 “delete”
  - 自定义的内存分配与释放（如在堆区无内存可分配的情况下自动使用磁盘空间）

```
void *ClassName::operator new(size_t size);
void ClassName::operator delete(void *p);
```

  - 使用系统的 “new” 和 “delete”

```
::new
::delete
```

- 重载运算符 “new[]” 和 “delete[]”
  - 为数组动态分配内存空间

```
void *ClassName::operator new[](size_t size);
void ClassName::operator delete[](void *p);
```

- 重载运算符 “,”

```
ClassName ClassName::operator,(ClassName obj);
```

- 类型转换运算符
  - 重载 “int”、“float”、· · · (不必指定返回类型)

```
Rational r = 3.5f;
class Rational
{
  int up;
  int down;
public:
  //成员函数 return
  operator float()
  {
    up/(float)down;
  }
};
```

- 目的
  - 将表达式传入函数 (函数指针)
- 本质
  - 可调用的代码单元
  - 类似于未命名的**inline**函数
- 实例

```cpp
#include <iostream>

using namespace std;

int main()
{
  int girls = 3, boys = 4;
  // lambda 函数
  auto totalChild = [](int x, int y)->int {return x + y;};
  cout << totalChild(girls, boys) << endl;

  return 0;
}
```

- 别称
  - **lambda** 表达式

- 完整定义

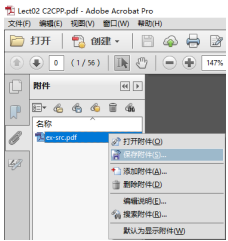
```
[capture list](params list) mutable exception -> return type{function body}
```

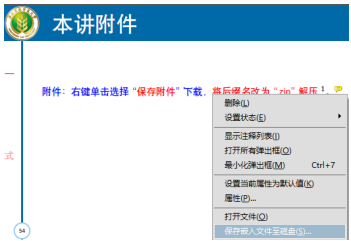
capture list	捕获外部变量列表（上下文中的变量）
params list	形参列表
mutable修饰符	用于说明可否修改外部捕获的外部变量
exception	异常设定
return type	返回类型
function body	函数体

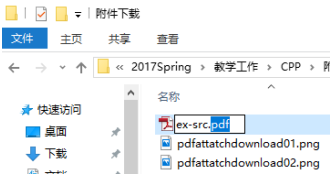
- 简化定义
  - [capture list](params list)->return type{function body}
  - [capture list](params list){function body}
  - [capture list]{function body}

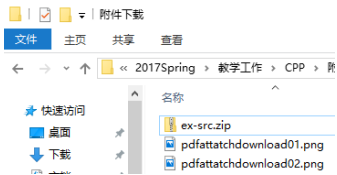
- 对于任何参数类型，如果仅仅只是读参数的值，应该作为const引用来传递。普通算术运算符、关系运算符、逻辑运算符都不会改变参数，应该以const &引用作为主要的参数传递方式。
- 当运算符函数是类的成员函数时，应该将其定义为const成员函数。
- 返回值的类型取决于运算符的具体含义。如果使用运算符的结果产生一个新值，就需要产生一个作为返回值的新对象，这个对象作为一个常量通过传值方式返回。
- 如果函数返回的是原有对象，则通常以引用方式返回，根据是否希望对返回的值进行运算来决定是否返回const引用。
- 所有赋值运算符均改变左值。为了使赋值结果能用于链式表达式，如a=b=c，应该返回一个改变了的左值的引用。一般赋值运算符的返回值是非const引用，以便能够对刚刚赋值的对象进行运算。

附件：右键单击该链接，选择“保存附件”下载，将后缀名改为“.zip”解压[1][2]。









[1]请退出全屏模式后点击该链接。

[2]以 Adobe Acrobat Reader 为例。

本讲结束，谢谢!
欢迎多提宝贵意见和建议

西北农林科技大学
NORTHWEST A&F UNIVERSITY
中国 · 杨凌