In The Name of Allah

سربارگذاری عملگرها در زبان C++

فهرست مطالب

- مباحث
  - توابع دوست
  - سربارگذاري عملگر جايگزيني (=)
  - اشاره‌گر this
  - سربارگذاري عملگرهاي حسابي
  - سربارگذاري عملگرهاي جايگزيني حسابي
  - سربارگذاري عملگرهاي رابطه‌اي
  - سربارگذاري عملگرهاي افزايشي و كاهشي

«سربارگذاری عملگرها»

مقدمه

هنگامی که کلاسی را تعریف می‌کنیم، در حقیقت یک نوع جدید را به انواع موجود اضافه کرده‌ایم. ممکن است بخواهیم اشیای این کلاس را در محاسبات ریاضی به کار ببریم.

اما چون عملگرهای ریاضی (مثل + یا = یا =*) چیزی راجع به اشیای کلاس جدید نمی‌دانند، نمی‌توانند به درستی کار کنند. C++ برای رفع این مشکل چاره اندیشیده و امکان *سربارگذاری عملگرها* را تدارک دیده است. **سربارگذاری عملگرها** به این معناست که به عملگرها تعاریف جدیدی اضافه کنیم تا بتوانند با اشیای کلاس مورد نظر به درستی کار کنند.

تابع دوست

- اعضایی از کلاس که به شکل خصوصی (private) اعلان می‌شوند فقط از داخل همان کلاس قابل دستیابی‌اند و از بیرون کلاس (درون بدنهٔ اصلی) امکان دسترسی به آن‌ها نیست.
- اما یک استثنا وجود دارد. *تابع دوست* تابعی است که عضو یک کلاس نیست اما اجازه دارد به اعضای خصوصی آن دسترسی داشته باشد.

به کد زیر نگاه کنید:

```
class Ratio
{  friend int numReturn(Ratio);
   public:
      Ratio();
      ~Ratio();
   private:
      int num, den;
}
int numReturn(Ratio r)
{  return r.num;
}
int main()
{  Ratio x(22, 7);1 – Friend function
  cout << numReturn(x) << endl;
}
```

سربارگذاری عملگر جایگزینی(=):

در بین عملگرهای گوناگون، عملگر جایگزینی شاید بیشترین کاربرد را داشته باشد.

هدف این عملگر، کپی کردن یک شی در شیء دیگر است.

مانند سازندۀ پیش‌فرض، سازندۀ کپی و نابودکننده، عملگر جایگزینی نیز به طور خودکار برای یک کلاس ایجاد می‌شود اما این تابع را می‌توانیم به شکل صریح درون کلاس اعلان نماییم.

افزودن عملگر جایگزینی به کلاس:

کد زیر یک رابط کلاس برای Ratio است که شامل سازندهٔ پیش‌فرض، سازندهٔ کپی و عملگر جایگزینی می‌باشد:

```
class Ratio
{  public:
     Ratio(int = 0, int = 1);
     Ratio(const Ratio&);
     void operator=(const Ratio&);
   private:
     int num, den;
};
```

به نحو اعلان عملگر جایگزینی دقت نمایید.

نام این تابع عضو، operator= است و فهرست آرگومان آن مانند سازندهٔ کپی می‌باشد یعنی یک آرگومان منفرد دارد که از نوع همان کلاس است که به طریقهٔ ارجاع ثابت ارسال می‌شود. عملگر جایگزینی را می‌توانیم به شکل زیر تعریف کنیم:

```
void Ratio::operator=(const Ratio& r)
{  num = r.num;
   den = r.den;
}
```

کد فوق اعضای داده‌ای شیء r را به درون اعضای داده‌ای شیئی که مالک فراخوانی این عملگر است، کپی می‌کند.

اشاره‌گر this:

در C++ می‌توانیم عملگر جایگزینی را به شکل زنجیره‌ای مثل زیر به کار ببریم:

```
x = y = z = 3.14;
```

اجرای کد بالا از راست به چپ صورت می‌گیرد. یعنی ابتدا مقدار 3.14 درون z قرار می‌گیرد و سپس مقدار z درون y کپی می‌شود و سرانجام مقدار y درون x قرار داده می‌شود. عملگر جایگزینی که در مثال قبل ذکر شد، نمی‌تواند به شکل زنجیره‌ای به کار رود.

سربارگذاری عملگر جایگزینی به شکل صحیح:

```
class Ratio
{  public:
      Ratio(int =0, int =1);          // default constructor
      Ratio(const Ratio&);            // copy constructor
      Ratio& operator=(const Ratio&);  // assignment operator
      // other declarations go here
   private:
      int num, den;
      // other declarations go here
};
Ratio& Ratio::operator=(const Ratio& r)
{  num = r.num;
   den = r.den;
   return *this;
}
```

توجه داشته باشید که عمل جایگزینی با عمل مقداردهی تفاوت دارد، هر چند هر دو از عملگر یکسانی استفاده می‌کنند. مثلا در کد زیر:

```
Ratio x(22,7);     // this is an initialization
Ratio y(x);        // this is an initialization
Ratio z = x;       // this is an initialization
Ratio w;
w = x;             // this is an assignment
```

سه دستور اول، دستورات مقداردهی هستند ولی دستور آخر یک دستور جایگزینی است.

دستور مقداردهی، سازندهٔ کپی را فرا می‌خواند ولی دستور جایگزینی عملگر جایگزینی را فراخوانی می‌کند.

سربارگذاری عملگرهای حسابی:

چهار عملگر حسابی + و – و * و / در همهٔ زبان‌های برنامه‌نویسی وجود دارند و با همهٔ انواع بنیادی به کار گرفته می‌شوند. قصد داریم سرباری را به این عملگرها اضافه کنیم تا بتوانیم با استفاده از آن‌ها، اشیای ساخت خودمان را در محاسبات ریاضی به کار ببریم.

عملگرهای حسابی به دو عملوند نیاز دارند. مثلا عملگر ضرب (*) در رابطهٔ زیر:

```
z = x*y;
```

با توجه به رابطهٔ فوق و آنچه در بخش قبلی گفتیم، عملگر ضرب سربارگذاری شده باید دو پارامتر از نوع یک کلاس و به طریق ارجاع ثابت بگیرد و یک مقدار بازگشتی از نوع همان کلاس داشته باشد. پس انتظار داریم قالب سربارگذاری عملگر ضرب برای کلاس Ratio به شکل زیر باشد:

```
Ratio operator*(Ratio x, Ratio y)
{  Ratio z(x.num*y.num, x.den*y.den);
   return z;
}
```

اگر تابعی عضو کلاس نباشد، نمی‌تواند به اعضای خصوصی آن کلاس دستیابد. برای رفع این محدودیت‌ها، تابع سربارگذاری عملگر ضرب را باید به عنوان تابع دوست کلاس معرفی کنیم. لذا قالب کلی برای سربارگذاری عملگر ضرب درون کلاس مفروض T به شکل زیر است:

```
Class T
{  friend T operator*(const T&, const T&);
   public:
      // public members
   private:
      // private members
}
```

و از آن‌جا که تابع دوست عضوی از کلاس نیست، تعریف بدنهٔ آن باید خارج از کلاس صورت پذیرد. در تعریف بدنهٔ تابع دوست به کلمهٔ کلیدی **friend** نیازی نیست و عملگر جداسازی حوزه :: نیز استفاده نمی‌شود:

```
T operator*(const T& x, const T& y)
{  T z;
   // required operations for z = x*y
```

در سربارگذاری عملگرهای حسابی + و – و / نیز از قالب‌های کلی فوق استفاده می‌کنیم با این تفاوت که در نام تابع سربارگذاری، به جای علامت ضرب * باید علامت عملگر مربوطه را قرار دهیم و دستورات بدنهٔ تابع را نیز طبق نیاز تغییر دهیم.

سربارگذاری عملگر ضرب برای کلاس Ratio:

```
class Ratio
{     friend Ratio operator*(const Ratio&, const Ratio&);
   public:
      Ratio(int = 0, int = 1);
      Ratio(const Ratio&);
      Ratio& operator=(const Ratio&);
      // other declarations go here
   private:
      int num, den;
      // other declarations go here
};
Ratio operator*(const Ratio& x, const Ratio& y)
{  Ratio z(x.num * y.num , x.den * y.den);
   return z;
}
int main()
{  Ratio x(22,7) ,y(-3,8) ,z;
   z = x;               // assignment operator is called
   z.print();  cout << endl;
   x = y*z;             // multiplication operator is called
   x.print();  cout << endl;
}
```

سربارگذاری عملگرهای جایگزینی حسابی:

به خاطر بیاورید که عملگرهای جایگزینی حسابی، ترکیبی از عملگر جایگزینی و یک عملگر حسابی دیگر است. مثلا عملگر =* ترکیبی از دو عمل ضرب * و سپس جایگزینی = است. نکتهٔ قابل توجه در عملگرهای جایگزینی حسابی این است که این عملگرها بر خلاف عملگرهای حسابی ساده، فقط یک عملوند دارند. پس تابع سربارگذاری عملگرهای جایگزینی حسابی بر خلاف عملگرهای حسابی، می‌تواند عضو کلاس باشد. سربارگذاری عملگرهای جایگزینی حسابی بسیار شبیه سربارگذاری عملگر جایگزینی است. قالب کلی برای سربارگذاری عملگر =* برای کلاس مفروض T به صورت زیر است:

```
class T
{  public:
     T& operator*=(const T&);
     // other public members
   private:
     // private members
};
```

بدنهٔ تابع سربارگذاری به قالب زیر است:

```
T& T::operator*=(const T& x)
{  // required operations
   return *this;
}
```

استفاده از اشاره‌گر *this باعث می‌شود که بتوانیم عملگر *= را در یک رابطهٔ زنجیره‌ای به کار ببریم. در C++ چهار عملگر جایگزینی حسابی += و -= و *= و /= وجود دارد. قالب کلی برای سربارگذاری همهٔ این عملگرها به شکل قالب بالا است فقط در نام تابع به جای *= باید علامت عملگر مربوطه را ذکر کرد و دستورات بدنهٔ تابع را نیز به تناسب، تغییر داد. مثال بعدی نشان می‌دهد که عملگر *= چگونه برای کلاس Ratio سربارگذاری شده است.

کلاس Ratio با عملگر =* سربارگذاری شده:

```
class Ratio
{  public:
      Ratio(int = 0, int = 1);
      Ratio& operator=(const Ratio&);
      Ratio& operator*=(const Ratio&);
      // other declarations go here
   private:
      int num, den;
      // other declarations go here
};
Ratio& Ratio::operator*=(const Ratio& r)
{  num = num*r.num;
   den = den*r.den;
   return *this;
}
```

سربارگذاری عملگرهای رابطه‌ای:

شش عملگر رابطه‌ای در C++ وجود دارد که عبارتند از: < و > و <= و => و == و =! .

این عملگرها به همان روش عملگرهای حسابی،یعنی به شکل توابع دوست سربارگذاری می‌شوند. اما نوع بازگشتی‌شان فرق می‌کند.

حاصل عبارتی که شامل عملگر رابطه‌ای باشد، همواره یک مقدار بولین است. یعنی اگر آن عبارت درست باشد، حاصل true است و اگر آن عبارت نادرست باشد، حاصل false است.

چون نوع بولین در حقیقت یک نوع عددی صحیح است، می‌توان به جای true مقدار 1 و به جای false مقدار 0 را قرار داد. به همین جهت نوع بازگشتی ر برای توابع سربارگذاری عملگرهای رابطه‌ای، از نوع int قرار داده‌اند.

قالب کلی برای سربارگذاری عملگر رابطه‌ای == به شکل زیر است:

```
class T
{  friend int operator==(const T&, const T&);
   public:
      // public members
   private:
      // private members
}
```

همچنین قالب کلی تعریف بدنهٔ این تابع به صورت زیر می‌باشد:

```
int operator==(const T& x,const T& y)
{  // required operations to finding result
   return result;
}
```

که به جای result یک مقدار بولین یا یک عدد صحیح قرار می‌گیرد. سایر عملگرهای رابطه‌ای نیز از قالب بالا پیروی می‌کنند.

```
class Ratio
{    friend int operator==(const Ratio&, c
     frined Ratio operator*(const Ratio&, c
     // other declarations go here
  public:
     Ratio(int = 0, int = 1);
     Ratio(const Ratio&);
     Ratio& operator=(const Ratio&);
     // other declarations go here
  private:
     int num, den;
     // other declarations go here
};
int operator==(const Ratio& x, const Ratio& y)
{  return (x.num * y.den == y.num * x.den);
}
```

چون اشیای کلاس atio ه صورت کسر $\frac{a}{b}$ هستند، بررسی تساوی $\frac{a}{b}$ و $\frac{c}{d}$ بررسی x==y معادل بررسی $a*d==b*c$ است که برای بررسی این تساوی می‌توانیم مقدار را بررسی کنیم. بدنهٔ تابع سربارگذاری در مثال همین رابطه را بررسی می‌کند.

سربارگذاری عملگرهای افزایشی و کاهشی:

عملگر افزایشی ++ و کاهشی -- هر کدام دو شکل دارند:

۱- شکل پیشوندی.

۲-شکل پسوندی.

هر کدام از این حالت‌ها را می‌توان سربارگذاری کرد.

قالب کلی برای سربارگذاری عملگر پیش‌افزایشی به شکل زیر است:

```
T T::operator++()
{  // required operations
   return *this;
}
```

این‌جا هم از اشاره‌گر *this استفاده شده. علت هم این است که مشخص نیست چه چیزی باید بازگشت داده شود. به همین دلیل اشاره‌گر *this به کار رفته تا شیئی که عمل پیش‌افزایش روی آن صورت گرفته، بازگشت داده شود.

اگر y یک شی از کلاس Ratio باشد و عبارت ++y ارزیابی گردد، مقدار 1 به y افزوده می‌شود اما چون y یک عدد کسری است، افزودن مقدار 1 به این کسر اثر متفاوتی دارد. فرض کنید y=22/7 باشد. حالا داریم:

$$++y = \frac{22}{7} + 1 = \frac{22+7}{7} = \frac{29}{7}$$

در عبارت y = x++; از عملگر پس‌افزایشی استفاده کرده‌ایم. تابع این عملگر باید طوری تعریف شود که مقدار x را قبل از این که درون y قرار بگیرد، تغییر ندهد. می‌دانیم که اشاره‌گر this* به شیء جاری (مالک فراخوانی) اشاره دارد. کافی است مقدار این اشاره‌گر را در یک محل موقتی ذخیره کنیم و عمل افزایش را روی آن مقدار موقتی انجام داده و حاصل آن را بازگشت دهیم. به این ترتیب مقدار this* تغییری نمی‌کند و پس از شرکت در عمل جایگزینی، درون y قرار می‌گیرد:

```
class Ratio
{  public:
      Ratio(int n=0, int d=1) : num(n) , den(d) { }
      Ratio operator++();          //pre-increment
      Ratio operator++(int);       //post-increment
      void print() {  cout << num << '/' << den << endl; }
   private:
      int num, den;
};
```

عملگرهای پیش‌کاهشی و پس‌کاهشی نیز به همین شیوهٔ عملگرهای پیش‌افزایشی و پس‌افزایشی سربارگذاری می‌شوند. غیر از این‌ها، عملگرهای دیگری نیز مثل عملگر خروجی (<<) ، عملگر ورودی (>>) ، عملگر اندیس ([]) و عملگر تبدیل نیز وجود دارند که می‌توان آن‌ها را برای سازگاری برای کلاس‌های جدید سربارگذاری کرد.

```
int main()
{  Ratio x(22,7) , y = x++;
   cout << "y = ";  y.print();
   cout << ", x = ";  x.print();}
Ratio Ratio::operator++(int)
{  Ratio temp = *this;
   num += den;
   return temp;
}
```