Bölüm 18 - Operatör Çoklu-Yükleme

İçerik

18.1 Giriş

- Bölüm 16 ve 17 de
  - ADT ler ve Sınıflar (**`classes`**)
  - Fonksiyon çağırma notasyonu belli sınıflar için oldukça problemli (özellikle matematiksel sınıflar için)

- Bu bölümde
  - Sınıf nesneleri ile çalışmak için C++ ın hazır operatörlerini kullanacağız

- Operatör yükleme
  - Kullanıcı tanımlı nesnelerle geleneksel operatörleri kullan
  - C++ a geçiş için doğrudan ve kolay yol
  - Çok dikkat gerektirir
    - Yükleme yanlış kullanıldığında programı anlamak güçtür

18.2 Operatör Yüklemenin Temelleri

- Okunabilirliği artırmak için çoklu-yükleme kullan
  - Çok fazla ve yanlış kullanımdan kaçın
- Format
  - Normal şekilde fonksiyon tanımlamasını yap
  - Fonksiyon adı anahtar **`operator`** olup yüklenecek olan operatörün sembolü onu takip eder
  - Toplama (**+**) operatörünü çoklu yüklemek için **`operator+`** kullanılabilir
- Atama operatörü (**=**)
  - Fazla olmamak kaydı ile her sınıf için kullanılabilir
  - *Üyebazlı atama*
  - Aynı şey adres operatörü (&) için de geçerli

18.3 Kısıtlamalar

- C++ daki çoğu operatör çoklu-yüklenebilir

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

- Operandların sayısı değiştirilemez
  - Unary operatörler unary, ve binary operatörler binary kalır
  - **&**, *****, **+** ve **–** operatörlerinin her birinin unary ve binary versiyonları vardır
    - Unary ve binary versiyonlar ayrı ayrı yüklenebilir

- Yeni operatör oluşturulamaz
  - Sadece varolan operatörleri kullan
- Hazır (built-in) tipler
  - Operatörleri çoklu-yükleyemezler
  - İki tamsayının toplam tanımını değiştiremezsiniz

18.4 Class Üyeleri Olarak Operatör Fonksiyonları - `friend` Fonksiyonları

- Operatör fonksiyonları
  - Üye yada üye-olmayan fonksiyonlar olabilir
- Atama operatörlerinin çoklu-yüklemesi
  - yani: `()`, `[]`, `->`, `=`
  - Operatör bir üye fonksiyon olmalıdır
- Üye fonksiyon olarak operatör fonksiyonu
  - En soldaki operand sınıfın bir nesnesi (veya nesneye referans) olmalı
  - Eğer sol operand farklı bir tipten ise, operatör fonksiyonu üye-olmayan bir fonksiyon olmalıdır
  - Eğer o sınıfın **`private`** veya **`protected`** üyelerine doğrudan erişim sağlanıyorsa, üye olmayan operatör fonksiyonu **`friend`** olmalıdır
- Üye-olmayan çoklu-yüklemeli operatör fonksiyonları
  - Operatörün değişimli olmasına izin verir

```
DevTamsayi buyukTamsayi;
int tamsayi;
buyukTamsayi = tamsayi+ buyukTamsayi;
```

veya

```
buyukTamsayi = buyukTamsayi + tamsayi;
```

18.5 Akış-ekleme ve Akış-çıkarma Fonksiyonlarının Yüklenmesi

- Çoklu-yüklemeli **<<** ve **>>** operatörleri
  - Sol operandları sırası ile **`ostream &`**, ve **`istream &`** tipinde olmalıdır
  - Üye-olmayan fonksiyon olmalıdır (sol operand sınıfın bir nesnesi değil)
  - Eğer private veri üyelerine erişebiliyorsa, bir **`friend`** fonksiyonu olmak zorundadır

18.5 Akış-ekleme ve Akış-çıkarma Fonksiyonlarının Yüklenmesi

```
1  // Fig. 18.3: fig18_03.cpp
2  // Akış-ekleme ve akış-çıkarma
3  // operatörleri çoklu-yüklemesi.
4  #include <iostream>
5
6  using std::cout;
7  using std::cin;
8  using std::endl;
9  using std::ostream;
10 using std::istream;
11
12 #include <iomanip>
13
14 using std::setw;
15
16 class TelNo {
17    friend ostream &operator<<( ostream&, const TelNo & );
18    friend istream &operator>>( istream&, TelNo & );
19
20 private:
21    char alanKodu[ 4 ];  // 3-digit alan kodu ve boşluk
22    char bolge[ 4 ];  // 3-digit bölge ve boşluk
23    char hat[ 5 ];       // 4-digit hat ve boşluk
24 };
25
26 // Çoklu ekleme operatörü ( cout << baziTelNo; ile kulllanmak
27 // istiyorsak
28 // üye fonksiyon olamaz).
29 ostream &operator<<( ostream &output, const TelNo &num )
30 {
```

```
31    output << "(" << num.alanKodu << ") "
32           << num.bolge << "-" << num.hat;
33    return output;      // cout << a << b << c; ye izin verir
34 }
35
36 istream &operator>>( istream &input, TelNo &num )
37 {
38    input.atla();                      // ( -i atla
39    input >> setw( 4 ) >> num.alanKodu; // alan kodu girdisi
40    input.atla( 2 );                   // ) -i atla ve bosluk
41    input >> setw( 4 ) >> num.bolge; // bolge girdisi
42    input.atla();                      // dash (-) yi atla
43    input >> setw( 5 ) >> num.hat;      // hattı gir
44    return input;       // cin >> a >> b >> c ; ye izin verir
45 }
46
47 int main()
48 {
49    TelNo tel; // tel nesnesi olustur
50
51    cout << "Telefon numarasını gir. (123) 456-7890 gibi:\n";
52
53    // cin >> tel, >> operatör fonksiyonunu çağırır
54    // çağırma operatörü operator>>( cin, tel ).
55    cin >> tel;
56
57    // cout << tel, << operatör fonksiyonunu çağırır
58    // çağırma operatörü operator<<( cout, tel ).
59    cout << "Girilen tel no: " << tel << endl;
60    return 0;
61 }
```

```
Telefon numarasını gir (123) 456-7890 gibi:
(800) 555-1212
Girilen tel no: (800) 555-1212
```

18.6 Unary Operatörler

- unary operatörleri çoklu-yükleme
  - Kesinlikle gerekli değilse **`friend`** fonksiyonlarından ve **`friend`** sınıflarından kaçının.
  - **`friend`** kullanımı sınıfın içerilmesini bozar.
  - Bir üye fonksiyon olarak:

```
class String {
  public:
     bool operator!() const;
     ...
  };
```

18.7 Binary Operatörler

- binary operatörlerin çoklu-yüklenmesi
  - Non-**static** üye fonksiyonu, tek argüment
  - Üye-olmayan fonksiyon, iki argüment

```
class String {
public:
   const String &operator+=( const String & );
   ...
};
```

Denk olarak

```
   y += z;
   y.operator+=( z );
```

Veya

```
class String {
friend const String &operator+=( String &,const String & );
   ...
};
```

Denk olarak

```
   y += z;
   operator+=( y, z );
```

18.8 Örnek Çalışma: Bir Dizi Sınıfı

- Aşağıdakilerin olduğu bir dizi sınıfı oluşturulacak:
  - İndis kontrolü
  - Dizi ataması
  - Boyutlarını bilen diziler
  - Tüm dizilerin **<<** ve **>>** ile çıktıları
  - **==** ve **!=** ile dizilerin karşılaştırılması

```
1  // Fig. 18.4: dizi1.h
2  // Basit dizi sınıfı (tamsayılar icin)
3  #ifndef DIZI1_H
4  #define DIZI1_H
5
6  #include <iostream>
7
8  using std::ostream;
9  using std::istream;
10
11 class Dizi {
12    friend ostream &operator<<( ostream &, const Dizi & );
13    friend istream &operator>>( istream &, Dizi & );
14 public:
15    Dizi ( int = 10 );                    // default constructor
16    Dizi( const Dizi & );                // constructor-ı kopyala
17    ~Dizi();                              // destructor
18    int alBoyut() const;                  // boyut-u gönder
19    const Dizi &operator=( const Dizi & ); // dizileri ata
20    bool operator==( const Dizi & ) const;  // karsılastır esitmi?
21
22    // İki dizi esit ise
23    // true gönder, aksi halde false gönder (operator== kullanıyor).
24    bool operator!=( const Dizi &dogru ) const
25       { return ! ( *bu == dogru ); }
26
27    int &operator[]( int );               // indis operator
28    const int &operator[]( int ) const;  // indis operator
29    static int alDiziSay();           // verilen dizi savısını
30                                          // gönder.
31 private:
32    int boyut; // dizi boyutu
33    int *ptr; // dizinin ilk elemanına pointer
34    static int diziSay;  // verilen dizi sayısı
```

fı

1. Class tanımı

1.1 fonksiyon prototipi

```
35 };
36
37 #endif
38 // Fig 18.4: dizi1.cpp
39 // Dizi sınıfı için üye fonksiyon tanımları
40 #include <iostream>
41
42 using std::cout;
43 using std::cin;
44 using std::endl;
45
46 #include <iomanip>
47
48 using std::setw;
49
50 #include <cstdlib>
51 #include <cassert>
52 #include "dizi1.h"
53
54 // dosya hedefinin statik veri üyesini gir
55 int Dizi::diziSay= 0;   // henüz nesne yok
56
57 // Dizi sınıfı için default constructor (default boyut 10)
58 Dizi::Dizi( int diziBoyut)
59 {
60    boyut = ( diziBoyut > 0 ? diziBoyut : 10 );
61    ptr = new int[ boyut ]; // dizi için yer oluştur
62    assert( ptr != 0 );    // bellek yetersiz ise dur
63    ++diziSay;
64
65    for ( int i = 0; i < boyut; i++ )
66       ptr[ i ] = 0;          // diziyi tanımla
```

fı

1. Header yükle

1.1 fonksiyon tanımı

1.2 `dizi` constructor-ı

```
67 }
68
69 // Dizi sınıfı icin constructor-ı kopyala
70 // sonsuz döngüyü önlemek icin bir referans almalı
71 Dizi::Dizi( const Dizi &ilk ) : boyut( ilk.boyut )
72 {
73    ptr = new int[ boyut ]; // dizive ver olustur
74    assert( ptr != 0 );    // bellek yetersiz ise dur
75    ++diziSay;
76
77    for ( int i = 0; i < boyut; i++ )
78       ptr[ i ] = ilk.ptr[ i ];  // ilk i nesneye kopyala
79 }
80
81 // Dizi sınıfı icin Destructor
82 Dizi::~Dizi()
83 {
84    delete [] ptr;
85    --diziSay;
86 }
87
88 // Dizinin boyutunu al
89 int Dizi::alBoyut() const { return boyut; }
90
91 // coklu atama operatörü
92 // const return olması : ( a1 = a2 ) = a3 den kacınır
93 const Dizi &Dizi::operator=( const Dizi &dogru )
94 {
95    if ( &dogru != bu ) {  // kendine atamayı kontrol et
96
97       // farklı boyutlu diziler icin, orjinal sol taraf dizisini
98       // boz, yeni sol taraf dizisini verlestir.
99       if ( boyut != dogru.boyut) {
100         delete [] ptr;         // yer ayarla
```

fı

1.3 `Dizi` destructor-ı

1.4 `operator=` (atama)

```
101          boyut = dogru.boyut;      // bu nesneyi yeniden boyutlandır
102          ptr = new int[ boyut ]; // dizi kopyası için yer ayarla
103          assert( ptr != 0 );     // yer yoksa dur
104       }
105
106       for ( int i = 0; i < boyut; i++ )
107          ptr[ i ] = dogru.ptr[ i ];  // diziyi nesneye kopyala
108    }
109
110    return *bu;    // x = y = z; ye izin verir
111 }
112
113 // İki dizi boyutu eşit mi? belirle
114 //
115 bool Dizi::operator==( const Dizi &dogru) const
116 {
117    if ( boyut != dogru.boyut )
118       return false;    // farklı boyutlu diziler
119
120    for ( int i = 0; i < boyut; i++ )
121       if ( ptr[ i ] != right.ptr[ i ] )
122          return false; // diziler eşit değil
123
124    return true;         // diziler eşit
125 }
126
127 // non-const Diziler için çoklu yüklemeli indis operatörü
128 // referans return bir soldeğer üretir
129 int &Dizi::operator[]( int indis )
130 {
131    // indis boyut dışı mı?
132    assert( 0 <= indis && indis < boyut );
```

fı

1.5 `operator==` (eşitlik)

1.6 `operator[]` (sabit olmayan diziler için indis)

```
133
134    return ptr[ indis ]; // referans return
135 }
136
137 // const diziler için çoklu yüklemeli indis operatörü
138 // const referans return bir sağdeğer üretir
139 const int &Dizi::operator[]( int indis ) const
140 {
141    // indis boyut dışı mı?
142    assert( 0 <= indis && indis < boyut );
143
144    return ptr[ indis ]; // const referans return
145 }
146
147 // verilen dizi nesnelerinin sayısını gönderir
148 // static fonksiyonlar const olamaz
149 int Dizi::alDiziSay() { return diziSay; }
150
151 // Dizi sınıfı için çoklu yüklemeli girdi operatörü
152 // dizinin tümü girdidir.
153 istream &operator>>( istream &input, Dizi &a )
154 {
155    for ( int i = 0; i < a.boyut; i++ )
156       input >> a.ptr[ i ];
157
158    return input;    // cin >> x >> y; e izin verir
159 }
160
161 // Dizi sınıfı için çoklu yüklemeli çıktı operatörü
162 ostream &operator<<( ostream &output, const Dizi &a )
163 {
```

fı

1.6 operator[]
(const diziler için indis)

1.7 alDiziSay

1.8 operator>>
(girdi dizisi)

1.9 operator<<
(çıktı dizisi)

```
164    int i;
165
166    for ( i = 0; i < a.boyut; i++ ) {
167       output << setw( 12 ) << a.ptr[ i ];
168
169       if ( ( i + 1 ) % 4 == 0 ) // her satırda 4 sayı
170          output << endl;
171    }
172
173    if ( i % 4 != 0 )
174       output << endl;
175
176    return output;   // cout << x << y; a izin verir
177 }
178 // Fig. 18.4: fig18_04.cpp
179 // Dizi sınıfı için ana program
180 #include <iostream>
181
182 using std::cout;
183 using std::cin;
184 using std::endl;
185
186 #include "dizi1.h"
187
188 int main()
189 {
190    // henüz nesne yok
191    cout << " Girilen dizi sayısı = "
192         << Dizi::alDiziSay() << '\n';
193
```

fı

1. Header yükle

```
194   // iki dizi oluştur
195   Dizi tam1( 7 ), tam2;
196   cout << " Girilen dizi sayısı = "
197        << Dizi::alDiziSay() << "\n\n";
198
199   // tam1 boyutunu ve değerlerini yaz
200   cout << " tam1 dizisinin boyutu: "
201        << tam1.alBoyut()
202        << "\n Dizi tanımlandıktan sonra:\n"
203        << tam1 << '\n';
204
205   // tam2 boyut ve değerlerini yaz
206   cout << " tam2 dizisinin boyutu: "
207        << tam2.alBoyut()
208        << "\n Dizi tanımlandıktan sonra:\n"
209        << tam2 << '\n';
210
211   // tam1 ve tam2 yi gir ve yaz
212   cout << " 17 tamsayı gir:\n";
213   cin >> tam1 >> tam2;
214   cout << " Girdiden sonra diziler:\n"
215        << "tam1:\n" << tam11
216        << " tam2:\n" << tam2 << '\n';
217
218   // çoklu yüklemeli (!=) operatorü kullan
219   cout << " Hesaplıyor: tam1!= tam2\n";
220   if ( tam1 != tam2 )
221      cout << " Eşit değiller\n";
222
223   // tam1 den tam3 dizisini oluştur
224   // boy ve içeriğini yaz
225   Dizi tam3( tam1 );
226
```

fı

1.1 nesneleri belirle

2. Fonksiyon çağır

```
227    cout << "\n tam3 dizisinin boyutu: "
228         << tam3.alBoyut()
229         << "\n Dizi tanımlandıktan sonra:\n"
230         << tam3 << '\n';
231
232    // (=) operatörü
233    cout << " tam2 yi tam1 e atıyor:\n";
234    tam1 = tam2;
235    cout << " tam1:\n" << tam1
236         << " tam2:\n" << tam2 << '\n';
237
238    // (==) operatörü
239    cout << " Hesaplıyor: tam1 == tam2\n";
240    if ( tam1 == tam2 )
241       cout << " Diziler eşit. \n\n";
242
243    // indis operatörü sağ değer üretir
244    cout << " tam1[5] : " << tam1[ 5 ] << '\n';
245
246    // indis operatörü soldeğer üretir
247    cout << " tam[5] e 1000 değerini atıyor \n";
248    tam1[ 5 ] = 1000;
249    cout << " tam1:\n" << tam1 << '\n';
250
251    //
252    cout << " 1000 değerini tam1[15] e atamayı deniyor" << endl;
253    tam1[ 15 ] = 1000;  // HATA: boyut dışı
254
255    return 0;
256 }
```

fı

2. Fonksiyon çağır

3. Yaz

```
Girilen dizi sayısı = 0
Girilen dizi sayısı = 2

Tam1 dizisinin boyutu : 7
Dizi tanımlandıktan sonra:
          0          0          0          0
          0          0          0

Tam2 dizsinin boyutu : 10
Dizi tanımlandıktan sonra:
          0          0          0          0
          0          0          0          0
          0          0

17 tamsayı gir:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
Girdiden sonra dizler:
tam1:
          1          2          3          4
          5          6          7
tam2:
          8          9         10         11
         12         13         14         15
         16         17

Hesaplıyor: tam1 != tam2
Eşit değiller

Tam3 dizisinin boyutu: 7
Dizi tanımlandıktan sonra:
          1          2          3          4
          5          6          7
```

18.8 Örnek Çalışma: Bir Dizi Sınıfı

```
Tam2 yi tam1 e atıyor:
tam11:
           8           9          10          11
          12          13          14          15
          16          17
tam2:
           8           9          10          11
          12          13          14          15
          16          17

Hesaplıyor: tam1 == tam2
Diziler eşit.

tam1[5] = 13
tam1[5] e 1000 değerini atıyor
tam1:
           8           9          10          11
          12        1000          14          15
          16          17

1000 değerini tam1[15] e atamayı deniyor
Assertion failed: 0 <= indis && indis < boyut, file Dizi1.cpp, line 95
abnormal program termination
```

18.9 Tipler Arası Dönüşüm

- Dönüştürme (cast) operatörü
  - Nesneleri hazır-tip veya diğer nesnelere dönüştürür
  - Dönüşüm operatörü bir non-**`static`** üye fonksiyon olmalıdır.
  - Bir **`friend`** fonksiyon olamaz
  - return tipi belirtmeyiniz

  Kullanıcı-tanımlı class **`A`** için

```
A::operator char *() const;       // A, char -a
A::operator int() const;          //A, int -e
A::operator digerSinif() const; //A, digerSinif -a
```

  - Derleyici **`(char *) s`** gördüğünde

    `s.operator char*()` -i çağırır

- Derleyici geçici nesneler oluşturmak için bu fonksiyonları çağırabilir
  - Eğer **`s`** **`char *`** tipinde değilse

```
cout << s;
```

  için

  `A::operator char *() const;` çağırır

18.10 ++ ve -- Yüklemesi

- Önce/sonra-artır/azalt operatörleri
  - Çoklu yüklenebilir
  - Derleyici ikisini nasıl ayırt eder?
  - Çoklu yüklemeli öncül versiyonları unary operatörlerdeki gibi Örneğin **`++d1`**; için **`d1.operator++();`**
- Soncul versiyonlar
  - Derleyici örneğin

    `d1++;` gördüğünde

    `d1.operator++( 0 );`

    üye fonksiyon çağrımını oluşturur.

    Prototip:

    `Gun::operator++( int );`