



## C++ - Modül 02

Ad-hoc çok biçimlilik, operatör aşırı  
yükleme ve Ortodoks Kanonik sınıf formu

### Özet:

*Bu doküman C++ modüllerinden Modül 02'nin alıştırma larını  
içermektedir.*

*Sürüm: 7.1*

# İçindekiler

<b>I</b>	<b>Giriş</b>	<b>2</b>
<b>II</b>	<b>Genel kurallar</b>	<b>3</b>
<b>III</b>	<b>Yeni kurallar</b>	<b>5</b>
<b>IV</b>	<b>Alıştırma 00: Ortodoks Kanonik Biçimdeki İlk Dersim</b>	<b>6</b>
<b>V</b>	<b>Alıştırma 01: Daha kullanışlı bir sabit nokta sayı sınıfına doğru</b>	<b>8</b>
<b>VI</b>	<b>Alıştırma 02: Şimdi konuşuyoruz</b>	<b>10</b>
<b>VII</b>	<b>Alıştırma 03: BSP</b>	<b>12</b>

# Bölüm I Giriş

*C++, Bjarne Stroustrup tarafından C programlama dilinin ya da "C with Classes" (Sınıflı C) dilinin bir uzantısı olarak yaratılmış genel amaçlı bir programlama dilidir (kaynak: [Wikipedia](#)).*

Bu modüllerin amacı sizi **Nesne Yönelimli Programlama ile** tanıştırmaktır. Bu, C++ yolculuğunuzun başlangıç noktası olacaktır. OOP öğrenmek için birçok dil önerilmektedir. Eski dostunuz C'den türetildiği için C++'ı seçmeye karar verdik. Bu karmaşık bir dil olduğundan ve işleri basit tutmak için kodunuz C++98 standardına uygun olacaktır.

Modern C++'ın pek çok açıdan çok farklı olduğunun farkındayız. Dolayısıyla, yetkin bir C++ geliştiricisi olmak istiyorsanız, 42 Common Core'dan sonra daha ileri gitmek size kalmış!

# Bölüm II Genel

## kurallar

### Derleme

- Kodunuzu c++ ve -Wall -Wextra -Werror bayrakları ile derleyin
- Eğer -std=c++98 bayrağını eklerseniz kodunuz yine de derlenmelidir

### Biçimlendirme ve adlandırma kuralları

- Alıştırma dizinleri şu şekilde adlandırılacaktır: ex00, ex01, ... , exn
- Dosyalarınızı, sınıflarınızı, işlevlerinizi, üye işlevlerinizi ve niteliklerinizi yönergelerde belirtildiği şekilde adlandırın.
- Sınıf adlarını **UpperCamelCase** biçiminde yazın. Sınıf kodu içeren dosyalar her zaman sınıf adına göre adlandırılacaktır. Örneğin: ClassName.hpp/ClassName.h, ClassName.cpp veya ClassName.tpp. Bu durumda, tuğla duvar anlamına gelen "BrickWall" sınıfının tanımını içeren bir başlık dosyanız varsa, adı BrickWall.hpp olacaktır.
- Aksi belirtilmedikçe, her çıktı mesajı bir yeni satır karakteriyle sonlandırılmalı ve standart çıktıda görüntülenmelidir.
- *Güle güle Norminette!* C++ modüllerinde herhangi bir kodlama stili zorunlu değildir. En sevdiğinizi takip edebilirsiniz. Ancak, akran değerlendiricilerinizin anlayamadığı bir kodun not veremeyecekleri bir kod olduğunu unutmayın. Temiz ve okunabilir bir kod yazmak için elinizden geleni yapın.

### İzinli/Yasak

Artık C'de kodlama yapmıyorsunuz. C++ zamanı! Bu nedenle:

- Standart kütüphanedeki neredeyse her şeyi kullanmanıza izin verilir. Bu nedenle, zaten bildiklerinize bağlı kalmak yerine, alışkın olduğunuz C işlevlerinin C++-ish sürümlerini mümkün olduğunca kullanmak akıllıca olacaktır.
- Ancak, başka herhangi bir harici kütüphane kullanamazsınız. Bu, C++11 (ve türetilmiş formlar) ve Boost kütüphanelerinin yasak olduğu anlamına gelir. Aşağıdaki fonksiyonlar da yasaktır: \*printf(), \*alloc() ve free(). Eğer bunları kullanırsanız, notunuz 0 olacaktır ve hepsi bu kadar.

- Aksi açıkça belirtilmedikçe, using namespace <ns\_name> ve arkadaş anahtar kelimeleri yasaktır. Aksi takdirde notunuz -42 olacaktır.
- **STL'yi sadece Modül 08 ve 09'da kullanmanıza izin verilmektedir.** Bunun anlamı: o zamana kadar **Kapsayıcı** (vektör/liste/harita/ve benzeri) ve **Algoritma** (<algorithm> başlığını içermesi gereken herhangi bir şey) kullanmayacaksınız. Aksi takdirde notunuz -42 olacaktır.

### Birkaç tasarım gereksinimi

- C++'da da bellek sızıntısı meydana gelir. Bellek ayırdığınızda (new anahtar sözcüğü), **bellek sızıntılarından** kaçınmalısınız.
- Modül 02'den Modül 09'a kadar, **aksi açıkça belirtilmediği sürece**, dersleriniz **Ortodoks Kanonik Formunda** tasarlanmalıdır.
- Bir başlık dosyasına konulan herhangi bir işlev uygulaması (işlev şablonları hariç) alıştırma için 0 anlamına gelir.
- Başlıklarınızın her birini diğerlerinden bağımsız olarak kullanabilmelisiniz. Bu nedenle, ihtiyaç duydukları tüm bağımlılıkları içermelidirler. Ancak, **include korumaları** ekleyerek çifte içerme sorunundan kaçınmalısınız. Aksi takdirde notunuz 0 olacaktır.

### Beni okuyun

- Gerekirse bazı ek dosyalar ekleyebilirsiniz (örneğin, kodunuzu bölmek için). Bu ödevler bir program tarafından doğrulanmadığından, zorunlu dosyaları teslim ettiğiniz sürece bunu yapmaktan çekinmeyin.
- Bazen bir alıştırmaların yönergeleri kısa görünebilir ancak örnekler, yönergelerde açıkça yazılmayan gereksinimleri gösterebilir.
- Başlamadan önce her modülü tamamen okuyun! Gerçekten, okuyun.
- Odin tarafından, Thor tarafından! Beyninizi kullanın!!!



Çok sayıda sınıf uygulamanız gerekecek. En sevdiğiniz metin düzenleyicinizi komut dosyası haline getiremediğiniz sürece bu sıkıcı görünebilir.



Egzersizleri tamamlamanız için size belirli bir miktar özgürlük verilir. Ancak, zorunlu kurallara uyun ve tembellik etmeyin. Pek çok faydalı bilgiyi kaçırsınız! Teorik kavramlar hakkında okumaktan çekinmeyin.

# Bölüm III Yeni

## kurallar


Şu andan itibaren, aksi açıkça belirtilmedikçe, tüm sınıflarınız **Ortodoks Kanonik Formunda** tasarlanmalıdır. Ardından, aşağıdaki dört gerekli üye işlevi uygulayacaklardır:

- Varsayılan kurucu
- Kopyalama kurucusu
- Kopyalama atama operatörü
- Yıkıcı

Sınıf kodunuzu iki dosyaya bölün. Başlık dosyası (.hpp/.h) sınıf tanımını içerirken, kaynak dosyası (.cpp) uygulamayı içerir.

## Bölüm IV

# Alıştırma 00: Ortodoks Kanonik Biçimdeki İlk Dersim

	Egzersiz : 00
Ortodoks Kanonik Formunda İlk Dersim	
Giriş dizini : <i>ex00/</i>	
Teslim edilecek dosyalar : <code>Makefile</code> , <code>main.cpp</code> , <code>Fixed.{h, hpp}</code> , <code>Fixed.cpp</code>	
Yasak fonksiyonlar : Hiçbiri	

Tam sayıları ve kayan noktalı sayıları bildiğini sanıyorsun. Ne kadar şirin.

Lütfen bu 3 sayfalık makaleyi ([1](#), [2](#), [3](#)) okuyun ve bilmediğinizi keşfedin. Hadi, okuyun.

Bugüne kadar, kodunuzda kullandığınız her sayı temelde ya bir tamsayı ya da kayan noktalı sayı ya da bunların varyantlarından (kısa, karakter, uzun, çift vb.) biriydi. Yukarıdaki makaleyi okuduktan sonra, tamsayıların ve kayan noktalı sayıların zıt özelliklere sahip olduğunu varsaymak güvenlidir.

Ama bugün işler değişecek. Yeni ve harika bir sayı türünü keşfedeceksiniz: **sabit noktalı sayılar**! Çoğu dilin skaler türlerinde her zaman eksik olan sabit noktalı sayılar, performans, doğruluk, aralık ve hassasiyet arasında değerli bir denge sunar. Bu, sabit noktalı sayıların neden özellikle bilgisayar grafikleri, ses işleme veya bilimsel programlama için uygun olduğunu açıklar.

C++'da sabit noktalı sayılar olmadığından, bunları toplayacaksınız. Berkeley'deki [bu makale](#) iyi bir başlangıçtır. Berkeley Üniversitesi'nin [bu bölümünü](#) okuyun. yoksa, Wikipedia sayfasının [bu bölümünü](#) okuyun.

Ortodoks Kanonik Formunda sabit noktalı bir sayıyı temsil eden bir sınıf oluşturun:

- Özel üyeler:
  - Sabit noktalı sayı değerini saklamak için bir **tamsayı**.
  - Kesirli bit sayısını saklamak için **statik** bir **sabit tamsayı**. Değeri her zaman 8 tamsayı değişmezi olacaktır.
- Kamu üyeleri:
  - Sabit noktalı sayı değerini 0 olarak başlatan varsayılan bir kurucu.
  - Bir kopya kurucusu.
  - Bir kopya atama operatörü aşırı yükü.
  - Bir yıkıcı.
  - Bir üye işlev `int getRawBits( void ) const`; sabit nokta değerinin ham değerini döndürür.
  - Bir üye işlev `void setRawBits( int const raw );` sabit noktalı sayının ham değerini ayarlar.

Bu kodu çalıştırıyorum:

```
#include <iostream>

int main( void ) {

    Sabit a;
    Sabit b( a );
    Sabit c;

    c = b;

    std::cout << a.getRawBits() << std::endl;
    std::cout << b.getRawBits() << std::endl;
    std::cout << c.getRawBits() << std::endl;

    0 döndür;
}
```


Şuna benzer bir çıktı vermelidir:

```
$> ./a.out
Varsayılan kurucu çağrıldı
Kopya kurucu çağrıldı
Kopya atama işlevi çağrıldı // <-- Uygulamınıza bağlı olarak bu satır eksik olabilir getRawBits üye işlevi çağrıldı
Varsayılan kurucu çağrıldı
getRawBits üye işlevi olarak
adlandırılan kopya atama işlevi
getRawBits üye işlevi olarak
adlandırılan 0
getRawBits üye işlevi çağrıldı 0
getRawBits üye işlevi çağrıldı 0
Yıkıcı çağrıldı
Yıkıcı çağrıldı
Yıkıcı çağrıldı
$>
```



## Bölüm V

### Alıştırma 01: Daha kullanışlı bir sabit nokta sayı sınıfına doğru

	Alıştırma 01
Daha kullanışlı bir sabit nokta sayı sınıfına doğru	
Giriş dizini : <i>ex01/</i>	
Teslim edilecek dosyalar : Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp	
İzin verilen işlevler : roundf (<cmath>'ten)	

Önceki alıştırma iyi bir başlangıçtı ancak sınıfımız oldukça kullanışsız. Sadece 0.0 değerini temsil edebilir.

Aşağıdaki genel kurucuları ve genel üye işlevlerini sınıfınıza ekleyin:

- Parametre olarak **sabit** bir **tamsayı** alan bir kurucu.  
Bunu karşılık gelen sabit nokta değerine dönüştürür. Kesirli bit değeri, egzersiz 00'da olduğu gibi 8'e başlatılır.
- Parametre olarak **sabit** bir **kayan noktalı sayı** alan bir kurucu.  
Bunu karşılık gelen sabit nokta değerine dönüştürür. Kesirli bit değeri, egzersiz 00'da olduğu gibi 8'e başlatılır.
- Bir üye işlev float toFloat( void ) const;  
sabit nokta değerini kayan nokta değerine dönüştürür.
- Bir üye işlev int toInt( void ) const;  
sabit nokta değerini bir tamsayı değerine dönüştürür. Ve

aşağıdaki fonksiyonu **Sabit** sınıf dosyalarına ekleyin:

- Parametre olarak aktarılan çıktı akışı nesnesine sabit noktalı sayının kayan noktalı gösterimini ekleyen ekleme ("") işlevinin aşırı yüklemesi.

kodu çalıştırmak:

```
#include <iostream>

int main( void ) {

    Sabit a;
    Sabit const b( 10 );
    Sabit const c( 42.42f );
    Sabit const d( b );

    a = Sabit( 1234.4321f );

    std::cout << "a is " << a << std::endl;
    std::cout << "b is " << b << std::endl;
    std::cout << "c is " << c << std::endl;
    std::cout << "d is " << d << std::endl;

    std::cout << "a is " << a.toInt() << " as integer" << std::endl;
    std::cout << "b is " << b.toInt() << " as integer" << std::endl;
    std::cout << "c is " << c.toInt() << " as integer" << std::endl;
    std::cout << "d is " << d.toInt() << " as integer" << std::endl;

    0 döndür;
}
```


Şuna benzer bir çıktı vermelidir:

```
$> ./a.out
Varsayılan kurucu Int
kurucu olarak adlandırılır
Float kurucu olarak
adlandırılır Copy kurucu
olarak adlandırılır
Float yapıcı olarak adlandırılan
kopya atama operatörü
Yıkıcı olarak adlandırılan kopya
atama operatörü
a 1234.43
b 10'dur
c 42.4219'dur.
d 10'dur
a tamsayı olarak 1234
b tamsayı olarak 10'dur
c tamsayı olarak 42'dir
d tamsayı olarak
10'dur Yıkıcı
olarak
adlandırılan Yıkıcı
olarak
adlandırılan Yıkıcı
olarak
adlandırılan
$>
```



## Bölüm VI

### Alıştırma 02: Şimdi konuşuyoruz

	Alıştırma 02
Şimdi konuşuyoruz.	
Giriş dizini : <i>ex02/</i>	
Teslim edilecek dosyalar : Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp	
İzin verilen işlevler : roundf (<cmath>'ten)	

Aşağıdaki operatörleri aşırı yüklemek için sınıfınıza genel üye fonksiyonlar ekleyin:

- 6 karşılaştırma operatörü: >, <, >=, <=, == ve !=.
- 4 aritmetik operatör: +, -, \* ve /.
- Sabit nokta değerini  $1 + \epsilon > 1$  gibi temsil edilebilir en küçük  $\epsilon$  değerinden artıracak veya azaltacak 4 artırma/azaltma (ön artırma ve son artırma, ön azaltma ve son azaltma) operatörü.

Bu dört genel aşırı yüklenmiş üye işlevi sınıfınıza ekleyin:

- Sabit noktalı sayılar üzerinde iki referansı parametre olarak alan ve en küçük olana bir referans döndüren statik bir üye fonksiyon min.
- Parametre olarak iki **sabit** referansı alan min statik üye fonksiyonu sabit noktalı sayılar ve en küçüğüne bir referans döndürür.
- Parametre olarak sabit noktalı sayılar üzerinde iki referans alan ve en büyük olana bir referans döndüren statik bir üye fonksiyon max.
- Parametre olarak iki **sabit** referansı alan max statik üye fonksiyonu sabit noktalı sayılar ve en büyüğüne bir referans döndürür.

Sınıfınızın her özelliğini test etmek size kalmıştır. Ancak, aşağıdaki kodu çalıştırarak:

```
#include <iostream>

int main( void ) {

    Sabit Sabit    a;
    const          b( Sabit( 5.05f ) * Sabit( 2 ) );

    std::cout << a << std::endl;
    std::cout << ++a << std::endl;
    std::cout << a << std::endl;
    std::cout << a++ << std::endl;
    std::cout << a << std::endl;

    std::cout << b << std::endl;

    std::cout << Fixed::max( a, b ) << std::endl;

    0 döndür;
}
```

Aşağıdaki gibi bir çıktı vermelidir (daha iyi okunabilirlik için, aşağıdaki örnekte kurucu/yıkıcı mesajlar kaldırılmıştır):


```
$> ./a.out
0
0.00390625
0.00390625
0.00390625
0.0078125
10.1016
10.1016
$>
```



Eğer 0 ile bölme işlemi yaparsanız, programın çökmesi kabul edilebilir

## Bölüm VII

### Alıştırma 03: BSP

	Alıştırma 03
	BSP
	Dönüş dizini : <i>ex03/</i>
	Teslim edilecek dosyalar : Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp, Point.{h, hpp}, Point.cpp, bsp.cpp
	İzin verilen işlevler : <code>roundf</code> (<cmath>'ten)

Artık işlevsel bir **Sabit** sınıfınız olduğuna göre, onu kullanmak güzel olacaktır.

Bir noktanın bir üçgenin içinde olup olmadığını gösteren bir fonksiyon uygulayın. Çok kullanışlı, değil mi?



BSP, ikili alan bölümlene anlamına gelir. Rica ederim. :)



Bu modülü egzersiz 03'ü yapmadan geçebilirsiniz.

Ortodoks Kanonik Formunda 2 boyutlu bir noktayı temsil eden **Point** sınıfını oluşturarak başlayalım:

- Özel üyeler:
  - Sabit bir x niteliği.
  - Sabit bir y niteliği.
  - İşe yarar başka bir şey.
- Kamu üyeleri:
  - x ve y değerlerini 0 olarak başlatan varsayılan bir kurucu.
  - Parametre olarak iki sabit kayan noktalı sayı alan bir kurucu. Bu parametreler ile x ve y'yi başlatır.
  - Bir kopya kurucusu.
  - Bir kopya atama operatörü aşırı yükü.
  - Bir yıkıcı.
  - İşe yarar başka bir şey.

Sonuç olarak, aşağıdaki fonksiyonu uygun dosyada uygulayın:

```
bool bsp( Point const a, Point const b, Point const c, Point const point);
```

- a, b, c: Sevgili üçgenimizin köşeleri.
- nokta: Kontrol edilecek nokta.
- Geri döner: Nokta üçgenin içindeyse true. Aksi takdirde False. Böylece, nokta bir tepe noktası veya kenar üzerindeyse False döndürür.

Sınıfınızın beklendiği gibi davrandığından emin olmak için kendi testlerinizi uygulayın ve teslim edin.

