



C++ - Modül 04

Alt tip çok biçimliliği, soyut sınıflar, arayüzler

Özet:

Bu doküman C++ modüllerinden Modül 04'ün alıştırmalarını içermektedir.

Sürüm: 11

İçindekiler

I	Giriş	2
II	Genel kurallar	3
III	Alıştırma 00: Polimorfizm	6
IV	Alıştırma 01: Dünyayı ateşe vermek istemiyorum	8
V	Alıştırma 02: Soyut sınıf	10
VI	Alıştırma 03: Arayüz ve özet	11
VII	Sunum ve akran değerlendirmesi	15

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, .cpp veya .hpp. Bu durumda, tuğla duvar anlamına gelen "BrickWall" sınıfının tanımını içeren başlık dosyanız varsa, adı BrickWall.hpp olacaktır.
- Aksi belirtilmedikçe, her çıktı mesajı bir yeni satır karakteriyle sonlandırılmalı ve standart 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 ad alanının <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. Bununla birlikte, **include korumaları** ekleyerek çift içirme 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 alıştırmanın yönergeleri kısa görünebilir ancak örnekler, açıkça yazılmayan gereksinimleri gösterebilir.
- Başlamadan önce her modülü tamamen okuyun! Gerçekten okuyun.
- Odin tarafından, Thor tarafından! Beynini kullan!!!



C++ projeleri için Makefile ile ilgili olarak, C'deki aynı kurallar geçerlidir (Makefile ile ilgili Norm bölümüne bakın).




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

Alıştırma 00: Polimorfizm

	Egzersiz : 00
Polimorfizm	
Giriş dizini : <i>ex00/</i>	
Teslim edilecek dosyalar : Makefile, main.cpp, *.cpp, *.{h, hpp}	
Yasak fonksiyonlar : Hiçbiri	

Her egzersiz için yapabileceğiniz **en eksiksiz testleri** sunmalısınız.
Her sınıfın kurucuları ve yıkıcıları belirli mesajlar göstermelidir. Tüm sınıflar için aynı mesajı kullanmayın.

Animal adında basit bir temel sınıfı uygulayarak başlayın. Bir korumalı niteliğe sahiptir:

- `std::string` türü;

Animal'dan miras alan bir **Dog** sınıfı uygulayın. Hayvan'dan miras alan bir **Kedi** sınıfı uygulayın.

Bu iki türetilmiş sınıf, adlarına bağlı olarak tür alanlarını ayarlamalıdır. Ardından, Dog'un türü "Dog" olarak ve Cat'in türü "Cat" olarak başlatılacaktır. Animal sınıfının türü boş bırakılabilir veya istediğiniz değere ayarlanabilir.

Her hayvan üye fonksiyonunu kullanabilmelidir:
`makeSound()`
Uygun bir ses çıkaracaktır (kediler havlamaz).

Bu kodu çalıştırmak, Animal'ın değil, Dog ve Cat sınıflarının özel seslerini yazdırmalıdır.

```
int main()
{
    const Hayvan* meta= new Hayvan();
    const Hayvan* j= new Dog();
    const Hayvan* i= new Kedi();

    std::cout<< j->getType()<< " "<< std::endl;
    std::cout<< i->getType()<< " "<< std::endl; i-
>makeSound(); //kedi sesini çıkaracak!
    j->makeSound();
    meta->makeSound();
    ---


    0 döndür;
}
```

Nasıl çalıştığını anladığınızdan emin olmak için, **WrongAnimal** sınıfından miras alan bir **WrongCat** sınıfı uygulayın. Yukarıdaki kodda Hayvan ve Kedi'yi yanlış olanlarla değiştirirseniz, WrongCat WrongAnimal sesini çıkarmalıdır.

Yukarıda verilenlerden daha fazla test uygulayın ve teslim edin.

Bölüm IV

Alıştırma 01: Dünyayı ateşe vermek istemiyorum

	Egzersiz : 01
Dünyayı ateşe vermek istemiyorum.	
Giriş dizini : <i>ex01/</i>	
Teslim edilecek dosyalar : Önceki alıştırmadan dosyalar+ *.cpp, *.{h, hpp}	
Yasak fonksiyonlar : Hiçbiri	

Her sınıfın kurucuları ve yıkıcıları belirli mesajları göstermelidir.

Bir **Beyin** sınıfı uygulayın. Fikirler adında 100 `std::string`'den oluşan bir dizi içerir. Bu şekilde, Köpek ve Kedi özel bir `Brain*` niteliğine sahip olacaktır. Yapım aşamasında, Köpek ve Kedi `new ()` kullanarak Beyinlerini oluşturacaktır; Yok edildikten sonra, Köpek ve Kedi Beyinlerini silecektir.

Ana fonksiyonunuzda, **Hayvan** nesnelerinden oluşan bir dizi oluşturun ve doldurun. Bunun yarısı **Köpek** nesneleri, diğer yarısı da **Kedi** nesneleri olacaktır. Programınızın yürütülmesinin sonunda, bu dizi üzerinde döngü oluşturun ve her bir Hayvanı silin. Köpekleri ve kedileri doğrudan Hayvan olarak silmelisiniz. Uygun yıkıcılar beklenen sırada çağrılmalıdır.

Bellek sızıntılarını kontrol etmeyi unutmayın.

Bir Köpeğin veya Kedinin kopyası sığ olmamalıdır. Bu nedenle, kopyalarınızın derin kopyalar olduğunu test etmelisiniz!

```
int main()
{
    const Hayvan* j= new Dog();
    const Hayvan* i= new Kedi();


    delete j; //bir sızıntı oluşturmamalıdır
    Silin;
    ---

    0 döndür;
}
```

Yukarıda verilenlerden daha fazla test uygulayın ve teslim edin.

Bölüm V

Alıştırma 02: Soyut sınıf

	Alıştırma : 02
Soyut sınıf	
Dönüş dizini : <i>ex02/</i>	
Teslim edilecek dosyalar : Önceki alıştırmadan dosyalar+ *.cpp, *.{h, hpp}	
Yasak fonksiyonlar : Hiçbiri	

Hayvan nesneleri yaratmak hiç de mantıklı değil. Doğru, hiç ses çıkarmıyorlar!


Olası hataları önlemek için, varsayılan Animal sınıfı örneklenebilir olmamalıdır.

Hayvan sınıfını düzeltin, böylece kimse onu örnekleyemez. Her şey eskisi gibi çalışmalıdır.

İsterseniz, Animal ögesine bir A öneki ekleyerek sınıf adını güncelleyebilirsiniz.

Bölüm VI

Alıştırma 03: Arayüz ve özet

	Egzersiz : 03
Arayüz ve özet	
Dönüş dizini : <i>ex03/</i>	
Teslim edilecek dosyalar : Makefile, main.cpp, *.cpp, *.{h, hpp}	
Yasak fonksiyonlar : Hiçbiri	

Arayüzler C++98'de mevcut değildir (C++20'de bile yoktur). Ancak, saf soyut sınıflar genellikle arayüz olarak adlandırılır. Bu nedenle, bu son alıştırmada, bu modülü anladığınızdan emin olmak için arayüzleri uygulamaya çalışalım.

Aşağıdaki **AMateria** sınıfının tanımını tamamlayın ve gerekli üye fonksiyonları uygulayın.

```
sınıf AMateria
{
    korumalı:
        [...]

    Halka açık:
        AMateria(std::string const & type);
        [...]

        std::string const & getType() const; //Madde türünü döndürür

        virtual AMateria* clone() const = 0;
        virtual void use(ICharacter& target);
};
```

Materias somut sınıfları **Ice** ve **Cure**'u uygulayın. Türlerini ayarlamak için adlarını küçük harflerle kullanın (Ice için "ice", Cure için "cure"). Elbette, clone() üye işlevi aynı türden yeni bir örnek döndürecektir (yani, bir Ice Materia'yı klonlarsanız, yeni bir Ice Materia elde edersiniz).

use(ICharacter&) üye işlevi görüntülenecektir:

- Buz: "* <isim>'e bir buz şimşegi fırlatır *"
- Tedavi: "* <isim>'in yaralarını iyileştirir *"

<name> parametre olarak geçirilen Karakterin adıdır. Açılı parantezleri (< ve >) yazdırmayın.



Bir Materia'yı diğerine atarken, türü kopyalamak mantıklı değildir.

Aşağıdaki arayüzü uygulayacak olan somut **Character** sınıfını yazınız:

```
sınıf ICharacter
{
    Halka açık:
        virtual ~ICharacter() {}
        virtual std::string const & getName() const= 0; virtual
        void equip(AMateria* m) = 0;
        virtual void unequip(int idx)= 0;
        virtual void use(int idx, ICharacter& target)= 0;
};
```

Karakterin 4 slotluk bir envanteri vardır, bu da en fazla 4 Materia anlamına gelir. Envanter yapım aşamasında boştur. Buldukları ilk boş yuvadaki Materia'ları donatırlar. Bu, şu sırayla demektir: 0. yuvadan 3. yuvaya kadar. Dolu bir envantere bir Materia eklemeye veya mevcut olmayan bir Materia'yı kullanmaya/donatmaya çalışırlarsa, hiçbir şey yapmayın (ancak yine de hatalar yasaktır). unequip() üye fonksiyonu Materia'yı SİLMEYELİDİR!



Karakterinizin yerde bıraktığı Materiaları istediğiniz gibi kullanın. unequip() işlevini ya da başka bir şeyi çağırmadan önce adresleri kaydedin, ancak bellek sızıntılarından kaçınmanız gerektiğini unutmayın.

use(int, ICharacter&) üye fonksiyonunun slot[idx]'teki Materia'yı kullanması ve hedef parametresini AMateria::use fonksiyonuna iletmesi gerekecektir.



Karakterinizin envanteri her tür AMateria'yı destekleyebilecek.

Karakteriniz, adını parametre olarak alan bir kurucuya sahip olmalıdır. Bir Karakterin herhangi bir kopyası (kopya kurucusu veya kopya atama operatörü kullanılarak) **derin** olmalıdır. Kopyalama sırasında, bir Karakterin Materiaları yenileri envanterlerine eklenmeden önce silinmelidir. Elbette, bir Karakter yok edildiğinde Materialar silinmelidir.

Aşağıdaki arayüzü uygulayacak somut **MateriaSource** sınıfını yazın:

```
sınıf IMateriaSource
{
    Halka açık:
        virtual ~IMateriaSource() {}
        virtual void learnMateria(AMateria*)= 0;
        virtual AMateria* createMateria(std::string const & type)= 0;
};
```

- **learnMateria(AMateria*)**
Parametre olarak aktarılan Materia'yı kopyalar ve daha sonra klonlanabilmesi için bellekte saklar. Karakter gibi, **MateriaSource** da en fazla 4 bilebilir. Bunların benzersiz olması gerekmez.
- **createMateria(std::string const &)**
Yeni bir Materia döndürür. Bu **MateriaSource** tarafından daha önce öğrenilen Materia'nın bir kopyasıdır ve türü parametre olarak aktarılanla eşittir. Tür bilinmiyorsa 0 döndürür.

Özetle, **MateriaSource**'unuz gerektiğinde oluşturmak için Materia'ların "şablonlarını" öğrenebilmelidir. Daha sonra, sadece türünü tanımlayan bir dize kullanarak yeni bir Materia üretebileceksiniz.

Bu kodu çalıştırıyorum:

```
int main()
{
    IMateriaSource* src= new MateriaSource();
    src->learnMateria(new Ice());
    src->learnMateria(new Cure());

    ICharacter* me= new Character("me");

    AMateria* tmp;
    tmp= src->createMateria("ice"); me-
    >equip(tmp);
    tmp= src->createMateria("cure"); me-
    >equip(tmp);

    ICharacter* bob= new Character("bob");

    me->use(0, *bob);
    me->use(1, *bob);

    bob'u sil;
    beni sil;
    src'yi sil;

    0 döndür;
}
```

Çıkmalı:

```
$> clang++ -W -Wall -Werror *.cpp
$> ./a.out | cat -e
* Bob'a bir buz şimşegi fırlatır *$
* Bob'un yaralarını iyileştirir *$
```

Her zamanki gibi, yukarıda verilenlerden daha fazla test uygulayın ve teslim edin.



Bu modülü egzersiz 03'ü yapmadan geçebilirsiniz.

Bölüm VII

Sunum ve akran değerlendirmesi

Ödevinizi her zamanki gibi Git deponuzda teslim edin. Savunma sırasında yalnızca deponuzdaki çalışmalar değerlendirilecektir. Doğru olduklarından emin olmak için klasörlerinizin ve dosyalarınızın adlarını iki kez kontrol etmekten çekinmeyin.



???????????? XXXXXXXXXX= \$3\$6b616b91536363971573e58914295d42