C++ - Module 05

Repetition ve Exceptions

Özet: Bu belge C++ modüllerinden Module 05'in alıştırmalarını içermektedir.

Versiyon: 11.1

İçindekiler

Bölüm I

Giriş

C++, Bjarne Stroustrup tarafından C programlama dilinin bir uzantısı veya "Classes'lı C" olarak yaratılmış genel amaçlı bir programlama dilidir (kaynak: Wikipedia).

Bu modüllerin amacı sizi **Object-Oriented Programming**'e (Nesne Yönelimli Programlama) tanıştırmaktır. Bu, C++ yolculuğunuzun başlangıç noktası olacaktır. OOP öğrenmek için birçok dil önerilmektedir. Eski dostunuz C'den türetildiği için C++'ı seçtik.

Bu karmaşık bir dil olduğu için ve işleri basit tutmak adına, kodunuz C++98 standardına uygun olacaktır.

Modern C++'ın birçok açıdan önemli ölçüde farklı olduğunun farkındayız. Bu nedenle, yetkin bir C++ geliştiricisi olmak istiyorsanız, 42 Common Core'dan sonra daha ileri gitmeniz size kalmıştır!

Bölüm II

Genel kurallar

Derleme (Compiling)

- Kodunuzu **c++** ve **-Wall -Wextra -Werror** flag'leri ile derleyin
- **-std=c++98** flag'ini ekleseniz bile kodunuz yine de derlenmeli

Formatlama ve isimlendirme kuralları

- Alıştırma dizinleri şu şekilde isimlendirilecektir: ex00, ex01, ... , exn
- Dosyalarınızı, class'larınızı, fonksiyonlarınızı, member function'larınızı ve attribute'larınızı yönergelerde belirtildiği şekilde isimlendirin.
- Class isimlerini **UpperCamelCase** formatında yazın. Class kodu içeren dosyalar her zaman class ismine göre isimlendirilecektir. Örneğin: ClassName.hpp/ClassName.h, ClassName.cpp, veya ClassName.tpp. Yani, tuğla duvar için "BrickWall" adında bir class tanımı içeren bir header dosyanız varsa, ismi BrickWall.hpp olacaktır.
- Aksi belirtilmedikçe, her çıktı mesajı bir newline karakteri ile bitmeli ve standart çıktıya gösterilmelidir.
- Goodbye Norminette! C++ modüllerinde herhangi bir kodlama stili zorlanmamaktadır. Favori stilinizi takip edebilirsiniz. Ancak akran değerlendiricilerinizin anlayamadığı kodun, notlandıramayacakları kod olduğunu unutmayın. Temiz ve okunabilir kod yazmak için elinizden geleni yapın.

İzin Verilen/Yasaklanan

Artık C'de kodlamıyorsunuz. C++ zamanı! Bu nedenle:

- Standart kütüphaneden neredeyse her şeyi kullanmanıza izin verilir. Dolayısıyla, bildiğiniz şeylere bağlı kalmak yerine, alışkın olduğunuz C fonksiyonlarının C++'a özgü versiyonlarını mümkün olduğunca kullanmanız akıllıca olacaktır.
- Ancak, başka herhangi bir harici kütüphane kullanamazsınız. Bu, C++11 (ve türevleri) ile Boost kütüphanelerinin yasak olduğu anlamına gelir. Aşağıdaki fonksiyonlar da yasaklanmıştır: *printf(), *alloc() ve free(). Bunları kullanırsanız, notunuz 0 olur ve hepsi bu kadar.
- Aksi açıkça belirtilmedikçe, **using namespace <ns_name>** ve **friend** keyword'leri yasaktır. Aksi takdirde, notunuz -42 olacaktır.
- STL'i **sadece Module 08 ve 09'da** kullanmanıza izin verilir. Bu şu anlama gelir: o zamana kadar **Container**'lar (vector/list/map, ve benzeri) ve **Algorithm**'ler (anything that requires including the <algorithm> header) kullanılamaz. Aksi takdirde, notunuz -42 olacaktır.

Birkaç tasarım gereksinimi

- C++'da da memory leakage (bellek sızıntısı) oluşur. Bellek ayırdığınızda (**new** keyword'ünü kullanarak), bellek sızıntısından kaçınmalısınız.
- Module 02'den Module 09'a kadar, class'larınız açıkça belirtilmedikçe **Orthodox Canonical Form**'da tasarlanmalıdır.

- Header dosyasına konulan herhangi bir fonksiyon implementasyonu (function template'ler hariç) alıştırma için 0 anlamına gelir.
- Header'larınızın her birini diğerlerinden bağımsız olarak kullanabilmelisiniz. Bu nedenle, ihtiyaç duydukları tüm bağımlılıkları içermeleri gerekir. Ancak, **include guard**'lar ekleyerek double inclusion sorununu önlemelisiniz. Aksi takdirde, notunuz 0 olacaktır.

Beni oku (Read me)

- İhtiyacınız varsa ek dosyalar ekleyebilirsiniz (yani kodunuzu bölmek için). Bu ödevler bir program tarafından doğrulanmadığından, zorunlu dosyaları teslim ettiğiniz sürece bunu yapmakta özgürsünüz.
- Bazen bir alıştırmanın yönergeleri kısa görünebilir, ancak örnekler talimatlarda açıkça yazılmayan gereksinimleri gösterebilir.
- Her modülü başlamadan önce tamamen okuyun! Gerçekten, yapın.
- Odin'e, Thor'a yemin ederim! Beyninizi kullanın!!!

> ⚠️ **DİKKAT** C++ projeleri için Makefile ile ilgili olarak, C'deki (Norm bölümündeki Makefile hakkında) ile aynı kurallar geçerlidir.

> 💡 **İPUCU** Bir sürü class implement etmeniz gerekecek. Favori metin editörünüzü script yazabiliyorsanız, bu sıkıcı gelmeyebilir.

> ℹ️ **BİLGİ** Alıştırmaları tamamlamak için size belirli bir özgürlük verilmiştir. Ancak, zorunlu kurallara uyun ve tembel olmayın. Birçok faydalı bilgiyi kaçırırsınız! Teorik kavramlar hakkında okumaktan çekinmeyin.

Bölüm III

AI Talimatları

● Bağlam

Bu proje, 42 eğitiminizin temel yapı taşlarını keşfetmenize yardımcı olmak için tasarlanmıştır.

Temel bilgi ve becerileri düzgün bir şekilde yerleştirmek için, AI araçlarını ve desteğini kullanırken düşünceli bir yaklaşım benimsemek esastır.

Gerçek temel öğrenme, gerçek entelektüel çaba gerektirir — zorluk, tekrar ve akran öğrenme alışverişleri yoluyla.

AI hakkında daha kapsamlı bir genel bakış için — bir öğrenme aracı olarak, 42 eğitiminin bir parçası olarak ve iş piyasasında bir beklenti olarak — lütfen intranet'teki özel SSS bölümüne başvurun.

● Ana mesaj

☛ Kısayollar olmadan güçlü temeller oluşturun.
☛ Gerçekten teknik ve güç becerileri geliştirin.
☛ Gerçek akran öğrenmeyi deneyimleyin, öğrenmeyi öğrenmeye ve yeni problemleri çözmeye başlayın.
☛ Öğrenme yolculuğu sonuçtan daha önemlidir.
☛ AI ile ilişkili riskler hakkında bilgi edinin ve yaygın tuzaklardan kaçınmak için etkili kontrol uygulamaları ve karşı önlemler geliştirin.

● Öğrenen kuralları:

- Özellikle AI'ya dönmeden önce, atanan görevlerinize akıl yürütmeyi uygulamalısınız.
- AI'dan doğrudan cevaplar istememelisiniz.
- 42'nin AI konusundaki global yaklaşımı hakkında bilgi edinmelisiniz.

● Aşama çıktıları:

Bu temel aşamada, aşağıdaki çıktıları elde edeceksiniz:

- Uygun teknik ve kodlama temelleri edinin.
- AI'nın bu aşamada neden ve nasıl tehlikeli olabileceğini bilin.

● Yorumlar ve örnek:

- Evet, AI'nın var olduğunu biliyoruz — ve evet, projelerinizi çözebilir. Ancak burada öğrenmek için buradasınız, AI'nın öğrendiğini kanıtlamak için değil. Sadece AI'nın verilen problemi çözebileceğini göstermek için zamanınızı (veya bizimkini) boşa harcamayın.
- 42'de öğrenmek cevabı bilmekle ilgili değildir — kendi muhakemenizi oluşturma yeteneğini geliştirmekle ilgilidir. AI size cevabı doğrudan verir, ancak bu sizi kendi akıl yürütmenizi oluşturmaktan alıkoyar. Ve akıl yürütme zaman, çaba alır ve başarısızlığı içerir. Başarıya giden yol kolay olmamalıdır.
- Sınavlar sırasında AI'nın mevcut olmadığını unutmayın — internet yok, akıllı telefon yok, vb. Öğrenme sürecinizde AI'ya çok fazla güvenip güvenmediğinizi hızlıca fark edeceksiniz.
- Akran öğrenimi sizi farklı fikirlere ve yaklaşımlara maruz bırakır, kişilerarası becerilerinizi ve farklı düşünme yeteneğinizi geliştirir. Bu, sadece bir botla sohbet etmekten çok daha değerlidir. Bu yüzden utangaç olmayın — konuşun, sorular sorun ve birlikte öğrenin!
- Evet, AI müfredatın bir parçası olacak — hem bir öğrenme aracı olarak hem de kendi başına bir konu olarak. Hatta kendi AI yazılımınızı oluşturma şansınız bile olacak. Geçeceğiniz crescendo yaklaşımı hakkında daha fazla bilgi edinmek için intranet'te mevcut dokümantasyona bakın.

✓ İyi uygulama:

Yeni bir kavramda takıldım. Yakındaki birine buna nasıl yaklaştıklarını soruyorum. 10 dakika konuşuyoruz — ve aniden tık ediyor. Anlıyorum.

✗ Kötü uygulama:

Gizlice AI kullanıyorum, doğru görünen bir kod kopyalıyorum. Akran değerlendirmesi sırasında hiçbir şeyi açıklayamıyorum. Başarısız oluyorum. Sınav sırasında — AI yok — tekrar takılıyorum. Başarısız oluyorum.

Bölüm IV

Exercise 00: Anneciğim, büyüyünce bürokrat olmak istiyorum!

	Exercise: 00
	Anneciğim, büyüyünce bürokrat olmak istiyorum!
Dizin:	ex00/
Teslim edilecek dosyalar:	Makefile, main.cpp, Bureaucrat.{h, hpp}, Bureaucrat.cpp
Yasak:	None

> ℹ️ **BİLGİ** Lütfen exception class'larının Orthodox Canonical Form'da tasarlanması gerekmediğini unutmayın. Ancak, diğer her class buna uymalıdır.

Ofisler, koridorlar, formlar ve bekleme kuyruklarından oluşan yapay bir kabusu tasarlayalım. Eğlenceli mi geliyor? Hayır mı? Çok kötü.

İlk olarak, bu devasa bürokratik makinedeki en küçük dişli ile başlayın: **Bureaucrat** (Bürokrat).

Bir **Bureaucrat** şunlara sahip olmalıdır:

- Sabit (constant) bir isim.
- 1'den (en yüksek olası derece) 150'ye (en düşük olası derece) kadar değişen bir grade (derece).

Geçersiz bir grade ile Bureaucrat oluşturma girişimi bir exception fırlatmalıdır: ya **Bureaucrat::GradeTooHighException** ya da **Bureaucrat::GradeTooLowException**.

Her iki attribute için getter'lar sağlayacaksınız: **getName()** ve **getGrade()**. Ayrıca bürokrat'ın grade'ini artırmak veya azaltmak için iki member function implement etmelisiniz. Grade aralık dışına çıkarsa, her iki fonksiyon da constructor'daki ile aynı exception'ları fırlatmalıdır.

> 💡 **UNUTMAYIN** Grade 1 en yüksek ve 150 en düşük olduğundan, grade 3'ü artırmak bürokrat için grade 2 ile sonuçlanmalıdır.

Fırlatılan exception'lar **try** ve **catch** blokları kullanılarak yakalanabilir olmalıdır:

```cpp
try
{
   /* do some stuff with bureaucrats */
}
catch (std::exception & e)
{
   /* handle exception */
}
```

Çıktıyı aşağıdaki formatta yazdırmak için insertion («) operator'ünün bir overload'ını implement etmelisiniz (açılı parantezler olmadan):

```
<name>, bureaucrat grade <grade>.
```

Her zamanki gibi, her şeyin beklendiği gibi çalıştığını kanıtlamak için bazı testler gönderin.

Bölüm V

Exercise 01: Düzen olun, sürünenler!

	Exercise: 01
	Düzen olun, sürünenler!
Dizin:	ex01/
Teslim edilecek dosyalar:	Önceki alıştırmadaki dosyalar + Form.{h, hpp}, Form.cpp
Yasak:	None

Artık bürokratlarınız olduğuna göre, onlara yapacak bir şeyler verelim. Bir yığın form doldurmaktan daha iyi bir aktivite olabilir mi?

Bir **Form** class'ı oluşturalım. Şunlara sahiptir:

- Sabit bir isim.
- İmzalanıp imzalanmadığını gösteren bir boolean (oluşturulurken, değildir).
- Onu imzalamak için gereken sabit bir grade.

- Onu execute etmek için gereken sabit bir grade.

Tüm bu attribute'lar **private**'dır, protected değil.

Form'un grade'leri Bureaucrat'ınkilerle aynı kurallara uyar. Bu nedenle, bir form'un grade'i aralık dışındaysa aşağıdaki exception'lar fırlatılacaktır: **Form::GradeTooHighException** ve **Form::GradeTooLowException**.

Daha önce olduğu gibi, tüm attribute'lar için getter'lar yazın ve form'un tüm bilgilerini yazdırmak için insertion («) operator'ünü overload edin.

Ayrıca, Form'a parametre olarak bir Bureaucrat alan bir **beSigned()** member function'ı ekleyin. Bürokrat'ın grade'i yeterince yüksekse (gerekli olandan büyük veya eşitse) form'un durumunu imzalı olarak değiştirir. Unutmayın, grade 1, grade 2'den daha yüksektir. Grade çok düşükse, **Form::GradeTooLowException** fırlatın.

Ardından, Bureaucrat class'ına bir **signForm()** member function'ı ekleyin. Bu fonksiyon formu imzalamaya çalışmak için **Form::beSigned()** çağırmalıdır. Form başarıyla imzalanırsa, şöyle bir şey yazdıracaktır:

```
<bureaucrat> signed <form>
```

Aksi takdirde, şöyle bir şey yazdıracaktır:

```
<bureaucrat> couldn't sign <form> because <reason>.
```

Her şeyin beklendiği gibi çalıştığından emin olmak için bazı testler implement edin ve gönderin.

Bölüm VI

Exercise 02: Hayır, 28C değil 28B formu gerekiyor...

	Exercise: 02
	Hayır, 28C değil 28B formu gerekiyor...
Dizin:	ex02/
Teslim edilecek dosyalar:	Makefile, main.cpp, Bureaucrat.[{h, hpp},cpp], + AForm.[{h, hpp},cpp], ShrubberyCreationForm.[{h, hpp},cpp], + RobotomyRequestForm.[{h, hpp},cpp], PresidentialPardonForm.[{h, hpp},cpp]
Yasak:	None

Artık temel formlarınız olduğuna göre, aslında bir şeyler yapan birkaç tane daha oluşturma zamanı.

Her durumda, base class Form bir **abstract class** olmalıdır ve bu nedenle **AForm** olarak yeniden adlandırılmalıdır. Form'un attribute'larının private kalması ve base class'a ait olması gerektiğini unutmayın.

Aşağıdaki concrete class'ları ekleyin:

- **ShrubberyCreationForm**: Gerekli grade'ler: sign 145, exec 137 Çalışma dizininde **<target>_shrubbery** adlı bir dosya oluşturur ve içine ASCII ağaçları yazar.
- **RobotomyRequestForm**: Gerekli grade'ler: sign 72, exec 45 Bazı delme sesleri çıkarır, ardından **<target>**'ın zamanın %50'sinde başarıyla robotize edildiğini bildirir. Aksi takdirde, robotominin başarısız olduğunu bildirir.
- **PresidentialPardonForm**: Gerekli grade'ler: sign 25, exec 5 **<target>**'ın Zaphod Beeblebrox tarafından affedildiğini bildirir.

Hepsi constructor'larında yalnızca bir parametre alır: form'un hedefi (target). Örneğin, evde çalılık dikmek istiyorsanız "home".

Şimdi, base form'a **execute(Bureaucrat const & executor) const** member function'ını ekleyin ve concrete class'larda form'un eylemini yürütmek için bir fonksiyon implement edin. Form'un imzalı olduğunu ve formu execute etmeye çalışan bürokrat'ın grade'inin yeterince yüksek olduğunu kontrol etmelisiniz. Aksi takdirde, uygun bir exception fırlatın.

Gereksinimleri her concrete class'ta mı yoksa base class'ta mı kontrol edeceğiniz (ve sonra formu execute etmek için başka bir fonksiyon çağıracağınız) size kalmıştır. Ancak, bir yol diğerinden daha zariftir.

Son olarak, Bureaucrat class'ına **executeForm(AForm const & form) const** member function'ını ekleyin. Formu execute etmeye çalışmalıdır. Başarılıysa, şöyle bir şey yazdırın:

```
<bureaucrat> executed <form>
```

Değilse, açık bir hata mesajı yazdırın.

Her şeyin beklendiği gibi çalıştığından emin olmak için bazı testler implement edin ve gönderin.

Bölüm VII

Exercise 03: En azından bu kahve yapmaktan iyidir

	Exercise: 03
	En azından bu kahve yapmaktan iyidir
Dizin:	ex03/

	Exercise: 03
Teslim edilecek dosyalar:	Önceki alıştırmalardan dosyalar + Intern.{h, hpp}, Intern.cpp
Yasak:	None

Bürokratlarımız için bütün gün form doldurmak çok acımasız olacağından, bu sıkıcı görevi üstlenecek intern'ler (stajyerler) vardır. Bu alıştırmada, **Intern** class'ını implement etmelisiniz. Intern'in ismi, grade'i ve benzersiz özellikleri yoktur. Bürokratların umursadığı tek şey işlerini yapmalarıdır.

Ancak, intern'in önemli bir yeteneği vardır: **makeForm()** fonksiyonu. Bu fonksiyon parametre olarak iki string alır: birincisi bir form'un ismini, ikincisi ise form'un hedefini temsil eder. İkinci parametreye başlatılmış hedefi ile (parametre olarak iletilen form ismine karşılık gelen) bir **AForm** nesnesine bir pointer döndürür.

Şöyle bir şey yazdırmalıdır:

```
Intern creates <form>
```

Sağlanan form ismi mevcut değilse, açık bir hata mesajı yazdırın.

Aşırı if/elseif/else yapısı kullanmak gibi okunaksız ve dağınık çözümlerden kaçınmalısınız. Bu tür bir yaklaşım değerlendirme sürecinde kabul edilmeyecektir. Artık Piscine'de (havuzda) değilsiniz. Her zamanki gibi, her şeyin beklendiği gibi çalıştığından emin olmak için her şeyi test etmelisiniz.

Örneğin, aşağıdaki kod "Bender"'ı hedefleyen bir **RobotomyRequestForm** oluşturur:

```cpp
{
  Intern someRandomIntern;
  AForm* rrf;

  rrf = someRandomIntern.makeForm("robotomy request", "Bender");
}
```

Bölüm VIII

Teslim ve Akran Değerlendirmesi

Ödevinizi her zamanki gibi Git repository'nize gönderin. Sadece repository'nizdeki çalışmalar savunma sırasında değerlendirilecektir. Klasör ve dosya isimlerinizin doğru olduğundan emin olmak için iki kez kontrol edin.

Değerlendirme sırasında, projenin kısa bir modifikasyonu bazen istenebilir. Bu, küçük bir davranış değişikliği, yazılacak veya yeniden yazılacak birkaç satır kod veya eklemesi kolay bir özellik içerebilir.

Bu adım her proje için geçerli olmasa da, değerlendirme kılavuzunda belirtilmişse buna hazırlıklı olmalısınız.

Bu adım, projenin belirli bir bölümünü gerçek anlamanızı doğrulamak için tasarlanmıştır. Modifikasyon, seçtiğiniz herhangi bir geliştirme ortamında (örneğin, her zamanki kurulumunuz) gerçekleştirilebilir ve birkaç dakika içinde yapılabilir olmalıdır — değerlendirme kılavuzunun bir parçası olarak belirli bir zaman çerçevesi tanımlanmadıkça.

Örneğin, bir fonksiyon veya script'te küçük bir güncelleme yapmanız, bir görüntüyü değiştirmeniz veya yeni bilgileri saklamak için bir veri yapısını ayarlamanız istenebilir, vb.

Detaylar (kapsam, hedef, vb.) değerlendirme kılavuzunda belirtilecektir ve aynı proje için bir değerlendirmeden diğerine değişebilir.