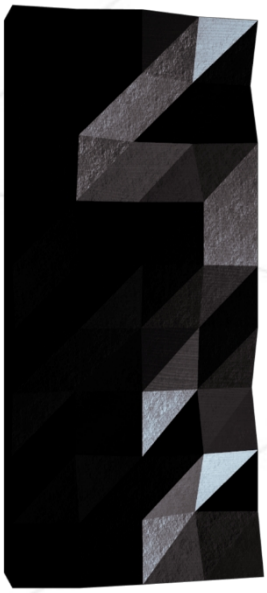

C++ - Modül 08

Şablonlanmış konteynerler, yineleyiciler, algoritmalar

Özet:
Bu doküman C++ modüllerinden Modül 08'in alıştırmalarını içermektedir.

Sürüm: 7

İçindekiler

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 birçok 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.

İzin Verildi/Yasaklandı

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. Bununla birlikte, **include korumaları** ekleyerek çift içerme sorunundan kaçınmalısınız. Aksi takdirde notunuz 0 olacaktır.

Beni oku

- 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, 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 kaçırırsınız! Teorik kavramlar hakkında okumaktan çekinmeyin.

Bölüm III

Modüle özgü kurallar

Bu modülde, alıştırmaların standart Kapsayıcılar ve standart Algoritmalar OLMADAN çözülebileceğini fark edeceksiniz.

Ancak, **bunları kullanmak tam olarak bu Modülün amacıdır**. STL'yi kullanmanıza izin verilir. Evet, **Kapsayıcıları** (vector/list/map/ve benzeri) ve **Algoritmaları** (<algorithm> başlığında tanımlanan) kullanabilirsiniz. Dahası, bunları olabildiğince çok kullanmalısınız. Bu nedenle, uygun olan her yerde bunları uygulamak için elinizden geleni yapın.

Bunu yapmazsanız, kodunuz beklendiği gibi çalışsa bile çok kötü bir not alırsınız. Lütfen tembellik yapmayın.

Şablonlarınızı her zamanki gibi başlık dosyalarında tanımlayabilirsiniz. Ya da isterseniz, şablon bildirimlerinizi başlık dosyalarına yazabilir ve bunların uygulamalarını .tpp dosyalarına yazabilirsiniz. Her durumda, başlık dosyaları zorunlu iken .tpp dosyaları isteğe bağlıdır.

Bölüm IV

Alıştırma 00: Kolay bulma

	Egzersiz : 00
Kolay buluntu	
Giriş dizini : *ex00/*	
Teslim edilecek dosyalar : `Makefile, main.cpp, easyfind.{h, hpp} ve isteğe bağlı dosya: easyfind.tpp`	
Yasak fonksiyonlar : `Hiçbiri`	

İlk kolay egzersiz, doğru adımla başlamanın yoludur.

T türünü kabul eden bir easyfind işlev şablonu yazın. İki parametre alır. Birincisi T tipindedir ve ikincisi bir tamsayıdır.

T'nin **tamsayılardan oluşan** bir kap olduğunu varsayarsak, bu fonksiyon ilk parametrede ikinci parametrenin ilk oluşumunu bulmalıdır.

Herhangi bir oluşum bulunamazsa, bir istisna atabilir veya seçtiğiniz bir hata değerini döndürebilirsiniz. Biraz ilhama ihtiyacınız varsa, standart konteynerlerin nasıl davrandığını analiz edin.

Elbette, her şeyin öngörüldüğü gibi çalıştığından emin olmak için kendi testlerinizi uygulayın ve teslim edin.

`İlişkisel kapsayıcıları işlemek zorunda değilsiniz.`

Bölüm V

Alıştırma 01:

Açıklık

	Egzersiz : 01
Açıklık	
Giriş dizini : *ex01/*	
Teslim edilecek dosyalar : `Makefile, main.cpp, Span.{h, hpp}, Span.cpp`	
Yasak fonksiyonlar : `Hiçbiri`	

En fazla N tamsayı saklayabilen bir **Span** sınıfı geliştirin. N, işaretsiz bir int değişkenidir ve yapıcıya aktarılan tek parametre olacaktır.

Bu sınıf, Span'e tek bir sayı eklemek için addNumber() adlı bir üye fonksiyona sahip olacaktır. Bu, doldurmak için kullanılacaktır. Zaten depolanmış N eleman varsa yeni bir eleman eklemeye yönelik herhangi bir girişim bir istisna fırlatmalıdır.

Ardından, iki üye işlevi uygulayın: shortestSpan() ve longestSpan()

Sırasıyla, saklanan tüm sayılar arasındaki en kısa aralığı veya en uzun aralığı (veya isterseniz mesafeyi) bulacak ve geri döndürecektir. Saklanan hiç sayı yoksa veya yalnızca bir sayı varsa, hiçbir aralık bulunamaz. Bu nedenle, bir istisna atılır.

Elbette kendi testlerinizi yazacaksınız ve bunlar aşağıdakilerden çok daha kapsamlı olacaktır. Span'ınızı en az 10.000 sayı ile test edin. Daha fazlası daha da iyi olacaktır.

Bu kodu çalıştırıyorum:

```
int main()
{
  Span sp = Span(5);

  sp.addNumber(6);
  sp.addNumber(3);
  sp.addNumber(17);
  sp.addNumber(9);
  sp.addNumber(11);

  std::cout << sp.shortestSpan() << std::endl;
  std::cout << sp.longestSpan() << std::endl;

  0 döndür;
}
```

Çıkmalı:

```
$> ./ex01
2
14
$>
```

Son olarak, Span'ınızı **bir dizi yineleyici** kullanarak doldurmak harika olurdu. addNumber() fonksiyonuna binlerce çağrı yapmak çok can sıkıcıdır. Tek bir çağrıda Span'ınıza birçok sayı eklemek için bir üye işlev uygulayın.

Eğer hiçbir fikriniz yoksa, Konteynerleri inceleyin. Bazı üye fonksiyonlar, konteynere bir dizi eleman eklemek için bir dizi iteratör alır.

Bölüm VI

Alıştırma 02: Mutasyona uğramış iğrençlik

	Alıştırma : 02
Mutasyona uğramış iğrençlik	
Dönüş dizini : *ex02/*	
Teslim edilecek dosyalar : `Makefile, main.cpp, MutantStack.{h, hpp} ve isteğe bağlı dosya: MutantStack.tpp`	
Yasak fonksiyonlar : `Hiçbiri`	

Şimdi, daha ciddi şeylere geçme zamanı. Tuhaf bir şey geliştirelim.

std::stack kabı çok güzeldir. Ne yazık ki, yinelenemeyen tek STL taşıyıcılarından biridir. Bu çok kötü.

Ama bunu neden kabul edelim ki? Özellikle de eksik özellikler yaratmak için orijinal yığını parçalama özgürlüğüne sahipsek.

Bu adaletsizliği gidermek için std::stack konteynerini yinelenebilir hale getirmeniz gerekir.

Bir **MutantStack** sınıfı yazın. Bir std::stack **cinsinden uygulanacaktır**. Tüm üye işlevlerinin yanı sıra ek bir özellik daha sunacaktır: **yineleyiciler**.

Elbette, her şeyin beklendiği gibi çalıştığından emin olmak için kendi testlerinizi yazacak ve teslim edeceksiniz.

Aşağıda bir test örneği bulabilirsiniz.

```
int main()
{
MutantStack<int>    mstack;

mstack.push(5);
mstack.push(17);

std::cout << mstack.top() << std::endl;

mstack.pop();

std::cout << mstack.size() << std::endl;

mstack.push(3);
mstack.push(5);
mstack.push(737);
//[...]
mstack.push(0);

MutantStack<int>::iterator it = mstack.begin();
MutantStack<int>::iterator ite = mstack.end();

++it;
--İşte;
while (it != ite)
{
    std::cout << *it << std::endl;
    ++it;
}
std::stack<int> s(mstack);
0 döndür;
}
```

İlk kez MutantStack ile çalıştırırsanız ve ikinci kez MutantStack'i örneğin bir std::list ile değiştirirseniz, iki çıktı aynı olmalıdır. Elbette, başka bir konteyneri test ederken, aşağıdaki kodu ilgili üye fonksiyonlarla güncelleyin (push() push_back() olabilir).