**SAKARYA ÜNİVERSİTESİ   BİLGİSAYAR MÜHENDİSLİĞİ**

**Google Test Framework ile C++ Yazılım Testi Gerçeklemesi**

**Öğrenciler**;  Oğuzhan İNCE, Savaş KAPLAN

**Danışman** **Öğretmen**;  Dr. M. Fatih ADAK

**Yazılım Testi Nedir ?**

Test, bir sistemi manuel veya otomatik yollarla deneyerek veya  değerlendirerek, belirlenmiş gereksinimleri karşıladığının doğrulanması veya  beklenen ile gözlenen sonuçlar arasındaki farkların belirlenmesi   
sürecidir.  Yazılım testi ise bir yazılımın sonsuz sayıdaki çalışma alanından, sınırlı  sayıda ve uygun şekilde  seçilmiş testler ile beklenen davranışlarını  karşılamaya yönelik, dinamik olarak yapılan doğrulama  faaliyetlerini  kapsamaktadır.

**Neden Gereklidir ?**

* Müşteriye sunulmadan önce ürün kalitesinden emin olmak,
* Yeniden çalışma (düzeltme) ve geliştirme masraflarını azaltmak,
* Geliştirme işleminin erken aşamalarında yanlışları saptayarak ileri  aşamalara yayılmasını    
  önlemek, böylece zaman ve maliyetten tasarruf  sağlamak,
* Müşteri memnuniyetini arttırmak ve izleyen siparişler için zemin  hazırlamak.

**Yazılım Test Süreçleri Nelerdir ?**

Genel olarak yazılım projeleri *analiz* -> *tasarım* -> *kodlama* ->  *test* -> *ürün* *süreçleri* izlenerek  geliştirilir. Bütün süreçler birbirini bu şekilde izlese de test süreci hiçbir zaman kodlama  sürecinin bitmesini beklemez. İdeal bir  yazılım test süreci;

*analiz -> tasarım -> ​test hazırlık süreci​-> kodlama ->  dinamik test süreci​ -> ​testin sonlandırılması -> ürün* şeklinde olmak durumundadır.

**Test Hazırlık Süreci**

Bu süreç, yazılım test süreçleri içindeki ilk aşama olmakla beraber, testin  efektif sonuçlar vermesi ve   
verimli olması açısından büyük öneme sahiptir.  Dolayısıyla, bir yazılımı iyi test edebilmek için, test   
işlemlerinden önce,  sağlıklı bir test hazırlık süreci kaçınılmazdır. Test hazırlık sürecinde  yapılması   
gereken birtakım standart işlemler şu şekilde sıralanır:

* Öncelikle test edilecek yazılıma ait analiz ve teknik tasarım aşamaları  ile ilgili dökümanlar

test ekibi tarafından incelenir.

* Yazılım içinde test edilecek ve edilmeyecek modüller belirlenir.
* Risk analizi yapılır ve yapılan değerlendirmeye göre dinamik test  aşamasında uygulanacak

olan test teknikleri ve metodları belirlenir.

* Dinamik testin uygulanacağı ortamlar ve bu ortamların ihtiyaçları  belirlenip, uygun şartlar

sağlanır.

* Test ekibi içinde görev paylaşımı ve zaman planlaması yapılır.
* Testin sonlandırma kriterleri belirlenir.
* Bir programa belirli girdiler (input) verildiğinde hangi çıkışların  (output) ne şekilde alınması

gerektiğini bildiren test case senoryaları  belirlenir.

* Dinamik testin hangi adımlarla ve ne şekilde uygulanacağının  belirtildiği test planı hazırlanır.

Yukarıda sıralanan test hazırlık sürecine ait aşamalar gerçeklendikten sonra  dinamik yazılım testi   
aşamalarına geçilir.

**Dinamik Test Süreci**

Bu süreç kodlama çalışmalarının bitmesine yakın bir dönemde başlar. Bulunan  tüm hatalar çözülmeden ve testin sonlandırma kriterleri sağlanmadan sona  ermez. Test edilecek yazılımın türüne  göre, dinamik olarak uygulanacak test  teknikleri ve bu tekniklerin uygulanma metotları farklılık gösterebilir.

Genel  olarak dinamik test süreci içinde ve sonrasında uygulanabilecek olan testler  ve test teknikleri şu   
şekilde sıralanır:

**● Birim Testi​ ​(Unit Testing) :​** Dinamik test sürecinin ilk aşaması olmakla  beraber, hataların erken bulunup düzeltilebilmesi açısından da bu  sürecin en önemli aşamasını oluşturur. Mikro ölçekte yapılan bu testte,  özel fonksiyonlar veya   
kod modülleri (fonksiyonlar, veri yapıları,  nesneler vb.) test edilir. Bu test, test uzmanlarınca değil   
programcılar  tarafından yapılır ve program kodunun ayrıntıları ile içsel tasarım  biçiminin bilinmesi gerekir. Uygulama kodu çok iyi tasarlanmış bir  mimaride değilse oldukça zor bir testtir.

● **Tümleyim Testi (Integration Testing) :​** Bir uygulamanın farklı  bileşenlerinin beraberce uyum içinde çalışıp çalışmadığını sınamak için  yapılan   
bir testtir. Bileşenler, modüller, bağımsız uygulamalar,  istemci/sunucu uygulamaları biçiminde olabilirler. Bu tür testlere,  özellikle istemci/sunucu uygulamaları ve dağıtık sistemlerin testinde  başvurulmaktadır. Bunun yanısıra uygulamaya yeni işlevsel elemanlar ya  da program modülleri eklendikçe sürekli test   
edilmesi işlemine de  “Artımsal Tümleyim Testi” adı verilir. Test uzmanları ve/veya  programcılar   
tarafından gerçekleştirilen testlerdir.

**● Regresyon Testi (Regression Testing) :​** Uygulamada ve uygulama  ortamlarında gerekli değişiklikler ve sabitlemeler yapıldıktan sonra  yeniden   
yapılan testlere çekilme (regresyon) testi denilir. Böylece,  önceki testlerde belirlenen sorunların giderildiğinden ve yeni hatalar  oluşmadığından emin olunur. Uygulamanın kaç kez yeniden test edilmesi    
gerektiğini belirlemek güçtür ve bu nedenle, özellikle uygulama  geliştirme döneminin sonlarına doğru   
yapılır.

● **Zorlanım – Performans Testi​ ​(Performance Testing) :​** Bu test, çoğu kez  "yük testi" ile aynı anlamda kullanılmaktadır. Aynı zamanda, beklenmedik  (normal   
olmayan) ağır yükler, belirli eylemler ve taleplerin çok fazla  artışı, çok yoğun sayısal işlemler, çok   
karmaşık sorgulamalar vb. ağır  koşullar altında olan bir sistemin işlevsellik testi (iş yapabilme  testi)   
olarak da tanımlanabilmektedir. Bir web sitesi için sistem  tepkisinin hangi noktada azaldığı veya yanıt   
veremez olduğunu belirlemek  için yapılan testler, performans testine örnek teşkil edebilir.

● **Kullanıcı Kabul Testi (User Acceptance Testing) :​** Son kullanıcı veya  müşteri siparişine (veya isteklerine) dayanan son test işlemidir.  Kullanıcıların,   
uygulamayı “kabul” etmeden önce, söz konusu uygulamanın  gereksinimlerini ne ölçüde karşılayıp   
karşılamadığını belirleyip, geri  dönüş yapabileceği testlerdir.

**● Beyaz Kutu Test Tekniği​ ​(White Box Testing Technic) :​** Beyaz kutu test  tekniğinin en genel tabiri kod testidir. Projenin hem kaynak kodu, hem  de derlenmiş   
kodu test edilir. Bu tür testler, uygulama kodunun iç  mantığı üzerindeki bilgiye bağlıdır. Yazılım  
kodundaki deyimler, akış  denetimleri, koşullar vb. elemanlar sınanır.

**● Kara Kutu Test Tekniği​ ​(Black Box Testing Technic ) :​** Test ekipleri  tarafından en çok kullanılan teknik olan kara kutu test tekniği adından  da anlaşılacağı gibi uygulamanın sadece derlenmiş kodu üzerinden test  edilmesi olarak bilinir. Bu test tekniğinde, yazılımın programatik  yapısı, tasarımı veya kodlama tekniği hakkında herhangi bir bilgi olması  gerekli değildir.   
Yazılımın gereksinimine duyulan şeylere yanıt verip  veremediği ve işlevselliği sınanmaktadır.

**Testin Sonlandırılması**

Yapılan testler sonucunda bulunan hatalar düzeltildikten sonra test  sonlandırma kriterleri (test hazırlık   
süreci) kontrol edilir. Eğer tüm  kriterlerin kabul edilebilir düzeyde sağlandığı tespit edilirse test    
sonlandırılır. Testin sonlandırılmasının ardından uygulama müşteri testine  açılır (Kullanıcı Kabul Testi).   
Müşterilerin bulduğu hatalar veya  değiştirilmesi istenilen noktalar gözden geçirilerek tekrar test ekibinin  kontrolüne sunulur. Bu kontrolden çıkan uygulama ürün aşamasına geçer ve  böylelikle yazılım test   
süreci sona erdirilerek, yazılım geliştirme sürecinin  son basamağına geçilmiş olunur.

**Unit Test nedir ?**

Birim Testi bir yazılımın en küçük birimlerinin test edilmesi demektir. Temel  bir örnek vermek istersek   
verdiğimiz iki sayıyı toplayan Topla metodumuz  olsun;

int topla(int sayi1 , int sayi2)  {     return sayi1 + sayi2;  }

Bu method için yazılacak unit test metodu yaklaşık olarak söyle olacaktır;

TEST\_METHOD(toplaTest)  {      int geriDonenDeger;      int beklenenDeger;      geriDonenDeger = topla( 3 + 5 );      beklenenDeger = 8;      Assert::AreEqual( beklenenDeger , geriDonenDeger );  }

Yapmamız gereken, en basit metodumuz için bile giriş parametresini göndererek  beklenen değerle   
metoddan geri dönen değeri karşılaştırmak. Test sistemini  çalıştırdığımız zaman yaptığımız testin   
çalıştığı yada çalışmadığı sonucunu  alıyoruz.

Sonuç olarak :

* Sistemin en küçük birimlerini hızlı bir şekilde test ederiz.
* Kodda hata varsa, hatanın nerede oluştuğunu saptamak kolaylaşır.
* Yazdığımız kodun neredeyse tamamının test edilmiş olmasını – test  sürekliliğini sağlar.

**Microsoft Visual Studio’da Unit Test Yazmak**

Visual Studio’da unit test yazmak için bir unit test projesi oluşturmamız  gerekiyor.

Bu işlemin bir ön şartı var, Visual Studio sadece Statik Kütüphane (.lib)  projeleri için test yazmaya izin   
veriyor. Eğer projemizi çalıştırabilir yapmak  istiyorsak Solution’ımıza bir çalıştırılabilir proje ekleyerek .lib yaptığımız  ana projemizi buradan çağırmak işleri epeyce kolaylaştıracaktır.

Statik Kütüphane yaptığımız projemize Unit Test eklemek için;

Solition Explorer penceresinden sağ tıklayarak Add -> New Project , ya da FILE  menüsünden   
Add - New  Project’i seçerek açılan pencerede soldan Test i  seçiyoruz. Bende 2 çeşit test projesi çıkıyor. Birisi Managed Test Project  diğeri Native Unit Test Project, Native olanı seçip projemizi isimlendirerek    
Solution’ımıza ekliyoruz. İsimlendirme işlemi anlaşılabilir olması için  genellikle testini oluşturduğunuz   
projenizin adının sonuna “Test” kelimesini  ekleyerek yapılıyor.

Oluşan test projemiz External Dependecies, Header Files, Resource Files ve  Source Files bölümlerinden oluşuyor. **Biz Source Files altına unit testlerimizi  ekleyerek devam edeceğiz**.  Projeyi ilk oluşturduğumuzda,**sources altında bize  örnek bir unit test sınıfı da oluşuyor**, ilk olarak onu kullanmaya  başlayabiliriz.

**Unit Test Sınıfı Nasıl Düzenlenir ?**

Tek yapılması gereken test edilecek/üzerinde çalıştığımız sınıfı, test  sınıfına include etmek ve bu   
muhtemelen;

‘#include "../ProjeAdi/TestEdilecekSinif.h"’ ifadesiyle gerçekleştirilecektir.

Her method ve metodun içindeki her durum için TEST\_METHOD’ları alt alta  eklenmelidir ve testleri   
çalıştırmak için TEST menüsünden ‘Run -> All Tests’

Test sonuç kontrolleri ‘Test Explorer’ ekranından yapılır.

Test sınıfımız yaklaşık olarak aşağıdaki gibi gözükecektir.

using namespace Microsoft::VisualStudio::CppUnitTestFramework;    namespace ProjenizinAdiTest    
{  TEST\_CLASS(SinifinizinAdiTest)  {  public:    TEST\_METHOD(MethodunuzunAdiTest)    
{  SinifinizinAdi TestObject;  int actualValue;  int expectedValue;  expectedValue = 1;    
string value = "000";  actualValue = TestObject.MethodunuzunAdi(value.c\_str());       
Assert::AreEqual(expectedValue, actualValue);   }    };  }

**Code Coverage Nedir ?**

Unit test işlemi ile ilgili önemli bir kavramdır, yazılan testlerin kontrol  ettiği kodun, yazılan koda oranı   
anlamına gelir. Genelde % olarak baz alınır.  Code coverage değeri ne kadar yüksekse unit testlerinden   
faydalanma oranı da o  derece yüksektir. Başka bir deyişle bir kodumuzda bir hata olduğunda, hata    
cover etmediğimiz bir kod bölümüne denk gelmişse bunu unit testler ile  bulamazsınız demektir.

**Google Test Nedir ?**

C++ Programlama dili için geliştirilen birim test kütüphanesidir. xUnit  mimarisine sahiptir. POSIX ve   
Windows platformlarında derlenebilir. ​Google  Test​ ​BSD​ Lisansına sahiptir. Diğer çatılar yada yeni yazılacak bir test  yazılımı ile karşılaştırıldığında ayrıcalıkları;   
xUnit, Fixtures, Mocks,  Exceptions, Macros, Templates, Automatic test discovery, a rich set of    
assertions, userdefined assertions, death testleri, fatal ve nonfatal  failures, various options for running   
the tests ve XML test report generation  destekler. Geliştirme sürecinde Google Test kullanılan bazı   
projeler;

● Chromium​ Projesi (Chrome browser ve Chrome OS)

● LLVM​ Compiler

● Protocol Buffers​ (Google's data interchange format)

● OpenCV​ Yapay Zeka (Library)

● Gromacs​ Moleküler Dinamik Simülasyon (Package)

**Google Test Gereklilikleri Nelerdir ?**

**Linux Gereklilikleri**

* GNU-compatible Make veya gmake
* POSIX-standard shell
* POSIX(-2) Regular Expressions (regex.h)
* A C++98-standard-compliant compiler

**Windows Gereklilikleri**

* Microsoft Visual C++ v7.1 veya daha yenisi

**Cygwin Gereklilikleri**

* Cygwin v1.5.25-14 veya daha yenisi

**Mac OS Gereklilikleri**

* Mac OS X v10.4 Tiger veya daha yenisi
* Xcode Geliştirme Araçları

**Katkıda bulunan kişiler için gereklilikler**

* [Python](https://www.python.org/) v2.3 veya daha yenisi
* [CMake](https://cmake.org/) v2.6.4 veya daha yenisi

**Google Test Genel Yapısı**

**Kurulum**

Daha sonra otomatik ve manuel kurulumlara değineceğiz önce Google Testini ve onu kullanan testlerinizi oluşturmak için, yapı sisteminize başlıklarını ve kaynak dosyalarını nerede bulacağını söylemeniz gerekir. Bunu yapmanın kesin yolu, hangi yapı sistemini kullandığınıza bağlıdır ve genellikle basittir.

Google Test'i $ {GTEST\_DIR} dizinine yerleştirdiğimizi varsayalım. Derlemek için bir kitaplık oluşturma hedefi (veya Visual Studio ve Xcode tarafından çağrılan bir proje) oluşturun.

${GTEST\_DIR}/src/gtest-all.cc

$ {GTEST\_DIR} / include sistem üstbilgi arama yolunda ve $ {GTEST\_DIR} normal başlık arama yolunda. Linux benzeri bir sistem ve gcc varsayıldığında, aşağıdakine benzer bir şey yapar:

g++ -I ${GTEST\_DIR}/include -I${GTEST\_DIR} \ -pthread -c ${GTEST\_DIR}/src/gtest-all.cc

ar -rv libgtest.a gtest-all.o

(Google Test, iş parçacığı kullandığı için -pthread'e ihtiyacımız var.)  
  
Ardından, test kaynak dosyamızı sistem başlığı arama yoluna $ {GTEST\_DIR}/include ile derlememiz ve onu gtest ve diğer gerekli tüm kütüphanelere bağlamamız gerekir:

g++ -I ${GTEST\_DIR}/include -pthread path/to/your\_test.cc libgtest.a \

-o senin\_testin  
  
Örnek olarak make/ dizini, GNU make'in bulunduğu sistemlerde Google Testi'ni oluşturmak için kullanabileceğiniz bir Makefile'ı içerir (ör. Linux, Mac OS X ve Cygwin). Google Test'in kendi testlerini yapmaya çalışmaz. Bunun yerine, sadece Google Test kitaplığı ve örnek bir test oluşturur. Kendi yapı komut dosyamız için bir başlangıç noktası olarak kullanabiliriz.  
  
Ortamımız için varsayılan ayarlar doğruysa, aşağıdaki komutlar başarılı olmalıdır:  
  
cd ${GTEST\_DIR}/make  
make  
./ornek1\_unittest  
  
Hatalar görürseniz make/Makefile dosyalarının içeriğini değiştirerek onları uzaklaştırmayı deneyebiliriz. make/Makefile'da bunun nasıl yapılacağı ile ilgili talimatlar vardır.

**Otomatik Kurulum**

Google Test ile Visual Studio’da çalışmak mümkün fakat bu gtest’i projemize nasıl dahil etmeliyiz ? Bunun için iki yöntemimiz mevcut olsa da her zaman için en iyi yol manuel olan yoldur.

* Google Test Runner (VS 2015’e kadar desktekler)
  + NuGet Packages ile projeye dahil edilir
* Google Test Adapter (VS 2017 destekler)
  + Araçlar > Uzantılar ve Güncelleştirmeler aracılığıyla
  + NuGet Packages ile projeye dahil edilir

**Manuel Kurulum**

* [GitHub](https://github.com) aracılığıyla [Google Test Framework](https://github.com/google/googletest)’ü bilgisayara indiriyoruz
* googletest dosyasını kütüphane olarak derliyoruz. Bunun için VS’da;
  + Dosya>Yeni>Proje>Win32Projesi’ni seçip adına googletest diyoruz.
  + Açılan pencerede ‘İleri’ diyerek Statik Kütüphane’yi işaretliyoruz.
  + Önceden Derlenmiş Üstbilgi’nin işaretli olma**ma**sına dikkat ederek Son diyoruz.
* Var olan projenize ekleyecekseniz Dosya>Yeni>Proje yerine Çözüm Gezgininde yer alan projenize sağ tıklayarak Yeni>Proje biçiminde ilerlemeniz gerekiyor.
* Oluşan googletest projesine sağ tıklayarak Özellikler(Alt+Enter) penceresini açıyoruz.
  + googletest Özellik Sayfaları altında VC++ Dizinleri içerisinde Ekleme Kodu Dizinleri sekmesine gelerek, C:\Users\kullaniciadi\İndirilenler\googletest-master\googletest

ve C:\Users\kullaniciadi\İndirilenler\googletest-master\googletest\include ekliyoruz.

* Çözüm Gezgininde yer alan googletest projesine sağ tıklayıp Ekle>Var Olan Öğe(Shift+Alt+A)

C:\Users\kullaniciadi\İndirilenler\googletest-master\googletest\src\ dizini içinden iki dosya ekliyoruz. Bunlar; gtest-all.cc ce gtest\_main.cc . Artık googletest projesine sağ tıklayıp Yapılandır diyebiliriz. Sonuç başarılı olacaktır.

* Unit Test projesi oluşturmamız gerekiyor ve bunun için;
  + Çözüm Gezgininde yer alan projemize sağ tıklayıp Yeni>Proje>Win32Projesi’ni seçip adına Google\_Unit\_Test\_Project diyoruz.
  + Açılan pencerede ‘İleri’ diyerek Konsol Uygulaması’nı işaretliyoruz.
  + Önceden Derlenmiş Üstbilgi’nin işaretli olma**ma**sına dikkat ederek Son diyoruz.
  + stdafx.h – targetver.h ve stdafx.cpp dosyalarını silebiliriz. Projemize bir etkisi yok.
* Google\_Unit\_Test\_Project’e sağ tıklayıp Özellikler(Alt-Enter)’e giderek;
  + Yapılandırma Özellikleri içinde C/C++’ı seçerek Ek İçeren Dizinler kısmına,

C:\Users\oguzhanozgur\Desktop\googletest-master\googletest

C:\Users\oguzhanozgur\Desktop\googletest-master\googletest\include

C:\Users\kullaniciadi\Belgeler\Visual Studio 2017\Projects\Win32Projesi

Dizinlerini ekliyoruz.

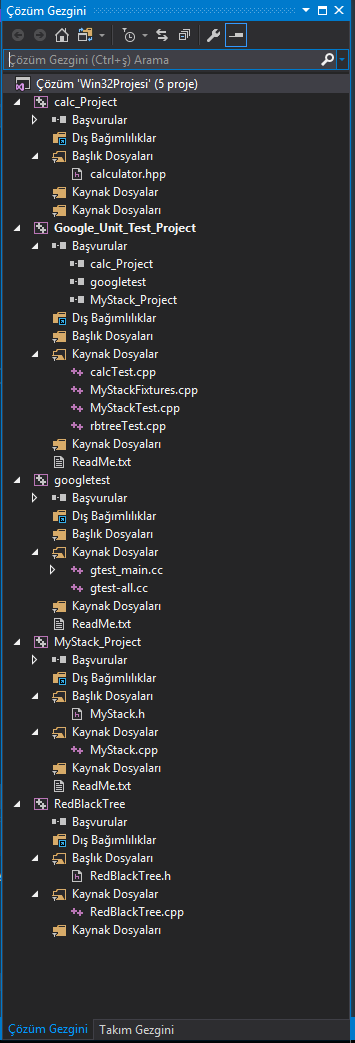
* Google\_Unit\_Test\_Project’e sağ tıklayıp Ekle > Başvuru’ya tıklıyoruz ve;
  + Projeler > Çözüm içinde yer alan googletest’i işaretliyoruz
  + Var olan projeye eklemişsek veya daha sonra test etmek üzere proje eklemişsek onları da seçerek Tamam diyoruz.

**Testler:**

* ASSERT\_XXX(): Hata oluşan yerde testi durdurur, geri kalan testleri çalıştırmaz
* EXPECT\_XXX(): Ölümcül olmayan hatalardır, hatayı gösterir ve teste devam eder

| **Test** | **Fatal** | **NonFatal** |
| --- | --- | --- |
| True | ASSERT\_TRUE(*condition*) | EXPECT\_TRUE(*condition*) |
| False | ASSERT\_FALSE(*condition*) | EXPECT\_FALSE(*condition*) |
| Equal | ASSERT\_EQ(*arg1*,*arg2*) | EXPECT\_EQ(*arg1*,*arg2*) |
| Not Equal | ASSERT\_NE(*arg1*,*arg2*) | EXPECT\_NE(*arg1*,*arg2*) |
| Less Than | ASSERT\_LT(*arg1*,*arg2*) | EXPECT\_LT(*arg1*,*arg2*) |
| Less Than or Equal | ASSERT\_LE(*arg1*,*arg2*) | EXPECT\_LE(*arg1*,*arg2*) |
| Greater Than | ASSERT\_GT(*arg1*,*arg2*) | EXPECT\_GT(*arg1*,*arg2*) |
| Greater Than or Equal | ASSERT\_GE(*arg1*,*arg2*) | EXPECT\_GE(*arg1*,*arg2*) |
| C String Equal | ASSERT\_STREQ(*str1*,*str2*) | EXPECT\_STREQ(*str1*,*str2*) |
| C String Not Equal | ASSERT\_STRNE(*str1*,*str2*) | EXPECT\_STRNE(*str1*,*str2*) |
| C String Case Equal | ASSERT\_STRCASEEQ(*str1*,*str2*) | EXPECT\_STRCASEEQ(*str1*,*str2*) |
| C String Case Not Equal | ASSERT\_STRCASENE(*str1*,*str2*) | EXPECT\_STRCASENE(*str1*,*str2*) |
| Verify that exception is thrown | ASSERT\_THROW(*statement*,*exception\_type*) | EXPECT\_THROW(*statement*,*exception\_type*) |
| Verify that exception is thrown | ASSERT\_ANY\_THROW(*statement*) | EXPECT\_ANY\_THROW(*statement*) |
| Verify that exception is NOT thrown | ASSERT\_NO\_THROW(*statement*) | EXPECT\_NO\_THROW(*statement*) |

**C++ Yazılım Testi Gerçeklemesi – Win32Projesi – Çözüm Gezgini**



**C++ Yazılım Testi Gerçeklemesi – Win32Projesi – calcProject**

calculator.hpp Başlık (Header) Dosyası içeriği

#ifndef CALCULATOR\_HPP

#define CALCULATOR\_HPP

#include <iostream>

namespace math {

enum {

ADD,

SUB,

MUL,

DIV,

MOD

};

class calculator

{

public:

calculator(){};

double eval(double \_val1, double \_val2, int \_op)

{

switch (\_op) {

case ADD:

return \_val1 + \_val2;

case SUB:

return \_val1 - \_val2;

case MUL:

return \_val1 \* \_val2;

case DIV:

if(\_val2 != 0)

return \_val1 / \_val2;

default:

std::cout << "Unsupported opereation";

};

}

};

}

#endif

**C++ Yazılım Testi Gerçeklemesi – Win32Projesi – Google\_Unit\_Test\_Project**

calcTest.cpp Test Kaynak Dosyası içeriği

#include "gtest/gtest.h"

#include "calc/calculator.hpp"

class testCalc : public ::testing::Test {

protected: //Gövde, public veya protected olarak tanımlanmalı, Private olamaz

void SetUp() { // Büyük 'U' ile yazılmalı

}

void TearDown() {

}

testCalc() {

}

~testCalc() {

}

math::calculator calc; //add, sub, mul, div kullanmak için tanımlandı

};

TEST\_F(testCalc, PositifToplamTest)

{

EXPECT\_EQ(8, calc.eval(3, 5, math::ADD));

EXPECT\_EQ(-2, calc.eval(3, 5, math::SUB));

EXPECT\_EQ(15, calc.eval(3, 5, math::MUL));

EXPECT\_EQ(0.6, calc.eval(3, 5, math::DIV));

}

TEST\_F(testCalc, NegatifToplamTest)

{

EXPECT\_EQ(-8, calc.eval(-3, -5, math::ADD));

EXPECT\_EQ(2, calc.eval(-3, -5, math::SUB));

EXPECT\_EQ(15, calc.eval(-3, -5, math::MUL));

EXPECT\_EQ(0.6, calc.eval(-3, -5, math::DIV));

}

**C++ Yazılım Testi Gerçeklemesi – Win32Projesi – Google\_Unit\_Test\_Project**

MyStackFixtures.cpp Test Kaynak Dosyası içeriği

#include "gtest\gtest.h"

#include "Win32Project2\MyStack.h"

class FooTest : public ::testing::Test {

protected: //Gövde, public veya protected olmalı void SetUp() { // Büyük 'U' ile yazılmalı

st.push(34);

st.push(26);

st.push(54);

}

void TearDown() {

}

FooTest() {

st.push(22);

}

~FooTest() {

}

MyStack st;

};

//Text with Fixture

TEST\_F(FooTest, fixtureTest) { //ilk argüman mutlaka sınıf adı olmalı

//Örneği kurmalıyız FooTest m; m.SetUp()

int val = st.pop(); //text fixture üyelerine artık buradan erişilebilir

EXPECT\_EQ(54, val);

EXPECT\_EQ(54, val) << "Bu değer 56 olmalı";

EXPECT\_EQ(54, val) << "Bu değer farklı olamaz" << val;

//m.TearDown();

}

//Text with Fixture

TEST\_F(FooTest, fixtureTest2) { int val = st.pop();

EXPECT\_EQ(54, val);

//m.TearDown();

}

**C++ Yazılım Testi Gerçeklemesi – Win32Projesi – Google\_Unit\_Test\_Project**

MyStackTest.cpp Test Kaynak Dosyası içeriği

#include "gtest\gtest.h"

#include "Win32Project2\MyStack.h"

class StackTest : public ::testing::Test {

protected: //Gövde, public veya protected olarak tanımlanmalı, Private olamaz

void SetUp() { // Büyük 'U' ile yazılmalı

}

void TearDown() {

}

StackTest() {

}

~StackTest() {

}

MyStack st;

};

TEST\_F(StackTest, simpleTest) { //test durumunun adı "StackTest" ve SimpleTest ile diğer testlerimiz (raporda da görebileceğimiz gibi) bu durumun içinde yer alır.

st.push(9); //9 değerini stack'e gönderiyoruz

EXPECT\_EQ(9, st.pop());

}

TEST\_F(StackTest, messageTest) {

st.push(28); //28 değerlerini stack'e gönderiyoruz

int val = st.pop(); //stack'e gönderilen son değeri val değişkenine çekiyoruz

EXPECT\_TRUE(val == 28) << "Tester Mesaji";

//val değişkeni verilen değere eşit mi testi & ekrana yazdırılan düzenlenebilir mesaj

//EXPECT\_TRUE(val != 28) << "Tester Mesaji";

}

TEST\_F(StackTest, nonfatalTest) {

st.push(9);

st.push(28); // sırayla 9 ve 28 değerlerini stack'e gönderiyoruz

int val = st.pop(); //stack'e gönderilen son değeri val değişkenine çekiyoruz

//Ölümcül Olmayan-Nonfatal assertion

EXPECT\_EQ(28, val); //val değişkeni verilen değere eşit mi testi

EXPECT\_NE(0, val); //değişkenin 0'a eşit olmadığının testi

EXPECT\_GT(29, val); //verilen değerin val değişkeninden daha büyük olduğunun testi

EXPECT\_LE(27, val); //verilen değerin val değişkeninden daha küçük olduğunun testi

EXPECT\_TRUE(val == 28);

EXPECT\_EQ(1, st.size());//stack içine gönderilen değer adeti testi

}

TEST\_F(StackTest, fatalTest) {

st.push(9);

st.push(28); // sırayla 9 ve 28 değerlerini stack'e gönderiyoruz

int val = st.pop(); //stack'e gönderilen son değeri val değişkenine çekiyoruz

//Ölümcül-Fatal assertion

ASSERT\_EQ(28, val); //val değişkeni verilen değere eşit mi ? testi

ASSERT\_NE(0, val); //0'a eşit olmadığının testi

ASSERT\_GT(29, val); //verilen değerin val değişkeninden daha büyük olduğunun testi

ASSERT\_LE(27, val); //verilen değerin val değişkeninden daha küçük olduğunun testi

ASSERT\_TRUE(val == 28);

}

TEST\_F(StackTest, stringTest) {

st.push(9);

st.push(28); // sırayla 9 ve 28 değerlerini val stack'e gönderiyoruz

int val = st.pop(); //stack'e gönderilen son değeri val değişkenine çekiyoruz

//String Check (String Kontrolü)

EXPECT\_STREQ("9 ", st.toString().c\_str());

EXPECT\_STRCASEEQ("9 ", st.toString().c\_str()); //Durumu göz ardı ederek

}

TEST\_F(StackTest, floatTest) {

st.push(9);

st.push(28); //sırayla 9 ve 28 değerlerini val stack'e gönderiyoruz

int val = st.pop(); //stack'e gönderilen son değeri val değişkenine çekiyoruz

//Floatin-point Comparison (Floating-point Karşılaştırması)

float x, y; //if (x==y)

EXPECT\_FLOAT\_EQ(7.00000, ((float)val) / 4);

EXPECT\_NEAR(6.0, ((float)val) / 5, 1);

}

TEST\_F(StackTest, doubleTest) {

// MyStack st;

st.push(9);

st.push(28); //sırayla 9 ve 28 değerlerini val stack'e gönderiyoruz

int val = st.pop(); //stack'e gönderilen son değeri val değişkenine çekiyoruz

//Floatin-point Comparison (Floating-point Karşılaştırması)

float x, y; //if (x==y)

EXPECT\_DOUBLE\_EQ(7.0000000000000, ((double)val) / 4);

EXPECT\_NEAR(6.0, ((float)val) / 5, 1);

}

**C++ Yazılım Testi Gerçeklemesi – Win32Projesi – Google\_Unit\_Test\_Project**

redblackTest.cpp Test Kaynak Dosyası içeriği

#include "gtest/gtest.h"

#include "RedBlackTree\RedBlackTree.h"

class redblackTest : public ::testing::Test {

protected: //Gövde, public veya protected olarak tanımlanmalı, Private olamaz

void SetUp() {// Büyük 'U' ile yazılmalı

}

void TearDown() {

}

redblackTest() {

}

~redblackTest() {

}

RBtree obj; //insert, search, del ve size kullanmak için tanımlandı

};

TEST\_F(redblackTest, searchTest)

{

obj.insert(6);

obj.insert(7);

obj.insert(9);

EXPECT\_EQ(true, obj.search(6));

EXPECT\_EQ(false, obj.search(5)) << "Agacta 7 degeri bulunmakta";

}

TEST\_F(redblackTest, insertTest1)

{

EXPECT\_EQ(true, obj.insert(6));

}

TEST\_F(redblackTest, insertTest2)

{

obj.insert(6);

obj.insert(7);

EXPECT\_EQ(2, obj.size());

}

TEST\_F(redblackTest, delTest1)

{

obj.insert(6);

obj.insert(7);

EXPECT\_EQ(true, obj.del(7));

}

TEST\_F(redblackTest, delTest2)

{

obj.insert(6);

obj.insert(7);

obj.insert(8);

obj.del(6);

EXPECT\_EQ(2, obj.size());

}

**C++ Yazılım Testi Gerçeklemesi – Win32Projesi – MyStack\_Project**

MyStack.h Başlık (Header) Dosyası içeriği

#pragma once

#include <vector>

class MyStack

{

std::vector<int> \_v;

public:

MyStack(void){}

~MyStack(void){}

void push(int);

int pop();

size\_t size();

std::string toString();

};

MyStack.cpp Kaynak Dosyası içeriği

#include "MyStack.h"

#include <sstream>

void MyStack::push(int data) {

this->\_v.push\_back(data);

}

int MyStack::pop() {

int ret = \_v.back();

\_v.pop\_back();

return ret;

}

size\_t MyStack::size()

{

return \_v.size();

}

std::string MyStack::toString() {

std::string ret = "";

std::stringstream sm;

for (size\_t i=0; i < \_v.size(); i++) {

sm << \_v[i] << " ";

}

return sm.str();

}

**C++ Yazılım Testi Gerçeklemesi – Win32Projesi – RedBlackTree\_Project**

RedBlackTree.cpp Kaynak Dosyası içeriği

#include <iostream>

using namespace std;

struct node

{

int key;

node \*parent;

char color;

node \*left;

node \*right;

};

class RBtree

{

node \*root;

node \*q;

public:

RBtree()

{

q = NULL;

root = NULL;

}

void insert();

void insertfix(node \*);

void leftrotate(node \*);

void rightrotate(node \*);

void del();

node\* successor(node \*);

void delfix(node \*);

void disp();

void display(node \*);

void search();

};

void RBtree::insert()

{

int z, i = 0;

cout << "\nEnter key of the node to be inserted: ";

cin >> z;

node \*p, \*q;

node \*t = new node;

t->key = z;

t->left = NULL;

t->right = NULL;

t->color = 'r';

p = root;

q = NULL;

if (root == NULL)

{

root = t;

t->parent = NULL;

}

else

{

while (p != NULL)

{

q = p;

if (p->key<t->key)

p = p->right;

else

p = p->left;

}

t->parent = q;

if (q->key<t->key)

q->right = t;

else

q->left = t;

}

insertfix(t);

}

void RBtree::insertfix(node \*t)

{

node \*u;

if (root == t)

{

t->color = 'b';

return;

}

while (t->parent != NULL&&t->parent->color == 'r')

{

node \*g = t->parent->parent;

if (g->left == t->parent)

{

if (g->right != NULL)

{

u = g->right;

if (u->color == 'r')

{

t->parent->color = 'b';

u->color = 'b';

g->color = 'r';

t = g;

}

}

else

{

if (t->parent->right == t)

{

t = t->parent;

leftrotate(t);

}

t->parent->color = 'b';

g->color = 'r';

rightrotate(g);

}

}

else

{

if (g->left != NULL)

{

u = g->left;

if (u->color == 'r')

{

t->parent->color = 'b';

u->color = 'b';

g->color = 'r';

t = g;

}

}

else

{

if (t->parent->left == t)

{

t = t->parent;

rightrotate(t);

}

t->parent->color = 'b';

g->color = 'r';

leftrotate(g);

}

}

root->color = 'b';

}

}

void RBtree::del()

{

if (root == NULL)

{

cout << "\nEmpty Tree.";

return;

}

int x;

cout << "\nEnter the key of the node to be deleted: ";

cin >> x;

node \*p;

p = root;

node \*y = NULL;

node \*q = NULL;

int found = 0;

while (p != NULL&&found == 0)

{

if (p->key == x)

found = 1;

if (found == 0)

{

if (p->key<x)

p = p->right;

else

p = p->left;

}

}

if (found == 0)

{

cout << "\nElement Not Found.";

return;

}

else

{

cout << "\nDeleted Element: " << p->key;

cout << "\nColour: ";

if (p->color == 'b')

cout << "Black\n";

else

cout << "Red\n";

if (p->parent != NULL)

cout << "\nParent: " << p->parent->key;

else

cout << "\nThere is no parent of the node. ";

if (p->right != NULL)

cout << "\nRight Child: " << p->right->key;

else

cout << "\nThere is no right child of the node. ";

if (p->left != NULL)

cout << "\nLeft Child: " << p->left->key;

else

cout << "\nThere is no left child of the node. ";

cout << "\nNode Deleted.";

if (p->left == NULL || p->right == NULL)

y = p;

else

y = successor(p);

if (y->left != NULL)

q = y->left;

else

{

if (y->right != NULL)

q = y->right;

else

q = NULL;

}

if (q != NULL)

q->parent = y->parent;

if (y->parent == NULL)

root = q;

else

{

if (y == y->parent->left)

y->parent->left = q;

else

y->parent->right = q;

}

if (y != p)

{

p->color = y->color;

p->key = y->key;

}

if (y->color == 'b')

delfix(q);

}

}

void RBtree::delfix(node \*p)

{

node \*s;

while (p != root&&p->color == 'b')

{

if (p->parent->left == p)

{

s = p->parent->right;

if (s->color == 'r')

{

s->color = 'b';

p->parent->color = 'r';

leftrotate(p->parent);

s = p->parent->right;

}

if (s->right->color == 'b'&&s->left->color == 'b')

{

s->color = 'r';

p = p->parent;

}

else

{

if (s->right->color == 'b')

{

s->left->color == 'b';

s->color = 'r';

rightrotate(s);

s = p->parent->right;

}

s->color = p->parent->color;

p->parent->color = 'b';

s->right->color = 'b';

leftrotate(p->parent);

p = root;

}

}

else

{

s = p->parent->left;

if (s->color == 'r')

{

s->color = 'b';

p->parent->color = 'r';

rightrotate(p->parent);

s = p->parent->left;

}

if (s->left->color == 'b'&&s->right->color == 'b')

{

s->color = 'r';

p = p->parent;

}

else

{

if (s->left->color == 'b')

{

s->right->color = 'b';

s->color = 'r';

leftrotate(s);

s = p->parent->left;

}

s->color = p->parent->color;

p->parent->color = 'b';

s->left->color = 'b';

rightrotate(p->parent);

p = root;

}

}

p->color = 'b';

root->color = 'b';

}

}

void RBtree::leftrotate(node \*p)

{

if (p->right == NULL)

return;

else

{

node \*y = p->right;

if (y->left != NULL)

{

p->right = y->left;

y->left->parent = p;

}

else

p->right = NULL;

if (p->parent != NULL)

y->parent = p->parent;

if (p->parent == NULL)

root = y;

else

{

if (p == p->parent->left)

p->parent->left = y;

else

p->parent->right = y;

}

y->left = p;

p->parent = y;

}

}

void RBtree::rightrotate(node \*p)

{

if (p->left == NULL)

return;

else

{

node \*y = p->left;

if (y->right != NULL)

{

p->left = y->right;

y->right->parent = p;

}

else

p->left = NULL;

if (p->parent != NULL)

y->parent = p->parent;

if (p->parent == NULL)

root = y;

else

{

if (p == p->parent->left)

p->parent->left = y;

else

p->parent->right = y;

}

y->right = p;

p->parent = y;

}

}

node\* RBtree::successor(node \*p)

{

node \*y = NULL;

if (p->left != NULL)

{

y = p->left;

while (y->right != NULL)

y = y->right;

}

else

{

y = p->right;

while (y->left != NULL)

y = y->left;

}

return y;

}

void RBtree::disp()

{

display(root);

}

void RBtree::display(node \*p)

{

if (root == NULL)

{

cout << "\nEmpty Tree.";

return;

}

if (p != NULL)

{

cout << "\n\t NODE: ";

cout << "\n Key: " << p->key;

cout << "\n Colour: ";

if (p->color == 'b')

cout << "Black";

else

cout << "Red";

if (p->parent != NULL)

cout << "\n Parent: " << p->parent->key;

else

cout << "\n There is no parent of the node. ";

if (p->right != NULL)

cout << "\n Right Child: " << p->right->key;

else

cout << "\n There is no right child of the node. ";

if (p->left != NULL)

cout << "\n Left Child: " << p->left->key;

else

cout << "\n There is no left child of the node. ";

cout << endl;

if (p->left)

{

cout << "\n\nLeft:\n";

display(p->left);

}

/\*else

cout<<"\nNo Left Child.\n";\*/

if (p->right)

{

cout << "\n\nRight:\n";

display(p->right);

}

/\*else

cout<<"\nNo Right Child.\n"\*/

}

}

void RBtree::search()

{

if (root == NULL)

{

cout << "\nEmpty Tree\n";

return;

}

int x;

cout << "\n Enter key of the node to be searched: ";

cin >> x;

node \*p = root;

int found = 0;

while (p != NULL&& found == 0)

{

if (p->key == x)

found = 1;

if (found == 0)

{

if (p->key<x)

p = p->right;

else

p = p->left;

}

}

if (found == 0)

cout << "\nElement Not Found.";

else

{

cout << "\n\t FOUND NODE: ";

cout << "\n Key: " << p->key;

cout << "\n Colour: ";

if (p->color == 'b')

cout << "Black";

else

cout << "Red";

if (p->parent != NULL)

cout << "\n Parent: " << p->parent->key;

else

cout << "\n There is no parent of the node. ";

if (p->right != NULL)

cout << "\n Right Child: " << p->right->key;

else

cout << "\n There is no right child of the node. ";

if (p->left != NULL)

cout << "\n Left Child: " << p->left->key;

else

cout << "\n There is no left child of the node. ";

cout << endl;

}

}

int main()

{

int ch, y = 0;

RBtree obj;

do

{

cout << "\n\t RED BLACK TREE ";

cout << "\n 1. Insert in the tree ";

cout << "\n 2. Delete a node from the tree";

cout << "\n 3. Search for an element in the tree";

cout << "\n 4. Display the tree ";

cout << "\n 5. Exit ";

cout << "\nEnter Your Choice: ";

cin >> ch;

switch (ch)

{

case 1: obj.insert();

cout << "\nNode Inserted.\n";

break;

case 2: obj.del();

break;

case 3: obj.search();

break;

case 4: obj.disp();

break;

case 5: y = 1;

break;

default: cout << "\nEnter a Valid Choice.";

}

cout << endl;

} while (y != 1);

return 1;

}

**C++ Yazılım Testi Gerçeklemesi**

**Win32Projesi**

Yapılandırma Çıkış verileri içeriği

1>------ Derleme başladı: Proje: MyStack\_Project, Yapılandırma: Debug Win32 ------

2>------ Derleme başladı: Proje: calc\_Project, Yapılandırma: Debug Win32 ------

1>C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\Common7\IDE\VC\VCTargets\Microsoft.CppBuild.targets(387,5): warning MSB8028: Ara dizin (Debug\) başka bir projeden (Win32Project2.vcxproj) paylaşılan dosyalar içeriyor. Bu, hatalı temizleme ve yeniden oluşturma davranışına neden olabilir.

2>C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\Common7\IDE\VC\VCTargets\Microsoft.CppBuild.targets(387,5): warning MSB8028: Ara dizin (Debug\) başka bir projeden (calc.vcxproj) paylaşılan dosyalar içeriyor. Bu, hatalı temizleme ve yeniden oluşturma davranışına neden olabilir.

1>Win32Project2.vcxproj -> C:\Users\oguzhanozgur\documents\visual studio 2017\Projects\Win32Project2\Debug\MyStack\_Project.lib

2>"calc.vcxproj" projesini oluşturma tamamlandı.

1>"Win32Project2.vcxproj" projesini oluşturma tamamlandı.

3>------ Derleme başladı: Proje: RedBlackTree\_Project, Yapılandırma: Debug Win32 ------

3>C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\Common7\IDE\VC\VCTargets\Microsoft.CppBuild.targets(387,5): warning MSB8028: Ara dizin (Debug\) başka bir projeden (RedBlackTree.vcxproj) paylaşılan dosyalar içeriyor. Bu, hatalı temizleme ve yeniden oluşturma davranışına neden olabilir.

3>Atlanıyor... (ilgili hiçbir değişiklik algılanmadı)

3>RedBlackTree.cpp

3>RedBlackTree.vcxproj -> C:\Users\oguzhanozgur\documents\visual studio 2017\Projects\Win32Project2\Debug\RedBlackTree\_Project.exe

3>RedBlackTree.vcxproj -> C:\Users\oguzhanozgur\documents\visual studio 2017\Projects\Win32Project2\Debug\RedBlackTree\_Project.pdb (Partial PDB)

3>"RedBlackTree.vcxproj" projesini oluşturma tamamlandı.

========== Oluşturma: 3 başarılı, 0 başarısız, 2 güncel, 0 atlandı ==========

**Google\_Unit\_Test\_Project**

Yapılandırma Çıkış verileri içeriği

1>------ Derleme başladı: Proje: MyStack\_Project, Yapılandırma: Debug Win32 ------

2>------ Derleme başladı: Proje: calc\_Project, Yapılandırma: Debug Win32 ------

2>C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\Common7\IDE\VC\VCTargets\Microsoft.CppBuild.targets(387,5): warning MSB8028: Ara dizin (Debug\) başka bir projeden (calc.vcxproj) paylaşılan dosyalar içeriyor. Bu, hatalı temizleme ve yeniden oluşturma davranışına neden olabilir.

2>"calc.vcxproj" projesini oluşturma tamamlandı.

1>C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\Common7\IDE\VC\VCTargets\Microsoft.CppBuild.targets(387,5): warning MSB8028: Ara dizin (Debug\) başka bir projeden (Win32Project2.vcxproj) paylaşılan dosyalar içeriyor. Bu, hatalı temizleme ve yeniden oluşturma davranışına neden olabilir.

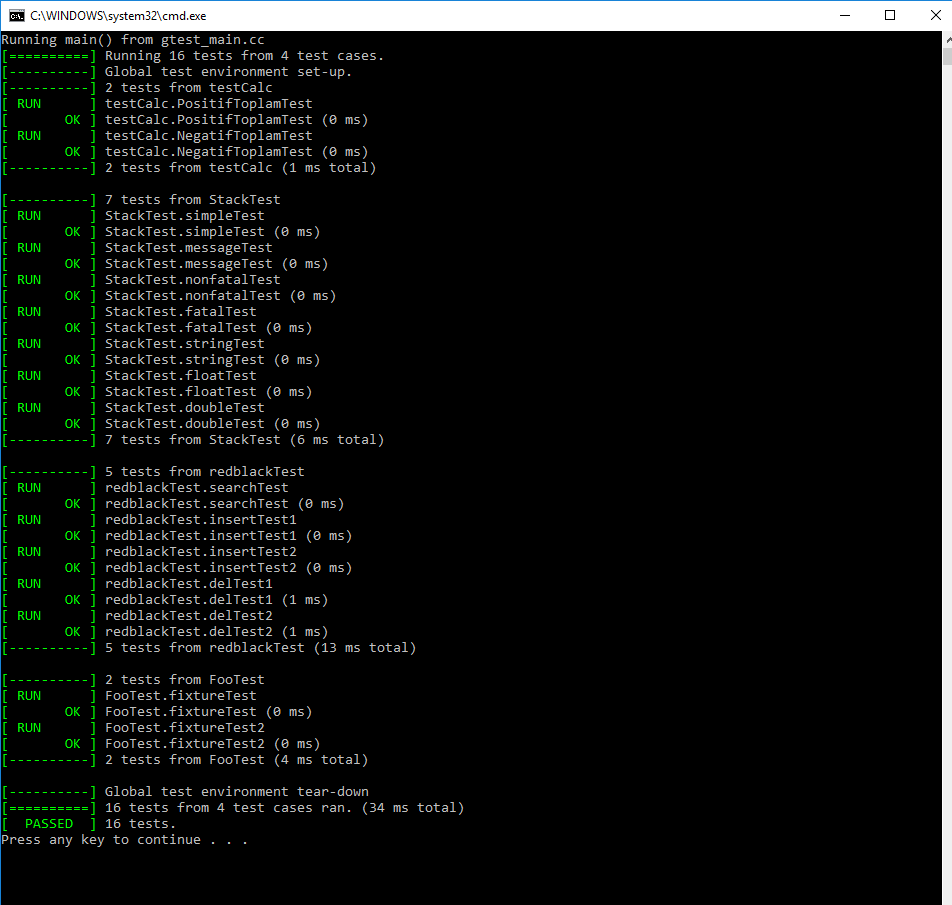
1>Win32Project2.vcxproj -> C:\Users\oguzhanozgur\documents\visual studio 2017\Projects\Win32Project2\Debug\MyStack\_Project.lib

1>"Win32Project2.vcxproj" projesini oluşturma tamamlandı.

========== Oluşturma: 2 başarılı, 0 başarısız, 2 güncel, 0 atlandı ==========

**Google\_Unit\_Test\_Project**

Gtest Terminal Raporu içeriği



**KAYNAKÇA**

1. <https://www.ibm.com/developerworks/aix/library/au-googletestingframework.html>
2. <https://en.wikipedia.org/wiki/Google_Test>
3. <https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks#C.2B.2B>
4. <https://www.slideshare.net/andreafrancia/google-c-testing-framework-in-visual-studio-2008>
5. <https://www.slideshare.net/hmarchezi/c-unit-test-with-google-testing-framework>
6. <https://www.slideshare.net/AbnerChihYiHuang/googletestframeworkpublic-140708010931phpapp01>
7. <https://www.slideshare.net/BunnyStupid/test-driven-development-and-unit-testing-with-examples-in-c>
8. <http://stackoverflow.com/questions/2565299/using-assert-and-expect-in-googletest>
9. <https://github.com/google/googletest/blob/master/googletest/docs/AdvancedGuide.md>
10. <https://github.com/google/googletest>
11. <http://www.coders-hub.com/2015/07/red-black-tree-rb-tree-using-c.html>
12. <boqian.weebly.com/c-programming.html>