**T.C.**

**SAKARYA ÜNİVERSİTESİ**

**BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ**

BSM 498 BİTİRME ÇALIŞMASI

**GOOGLE TEST ÇATISI İLE C++ BİRİM TESTİ**

GERÇEKLEMESİ

U161210071 Oğuzhan İNCE

U161210070 Savaş KAPLAN

|  |  |  |
| --- | --- | --- |
| Fakülte Anabilim Dalı  Tez Danışmanı | :  : | BİLGİSAYAR MÜHENDİSLİĞİ  Arş. Gör. M. Fatih ADAK |

2016-2017 Bahar Dönemi

T.C.

SAKARYA ÜNİVERSİTESİ

**BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ**

GOOGLE TEST ÇATISI İLE C++ BİRİM TESTİ

GERÇEKLEMESİ

BSM 498 - BİTİRME ÇALIŞMASI

Oğuzhan İNCE, Savaş KAPLAN

|  |  |  |
| --- | --- | --- |
| Fakülte Anabilim Dalı | : | BİLGİSAYAR MÜHENDİSLİĞİ |

Bu tez .. / .. / … tarihinde aşağıdaki jüri tarafından oybirliği / oyçokluğu ile kabul edilmiştir.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| ………………. | ………………. | ………………. |
| Jüri Başkanı | Üye | Üye |

**ÖNSÖZ**

Yazılımlar, tek başlarına değilde bir yazılım eko sistemi halinde birbirleri ile entegre olarak üretilmesi, yazılım projelerinde **kalite** kelimesinin daha da altını çizilir hale getirmiştir. Kaliteyi sağlamak için Yazılım Testi olarak genel bir kategoride;  yazılımın dökümantasyon ve tasarım aşamasından başlayarak hiçkimse tarafından kullanılmayacak hale gelmesi yada genellikle desteğin çekilmesi yani yeni bir sürümünün çıkarılmayacağı duyuruluncaya kadar devam eder. Yazılım Testi; yazılımın olabildiğince en az kusur ve hata ile üretilip kullanıcıya sunulmasını amaçlayan işlemlerin bütünüdür diyebiliriz. Burada yapılan C++ programlama dili ile yazılmış bir takım yazılımların google test çatısını kullanarak bir yazılım testi neden ve nasıl gerçekleştirilir, test sonuçları nasıl yorumlanır ve yazılım testine neden ihtiyaç duyarız gibi sorulara cevap vermektir. Şimdiye kadar fazla önemsenmeyen fakat günümüzde yazılım test mühendisliği ünvanı ile çalışan yüzlerce insanın yaptıkları işin önemini ve gerekliliğini de kavrayacağız.

**İÇİNDEKİLER**

|  |  |
| --- | --- |
| ÖNSÖZ……...................................................................................................... | iii |
| İÇİNDEKİLER.................................................................................................. | iv |
| SİMGELER VE KISALTMALAR LİSTESİ.................................................... | vii |
| ŞEKİLLER LİSTESİ......................................................................................... | vii |
| ÖZET................................................................................................................. | ix |
|  |  |
|  |  |
| BÖLÜM 1. |  |
| GİRİŞ................................................................................................................ | 1 |
| * 1. Yazılım Testi ................................................................................... | 1 |
| 1.2. Neden ve Gereklilikleri .................................................................. | 1 |
|  |  |
| BÖLÜM 2. |  |
| YAZILIM TEST SÜREÇLERİ........................................................................ | 2 |
| 2.1. Hazırlık Süreci................................................................................. | 2 |
| 2.2. Dinamik Test Süreci........................................................................ | 3 |
| 2.3. Birim Testi....................................................................................... | 3 |
| 2.4. Tümleyim Testi................................................................................ | 4 |
| 2.4. Regresyon Testi................................................................................ | 4 |
| 2.4. Zorlanım – Performans Testi............................................................ | 4 |
| 2.4. Kullanıcı Kabul Testi........................................................................ | 5 |
| 2.4. Beyaz Kutu Test Tekniği.................................................................. | 5 |
| 2.4. Kara Kutu Test Tekniği.................................................................... | 5 |
| 2.4. Testin Sonlandırılması...................................................................... | 5 |
|  |  |
| BÖLÜM 3. |  |
| BİRİM TESTİ.................................................................................................... | 6 |
| 3.1. Code Coverage.................................................................................. | 8 |
|  |  |
| BÖLÜM 4. |  |
| C++ YAZILIM TESTİ....................................................................................... | 9 |
| 4.1. Google C++ Birim Testi Çatısı......................................................... | 9 |
| 4.1. Google C++ Birim Testi Çatısı İçin Gereklilikler............................ | 10 |
| 4.2.1. Linux Gereklilikleri................................................................ | 10 |
| 4.2.2. Windows Gereklilikleri .................….................................... | 10 |
| 4.2.3. Cygwin Gereklilikleri............................................................. | 10 |
| 4.2.3. Mac OS Gereklilikleri............................................................. | 10 |
| 4.2.3. Katkıda Bulunan Kişiler İçin Gereklilikler............................. | 10 |
| 4.3. Kurulum Yöntemleri......................................................................... | 11 |
| 4.2.3. Otomatik Kurulum.................................................................. | 11 |
| 4.2.3. Manuel Kurulum..................................................................... | 12 |
| 4.3. Test Yöntemleri................................................................................ | 14 |
|  |  |
| BÖLÜM 5. |  |
| C++ BİRİM TESTİ GERÇEKLEMESİ...........……….......………………...... | 15 |
| 5.1. Google Birim Testi VS Projesi......................................................... | 10 |
| 5.1.1. Ekrana Yazdırma Testi........................................................... | 10 |
| 5.1.2. Dört İşlem Hesaplama Testi................................................... | 10 |
| 5.1.3. Text Fixture Testi................................................................... | 10 |
| 5.1.4. Stack Testi.............................................................................. | 10 |
| 5.1.4.1. Stack Test Senaryoları............................................... | 10 |
| 5.1.5. Kırmızı Siyah Ağaç Testi....................................................... | 10 |
| 5.1.5.1. Kırmızı Siyah Ağaç Test Senaryoları........................ | 10 |
| BÖLÜM 6. |  |
| SONUÇLAR...........................………….…………………………………...... | 12 |
|  |  |
|  |  |
| KAYNAKLAR……………………………………………………………….. | 13 |
| EK A………………………………………………………………………….. | 14 |
| ÖZGEÇMİŞ……………………………………………….………………….. | 16 |
|  |  |
|  |  |
| BSM 498 BİTİRME ÇALIŞMASI DEĞERLENDİRME VE SÖZLÜ SINAV TUTANAĞI………………………………………………………………… | 17 |

SİMGELER VE KISALTMALAR LİSTESİ

|  |  |
| --- | --- |
| Gtest  SK  VS  MS  RBTree | : Google Birim Testi Çatısı  : Statik Kütüphane  : Microsoft Visual Studio 2017  : Microsoft  : Kırmızı Siyah Ağaç |

**ŞEKİLLER LİSTESİ**

|  |  |  |
| --- | --- | --- |
| Şekil 1.1. | Yazılım Testi Planlama..........……………………………... | 1 |
| Şekil 2.1. | Yazılım Testi Süreci Şeması..……………………………... | 2 |
| Şekil 3.1. | Birim Testi Gerçekleştirilecek olan hesaplama.hpp.............. | 7 |
| Şekil 3.2. | hesaplama.hpp Birim Testi Kodları…...…………………... | 7 |
| Şekil 4.1. | Code Coverage rapor gösterimi......………………………... | 8 |
| Şekil 4.2. | Code Coverage %’lik rapor gösterimi..........……..........…... | 8 |
| Şekil 5.1. | NuGet Paket Yönetimi...........……………………………... | 11 |
| Şekil 5.2. | Gtest Statik Kütüphanesi........……………………………... | 13 |
| Şekil 5.3. | Win32Projesi Çözüm Gezgini Gösterimi.......……………... | 15 |
| Şekil 5.4. | Oluşturulan Test Senaryoları..........………………………... | 15 |
| Şekil 5.4.1. | Dört İşlem Hesaplama Senaryosu..............………………... | 16 |
| Şekil 5.4.2. | Stack Test Senaryoları............……………………………... | 16 |
| Şekil 5.4.3. | Kırmızı Siyah Ağac Test Senaryoları.................................... | 16 |
| Şekil 5.4.4. | OutputTest.cpp dosyası gösterimi..........…………………... | 17 |
| Şekil 5.4.5. | calculator.hpp ve calcTest.cpp dosyası.................................. | 17 |
| Şekil 5.4.6. | Text Fixture Testi’nin gösterimi........……………………... | 18 |
| Şekil 5.5. | MyStack.cpp ve MyStack.h dosyaları........………………... | 19 |
| Şekil 5.6. | StackTest.cpp üzerinde sınıf gösterimi................................. | 20 |
| Şekil 5.7. | Push, top, pop, fatal ve nonFatal teslerinin gösterimi……... | 20 |
| Şekil 5.8. | String, float ve double teslerinin gösterimi....……………... | 21 |
| Şekil 6.1. | Red Black Tree kodlarının ekran çıktısı.....................……... | 22 |
| Şekil 6.2. | RBTree dengelenmeden önceki hali gösterimi......………... | 22 |
| Şekil 6.3. | RBTree dengelenmeden sonraki hali gösterimi......……...... | 22 |
| Şekil 6.4. | redblackTest.cpp dosyası sınıf gösterimi.......................…... | 23 |
| Şekil 6.5. | redblackTest.cpp dosyası test case’leri gösterimi..........…... | 23 |
| Şekil 7.1. | Tüm test senaryolarının ekran çıktısı gösterimi.................... | 24 |

**ÖZET**

Anahtar kelimeler: Yazılım Testi, Google Test, C, C++, Birim Testi

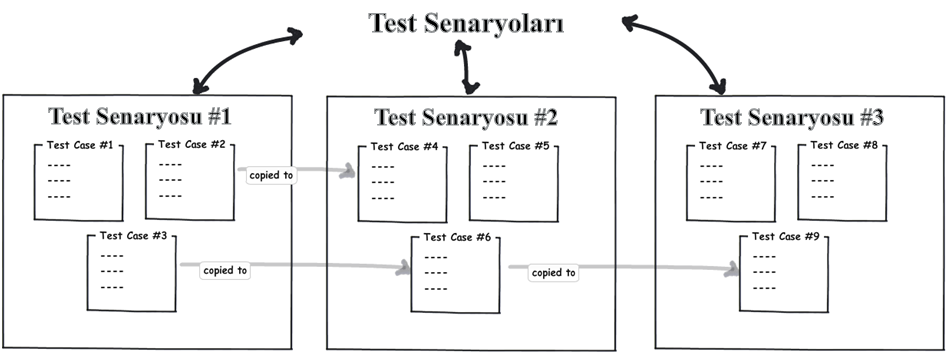
Yazılımlar, tek başlarına değilde bir yazılım eko sistemi halinde birbirleri ile entegre olarak üretilmesi, yazılım projelerinde kalite kelimesinin daha da altını çizilir hale getirmiştir. Kaliteyi sağlamak için Yazılım Testi olarak genel bir kategoride;  yazılımın dökümantasyon ve tasarım aşamasından başlayarak hiçkimse tarafından kullanılmayacak hale gelmesi yada genellikle desteğin çekilmesi yani yeni bir sürümünün çıkarılmayacağı duyuruluncaya kadar devam eder. Yazılım Testi; yazılımın olabildiğince en az kusur ve hata ile üretilip kullanıcıya sunulmasını amaçlayan işlemlerin bütünüdür diyebiliriz. Burada yapılan C++ programlama dili ile yazılmış bir takım yazılımların google test çatısını kullanarak bir yazılım testi neden ve nasıl gerçekleştirilir, test sonuçları nasıl yorumlanır ve yazılım testine neden ihtiyaç duyarız gibi sorulara cevap vermektir. Şimdiye kadar fazla önemsenmeyen fakat günümüzde yazılım test mühendisliği ünvanı ile çalışan yüzlerce insanın yaptıkları işin önemini ve gerekliliğini de kavrayacağız.

# GİRİŞ

* 1. Yazılım Testi

Test, bir sistemi manuel veya otomatik yollarla deneyerek veya değerlendirerek,

belirlenmiş gereksinimleri karşıladığının doğrulanması veya beklenen ile  gözlenen  sonuçlar arasındaki farkların belirlenmesisürecidir. 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. Bir düzeni ve mutlaka planlaması yapılan bir süreçtir. [1].

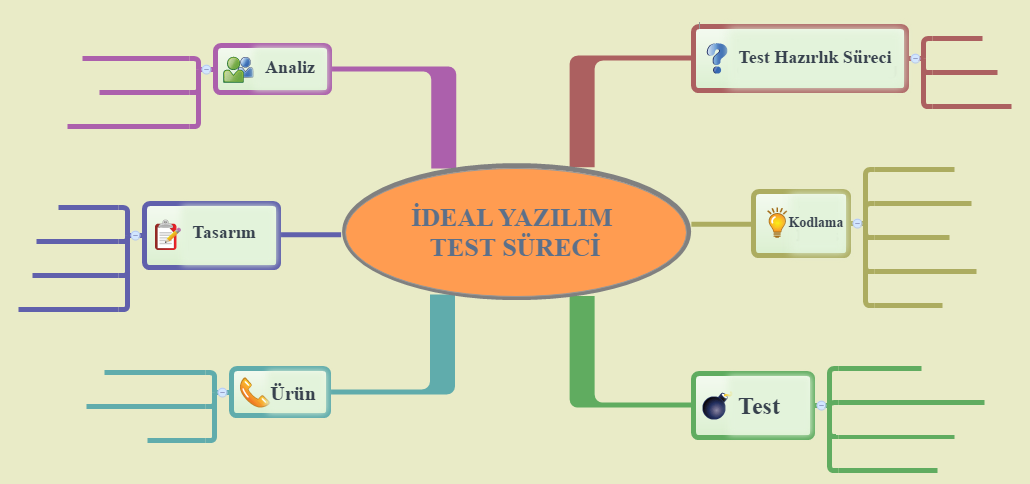


Şekil .1. Yazılım Testi Planlama

* 1. Neden ve Gereklilikleri
* 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.

# YAZILIM TEST SÜREÇLERİ

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.



Şekil 2.1. Yazılım Testi Süreci Şeması

* + 1. 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.
  + 1. 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 ne sırasıyla değinelim.

* + - 1. 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.

* + - 1. 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.

* + - 1. 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.

* + - 1. 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.

* + - 1. 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.

* + - 1. 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. Yazılımkodundaki deyimler, akış denetimleri, koşullar vb. elemanlar sınanır.

* + - 1. 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.

* + 1. 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.

# BİRİM TESTİ

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;

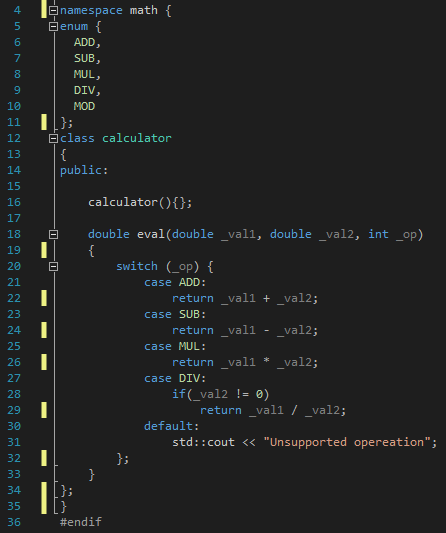
TEST\_METHOD(toplaTest)  {      int geriDonenDeger;      int beklenenDeger;      geriDonenDeger = topla( 3 + 5 );      beklenenDeger = 8;      Assert::AreEqual( beklenenDeger , geriDonenDeger );  }

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.

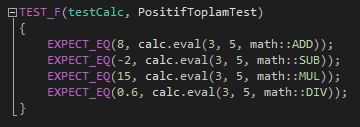
Sonuç olarak;

* 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.

Basit bir c++ dilinde hesaplama projesi ile birim testinin yazımı ve amacını pekiştirmek için eklenmiş olan örnek görseller Şekil 3.1. ve Şekil 3.2. deki gibidir.



Şekil 3.1. Birim Testi Gerçekleştirilecek Olan hesaplama.hpp Kodları



Şekil 3.2. hesaplama.hpp Birim Testi Kodları

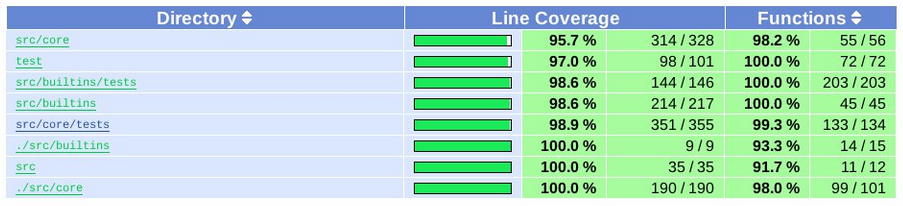
* + 1. Code Coverage

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.

Basit bir c++ dilinde birim testinin code coverage’ını hesaplamak için kullanılan lcov (gcc) programının raporları, code coverage’ın amacını pekiştirmek için eklenmiş olan örnek görseller Şekil 4.1. ve Şekil 4.2. deki gibidir.



Şekil 4.1. Code Coverage rapor gösterimi



Şekil 4.2. Code Coverage %’lik rapor gösterimi

**Tablolar Hakkında**-Tablolar tüm tez boyunca ortada olmalıdır. Tablolar metin dışına taşmamalıdır. Tablo açıklaması bir satır uzunluğundan küçükse ortalanır. Bir satır veya birden fazla iki yana yaslanır. Tablo açıklaması 9 punto Times New Roman ve tablonun üzerinde olmalıdır.

**Burayı çıktı almadan önce siliniz.**

# C++ YAZILIM TESTİ

C++ Programlama dili için birbirinden farklı test çatıları bulunmakta. Fakat c++ için genel olarak en uygun test yöntemi birim test ve birim test alanında en kullanışlı çatı google test olduğu için birim test gerçeklemesi için google test çatısı kullanıldı. C++ programlama dili test için geliştirilmiş bilinen test çatıları şunlardır;

* Google Test
* CppUnit
* Boost.Test
* CppUnitLite
* NanoCppUnit
* Unit++
* CxxTest
* Jungle
  1. Google C++ Birim Test Çatısı

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)

* 1. Google C++ Test Çatısı için Gereklilikler
     1. Linux Gereklilikleri
* GNU-compatible Make veya gmake
* POSIX-standard shell
* POSIX(-2) Regular Expressions (regex.h)
* A C++98-standard-compliant compiler
  + 1. Windows Gereklilikleri
* Microsoft Visual C++ v7.1 veya daha yenisi
  + 1. Cygwin Gereklilikleri
* Cygwin v1.5.25-14 veya daha yenisi
  + 1. Mac OS Gereklilikleri
* Mac OS X v10.4 Tiger veya daha yenisi
* Xcode Geliştirme Araçları
  + 1. Katkıda bulunan kişiler için gereklilikler
* Python v2.3 veya daha yenisi
* CMake v2.6.4 veya daha yenisi
  1. Kurulum Yöntemleri

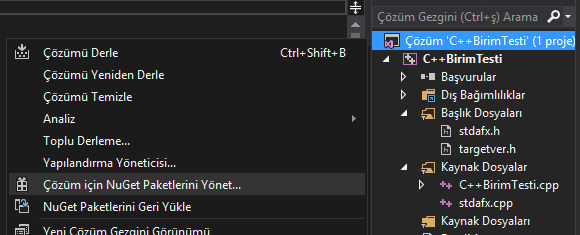
Visual Studio’da unit test yazmak için bir unit test projesi oluşturmamız  gerekiyor. Bu işlemin bir ön şartı var, Visual Studio sadece Statik Kütüphane (.lib)  projeleri için test yazmaya izin veriyor. Eğer projemizi çalıştırabilir yapmak istiyorsak Solution’ımıza bir çalıştırılabilir proje ekleyerek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

* + 1. Otomatik Kurulum

Google Test ile Visual Studio’da çalışmak mümkün fakat bu gtest’i projemize nasıl dahil etmeliyiz ? Bunun için iki yöntem mevcut fakat en stabil olan manuel kurulum.

Otomatik Kurulum maddelerinde bahsi geçen NuGet paketlerinin kurulumunu pekiştirmek için eklenmiş olan örnek görsel Şekil 5.1 deki gibidir.

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



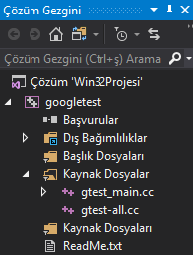
Şekil 5.1. NuGet Paketlerini Yönetimi

* + 1. Manuel Kurulum

GitHub aracılığıyla [Google Test Framework](https://github.com/google/googletest)’ü bilgisayara indirip aşağıdaki adımları takip ederek kurulumu tamamlayalım.

* 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 işaretlini kaldırıp 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 işaretlini kaldırıp 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\VisualStudio2017\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.

GTest Statik Kütüphane’nin manuel kurulumunu pekiştirmek için eklenmiş olan kurulumun son halinin görseli Şekil 5.2 deki gibidir.



Şekil 5.2. Gtest Statik Kütüphanesi

* 1. Test Yöntemleri

ASSERT\_XXX(): Hata oluşan yerde testi durdurur, geri kalan testleri çalıştırmaz

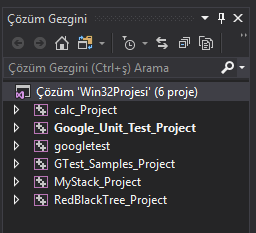
EXPECT\_XXX(): Ölümcül olmayan hatalardır, hatayı gösterir ve teste devam eder

| **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++ BİRİM TESTİ GERÇEKLEMESİ

Ekte kaynak kodları yer alan “Win32Projesi” adında yeni bir proje oluşuturuldu ve içine Manuel Kurulum Yöntemi bölümünde bahsedildiği gibi Gtest SK’si yüklendi.

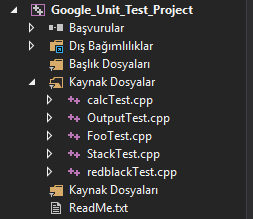
Gtest SK’side dahil olmak üzere 6 proje oluşturuldu [1]. Bunlardan bir tanesi de Gtest’in örnekler klasöründen alınan 3 test senaryosunu içermekte.



Şekil 5.3. Win32Projesi Çözüm Gezgini Gösterimi

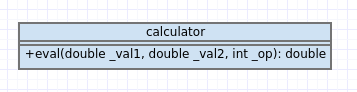
* 1. Google Birim Testi VS Projesi

“googletest-master/samples” dizininde 11 test senaryosu mevcut ve bu senaryolardan da yola çıkarak Şekil 5.4. de gösterildiği gibi 5 ayrı test senaryosu oluşturduk.

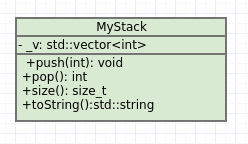


Şekil 5.4. Oluşturulan Test Senaryoları

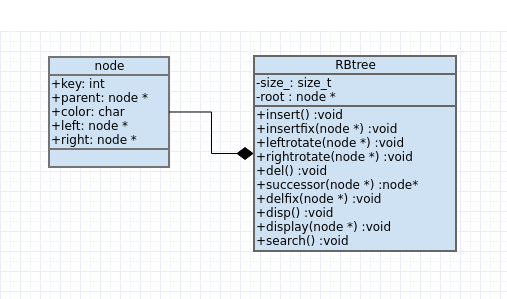
Oluşturulan senaryolara ait gösterimler aşağıdaki gibidir.



Şekil 5.4.1. Dört İşlem Hesaplama Senaryosu



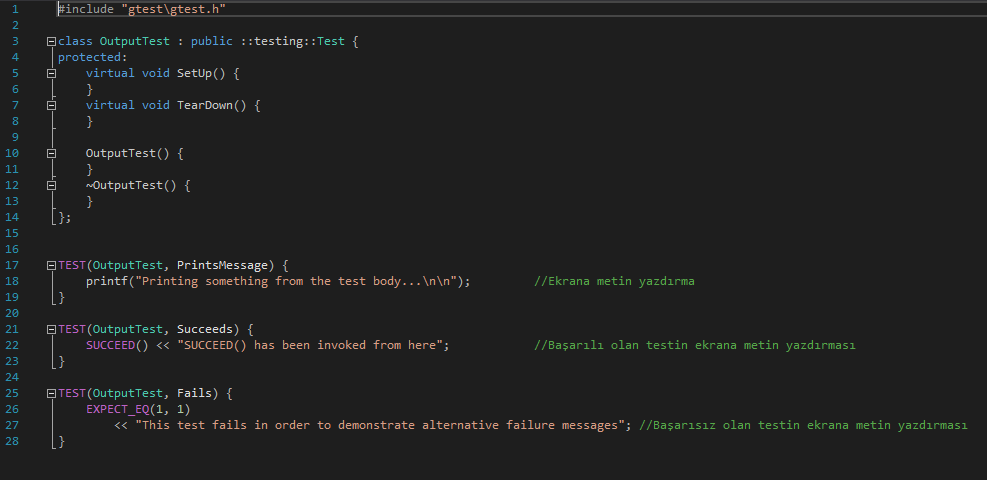
Şekil 5.4.2. Stack Test Senaryoları



Şekil 5.4.3. Kırmızı Siyah Arama Ağacı Test Senaryoları

* + 1. Ekrana Yazdırma Testi

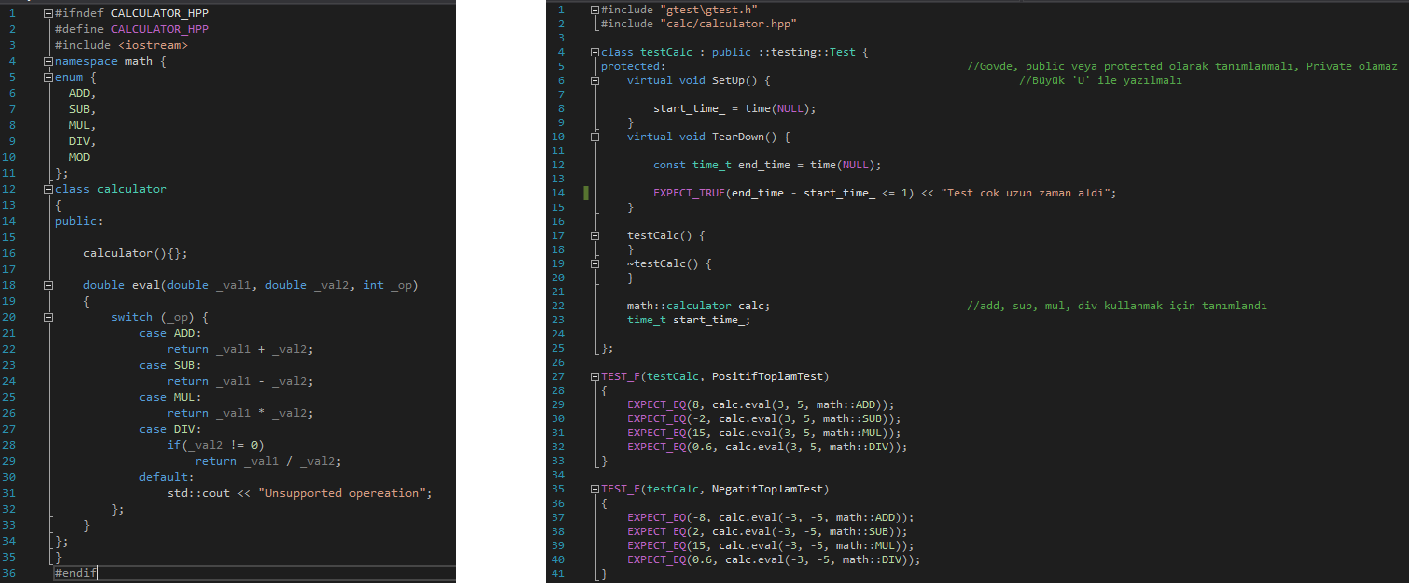
Herhangibir başlık dosyası veya kaynak dosya kullanmadan sadece Gtest’in ekrana yazdırma işlemleri için kullanılan parametreleri ile oluşturulan basit bir testtir [1].



Şekil 5.4.4. OutputTest.cpp dosyası

* + 1. Dört İşlem Hesaplama Testi

İşlemleri yapan fonksiyonların tanımı calc\_Project altında calculator.hpp başlık dosyasına yapılmıştır. Daha sonra bu fonksiyonun testi Google\_Unit\_Test\_Project altında calcTest.cpp dosyasına Şekil 5.5. deki gibi kodlandı.

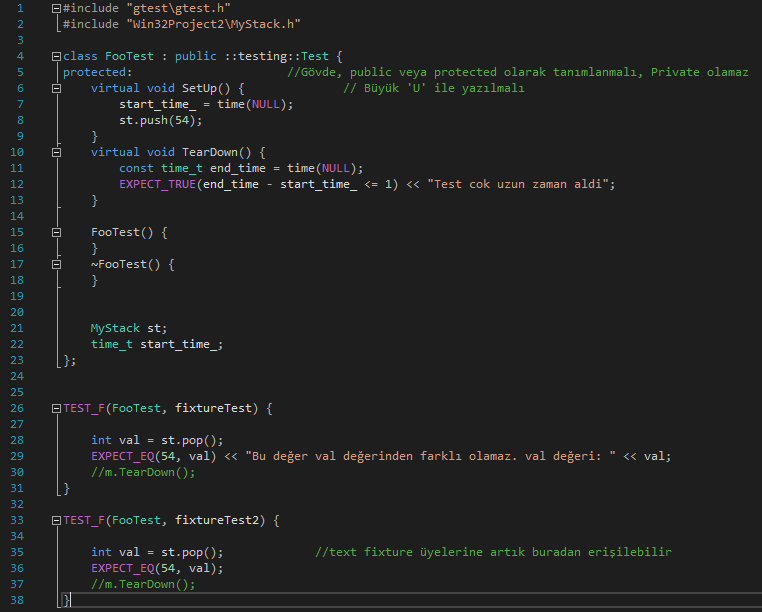


Şekil 5.4.5. calculator.hpp başlık dosyası ve birim testini içeren calcTest.cpp dosyası

* + 1. Text Fixture Testi

Test case’leri tek bir senaryodan birden fazla olan büyük projelerde, her case için aynı tanımlamaları tekrar tekrar yapmak durumunda kalmamak için global değişken gibi kullanımını Şekil 5.6 daki gibi bir sınıf oluşturup, içerisinde tanımlamamızı yapabiliriz. Tek dikkat etmemiz gereken nokta caseleri oluşturuken TEST yerinde TEST\_F biçiminde isimlendirmek olacaktır.

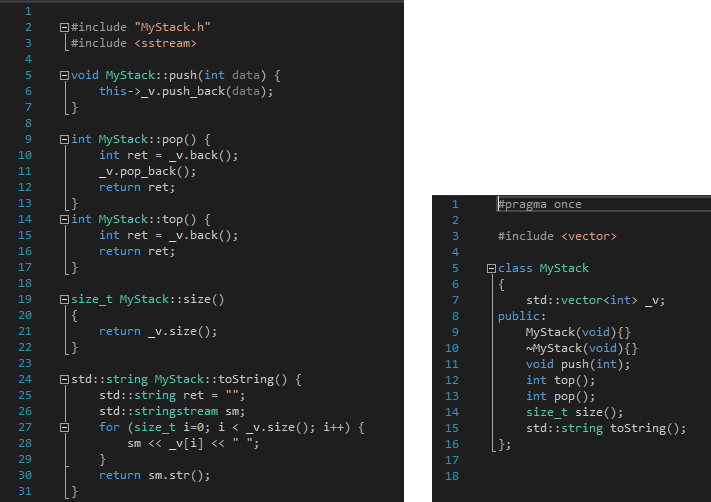
Bunun gibi test kodlarını barındıran diğer görsellere ait test çıktısı   
Sonuç Bölümü’nde gösterilmektedir.



Şekil 5.4.6. Text Fixture Testi’nin gösterimi FooTest.cpp dosyası

* + 1. Stack Testi

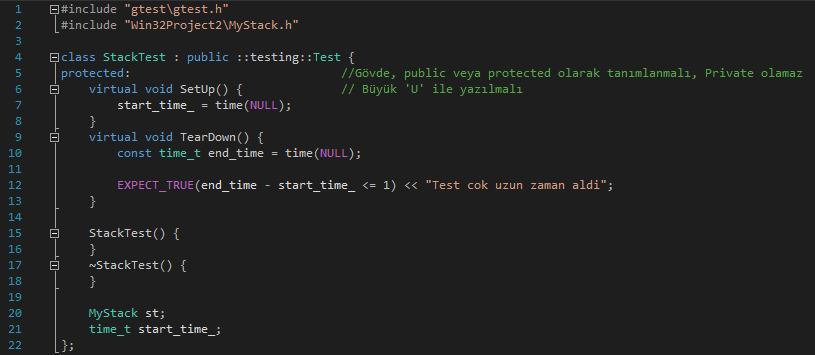
Bu testte MyStack\_Project aldınta MyStack.cpp ve MyStack.h adında kaynak ve başlık dosyaları kullanıldı. Kullanılan dosya içerisinde stack’e veri gönderme alma işlerinde kullanacağımız pop push gibi tanımlamalar Şekil 5.7. deki gibi kodlandı.



Şekil 5.5. MyStack.cpp ve MyStack.h kaynak ve başlık dosyaları

StackTest adında kaynak dosyası oluşturuldu ve içine MyStack.h başlık dosyası include edildi. Text Fixture testinde anlatıldığı gibi MyStack st; tanımı ile tüm test case’lerimizden stack’e erişebilmek için sınıf oluşturuldu.

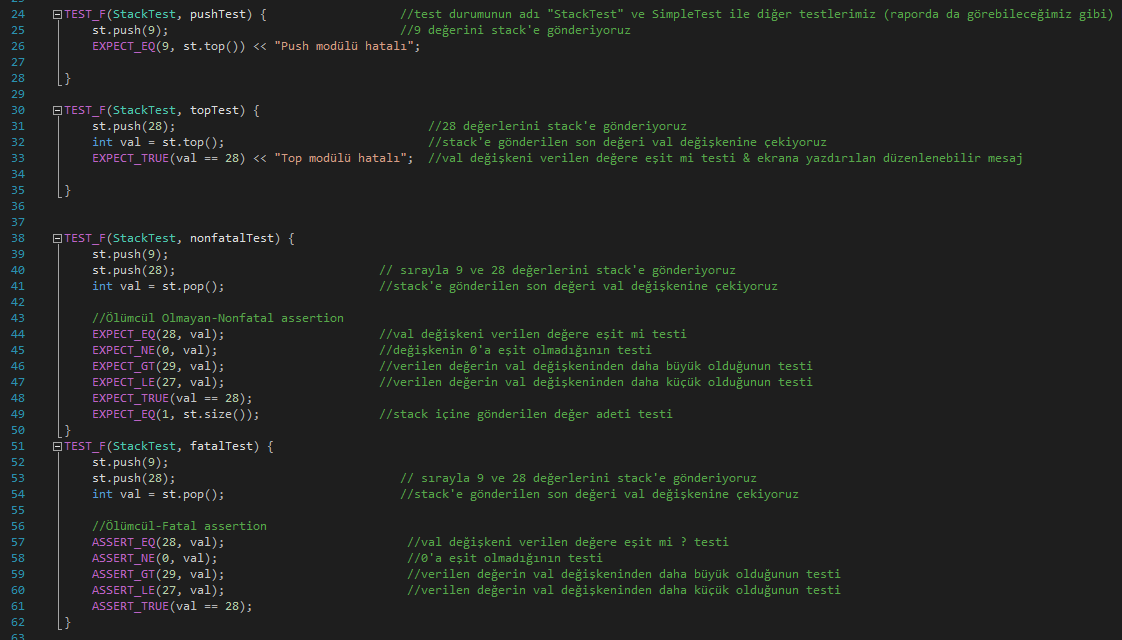
Bu sınıfta Şekil 5.8. de görüldüğü üzere fazladan time\_t bulunmakta. Bu tanımlamalar test bittikden sonra çalışmak üzere teardown altında testin uzun sürdüğü vakit ekrana çıktı vermesi için yapılmıştır.



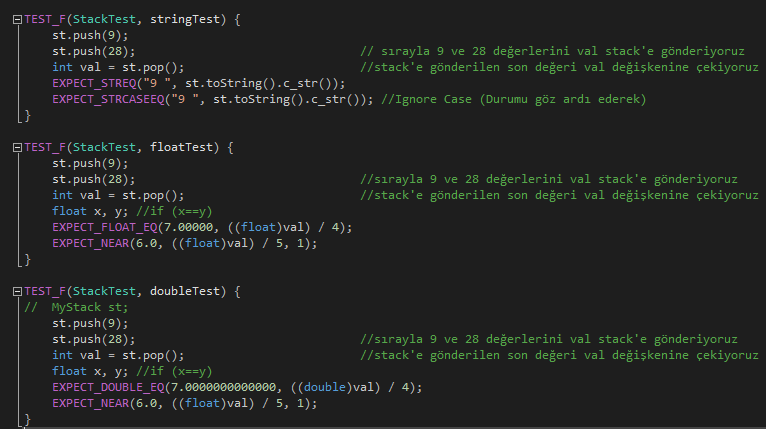
Şekil 5.6. StackTest.cpp üzerinde sınıf gösterimi

* + - 1. Stack Test Senaryoları

Senaryolar arasında syf.14’te gösterilen ölümcül ve ölümcül olmayan test yöntemleri ve stack kullanılarak pop, push, top, string, float, double gibi testler yer almaktadır.



Şekil 5.7. StackTest.cpp üzerinde push, top, pop, fatal ve nonFatal testlerinin gösterimi



Şekil 5.8. StackTest.cpp üzerinde string, float ve double testlerinin gösterimi

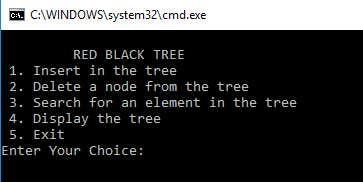
* + 1. Kırmızı Siyah Arama Ağacı Testi (RedBlack Tree)

Projede başlık dosyası olarak redblacktree.h, kırmızı siyah arama ağacı açık kaynak kodları kullanılarak kodlanmış ve test dosyasına include edilmiştir.

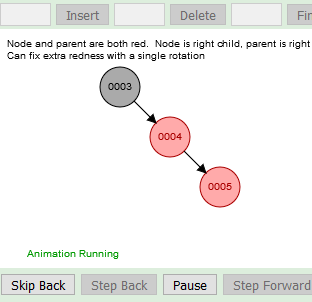
Red Black Tree; veriyi ağaçta (tree) tutarken, ağacın dengeli (balanced) olmasını sağlayan bir algoritmadır. Algoritma, veriyi tutuş şekli sayesinde, arama, ekleme veya silme gibi temel işlemlerin en kötü durum analizi (worst case analysis) O(logn)’dir, yani algoritma n elman için bu işlemleri en kötü O(logn) zamanda yapmaktadır.

Kırmızı-siyah ağaçlar (red-black trees) tanım itibariyle ikili arama ağaçlarıdır (binary search tree) ve bu anlamda, herhangi bir düğümün solunda kendisinden küçük ve sağında ise büyük verilerin durması beklenir. Ağaçta ayrıca her düğüm için bir renk özelliği tutulur. Yani bir düğüm kırmızı veya siyah renk özelliği taşıyabilir.

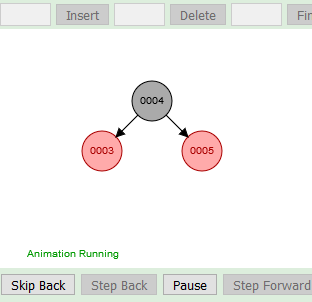
Test Gerçekleme projemize eklemiş olduğumuz kırmızı siyah ağaç c++ programının ekran çıktısı Şekil … deki gibidir. Kırmızı siyah ağaç mantığının daha iyi anlaşılması adına dengeleme görselleri ise Şekil … deki gibidir.



Şekil 6.1. Red Black Tree kodlarının ekran çıktısı

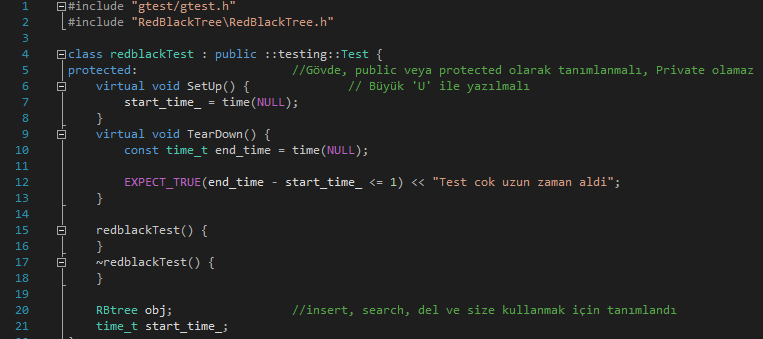


Şekil 6.2. Red Black Tree dengelenmeden önceki gösterim

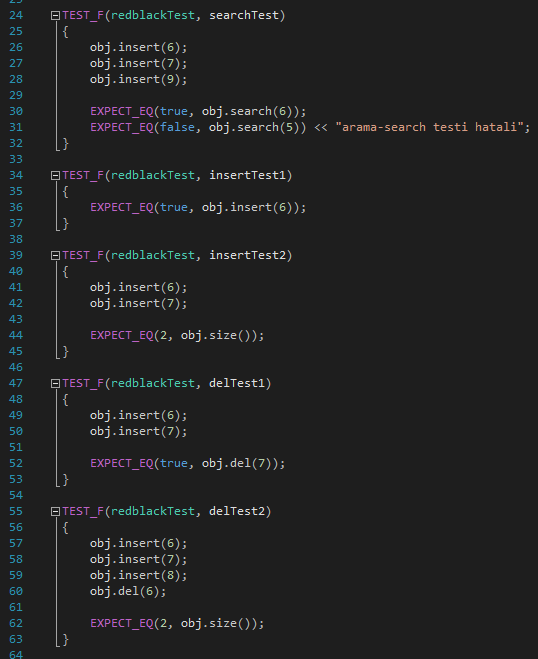


Şekil 6.3. Red Black Tree dengelenmeden sonraki gösterim

* + - 1. Kırmızı Siyah Ağaç Test Senaryoları



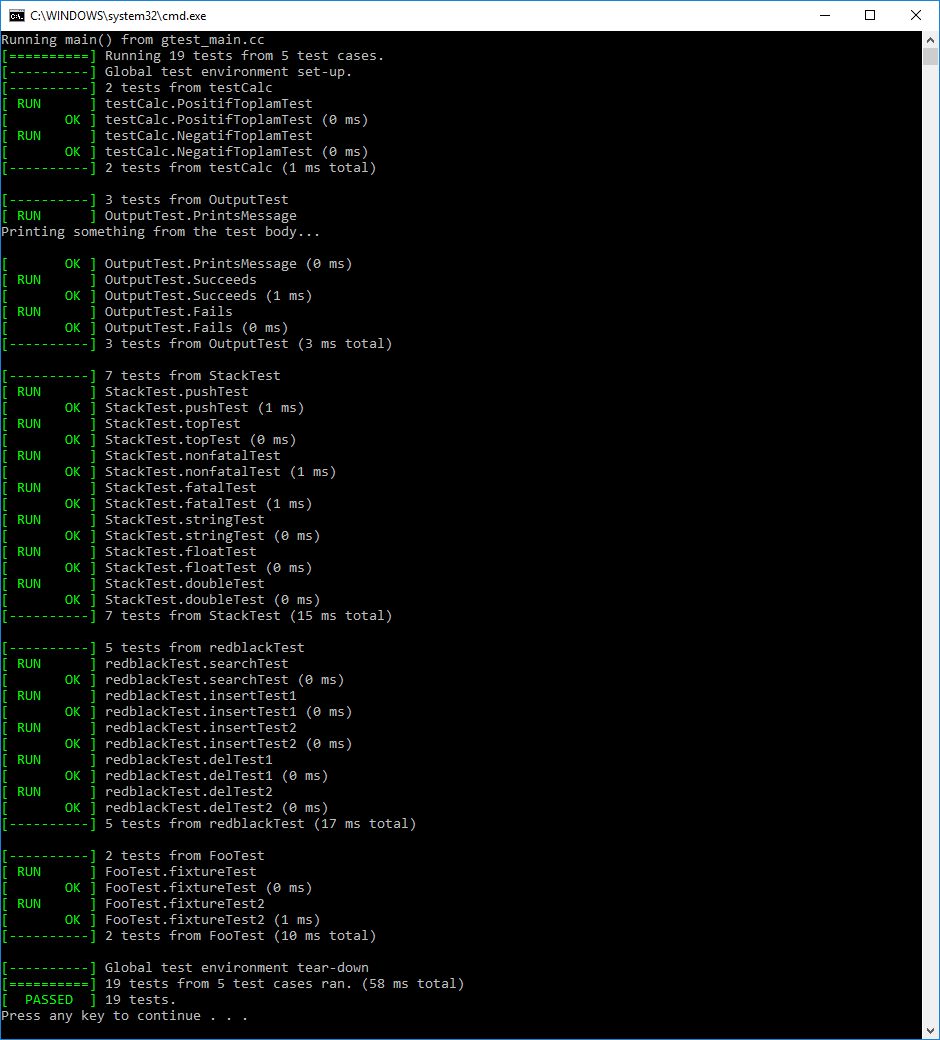
Şekil 6.4. redblackTest.cpp dosyası sınıf gösterimi



Şekil 6.5. redblackTest.cpp dosyası test case’leri gösterimi

# SONUÇLAR

Yapılandırma çıktısında hata ve uyarılar giderildi ve tüm case’leri başarılı bir şekilde çalıştırıldı. Programın son hali Şekil 6.1. deki gibidir.



Şekil 7.1 Tüm test senaryolarının ekran çıktısı gösterimi

**KAYNAKLAR**

|  |  |  |
| --- | --- | --- |
| [1] |  | https://testcollab.com/blog/repurposing-test-cases-with-tags/,  Erişim Tarihi: 05.05.2017. |
| [2] |  | https://en.wikipedia.org/wiki/Google\_Test, Erişim Tarihi: 05.05.2017. |
| [3] |  | https://github.com/google/googletest, Erişim Tarihi: 01.04.2017. |
| [4] |  | https://www.slideshare.net/hmarchezi/c-unit-test-with-google-testing-framework, Erişim Tarihi: 01.04.2017. |
| [5] |  | https://en.wikipedia.org/wiki/List\_of\_unit\_testing\_frameworks#C.2B.2B,  Erişim Tarihi: 01.04.2017. |
| [6] |  | https://www.ibm.com/developerworks/aix/library/au-googletestingframework.html, Erişim Tarihi: 01.04.2017. |
| [7] |  | http://stackoverflow.com/questions/2565299/using-assert-and-expect-in-googletest, Erişim Tarihi: 01.04.2017. |
| [8] |  | http://www.coders-hub.com/2015/07/red-black-tree-rb-tree-using-c.html, Erişim Tarihi: 05.01.2013. |
| [9] |  | https://www.slideshare.net/andreafrancia/google-c-testing-framework-in-visual-studio-2008, Erişim Tarihi: 05.01.2013. |
| [10] |  | https://www.cs.usfca.edu/~galles/visualization/RedBlack.html,  Erişim Tarihi: 10.05.2017. |

**ÖZGEÇMİŞ**

Oğuzhan İnce, 26.11.1993 de Diyarbakır’da doğdu. İlk, orta ve lise eğitimini Eskişehir’de tamamladı. 2009 yılında Eskişehir T. Anadolu Teknik Lisesi, Bilgisayar Bölümü’nden, stajını Gelişim Okulları Bilgi-İşlem Departmanı’nda tamamladıktan sonra Eskişehir Cumhuriyet Anadolu Lisesi’ne geçiş yaparak 2012’de mezun oldu. 2013 yılında Gaziantep Üniversitesi Bilgisayar Mühendisliği Bölümü’nü kazandı. 2016 yılında Sakarya Üniversitesi Bilgisayar Mühendisliği Bölümü’ne yatay geçiş yaptı ve halen Sakarya Üniversitesi’nde öğrenciliğine devam etmektedir.

Savaş Kaplan, 27.08.1990 da Diyarbakir'da doğdu. İlk ve orta eğitimini Diyarbakır'da, lise eğitimini Bursa'da tamamladı. 2009-2013 yılları arasında Uludağ Üniversitesi Fizik Bölümü eğitimini tamamladiktan sonra 2013 yılında Gaziantep Üniversitesi Bilgisayar Mühendisliği Bölümü’ne başladı. 2016 yilinda Sakarya Üniversitesi Bilgisayar Muhendisliği’ne yatay geçis yapti ve halen Sakarya Üniversitesi Bilgisayar Mühendisliği Bölümü’nde öğrenciliğine devam etmektedir.

**BSM 498 BİTİRME ÇALIŞMASI**

**Değerlendİrme ve Sözlü Sınav Tutanağı**

**KONU** : GOOGLE TEST ÇATISI İLE C++ BİRİM TESTİ GERÇEKLEMESİ

**ÖĞRENCİLER:** U161210071 OĞUZHAN İNCE, U161210070 SAVAŞ KAPLAN

|  |  |  |  |
| --- | --- | --- | --- |
| Değerlendirme Konusu | İstenenler | Not Aralığı | Not |
| **Yazılı Çalışma** |  |  |  |
| **Çalışma klavuza uygun olarak hazırlanmış mı?** | x | 0-5 |  |
| **Teknik Yönden** |  |  |  |
| **Problemin tanımı yapılmış mı?** | x | 0-5 |  |
| Geliştirilecek yazılımın/donanımın mimarisini içeren blok şeması (yazılımlar için veri akış şeması (dfd) da olabilir) çizilerek açıklanmış mı? |  |  |  |
| Blok şemadaki birimler arasındaki bilgi akışına ait model/gösterim var mı? |  |  |  |
| Yazılımın gereksinim listesi oluşturulmuş mu? |  |  |  |
| Kullanılan/kullanılması düşünülen araçlar/teknolojiler anlatılmış mı? |  |  |  |
| Donanımların programlanması/konfigürasyonu için yazılım gereksinimleri belirtilmiş mi? |  |  |  |
| UML ile modelleme yapılmış mı? |  |  |  |
| Veritabanları kullanılmış ise kavramsal model çıkarılmış mı? (Varlık ilişki modeli, noSQL kavramsal modelleri v.b.) |  |  |  |
| Projeye yönelik iş-zaman çizelgesi çıkarılarak maliyet analizi yapılmış mı? |  |  |  |
| Donanım bileşenlerinin maliyet analizi (prototip-adetli seri üretim vb.) çıkarılmış mı? |  |  |  |
| Donanım için gerekli enerji analizi (minimum-uyku-aktif-maksimum) yapılmış mı? |  |  |  |
| Grup çalışmalarında grup üyelerinin görev tanımları verilmiş mi (iş-zaman çizelgesinde belirtilebilir)? |  |  |  |
| Sürüm denetim sistemi (Version Control System; Git, Subversion v.s.) kullanılmış mı? |  |  |  |
| Sistemin genel testi için uygulanan metotlar ve iyileştirme süreçlerinin dökümü verilmiş mi? |  |  |  |
| Yazılımın sızma testi yapılmış mı? |  |  |  |
| Performans testi yapılmış mı? |  |  |  |
| Tasarımın uygulamasında ortaya çıkan uyumsuzluklar ve aksaklıklar belirtilerek çözüm yöntemleri tartışılmış mı? |  |  |  |
| **Yapılan işlerin zorluk derecesi?** | x | 0-25 |  |
| **Sözlü Sınav** |  |  |  |
| **Yapılan sunum başarılı mı?** | x | 0-5 |  |
| **Soruları yanıtlama yetkinliği?** | x | 0-20 |  |
| **Devam Durumu** |  |  |  |
| **Öğrenci dönem içerisindeki raporlarını düzenli olarak hazırladı mı?** | x | 0-5 |  |
| **Diğer Maddeler** |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| **Toplam** |  |  |  |

Danışman (Jüri adına): Ars. gor. m. fatih adak

danışman imzası: