

**FIRAT ÜNİVERSİTESİ**

TEKNOLOJİ FAKÜLTESİ

Yazılım Mühendisliği Bölümü

**YMH319 – Programlama Dilleri**

**Proje Uygulaması ve Dokümantasyonu**

**Bütünleme Sınavı**

170541614 Hibe Alabd

**Ocak – 2021**

**İÇİNDEKİLER**

**Başlık Sayfa No**

**Bölüm 1: Dil Genel Bilgiler …………………………………………….……………………………. 3**

**Bölüm 2: Sözdizimi Yapısı ………...………………………………………………………………. 4**

**Bölüm 2.1: Sözdizimi İfade Kuralları…….………………………………………………………. 4**

**Bölüm 2.2: Değişken Tanımlama Sözdizimi 1.………………………………………………. 4**

**Bölüm 2.3: Değişken Tanımlama Sözdizimi 2 ………………………………………………. 4**

**Bölüm 2.4: Matematiksel İşlem Sözdizimi …..………………………………………………. 5**

**Bölüm 2.5: Yazdırma İşlemi Sözdizimi …..………….…………………………………………. 5**

**Bölüm 3: Tasarım ….………………….…………………………………………………………………. 5**

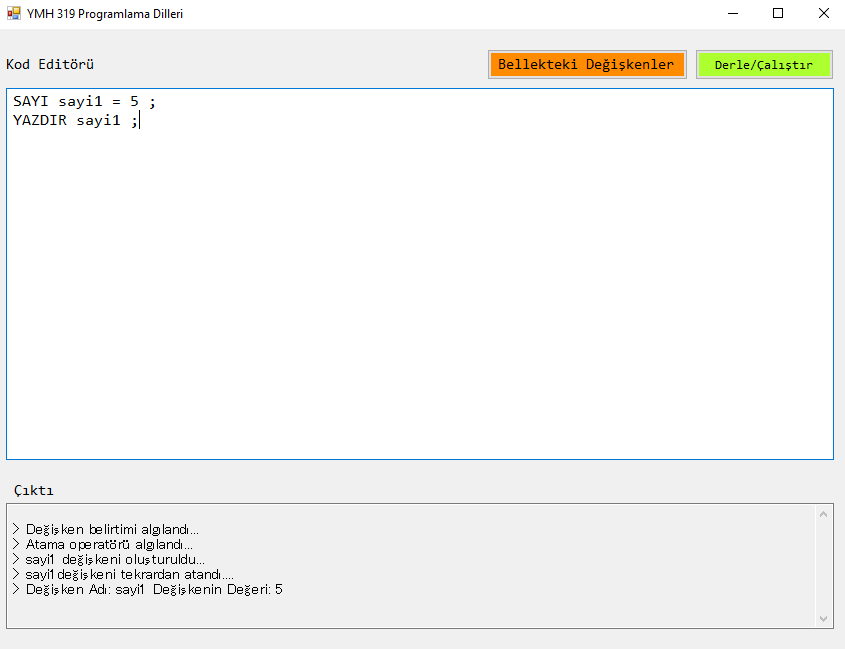
**Bölüm 3.1: Hata Ayıklama ………..….………………….....………..……………………………. 6**

**Bölüm 1: Dil Genel Bilgiler**

İlk olarak bahsedilmesi gereken konu bu dilin hangi amaca yönelik olduğudur. Çocuklara özel olarak geliştirilmiş olan bu dilin sert ve bir o kadar da esnek olmayan bir yapısı bulunmaktadır. Bunun asıl amacı çocukların katı kurallara sahip olan bu dille zorluklara alışıp daha sonra öğrenecekleri programlama dillerinde bir o kadar rahatlık sağlamaları amaçlanmaktadır.

Dil içerisinde temel seviyede, değişken tipleri, değişken atama ve işlemleri ayrıca bir o kadar da gelişmiş olan matematiksel işlem yorumlayıcısı bulunmaktadır. Dilin asıl odaklandığı nokta ise bu matematiksel işlemlerdir. Şu anlık sadece dört işlemi desteklemektedir.

Dil, C# dili temel alınarak gerçekleştirilmiştir. Yani arkasında C# tarafında çalışan komutlar ile dilin derlenmesi ve işlenmesi sağlanmaktadır. Ayrıca editör penceresi ve işlevleri de yine C# geliştirme ortamı ile gerçekleştirilmiştir.



*Şekil 1.1 Kod Editörü Penceresi*

Yukarıdaki pencereyi(Bkz: Şekil 1.1) açıklamak gerekirse; ilk olarak göze çarpan iki adet buton bulunmakta. İlk olarak yeşil renkte olan “Derle/Çalıştır” butonu, aşağısında yer alan komut penceresinin içerisinde yer alan komutları derler ve çalıştırır. Ardından çalışan bu komutlar en aşağıda yer alan “Çıktı” bölümünde sonuçlarını gösterirler. Turuncu renkte olan olan buton ise programlama dilinin çalışma zamanındaki mevcut olan değişkenleri çıktı ekranına listelemeye yarar.

**Bölüm 2: Sözdizimi Yapısı**

Dilin sözdizimi yapısı Bölüm 1’de bahsedildiği üzere oldukça katı kurallar üzerine kurulmuştur. Bu kurallar aşağıda yer alan yapıya göre inşa edilmiştir.

**Bölüm 2.1: Sözdizimi İfade Kuralları:**

Syntax kuralları ile ilgili açıklamalar yapılırken bazı semboller ve ifadeler kullanılmıştır. Bu ifade ve semboller aşağıda yer almaktadır. Kuralları okumadan önce göz atmanız tavsiye edilir.

1. () arasına yazılan ifadelerden ***sadece* *biri*** olmak zorundadır.
2. [] arasına yazılan ifade ***sıralı(ardışıl) kural*** anlamına gelmektedir.
3. {} arasına ise ardışıl kuralların ***birleştirilmesi*** ifade edilir. Ve “,” işareti ile ayrılırlar.
4. | ifadesi ***veya*** anlamına gelmektedir.
5. <satır sonu>: Satırın ***sonunu*** işaret etmektedir. Syntax: “;”
6. <işlemler>: Söz konusu programlama dili kullanılarak yapılan işlemler için örnek bir ifadedir.
7. <boşluk>: Klavyeden “space” tuşuna basılarak girilen ifadeyi temsil etmektedir.
8. <atama operatörü>: Bir değişkene değer ataması yapan operatörü temsil etmektedir. Syntax: “=“

**Değişken Tanımlama Sözdizimi 1:**

Değişkenin tanımlanması, sayısal bir değeri alarak ya da başka bir değişkenden o değişkenin değerini alarak atanması işlemi olmak üzere iki türlüdür. Bir değişkenin başlatılması için ilk olarak bir sayısal değer atanmak zorundadır. Şu anki versiyonda bellekte referans gösterme durumu söz konusu değildir.

**Syntax Kuralı:**

<değişken tipi><boşluk><değişken adı><boşluk><atama operatörü><boşluk>(<değer>|<değişken>)<boşluk><satır sonu>

**Değişken Tanımlama Sözdizimi 2:**

İkinci değişken başlatma yöntemi ise, atama operatörünün sağ tarafında bir matematiksel bir işlem gerçekleştirip bunun sonucunu değişkene vermektir.

**Syntax Kuralı:**

<değişken tipi><boşluk><değişken adı><boşluk><atama operatörü><boşluk>[(<değer>|<değişken>)<matematiksel operatör> (<değer>|<değişken>)]<satır sonu>

**Bölüm 2.4: Matematiksel İşlem Sözdizimi:**

Yukarıda bulunan [(<değer>|< değişken >)<matematiksel operator>(<değer>|< değişken >)] ifadesinde yer alan ardışıl kurala göre; ilk olarak ***değişken veya değer***, sonra ***operatör*** ve daha sonra tekrardan ***değer veya değişken*** ifade gelmek zorundadır. Ardışıl kurallar yan yana birleştirilebilir. Bunlar ise {} ifadesi ile gösterilir. Örneğin:

{[(<değer>|<değişken>)<matematiksel operator>(<değer>|<değişken >)],[ <matematiksel operator>(<değer>|< değişken >) ] }

Dikkat edileceği üzere matematiksel işlemlerde yani matematiksel operatörün kullanıldığı yerlerde <boşluk> ifadesi kullanılmamaktadır. Bu ifadenin durumlara göre kullanımı dili katı bir hale getirmektedir.

**Bölüm 2.5: Yazdırma İşlemi Sözdizimi:**

Çıktı ekranına bir değişkeni, değeri ya da matematiksel bir ifadenin sonucunu yazdırabiliriz. Bunu aşağıdaki kural yapısına göre yaparız.

**Syntax Kuralı:**

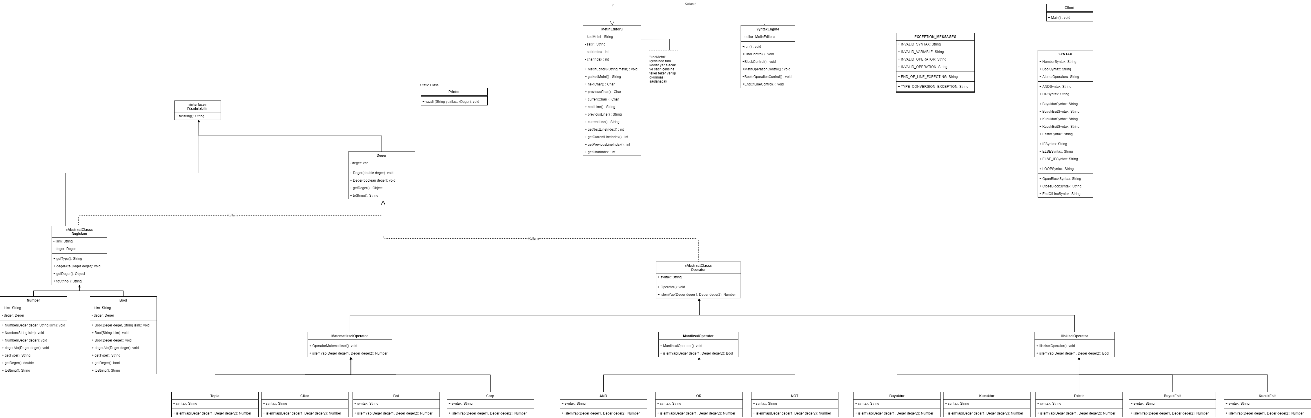
**YAZDIR**<boşluk>(<değer>|<değişken>)<boşluk><satır sonu>

**YAZDIR**<boşluk>{[(<değer>|<değişken>)<matematiksel operator>(<değer>|<değişken >)],[ <matematiksel operator>(<değer>|< değişken>) ] } <boşluk><satır sonu>

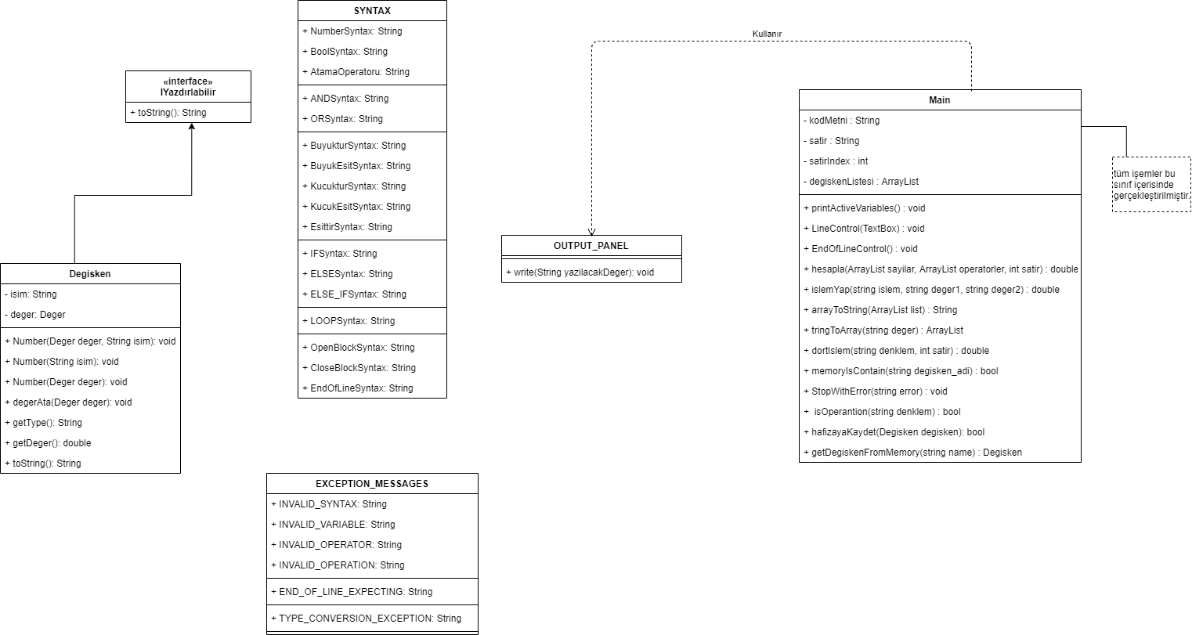
**Bölüm 3: Tasarım**

İlk bölümde de bahsedildiği gibi dilin gerçekleştirilmesi C# ortamında yapılmıştır. C# dilinin sunmuş olduğu Nesne Tabanlı Programlama yapısı sayesinde söz konusu dilin tasarımı, bu programlama paradigması prensipleri doğrultusunda gerçekleştirildi.

İlk olarak çok büyük bir tasarım benimsendikten sonra(Bkz: Şekil 3.1) bunun bir hata olduğunun farkına vardıktan sonra proje, şekil 3.2 de yer alan UML Class diyagramına indirgenmiştir.



*Şekil 3.1 Projenin ilk UML Class tasarımı*



*Şekil 3.2 Projenin indirgenmiş UML Class tasarımı*

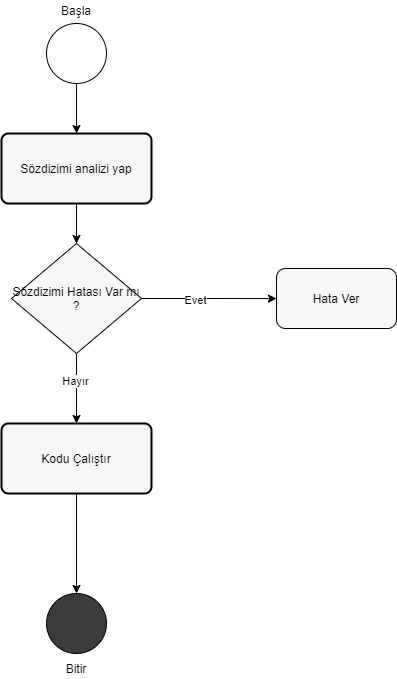
Projenin şu an hazır olan tasarımı şekil 3.2 de yer almaktadır. Bu tasarım benimsenerek kodlama gerçekleştirilmiştir.

**Bölüm 3.1: Hata Ayıklama:**

Projenin en zor olan kısmı ise ilk olarak kodun sözdizimi olarak doğru olup olmadığını kontrol etmekti. Ardından ise sözdizimi analizinden geçmiş kod çalıştırılacak. Hata ayıklama işlemini, programlama dilinin katı bir yapıya sahip olmasından dolayı bunu yapmak için aşağıdaki gibi bir yapı tercih edildi.

|  |
| --- |
| 1. TİP tanımı gelmişse    1. <Boşluk> karakteri varmı?       1. Varsa: atama operatörüne kadar ya da <endofline> karakterine kadar oku. Bu yere kadar olan kısım değişkenin adıdır.(ADIM 1B’YE DEVAM ET)       2. Yoksa INVALID SYNTAX hatası ver    2. <Atama operatörü> varmı?       1. Varsa: Sağında sayı mı var?          1. Varsa: Ve <endofline> a kadar başka bir değer ve operatör yoksa bunu değişkenin değeri yap.          2. Varsa: Ve yanında operatör var ise 4.adıma(işlem kontrolleri) atla Eğer yanında .          3. Yoksa: Ve <endofline> a kadar başka bir değer ve operatör yoksa değişkeni oluştur ancak değerini atama.       2. Varsa: Sağında değişken mi var?          1. Varsa: Ve <endofline> a kadar başka bir değer ve operatör yoksa bu değişkenin değerini söz konusu değişkenin değeri yap.          2. Varsa: Ve <endofline> a kadar başka bir değer ve operatör var ise adım 4(işlem kontrolleri)’e atla. 2. Satır başında <endofline> varsa INVALID SYNTAX hatası ver 3. Satır sonunda <endofline> yoksa END\_OF\_LINE\_EXPECTING hatası ver 4. İŞLEM KONTROLLERİ |

Aşağıda yer alan akış diyagramında ise genel bir işleyiş yer almaktadır. Dikkat edildiği üzere Program hata verdikten sonra son bulmamakta. Değişkenlerin devamlılığını görmek için bu şekilde bir yapı oluşturuldu.



*Şekil 3.1.1 Genel Akış Şeması*

Programlama dilinin tasarımı genel olarak şekillerden ziyade sözdizimi bildirimi ve genel işleyiş hakkındadır. Zaten programın kodlarında açıklanan yapılar açıkça görülmektedir.