T.C.
İSTANBUL SABAHATTİN ZAİM ÜNİVERSİTESİ
MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

2019-2020 EĞİTİM ÖĞRETİM YILI BAHAR YARIYILI

BIM406 DERLEYİCİ TASARIMI DERSİ FİNAL PROJESİ

Hoca: Dr. Öğr. Üyesi ALİ HAMİTOĞLU

AHMET BEDİRHAN SAĞIR 030414026

SILAY POYRAZ 030414020

İSTANBUL, 2020

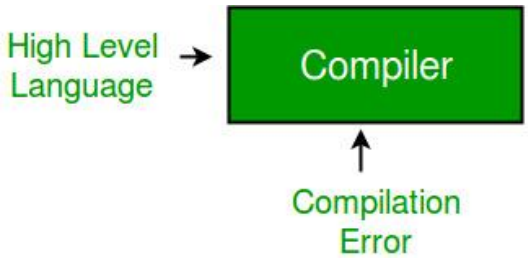
AHMET BEDİRHAN SAĞIR 030414026-SILAY POYRAZ 030414020

KONU BAŞLIKLARI

- Derleyici Tasarımı nedir?
- Tanımlar-Açıklamalar
- Parsing Aşamaları
- Kullanılan Programlama Dili ve IDE
- Proje Kodu ve Açıklamaları

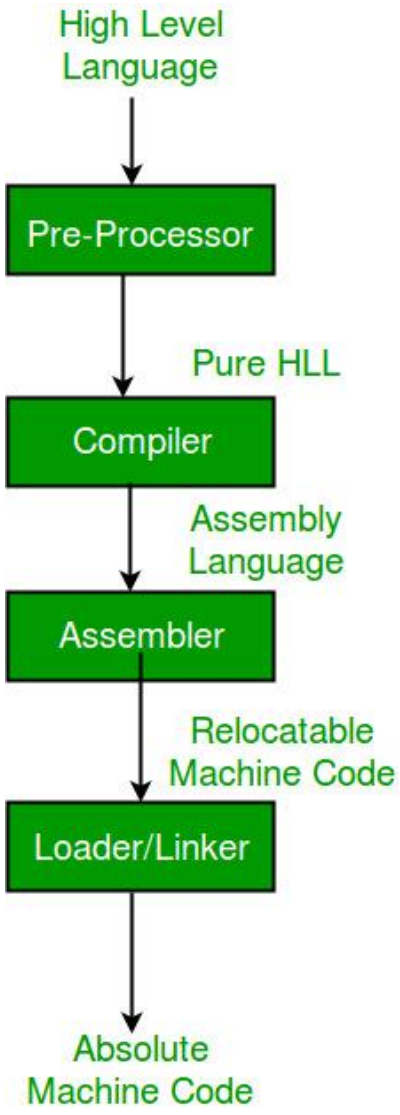
1. Derleyici Tasarımı Nedir?

Derleyici , yüksek seviyeli dilde (Kaynak Dil) yazılmış bir programı düşük seviyeli dile (Nesne / Hedef / Makine Dili) dönüştüren bir yazılımdır.



- 'A' makinesinde çalışan ve başka bir 'B' makinesi için kod üreten **Çapraz Derleyici** . Derleyicinin üzerinde çalıştığı platformdan başka bir platform için kod oluşturabilir.
- **Kaynaktan Kaynağa**

Derleyici veya transcompiler veya transpiler, bir programlama dilinde yazılan kaynak kodunu başka bir programlama dilinin kaynak koduna çeviren bir derleyicidir.



- **Dil işleme sistemleri (Derleyici kullanarak) -**

Bir bilgisayarın Yazılım ve Donanımın mantıklı bir montajı olduğunu biliyoruz. Donanım bir dili biliyor, bizim için kavraması zor, sonuç olarak yüksek seviyeli dilde programlar yazma eğilimindeyiz, bu da düşünceleri kavramamız ve sürdürmemiz için çok daha az karmaşık. Şimdi bu programlar bir dizi dönüşümden geçiyor, böylece kolayca makine kullanılabiliyorlar. Burada dil işlem sistemleri devreye girer.

- **Üst Seviye Dil -** Bir program #define veya #include veya #define gibi #include yönergeleri içeriyorsa buna HLL adı verilir. İnsanlara daha yakın, ancak makinelerden uzak. Bu (#) etiketlere işlemci öncesi yönergeler denir. Ön işlemciyi ne yapılacağı konusunda yönlendiriyorlar.
- **Ön İşlemci - Ön** işlemci, dosya genişletme adı verilen dosyaları ve tüm #define yönergelerini makro genişletme kullanarak dahil ederek tüm #include yönergelerini kaldırır. Dosya dahil etme, büyütme, makro işleme vb.
- **Makina Dili -** Ne ikili biçimde ne de yüksek düzeyde. Makine talimatlarının ve yürütme için gerekli diğer bazı yararlı verilerin bir kombinasyonu olan bir ara durumdur.
- **Assembler -** Her platform için (Hardware + OS) bir montajcımız olacak. Evrensel değiller çünkü her platform için bir tane var. Montajcının çıktısına nesne dosyası denir. Montaj dilini makine koduna çevirir.
- **Yorumlayıcı -** Bir yorumlayıcı, tıpkı bir derleyici gibi, üst seviye dili düşük seviye makine diline dönüştürür. Ancak girdiyi okuma biçimleri farklıdır. Derleyici bir kerede girişleri okur, işlemeyi yapar ve kaynak kodunu yürütürken, yorumlayıcı aynı satır satır yapar. Derleyici tüm programı tarar ve bir bütün olarak

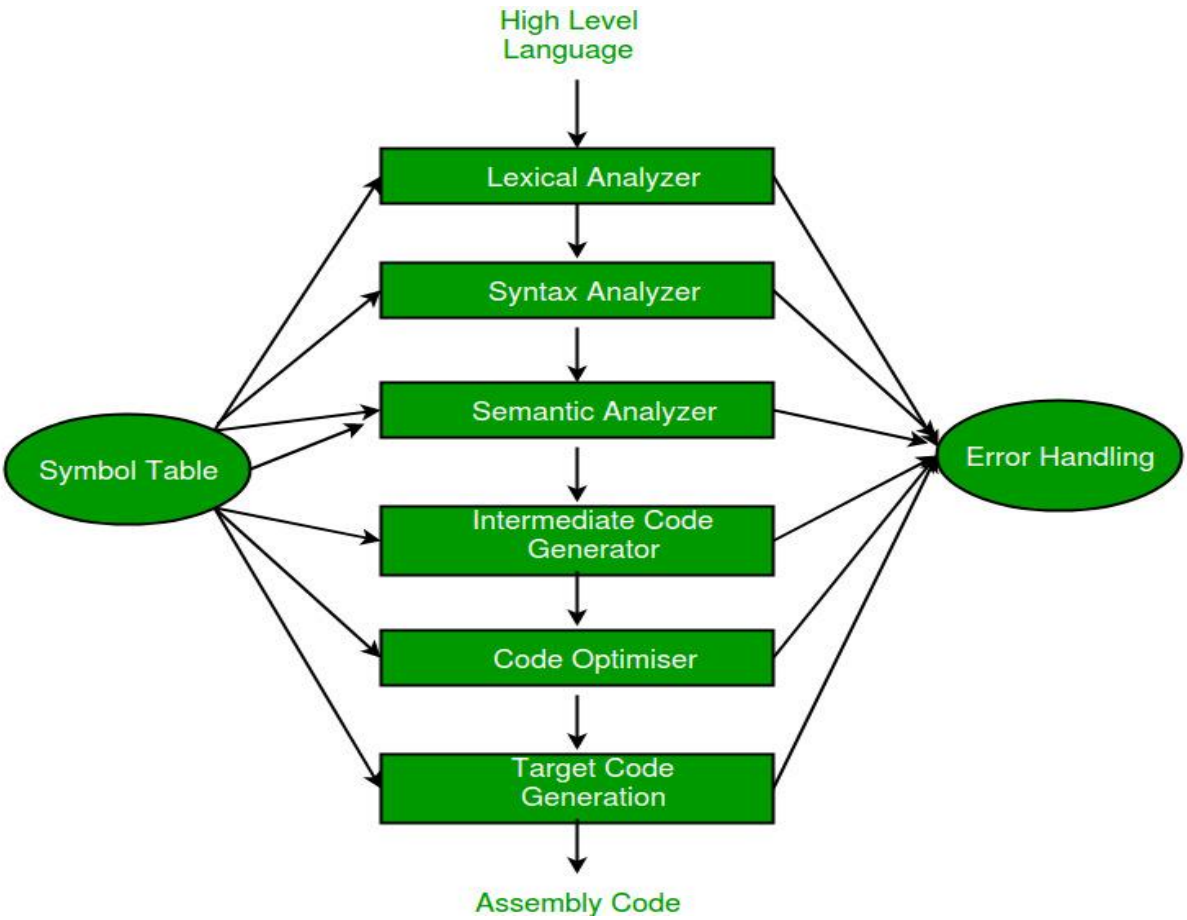
makine koduna çevirirken, bir tercüman programı her seferinde bir ifade çevirir. Yorumlanan programlar genellikle derlenen programlara göre daha yavaştır.

- **Yer Değiştirilebilir Makine Kodu -** Herhangi bir noktada yüklenebilir ve çalıştırılabilir. Program içindeki adres, program hareketi için işbirliği yapacak şekilde olacaktır.
- **Loader / Linker -** Yeniden yüklenebilir kodu mutlak koda dönüştürür ve programı çalıştırmaya çalışır. Bağlayıcı, yürütülebilir olmasını sağlamak için çeşitli nesne dosyalarını tek bir dosyaya yükler. Ardından yükleyici belleğe yükler ve yürütür.

Bir Derleyicinin Aşamaları -
Derlemenin birçok parçası olan iki ana derleme aşaması vardır. Her biri önceki seviyenin çıktısından girdi alır ve koordineli bir şekilde çalışır.

Analiz Aşaması - Verme kaynak kodundan bir ara gösterim oluşturulur:

1. Sözcüksel Analiz
2. Sözdizimi Analizörü
3. Anlamsal Çözümleyici
4. Ara Kod Üreticisi



2. Tanımlar-Açıklamalar

- Token (Belirteç): Belirteç, bilgisayar dili programının derleyici için anlamlı olan en küçük öğesidir (karakteri). Ayrıştırıcı, bunları belirteç olarak tanımalıdır: tanımlayıcılar, anahtar kelimeler, değişmez değerler, işleçler, noktalama işaretleri ve diğer ayırıcılar.

- TokenGetir: Verilen input stringindeki stringi oluşturan her bir tokeni, işlenmek üzere stringten almak demektir. GetToken() demektir.

- Gramerdeki büyük harfler terminal-olmayan sembolleri ve tek tırnak içinde verilenler terminal sembolleri gösterir. Terminal-olmayan semboller gramerin tanımında kullanılır, gerçek programda (aşağıda örneği verilimiştir) kullanılmazlar. Tek tırnak içindeki her sembol gerçek programda aynen görülebilirler/kullanılabilir.
- Gramerde kullanılan { ve } sebmolleri hem terminal sembolü olarak kullanılmış (tek tırnak içindeyse) hem de terminal olmayan sembolün sıfır veya daha fazla tekrar edeceği anlamında kullanılmıştır (tek tırnaksız). Aynı şekilde (ve) sembolleri de hem gruplama amacıyla (tek tırnaksız) hem de terminal sembolü olarak kullanılmıştır (tek tırnak içindeyse).
- Gramerin sol tarafında terminal olmayan semboller, sağ tarafında bu terminal olmayan sembollerin nelere gidebileceği verilmiştir. Sağ tarafta terminal olmayan ve terminal sembollerin karışımı görülebilir. Sol ve sağ taraf aradaki okla beraber bir gramer kuralını oluşturur.
- Her kuralın sol ve sağ tarafı birbirinden → sembolüyle ayrılmıştır. Ayrıca satır sayısını azaltmak için birden fazla kural aynı satırda verilmiştir. Örneğin C (Cümle) terminal-olmayan sembolü sağında beş farklı terminal-olmayan sembole gitmiştir. Dolayısıyla her kuralın sağındaki dikey çubuk (|) "veya" anlamı taşır. Yani bir program cümlesi aslında bir şartlı cümle (I) veya döngü (W) vs olabilir.
- P terminal-olmayan kuralının sonundaki terminal sembol (tek tırnak içindeki nokta) programın bitişini temsil ediyor.
- I (IF) terminal-olmayan kuralının içindeki ? ve : terminal sembolleri bazı programlama dillerinde kullanılan kısa şartlı cümlecik yazım yönteminden alıntıdır. Dolayısıyla şartlı cümlelerin yazımını kolaylaştırarak programı kısaltır. I terminal olmayan kuralın ilk kısmı if-then-else yapısını ikincisi ise if-then yapısını temsil eder.
- I (IF) ve W (While döngüsü) terminal-olmayan kuralının Boolean kısımları (kuraldaki E'ler) C dilinde olduğu gibidir. 0 (sıfır) değeri "yanlış"ı başka herhangi bir değer "doğru"yu gösterir.

Gramer

P → {C} '.'

C → I | W | A | Ç | G

I → '[' E '?' C{ C } ':' C{ C } ']' | '[' E '?' C{C} ']'

W → '{' E '?' C{C} '}'

A → K '=' E ';'

Ç → '<' E ';'

G → '>' K ';'

E → T {('+' | '-') T}

T → U {('*' | '/' | '%') U}

U → F '^' U | F

F → '(' E ')' | K | R

K → 'a' | 'b' | … | 'z'

R → '0' | '1' | … | '9'

Lejand

P: program

C: Cümle

I: IF cümlesi

W: While döngüsü

A: Atama cümlesi

Ç: Çıktı cümlesi

G: Girdi cümlesi

E: Aritmetik İfade

T: Çarpma-bölme-mod terimi

U: Üslü ifade

F: Gruplama ifadesi

K: Küçük harfler

R: Rakamlar

3. Parsing Aşamaları

- First() veya Leading() - Follow() veya Trailing ()
- First ve Follow fonksiyonlarını kullanarak tahmin ayrıştırma tablosu oluşturulur.(Parsing table)
- Yığın uygulaması (Stack Implementation)
- Ardından tablonun yardımıyla tablo giriş dizesi ayrıştırılır. (Parse input string)

4. Kullanılan Programlama Dili ve Geliştirme Ortamı (IDE)

Dil: **C#**
IDE: **Microsoft Visual Studio 2019**

5. Proje Kodu ve Açıklamaları

```
using System;

namespace DerleyiciProje
{

    /* Gramer:
P →  {C} '.'
C  → I | W | A | Ç | G
I   → '[' E '?'  C{ C } ':' C{ C } ']' | '[' E '?'  C{C} ']'
W → '{' E '?'  C{C} '}'
A  → K '=' E ';'
Ç  → '<' E ';'
G → '>' K ';'
E → T {('+' | '-') T}
T → U {('*' | '/' | '%') U}
U → F '^' U | F
F → '(' E ')' | K | R
K → 'a'  |  'b'  | … |  'z'
R → '0'  |  '1'  | … |  '9'

P: program
C: Cümle
I: IF cümlesi
W: While döngüsü
A: Atama cümlesi
Ç: Çıktı cümlesi
G: Girdi cümlesi
E: Aritmetik İfade
T: Çarpma-bölme-mod terimi
U: Üslü ifade
F: Gruplama ifadesi
K: Küçük harfler
R: Rakamlar


         */
    class Program
    {
         public static int sayac = 0; //sayacı tokenleri ilerletme için kullanıyoruz
         public static string token = "";//input için tokenimizi oluşturduk
         public static char[] TokenGetir; //getToken'i tanımladık. Input'u klavyeden
almayacağımız için dizi olarak tanımladık
        static void Main(string[] args)
        {
            token = "n=9;( n-2^5><n;n=n+1)a%8/4?;}."; //Deneme stringi

            //Token için gerekli input stringini klavyeden de alabiliriz.
            //Console.WriteLine("Bir Input Stringi Giriniz: \n");
            // token= Console.ReadLine();

            Console.WriteLine("Input Stringi: "+token);//ifadeyi yazdırdık
            TokenGetir = token.ToCharArray();//tokeni char dizisine çevirdik. Çünkü
tokenimizi getToken de kullanacağız
            P();//Gramer her zaman Programdan başlar
        }
        //Grameri oluşturalım
        //Gramerimiz P yani Program başlar
```

```
    //Gramer Programdan yani P() den başlar
 public static void P()
        {
            if(token.Contains(" . "))
            {
                Environment.Exit(0);//Tokende . varsa, program biter
            }
            else
            {
                //eğer . yoksa, C fonksiyonuna yani Cümleye geç. Her paragraf
cümlelerden oluşur.
                Console.WriteLine("Program: "+ TokenGetir[sayac]);
                C();
            }
        }

        //Her program Cümlelerden yani C() den oluşur
        public static void C()
        {
            if (TokenGetir[sayac] == '[')
            {
                sayac = sayac + 1;//eğer input stringte '[' varsa, sonraki ifadeyi
getirmek için sayaç değeri 1 artar
                Console.WriteLine("Cümle: ", TokenGetir[sayac]); //cümleyi yazdırdık
                I(); //if ifadesi çalışacak
            } else if (TokenGetir[sayac]=='(')
            {
               sayac = sayac + 1;;//eğer input stringte '(' varsa, sonraki ifadeyi
getirmek için sayaç değeri 1 artar
                W(); //while döngüsü çalışacak
            }
            bool dogru = false; //doğruluk kontrolü
            //isteneni bulmak için a-z arasındaki küçük harfleri getirdik. Bu harfler
bizim terminal ifadelerimizdir.
            for (char i = 'a'; i <= 'z'; i++)
            {
                if (TokenGetir[sayac] == i)
                {
                    dogru = true;
                }
                if (dogru)
                {
                    Console.WriteLine("Cümle: "+ TokenGetir[sayac]); //cümleyi
yazdırdık
                    A(); //atama cümlesi çalışacak
                }
                if (TokenGetir[sayac] =='<')
                {
                    sayac = sayac + 1; //sayac++ ifadesi yerine yazabiliriz
                    Ç(); //çıktı cümlesi çalışacak
                }
                if (TokenGetir[sayac] =='>')
                {
                    sayac = sayac + 1;
                        G(); //girdi cümlesi çalışacak
                }
            }
        }
```

```
        //IF cümlesi tanımlandı
          public static void I()
        {
            E();//önce aritmetik ifade çalışır
            bool boo = false; //kontrol ifadesi
            //ifadede ? varsa, if ifadesi vardır
            if (TokenGetir[sayac] == '?')
            {
                Console.WriteLine("IF Cümlesi: " + TokenGetir[sayac]);
                boo = true;
            }
            if (!boo)
            {
                //kontrol ifademiz boo yoksa hata vererek programı kapatır.
                Console.Error.WriteLine("ERROR!");
                Environment.Exit(0);
            }
            else
            {
                //boo ifadesi varsa, onu bulana kadar tokeni 1 arttırır
                sayac = sayac + 1;
            }
            C();//Gramere göre en sonda cümle çalışır.

        }
       //While döngüsü fonksiyonu
        public static void W()
        {
            E(); //önce aritmetik ifade çalışacak
            bool doruluk = false;//doğruluk kontrolü için tanımladık
            //? yoksa, kontrol doğrudur ve while döngüsü başlar
            if (TokenGetir[sayac]!='?')
            {
                Console.WriteLine("While Döngüsü: " +TokenGetir[sayac]);
                doruluk = true;
            }
            if (doruluk)
            {
                sayac = sayac + 1;
            }
            C(); //en son cümle çalışır
        }
//Atama cümlesi tanımlandı
        public static void A()
        {
            bool d = false, d1 = false;
            K(); //önce küçük harfler ifadesi çalışacak
            if (TokenGetir[sayac] == '=')
            {
                Console.WriteLine("Atama Cümlesi: " + TokenGetir[sayac]);
                A();
                d = true;
            }
            if (d)
            {
                sayac = sayac + 1;
            }
            if (TokenGetir[sayac] == ';')
            {
```

```
                Console.WriteLine("Atama Cümlesi: " + TokenGetir[sayac]);

                d1 = true;
            }
            if (d1)
            {
                sayac = sayac + 1;
            }

        }
            //Çıktı cümlesi tanımlandı
        public static void Ç()
        {
            //"<" koşulu en başta belirtildiği için yazmadık
            E();//önce E çalışır. Bu gramerin kuralıdır.
            bool a = false;
            if (TokenGetir[sayac] == ';')
            {
                a = true;
            }
            if (a)
            {
                sayac = sayac + 1;
            }

        }
        //girdi cümlesi tanımlandı
        public static void G()
        {    //">" koşulu en başta belirtildiği için yazmadık
            K();//önce K çalışır. Bu gramerin kuralıdır.
            bool c = false;
            if (TokenGetir[sayac] == ';')
            {
                Console.WriteLine("Girdi Cümlesi: " + TokenGetir[sayac]);
            }
            if (c)
            {
                sayac = sayac + 1;
            }

        }
//aritmetik ifade tanımlandı
        public static void E()
        {
            T(); //önce T çalışır. Bu gramerin kuralıdır.
            bool z = false;
            if (TokenGetir[sayac] == '+' || TokenGetir[sayac] == '-')
            {
                z = true;
                Console.WriteLine("Aritmetik İfade: " + TokenGetir[sayac]);
            }
            if (z)
            {
                sayac = sayac + 1;
            }
            T();//en son T çalışır. Bu gramerin kuralıdır.

        }
         //Çarpma-Bölme-Modulo tanımlandı
        public static void T()
```

```
        {
            U(); //önce U çalışır. Bu gramerin kuralıdır.
            bool k = false;
            if(TokenGetir[sayac]=='*'|| TokenGetir[sayac] == '/'|| TokenGetir[sayac]
== '%')
            {
                k = true;
                Console.WriteLine("Çarpma-Bölme-Mod ifadesi: " + TokenGetir[sayac]);
            }
            if (k)
            {
                sayac = sayac + 1;
            }
            U();
        }
        public static void U()
        {
            F(); //önce F çalışır. Bu gramerin kuralıdır.
            bool s = false;
            if (TokenGetir[sayac] == '^')//üs ifadesi varsa
            {
                Console.WriteLine("Üs ifadesi: " + TokenGetir[sayac]);
                s = true;
            }
            if (s)
            {
                sayac = sayac + 1;
                U();
            }
            else
            {
                F();
            }

        }
        public static void F()
        {
            bool s1 = false, s2 = false;
            if(TokenGetir[sayac]=='(' && TokenGetir[sayac] == ')') //'(' ve ')'
ifadeleri varsa, E() Çalışır.
            {
                E();
            }

            for(char i = '0'; i <= '9'; i++)
            { //ifadede rakam varsa, rakam ifadesinde çalışır

                if (TokenGetir[sayac] == i)
                {
                    Console.WriteLine("Rakam ifadeleri: " + TokenGetir[sayac]);

                    s1 = true;
                }
            }
            if (s1)
            {
                R();
            }
//ifadede küçük harf varsa, küçük harf ifadesinde çalışır
```

```
            for(char i = 'a'; i <= 'z'; i++)
            {
                if (TokenGetir[sayac] == i)
                {
                    s2 = true;
                }
            }

            if (s2)
            {
                K();
            }
        }

//Küçük harfleri tanımladık
        public static void K()
        {
            for (char i='a';i<='z';i++)
            {
//ifadede küçük harf varsa, küçük harf ifadesinde çalışır

                if (TokenGetir[sayac] == i)
                {
                    Console.WriteLine("küçük harf ifadeleri: " + TokenGetir[sayac]);
                    break;
                }
            }
            sayac = sayac + 1;

        }

//Rakam ifadelerini tanımladık
        public static void R()
        {
            for(char i = '0'; i <= '9'; i++)
            {
//ifadede rakam varsa, rakam ifadesinde çalışır
                if (TokenGetir[sayac] == i)
                {
                    Console.WriteLine("Rakam ifadeleri: " + TokenGetir[sayac]);
                }
            }
            sayac++;
        }

    }
       }
```