PROGRAMMING LANGUAGE

Path\_maker Language Description

Emre ŞAHİN-05170000067

Berna BİRCAN-05170000121

İçindekiler:

Analiz……………………………………………………….2-5

-Lexical Analyzer……………………………………………………2

-Parser……………………………………………………………….3-5

Yazılım Geliştirme İçin Harcanan Süreler…….6

Kaynak Kod…………………………………………….7-24

Çıktılar………………………………………………….25-34

-X.pmk…………………………………………………………………25

-Code.lex……………………………………………………… 26-27

-Consol……………………………………………………………28-34

Analiz

Problemin Tanımı:

Path\_maker klasör ağaçları oluşturmak için yazılmış bir kodlama dildir. Bizden path\_maker dili için interpreter yazmamız istenmektedir.

Lexical Analyzer

Konsoldan input olarak verilen kaynak kod ismini kullanıcıdan alıyoruz ve dosya isminin sonuna “.pmk” uzantısını ekliyoruz. Daha dosyadan okuma işlemini başlatıyoruz. Dosyadan gelen bütün karakterleri dinamik olarak oluşturduğumuz global bir değişken olan “file” dizisine atıyoruz. Böylelikle fonksiyonlarımızda “file” dizisine ulaşabiliyoruz.

Lexical analyzer için 3 fonksiyon kullanıyoruz: “isPath()”,”isCommand()”,”isOther()”.

“isPath()” fonksiyonumuzda, gelen karakterlerle oluşturulan kelime path data type ise ekrana ve “code.lex” dosyasına yazdırılır. Herhangi bir lexical hata bulunması durumunda kullanıcıya bildirilerek programdan çıkılır.

“isCommand()” fonksiyonumuzda, gelen karakterlerle oluşturulan kelime command ise ekrana ve “code.lex” dosyasına yazdırılır. Herhangi bir lexical hata bulunması durumunda kullanıcıya bildirilerek programdan çıkılır.

”isOther()” fonksiyonumuzda, gelen karakter “{ “ , ” }” , ” ; ” dan herhangi biriyse ekrana ve code.lex dosyasına yazdırılır .Herhangi bir lexical hata bulunursa bulunması durumunda kullanıcıya bildirilerek programdan çıkılır.

“file” dizisinin sonuna ulaşana kadar bu fonksiyonlar sıra ile çağırılır.

Lexical Errors:

Path data type’ da path oluşturma kurallarına aykırı bir path varsa kullanıcıya bildirilir ve program kapanır.

Path\_maker diline ait olmaya kelimler varsa kullanıcıya bildirilir ve program kapanır.

Parser

Code.lex dosyasından gelen bütün token’ler ve tipleri için iki boyutlu global “fullTokenList” ve “fullTokenTypeList” dizilerini tanımladık böylelikle fonksiyonlarımızda bu dizileri kullandık. “fullTokenList”in içine token’leri ; “fullTokenTypeList”in içine ise token tiplerini yerleştirdik.

Bulunduğumuz dosyanın konumunu tutmak için global “CurrentDirectory” dizisini ve fonksiyonlarımızda klasör kontrollerini yapmak için “CurrentDirectory”nin kopyası olan “TempCurrentDirectory” dizisini kullandık.

Önemli Not: Root path seçimini projenin kodunun bulunduğu konum olarak belirledik ve root’dan daha geriye gitmeyi engelleyerek kullanıcıya uyarı mesajı verdirdik. Eğer root’da iken \* karakteri gelirse, kullanıcı uyarı mesajı ile karşılaşcaktır. Örnek olarak root’da iken if<\*/deneme> gelirse if işlemi false olacaktır fakat ifnot<\*/deneme> gelirse if in sonucu false olduğu için ifnot işleminin sonucu true olcağı için program ifnot bloğunu çalıştıracaktır.

Hangi token da olduğumuzu bulmak için ise “globalCounter” değişkenimizi kullandık. Her yeni token okuduğumuz da o anki fonksiyonun içinde “globalCounter” bir arttırılır.” globalCounter”, toplam token sayısına eşit ise parser işlemi başarılı bir şekilde gerçekleşmiştir.

Parse işlemi için recursive descent metodunu kullandık. Bu metodu uygulamak için BNF oluşturduk:

ProgramBegin() -> ifStatement() | ifNotStatement() | commandStatement() | NULL

ifStatement() -> "if" + pathStatement() + commandStatement() | blockStatement()

ifNotStatement() -> "ifnot" + pathStatement() + commandStatement() | blockStatement()

pathStatement() -> "<" + "abc...z,\*,\_,/" + ">"

commandStatement() -> "make" | "go" + pathStatement() + ";" + ProgramBegin()

blockStatement() -> "{" + ProgramBegin() + "}" + ProgramBegin()

BNF’i kullanabilmek için 6 farklı fonksiyon oluşturduk. Bunlar: ProgramBegin(), ifStatement(),ifNotStatement(),commandStatement(),pathStatement(), blockStatement().

“ProgramBegin()” fonksiyonumuzda, gelen token “if” ise ifStatement() ; command ise “commandStatement()” çağırılır. Eğer bunlardan biri değilse fonksiyondan çıkış yapılır.

“ifStatement()” fonksiyonumuzda, “if”ten sonra gelen token path ise “pathStatement()” çağırılır. “CurrentDirectory” deki konumumuz ve if’in yanındaki path token , arasına “/” konularak birleştirilir. Bu işlemin sonucu klasör ağacında varsa ve bir command geliyorsa “commandStatement()”i çağırırız ; block geliyorsa “blockStatement()”i çağırırız.Eğer işlemin sonucu klasör ağacında yoksa, gelen token’lar içindeki “LeftCurlyBracket“sayısı “RightCurlyBracket” sayısına eşit olmadığı sürece “globalCounter” bir arttırılır. Böylelikle “if” bloğu atlanmış olur.

“ifNotStatement()” fonksiyonumuzda, “ifnot”tan sonra gelen token path ise “pathStatement()” çağırılır. “CurrentDirectory” deki konumumuz ve if’in yanındaki path token , arasına “/” konularak birleştirilir. Bu işlemin sonucu klasör ağacında yoksa ve bir command geliyorsa “commandStatement()”i çağırırız ; block geliyorsa “blockStatement()”i çağırırız.Eğer işlemin sonucu klasör ağacında varsa, gelen token’lar içindeki “LeftCurlyBracket” sayısı “RightCurlyBracket” sayısına eşit olmadığı sürece “globalCounter” bir arttırılır. Böylelikle “ifnot” bloğu atlanmış olur.

“pathStatement(char data[])” fonksiyonumuzda, path data type’ı ayrıştırılırken herhangi bir parser hatası bulunmazsa; bu fonksiyona gelen “data”, eğer “make” ise path içindeki klasörleri sırayla oluşturur ve bu sırada \* gelirse bir üst konuma çıkıp oradan oluşturmaya devam eder. Fonksiyona gelen “data” “go” ise verilen path içindeki konuma sırayla gidilir ve bu sırada \* gelirse bir üst konuma çıkıp oradan ilerlemeye devam eder.

“commandStatement()” fonksiyonumuzda, gelen token “make” ya da “go” ise “pathstatement()”e gidilir. Daha sonra “globalCounter” arttırılarak gelen token’in “;” olması beklenir. Son olarak “pathStatement()” çağırılarak BNF’in başına dönülür.

“blockStatement()” fonksiyonumuzda, gelen token “}” ise “globalCounter” arttırılır ve “ProgramBegin()” çağırılır.

Parser Errors:

“İf” veya “ifnot”tan sonra path gelmez ise kullanıcıya hata bildirilir ve programdan çıkılır.

“İf” veya “ifnot”tan sonra gelen path’ten sonra command veya block yoksa ise kullanıcıya hata bildirilir ve programdan çıkılır.

“make” veya “go” komutlarından sonra path gelmezse kullanıcıya hata bildirilir ve programdan çıkılır.

“make” veya “go”dan sonra gelen path’ten sonra “;” yoksa kullanıcıya hata bildirilir ve programdan çıkılır.

Her bloğun başında ve sonunda “{“, “}” olmak zorundadır; yok ise kullanıcıya hata bildirilir ve programdan çıkılır.

Root dizindeyken bir üst seviyeye çıkılmak istenirse uyarı mesajı verilir.

“make” ile oluşturmak istenen dosya zaten varsa uyarı mesajı verilir.

“go” ile gidilmek istenen dosya kısmen veya tamamen yoksa uyarı mesajı verilir.

Yazılım Geliştirme İçin Harcanan Süreler:

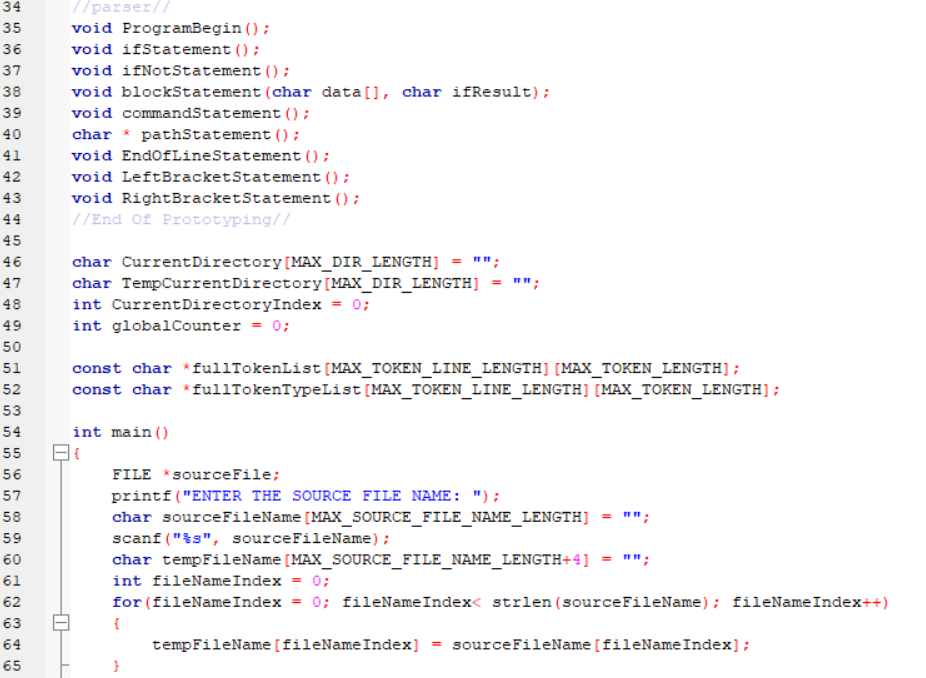
Analiz için yaklaşık 1 saat

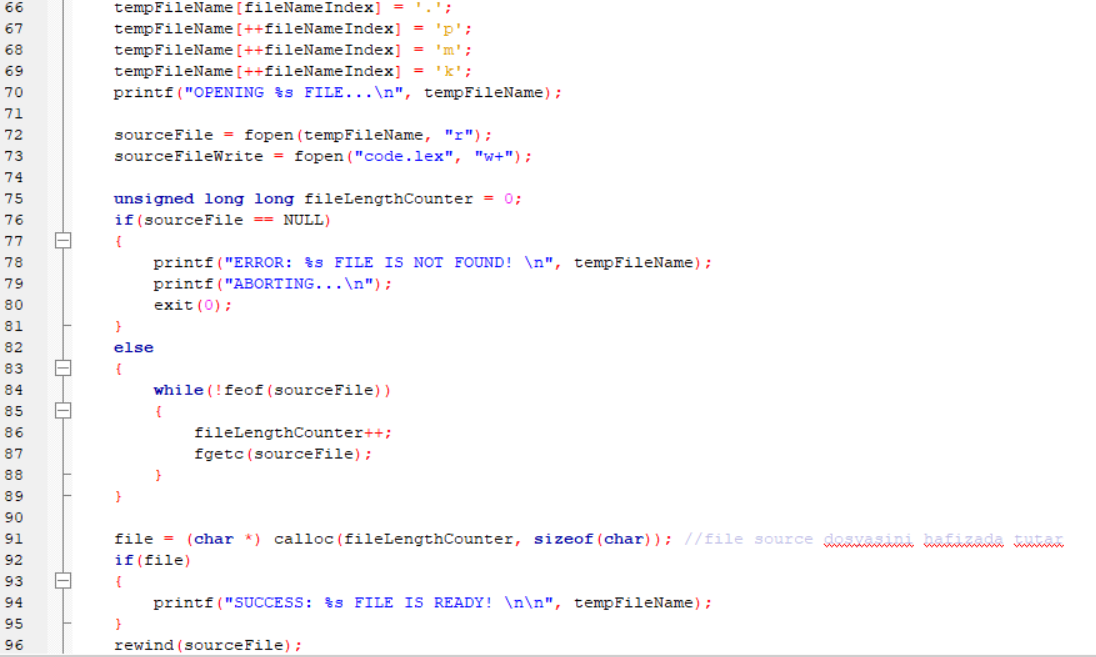
Gerçekleştirim için yaklaşık 3-4 gün

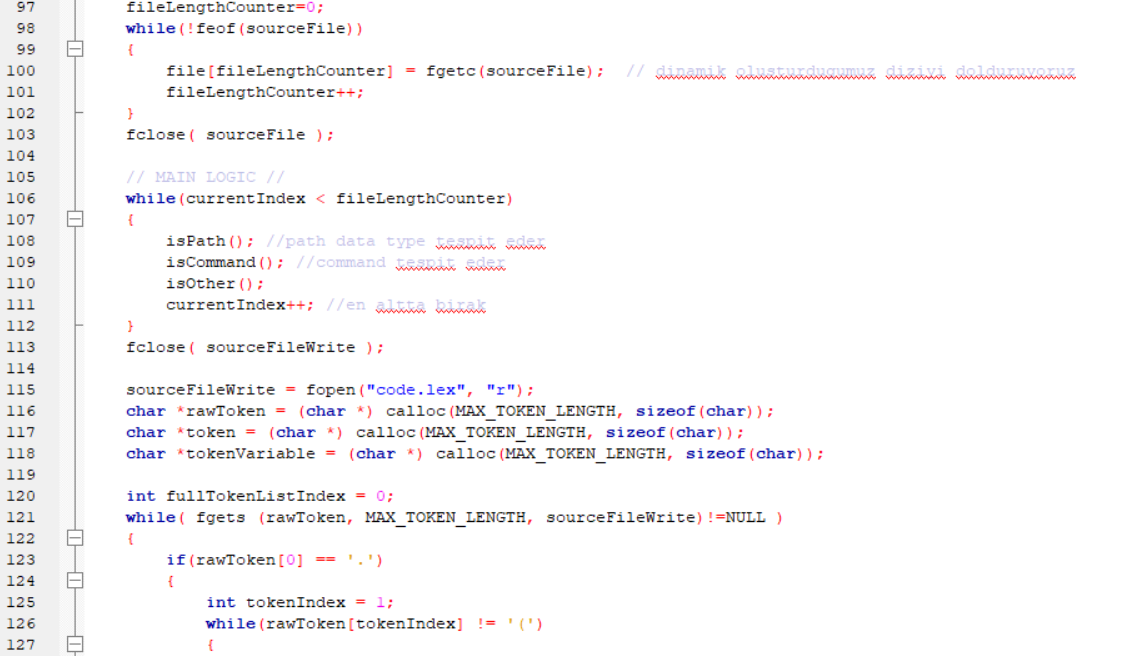
Test için yaklaşık 4 saat

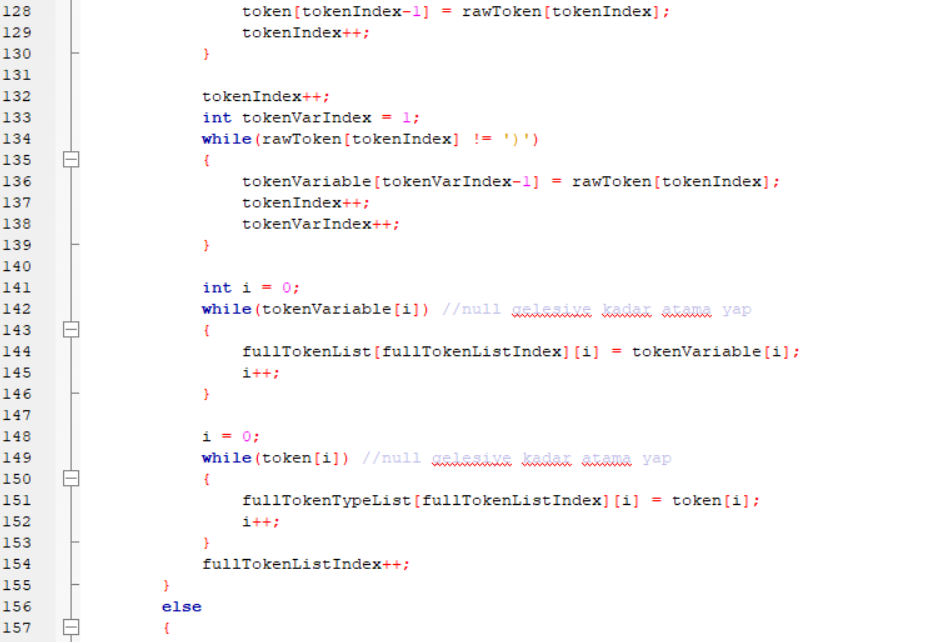
Raporlama için yaklaşık 2 saat

Kaynak Kod







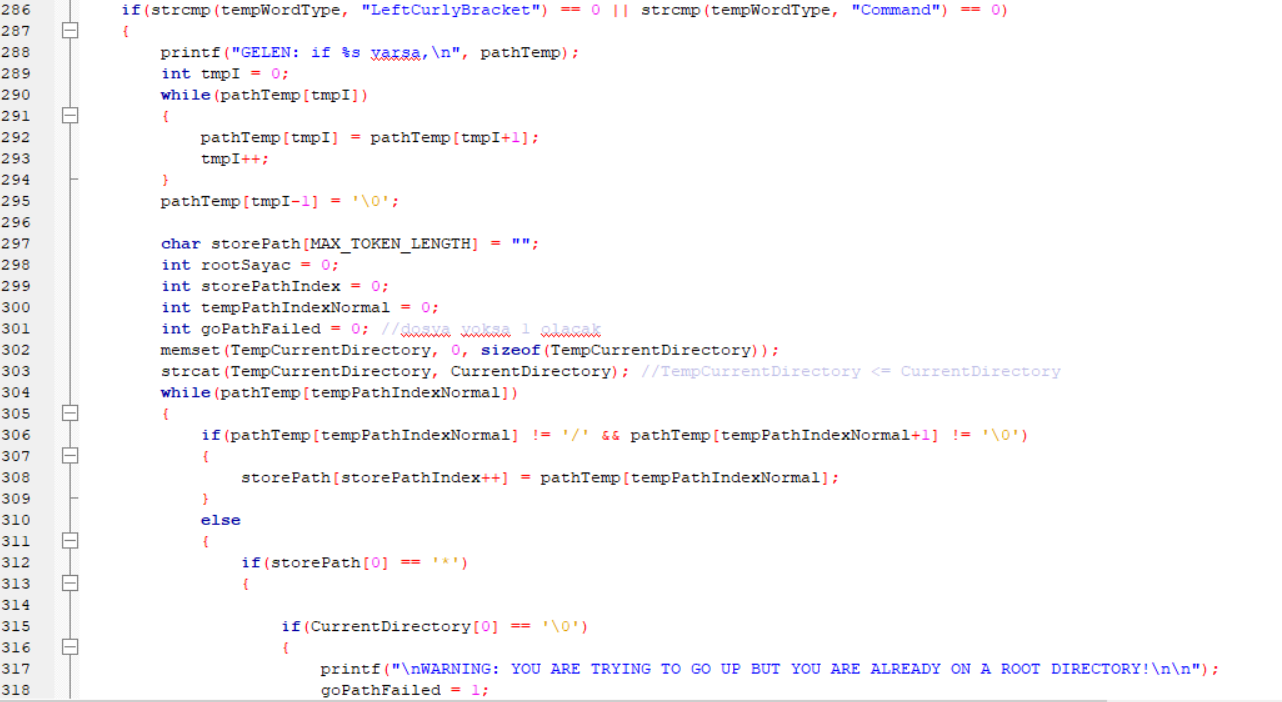




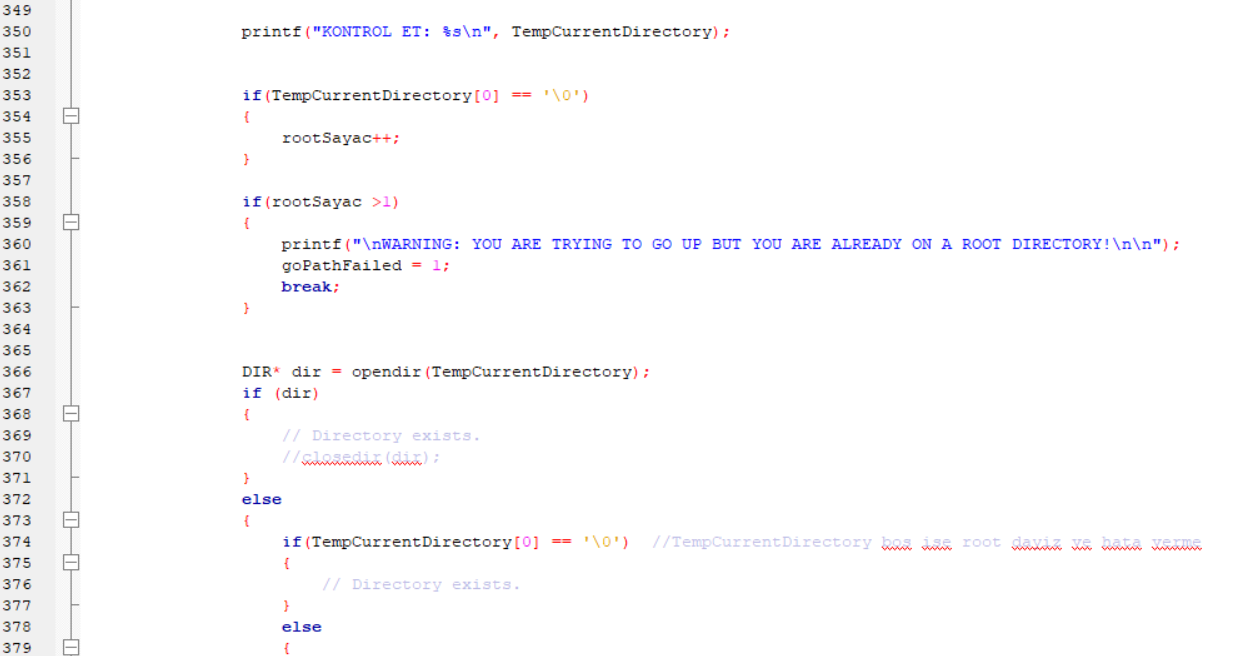


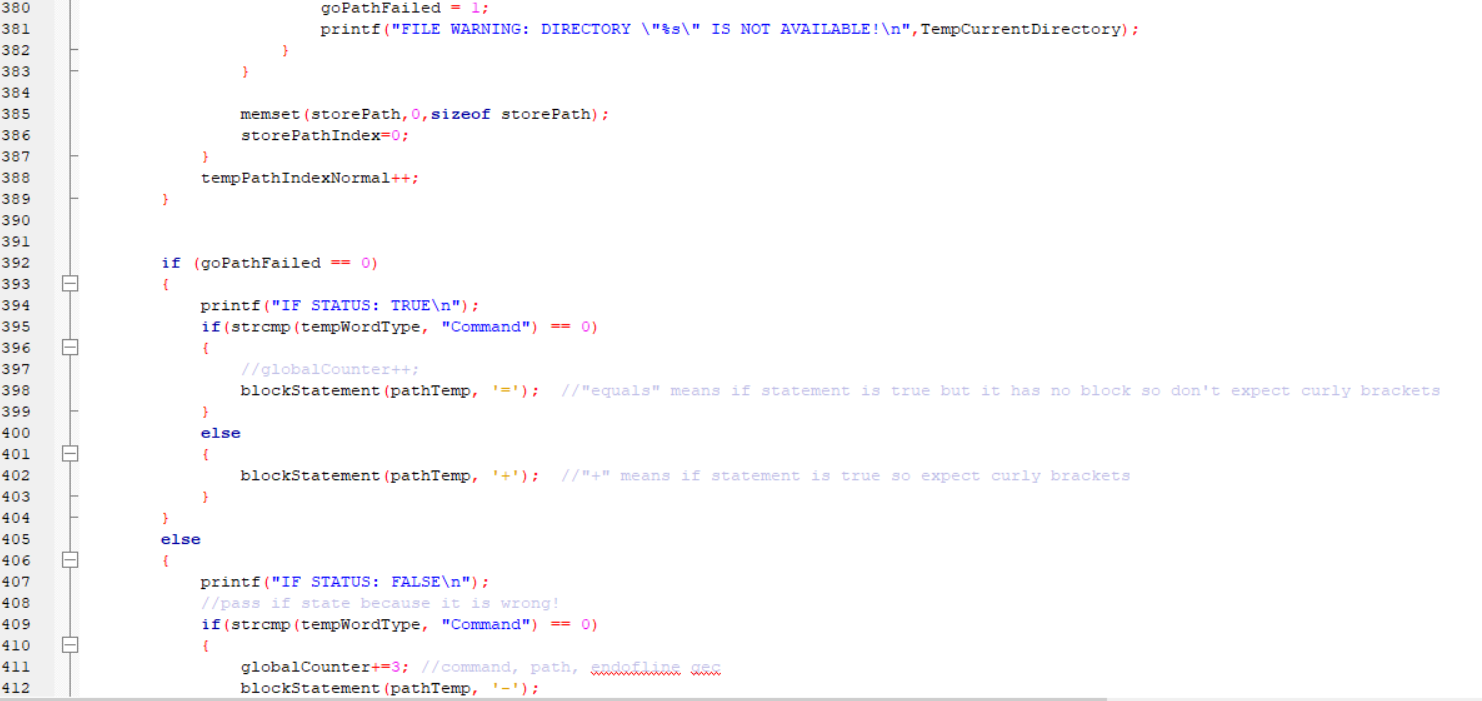








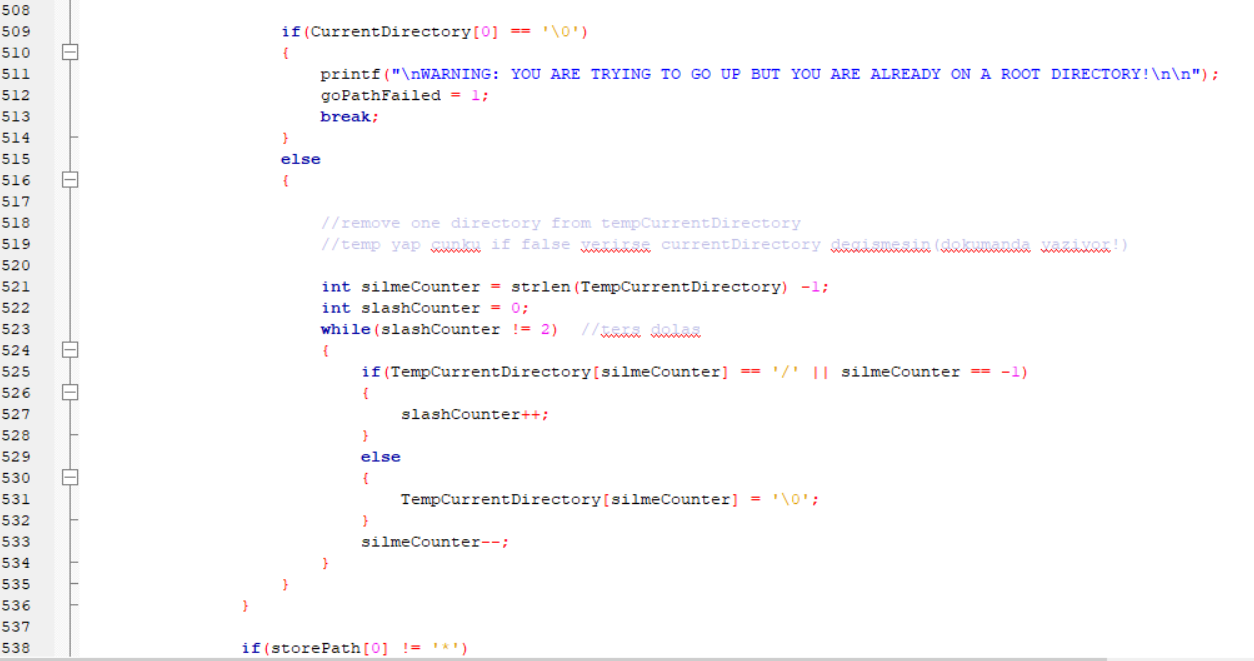


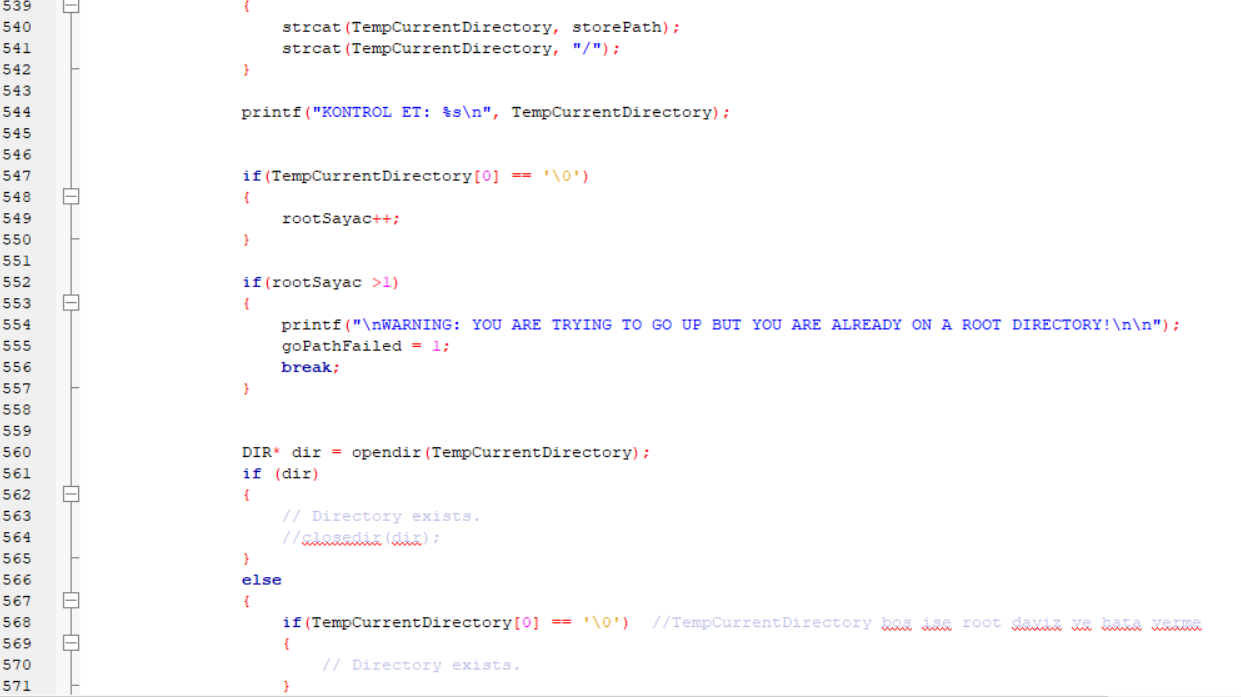


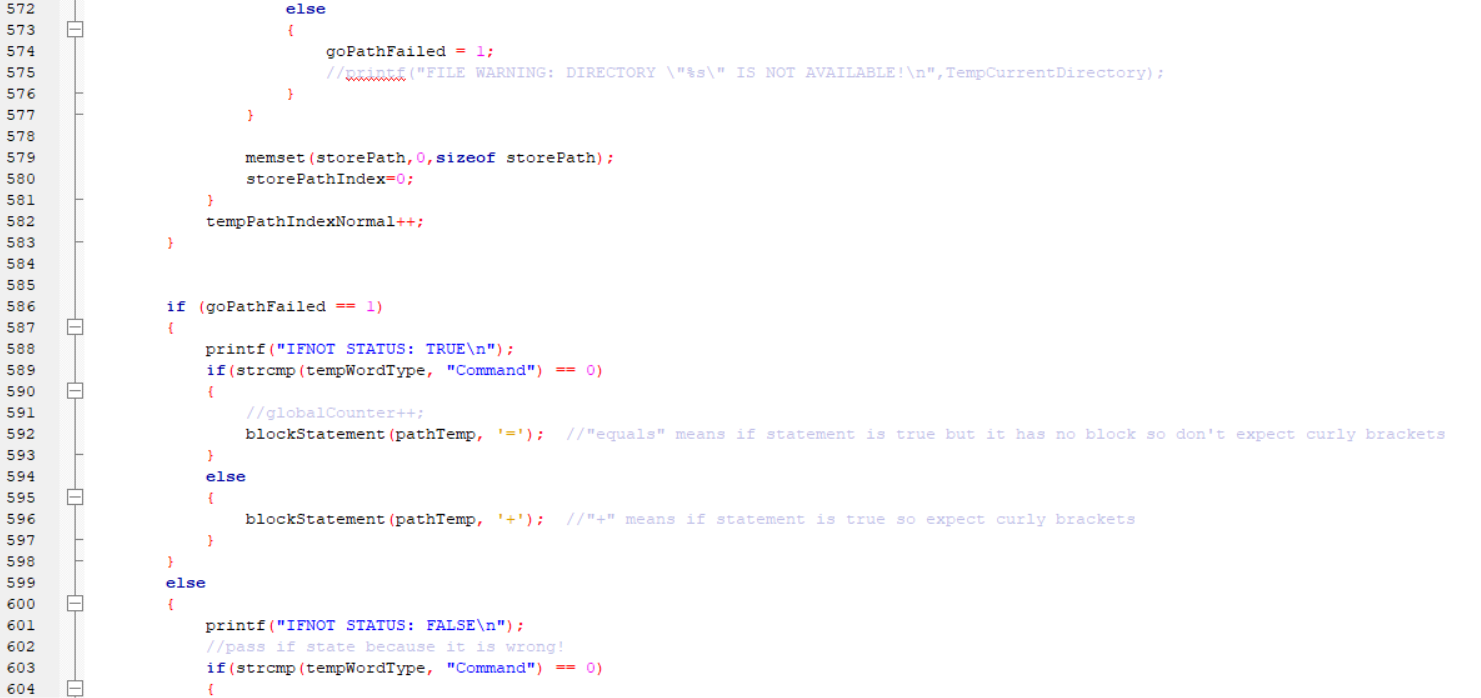












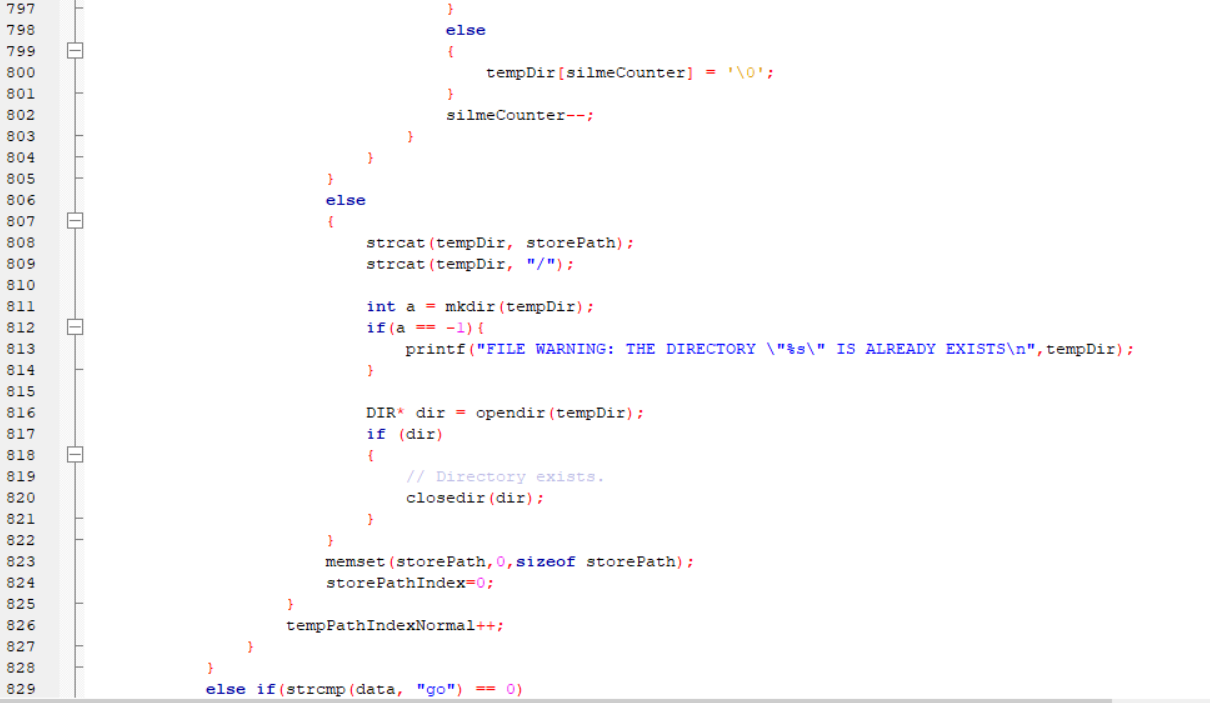


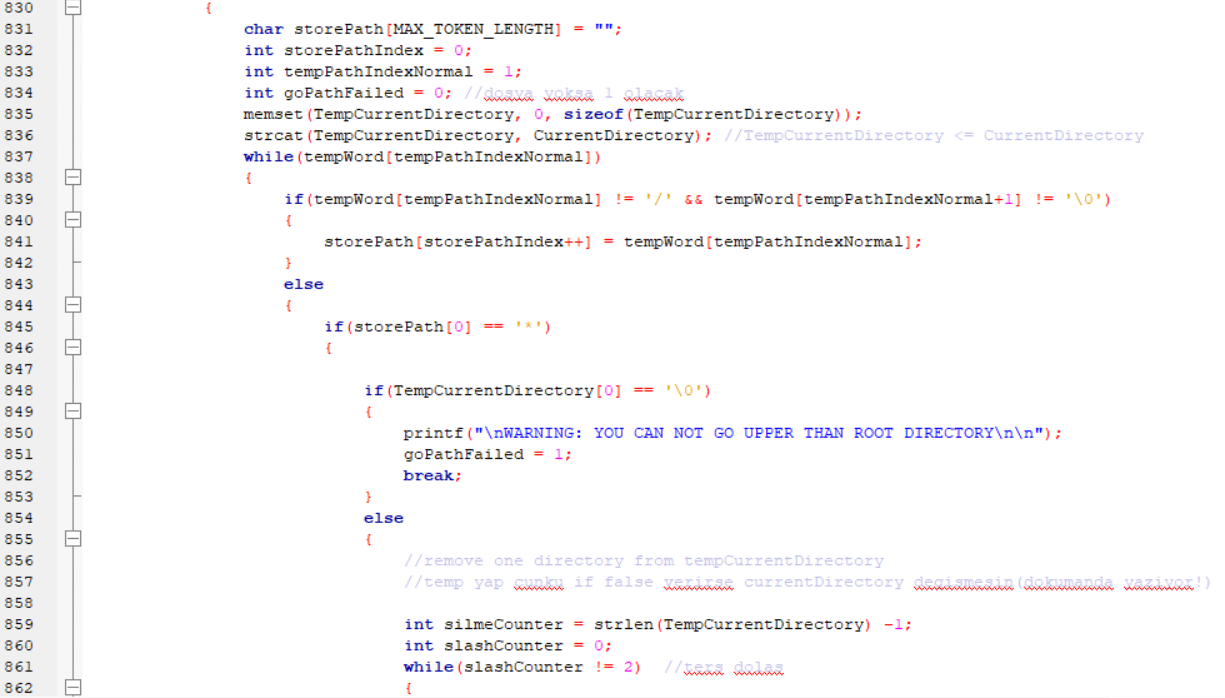






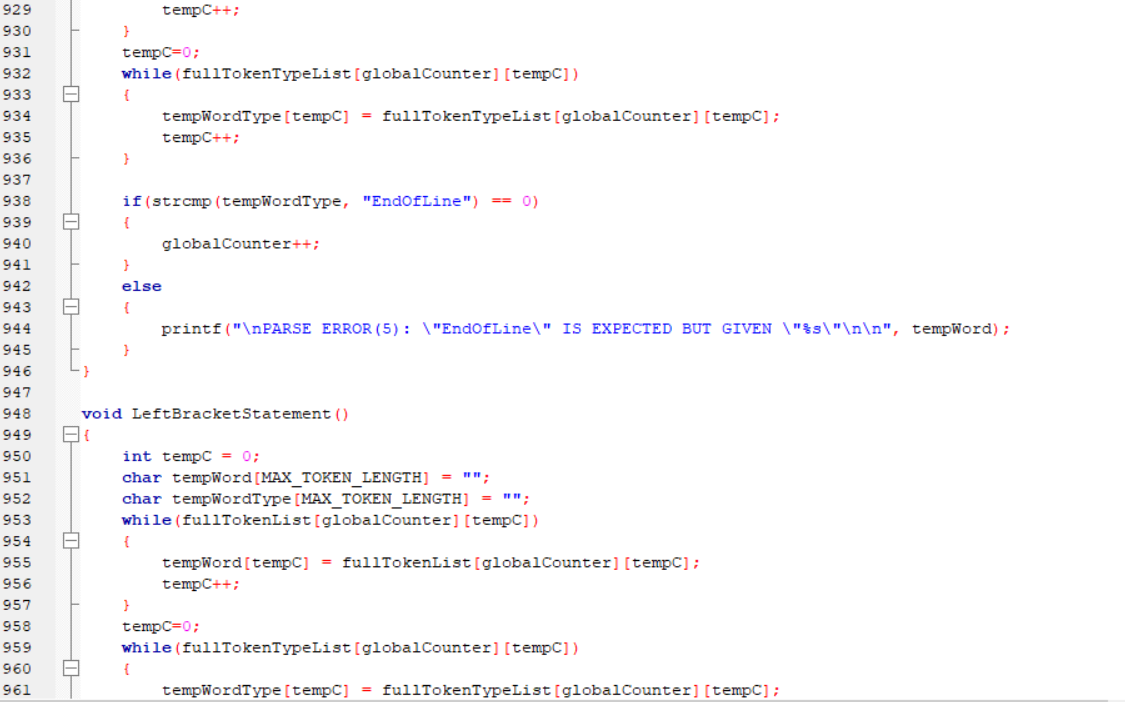


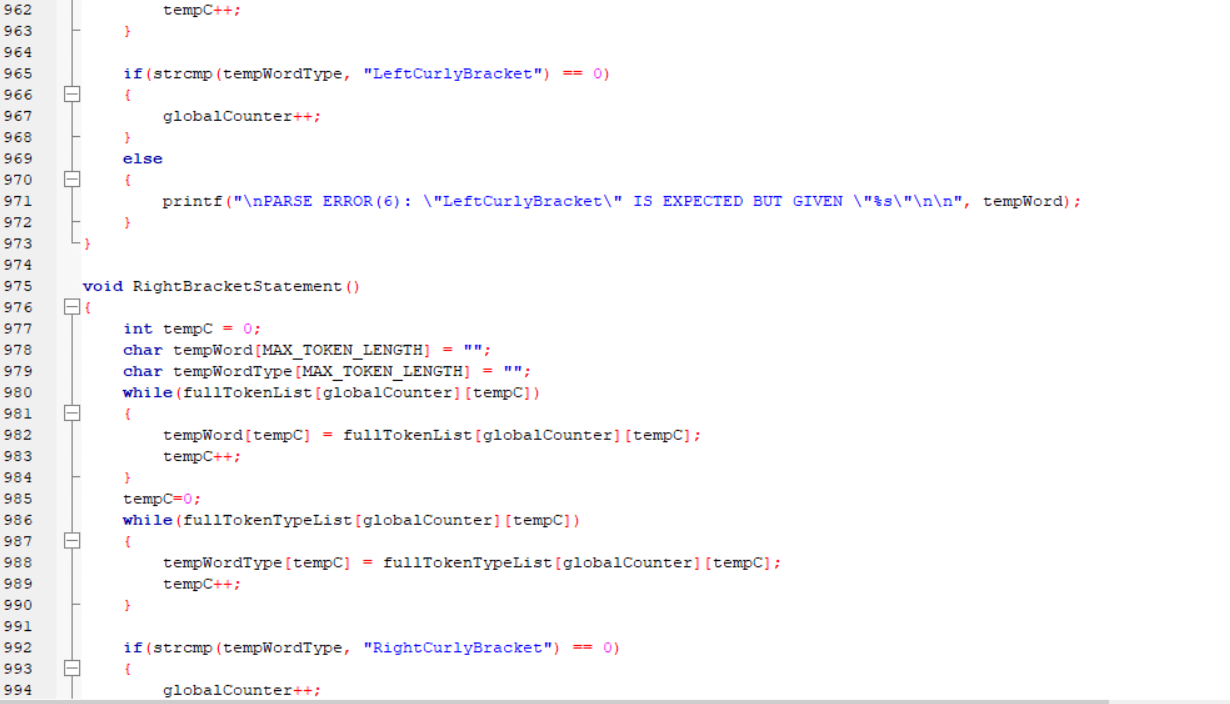


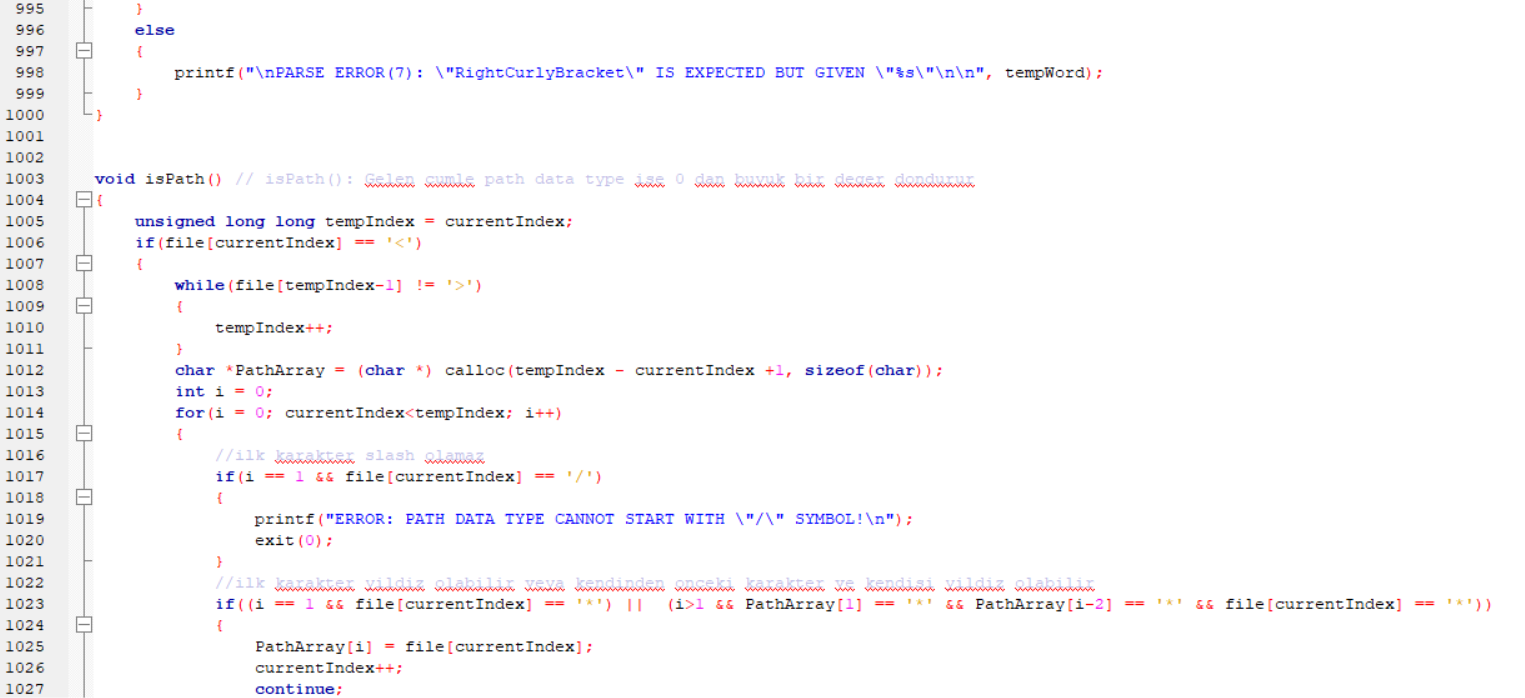


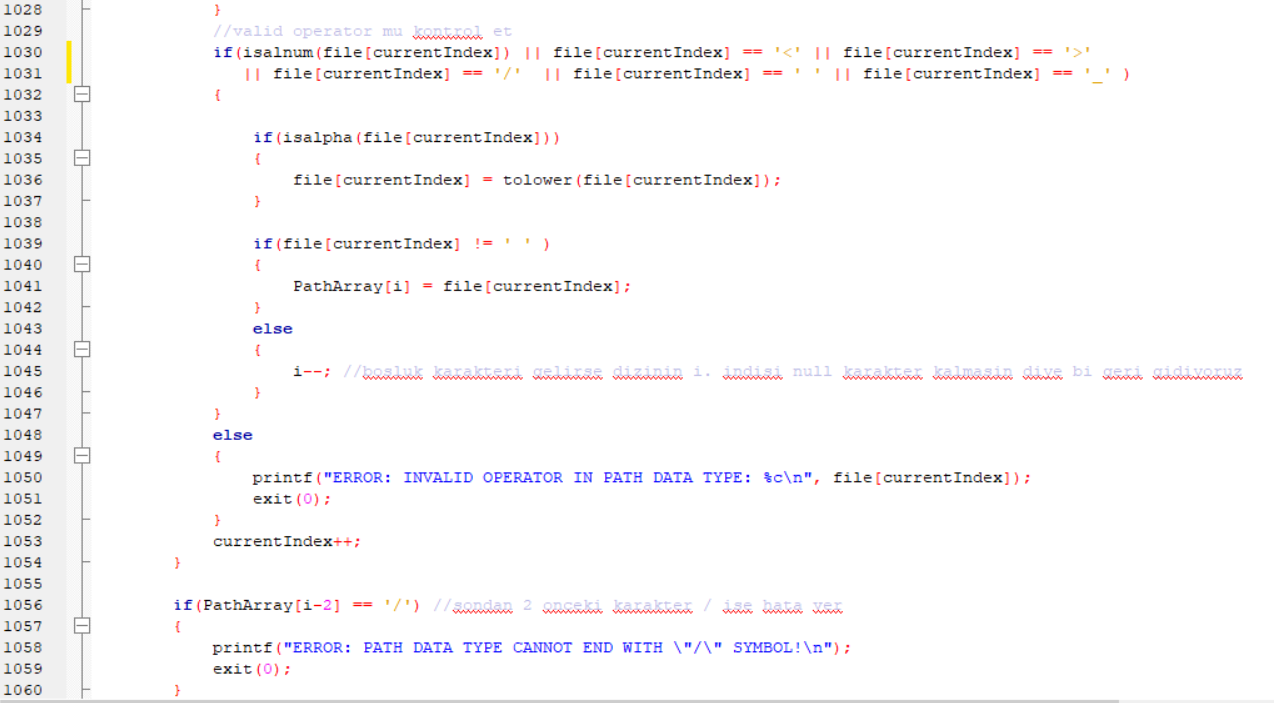














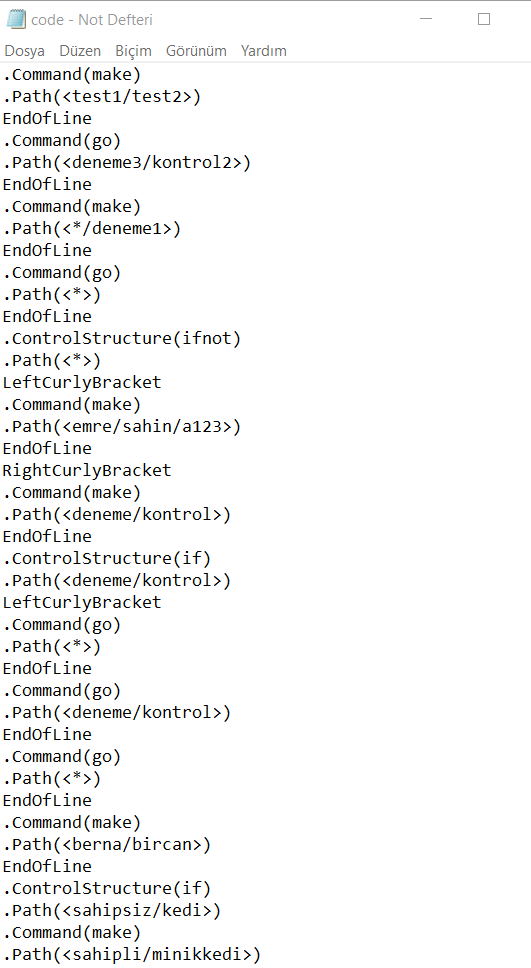


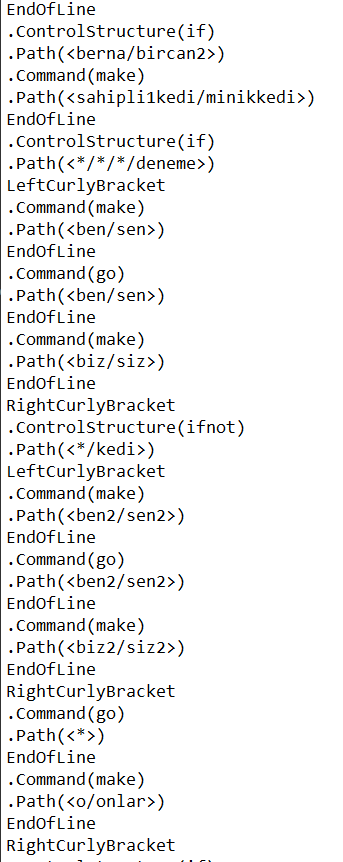
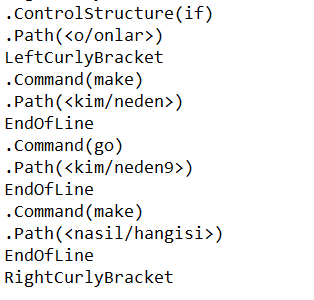
Çıktılar

X.pmk:



Code.lex:



Consol:



