KARADENİZ TEKNİK ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
BİLGİSAYAR SİSTEMLERİ LABORATUARI

BISON AYRIŞTIRICI (PARSER) ÜRETECİ

Bison, durumdan bağımsız gramerler (CFG, Context-free grammar) için LALR (Lookahead LR) ya da GLR (Generalized LR) ayrıştırma algoritmalarını kullanan ayrıştırıcılar üreten genel amaçlı bir ayrıştırıcı üretecidir (parser generator). Ayrıştırıcı üreteçleri sayesinde, CFG kullanılarak ifade edilebilen dilleri ayrıştırabilen bilgisayar programları (hesap makinesi, derleyiciler, vs) üretilebilir. Bu föyde, CFG ile ifade edilebilen bir dili işleyebilen bir ayrıştırıcının, Bison ayrıştırıcı üreteci kullanılarak nasıl oluşturulacağı üzerinde durulacaktır.

1. Durumdan Bağımsız Gramerler

Bison 'un, belli bir paterne sahip girdileri işleyebilen bir ayrıştırıcı program üretebilmesi için, program girdilerinin CFG kullanılarak ifade edilebilmesi gerekir. CFG, giriş diline ait kelimelerin sözdizimsel olarak gruplanması (syntactic grouping) ve her bir sözdizimsel gruplamanın hangi tür kelimelerden oluşacağını belirleyen kuralların (production) tanımlanmasıyla elde edilir. C dilindeki *ifade* (expression) yapıları sözdizimsel gruplamaya bir örnektir. C dilindeki ifade yapılarını tanımlamak için aşağıdaki kurallar kullanılabilir:

Kural	Örnek
Fonksiyon çağrısı bir ifadedir	*a = sum(b,c);* //sum(b,c) bir ifadedir
Tamsayı bir değer bir ifadedir	*a = b;* //b, bir ifadedir
Bir ifade, "ifade+ifade" formatında olabilir	*a = sum(b,c) + b;* //"sum(b,c) + b" bir ifadedir

Tablo 1. C programlama dilindeki ifade oluşturma kurallarından bazıları

Sözdizimsel gruplama için kuralların oluşturulmasında kullanılan en yaygın gösterim sistemi BNF (Backus Naur Form)'dir. BNF kullanılarak ifade edilebilen tüm gramerler, durumdan bağımsız gramer olarak isimlendirilir.

Durumdan bağımsız dillerdeki her bir sözdizimsel gruplama *sembol* olarak isimlendirilir. Dildeki daha küçük yapıların bir araya gelmesiyle oluşan sembollere *terminal-olmayan* (nonterminal) sembol adı verilir. Diğer bir deyişle, terminal-olmayan sembollerin her biri için ayrı bir gramer kuralı mevcut olmalıdır. Eğer bir sembol daha küçük dil yapılarına bölünemiyorsa *terminal* sembol olarak adlandırılır. Ayrıştırılacak giriş dilindeki terminal sembollere karşılık düşen kısımlara *token* ismi verilir. Tablo 2, token 'lerine ayrıştırılmış bir C kodunu göstermektedir:

```
void                    // 'void' anahtar kelimesi
cube (int x)            // belirteç, aç-parantez, 'int' anahtar kelimesi, belirteç, kapat-parantez
{                       // aç-büyük-parantez
  return x * x * x;     // 'return' anahtar kelimesi, belirteç, asterisk, belirteç, asterisk, belirteç, noktalı virgül
}                       // kapat-büyük-parantez
```

Tablo 2. Token 'lerine ayrıştırılmış bir C kodu

Durumdan bağımsız gramerlerde, *başlangıç sembolü* (start symbol) adı verilen özel bir terminal-olmayan sembol mevcuttur. Bir kelime dizisinin, ait olduğu CFG tarafından kabul edilebilmesi için, kökü başlangıç sembolü olan bir sözdizimsel ağacın (syntax tree) üretilebilmesi gerekir.

2. Bison Gramer Tanımlama Dosyası

Bison'un durumdan bağımsız bir gramer için bir ayrıştırıcı üretebilmesi için, ilgili gramer kurallarının Bison sözdizimine uygun olacak bir şekilde bir *gramer tanımlama dosyasında* tanımlanması gerekir. Bison sözdiziminde, terminal-olmayan bir sembol, tamamı küçük harfler içeren bir belirteç (identifier) ile temsil edilir. `expr`, `stmt` ve `declaration` bu belirteçlere örnek olarak verilebilir. Terminal sembol isimleri ise terminal-olmayan sembollerden ayırt edilebilmesi için tamamı büyük harf içerecek şekilde belirlenir. `INTEGER`, `IDENTIFIER`, `IF` ve `RETURN` terminal sembol isimlerine örnek olarak gösterilebilir. Ayrıca, sadece bir karakter içermesi durumunda, terminal sembol tırnak işaretleri (’’) içerisinde gösterilir. Aşağıdaki satırlar, C dilindeki `return` deyiminin (statement) Bison sözdiziminde nasıl ifade edildiğini göstermektedir:

```
stmt: RETURN expr ';'
    ;
```

Yukarıdaki `expr` sembolünden sonra tırnak işaretleri arasında içerilen noktalı virgül, C dilindeki `return` deyiminin sözdizimine aittir. Bununla birlikte, `stmt` belirtecinden sonra gelen `:` ve tırnak işaretleri arasında içerilmeyen `;` karakterleri ise Bison sözdiziminde ifade edilen her gramer kuralında bulunması gereken karakterlerdir. Bison gramer tanımlama dosyasının genel formatı Tablo 3 'te gösterilmiştir:

```
%{
Prolog
%}

Bison bildirimleri

%%
Gramer kuralları
%%
Epilog
```

Tablo 3. Bison gramer tanımlama dosyası formatı

'`%%`', '`%{`' ve '`%}`' sembolleri, her Bison gramer tanımlama dosyasında bulunur ve farklı bölümleri birbirinden ayırmak için kullanılır.

`Prolog` bölümü, semantik aksiyonlarda kullanılan veri tiplerini ve değişkenleri tanımlamak için kullanılır. Ayrıca, bu bölüm, C önişlemci (preprocessor) komutları kullanılarak çeşitli makrolar tanımlamak amacıyla da kullanılabilir. Bunun yanında, veri tipleri, makro ve değişken tanımları içeren *başlık* (header) dosyalarına ait `#include` önişlemci komutları, `yylex` ve `yyerror` token üreteci (lexical analyzer) ve hata raporlama fonksiyonlarına ait bildirimler (declaration) `Prolog` bölümünde temin edilmelidir. `Bison bildirimleri` bölümü, terminal ve terminal-olmayan sembollerin isimlerini listelemek için kullanılır. Ayrıca, operatör öncelikleri ve çeşitli sembollere ait semantik değerlerin veri tipleri de bu bölümde belirlenir. `Gramer kuralları` bölümü, terminal-olmayan

sembollerin, daha küçük dil yapılarından nasıl meydana gelebileceğini tanımlamak için kullanılır. `Epilog` bölümü, genellikle `Prolog` bölümünde bildirimi yapılmış olan fonksiyonların gövdelerini (body) içerir.

3. Semantik Aksiyonlar

Bir terminal sembol, `INTEGER`, `IDENTIFIER` ya da `','` gibi semboller ile ifade edilir. Bu semboller, herhangi bir token dizisinin doğru sözdizimine sahip olması için nasıl bir yapıda olması gerektiğinin belirlenmesinde kullanılır. Fakat, ayrıştırıcı bir programın, herhangi bir token dizisinin, CFG ile temsil edilen dile ait olup olmadığını kontrol etmesi tek başına yeterli değildir. Ayrıştırıcının, her bir terminal sembolün mevcut değerini de bilmesi gerekir. Örneğin, ayrıştırıcı, `INTEGER` pozisyonundaki `23`, `5456` ve `654378` değerlerini birbirinden ayırt edebilmelidir. Öte yandan, terminal semboller yanında terminal-olmayan semboller de semantik bir değere sahip olabilir.

Bison gramer tanımlama dosyasındaki bir gramer kuralı, C deyimlerinden oluşan bir semantik aksiyon kısmına sahiptir. Semantik aksiyon, terminal olmayan sembollerin semantik değerlerinin, sembolü oluşturan daha küçük dil yapılarının semantik değerleri hesaba katılarak hesaplanmasını sağlar. Herhangi bir gramer kuralının giriş token dizisinin bir altkümesiyle her eşleşmesinde, kurala ait semantik aksiyon ayrıştırıcı tarafından koşulur. Aşağıdaki gramer kuralı, C dilindeki bir ifadenin (expression) iki alt ifadenin toplamı olabileceğini göstermektedir:

```
expr: expr '+' expr        { $$ = $1 + $3; }
    ;
```

Yukarıdaki gramer kuralındaki semantik aksiyon, toplam ifadesinin semantik değerinin iki alt ifadenin semantik değerinden nasıl elde edileceğini göstermektedir.

4. LALR Gramerleri

Bison, hemen hemen tüm durumdan bağımsız gramer türlerini işleyebilse de, özellikle LALR(1) gramerlerini işleyebilmesi için optimize edilmiştir. LALR(1) gramerleri, girdinin ayrıştırılmış kısmına ve henüz ayrıştırılmamış kısmının belli bir bölümüne (lookahead) bakılarak, uygulanacak bir sonraki gramer kuralının ne olacağına karar verilebilmesine olanak sağlar. Bu nedenle, LALR(1) gramerleri deterministik gramerlerdir.

5. GLR Gramerleri

Bazı durumlarda, Bison 'un standart LALR(1) ayrıştırma algoritması, o ana kadar ayrıştırılan token dizisini esas alarak uygulanacak bir sonraki gramer kuralının ne olacağına karar veremez. Bu kararsızlığın nedenleri, uygulanması muhtemel birden fazla gramer kuralının mevcut olması ya da mevcut gramer kurallarından birinin hemen ya da daha fazla token okunduktan sonra uygulanması (reduction) arasında bir seçim yapılamamasıdır. İlk neden *reduce/reduce*, ikinci neden ise *shift/reduce* çelişkisi olarak bilinir. Bu çelişkilerin çözümlenmesi için GLR ayrıştırma algoritması kullanılır. GLR ayrıştırma algoritması her bir olasılığı değerlendirmek için ayrıştırıcıyı çoğullar (clone). Çoğullama süresince, her bir ayrıştırıcıya ait semantik aksiyonlar kaydedilir, fakat icra edilmez. Çoğullanmış ayrıştırıcılardan sözdizim hatasına neden olanlar semantik aksiyonları icra edilmeden sonlandırılır. Tablo 4 'teki Bison grameri Pascal programlama dilindeki tip bildirimlerini (type declarations) ayrıştırmaktadır:

```
%token TYPE DOTDOT ID

%left '+' '-'
%left '*' '/'

%%

type-decl : TYPE ID '=' type ';'
      ;

type : '(' id_list ')'
    | expr DOTDOT expr
    ;

id_list : ID
    | id_list ',' ID
    ;

expr : '(' expr ')'
    | expr '+' expr
    | expr '-' expr
    | expr '*' expr
    | expr '/' expr
    | ID
    ;
```

Tablo 4. Pascal programlama dilindeki tip bildirimlerini ayrıştıran Bison grameri

Örneğin, Bison, aşağıdaki Pascal satırını ayrıştırmak için LALR(1) ayrıştırma algoritmasını kullandığında reduce/reduce çelişkisi tespit eder:

```
type t = (a)..b;
```

Bunun nedeni, `(a)` tokeni için `type` terminal-olmayan sembolüne ait gramer kurallarından hangisinin (`'(' id_list ')'` ya da `expr DOTDOT expr`) uygulanması gerektiğine karar verilememesidir. Böyle bir durumda, LALR(1) ayrıştırma algoritması muhtemel gramer kurallarından gramer tanımlama dosyasında daha önce tanımlanmış olanını (`'(' id_list ')'`) bir sonraki uygulanacak gramer kuralı olarak tayin eder. Böyle bir karar mekanizması yukarıdaki doğru sözdizimine sahip token dizisinin yanlışlıkla reddedilmesine neden olur. Bu tarz yanlışlıkların önüne geçebilmek için gramer tanımlama dosyasına aşağıdaki satırlar eklenerek (`%%` 'dan hemen önce) GLR ayrıştırma algoritması aktifleştirilmelidir:

```
%glr-parser
%expect-rr 1
```

`%expect-rr 1` satırı, Bison 'un reduce/reduce çelişkileri için herhangi bir uyarı üretmemesini sağlar.

6. Ayrıştırıcı Dosyası (Parser File)

Bison, program çıktısı olarak gramer tanımlama dosyasındaki gramer kuralları tarafından tanımlanan dili ayrıştıran bir C kaynak dosyası üretir. Bu C kaynak dosyasına *ayrıştırıcı dosyası* adı verilir. Ayrıştırıcı dosyası içerisindeki C kaynak kodu, tanımlı gramer kurallarını esas alarak token 'leri

terminal-olmayan sembolleri oluşturmak amacıyla gruplar ve bazı değişiklikler yapılması koşuluyla ayrıştırıcı bir program (derleyici, yorumlayıcı, vs) üretmek amacıyla kullanılabilir.

Bison ayrıştırıcı dosyası, ismi `yyparse` olan ve gramer tanımlama dosyasındaki grameri gerçekleyen bir fonksiyon tanımı içerir. Bu fonksiyon, tamamıyla fonksiyonel bir ayrıştırıcı program üretmek için tek başına yeterli değildir. Ayrıca, Bison ayrıştırıcısı, yeni bir token 'e ihtiyaç duyduğunda *sözcüksel analizörü* (lexical analyzer) çağırır. Sözcüksel analizör kaynak kodunun Bison 'a temin edilmesi gerekir. Sözcüksel analizör, giriş dilindeki karakterleri gruplayarak token 'leri oluşturur. Bu fonksiyonlara ilave olarak, *hata raporlama* fonksiyonu (error reporting) ve her C kodunda bulunması gereken `main` fonksiyonunun da oluşturulması gerekir. `main` fonksiyonu, `yyparse` fonksiyonunu çağıracak şekilde düzenlenmelidir.

7. Bison Kullanımına Dair Adımlar

Bison kullanılarak, bir derleyici ya da yorumlayıcı üretilmesi için gerçeklenmesi gereken adımlar aşağıdaki gibidir:

1. Gramer tanımlama dosyası kullanılarak gramer kurallarının tanımlanması ve her bir gramer kuralı için C deyimleri kullanılarak semantik aksiyonların oluşturulması
2. *Lex* kullanılarak ya da C programı yazılarak, giriş diline ait tokenleri oluşturacak sözcüksel analizörün yazılması
3. Bison ayrıştırıcı dosyasındaki `main` fonksiyonunun `yyparse` fonksiyonunu çağıracak şekilde düzenlenmesi
4. Hata raporlama fonksiyonlarının oluşturulması
5. Bison tarafından üretilen kaynak kodun C derleyicisi kullanılarak derlenmesi
6. Çalıştırılabilir *exe* dosyasının üretilmesi için nesne kodunun bağlantılanması (link)

8. Postfix Hesap Makinesi Uygulaması (`pcalc`)

Bu uygulamaya ait gramer tanımlanırken aritmetik operatör önceliklerinin hesaba katılmasına gerek yoktur.

8.1. Bildirimler (Declarations)

Tablo 5 'te listelenen `pcalc` gramer dosyasına ait satırlarda `/*` ve `*/` sembolleri arasına yerleştirilen metinler, açıklamaları (comments) temsil etmektedir.

```
/* Postfix Hesap Makinesi Uygulaması */

%{
   #define YYSTYPE double
   #include <math.h>
   int yylex (void);
   void yyerror (char const *);
%}
%token NUM

%% /* Gramer kuralları ve ilgili semantik aksiyonlar bir sonraki
satırdan itibaren tanımlanır */
```

Tablo 5. `pcalc` 'a ait prolog bölümü ve Bison bildirimleri

`#define` önişlemci direktifi, terminal ve terminal-olmayan sembollere ait semantik değerlerin tipini (bu örnek için `double`) belirleyen `YYSTYPE` makrosunu tanımlamaktadır. `#include <math.h>` satırı, `pow` üstel fonksiyonunun bildirimini yapmak amacıyla, ilgili başlık dosyasını koda dahil etmek için kullanılmaktadır. `int yylex (void)` ve `void yyerror (char const *)`, sırasıyla sözcüksel analizör ve raporlama fonksiyonları için ileri bildirimleri (forward declarations) gerçekleştirir. `%token NUM` ifadesi, gramer kurallarında kullanılan ve birden fazla karakter içeren tek token tipinin (terminal sembol) `NUM` olduğunu işaret etmektedir.

8.2. Gramer Kuralları

`pcalc` gramer kuralları, `input`, `line` ve `exp` olmak üzere 3 adet terminal-olmayan sembol içermektedir. Semantik değerleri hesaplamada kullanılan `$$` işareti, terminal-olmayan sembole ait semantik değeri simgelemektedir.

```
input: /* empty */
     | input line
;

line: '\n'
     | exp '\n' { printf ("\t%.10g\n", $1); }
;

exp: NUM { $$ = $1; }
     | exp exp '+' { $$ = $1 + $2; }
     | exp exp '-' { $$ = $1 - $2; }
     | exp exp '*' { $$ = $1 * $2; }
     | exp exp '/' { $$ = $1 / $2; }
     /* Kuvvet fonksiyonu */
     | exp exp '^' { $$ = pow ($1, $2); }
     /* Tek operandlı çıkarma işlemi */
     | exp 'n' { $$ = -$1; }
;
%%
```

Tablo 6. `pcalc` gramer kuralları

`input` sembolü, `pcalc` girdisinin boş bir karakter dizisinden (string) oluşabileceği gibi ardarda listelenmiş satırlardan da oluşabileceğini ifade etmektedir. `line` sembolü, her bir giriş satırı için muhtemel formatları listelemektedir. Buna göre, her bir satır yalnızca *yeni-satır* (newline) karakterini içerebileceği gibi (böyle bir satır boş bir satırı ifade eder), aritmetik bir ifade ve bunun peşinden gelen yeni-satır karakterini birlikte de içerebilir. `line` sembolüne ait semantik değer, ilgili semantik aksiyon `$$` 'a herhangi bir değer atamadığından başlatılmadan kalır. Normal koşullarda sorun teşkil edecek olan bu durum, `line` 'ın semantik değeri başka bir gramer kuralında kullanılmadığı için problem oluşturmaz. `exp` sembolü, farklı aritmetik ifade türlerinin her biri için ayrı gramer kurallarına sahiptir. `exp` tarafından temsil edilen aritmetik ifade türleri, yalnızca sayırlardan oluşan aritmetik ifadeler, toplama, çıkarma, çarpma, bölme, kuvvet fonksiyonu ve tek operandlı çıkarma işlemidir.

8.3. `main` ve `yyerror` Fonksiyonları

Tablo 7 ‘de görüldüğü gibi, pcalc ‘a ait `main` fonksiyonunun görevi `yyparse()` fonksiyonunu çağırmaktır:

```
int
main (void)
{
      return yyparse ();
}
```

Tablo 7. `main()` fonksiyonu

`yyerror` fonksiyonu, `yyparse` tarafından, herhangi bir sözdizim hatası (syntax error) algılanması durumunda çağrılır. Ekrana yazdırılacak hata mesajının ne olacağı `yyparse` tarafından belirlenir:

```
#include <stdio.h>

void
yyerror (char const *s)
{
      fprintf (stderr, "%s\n", s);
}
```

Tablo 8. `yyerror()` fonksiyonu

8.4. `yylex` Fonksiyonu (Sözcüksel Analizör - Lexical Analyzer)

Sözcüksel analizörün görevi, karakter dizilerini token’lere dönüştürmektir. Ayrıştırıcı program, sözcüksel analizörün ürettiği token’leri kullanarak gramer kurallarının uygulanabilirliğini denetler.

`pcalc` ‘a ait sözcüksel analizör, Tablo 9 ‘da gösterildiği gibi `yylex` fonksiyonu içerisinde gerçeklenmelidir. `yylex` fonksiyonu, giriş dizisindeki boşluk ve tab karakterlerini gözardı eder. Giriş dizisindeki her bir sayıyı, `double` olarak değerlendirir ve her bir sayı için `NUM` token ‘ini geri döndürür. Her bir sayıya ait semantik değer, `yylval` global değişkeninde saklanır. Sayılar haricindeki karakterler ayrı bir token olarak değerlendirilir ve okunan karakterin kendisi geriye döndürülür.

```
#include <ctype.h>

int
yylex (void)
{
      int c;
      /* Boşluk ve tab karakterlerini atla */
      while ((c = getchar ()) == ' ' || c == '\t');
      /* Sayıları işle */
      if (c == '.' || isdigit (c))
      {
            ungetc (c, stdin);
            scanf ("%lf", &yylval);
            return NUM;
      }
      /* Girdi sonuna gelindiğini belirt */
      if (c == EOF)
            return 0;
      /* Sayısal olmayan karakter için sayısal kod döndür */
      return c;
}
```

Tablo 9. `pcalc` 'a ait `yylex()` fonksiyon tanımı

8.5. Ayrıştırıcı Kodunun Üretilmesi ve Derlenmesi

Bison gramer tanımlama dosyası (`pcalc.y`) oluşturulduktan sonra, Linux komut satırında aşağıdaki komut koşularak ayrıştırıcıya ait C kodu üretilir:

```
$ bison pcalc.y
```

Yukarıdaki komut, `pcalc.tab.c` isimli ayrıştırıcı C kodunu üretir. Ayrıştırıcıya ait C kodunun aşağıdaki komut kullanılarak derlenmesi gerekir:

```
$ gcc –lm –o pcalc pcalc.tab.c
```

`-lm` bağlantılama (linking) opsiyonu, `pow` fonksiyonunun tanımının `math` kütüphanesinde aranması için kullanılır. Yukarıdaki komut koşulduktan sonra, `pcalc` çalıştırılabilir (executable) dosyası üretilir. Aşağıdaki satırlar, örnek girdi satırları üzerinde `pcalc` ayrıştırıcısının çalışmasını göstermektedir:

```
3 7 + 3 4 5 * + - n   (Girdi Satırı 1)
13
5 6 / 4 n +                (Girdi Satırı 2)
-3.166666667
3 4 ^                      (Girdi Satırı 3)
81
```

9. Deney Hazırlığı

1. BNF (Backus Naur Form) notasyonu kullanılarak bir CFG (Context Free Grammar) 'nin nasıl tanımlanacağını öğreniniz.
2. LALR ve GLR ayrıştırma algoritmaları hakkında bilgi edininiz. Bison 'da bu algoritmaların nasıl kullanılacağını öğreniniz.
3. Bison gramer tanımlama dosyasının formatını öğreniniz.

4. Bison gramer tanımlama dosyasında her bir gramer kuralı için belirtilen semantik aksiyonların kullanım amacını ve sözdizim yapısını kavrayınız.
5. Bison kullanılarak bir ayrıştırıcı kaynak dosyası üretebilmek için hangi adımların gerekli olduğunu öğreniniz. Ayrıştırıcı dosya formatını inceleyeniz.
6. gcc (GNU C Compiler) kullanılarak Linux komut satırında bir kaynak dosyasının nasıl derleneceğini öğreniniz.
7. Genel anlamda sözcüksel analizör (Lexical Analyzer) 'ün işlevi ve Bison içerisinde kaynak kod seviyesinde nasıl gerçekleneceğine dair bilgi edininiz.
8. Aritmetik ifadelerde kullanılan *postfix* ve *infix* notasyonlar hakkında bilgi edininiz.
9. Bison gramerinde operatör önceliklerinin nasıl tanımlanabileceğini öğreniniz. `%left`, `%right` ve `%prec` Bison ifadelerinin anlamlarını kavrayınız.

10. Deney Tasarımı ve Uygulaması

1. Öncelikle, masaüstünde bulunan postfix hesap makinesine ait `pcalc.y` gramer tanımlama dosyasını inceleyeniz.
   a. Dosyanın genel formatını inceleyiniz.
   b. Önişlemci direktifleri, ileri bildirimler (forward declarations) ve fonksiyon gövdelerinin dosya içerisindeki konumunu iyice kavrayınız.
   c. `%token` satırının ne amaçla kullanıldığını anlayınız.
2. ~~Masaüstündeki *Cygwin* programını çalıştırınız~~. Bu yıl deney Ubuntu işletim sistemi üstünde gerçekleştirilecektir.("*sudo apt-get install flex bison*" komutu ile gerekli bison ve flex paketleri kurulabilir.)
3. Çalışma dizinini (pcalc.y dosyasının bulunduğu dizin) değiştiriniz.
4. Aşağıdaki komutu kullanarak ayrıştırıcı kaynak dosyasını oluşturunuz:
   ```
   $ bison pcalc.y
   ```
5. Oluşturulan `pcalc.tab.c` ayrıştırıcı kaynak dosyasının yapısını inceleyiniz.
   a. `yyparse()` fonksiyonunu inceleyerek gramer tanımlama dosyasındaki gramer kurallarının kod seviyesinde nasıl gerçeklendiğini anlayınız.
   b. Gramer tanımlama dosyasındaki `Epilog` bölümündeki fonksiyon gövdelerinin `pcalc.tab.c` kaynak kodu içerisine değiştirilmeden kopyalandığını gözlemleyiniz.
6. Aşağıdaki komutları kullanarak `pcalc.tab.c` kaynak kodunu derleyiniz ve çalıştırınız:
   ```
   $ gcc –lm –o pcalc pcalc.tab.c
   $ ./pcalc
   ```
7. Aşağıdaki aritmetik ifadelerin sonuçlarını hesaplayınız:
   a. `4 5 + 6 7 9 +*-` b. `5 1 2 3 + - * 20 / 5 *`
8. *Infix* notasyondaki aritmetik ifadelerin sonucunu hesaplamak için `pcalc.y` gramer dosyasını uygun şekilde değiştirerek (operatör önceliklerini ve içiçe geçmiş (nested) parantezleri de hesaba katarak) `icalc.y` dosyasını oluşturunuz.
   a. Operatör önceliklerini düşük öncelikten yüksek önceliğe doğru `+`, `-`, `*`, `/`, `unary minus` (örnek: -5) ve `^` (üst fonksiyonu) olarak kabul edebilirsiniz.
9. `icalc.y` dosyası için yukarıdaki adımları tekrarlayarak `icalc.exe` çalıştırılabilir dosyasını oluşturunuz ve aşağıdaki aritmetik işlemlerin sonucunu hesaplatınız:
   a. `4 + 4.5 - (34/(8*3+-3))` b. `10 ^ 5 * 8 / 5 + 3`

11. Deney Soruları

1. İnfix hesap makinesinde sözdizimi hatalı olan girdilere rastlanması durumunda bile programın çalışmaya devam etmesi nasıl sağlanır? Bu amaçla ilgili gramer tanımlama dosyasını uygun şekilde değiştiriniz. Bu amaçla Bison'a ait `error` anahtar kelimesinden yararlanabilirsiniz.
2. İnfix notasyon kullanan hesap makinesinde / işlemi gerçekleştirildiğinde sıfıra bölme hatası oluşup oluşmadığını test eden semantik aksiyonu yazınız. Ayrıca semantik aksiyon içerisinde, hatanın oluştuğu girdi satırına ait konum bilgisini (0 değerli paydayı içeren girdi ifadesinin konumu) ekrana yazdırınız.

12. Deney Raporu

Deney raporu için size hazır verilecek şablon grup adına doldurulacaktır. Bu şablonda aşağıdaki sorular dışında da sorular olabilir.

1. LALR ve GLR ayrıştırma algoritmalarını kıyaslayınız.
2. yacc ve Bison ayrıştırıcı üreteçleri arasındaki uyumluluğu araştırınız.
3. Deney sorularını cevaplayınız.
4. Deney raporu için size hazır verilecek şablon grup adına deney süresince doldurulacaktır.

13.Deney Kuralları

1. Geç kalanlar deneye alınmayacaktır.
2. Deney sırasında izin almadan cep telefonu kullanımı yasaktır.
3. Deney sırasında diğer gruplarla etkileşime girmek yasaktır.
4. Uygulama veya raporda kopyaya tolerans gösterilmeyecektir.
5. Diğer grupları rahatsız etmemek adına düşük sesle grup içi tartışılabilir.
6. Deney bittikten sonra deney masaları derli toplu/düzenli bırakılmalıdır.
7. Rapor, resmi izin gibi haller dışında telafi yapılmayacaktır.

14.Deneye Gelmeden Araştırılması İstenilenler:

1. Regular Expressions nedir, ne işe yarar?
2. Parser nedir, ne işe yarar?
3. Bison nedir, ne işe yarar, nasıl çalışır?
4. Temel Linux komutları

15.Deneyde Yapılacaklar:

1. Düzenli ifadeler yardımı ile metin içerisinde arama yapma (Notepad++ kullanacağız)
2. Pcalc.y dosyasından faydalanarak infix notasyonunda çalışan hesap makinesi üreten icalc.y dosyası oluşturulması
3. Deney sorularının yapılması
4. Metni kelimelere bölen bir lex fonksiyonu yazılması
5. Deney raporunun doldurulması

16.Kaynaklar

[1] Charles Donnelly, Richard Stallman, "Bison: Tha Yacc-compatible Parser Generator", 2010

[2] http://www.gnu.org/software/bison/manual/html_node/index.html