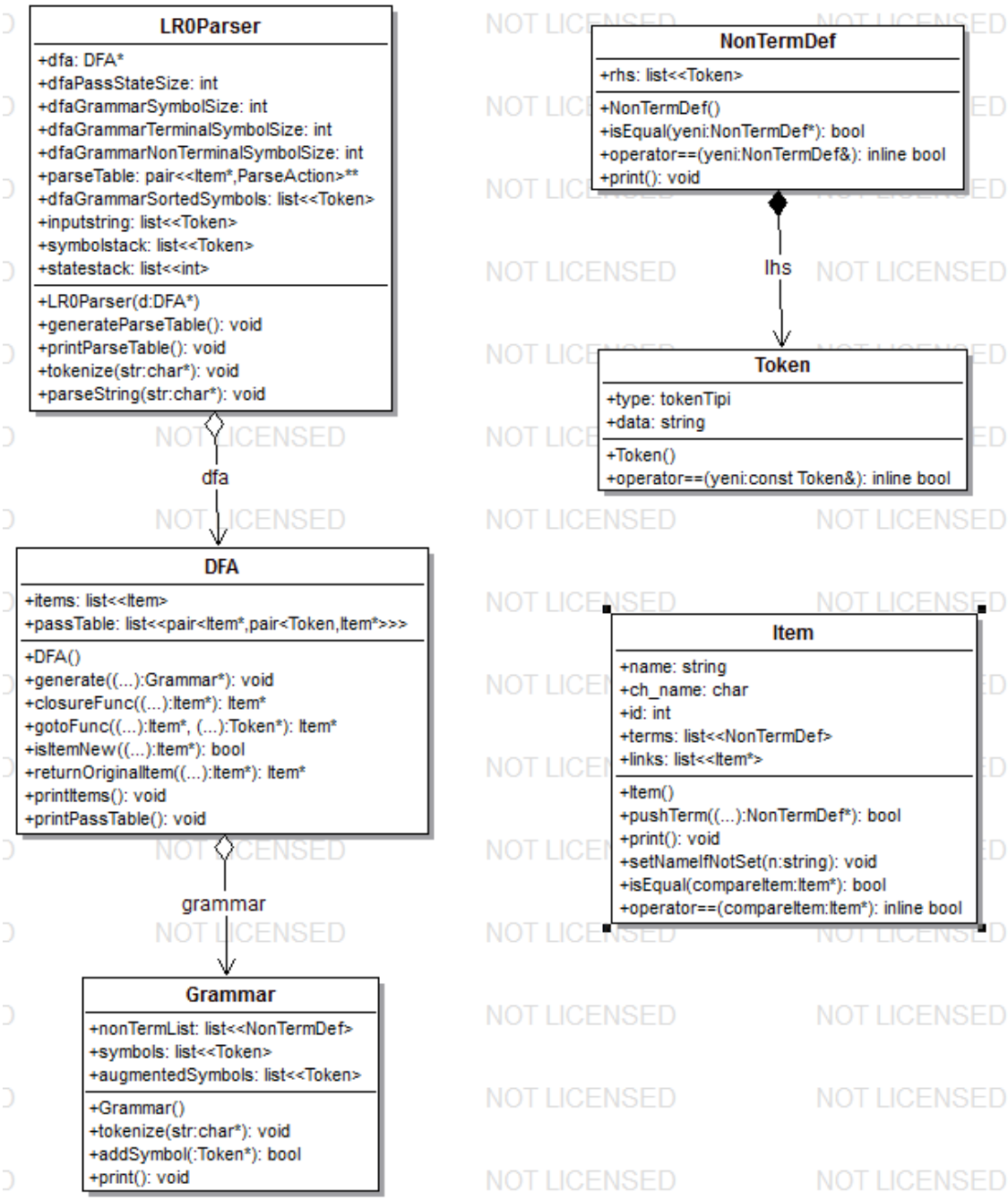
DERLEYİCİ TASARIMI
ÖDEV-2 RAPORU

Nadia Erdoğan

Mustafa Cantürk



(Image generated by unlicensed version of UMLStudio.)

Sınıf tanımları:

Token: tokenize() fonksiyonu sonucunda gelen stringi Token'lara ayrılır. Her token'ın kendisine özel tipi mevcuttur. Genel token tipleri: TERMINAL, NON_TERMINAL, ITERATOR, TERMINATOR. TERMINAL terminal semboller için kullanılır. Non terminal semboller için kullanılır. ITERATOR LR(0) parserda gezinme amaçlı kullanılan "." (nokta) sembolü için kullanılır. TERMINATOR ise sonlanma sembolü "$" ı temsil etmek içindir.

NonTermDef: Tokenlardan oluşan bir gramer ifadesidir. Örnek ifade: “S -> A + E”. NonTermDef sınıfı lhs ve rhs olmak üzere iki ana parçadan oluşur. (lhs tanımı yukarıda UML şemasında çıkmamış.) lhs tanımı tek Token’dan oluşurken, rhs ise token listesi ile temsil edilir. Çünkü solda ifade tek bir Token’dır, sağdaki ifade ise birden fazla Token içerebilir. “S -> A + E” temsili için lhs “S” iken, rhs “A + E” dir.

NonTermDef sınıfı Grammar sınıfının bir elemanı olarak kullanılır. Yani grameri temsil etmek için NonTermDef sınıfı kullanılacaktır. Bunun yanında LR(0) parser için noktalı gezinme terimlerini de NonTermDef sınıfıyla temsil ediyorum. Yani “S -> . A + E” veya “S -> A . + E” gibi ifadelerde NonTermDef sınıfıyla temsil edilmektedir. NonTermDef sınıfını bu açıdan Item’ların elemanlarını temsil etmek için de kullandım.

Grammar: Dosyadan okunan grameri temsil etmek için kullanılır. Alt elemanları NonTermDef sınıfının elemanlarından oluşur.

Item: Closure ve Goto fonksiyonlarının sonucu olarak dönen item’lar bu sınıfta tutulur. Alt elemanları NonTermDef sınıfıyla temsil edilir.

DFA: DFA’nın oluşturulması için gerekli veri yapılarını ve fonksiyonların tanımlandığı sınıftır. İçinde Item’ların listesini tutar. Bu veri yapısı, Item elemanlarının listesi olarak tanımlanır. Bunun yanında geçiş tablosu (passtable) veri yapısı da bu sınıfta tanımlanır. Geçiş tablosu liste ile gerçeklenmiştir. Tablonun bir elemanı pair<Item,pair<Token, Item>> ile temsil edilmektedir. Veri yapısının içerik temsili ise pair<bulunulan_item,pair<geçilen token, geçilen Item>> şeklindedir. Daha basit bir ikili gösterimle [(Mevcut Item), [(Geçilen Token),(Geçilen Item)]] şeklindedir. DFA’nın oluşturulması için, içinde Grammar sınıfını kullanır.

DFA sınıfı, düzenli otomat oluşturulurken kullanılan closure ve goto fonksiyonlarını da barındırır. Bu fonksiyonlar aslında her çalıştığında yeni Item’lar oluştururlar. Böyle bir algoritma sonsuza kadar Item oluşturan bir yapı içerir. Item daha önce oluşturulmuş ise ona bağlamak yerine yenisi oluşturur ve ilgili işaretçi, yeni oluşturulan düğümü işaret eder. Bu yanlış mantığı aşmak için yardımcı fonksiyonlar da bu sınıf içerisinde yazılmıştır. BU yardımcı fonksiyonlar isItemNew() ve returnOriginalItem() isimli fonksiyonlardır. isItemNew() yeni oluştulan Item’ın daha önce oluşturulup oluştulmadığına bakar. Eğer daha önce oluşturulmuşsa, returnOriginalItem() fonksiyonu devreye girer. returnOriginalItem() fonksiyonu ise daha önce oluşturulmuş olan Item’ı bulur ve onun adresini döner. Böylece Item’lar arasında tekillik sağlanmış olur.

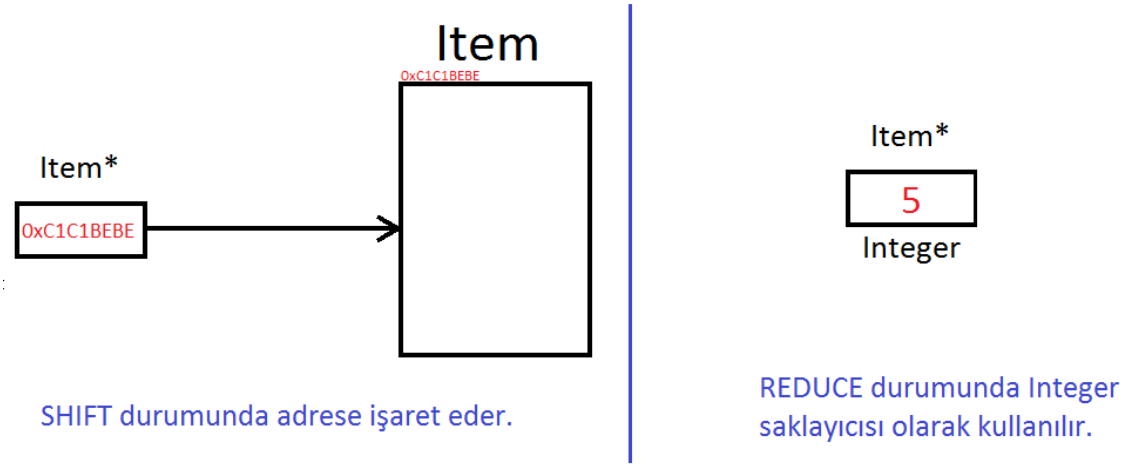
LR0Parser: Bu sınıf yazılımdaki en üst ve ana sınıftır. İçinde DFA sınıfını kullanır. Temeldeki işlevi parse tablosunu yaratmak ve kendisine verilen string’i tokenize edip uyumlu olup olmadığını kontrol etmektir.

Parse tablosu “MxN”’lik bir matris olarak temsil edilmiştir. M, artırılmış gramerdeki (augmented gramer, yani [S’] (S üssü) ‘nün dahil olduğu gramer.) elemanların sayısını temsil eder. N ise DFA’da oluşturulan Item’lerin sayısını temsil eder. Parse tablosunda terminal semboller ile non-terminal sembollerin eylemleri farklıdır. Terminal semboller SHIFT ve REDUCE eylemine sahip iken, nonterminal semboller ise GOTO eylemine sahiptir. Yani sadece başka bir duruma geçiş yapabilir. “$” sembolü ise sadece REDUCE ve ACCEPT eylemlerini kabul eden özel terminal sembol olarak temsil edilir. Parse tablosunda M ile temsil edilen, ilk elemanlar gramerdeki terminal sembollerden oluşur. Terminal sembollerden sonra “$” sembolü temsil edilir. Ondan sonra ise non terminal semboller gelir. Yatayda bakdığımızda tablonın temsili bu şekildedir. Dikeyden bakılınca tablonun her bir satırı bir Item’i temsil eder.

Parse tablosunun her bir elemanı pair<Item*, ParseAction> ikilisi temsil eder. ParseAction SHIFT, REDUCE, PARSE_ACCEPT ve PARSE_UNINITIALIZED durumlarında olabilir. Tablonun bir elemanına hiç atama yapılmamışsa eylemi PARSE_UNINITALIZED ile temsil edilir. Eğer ilgili hücrede indirgeme işlemi gerçekleştirilecekse REDUCE, kaydırma işlemi gerçekleştirilecekse SHIFT eylemi tanımlanır. Kabul durumu ise PARSE_ACCEPT ile temsil edilir.

Parse tablosunun non-terminallere ait tarafı herhangi bir eylem gerçekleştirmemektedir. Bu nedenle ilgili tarafın ParseAction'ı her durumda PARSE_UNINITALIZED olarak tanımlanmıştır.

Item* özelliği SHIFT ve REDUCE eylemine göre farklı davranış sergiler. SHIFT eyleminde geçilecek durumu temsil eder. REDUCE eyleminde ise burada tutulması gereken bilgi aslında Item durum bilgisi değildir. Burada gramerde bulunan kaçıncı ifadeye denk geldiğinin bilgisi tutulmaktadır. REDUCE eylemi bu bilgiyi ileride mevcut bir ifadeyi daha üst bir non terminal ifadeye dönüştürmek için kullanacaktır. Mesela "S -> A + E" ifadesi için; elimizde "A + E" olacaktır ve bu ifade "S" ifadesine indirgenecektir. Parse tablosunda ise "S -> A + E" nin gramerdeki ifadelerden kaçıncısı olduğunun bilgisi tutulmaktadır. Fakat bu durum "Item*" tip bilgisi ile örtüşmemektedir. Tutulmak istenen değerin tipi integer tipindedir. Buradaki sorunu çözmek için REDUCE durumunda Item pointer'ını saklayıcı olarak kullanmaya karar verdim. Item pointer'ı REDUCE durumunda bir bellek adresi yerine gramerde kaçıncı ifadeye denk geldiğinin sayısal değerini tutacaktı. Çünkü pointer tipi de aslında bir saklayıcıydı ve boyutları da aynı integer'ın olduğu gibi 4 bayt idi. Böyle olmasa bile tutulacak sayı miktarı gramerdeki eleman sayısıyla kısıtlıydı.



Bu mantığı kodda ise bu şekilde gerçekledim:

```
//use pointer allocation to store data :)
tmpItemPtr = (Item*)i;
tmpParseTableCell = make_pair(tmpItemPtr, tmpParseAct);
parseTable[ititem->id][j] = tmpParseTableCell;
```

Fakat bu yarattığım özel durum derleme yaparken hata almama sebep oldu. Integer'dan pointer'a dönerken warning verdi. Fakat pointer'dan integer'a dönüşümler için error hataları aldım. Bu sorunu aşmak için g++'ya-fpermissive parametresini vermemi istedi.

(Burada kırmızıyla işaretlediğim 2 farklı nokta daha var. Birincisi g++'ın sürümünün 4.8 olduğu. İkincisi ise derleme parametrelerinde –std=c++11 olması. Bunlara da ileride değineceğim.)

```
[root@localhost compiler]# ls -l
total 40
-rw-r--r-- 1 root root 34209 Dec  4 11:03 lr0parser.cpp
-rw-r--r-- 1 root root    68 Dec  4 11:03 testgrammar.txt
[root@localhost compiler]# g++ --version
g++ (GCC) 4.8.2 20140120 (Red Hat 4.8.2-16)
Copyright (C) 2013 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

[root@localhost compiler]# g++ -std=c++11 lr0parser.cpp -o lr0parser
lr0parser.cpp: In member function 'void LR0Parser::generateParseTable()':
lr0parser.cpp:759:37: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
                tmpItemPtr = (Item*)i;
                                    ^
lr0parser.cpp:870:37: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
                tmpItemPtr = (Item*)tmpInteger;
                                    ^
lr0parser.cpp: In member function 'void LR0Parser::printParseTable()':
lr0parser.cpp:929:52: error: cast from 'Item*' to 'int' loses precision [-fpermissive]
                reduceVal = (int) parseTable[i][j].first;
                                                   ^
lr0parser.cpp: In member function 'void LR0Parser::parseString(char*)':
lr0parser.cpp:1136:75: error: cast from 'Item*' to 'int' loses precision [-fpermissive]
                        tmp_item = (int)parseTable[statestack.front()][j].first;
                                                                           ^
[root@localhost compiler]#
```

g++'a istenildiği gibi –fpermissive parametresini verdiğimde az önceki error'lar warning'e döndü. Programım derlendi ve çalıştırılabilir dosyası oluştu.

```
[root@localhost compiler]# ls -l
total 40
-rw-r--r-- 1 root root 34209 Dec  4 11:03 lr0parser.cpp
-rw-r--r-- 1 root root    68 Dec  4 11:03 testgrammar.txt
[root@localhost compiler]# g++ -std=c++11 -fpermissive lr0parser.cpp -o lr0parser
lr0parser.cpp: In member function 'void LR0Parser::generateParseTable()':
lr0parser.cpp:759:37: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
                tmpItemPtr = (Item*)i;
                                    ^
lr0parser.cpp:870:37: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
                tmpItemPtr = (Item*)tmpInteger;
                                    ^
lr0parser.cpp: In member function 'void LR0Parser::printParseTable()':
lr0parser.cpp:929:52: warning: cast from 'Item*' to 'int' loses precision [-fpermissive]
                reduceVal = (int) parseTable[i][j].first;
                                                   ^
lr0parser.cpp: In member function 'void LR0Parser::parseString(char*)':
lr0parser.cpp:1136:75: warning: cast from 'Item*' to 'int' loses precision [-fpermissive]
                        tmp_item = (int)parseTable[statestack.front()][j].first;
                                                                           ^
[root@localhost compiler]# ls -l
total 176
-rwxr-xr-x 1 root root 138537 Dec  4 20:45 lr0parser
-rw-r--r-- 1 root root  34209 Dec  4 11:03 lr0parser.cpp
-rw-r--r-- 1 root root     68 Dec  4 11:03 testgrammar.txt
[root@localhost compiler]#
```

Parse tablosunu yaratırken karşılaştığım sorunları, bunları çözmeye yönelik yaptığım hareketleri anlattım.

ÖTELE-İNGİRGE ÇELİŞKİSİNİN ÇÖZÜLMESİ

Bunun yakında parse tablosuyla alakalı SHIFT-REDUCE çelişkisi durumunu yaşadım. “+” ve “:=” tokenları için program düzgün çalışmıyordu. Çünkü shift yapması gerektiği yerde reduce yapıyordu.

Burada ie parse tablsonun yaratılma aşamasına müdahale ettim. İlk yazdığım kod parse tablosunu sırasıyla,

1. SHIFT ve non-terminal durumlarını bul ve tabloya yaz.
2. REDUCE durumlarını bul ve tabloya yaz.
3. Sonlanma karakteri için (“$”) ACCEPT ve REDUCE durumlarını tabloya yaz.

şeklinde oluşturuyordu.

1 ve 2’nin yapılma sırasını değiştirdim.

Tablonun yeni yaratılma aşaması şu şekilde oldu:

1. REDUCE durumlarını bul ve tabloya yaz.
2. SHIFT ve non-terminal durumlarını bul ve tabloya yaz.
3. Sonlanma karakteri için (“$”) ACCEPT ve REDUCE durumlarını tabloya yaz.

Bu sayede en son SHIFT durumları yazıldığı için tablodaki REDUCE durumlarının üzerine basmış oluyor ve tabloda en son SHIFT operasyonu kalmış oluyordu. Bu şekliyle çeşitli string girdileri denedim, program çalıştı. ÖTELE-İNDİRGE çelişkisi çözülmüş oldu.

LR0Parser sınıfına geri dönersek;

Bu sınıf parseString() fonksiyonunu içerisinde barındırır. Bu fonksiyon kendisine gelen metni tokenize eder ve ana LR(0) parse operasyonunu yürütmeye başlar. Eğer girilen string verilen gramer ile uyuşuyorsa ekrana “INPUT ACCEPTED!” yazar. Eğer uyuşmuyorsa “INPUT REJECTED!” yazar.

DERLEME

Programın derlenmesi için g++’nın 4.8 sürümüne ihtiyaç var. 1200 satır kadar kod yazmışım. Neredeyse kullandığım bütün veri yapılarını STL listesiyle gerçekledim. Fakat listenin istenilen elemanına dizilerdeki gibi bir adımda erişemiyorsunuz. İstediğim elemanı bulmak için bol bol for ile eleman taraması gerekti. Listeyi ise iterator ile tarıyoruz. Bu kullanımda for döngüsünün görüntüsü ve içindeki kullanımı zaman zaman karmaşıklaşabiliyor. Özellikle içiçe for’larda. C++ dili için bunun yolu bu. Bunun yerine kullanımı daha kolay ve sentaksı daha basit olan “for (Token it : tokenlist)” gibi bir for ifadesi kullandım. Program kullandığım IDE tarafından derlenirken, linux’a attığımda derlenmiyordu. Ana sebebini araştırdığımda ise kullandığım basit ve kolay diye nitelediğim for ifadesinin C++11 standardıyla geliyor olduğu öğrendim. Fakat çok fazla kullanım yaptığım için ve yazdığım kod 1200 satır civarında olduğu için eski sürüm için yeniden düzenlemek çok zordu.

C++11 standardı ise g++’nın 4.8 sürümüyle beraber desteklenen bir özellikmiş.

Bu nedenlerle;

Sürümü **4.8** olan bir **g++** derleyicisi için derleme parametreleri şu şekildedir:

```
g++ -std=c++11 -fpermissive lr0parser.cpp -o lr0parser
```

Ben testlerimi Centos 7 linux dağıtımıyla gerçekleştirdim. Windows'ta ise MinGW'nin son sürümünü kullanarak çalıştım.

PROGRAM ÇIKTILARI

Program girdi çıktı olarak sadece bulunduğu klasörde bulunan testgrammar.txt dosyasından okuma yapar. Programı test etmek için girilen doğru ve yanlış stringler hardcoded şekilde programın içinde tanımlanmıştır. Denen doğru string ve yanlış string aşağıda özel olarak belirtilmiştir.

Program çıktıları itibariyle;

1. Dosyadan okuduğu grameri ekrana basar.
2. DFA'yı oluşturur ve bunun Item'larını ekrana basar.
3. Geçiş tablosunu (passtable) oluşturur ve bunu ekrana basar.
4. Parse tablosunu oluşturur ve bunu ekrana basar.
5. Biri doğru, diğeri yanlış olmak üzere 2 tane girdi dener, bunların sonuçlarını ekrana basar.

```
// denenen doğru string
str[80] = "id := id ; id + id ; id := id $";
// denenen yanlış string (virgül hatası yapmış :))
str2[80] = "id := id id + id ; id := id $";
```

```
[root@localhost compiler]# cat testgrammar.txt
S' -> S $
S -> S ; A
S -> A
A -> E
A -> id := E
E -> E + id
E -> id
[root@localhost compiler]# ./lr0parser
S' -> S $
S -> S ; A
S -> A
A -> E
A -> id := E
E -> E + id
E -> id

GENERATING ITEMS:
0th ITEM -->
S' -> . S $
S -> . S ; A
S -> . A
A -> . E
A -> . id := E
E -> . E + id
E -> . id

1th ITEM -->
S' -> S . $
S -> S . ; A
```

```
2th ITEM -->
S -> A .

3th ITEM -->
A -> E .
E -> E . + id

4th ITEM -->
A -> id . := E
E -> id .

5th ITEM -->
S -> S ; . A
A -> . E
A -> . id := E
E -> . E + id
E -> . id

6th ITEM -->
E -> E + . id

7th ITEM -->
A -> id := . E
E -> . E + id
E -> . id

8th ITEM -->
S -> S ; A .

9th ITEM -->
E -> E + id .

10th ITEM -->
A -> id := E .
E -> E . + id

11th ITEM -->
E -> id .

PRINTING PASSTABLE
I0 -> [S] I1
I0 -> [A] I2
I0 -> [E] I3
I0 -> [id] I4
I1 -> [;] I5
I3 -> [+] I6
I4 -> [:=] I7
I5 -> [A] I8
I5 -> [E] I3
I5 -> [id] I4
I6 -> [id] I9
I7 -> [E] I10
I7 -> [id] I11
I10 -> [+] I6


PRINTING PARSE TABLE:
;         id        :=        +         $         S'        S         A         E
-------------------------------------------------------------------------------------
          S I4                                                  I1        I2        I3
S I5                                    ACC
R 2       R 2       R 2       R 2       R 2
R 3       R 3       R 3       S I6      R 3
R 6       R 6       S I7      R 6       R 6
          S I4                                                            I8        I3
          S I9
          S I11                                                                     I10
R 1       R 1       R 1       R 1       R 1
R 5       R 5       R 5       R 5       R 5
```

```
R 4       R 4       R 4       S I6      R 4
R 6       R 6       R 6       R 6       R 6


TESTING INPUT:
PRINTING INPUT: id := id ; id + id ; id := id $
INPUT ACCEPTED!
PRINTING INPUT: id := id id + id ; id := id $
INPUT REJECTED!
```

```
[root@localhost compiler]# ./lr0parser
S' -> S $
S -> S ; A
S -> A
A -> E
A -> id := E
E -> E + id
E -> id

GENERATING ITEMS:
0th ITEM -->
S' -> . S $
S -> . S ; A
S -> . A
A -> . E
A -> . id := E
E -> . E + id
E -> . id

1th ITEM -->
S' -> S . $
S -> S . ; A

2th ITEM -->
S -> A .

3th ITEM -->
A -> E .
E -> E . + id

4th ITEM -->
A -> id . := E
E -> id .

5th ITEM -->
S -> S ; . A
A -> . E
A -> . id := E
E -> . E + id
E -> . id

6th ITEM -->
E -> E + . id

7th ITEM -->
A -> id := . E
E -> . E + id
E -> . id

8th ITEM -->
S -> S ; A .

9th ITEM -->
E -> E + id .

10th ITEM -->
A -> id := E .
E -> E . + id

11th ITEM -->
E -> id .
```

```
10th ITEM -->
A -> id := E .
E -> E . + id

11th ITEM -->
E -> id .

PRINTING PASSTABLE
I0 -> [S] I1
I0 -> [A] I2
I0 -> [E] I3
I0 -> [id] I4
I1 -> [;] I5
I3 -> [+] I6
I4 -> [:=] I7
I5 -> [A] I8
I5 -> [E] I3
I5 -> [id] I4
I6 -> [id] I9
I7 -> [E] I10
I7 -> [id] I11
I10 -> [+] I6


PRINTING PARSE TABLE:
;         id        :=        +         $         S'        S         A         E
-------------------------------------------------------------------------------------
          S I4                                                  I1        I2        I3
S I5                                    ACC
R 2       R 2       R 2       R 2       R 2
R 3       R 3       R 3       S I6      R 3
R 6       R 6       S I7      R 6       R 6
          S I4                                                            I8        I3
          S I9
          S I11                                                                     I10
R 1       R 1       R 1       R 1       R 1
R 5       R 5       R 5       R 5       R 5
R 4       R 4       R 4       S I6      R 4
R 6       R 6       R 6       R 6       R 6


TESTING INPUT:
PRINTING INPUT: id := id ; id + id ; id := id $
INPUT ACCEPTED!
PRINTING INPUT: id := id id + id ; id := id $
INPUT REJECTED!
[root@localhost compiler]#
```