**گزارش پروژه کامپایلر – ریحانه سلجوقی**

**توضیح ساختمان داده ها:**

**یک map به نام rules داریم ک در آن key ها سمت چپ قواعد و value ها ، آرایه ای از سمت راست قواعد هستند.**

**۳ مپ به نام های :**

Terminal_idx

nterminal_idx

rules_idx

**داریم ک از آن ها برای داشتن ایندکس های هر کدام به منظور ساخت جدول استفاده میکنیم.**

**یک ست از ترمینال ها داریم .**

**و در آخر یک ماتریس دو بعدی برای ساخت جدول.**

```
static ConcurrentHashMap<String, String[]> rules = new ConcurrentHashMap<>();
static Set<String> terminals = new HashSet<>();
static ArrayList<String> terminal_idx = new ArrayList<>();
static ArrayList<String> nterminal_idx = new ArrayList<>();
static ArrayList<String> rules_idx = new ArrayList<>();
static int[][] p_table;
```

**در ابتدا در تایع مین تعداد قواعد را ورودی میگیریم و مپ rules را به ترتیب میسازیم.**

**ترمینال ها در این پیاده سازی ، حروف کوچک هستند. پس هرجا حروف کوچک را دیدیم به ست ترمینال ها اضافه میکنیم. (در اخر باید اپسیلون را از آن حذف کنیم (علامت اپسیلون در کد ما ~ می باشد))**

**در این پروژه ، باید توابع اصلی first، follow ، ساخت جدول پارس و پارس کردن استرینگ را پیاده سازی کنیم.**

- **نحوه پیاده سازی first:**

A-->BC

فرست A در اینجا می شود فرست B ولی اگر B تهی شود، آنگاه فرست A میشود :

First(C) U {First(B) – epsilon}

علت کم کردن اپسیلون این است که ما در این پیاده سازی زمانی اپسیلون را به فرست های A اضافه میکنیم ک بتوانیم به جای A دقیقا فقط اپسیلون قرار دهیم . ولی در اینجا هنوز مشخص نیست که آیا C هم اپسیلون تولید می کند یانه.

A--> epsilon

در این حالت مستقیما اپسیلون را به فرست های A اضافه میکنیم.

و در آخر اگر واژه ای ترمینال باشد، فرست آن خودش است.

- حالا تابع فرست را میبینیم:

ابتدا چک میکنیم اگر این قاعده ای ک میخواهیم فرست های آن را بدست بیاوریم، اولین واژه ی آن ، حرف کوچک بود یا تهی بود ، همان را در result قرار میدهیم و ریترن میکنیم.

```
private static String[] first(String rhs, ArrayList<String> left_recur) {
    String firstChar = String.valueOf(rhs.charAt(0));
    if (!Character.isUpperCase(rhs.charAt(0)) | rhs.charAt(0) == '~') {
        String[] res = {firstChar};
        return res;
    }
```

در غیر اینصورت با یک ناپایانه مواجه شده ایم ک باید فرست آن را پیدا کنیم تا فرست واژه فعلی بدست بیاید.

در ادامه همان توضیحات اولیه برای فرست پیاده سازی میشود.

دقت شود که در فرست یک مشکل ممکن است بوجود بیاید:

مثلا این گرامر:

(از نوشتن ناپایانه ها برای شلوغ نشدن گرامر صرف نظر کردیم)

A-->B

B-->A|C

C-->A

در لوپ می افتاد.

**چون :**

**برای فرست A باید فرست B بدست بیاید و برای فرست B باید C و A بدست بیاید و برای C باید فرست A بدست بیاید و همینطور در لوپ خواهیم افتاد.**

**روش حل این مشکل بدین صورت است که یک arraylist به نام left recur به تابع فرست پاس میدهیم که شامل ناپایانه هایی که در مسیر پیدا کردن فِرست آن ها را هم بررسی نمودیم .**

**حالا هر بار چک میکنیم که اگر این ناپایانه ای که الان در حال بررسی آن هستیم در این لیست بود ، یعنی چپ گردی دارم داریم و در لوپ می افتیم و در اینجا برنامه به ما می گوید که چپ‌گردی وجود دارد و تمام میشود.**

```
left_recur.add(firstChar);
for (String s : nextRules) {
    if(left_recur.contains(String.valueOf(s.charAt(0)))){
        System.out.println("It has left recursion. We can not parse it");
        System.exit( status: 0);
    }
    left_recur.add(String.valueOf(s.charAt(0)));
    String[] temp = first(s, left_recur);
    firsts.addAll(Arrays.asList(temp));
}
```

❖ **نحوه پیاده سازی follow:**

**اول از همه اینکه علامت $ در فالوهای ناپایانه گرامر شروع کننده می آید .**

**حالا قواعد بدین صورت است که اگر**

A--> c B

**فالوهای B همان فالوهای A خواهند بود .**

**اگر**

A--> B D

**فالوهای B ، فرست D هستند (اگر D تهی نشود) .**

**ولی اگر D تهی شود ، همانطور که واضح است انگار هم حالت اول بوجود می آید و هم حالت دوم پس:**

**فالوهای B می شود:**

{first (D)- epsilon } ∪ follow(A)

- **حالا تابع فالو را میبنیم :**

**ابتدا فالوی ناپایانه S که شروع است را $ قرار میدهیم .**

**حالا در قواعد سمت راست فور میزنیم و هرجا که قاعده فعلی را پیدا کردیم، باید فرست قاعده ی بعد آن را طبق توضیحات قبل پیدا کنیم که فالوی واژه فعلی را تشکیل می دهد.**

**اگر این فرست ها شامل تهی باشند آن را باید ریمو کنیم و حالا فالوهای S را پیدا کنیم و به نتیجه قبلی اضافه کنیم.**

```
private static String[] follow(String r, String prev) {
    Set<String> follows = new HashSet<>();
    String[] res = null;
    if (r.equals("S")) {
        follows.add("$");
    }
    for (String s : rules.keySet()) {
        String[] rhs = rules.get(s);
        for (String rhs_sub : rhs) {
            if (rhs_sub.contains(r)) {
                int idx = rhs_sub.indexOf(r);
                rhs_sub = rhs_sub.substring(idx + 1);

                if (rhs_sub.length() != 0) {
                    res = first(rhs_sub, new ArrayList<>());
                    ArrayList<String> res_temp = new ArrayList<>();
                    for (int i = 0; i < res.length; i++) {
                        res_temp.add(res[i]);
                    }
                    if (res_temp.contains("~")) {
                        ArrayList<String> newAns = new ArrayList<>();
                        res_temp.remove( o: "~");
                        String[] answers = follow(s, r);
                        newAns.addAll(res_temp);
                        if (answers != null) {
                            newAns.addAll(Arrays.asList(answers));
                        }
                        res = newAns.toArray(new String[0]);
                    }
```

**پس واضحا طبق توضیحات داده شده این توابع بصورت بازگشتی پیاده سازی می شوند.**

**برای فالو هم چون ممکن است دچار لوپ شویم باید قبلی را پاس بدهیم و اگر کاراکتر مورد بررسی فعلی با قبلی مساوی بود (یعنی در لوپ افتادیم) کاری انجام نمیدهیم.**

```
else if (!r.equals(s)) {
  if (prev != null) {
      if (!prev.equals(s)) {
          res = follow(s, r);
      }
  } else
      res = follow(s, r);
```

❖ توابع findfirst و findfollow

**هر کدام یک مپ از استرینگ به ست برمیگردانند که در آن برای هر واژه فرست و فالوهایش در ست آمده است.**

**در این توابع، توابع اصلی پیدا کردن فرست و فالو را صدا میکنیم.**

**در findfollow یک کار اضافی تر از findfirst داریم که به این علت است که :**

**اگر حالتی داشته باشیم که**

A-->B

B-->A

**یک لوپ خواهیم داشت که باید این را هندل کنیم.**

**پس روی سمت راست قاعده A فور میزنیم و حرف آخر آنهارو میگیریم رو اگر این حرف آخر ناپایانه بود ، دوباره روی قواعد این ناپایانه (در مثال B ) فور میزنیم و اگر قاعده ای برای B پیدا شد ک حرف آخر آن A بود، برای آن ناپایانه ای که تعداد فالوهای آن صفر است، فالوهای دیگری را قرار میدهیم.**

```
    follows.put(s, follow);
for (int i = 0; i < rhs.length; i++) {
    String sub = rhs[i].substring(rhs[i].length() - 1);
    Character my_char = sub.charAt(0);
    if (Character.isUpperCase(my_char)) {
        for (int k = 0; k < rules.get(sub).length; k++) {
            if (rules.get(sub)[k].substring(rules.get(sub)[k].length() - 1).equals(s)) {
                if (follows.get(sub).size() == 0)
                    follows.put(sub, follows.get(s));
            }
        }
    }
}
```

❖ حالا باید تیبل را بسازیم .

**از مپ هایی ک ایندکس می دهند کمک میگیریم.**

**روی تمام ناپایانه ها فور میزنیم .**

**اگر فرست های یک ناپایانه شامل تهی باشند، پس باید فالوها را نیز در جدول مقدار دهی کنیم. پس فالوهای آن راهم در جواب نهایی اضافه میکنیم.**

**حواسمان هست که خود تهی را از جواب خارج کنیم (در جدول پایانه تهی قرار نمیدهیم).**

```
private static void createTable(ConcurrentHashMap<String, Set<String>> firsts
        , ConcurrentHashMap<String, Set<String>> follows) {
    for (String s : rules.keySet()) {
        String[] rhs = rules.get(s);
        for (String r : rhs) {
            String[] res = first(r, new ArrayList<>());
            ArrayList<String> res_temp = new ArrayList<>(Arrays.asList(res));
            if (res_temp.contains("~")) {
                if (res_temp.size() == 1) {
                    ArrayList<String> firstFollow = new ArrayList<>();
                    Set<String> follows_temp = follows.get(s);
                    firstFollow.addAll(follows_temp);
                    res_temp = firstFollow;
                } else {
                    res_temp.remove( o: "~");
                    res_temp.addAll(follows.get(s));
                }
```

**حالا روی این جواب فور میزنیم و ایندکس هارا در جدول پیدا میکنیم اگر مقدار دهی شده باشد، یعنی اینکه گرامر ll1 نیست چون در یک خانه بیش از دو مقدار قرار گرفته است . ولی در غیر اینصورت آن خانه را مقداردهی میکنیم و شماره قاعده ای ک تولید کننده این پایانه از طریق این ناپایانه است را می نویسیم .(از rules_idx )**

```
for (String result : res_temp) {
    int x_indx = nterminal_idx.indexOf(s);
    int y_indx = terminal_idx.indexOf(result);
    if (p_table[x_indx][y_indx] == -1) {
        p_table[x_indx][y_indx] = rules_idx.indexOf(s + "->" + r);
    } else if (p_table[x_indx][y_indx] == rules_idx.indexOf(s + "->" + r)) {
        continue;
    } else {
        System.out.println("this is not a LL1 grammar babayyyeeeee");
        System.exit( status: 0);
    }
}
```

- **پارس کردن**

**برای پارس کردن ، باید یک استک و یک بافر داشته باشیم که در آن کلمه را در بافر وارد میکنیم و از ابتدای این کلمه شروع به خواندن میکنیم و در استک هم از انتها شروع ب خواندن و پاپ کردن میکنیم.**

ابتدا $ و S را در استک میگذاریم و کلمه مورد بررسی را هم در صف یا همان بافر قرار می دهیم.در انتهای بافر هم$ را میگذاریم .

```
private static void parse(String input) {
    Stack<String> stack = new Stack<>();
    stack.push( item: "$");
    stack.push( item: "S");
    Queue<String> buffer = new LinkedList<>();
    String[] split = input.split( regex: "");
    for (int i = 0; i < input.length(); i++) {
        buffer.add(split[i]);
    }
    buffer.add("$");
```

حالا :

اگر که سایز بافر و استک هر دو یک باشد، یعنی اینکه استرینگ ما پارس شده است یعنی اینکه هر دو به $ رسیده اند.

```
if(stack.size() == 1 && buffer.size() == 1){
    System.out.println("input String can be parsed");
    System.exit( status: 0);
```

اگر در استک یک ناپایانه باشد، باید شماره های تایپ پشته و صف را بدست بیاوریم (در جدول) و اگر در جدول مقدار دهی شده بود، یعنی میتوان این قسمت را خواند پس باید از استک پاپ کنیم و آن قاعده ای که از طریق پارس تیبل بدست امده است را دوباره در استک پوش کنیم . اگر که در جدول مقداردهی نشده بود یعنی این استرینگ قابل پارس کردن توسط این گرامر نیست.

و یا اگر که اصلا واژه های این استرینگ در زبان وجود نداشت، باز هم قابل پارس کردن نیست.

```
else if(nterminal_idx.contains(stack.peek())){
    int x = nterminal_idx.indexOf(stack.peek());
    int y = terminal_idx.indexOf(buffer.peek());
    if(x == -1 || y == -1){
        System.out.println(" input string can not be parsed");
        System.exit( status: 0);
    }
    if(p_table[x][y] != -1){
        String rule = rules_idx.get(p_table[x][y]);
        String[] left_and_right = rule.split( regex: "->");
        String[] splited_rule = left_and_right[1].split( regex: "");
        stack.pop();
        if(!splited_rule[0].equals("~")) {
            for (int i = splited_rule.length - 1; i >= 0; i--) {
                stack.push(splited_rule[i]);
```

**اگر که ابتدای استک و صف هر دو یک مقدار باشند، هردو را پاپ میکنیم( پایانه ها)**

**و اگر هیچ کدام از حالات نباشد باز یعنی اینکه استرینگ قابل پارس کردن نیست.**

```
    }else{
        System.out.println("invalid input string");
        System.exit( status: 0);
    }
}else{
    if(stack.peek().equals(buffer.peek())){
        stack.pop();
        buffer.remove();
    }else {
        System.out.println("invalid input string");
        System.exit( status: 0);
    }
}
```

## ❖ چند نمونه تست کیس و بررسی برای صحت کد:

1. **تست کیس بررسی شده در جزوه:**

```
5
S->TQ
Q->+S|~
T->FW
W->*T|~
F->i|(S)
*** Rules with their corresponding number ***
                0: S->TQ
                1: Q->+S
                2: Q->~
                3: T->FW
                4: W->*T
                5: W->~
                6: F->i
                7: F->(S)
```

**پارس تیبل:**

```
*** Parse table of the given Grammar ***

   (  i  )  *  +  $

Q  -  -  2  -  1  2

S  0  0  -  -  -  -

T  3  3  -  -  -  -

F  7  6  -  -  -  -

W  -  -  5  4  5  5
```

**روند پارس شدن یک رشته ورودی:**

```
Enter your Token input: i+i*(i*i)

buffer: [i, +, i, *, (, i, *, i, ), $]
stack: [$, S]

buffer: [i, +, i, *, (, i, *, i, ), $]
stack: [$, Q, T]

buffer: [i, +, i, *, (, i, *, i, ), $]
stack: [$, Q, W, F]

buffer: [i, +, i, *, (, i, *, i, ), $]
stack: [$, Q, W, i]

buffer: [+, i, *, (, i, *, i, ), $]
stack: [$, Q, W]

buffer: [+, i, *, (, i, *, i, ), $]
stack: [$, Q]

buffer: [+, i, *, (, i, *, i, ), $]
stack: [$, S, +]

buffer: [i, *, (, i, *, i, ), $]
stack: [$, S]

buffer: [i, *, (, i, *, i, ), $]
stack: [$, Q, T]
```

1

```
buffer: [i, *, (, i, *, i, ), $]
stack: [$, Q, W, F]

buffer: [i, *, (, i, *, i, ), $]
stack: [$, Q, W, i]

buffer: [*, (, i, *, i, ), $]
stack: [$, Q, W]

buffer: [*, (, i, *, i, ), $]
stack: [$, Q, T, *]

buffer: [(, i, *, i, ), $]
stack: [$, Q, T]

buffer: [(, i, *, i, ), $]
stack: [$, Q, W, F]

buffer: [(, i, *, i, ), $]
stack: [$, Q, W, ), S, (]

buffer: [i, *, i, ), $]
stack: [$, Q, W, ), S]

buffer: [i, *, i, ), $]
stack: [$, Q, W, ), Q, T]

buffer: [i, *, i, ), $]
stack: [$, Q, W, ), Q, W, F]
```

2

```
buffer: [i, *, i, ), $]
stack: [$, Q, W, ), Q, W, F]

buffer: [i, *, i, ), $]
stack: [$, Q, W, ), Q, W, i]

buffer: [*, i, ), $]
stack: [$, Q, W, ), Q, W]

buffer: [*, i, ), $]
stack: [$, Q, W, ), Q, T, *]

buffer: [i, ), $]
stack: [$, Q, W, ), Q, T]

buffer: [i, ), $]
stack: [$, Q, W, ), Q, W, F]

buffer: [i, ), $]
stack: [$, Q, W, ), Q, W, i]

buffer: [), $]
stack: [$, Q, W, ), Q, W]

buffer: [), $]
stack: [$, Q, W, ), Q]

buffer: [), $]
stack: [$, Q, W, )]
```

3

```
buffer: [), $]
stack: [$, Q, W, )]

buffer: [$]
stack: [$, Q, W]

buffer: [$]
stack: [$, Q]

buffer: [$]
stack: [$]

input String can be parsed
```

4

2. حالا یک تست کیس دیگر که به تعداد زیادی چپگردی دارد:

```
4
A->A|B|AB|c
B->A|BA|AB|C
C->A|*|fA|+BD
D->)
*** Rules with their corresponding number ***
                0: A->A
                1: A->B
                2: A->AB
                3: A->c
                4: B->A
                5: B->BA
                6: B->AB
                7: B->C
                8: C->A
                9: C->*
                10: C->fA
                11: C->+BD
                12: D->)

*** Parse table of the given Grammar ***

It has left recursion. We can not parse it
```

3. حالا یک تست دیگر برای حالتی که گرامر واقعا ll1 نیست( نه به علت چپگردی)

(همان گرامر داخل جزوه)

```
5
S->TQ
Q->+S|~
T->FW
W->*T|~
F->i|(S)|i[S]
*** Rules with their corresponding number ***
                0: S->TQ
                1: Q->+S
                2: Q->~
                3: T->FW
                4: W->*T
                5: W->~
                6: F->i
                7: F->(S)
                8: F->i[S]

*** Parse table of the given Grammar ***

this is not a LL1 grammar babayyyeeeee
```

تمام حالات بررسی شد و مشخص است که کد عملکرد درستی دارد.