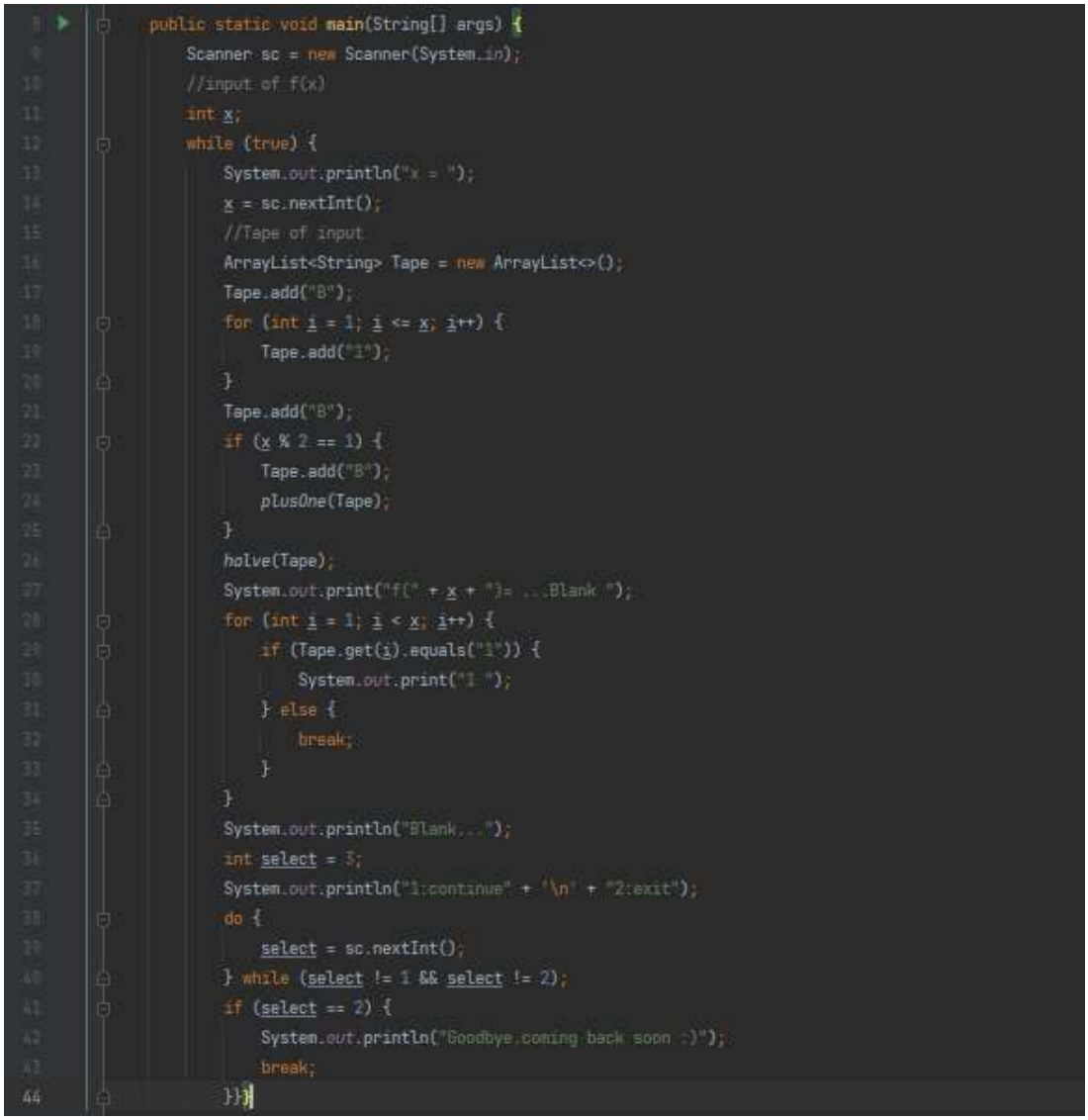
گزارشکار پروژه

نام ونام خانوادگی : اشکان مقرب رازلیقی

شماره دانشجویی : ۹۸۲۳۰۸۱

نام استاد : دکترعامری

سوال ۱ (ii) :



```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    //input of f(x)
    int x;
    while (true) {
        System.out.println("x = ");
        x = sc.nextInt();
        //Tape of input
        ArrayList<String> Tape = new ArrayList<>();
        Tape.add("B");
        for (int i = 1; i <= x; i++) {
            Tape.add("1");
        }
        Tape.add("B");
        if (x % 2 == 1) {
            Tape.add("B");
            plusOne(Tape);
        }
        halve(Tape);
        System.out.print("f(" + x + ")= ...Blank ");
        for (int i = 1; i < x; i++) {
            if (Tape.get(i).equals("1")) {
                System.out.print("1 ");
            } else {
                break;
            }
        }
        System.out.println("Blank...");
        int select = 3;
        System.out.println("1:continue" + '\n' + "2:exit");
        do {
            select = sc.nextInt();
        } while (select != 1 && select != 2);
        if (select == 2) {
            System.out.println("Goodbye.coming back soon :)");
            break;
        }}}
```

توضیحات : در متد main، درابتدا یک آرایه برای نگه داشتن عناصر روی نواردرنظرمی گیریم و سپس ورودی تابع (x) را از کاربر می گیریم.سپس به ابتدای آن کاراکتر B را که نشان دهنده Blank است، اضافه می کنیم . سپس با توجه به اینکه عدد ما چه مقداری دارد به همان تعداد، ۱ به نوار اضافه می کنیم. بعد ازآن ، اگرورودی زوج بود، یک B(به منظور نشان دادن انتهای نوار) و اگر فرد بود ، دوB(B دوم درمتد plusOne به ۱ تبدیل می شود) به انتهای نوار اضافه می کنیم. درانتها با استفاده ازمتد halve، تعداد ۱ ها روی نوار را به نصف کاهش می دهیم و خروجی را چاپ می کنیم.

```java
private static void plusOne(ArrayList<String> Tape) {
    int j = 1;
    String Q = "q0";
    while (true) {
        if (Q.equals("q0") && Tape.get(j).equals("1")) {
            Q = "q0";
            Tape.set(j, "1");
            j++;
        } else if (Q.equals("q0") && Tape.get(j).equals("B")) {
            Q = "q1";
            Tape.set(j, "1");
            j--;
        } else if (Q.equals("q1") && Tape.get(j).equals("1")) {
            Q = "q1";
            Tape.set(j, "1");
            j--;
        } else if (Q.equals("q1") && Tape.get(j).equals("B")) {
            Tape.set(j, "B");
            break;
        }
    }
}
```

توضیحات : درمتد plusOne، به دلیل این که ورودی فرد بوده، قصد داریم آن B اضافه که به انتهای نوار اضافه کردیم را به ۱ تبدیل کنیم .

این تابع مطابق تابع انتقال زیر پیاده سازی شده است :

S(q0,1) = (q0,1,R)

دراین قسمت تا جایی که ۱ وجود دارد به سمت راست می رویم.

S(q0,B) =(q1,1,L)

دراین قسمت به آن B اضافه می رسیم و به ۱ تبدیلش می کنیم و به چپ می رویم.

S(q1,1) = (q1,1,L)

دراین قسمت بعد از ۱ کردن B اضافه، تا جایی که ۱وجود دارد به سمت چپ می رویم.

S(q1,B) =(qf,B,R)

درانتها وقتی به اولین B که در ابتدای نواربود رسیدیم، به حالت نهایی می رویم.

متد halve ازدو بخش تشکیل شده است که در بخش اول، قصد داریم همه ۱ هایی که روی نوار ورودی وجود دارد به X تبدیل کنیم و سپس در بخش دوم هرکدام ازX های نیمه اول نواررا به ۱ تبدیل میکنیم و به ازای هربار۱ کردن ، یک ۱ از انتهای نوار را به B تبدیل می کنیم .

```
80      private static void halve(ArrayList<String> Tape) {
81          int j = 1;
82          String Q = "q2";
83          while (true) {
84              if (Q.equals("q2") && Tape.get(j).equals("1")) {
85                  Q = "q3";
86                  Tape.set(j, "X");
87                  j++;
88              } else if (Q.equals("q3") && Tape.get(j).equals("1")) {
89                  Q = "q3";
90                  Tape.set(j, "X");
91                  j++;
92              } else if (Q.equals("q3") && Tape.get(j).equals("B")) {
93                  Q = "q4";
94                  Tape.set(j, "B");
95                  j--;
96              } else if (Q.equals("q4") && Tape.get(j).equals("X")) {
97                  Q = "q4";
98                  Tape.set(j, "X");
99                  j--;
100             } else if (Q.equals("q4") && Tape.get(j).equals("B")) {
101                 Q = "q5";
102                 Tape.set(j, "B");
103                 j++;
104                 break;
105             }
```

توضیحات بخش اول : این قسمت مطابق تابع انتقال زیرپیاده سازی شده است :

S(q2,1) = (q3,X,R)

دراین قسمت با دیدن اولین ۱ آن را به X تبدیل میکنیم وبه راست می رویم .

S(q3,1) =(q3,X,R)

دراین قسمت تا جایی که ۱ وجود دارد به سمت راست می رویم و۱ها را به X تبدیل می کنیم.

S(q3,B) =(q4,B,L)

دراین قسمت به B که درانتهای نوار است می رسیم و به سمت چپ می رویم.

S(q4,X) =(q4,X,L)

دراین قسمت تا جایی که X وجود دارد به سمت چپ می رویم.

S(q4,B) =(q5,B,R)

درانتها وقتی به اولین B که درابتدای نواربود رسیدیم، به حالت q5 می رویم.

```
107         while (true) {
108             if (Q.equals("q5") && Tape.get(j).equals("X")) {
109                 Q = "q6";
110                 Tape.set(j, "1");
111                 j++;
112             } else if (Q.equals("q6") && Tape.get(j).equals("X")) {
113                 Q = "q6";
114                 Tape.set(j, "X");
115                 j++;
116             } else if (Q.equals("q6") && Tape.get(j).equals("B")) {
117                 Q = "q7";
118                 Tape.set(j, "B");
119                 j--;
120             } else if (Q.equals("q7") && Tape.get(j).equals("X")) {
121                 Q = "q8";
122                 Tape.set(j, "B");
123                 j--;
124             } else if (Q.equals("q8") && Tape.get(j).equals("X")) {
125                 Q = "q8";
126                 Tape.set(j, "X");
127                 j--;
128             } else if (Q.equals("q8") && Tape.get(j).equals("1")) {
129                 Q = "q5";
130                 Tape.set(j, "1");
131                 j++;
132             } else if (Q.equals("q5") && Tape.get(j).equals("B")) {
133                 Tape.set(j, "B");
134                 break;
135             }}}}
```

توضیحات بخش دوم : این قسمت مطابق تابع انتقال زیرپیاده سازی شده است :

S(q5,X) =(q6,1,R)

دراین قسمت با دیدن اولین X آن را به ۱ تبدیل می کنیم و به سمت راست می رویم.

S(q6,X) =(q6,X,R)

دراین قسمت تا جایی که X وجود دارد به سمت راست می رویم.

S(q6,B) =(q7,B,L)

دراین قسمت به B که درانتهای نوار است می رسیم و به سمت چپ می رویم.

S(q7,X) =(q8,B,L)

دراین قسمت به اخرین X می رسیم و آن را به B تبدیل می کنیم.

S(q8,X) =(q8,X,L)

دراین قسمت بعد ازتبدیل آخرین X به B، تا جایی که X وجود دارد به سمت چپ می رویم.

S(q8,1) = (q5,1,R)

دراین قسمت با رسیدن به آخرین۱ به حالت اولیه q5 وسمت راست می رویم.

S(q5,B) =(qf,B,L)

این قسمت شرط خاتمه تابع است و اگر درحالتq5 بودیم وB روی نواربود، بدین معنا است که نیمه اول نوار تماما به ۱ ونیمه دوم نوارتماما به B تبدیل شده است.

چند نمونه Test case :

```
x =
5
f(5)= ...Blank 1 1 1 Blank...
```

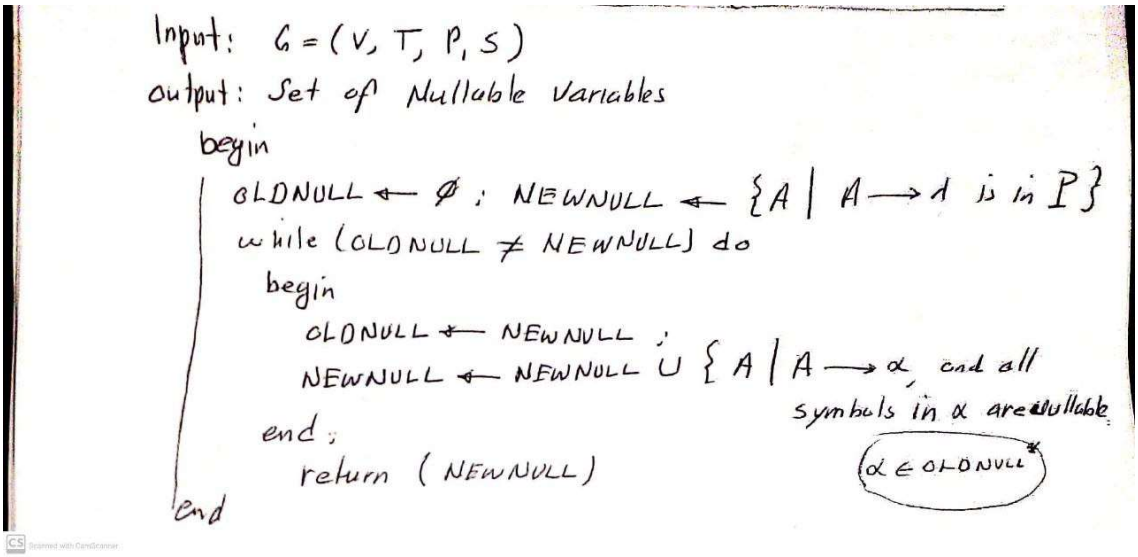
```
x =
10
f(10)= ...Blank 1 1 1 1 1 Blank...
```

سوال ۲ :

findNullVariables Method in contextFree Class

```java
public ArrayList<Character> findNullVariables(ArrayList<String> grammarRules) {
    ArrayList<Character> newNullVariables = new ArrayList<>();
    ArrayList<Character> oldNullVariables = new ArrayList<>();
    for (String st : grammarRules) {
        if (st.charAt(3) == '.') {
            newNullVariables.add(st.charAt(0));
        }
    }
    while(!newNullVariables.equals(oldNullVariables)){
        oldNullVariables.clear();
        oldNullVariables.addAll(newNullVariables);
    for (String st : grammarRules) {
        int flag = 0;
        for (int i = 3; i < st.length(); i++) {
            if (!newNullVariables.contains(st.charAt(i))) {
                flag = 1;
                break;
            }
        }
        if (flag == 0) {
            if(!newNullVariables.contains(st.charAt(0))) {
                newNullVariables.add(st.charAt(0));
            }
        }
    }
}return newNullVariables;}
```

توضیحات : برای اینکه قواعد لامبدا را حذف کنیم، نیاز است که ابتداnullVariables را پیدا کنیم. دراین متد با استفاده ازالگوریتم زیرnullVariables را پیدا می کنیم.



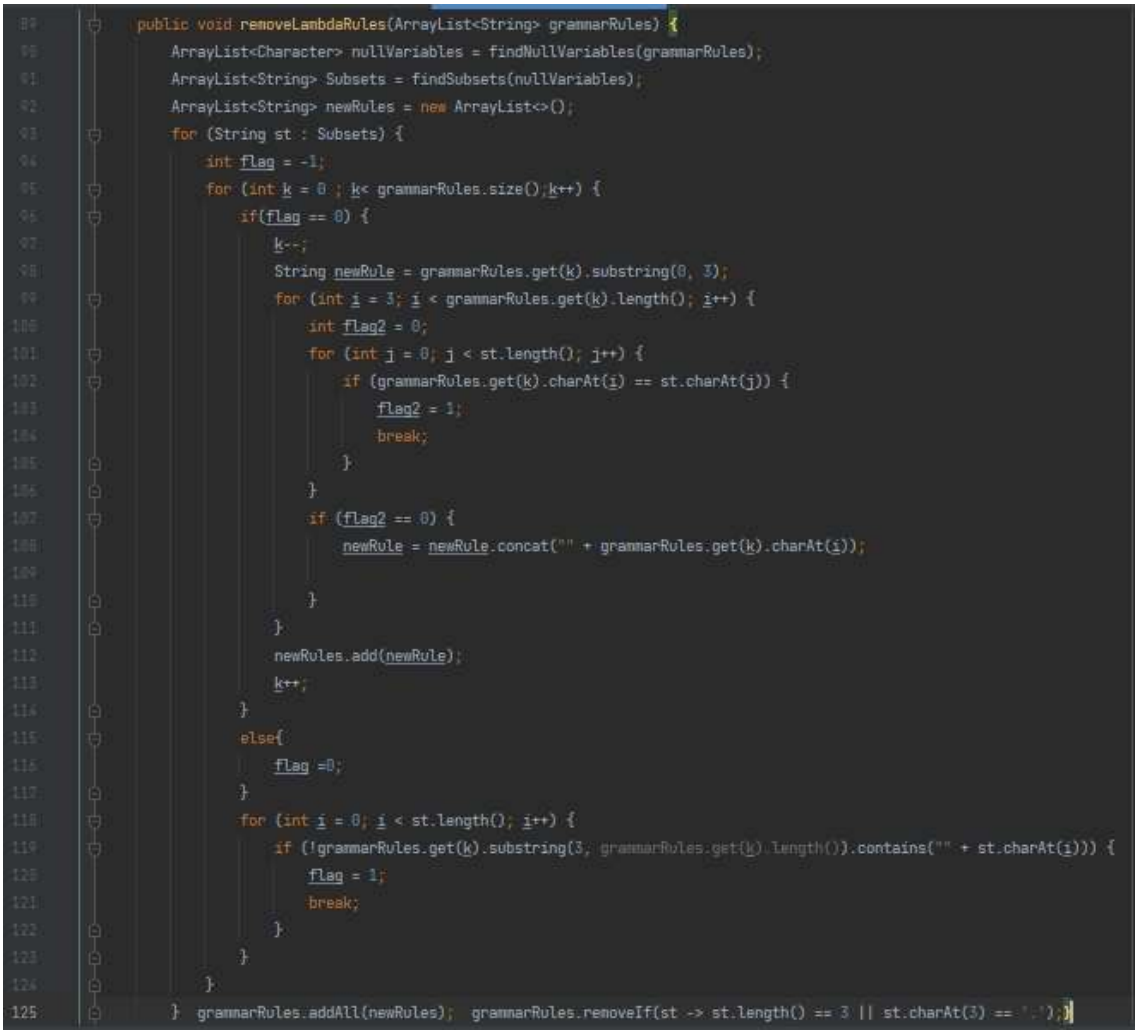
findSubsets Method in contextFree Class

```java
private ArrayList<String> findSubsets(ArrayList<Character> nullVariables) {
    ArrayList<String> Subsets = new ArrayList<>();
    int n = nullVariables.size();
    // Run a loop for finding all 2^n
    // subsets one by one
    for (int i = 1; i < (1 << n); i++) {
        String Subset = "";
        // find current subset
        for (int j = 0; j < n; j++) {

            // (1<<j) is a number with jth bit 1
            // so when we 'and' them with the
            // subset number we get which numbers
            // are present in the subset and which
            // are not
            if ((i & (1 << j)) > 0) {
                Subset = Subset.concat("" + nullVariables.get(j));
            }
        }
        Subsets.add(Subset);
    }
    return Subsets;
}
```

توضیحات : برای اینکه قواعد لامبدا را حذف کنیم نیاز است که پس از پیداکردن nullVariables، به ازای هرزیرمجموعه غیرتهی از آن ها، اعضای آن زیرمجموعه را ازسمت راست هرقاعده ای که شامل همه آن ها می شود، حذف کنیم ویک قاعده جدید تولید کنیم. دراین متد، همه زیرمجموعه های غیرتهی ازnullVariables برگردانده می شود.

removeLambdaRules Method in contextFree Class



```
public void removeLambdaRules(ArrayList<String> grammarRules) {
    ArrayList<Character> nullVariables = findNullVariables(grammarRules);
    ArrayList<String> Subsets = findSubsets(nullVariables);
    ArrayList<String> newRules = new ArrayList<>();
    for (String st : Subsets) {
        int flag = -1;
        for (int k = 0 ; k< grammarRules.size();k++) {
            if(flag == 0) {
                k--;
                String newRule = grammarRules.get(k).substring(0, 3);
                for (int i = 3; i < grammarRules.get(k).length(); i++) {
                    int flag2 = 0;
                    for (int j = 0; j < st.length(); j++) {
                        if (grammarRules.get(k).charAt(i) == st.charAt(j)) {
                            flag2 = 1;
                            break;
                        }
                    }
                    if (flag2 == 0) {
                        newRule = newRule.concat("" + grammarRules.get(k).charAt(i));

                    }
                }
                newRules.add(newRule);
                k++;
            }
            else{
                flag =0;
            }
            for (int i = 0; i < st.length(); i++) {
                if (!grammarRules.get(k).substring(3, grammarRules.get(k).length()).contains("" + st.charAt(i))) {
                    flag = 1;
                    break;
                }
            }
        }
    } grammarRules.addAll(newRules);  grammarRules.removeIf(st -> st.length() == 3 || st.charAt(3) == '.');}
```

توضیحات : دراین قسمت همان طور که در توضیحات قسمت قبل گفته شد، به ازای هرزیرمجموعه غیرتهی ازnullVariables، اعضای آن زیرمجموعه را ازسمت راست هرقاعده ای که شامل همه آن ها می شود، حذف ویک قاعده جدید تولیدمی کنیم.

در انتها هر قاعده ای که به شکل X-> باشد(دلیل به وجودآمدن همچین قاعده ای این است که به عنوان مثال قاعده اصلی به شکل X->AB بوده و A,B عضوی از nullVariables بوده اند و پس از انجام مراحل بالا، A,B حذف شده اند) یا X->.

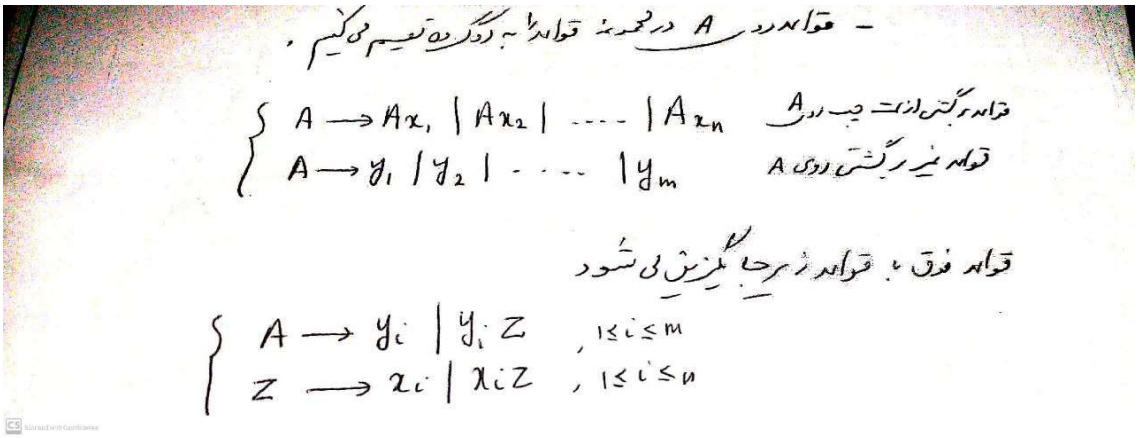
(. نماد لامبدا است) را حذف می کنیم.

removeLeftRecursionRules Method in contextFree Class

```
133 @   public void removeLeftRecursionRules(ArrayList<String> grammarRules) {
134         ArrayList<Character> variables = new ArrayList<>();
135         for (String grammarRule : grammarRules) {
136             if (grammarRule.charAt(0) == grammarRule.charAt(3)) {
137                 if (!variables.contains(grammarRule.charAt(0))) {
138                     variables.add(grammarRule.charAt(0));
139                 }
140             }
```

```
142         ArrayList<Character> variablesWithLeftRecursionRules = new ArrayList<>();
143         ArrayList<String> newRules = new ArrayList<>();
144         for (char variable : variables) {
145             ArrayList<String> leftRecursions = new ArrayList<>();
146             ArrayList<String> nonLeftRecursions = new ArrayList<>();
147             for (String grammarRule : grammarRules) {
148                 if (grammarRule.charAt(0) == variable) {
149                     if (grammarRule.charAt(3) == variable) {
150                         leftRecursions.add(grammarRule);
151                         if (!variablesWithLeftRecursionRules.contains(variable)) {
152                             variablesWithLeftRecursionRules.add(variable);
153                         }
154                     } else {
155                         nonLeftRecursions.add(grammarRule);
156                     }
157                 }
158             }
159             for (String nonLeftRecursion : nonLeftRecursions) {
160                 char newVariable = (char) ('Z' - variablesWithLeftRecursionRules.size() + 1);
161                 newRules.add(nonLeftRecursion + newVariable);
162             }
163             for (String leftRecursion : leftRecursions) {
164                 char newVariable = (char) ('Z' - variablesWithLeftRecursionRules.size() + 1);
165                 String newRule = newVariable + "->";
166                 newRule = newRule.concat(leftRecursion.substring(4, leftRecursion.length()));
167                 newRules.add(newRule);
168                 newRules.add(newRule + newVariable);
169             }
170         }
171         Iterator<String> it = grammarRules.iterator();
172         for (char variable : variablesWithLeftRecursionRules) {
173             while (it.hasNext()) {
174                 String grammarRule = it.next();
175                 if (grammarRule.charAt(0) == variable && grammarRule.charAt(3) == variable) {
176                     it.remove();
177                 }
178             }
179         }
180         grammarRules.addAll(newRules);}
```

توضیحات : دراین متد، درابتدا(خطوط۱۳۵ تا ۱۴۰) متغیرهایی که دارای قاعده بازگشتی ازسمت چپ هستند را شناسایی می کنیم(شرط این که یک قاعده به شکل X->Y... بازگشتی از سمت چپ باشد این است که X=Y باشد). سپس، با استفاده از الگوریتم زیرقواعد غیر بازگشتی از سمت چپ معادل با این قواعد را تولید می کنیم.



درانتها(خطوط ۱۷۱ تا ۱۷۹)، تمام قواعد برگشتی از سمت چپ را ازمجموعه قواعد حذف می کنیم وقواعد جدید تولید شده را به آن اضافه می کنیم.

removeUnitRules Method in contextFree Class

```
public void removeUnitRules(ArrayList<String> grammarRules) {
    ArrayList<String> UnitRules = new ArrayList<>();
    for (String grammarRule : grammarRules) {
        if (Character.isUpperCase(grammarRule.charAt(3)) && grammarRule.length() == 4) {
            UnitRules.add(grammarRule);
        }
    }
    ArrayList<Character> additionalVariables = new ArrayList<>();
    ArrayList<String> newRules = new ArrayList<>();
    for (String UnitRule : UnitRules) {
        if (!additionalVariables.contains(UnitRule.charAt(0))) {
            ArrayList<Character> dependency = new ArrayList<>();
            dependency.add(UnitRule.charAt(0));
            dependency.add(UnitRule.charAt(3));
            char ch = UnitRule.charAt(3);
            for (String unitRule : UnitRules) {
                if (unitRule.charAt(0) == ch) {
                    additionalVariables.add(ch);
                }
            }
            for (String grammarRule : grammarRules) {
                if (grammarRule.charAt(0) == ch && Character.isUpperCase(grammarRule.charAt(3)) && grammarRule.length() == 4) {
                    dependency.add(grammarRule.charAt(3));
                    ch = grammarRule.charAt(3);
                    for (String unitRule : UnitRules) {
                        if (unitRule.charAt(0) == ch) {
                            additionalVariables.add(ch);
                        }
                    }
                }

            }
            for (int i = dependency.size() - 1; i >= 0; i--) {
                for (String grammarRule : grammarRules) {
                    if (grammarRule.charAt(0) == dependency.get(i) && (grammarRule.length() > 4 || Character.isLowerCase(grammarRule.charAt(3)))) {
                        for (int j = i - 1; j >= 0; j--) {
                            newRules.add(dependency.get(j) + "->" + grammarRule.substring(3, grammarRule.length()));}}}
                        grammarRules.removeIf(st -> dependency.contains(st.charAt(0)) && dependency.contains(st.charAt(3)) && st.length() == 4);
        }
    }
    grammarRules.addAll(newRules);
    ArrayList<String> temp = new ArrayList<>();
    for (String grammarRule : grammarRules) {
        if (!temp.contains(grammarRule)) {
            temp.add(grammarRule);
        }
    }
    grammarRules.clear();
    grammarRules.addAll(temp);
}
```

توضیحات : دراین متد، درابتدا(خطوط ۱۸۹ تا ۱۹۴) قواعد واحد را شناسایی می کنیم(شرط این که یک قاعده به شکل X-> ...واحد باشد این است که طول قاعده (با

احتساب – و< هرکدام به عنوان یک کاراکتر) دقیقا ۴ باشد و اولین و تنهاترین حرف در سمت راست قاعده از حروف بزرگ انگلیسی باشد(Variable) باشد.

سپس به ازای هرقاعده واحد، تمام متغیرهایی که به متغیری که درسمت چپ قاعده واحد فعلی وابسته هستند را پیدا می کنیم (خطوط ۱۹۷ تا ۲۱۹). پس از آن، از اخرین متغیرکه در آرایه وابستگی وجود دارد شروع کرده وبه ازای هرمتغیری که قبل از آن دراین آرایه وجود دارد و هرقاعده غیرواحدی که مربوط به آن است ، یک قاعده جدید که سمت چپ آن متغیری که قبل از این متغیردرآرایه وابستگی و سمت راست آن سمت راست قاعده غیرواحد مذکورمی باشد، تولید می کنیم و این کار را برای متغیرهای قبل از آن تا متغیر با index =1 تکرارمی کنیم و درانتها قواعد واحدی را از قواعد گرامرکه سمت چپ و راست آن ها عضوی از آرایه وابستگی باشد، حذف می کنیم(خطوط ۲۲۰ تا ۲۲۷).

درخطوط ۲۲۹ تا ۲۳۷ اگر قواعد گرامر بعد از اضافه کردن قواعد جدید دارای قواعد تکراری باشد، آن قواعد تکراری را از قواعد گرامر حذف می کنیم.

convertToGreibachNormalForm Method in contextFree Class

```
private void convertToGreibachNormalForm(ArrayList<String> grammarRules) {
    ArrayList<String> newRules = new ArrayList<>();
    ArrayList<String> removedRules = new ArrayList<>();
    for (int j = 0; j < grammarRules.size(); j++) {
        if (Character.isLowerCase(grammarRules.get(j).charAt(3))) {
            for (int i = 4; i < grammarRules.get(j).length(); i++) {
                if (Character.isLowerCase(grammarRules.get(j).charAt(i))) {
                    newRules.add(Character.toUpperCase(grammarRules.get(j).charAt(i)) + "->" + grammarRules.get(j).charAt(i));
                    grammarRules.set(j, grammarRules.get(j).substring(0, 4) + Character.toUpperCase(grammarRules.get(j).charAt(4)) +
                            grammarRules.get(j).substring(5, grammarRules.get(j).length()));
                }
            }
        } else {
            removedRules.add(grammarRules.get(j));
            for (String grammarRule : grammarRules) {
                if (grammarRule.charAt(0) == grammarRules.get(j).charAt(3)) {
                    newRules.add(grammarRules.get(j).charAt(0) + "->" + grammarRule.substring(3, grammarRule.length())
                            + grammarRules.get(j).substring(4, grammarRules.get(j).length()));
                }
            }
        }
    }
    ArrayList<String> temp = new ArrayList<>();
    for (String newRule : newRules) {
        if (!temp.contains(newRule)) {
            temp.add(newRule);
        }
    }
    grammarRules.removeIf(removedRules::contains);
    grammarRules.addAll(temp);
}
```

توضیحات : دراین متد، قصد داریم قواعد را به فرم نرمال گریباخ درآوریم (ینی قواعد به یکی از دو شکل X->a یا X->a V1V2.... باشد که a عضو ترمینال ها و Vi ها عضو Variables هستند). برای این کار، درابتدا(خطوط ۲۵۱ تا ۲۵۹) سمت راست

هرقاعده گرامر که با حروف کوچک شروع شده را درنظرگرفتیم واگرحروف بعد ازآن کوچک بودند آن ها را به حروف بزرگ تبدیل کردیم (حروف بزرگ نشان دهنده Variable شدنشان است) و هم چنین یک قاعده جدید به شکل X->x که x آن حرف کوچک است وX حرف بزرگ متناظر با آن است، تولید می کنیم.

اما اگرسمت راست قاعده ای با حروف کوچک شروع نشده باشد، به ازای هرقاعده ای از قواعد گرامر که سمت چپ آن ، حرف اول سمت راست قاعده مذکورباشد، یک قاعده جدید که سمت چپ آن سمت چپ قاعده مذکورو سمت راست آن همان سمت راست قاعده مذکور با این تفاوت که به جای اولین حرف آن ، سمت راست قاعده دومی است، تولید می کنیم(خطوط ۲۶۰ تا ۲۶۹).

درانتها، ازقواعد گرامر قواعد گفته شده درپاراگراف قبل را حذف می کنیم و قواعد جدید غیرتکراری را به آن اضافه می کنیم.

normalize Method in contextFree Class

```
11      public void normalize(ArrayList<String> grammarRules) {
12          removeLambdaRules(grammarRules);
13          removeLeftRecursionRules(grammarRules);
14          removeUnitRules(grammarRules);
15          convertToGreibachNormalForm(grammarRules);
16      }
```

توضیحات : دراین متد، به ترتیب متد های removeLambdaRules و removeLeftRecursionRules و removeUnitRules و convertToGreibachNormalForm راروی قواعد گرامرمستقل از متن داده شده اعمال می کنیم تا به فرم نرمال گریباخ آن برسیم.

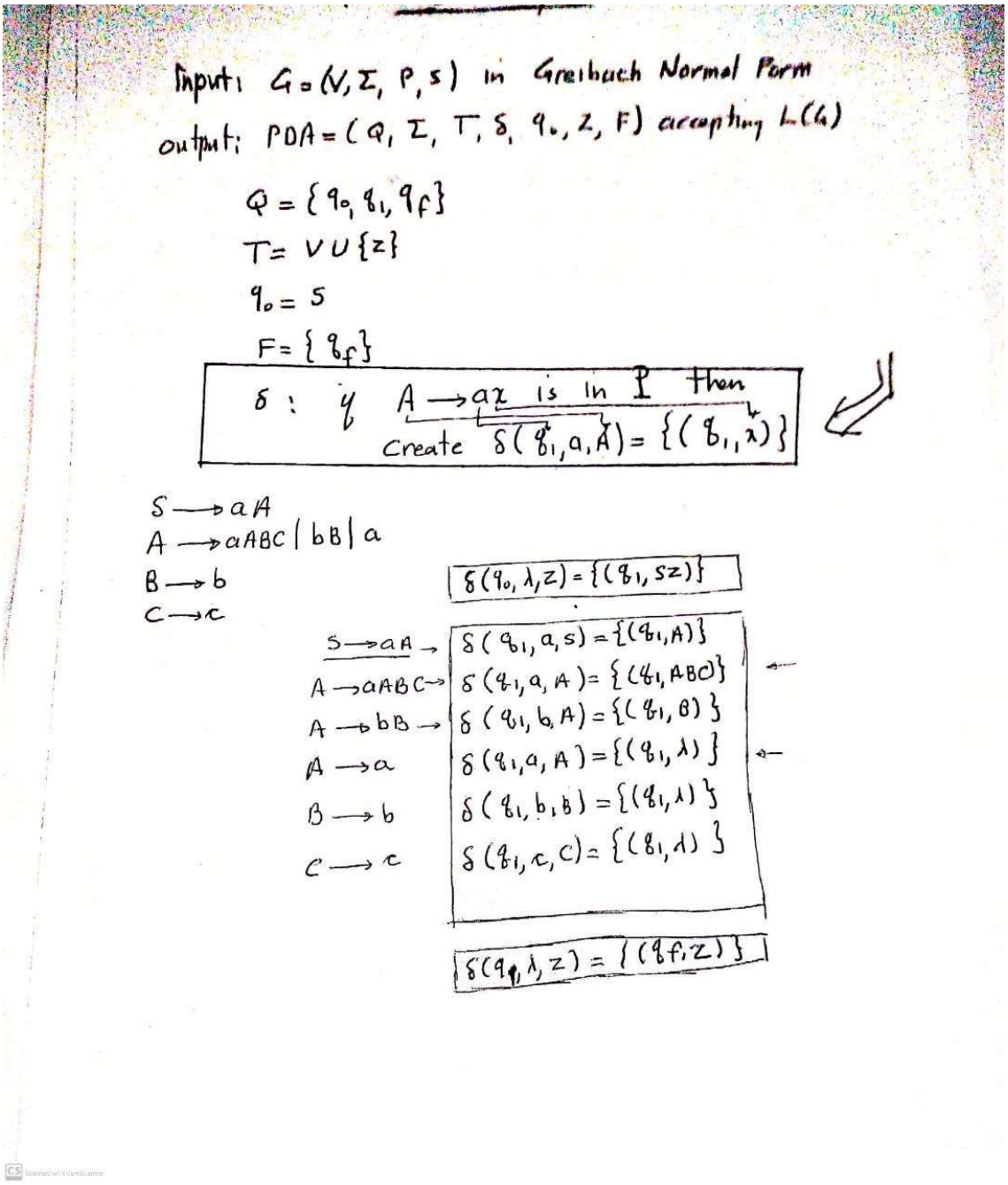
checkMembershipByPDA Method in contextFree Class

```java
public boolean checkMembershipByPDA(ArrayList<String> grammarRules, String word, int index, Stack<Character> stack) {
    ArrayList<String> relatedRules = new ArrayList<>();
    for (String grammarRule : grammarRules) {
        if (grammarRule.charAt(3) == word.charAt(index) && stack.peek() == grammarRule.charAt(0)) {
            relatedRules.add(grammarRule);
        }
    }
    for (String relatedRule : relatedRules) {
        Stack<Character> temp = new Stack<>();
        for (Character variable : stack) {
            temp.push(variable);
        }
        temp.pop();

        for (int i = relatedRule.length() - 1; i >= 4; i--) {
            temp.push(relatedRule.charAt(i));
        }
        if (index == word.length() - 1) {
            if (temp.peek() == '$') {
                return true;
            }
        } else {
            if (checkMembershipByPDA(grammarRules, word,  index: index + 1, temp)) {
                return true;
            }
        }
    }
    return false;
}
```

توضیحات :دراین متد، قصد داریم عضویت یک رشته را با استفاده ازPDA مربوط به گرامر مستقل از متن داده شده بررسی کنیم. شرط پذیرش یک رشته این است که ازبین تمام حالت های ممکن برای آن رشته،حداقل دریک حالت بعد از به پایان رسیدن رشته، در پشته تنها $ مانده باشد.

برای پیاده سازی این PDA ازالگوریتم زیراستفاده شده است:



توجه : گرامر داده شده درانتهای الگوریتم بالا در انتهای گزارشکاربه عنوان ورودی داده می شود و عضویت چندین رشته توسط متد مذکوربررسی می شود.

convertToSimpleForm Method in regular Class

```java
public void convertToSimpleForm(ArrayList<String>grammarRules){
    ArrayList<String>newRules = new ArrayList<>();
    int count = - 1;
    for(int i = 0 ; i <grammarRules.size();i++){
        if(grammarRules.get(i).length()>=5 && Character.isLowerCase(grammarRules.get(i).charAt(4))){
            count++;
            String st = grammarRules.get(i).substring(4,grammarRules.get(i).length());
            char newVariable1 = (char) ('M' +count);
            grammarRules.set(i,grammarRules.get(i).substring(0,4) + newVariable1) ;
            char newVariable2 ='.';
            if(st.length() == 1){
                newRules.add(newVariable1 + "->" +st);
            }
            for(int j = 0 ; j<st.length()-1;j++){
                if(j == st.length()-2){
                    newRules.add(newVariable1 + "->" +st.substring(j,j+2));
                }
                else {
                    count++;
                    newVariable2 = (char) ('M' + count);
                    newRules.add(newVariable1 + "->" + st.charAt(j) + newVariable2);
                    newVariable1 = newVariable2;

                }
            }
        }
    }
    grammarRules.addAll(newRules);
}
```

توضیحات : این متد بعد از حذف قواعد لامبدای گرامر منظم انجام می شود وقصد داریم قواعدی به عنوان مثال A->bbB را به دو قاعده A->bC وC->bB تبدیل کنیم.

change Method in regular Class

```java
public void change(ArrayList<String>grammarRules){
    for(int i = 0 ; i<  grammarRules.size();i++){
        if(grammarRules.get(i).length() == 4 && Character.isLowerCase(grammarRules.get(i).charAt(3))){
            grammarRules.set(i,grammarRules.get(i) + "f");
        }
    }
}
```

توضیحات : در این متد، قصد داریم به سمت راست هرقاعده ای که سمت راستش تنها یک حرف کوچک (ترمینال) دارد و طولش ۴ است، حرف f را که نشان دهنده یکی از حالات نهایی است اضافه کنیم.

checkMembershipByFA Method in regular Class

```java
public boolean checkMembershipByFA(ArrayList<String> grammarRules, ArrayList<Character>finalStates,String word, int index, char Q) {
    ArrayList<String> relatedRules = new ArrayList<>();
    for (String grammarRule : grammarRules) {
        if (grammarRule.charAt(3) == word.charAt(index) && Q == grammarRule.charAt(0)) {
            relatedRules.add(grammarRule);
        }
    }
    for (String relatedRule : relatedRules) {
        char temp = relatedRule.charAt(4);
        if (index == word.length() - 1) {
            if (finalStates.contains(temp)){
                return true;
            }
        } else {
            if (checkMembershipByFA(grammarRules,finalStates, word,  index: index + 1, temp)) {
                return true;
            }
        }
    }
    return false;
}
```

توضیحات :دراین متد، قصد داریم عضویت یک رشته را با استفاده ازFA مربوط به گرامرمنظم داده شده بررسی کنیم. شرط پذیرش یک رشته این است که ازبین تمام حالت های ممکن برای آن رشته،حداقل دریک حالت بعد از به پایان رسیدن رشته، دریکی ازحالات نهایی باشیم.

main Method in P_2_9823081 Class

```java
public static void main(String[] args) {
    while (true) {
        System.out.println("1:check regularity" + '\n' + "2:check being context free" + '\n' + "3:Quit");
        Scanner sc = new Scanner(System.in);
        String select = "";
        do {
            select = sc.next();
        } while (!select.equals("1") && !select.equals("2") && !select.equals("3"));
```

توضیحات : دراین قسمت از این متد، ازکاربر می خواهیم که بین سه گزینه بررسی منظم بودن، بررسی مستقل از متن بودن و یا خروج یکی را انتخاب کند.

```java
        if (select.equals("1")){
            ArrayList<String> grammarRules = new ArrayList<>();
            System.out.println("please enter your rules one by one(with this format : leftSide->rightSide) and enter -1 at the end");
            int flag = 0;
            while (flag != -1) {
                String grammarRule = sc.next();
                if (grammarRule.equals("-1")) {
                    flag = -1;
                } else {
                    if (grammarRule.charAt(1) == '-' && grammarRule.charAt(2) == '>' && Character.isUpperCase(grammarRule.charAt(0))) {
                        int counter = 0;
                        for(int i = 3 ; i <grammarRule.length();i++){
                            if(Character.isUpperCase(grammarRule.charAt(i))){
                                counter++;
                            }
                        }
                        if(counter <=1) {
                            grammarRules.add(grammarRule);
                        }
                        else{
                            System.out.println(" input grammar is not regular");
                        }
                    } else {
                        System.out.println("invalid format or input grammar is not regular");
                        flag = -1;
                    }
                }
            }
            if(grammarRules.isEmpty()){
                continue;
            }
            System.out.println("input grammar is regular");
```

توضیحات : بعد از انتخاب بررسی منظم بودن توسط کاربر، در این قسمت از متد به بررسی منظم بودن گرامری که کاربر وارد می کند می پردازیم.(شرط منظم بودن این است که سمت چپ هرقاعده گرامر داده شده یک حرف بزرگ (Variable) وسمت راست آن شامل فقط یک حرف بزرگ (Variable)باشد.

```
String sel = "";
contextFree contextfree = new contextFree();
regular reg = new regular();
ArrayList<Character> finalStates = new ArrayList<>(contextfree.findNullVariables(grammarRules));
finalStates.add('f');
contextfree.removeLambdaRules(grammarRules);
reg.convertToSimpleForm(grammarRules);
reg.change(grammarRules);
contextfree.removeUnitRules(grammarRules);
while (true) {
    System.out.println("do you want to check membership of a word?" + '\n' + "1:yes" + '\n' + "2:No");
    do {
        sel = sc.next();
    } while (!sel.equals("1") && !sel.equals("2"));
    if (sel.equals("2")) {
        break;
    } else {
        System.out.println("please enter the word:");
        String word = sc.next();
        if (reg.checkMembershipByFA(grammarRules,finalStates, word,  index: 0,  Q: 'S')) {
            System.out.println("accepted");
        } else {
            System.out.println("Not accepted");
        }
    }
}
}
```

توضیحات : بعد از اینکه مشخص شد گرامر وارد شده از سمت کاربر منظم است، ابتدا به کمک متد findNullVariables از کلاس contextFree حالات نهایی FA را بدست می آوریم . و بعد ازآن حالت f را که در متد change از کلاس regular به کاررفت، به آن ها اضافه می کنیم.سپس قواعد لامبدا را به کمک متد removeLambdaRules از کلاس contextFree حذف می کنیم و پس ازآن به ترتیب دو متد convertToSimpleForm وchange ازکلاس regular را روی قواعد گرامر اعمال می کنیم و بعد از آن به کمک متد removeUnitRules ازکلاس contextFree قواعد واحد را نیز حذف می کنیم . درانتها به کمک متد checkMembershipByFA از کلاس regular عضویت رشته وارد شده توسط کاربررا بررسی و نتیجه را چاپ می کنیم.

```
else if (select.equals("2")) {
    ArrayList<String> grammarRules = new ArrayList<>();
    System.out.println("please enter your rules one by one(with this format : leftSide->rightSide) and enter -1 at the end");
    int flag = 0;
    while (flag != -1) {
        String grammarRule = sc.next();
        if (grammarRule.equals("-1")) {
            flag = -1;
        } else {
            if (grammarRule.charAt(1) == '-' && grammarRule.charAt(2) == '>' && Character.isUpperCase(grammarRule.charAt(0)))
                grammarRules.add(grammarRule);
            } else {
                System.out.println("invalid format or input grammar is not context free");
                flag = -1;
            }
        }
    }
    if(grammarRules.isEmpty()){
        continue;
    }
    System.out.println("input grammar is context free");
```

توضیحات : بعد از انتخاب بررسی مستقل از متن بودن توسط کاربر، در این قسمت از متد به بررسی مستقل از متن بودن گرامری که کاربر وارد می کند می پردازیم.(شرط مستقل ازمتن بودن این است که سمت چپ هرقاعده گرامر داده شده یک حرف بزرگ (Variable) باشد.

```
            String sel = "";
            contextFree contextfree = new contextFree();
            contextfree.normalize(grammarRules);
            while (true) {
                System.out.println("do you want to check membership of a word?" + '\n' + "1:yes" + '\n' + "2:No");
                do {
                    sel = sc.next();
                } while (!sel.equals("1") && !sel.equals("2"));
                if (sel.equals("2")) {
                    break;
                } else {
                    System.out.println("please enter the word:");
                    String word = sc.next();
                    Stack<Character> stack = new Stack<>();
                    stack.push( item: '$');
                    stack.push( item: 'S');
                    if (contextfree.checkMembershipByPDA(grammarRules, word,  index: 0, stack)) {
                        System.out.println("accepted");
                    } else {
                        System.out.println("Not accepted");
                    }
                }
            }
        }
        else {
            break;
        }
    }
}
}
```

توضیحات : بعد از اینکه مشخص شد گرامر وارد شده از سمت کاربر مستقل ازمتن است، ابتدا به کمک متد normalize از کلاس contextFree آن گرامر را به فرم

نرمال گریباخ درمی آوریم سپس با استفاده از متد checkMembershipByPDA از همان کلاس عضویت رشته وارد شده توسط کاربررا بررسی و نتیجه را چاپ می کنیم.

<u>یک نمونه گرامرمنظم و بررسی عضویت چند رشته درآن :</u>

S->abcS

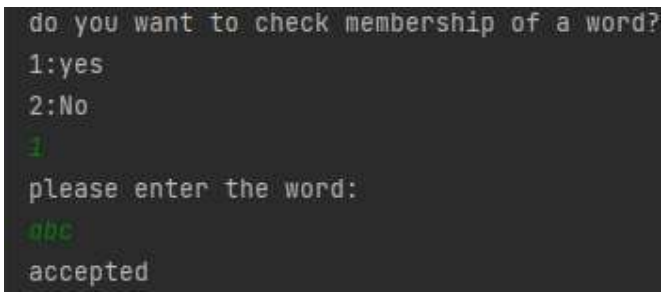
S->A

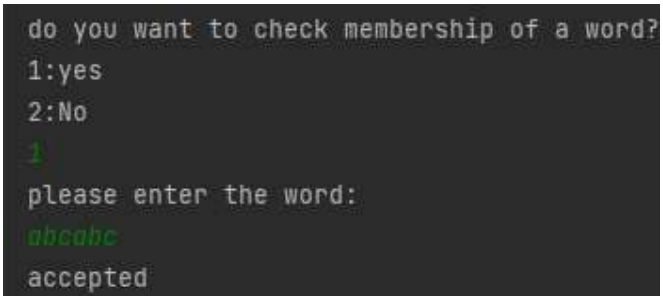
A->aaA

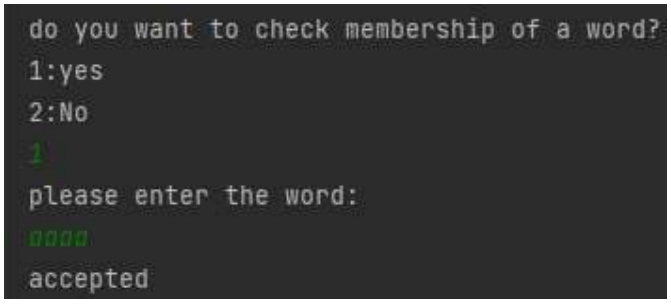
A->B

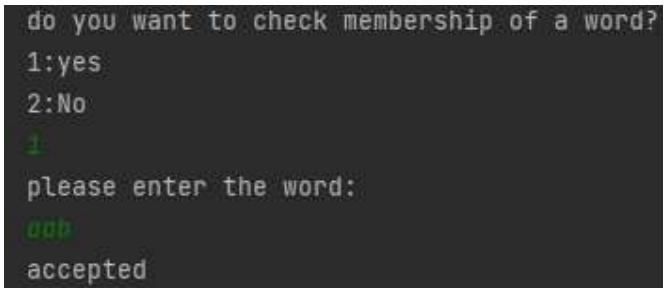
B->bB

B->.









```
do you want to check membership of a word?
1:yes
2:No
1
please enter the word:
aabb
accepted
```

```
do you want to check membership of a word?
1:yes
2:No
1
please enter the word:
aba
Not accepted
```

```
do you want to check membership of a word?
1:yes
2:No
1
please enter the word:
a
Not accepted
```

یک نمونه گرامرمستقل ازمتن و بررسی عضویت چند رشته درآن :

S->aA

A->aABC

A->bB

A->a

B->b

C->c

```
please enter your rules one by one(with this format : leftSide->rightSide) and enter -1 at the end
S->aA
A->aABC
A->bB
A->a
B->b
C->c
-1
input grammar is context free
```

```
do you want to check membership of a word?
1:yes
2:No
1
please enter the word:
a
Not accepted
```

```
do you want to check membership of a word?
1:yes
2:No
1
please enter the word:
aa
accepted
```

```
do you want to check membership of a word?
1:yes
2:No
1
please enter the word:
abb
accepted
```

```
do you want to check membership of a word?
1:yes
2:No
1
please enter the word:
aaabc
accepted
```

```
do you want to check membership of a word?
1:yes
2:No
1
please enter the word:
aabbc
Not accepted
```

```
do you want to check membership of a word?
1:yes
2:No
1
please enter the word:
aaabbc
accepted
```