```
1  //#################### Canonical Collection Generator ####################
2  //#################### CODED BY: AHMED ALI ALSAMMAN ######################
3  // Input: Grammar G
4  // Output: Canonical Collection LR0 Items
5  using System;
6  using System.Collections.Generic;
7  using System.Collections;
8  using System.Linq;
9
10 namespace CanonicalCollection
11 {
12     class Program
13     {
14         // static List<string> G = new List<string>() { "S→aABe", "A→Abc", "A→b",
              "B→d" };
15         static List<char> X = new List<char>(); // for input symbols
16
17         static List<string> G = new List<string>() { "E→E+T", "E→T", "T→T*F",
            "T→F", "F→(E)", "F→i" };
18
19         static ArrayList collection = new ArrayList();// for canonical items
            collection
20
21         static void Main()
22         {
23             int itemIndex;
24             Console.OutputEncoding = System.Text.Encoding.UTF8; // to write → on
                console
25             List<int> visited = new List<int>(); // to keep track of finished
                items
26
27             //Get the list of input symbols of the grammar
28             foreach (string pr in G)
29                 foreach (char ch in pr)
30                     if (ch != '→' && !X.Contains(ch))
31                         X.Add(ch);
32
33             string augG = G[0][0] + "'→" + G[0][0]; // augmented grammar G' (E'-
                >E)
34             string prod = augG.Substring(0, 3) + '.' + augG.Substring(3); //
                perform closure for E'->.E to get I0
35
36             List<string> Item = closure(new List<string>() { prod });  // perform
                closure for E'->.E to get I0
37             collection.Add(Item);
38
39             print(Item);
40
41             int itemNo = 0;// to keep track of generated item number
42             int prevCount; // to keep track of previous number of elements
43
44
```

```
45              do
46              {
47                  prevCount = collection.Count;
48
49                  ArrayList coll = new ArrayList(collection);
50
51                  foreach (List<string> I in coll) // perform Goto for every item I
                       in the collection
52                  {
53                      int index = coll.IndexOf(I);
54
55                      if (!visited.Contains((index))) // check weather current item
                         I has done or not
56                      {
57
58                          visited.Add(index); // mark current I as visited
59                          foreach (char x in X) // check all input symbols against
                        current Item I
60                          {
61                              Item = new List<string>();
62                              Item = Goto(I, x);
63                              itemIndex = itemExists(Item);
64
65                              if (Item.Count > 0 && itemIndex == -1) // non zero
                        elements item & not found in the collection
66                              {
67                                  collection.Add(Item); // add a new item to the
                        collection
68                                  itemNo++;
69                                  print(Item, x, index, itemNo);
70                              }
71
72                              else if (itemIndex != -1) // item found
73                              { // refere to the item
74                                  Console.WriteLine("Goto(I{0},{1})=I{2}", index,
                        x, itemIndex);
75                              }
76                          }
77                      }
78                  }
79
80              } while (prevCount < collection.Count); // exits whenever no new
                   items were derived
81          }
82
83
84
85
86
87
88
89
```

```
90          // Following are two forms of print method with different parameters
              (polymorphism)
91          private static void print(List<string> Item) // to print item0
92          {
93              Console.WriteLine("Canonical Collection");
94              Console.WriteLine("--------------------\n\n");
95              Console.WriteLine("I0");
96              Console.WriteLine("----------------");
97              foreach (string pr in Item)
98              Console.WriteLine(pr);
99          }
100
101
102         private static void print(List<string> Item, char x, int index, int
              itemNo) // to print items from I1 and up.
103         {
104            Console.WriteLine("\n\nI{0} = goto(I{1},{2})", itemNo, index, x);
105            Console.WriteLine("----------------");
106            foreach (string pr in Item)
107            Console.WriteLine(pr);
108            Console.WriteLine();
109
110         }
111
112
113         private static List<string> Goto(List<string> I, char x)
114         {
115             List<string> item = new List<string>();
116            // List<string> part2 = new List<string>();
117             foreach (string pr in I)
118             {
119                 int dotIndx = pr.IndexOf('.');
120                 if (dotIndx != pr.Length - 1 && pr[dotIndx + 1] == x) // if dot
                      isn't the last
121                    //character and the next character matched.
122                 {
123                     string newPr = pr.Substring(0, dotIndx) + pr[dotIndx + 1]
124                           + '.' + pr.Substring(dotIndx + 2);// move dot to the
                       right
125                     dotIndx = newPr.IndexOf('.'); //dotIndx++;
126                     //perform closure if necessary
127                     if (dotIndx != newPr.Length - 1 && Char.IsUpper(newPr[dotIndx
                        + 1]))//if dot isn't the last char.
128                     // and followed by a non terminal
129                     item = closure(new List<string>() { newPr });
130                     else
131                       item.Add(newPr);
132                 }
133             }
134
135             return item;
136         }
```

```
137
138         private static List<string> closure(List<string> Item)
139         {
140             List<char> closured = new List<char>();// for closured NT
141             int prevCount;
142             do
143             {
144                 prevCount = Item.Count();
145                 List<string> I = new List<string>(Item);
146                 foreach (string prod in I)
147                 {
148
149                     char symbol = prod[prod.IndexOf('.') + 1];
150
151
152
153                     if (Char.IsUpper(symbol) && !closured.Contains(symbol)) //
                        perform closure
154                     {
155                         foreach (string gProd in G)
156                             if (gProd[0] == symbol)
157                                 Item.Add(gProd.Substring(0, 2) + "." +
                        gProd.Substring(2));
158                         closured.Add(symbol); // mark symbol as closured
159                       }
160                   }
161               } while (prevCount < Item.Count());
162             return Item;
163         }
164
165         private static int itemExists(List<string> Item)
166         {
167          // A method to check out weather the received item is new or not.
168          // It will check the received item against all items in the collection
169          // first by matching the number of elements
170          // and second by matching the production rules one to one
171
172             bool matched;
173             int itemIndex = 0;
174
175             foreach (List<string> I in collection)
176             {
177                 int prodIndex = 0;
178                 matched = true;
179               if(Item.Count == I.Count) //perform matching if they have the same
                  number of elemets
180               {
181
182
183
184
185
```

```
186
187                     foreach (string pr in Item)
188                     {
189                         if (!I[prodIndex].Equals(pr)) // if (I[index]!=pr)
190                         {
191                             matched = false;
192                             break; // skip to the next item I in the collection
193                         }
194                         prodIndex++;
195                     }
196                     if (matched) return itemIndex;
197                 }
198                   itemIndex++;
199             }
200             return -1;
201         }
202     }
203 }
204
```