Exercise 4
Recursive Decent Parser

Praveen Kumar R
312217104114

February 1, 2020

Aim

To Write a C program that eliminates the left recursion in a grammar and produce left recursion free productions.

Left Recursion

A production is said to have left recursion if the first symbol of the production is the non-terminal on the left hand side of the production.

$$A \rightarrow A\alpha_1|A\alpha_2|A\alpha_3\ ..|A\alpha_m|\beta_1|\beta_2|\ ..|\beta_n$$

where $\alpha_i, \beta_i \in (N \cup T)^*$. The production has to be rewritten as,

$$A \rightarrow \beta_1 A'|\beta_2 A'|\beta_3 A'|..|\beta_n A'$$
$$A' \rightarrow \alpha_1 A'|\alpha_2 A'|\alpha_3 A'..|\alpha_m A'|\epsilon$$

C Program

```c
#include<stdio.h>
#include<string.h>

void  slice(char a[],int start,int end,char temp[]){
  int pos = 0;
  temp[pos]='\0';
  int i = start;
  while(i<end){
    temp[pos]=a[i];

    pos++;
    i++;
```

```
  }
  temp[pos]='\0';
}
char getNonTerminal(char p[]){
  return p[0];
}


int getProductionRHS(char p[],char prod[100][100]){
  int i = 0;
  while(p[i]!='>') i++;
  int k = 0,t = 0,j = 0;
  char temp[100];
  while(p[i]!='\0'){
    i++;
    if(p[i]=='|'||p[i]=='\0'){
      temp[j] = '\0';
      strcpy(prod[k++],temp);
      j =0;
      temp[j] = '\0';
    }
    else{
      temp[j++] = p[i];
    }

  }
  return k;
}

int isLeftRecurrsion(char nt,char p[100][100],int n){
  for(int i = 0;i<n;i++){
    if(p[i][0]==nt) return 1;
  }
  return 0;
}

int getBetaList(char nt,char p[100][100],int n,char beta[100][100]){
  int k = 0;
   for(int i = 0;i<n;i++){
     if(p[i][0]!=nt) {
       strcpy(beta[k++],p[i]);
     }
  }
   return k;
}
```

```
int getAlphaList(char nt,char p[100][100],int n,char alpha[100][100]){
  int k = 0;
  char temp[100];
   for(int i = 0;i<n;i++){
     if(p[i][0]==nt) {
       slice(p[i],1,strlen(p[i]),temp);
       strcpy(alpha[k++],temp);
     }
  }
   return k;
}

void getProduction1(char nt,char beta[100][100],char prod1[100][100],int n){
  if(n!=0){
    for(int i = 0;i < n ; i++){
      char temp[100]="",temp1[5];
      strcat(temp,beta[i]);
      temp1[0]=nt;
      temp1[1]='\'';
      temp1[2]='\0';
      strcat(temp,temp1);
      strcpy(prod1[i],temp);
    }
  }
  else{
    char temp1[5];
    temp1[0]=nt;
    temp1[1]='\'';
    temp1[2]='\0';
    strcpy(prod1[0],temp1);
  }
}

void getProduction2(char nt,char alpha[100][100],char prod2[100][100],int n){
  int i;
  for( i = 0;i < n ; i++){
    char temp[100]="";
    strcat(temp,alpha[i]);
    int l = strlen(temp);
    temp[l]=nt;
    temp[l+1]='\'';
    temp[l+2] = '\0';
    strcpy(prod2[i],temp);
  }
  strcpy(prod2[i],"epsilon");
}
```

```
 void displayProduction(char nt[],char prod[100][100],int n){
   printf("%s->%s",nt,prod[0]);
   for(int i = 1; i < n; i++){
     printf("|%s",prod[i]);
   }
   printf("\n");
 }

int main()
{
  char a[100][100];
  int n;
  printf("Enter the number of productions: ");
  scanf("%d",&n);
  printf("Enter %d productions: Example: A->abaA\n",n);
  for(int i = 0; i<n;i++)
    {
      scanf("%s",a[i]);
    }
  printf("\nGrammar after removing all left recurrsion: \n\n");
  for(int i = 0;i<n;i++){
    char nt[5];
    char prod[100][100],alpha[100][100],beta[100][100];
    char modifiedProduction1[100][100],modifiedProduction2[100][100];
    int at,bt,k;
    nt[0] = getNonTerminal(a[i]);
    nt[1] = '\0';
    k = getProductionRHS(a[i],prod);
    if(isLeftRecurrsion(nt[0],prod,k)){
      at = getAlphaList(nt[0],prod,k,alpha);
      bt = getBetaList(nt[0],prod,k,beta);
      getProduction1(nt[0],beta,modifiedProduction1,bt);
      getProduction2(nt[0],alpha,modifiedProduction2,at);
      char newnt[5];
      newnt[0] = nt[0];
      newnt[1] = '\'';
      newnt[2] = '\0';
      displayProduction(nt,modifiedProduction1,bt);
      displayProduction(newnt,modifiedProduction2,at+1);
    }
    else{
      displayProduction(nt,prod,k);
    }

  }
```

```
  return 0;

}
```

Sample Input & Output 1

```
Enter the number of productions: 3
Enter 3 productions: Example: A->abaA
A->Abc|AAbc|BBca|Sasa
B->CbaB|ccac|CCAC
C->Baac|CbbA

Grammar after removing all left recurrsion:

A->BBcaA'|SasaA'
A'->bcA'|AbcA'|epsilon
B->CbaB|ccac|CCAC
C->BaacC'
C'->bbAC'|epsilon
```