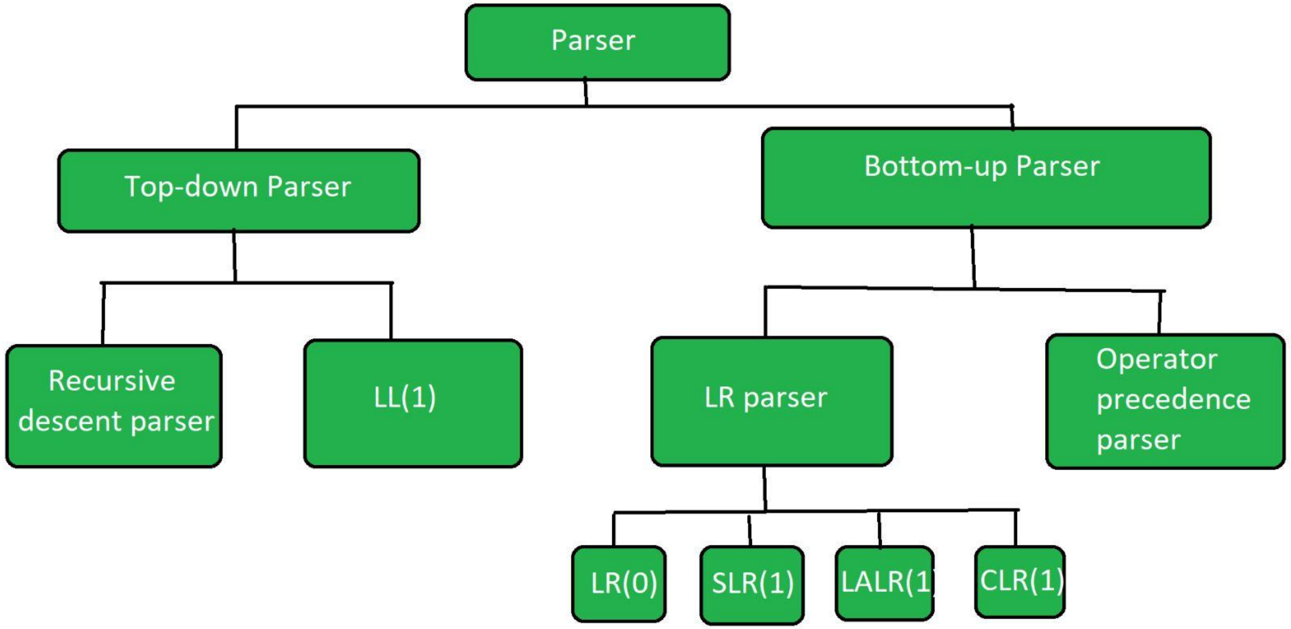
Parser Generator

Introduction

The **CLR parser** stands for canonical LR parser.It is a more powerful LR parser. It makes use of lookahead symbols. This method uses a large set of items called LR(1) items. The main difference between LR(0) and LR(1) items is that, in LR(1) items, it is possible to carry more information in a state, which will rule out useless reduction states. This extra information is incorporated into the state by the lookahead symbol. The general syntax becomes [A->∝.B, a]
where A->∝.B is the production and a is a terminal or right end marker $
LR(1) items=LR(0) items + look ahead

Parsing

The **parser** is that phase of the compiler which takes a token string as input and with the help of existing grammar, converts it into the corresponding Intermediate Representation(IR). The parser is also known as *Syntax Analyzer.*



Types of Parser:

The parser is mainly classified into two categories, i.e. Top-down Parser, and Bottom-up Parser. These are explained below:

Top-Down Parser:

The top-down parser is the parser that **generates parse for the given input string** with the help of grammar productions by expanding the non-terminals i.e. it starts from the start symbol and ends on the terminals. It uses left most derivation. Further Top-down parser is classified into 2 types: A recursive descent parser, and Non-recursive descent parser.

Bottom-up Parser:

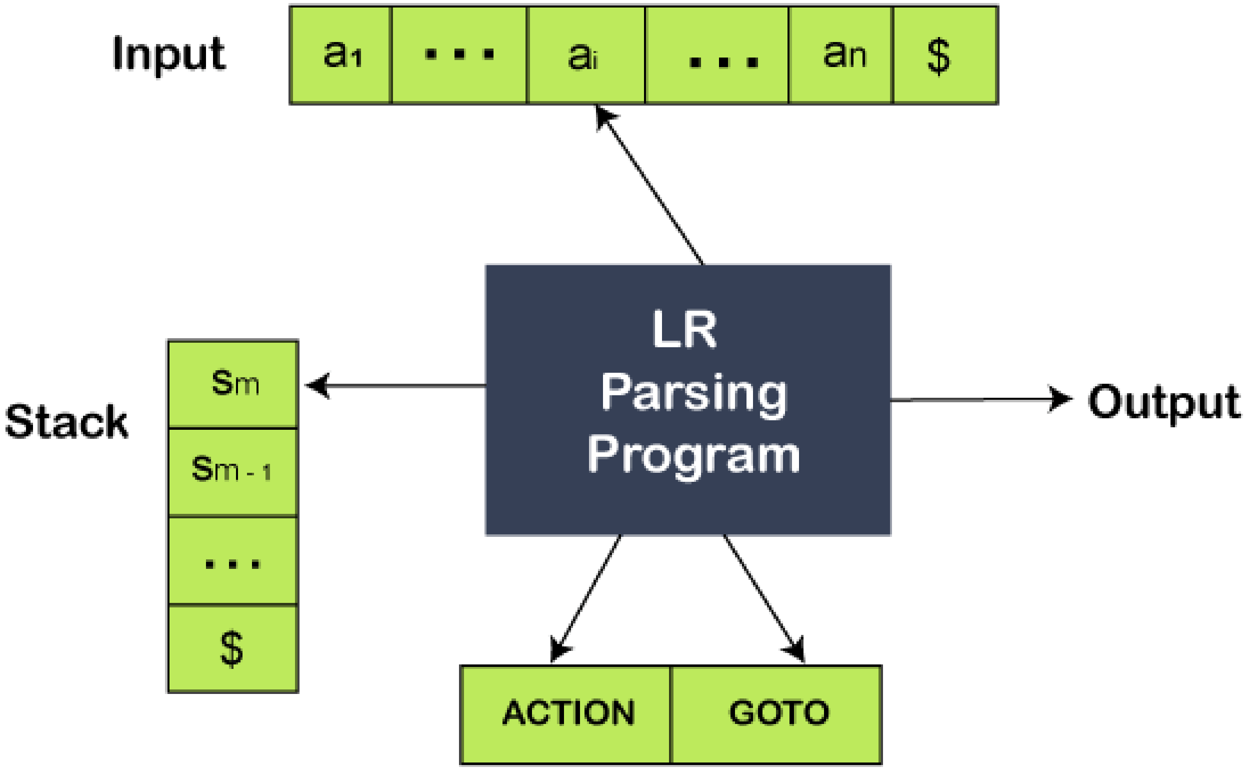
Bottom-up Parser is the parser that generates the parse tree for the given input string with the help of grammar productions by compressing the non-terminals i.e. it starts from non-terminals and ends on the start symbol. It uses the reverse of the rightmost derivation.

Further Bottom-up parser is classified into two types: LR parser, and Operator precedence parser.

Component Diagram

Various steps involved in the CLR (1) Parsing:

- For the given input string write a context free grammar
- Check the ambiguity of the grammar
- Add Augment production in the given grammar
- Create Canonical collection of LR (0) items
- Draw a data flow diagram (DFA)
- Construct a CLR (1) parsing table



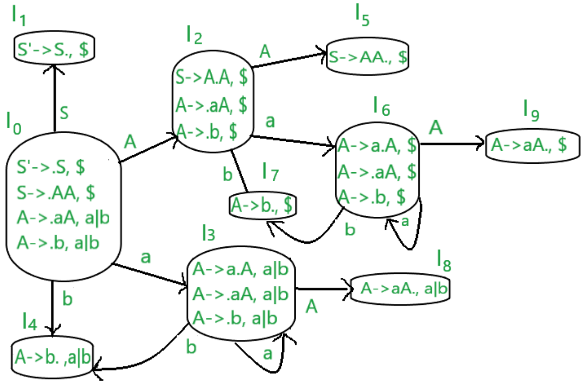
Canonical Parser

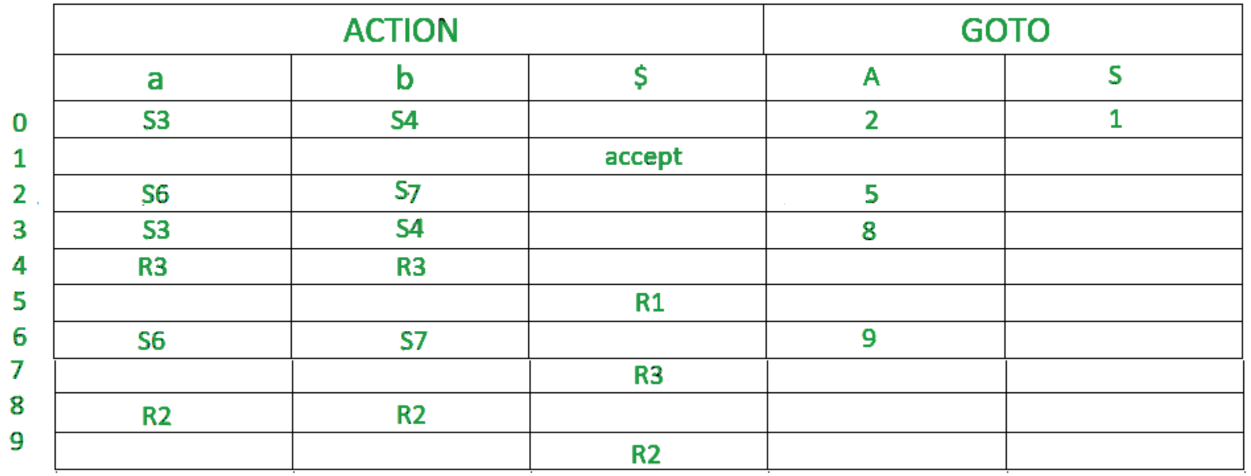
The CLR parser stands for canonical LR parser.It is a more powerful LR parser.It makes use of lookahead symbols. This method uses a large set of items called LR(1) items. The main difference between LR(0) and LR(1) items is that, in LR(1) items, it is possible to carry more information in a state, which will rule out useless reduction states. This extra information is incorporated into the state by the lookahead symbol. The general syntax becomes

[A->∝.B,a]

where A->∝.B is the production and a is a terminal or right end marker $

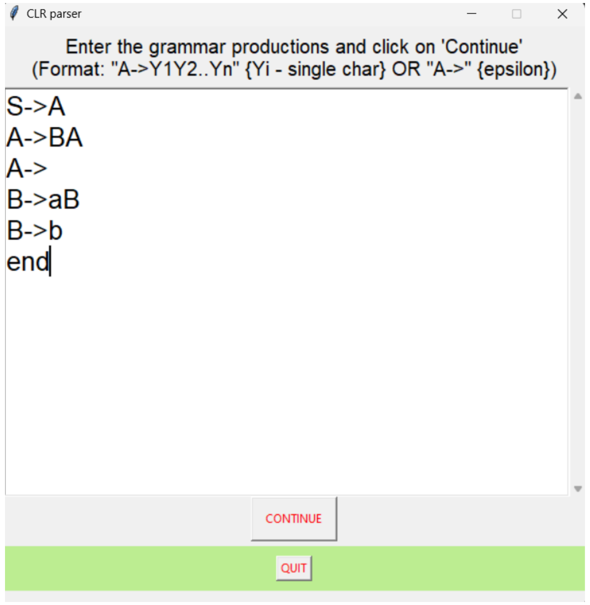
LR(1) items=LR(0) items + look ahead





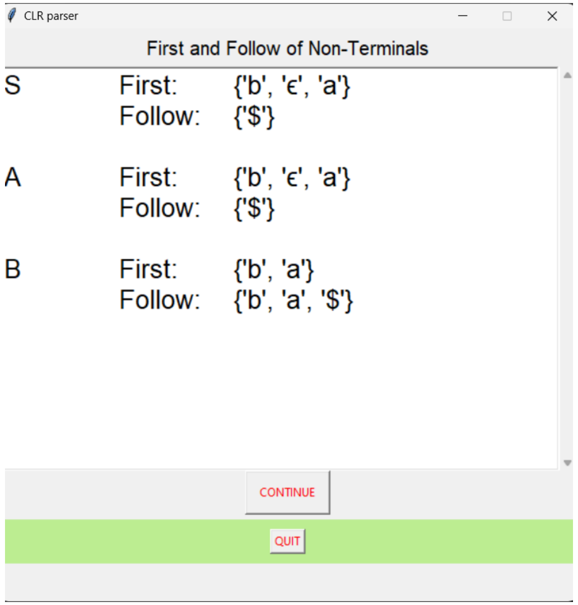
	ACTION			GOTO	
	a	b	$	A	S
0	S3	S4		2	1
1			accept		
2	S6	S7		5	
3	S3	S4		8	
4	R3	R3			
5			R1		
6	S6	S7		9	
7			R3		
8	R2	R2			
9			R2		

Our Project


CLR parser

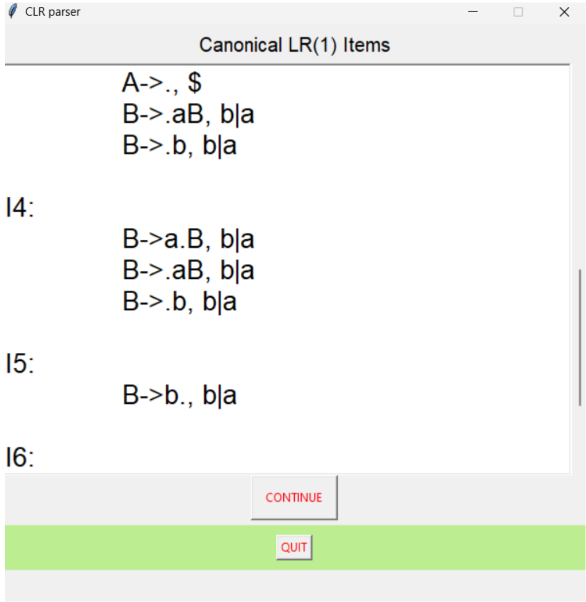
Enter the grammar productions and click on 'Continue'
(Format: "A->Y1Y2..Yn" {Yi - single char} OR "A->" {epsilon})

```
S->A
A->BA
A->
B->aB
B->b
end
```


CLR parser

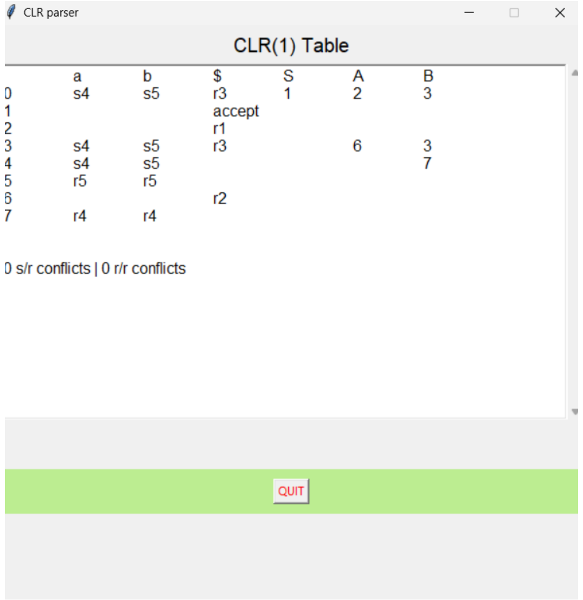
First and Follow of Non-Terminals

```
S            First:     {'b', 'ϵ', 'a'}
             Follow:    {'$'}

A            First:     {'b', 'ϵ', 'a'}
             Follow:    {'$'}

B            First:     {'b', 'a'}
             Follow:    {'b', 'a', '$'}
```


CLR parser

Canonical LR(1) Items

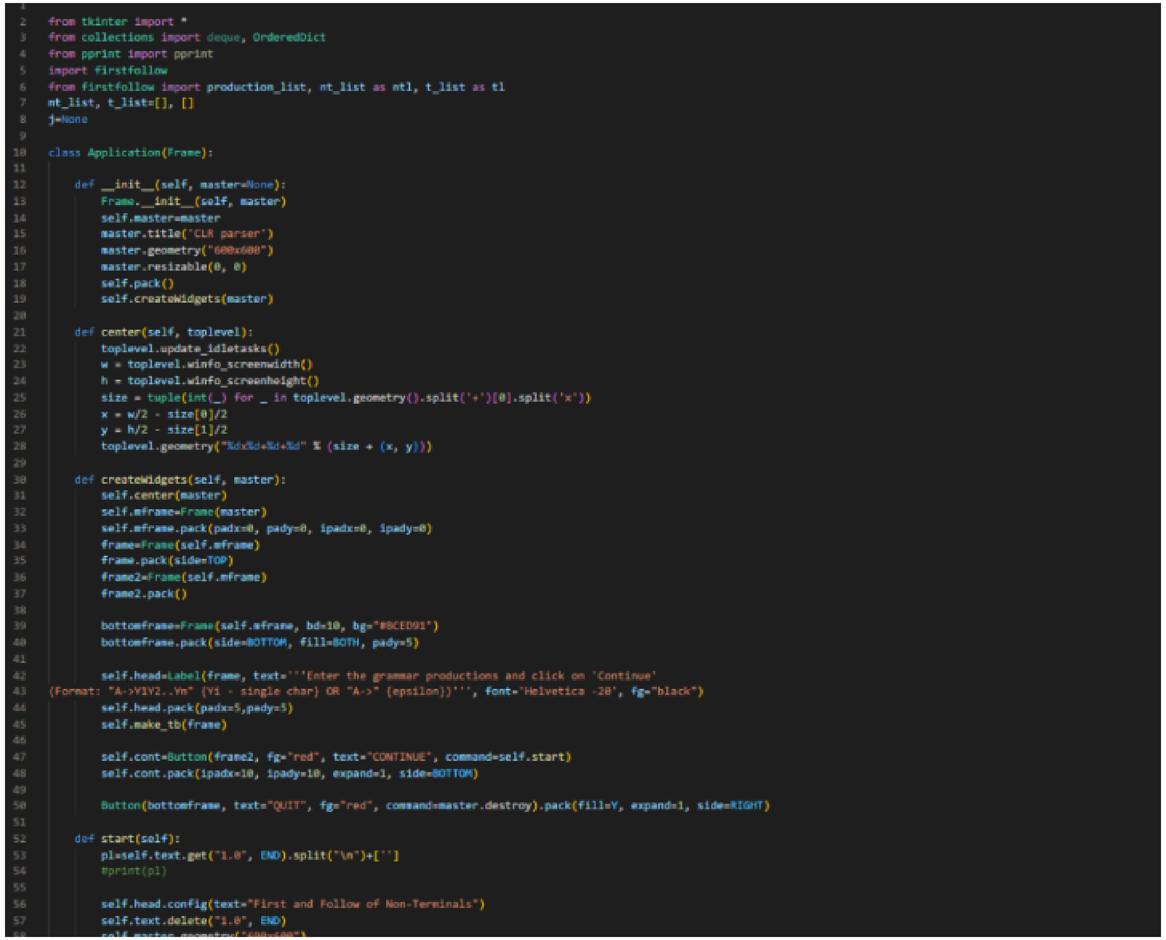
```
             A->., $
             B->.aB, b|a
             B->.b, b|a

I4:
             B->a.B, b|a
             B->.aB, b|a
             B->.b, b|a

I5:
             B->b., b|a

I6:
```


CLR parser

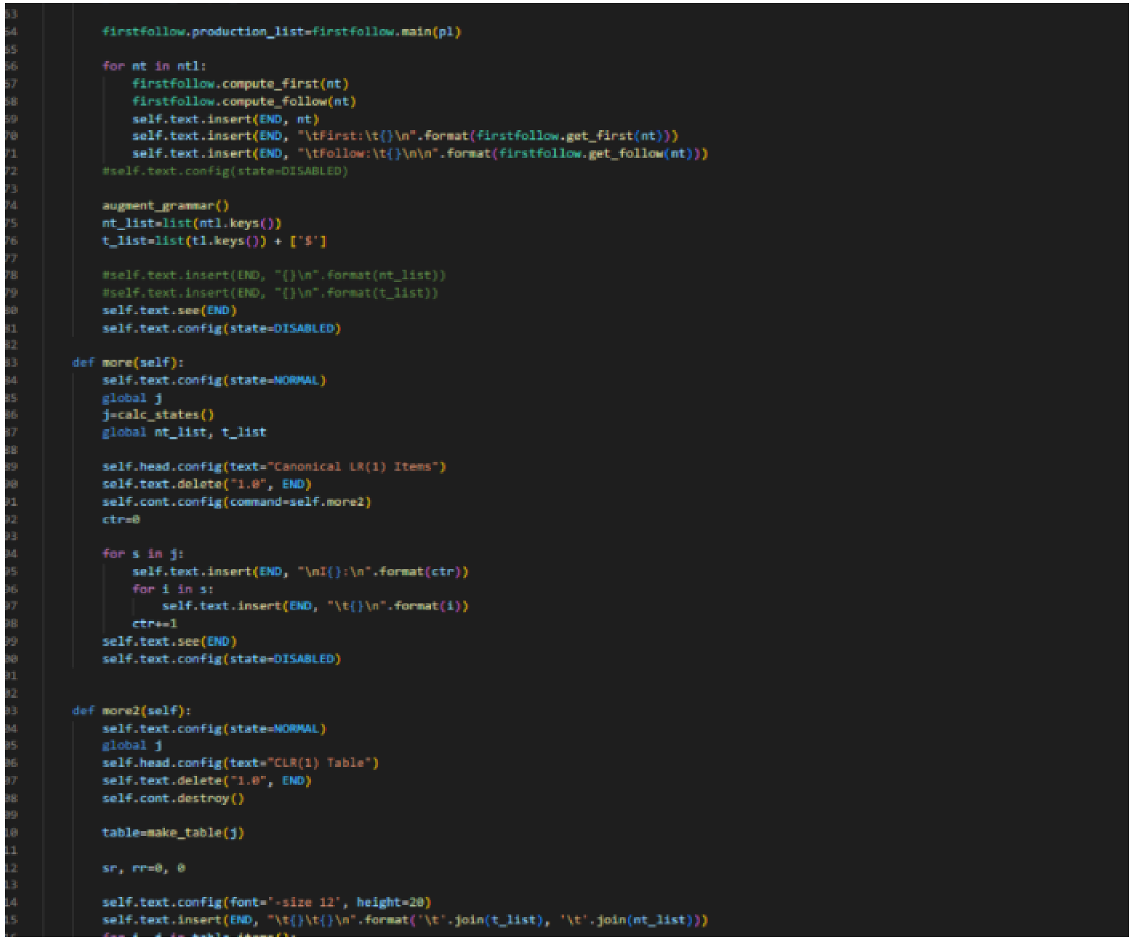
CLR(1) Table

```
        a       b       $       S       A       B
0       s4      s5      r3      1       2       3
1                       accept
2                       r1
3       s4      s5      r3              6       3
4       s4      s5                              7
5       r5      r5
6                       r2
7       r4      r4

0 s/r conflicts | 0 r/r conflicts
```

Code Snippet

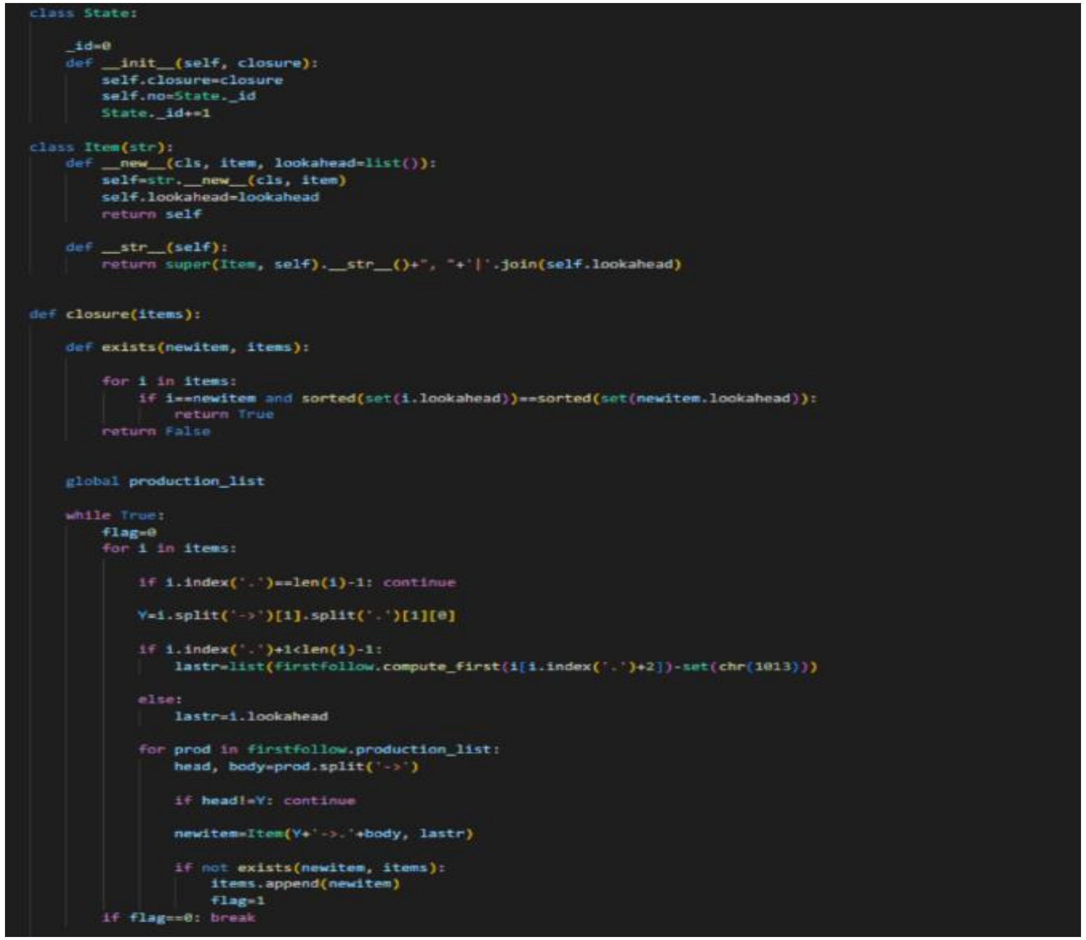


```python
from tkinter import *
from collections import deque, OrderedDict
from pprint import pprint
import firstfollow
from firstfollow import production_list, nt_list as ntl, t_list as tl
nt_list, t_list=[], []
j=None

class Application(Frame):

    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.master=master
        master.title('CLR parser')
        master.geometry("600x600")
        master.resizable(0, 0)
        self.pack()
        self.createWidgets(master)

    def center(self, toplevel):
        toplevel.update_idletasks()
        w = toplevel.winfo_screenwidth()
        h = toplevel.winfo_screenheight()
        size = tuple(int(_) for _ in toplevel.geometry().split('+')[0].split('x'))
        x = w/2 - size[0]/2
        y = h/2 - size[1]/2
        toplevel.geometry("%dx%d+%d+%d" % (size + (x, y)))

    def createWidgets(self, master):
        self.center(master)
        self.mframe=Frame(master)
        self.mframe.pack(padx=0, pady=0, ipadx=0, ipady=0)
        frame=Frame(self.mframe)
        frame.pack(side=TOP)
        frame2=Frame(self.mframe)
        frame2.pack()

        bottomframe=Frame(self.mframe, bd=10, bg="#BCED91")
        bottomframe.pack(side=BOTTOM, fill=BOTH, pady=5)

        self.head=Label(frame, text='''Enter the grammar productions and click on 'Continue'
(Format: "A->Y1Y2..Yn" {Yi - single char} OR "A->" {epsilon})''', font='Helvetica -20', fg="black")
        self.head.pack(padx=5,pady=5)
        self.make_tb(frame)

        self.cont=Button(frame2, fg="red", text="CONTINUE", command=self.start)
        self.cont.pack(ipadx=10, ipady=10, expand=1, side=BOTTOM)

        Button(bottomframe, text="QUIT", fg="red", command=master.destroy).pack(fill=Y, expand=1, side=RIGHT)

    def start(self):
        pl=self.text.get("1.0", END).split("\n")+['']
        #print(pl)

        self.head.config(text="First and Follow of Non-Terminals")
        self.text.delete("1.0", END)
        self.master.geometry("600x600")
```

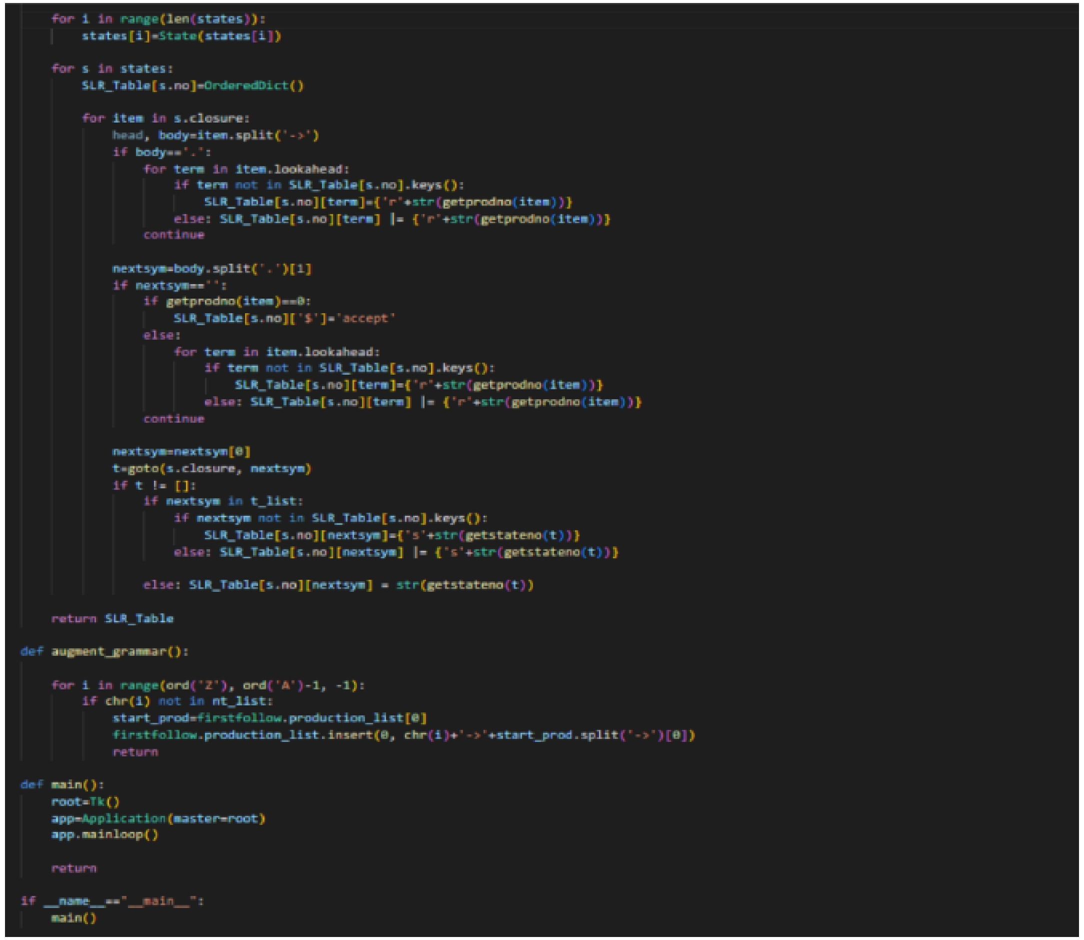


```python

        firstfollow.production_list=firstfollow.main(pl)

        for nt in ntl:
            firstfollow.compute_first(nt)
            firstfollow.compute_follow(nt)
            self.text.insert(END, nt)
            self.text.insert(END, "\tFirst:\t{}\n".format(firstfollow.get_first(nt)))
            self.text.insert(END, "\tFollow:\t{}\n\n".format(firstfollow.get_follow(nt)))
        #self.text.config(state=DISABLED)

        augment_grammar()
        nt_list=list(ntl.keys())
        t_list=list(tl.keys()) + ['$']

        #self.text.insert(END, "{}\n".format(nt_list))
        #self.text.insert(END, "{}\n".format(t_list))
        self.text.see(END)
        self.text.config(state=DISABLED)

    def more(self):
        self.text.config(state=NORMAL)
        global j
        j=calc_states()
        global nt_list, t_list

        self.head.config(text="Canonical LR(1) Items")
        self.text.delete("1.0", END)
        self.cont.config(command=self.more2)
        ctr=0

        for s in j:
            self.text.insert(END, "\nI{}:\n".format(ctr))
            for i in s:
                self.text.insert(END, "\t{}\n".format(i))
            ctr+=1
        self.text.see(END)
        self.text.config(state=DISABLED)


    def more2(self):
        self.text.config(state=NORMAL)
        global j
        self.head.config(text="CLR(1) Table")
        self.text.delete("1.0", END)
        self.cont.destroy()

        table=make_table(j)

        sr, rr=0, 0

        self.text.config(font='-size 12', height=20)
        self.text.insert(END, "\t{}\t{}\n".format('\t'.join(t_list), '\t'.join(nt_list)))
        for i, j in table.items():
```

Code Snippet



```python
class State:

    _id=0
    def __init__(self, closure):
        self.closure=closure
        self.no=State._id
        State._id+=1

class Item(str):
    def __new__(cls, item, lookahead=list()):
        self=str.__new__(cls, item)
        self.lookahead=lookahead
        return self

    def __str__(self):
        return super(Item, self).__str__()+", "+'|'.join(self.lookahead)


def closure(items):

    def exists(newitem, items):

        for i in items:
            if i==newitem and sorted(set(i.lookahead))==sorted(set(newitem.lookahead)):
                return True
        return False


    global production_list

    while True:
        flag=0
        for i in items:

            if i.index('.')==len(i)-1: continue

            Y=i.split('->')[1].split('.')[1][0]

            if i.index('.')+1<len(i)-1:
                lastr=list(firstfollow.compute_first(i[i.index('.')+2])-set(chr(1013)))

            else:
                lastr=i.lookahead

            for prod in firstfollow.production_list:
                head, body=prod.split('->')

                if head!=Y: continue

                newitem=Item(Y+'->.'+body, lastr)

                if not exists(newitem, items):
                    items.append(newitem)
                    flag=1
        if flag==0: break
```



```python
    for i in range(len(states)):
        states[i]=State(states[i])

    for s in states:
        SLR_Table[s.no]=OrderedDict()

        for item in s.closure:
            head, body=item.split('->')
            if body=='.':
                for term in item.lookahead:
                    if term not in SLR_Table[s.no].keys():
                        SLR_Table[s.no][term]={'r'+str(getprodno(item))}
                    else: SLR_Table[s.no][term] |= {'r'+str(getprodno(item))}
                continue

            nextsym=body.split('.')[1]
            if nextsym=='':
                if getprodno(item)==0:
                    SLR_Table[s.no]['$']='accept'
                else:
                    for term in item.lookahead:
                        if term not in SLR_Table[s.no].keys():
                            SLR_Table[s.no][term]={'r'+str(getprodno(item))}
                        else: SLR_Table[s.no][term] |= {'r'+str(getprodno(item))}
                continue

            nextsym=nextsym[0]
            t=goto(s.closure, nextsym)
            if t != []:
                if nextsym in t_list:
                    if nextsym not in SLR_Table[s.no].keys():
                        SLR_Table[s.no][nextsym]={'s'+str(getstateno(t))}
                    else: SLR_Table[s.no][nextsym] |= {'s'+str(getstateno(t))}

                else: SLR_Table[s.no][nextsym] = str(getstateno(t))

    return SLR_Table

def augment_grammar():

    for i in range(ord('Z'), ord('A')-1, -1):
        if chr(i) not in nt_list:
            start_prod=firstfollow.production_list[0]
            firstfollow.production_list.insert(0, chr(i)+'->'+start_prod.split('->')[0])
            return

def main():
    root=Tk()
    app=Application(master=root)
    app.mainloop()

    return

if __name__=="__main__":
    main()
```

Thank You