برای طراحی کامپایلر از کتابخانه ی regular expression (re) زبان پایتون کمک گرفته شده. [1]

ابتدا تابع compiler صدا زده میشود.

```
323  def compiler():
324      file_py=open('p.txt','r').read()
325      # print(file_py)
326      file_py_sp=file_py.split('\n')
327      # print(len(file_py_sp))
328      # give code to lexical analyzer
329      for index,code_line in enumerate(file_py_sp,start=1):
330          lexical_analysis(index,code_line,file_py)
331      show_key()
332      # give code to syntax analyzer
333      for index,code_line in enumerate(file_py_sp,start=1):
334          # if index==len(file_py_sp):
335          #     break
336          # print(index,code_line)
337          syntax_analysis(index,code_line,file_py_sp)
338
339
```

این تابع فایل p.txt را میخواند. این فایل برنامه پایتون است که در نهایت به کد زبان c تبدیل میشود. فایل را خط به خط split میکنیم و با حلقه for برای هر خط تابع lexical_analysis را صدا میزنیم. Index شماره خط است که چون در enumerate متغیر start=1 است پس از صفر شروع نمیشود و شماره خط های فایل درست است. تابع lexcal_analysis در ادامه توضیح میدهیم. بخش lexical analysis:

```
33  def lexical_analysis(index,code,file_py):
34      global x
35      # all pattern
36
37      # pattern for symbol key
38      t_for=r'for'
39      t_while=r'while'
40      t_if=r'if'
41      t_elif=r'elif'
42      t_else=r'else'
43      t_break=r'break'
44      t_continue=r'continue'
45      t_return=r'return'
46      t_try=r'try'
47      t_except=r'except'
48      t_in=r'in'
49      t_range=r'range'
50      t_print=r'print'
51      t_input=r'input'
52
53      # pattern for variable
54      t_variable= r'/s[a-zA-Z0-9]*=[a-zA-Z0-9]*/s'
55      # pattern for operator
56      t_equal=r'=='
57      t_equal_not=r'!='
58      t_more=r'>'
59      t_less=r'<'
60      t_more_equal=r'>='
61      t_less_equal=r'<='
62      # t_and=r'and'
63      # t_or=r'or'
```

```
63      # t_or=r'or'
64      # t_not=r'not'
65
66      t_plus_equal=r'+='
67      t_mines_equal=r'-='
68      t_multi_equal=r'*='
69      t_dev_equal=r'/='
70
71      t_plus=r'+'
72      t_mines=r'-'
73      t_multi=r'*'
74      t_dev=r'/'
75      t_bagi=r'%'
76
77
78
79
80      token_key=[t_for,t_while,t_if,t_elif,t_else,t_break,t_continue
81      ,t_return,t_try,t_except,t_in,t_range,t_print,t_input]
82
83      token_op=[
84          t_equal,
85          t_equal_not,
86          t_more,
87          t_less,
88          t_more_equal,
89          t_less_equal,
90          # t_and,
91          # t_or,
92          # t_not,
93          t_plus_equal,
94          t_mines_equal,
95          t_multi_equal,
```

در این تابع ابتدا pattern هایی برای عملگر ها و متغیر ها و کلمات کلیدی مثل for, if,while... تعریف کردیم. مثلا برای for خود کلمه ی "for" را قرار داده ایم. حرف r قبل از string ها آنها را به raw string تبدیل میکند. اپراتور های مقایسه ای و عملگر های انتساب هم تعریف شده اند. عملگر های عادی/*... هم

تعریف شده اند. سپس همه ی symbol key ها (مثل ... for,if) را در لیستی به نام token_key قرار داده شد و مقایسه ای ها و عملگرها و اپراتور ها در لیست token_op.

```
103
104     # get all key and append to list
105     for tk in token_key:
106         if tk in code:
107             if tk not in symbol_key:
108                 symbol_key.append(tk)
109
110     # get op in code
111     for tk in token_op:
112         if tk in code:
113             if tk not in op_list  :
114                 op_list.append(tk)
115
116     # get variable in code
117     # a=12 a+=1
118     pat=r' [a-zA-Z]{1} |^[a-zA-Z][a-zA-Z0-9]*=[0-9a-zA-Z"\']*'
119     x=re.findall(pat,file_py,re.MULTILINE)
120
```

هر خط کد که به تابع lexical_analysis داده میشود با متغیر code معلوم میشود. و بعد symbol_keyها و عملیات هایی که در code هستند در دو لیست ذخیره میکنیم.(op_list و symbol_key)

توضیح برای متغیر pat : عملیات انتساب

{1} در جلوی [a-zA-Z] به این معناست که یکبار حرف دیده شود. و * جلوی

[a-zA-z0-9] به همان معنی closure است. و بعد از تساوی " یا ' هم میتواند باشد.

متد show_key هم نمایش موارد بالاست. یعنی symbol_key و op_list نمایش متغیر ها و کلمات کلیدی و عملیات هایی که در فاز lexical یافت شده.

بخش syntax analysis:

در صفحه بعد کد این بخش آمده. به این تابع شماره خط هر کد index و هر خط کد everylinecode و همچنین تمام کد داده شده allcode. در ادامه نحوه ساخت دستورات مختلف توصیف شده. برای مثال در دستور for: *\s به معنی این است که هرچقدر فاصله وجود داشت نادیده گرفته شود. به این دلیل که در پایتون در حلقه های تو در توindentation نیاز است، این کار برا سادگی انجام شده. بعد کلمه for امده و بعد دوباره یک *\s که باز به معنی نادیده گرفتن فاصله است.سپس اسم متغیر حلقه با *[a-zA-Z] مشخص شده و بعد in range و سپس یک پرانتز باز میکنیم [()] و در انتها هم میبندیم. درون پرانتز دو عدد داریم که هر کدام با *[0-9] مشخص شده اند و بینشان کاما است.و بعد دو نقطه. بقیه تعاریف هم مشابه هستند. در while مقایسه ها با [==|!=|>=|<=|>|<] مشخص شده. همه این pattern ها را در متغی token قرار میدهیم. حال در ادامه به بررسی مطابق بودن کد دریافتی با قوانینی که تعریف کردیم میپردازیم. تابع re.findall(tk,everylinecode) بررسی میکند که آیا هر خط کد ورودی با

Tk که همان قوانین بالا هستند که در متغیر token همه شان را گذاشتیم مطابقت دارد یا نه. اگر درست بود متغیر compire یک لیست مقدار دار میشود و none نمیشود. وسپس اگر طول all_syntax برابر یک نشد این به معنای خطا است و انرا چاپ میکنیم و خطی که ارور دارد با index مشخص میکنیم وگرنه کد درست است و چاپ میکنیم که این خط syntax درست دارد. و سپس compile_to_c() صدا میزنیم.

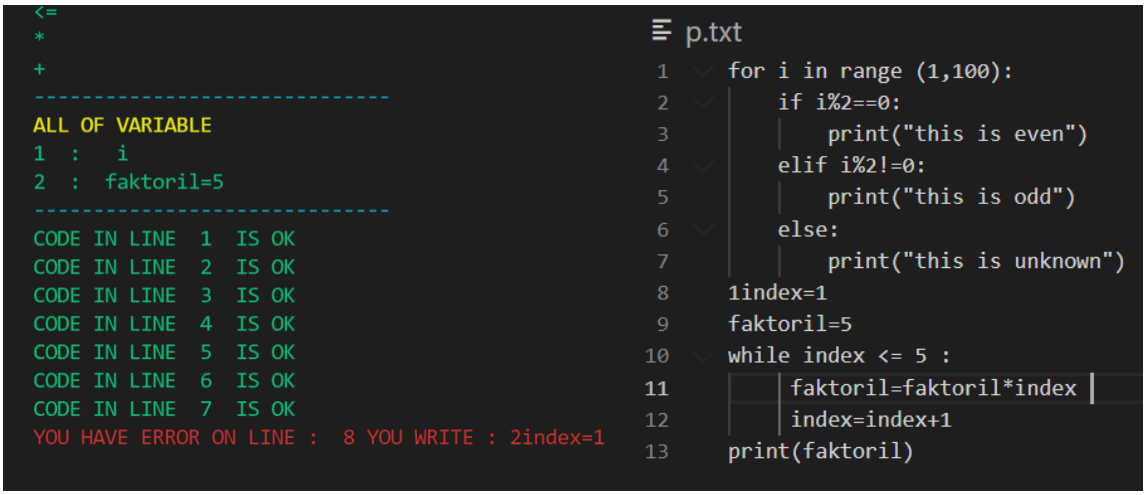
```
counter=1
def syntax_analysis(index,everylinecode,allcode):
    global counter
    # pattern
    # for i in range(1,100):
    pat_for=r'\s*for \s*[a-zA-Z]* in range\s*[(][0-9]*,[0-9]*[)]\s*:'
    # while i<10:
    pat_while=r'\s*while \s*[a-zA-Z0-9_]*\s*[==|!=|>|<|>=|<=]*\s*[0-9a-zA-Z\'"]*\s*:'
    # if i==0:
    pat_if1=r'\s*^if \s*[a-zA-Z0-9_]*\s*[==|!=|>|<|>=|<=]*[0-9a-zA-Z"\']*\s*:'
    # if i%2==0:
    pat_if2=r'^\s*if \s*[a-zA-Z0-9_]*\s*[%]\s*[0-9]*\s*[==|!=|>|<|>=|<=]*\s*[0-9]*\s*:'
    # elif i==4:
    pat_elif1=r'\s*elif \s*[a-zA-Z0-9_]*\s*[==|!=|>|<|>=|<=]*[0-9a-zA-Z"\']*\s*:'
    # elif i%2!=0:
    pat_elif2=r'^\s*elif \s*[a-zA-Z0-9_]*\s*[%]\s*[0-9]*\s*[==|!=|>|<|>=|<=]*\s*[0-9]*\s*:'
    # else:
    pat_else=r'\s*else\s*:'
    # print('hi')
    pat_print1=r'\s*print\s*[(]\s*[\'|"]\s*[a-zA-Z0-9_ ]*\s*[\'|"]\s*[)]'
    # print(i)
    pat_print2=r'\s*print\s*[(]\s*[a-zA-Z0-9_]*\s*[)]'
    # a=a+1
    pat_op=r'[a-zA-Z0-9_]*\s*=\s*[a-zA-Z0-9_]*\s*[+|-|*|/]\s*[0-9a-zA-Z_"\']*'
    # a=1 name='mohsen'
    pat_define_var=r'\s*^[a-zA-Z][a-zA-Z0-9_]*\s*=\s*["0-9a-zA-Z_\'"]*'

    token=[pat_for,pat_while,pat_if1,pat_if2,pat_elif1,pat_elif2,pat_else,pat_print1,pat_print2,pat_op,pat_define_var]
    all_syntax=[]
    for tk in token:
        compire=re.findall(tk,everylinecode,re.MULTILINE)
        compire=''.join(compire)
        if compire=='' or compire==' ' or compire==None:
```

```
    all_syntax=[]
    for tk in token:
        compire=re.findall(tk,everylinecode,re.MULTILINE)
        compire=''.join(compire)
        if compire=='' or compire==' ' or compire==None:
            continue
        all_syntax.append(compire)
    # print(all_syntax)
    if len(all_syntax)!=1:
        print(Fore.RED+'you have error on line : '.upper(),index,'you write : '.upper()+everylinecode)
        print(Fore.WHITE+'')
        quit ()
    else:
        print(Fore.GREEN+'code in line '.upper(),index,' is ok'.upper())
        counter+=1
        if counter==len(allcode):
            compile_to_c()
            pass



    # print(all_line_syntax)
c_code="""
#include<stdio.h>\n
void main()
{}
"""
def compile_to_c():
    global c_code
```

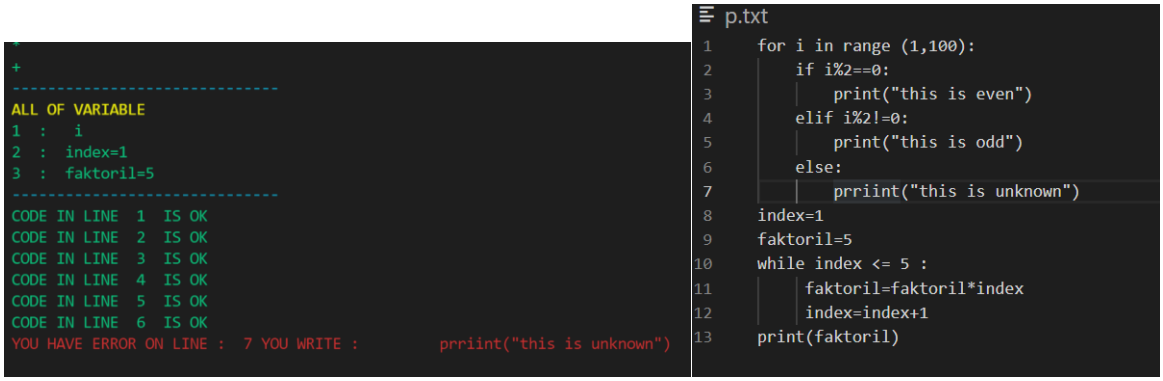
چند مثال از خطای syntax گرفتن:

در فایل py.txt مثلا به جای while مینویسیم whieele میبینیم که خطا را در خط 10ام تشخیص داد.

یا مثلا اسم متغیر index را به 1index تغییر دهیم. در خط 8 خطا گرفت.



یا تابع print را تغییر دهیم به prriint. میبینیم که خطا میگیرد در خط 7.



بخش تبدیل به کد c:

patter های زبانc را تعریف میکنیم مثلا برای for , while,... برای راحتی در اینجاpattern زبان پایتون فقط آن بخشهایی از کد را که نیاز داریم گذاشتیم.

```
c_code="""
#include<stdio.h>\n
void main()
{}
"""
def compile_to_c():
    global c_code
    file=open('p.txt','r').read().split('\n')

    # define var
    if_list=[]

    # define pattern
    # python pattern
    pat_while_py=r'\s*[a-zA-Z0-9_]*\s*[==|>|<|>=|<=|!=]*\s*[a-zA-Z0-9"\']*\s*'
    pat_print_py=r'["|\'][ a-zA-Z0-9_]*["|\']\s*|\s*[(][a-zA-Z0-9_]*[)]'
    pat_for_py=r'\s*[0-9]*\s*,\s*[0-9]*\s*'
    pat_if_py=r'\s*[a-zA-Z0-9_%]*\s*[==|!=|>|<|>=|<=]*\s*[a-zA-Z0-9_\'"]*\s*'
    # pat_if1=r'\s*^if \s*[a-zA-Z0-9_]*\s*[==|!=|>|<|>=|<=]*[0-9a-zA-Z"\']*\s*:'
    pat_op_py=r'[a-zA-Z0-9_]*\s*=\s*[a-zA-Z0-9_]*\s*[+|-|*|/]\s*[0-9a-zA-Z_"\']*\s*'
    # a=1 name='mohsen'
    pat_var_py=r'\s*^[a-zA-Z][a-zA-Z0-9_]*\s*=\s*["0-9a-zA-Z_\'"]*\s*'
    # c pattern
    pat_for_c='for(int i={};i<{};i++)'
    pat_while_c='while ({})'
    pat_print_c='printf({})'
    pat_if_c='if({})'
    pat_else_if_c='else if({})'
    pat_else_c='else'
    print_=''

    # help c
    for_=''
```

مثلا while فقط به متغیر هایی که مقایسه میشوند و عملیات آن توجه میکنیم.

یا مثلا توضیح print در:

```
pat_print_py=r'["|\'][ a-zA-Z0-9_]*["|\']\s*|\s*[(][a-zA-Z0-9_]*[)]'
```

\' به معنای single quotation است و | به معنی یا. جلوی print یا یک متغیر قرار میگیرد یا یک رشته حرفی. پس در وسط یک | داریم. پس در اینجا برای راحتی خود کلمه print را نیاوردیم چون نیاز به آن نداریم. البته در بخش syntax analysis آوردیم.

در اول تابع هم فایل p.txt را میخوانیم و آنرا بر اساس خط split میکنیم و حاصل در متغیر file میریزیم.

توضیح pattern های زبان C:

```
pat_for_c='for(int i={};i<{};i++)'
```

در شکل بالا از formatted string استفاده کردیم. یعنی {} یک متغیر باید در ان قرار بگیرد. که با کمک گرفتن از pattern های زبان پایتون که خودمان در بالا تعریف کردیم متغیر ها را از کد پایتون در میآورد و در ادامه این متغیر ها را توی {} قرار میدهیم.

در حلقه for بعدی هر خط file را بررسی میکنیم. باز با توجه به pattern هایی که تعریف کردیم و تابع re.findall هر خط را با pattern زبان پایتون مقایسه میکنیم و آنگاه با استفاده از تابع format که patter های زبان c را پر میکنیم یعنی درون {} مقادیر را جایگذاری میکنیم. حاصل در_for یا_if و...

ذخیره میشود.

```
    # help c
    for_=''
    if_=''
    elif_=''
    else_=''
    print_=[]

    for f in file[0:8]:
        if 'for' in f:
            x=re.findall(pat_for_py,f,re.MULTILINE)
            x=''.join(x)
            x=x.split(',')
            for_=pat_for_c.format(str(x[0]),str(x[1]))+'\n{}'
            # print(for_)
        if 'if' in f and 'elif' not in f:
            x=re.findall(pat_if_py,f,re.MULTILINE)
            x=''.join(x)
            x=x.split('if')
            if len(x)>1:
                if_=pat_if_c.format(str(x[1]))+'\n'+"{}"
                # print(if_)

        if 'elif' in f:
            x=re.findall(pat_if_py,f,re.MULTILINE)
            x=''.join(x)
            x=x.split('elif')
            # print(x)
            if len(x)>1:
                elif_=pat_else_if_c.format(str(x[1]))+'\n'+"{}"
                # print(elif_)

        if 'else' in f:
            else_='else'+"\n"+"{}"
            # print(else_)

        if  'print' in f :
            x=re.findall(pat_print_py,f,re.MULTILINE)
            x=''.join(x)
            print_.append(pat_print_c.format(str(x))+';')
            # print(print_)

    char=''
    list_=[if_,elif_,else_]
    if len(list_)==len(print_):
        char+=list_[0].format('{'+'\n'+print_[0]+"\n"+"}")+'\n'
        char+=list_[1].format('{'+'\n'+print_[1]+"\n"+"}")+'\n'
        char+=list_[2].format('{'+'\n'+print_[2]+"\n"+"}")+'\n'

    # print(char)

    char=for_.format('{'+'\n'+char+"\n"+"}"+'\n')
    # print(for_)


    while_=''
    pr_=''

    for i,f in enumerate(file[7:]):
        if i==0:
            char+='int '+f+';'+'\n'
        elif i==1:
            char+='int '+f+';'+'\n'
```

متغیر f به هر خط کد اشاره دارد.

```
        if i==0:
            char+='int '+f+';'+'\n'
        elif i==1:
            char+='int '+f+';'+'\n'
        elif i==2:
            x=re.findall(pat_while_py,f,re.MULTILINE)
            x=''.join(x)
            x=x.split(' ')
            # print(x)
            if len(x)>1:
                while_=pat_while_c.format(str(x[1])+str(x[2])+str(x[3]))+"\n"+"{"+"\n"
                char+=while_
        elif i==3:
            char+=f+";"+"\n"
        elif i==4:
            char+=f+";"+"\n"+"}"+'\n'
        elif i==5:
            x=re.findall(pat_print_py,f,re.MULTILINE)
            x=''.join(x)
            if x[0]=='(' and x[-1]==')':
                x=x[1:-1]
            pr_=pat_print_c.format(str(x))+';'
            char+=pr_




        # print(char)
    c_code=c_code.format("{"+"\n"+char+"\n"+"}")
    # print(c_code)

    file2=open('c.txt','w')
    file2.write(c_code)
```

```
        if 'if' in f and 'elif' not in f:
```

در این خط چون کلمه if در elif هم وجود دارد با and آنرا handle کردیم.

```
            for_=pat_for_c.format(str(x[0]),str(x[1]))+'\n{}'
```

متغیر x یک لیست است چون از تابع join استفاده کردیم. (مثل فاز syntax است)

بخش دوم فایل p.txt هم به همین صورت ترجمه میشود. در نهایت حاصل ترجمه را در char میریزیم.

و سپس حاصل نهایی با تابع format در متغیر c_code قرار میگیرد و یک فایل c.txt درست میکنیم و با تابع write c_code را در آن میریزیم. این کد c قابل اجراست و کامپایل میشود.