Ahsanullah University of Science and Technology

Department of Computer Science and Engineering

CSE 4130: Formal Languages and Compilers Lab

Fall 2020

Project Report

A Simple Compiler Based on Scanning and Filtering, Lexical Analysis, Symbol Table Construction and Management, Detecting Simple Syntax Errors in addition to Use of CFGs for Parsing

Lab Section: B1

Submitted by

Alam Khan 170204084

Submitted To

Mr. Md. Aminur Rahman
Assistant Professor
Department of CSE, AUST

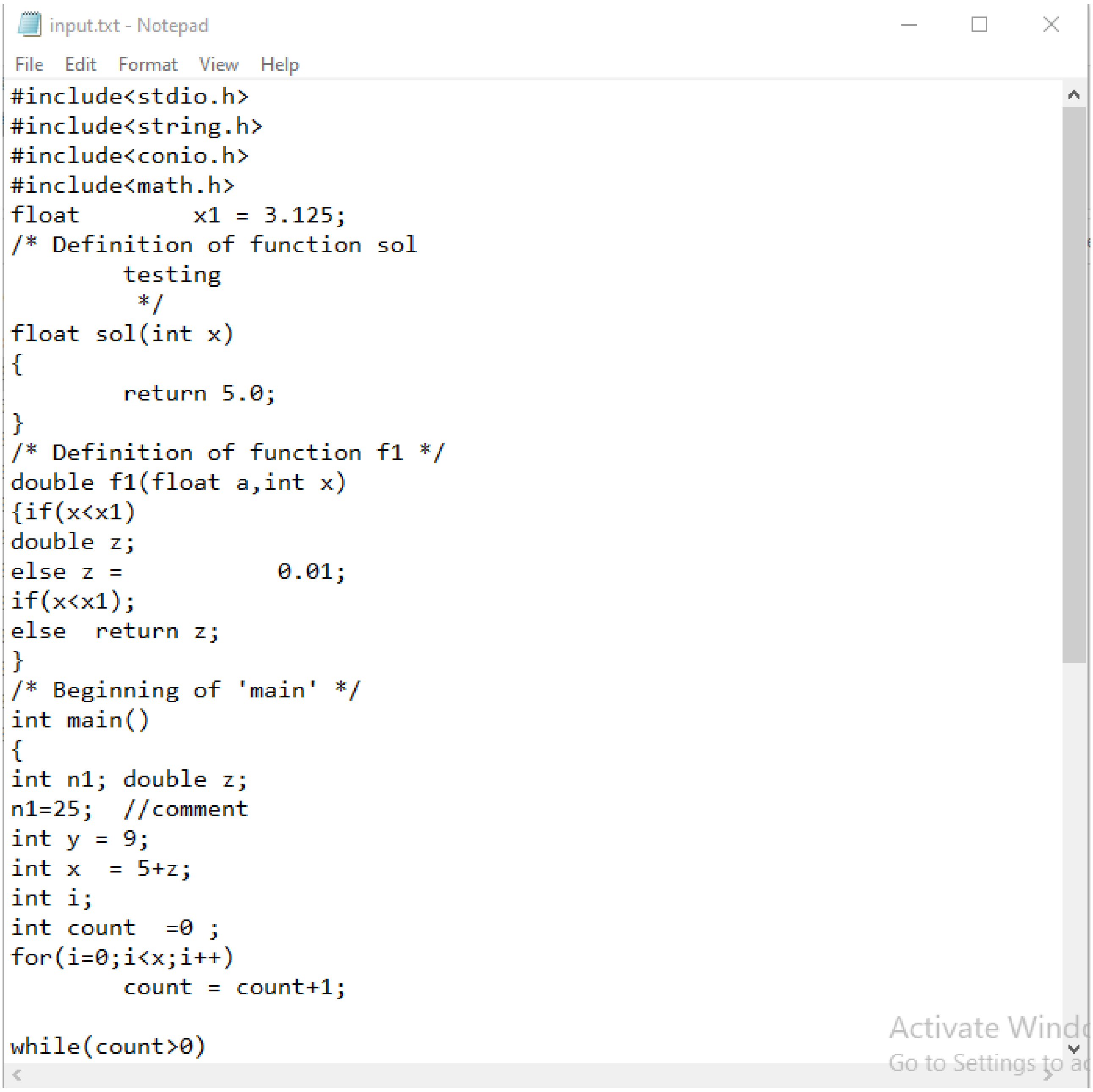
Date of Submission: 11 October, 2021

Contents

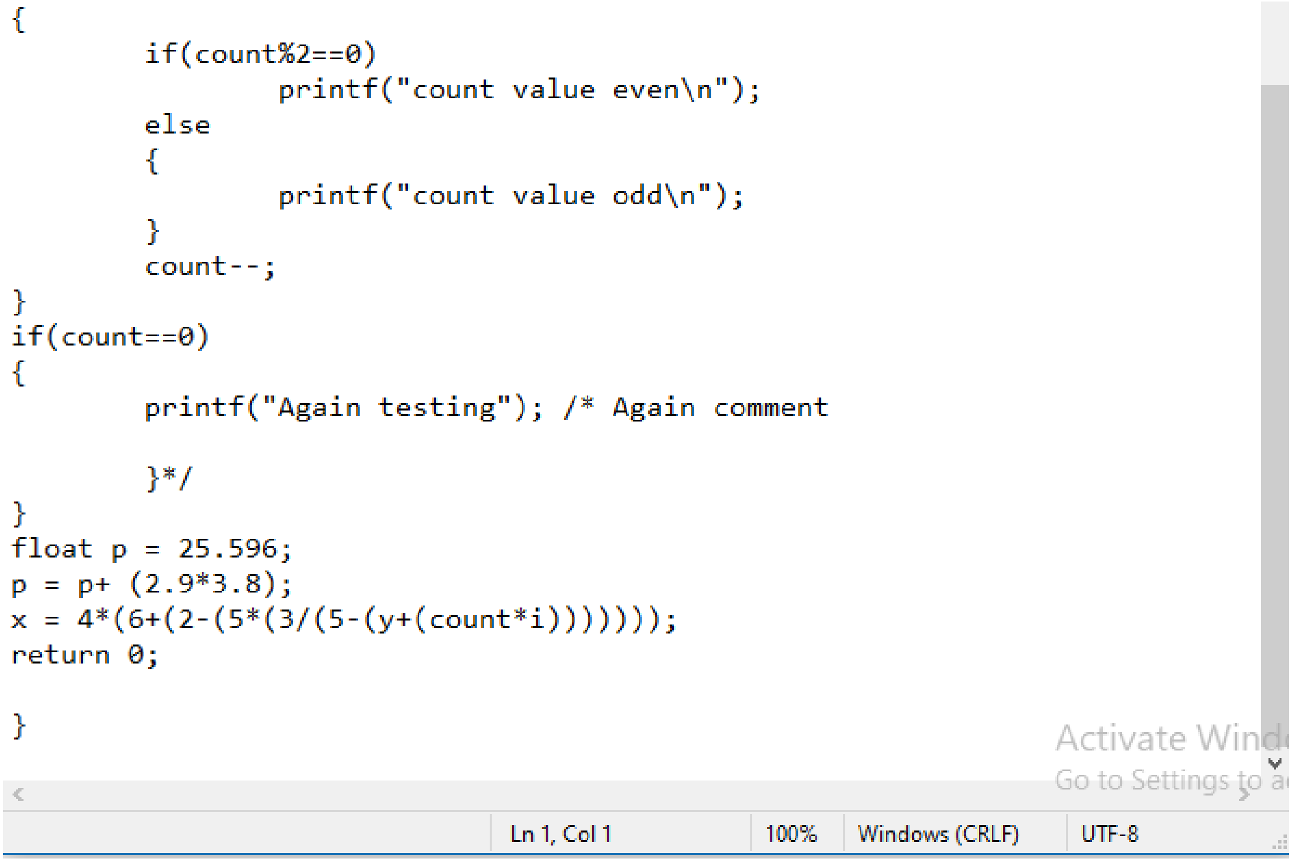
1 Introduction

A compiler is a special program that processes statements written in a particular programming language and turns them into machine language or "code" that a computer's processor uses. Typically, a programmer writes language statements in a language such as Pascal or C one line at a time using an editor.

2 Step 0: Input Step



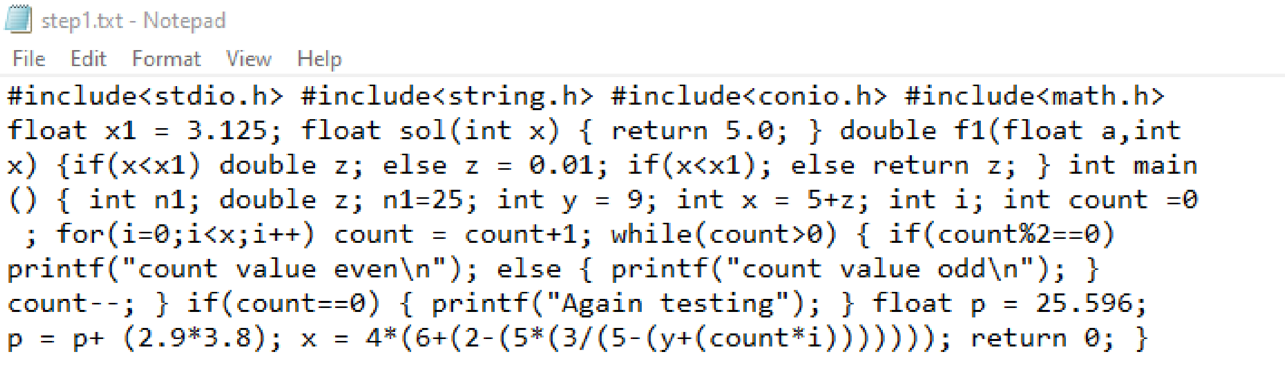
```
#include<stdio.h>
#include<string.h>
#include<conio.h>
#include<math.h>
float        x1 = 3.125;
/* Definition of function sol
        testing
         */
float sol(int x)
{
        return 5.0;
}
/* Definition of function f1 */
double f1(float a,int x)
{if(x<x1)
double z;
else z =           0.01;
if(x<x1);
else  return z;
}
/* Beginning of 'main' */
int main()
{
int n1; double z;
n1=25;  //comment
int y = 9;
int x  = 5+z;
int i;
int count  =0 ;
for(i=0;i<x;i++)
        count = count+1;

while(count>0)
```



```
{
        if(count%2==0)
                printf("count value even\n");
        else
        {
                printf("count value odd\n");
        }
        count--;
}
if(count==0)
{
        printf("Again testing"); /* Again comment

        }*/
}
float p = 25.596;
p = p+ (2.9*3.8);
x = 4*(6+(2-(5*(3/(5-(y+(count*i)))))));
return 0;

}
```

Figure 1: Step 0 Input Step

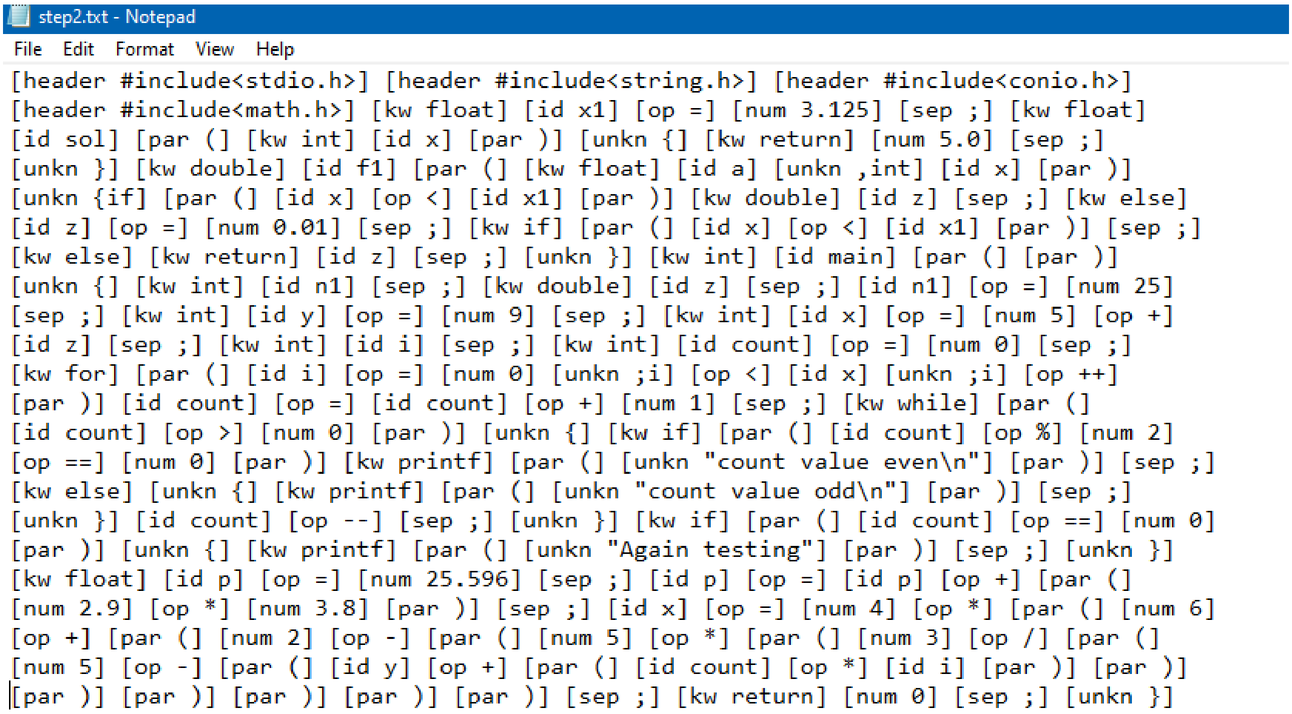
3 Step 1 Scanning and Filtering a Source Program



```
#include<stdio.h> #include<string.h> #include<conio.h> #include<math.h>
float x1 = 3.125; float sol(int x) { return 5.0; } double f1(float a,int
x) {if(x<x1) double z; else z = 0.01; if(x<x1); else return z; } int main
() { int n1; double z; n1=25; int y = 9; int x = 5+z; int i; int count =0
 ; for(i=0;i<x;i++) count = count+1; while(count>0) { if(count%2==0)
printf("count value even\n"); else { printf("count value odd\n"); }
count--; } if(count==0) { printf("Again testing"); } float p = 25.596;
p = p+ (2.9*3.8); x = 4*(6+(2-(5*(3/(5-(y+(count*i))))))); return 0; }
```

Figure 2: Step 1 Scanning and Filtering a Source Program

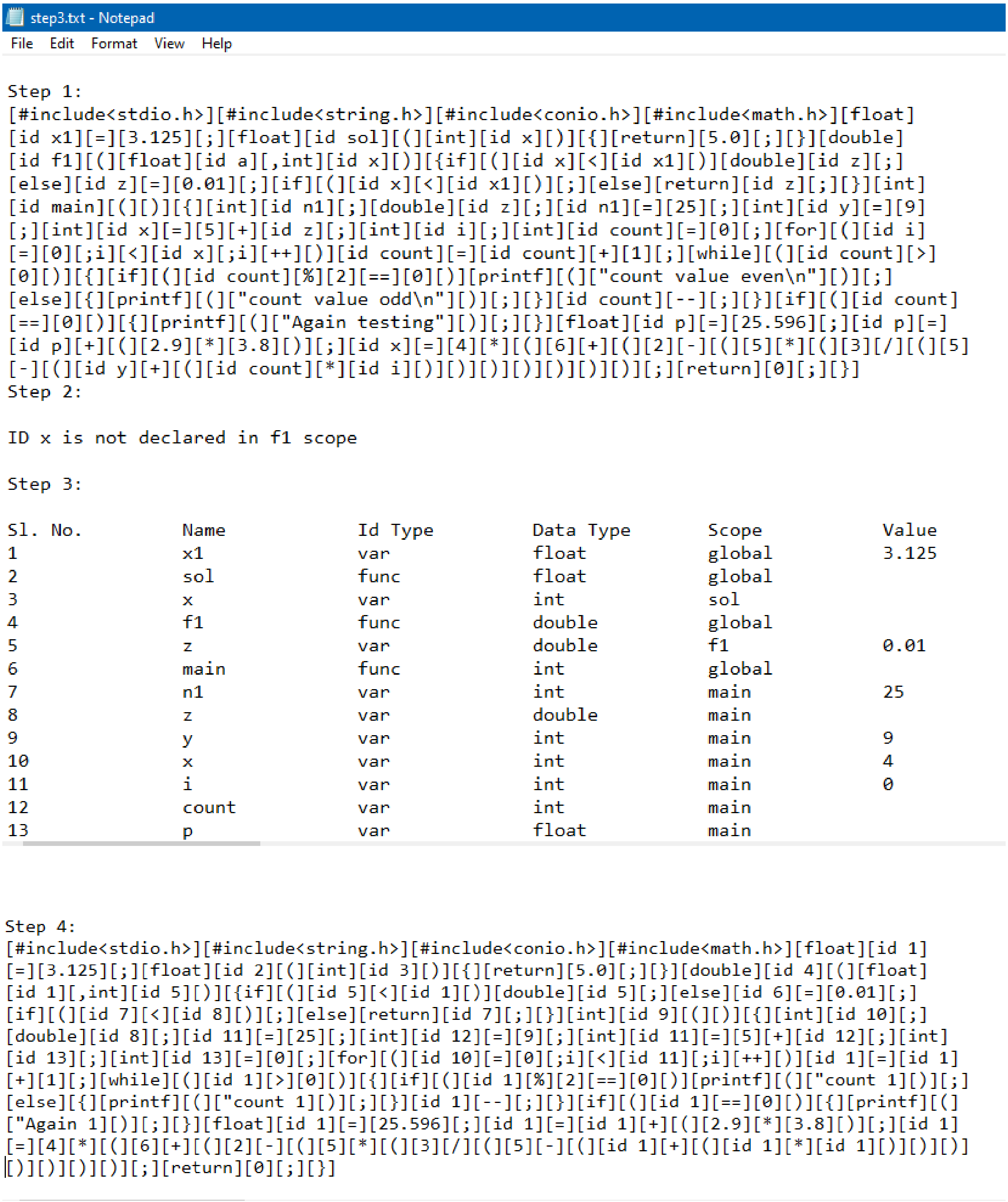
4 Step 2: Lexical Analysis



```
[header #include<stdio.h>] [header #include<string.h>] [header #include<conio.h>]
[header #include<math.h>] [kw float] [id x1] [op =] [num 3.125] [sep ;] [kw float]
[id sol] [par (] [kw int] [id x] [par )] [unkn {] [kw return] [num 5.0] [sep ;]
[unkn }] [kw double] [id f1] [par (] [kw float] [id a] [unkn ,int] [id x] [par )]
[unkn {if] [par (] [id x] [op <] [id x1] [par )] [kw double] [id z] [sep ;] [kw else]
[id z] [op =] [num 0.01] [sep ;] [kw if] [par (] [id x] [op <] [id x1] [par )] [sep ;]
[kw else] [kw return] [id z] [sep ;] [unkn }] [kw int] [id main] [par (] [par )]
[unkn {] [kw int] [id n1] [sep ;] [kw double] [id z] [sep ;] [id n1] [op =] [num 25]
[sep ;] [kw int] [id y] [op =] [num 9] [sep ;] [kw int] [id x] [op =] [num 5] [op +]
[id z] [sep ;] [kw int] [id i] [sep ;] [kw int] [id count] [op =] [num 0] [sep ;]
[kw for] [par (] [id i] [op =] [num 0] [unkn ;i] [op <] [id x] [unkn ;i] [op ++]
[par )] [id count] [op =] [id count] [op +] [num 1] [sep ;] [kw while] [par (]
[id count] [op >] [num 0] [par )] [unkn {] [kw if] [par (] [id count] [op %] [num 2]
[op ==] [num 0] [par )] [kw printf] [par (] [unkn "count value even\n"] [par )] [sep ;]
[kw else] [unkn {] [kw printf] [par (] [unkn "count value odd\n"] [par )] [sep ;]
[unkn }] [id count] [op --] [sep ;] [unkn }] [kw if] [par (] [id count] [op ==] [num 0]
[par )] [unkn {] [kw printf] [par (] [unkn "Again testing"] [par )] [sep ;] [unkn }]
[kw float] [id p] [op =] [num 25.596] [sep ;] [id p] [op =] [id p] [op +] [par (]
[num 2.9] [op *] [num 3.8] [par )] [sep ;] [id x] [op =] [num 4] [op *] [par (] [num 6]
[op +] [par (] [num 2] [op -] [par (] [num 5] [op *] [par (] [num 3] [op /] [par (]
[num 5] [op -] [par (] [id y] [op +] [par (] [id count] [op *] [id i] [par )] [par )]
[par )] [par )] [par )] [par )] [par )] [sep ;] [kw return] [num 0] [sep ;] [unkn }]
```

Figure 3: Step 2 Lexical Analysis

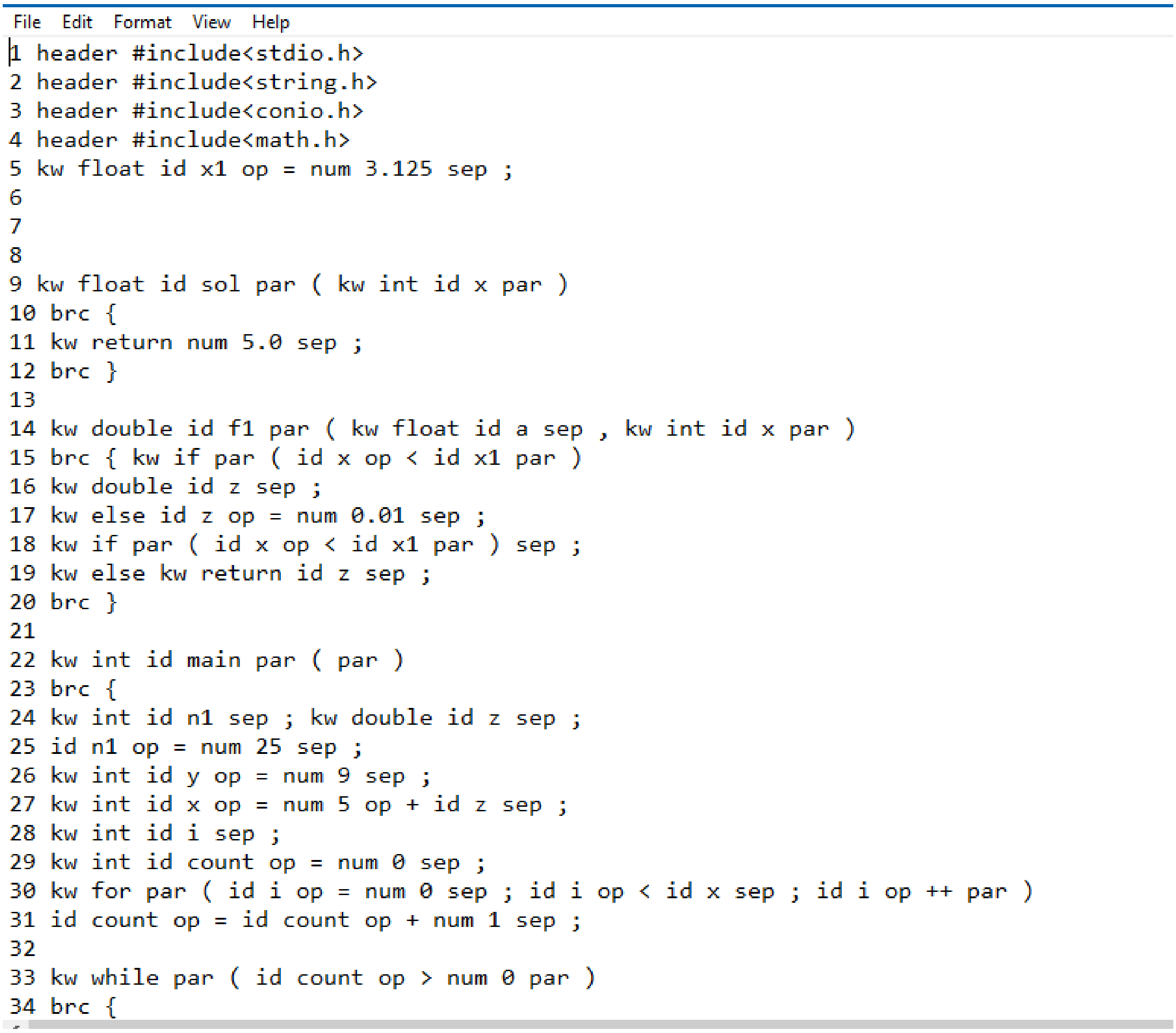
5 Step 3: Symbol Table Construction and Management



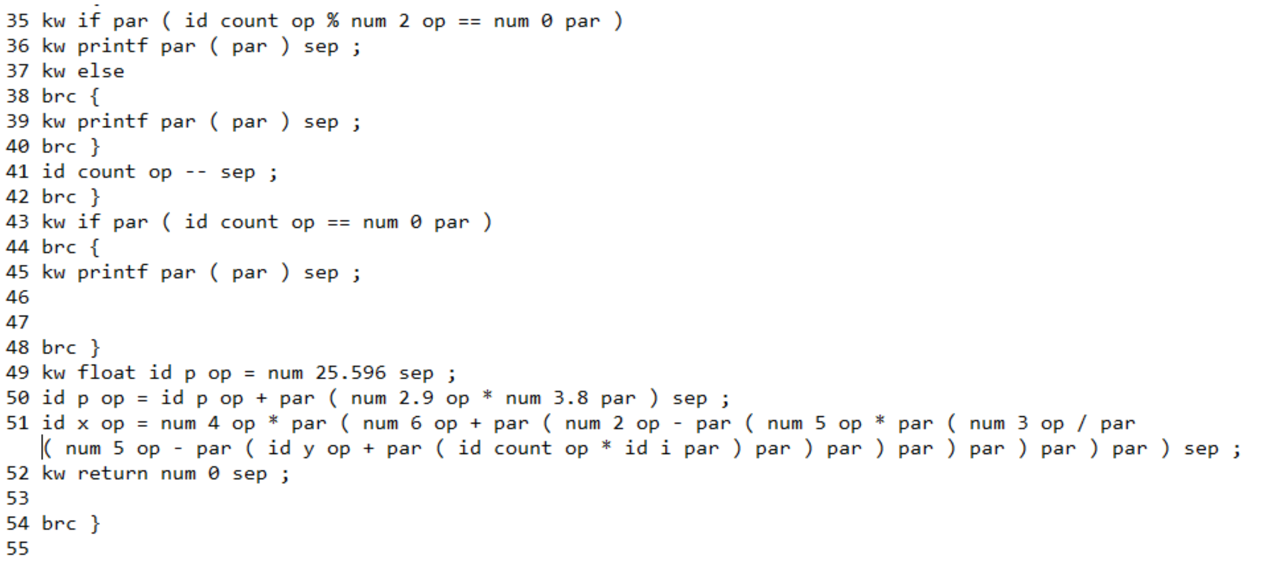
```
Step 1:
[#include<stdio.h>][#include<string.h>][#include<conio.h>][#include<math.h>][float]
[id x1][=][3.125][;][float][id sol][(][int][id x][)][{][return][5.0][;][}][double]
[id f1][(][float][id a][,int][id x][)][{if][(][id x][<][id x1][)][double][id z][;]
[else][id z][=][0.01][;][if][(][id x][<][id x1][)][;][else][return][id z][;][}][int]
[id main][(][)][{][int][id n1][;][double][id z][;][id n1][=][25][;][int][id y][=][9]
[;][int][id x][=][5][+][id z][;][int][id i][;][int][id count][=][0][;][for][(][id i]
[=][0][;i][<][id x][;i][++][)][id count][=][id count][+][1][;][while][(][id count][>]
[0][)][{][if][(][id count][%][2][==][0][)][printf][(]["count value even\n"][)][;]
[else][{][printf][(]["count value odd\n"][)][;][}][id count][--][;][}][if][(][id count]
[==][0][)][{][printf][(]["Again testing"][)][;][}][float][id p][=][25.596][;][id p][=]
[id p][+][(][2.9][*][3.8][)][;][id x][=][4][*][(][6][+][(][2][-][(][5][*][(][3][/][(][5]
[-][(][id y][+][(][id count][*][id i][)][)][)][)][)][)][)][;][return][0][;][}]
Step 2:

ID x is not declared in f1 scope

Step 3:

Sl. No.         Name            Id Type         Data Type       Scope           Value
1               x1              var             float           global          3.125
2               sol             func            float           global
3               x               var             int             sol
4               f1              func            double          global
5               z               var             double          f1              0.01
6               main            func            int             global
7               n1              var             int             main            25
8               z               var             double          main
9               y               var             int             main            9
10              x               var             int             main            4
11              i               var             int             main            0
12              count           var             int             main
13              p               var             float           main


Step 4:
[#include<stdio.h>][#include<string.h>][#include<conio.h>][#include<math.h>][float][id 1]
[=][3.125][;][float][id 2][(][int][id 3][)][{][return][5.0][;][}][double][id 4][(][float]
[id 1][,int][id 5][)][{if][(][id 5][<][id 1][)][double][id 5][;][else][id 6][=][0.01][;]
[if][(][id 7][<][id 8][)][;][else][return][id 7][;][}][int][id 9][(][)][{][int][id 10][;]
[double][id 8][;][id 11][=][25][;][int][id 12][=][9][;][int][id 11][=][5][+][id 12][;][int]
[id 13][;][int][id 13][=][0][;][for][(][id 10][=][0][;i][<][id 11][;i][++][)][id 1][=][id 1]
[+][1][;][while][(][id 1][>][0][)][{][if][(][id 1][%][2][==][0][)][printf][(]["count 1][)][;]
[else][{][printf][(]["count 1][)][;][}][id 1][--][;][}][if][(][id 1][==][0][)][{][printf][(]
["Again 1][)][;][}][float][id 1][=][25.596][;][id 1][=][id 1][+][(][2.9][*][3.8][)][;][id 1]
[=][4][*][(][6][+][(][2][-][(][5][*][(][3][/][(][5][-][(][id 1][+][(][id 1][*][id 1][)][)][)]
[)][)][)][)][;][return][0][;][}]
```

Figure 4: Step 3 Symbol Table Construction and Management

6 Step 4: Detecting Simple Syntax Errors



```
1 header #include<stdio.h>
2 header #include<string.h>
3 header #include<conio.h>
4 header #include<math.h>
5 kw float id x1 op = num 3.125 sep ;
6
7
8
9 kw float id sol par ( kw int id x par )
10 brc {
11 kw return num 5.0 sep ;
12 brc }
13
14 kw double id f1 par ( kw float id a sep , kw int id x par )
15 brc { kw if par ( id x op < id x1 par )
16 kw double id z sep ;
17 kw else id z op = num 0.01 sep ;
18 kw if par ( id x op < id x1 par ) sep ;
19 kw else kw return id z sep ;
20 brc }
21
22 kw int id main par ( par )
23 brc {
24 kw int id n1 sep ; kw double id z sep ;
25 id n1 op = num 25 sep ;
26 kw int id y op = num 9 sep ;
27 kw int id x op = num 5 op + id z sep ;
28 kw int id i sep ;
29 kw int id count op = num 0 sep ;
30 kw for par ( id i op = num 0 sep ; id i op < id x sep ; id i op ++ par )
31 id count op = id count op + num 1 sep ;
32
33 kw while par ( id count op > num 0 par )
34 brc {
```



```
35 kw if par ( id count op % num 2 op == num 0 par )
36 kw printf par ( par ) sep ;
37 kw else
38 brc {
39 kw printf par ( par ) sep ;
40 brc }
41 id count op -- sep ;
42 brc }
43 kw if par ( id count op == num 0 par )
44 brc {
45 kw printf par ( par ) sep ;
46
47
48 brc }
49 kw float id p op = num 25.596 sep ;
50 id p op = id p op + par ( num 2.9 op * num 3.8 par ) sep ;
51 id x op = num 4 op * par ( num 6 op + par ( num 2 op - par ( num 5 op * par ( num 3 op / par
   ( num 5 op - par ( id y op + par ( id count op * id i par ) par ) par ) par ) par ) par ) par ) sep ;
52 kw return num 0 sep ;
53
54 brc }
55
```

Figure 5: Step 4 Adding Line Numbers and Removing Comments

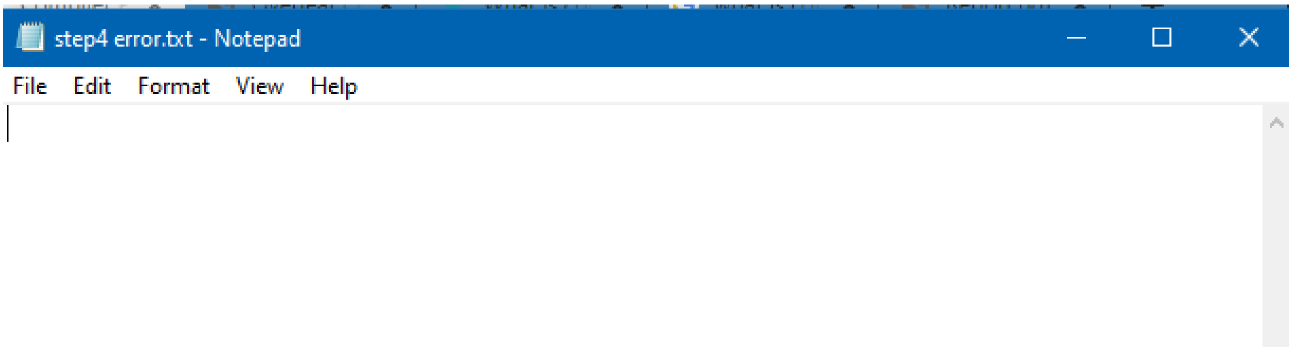
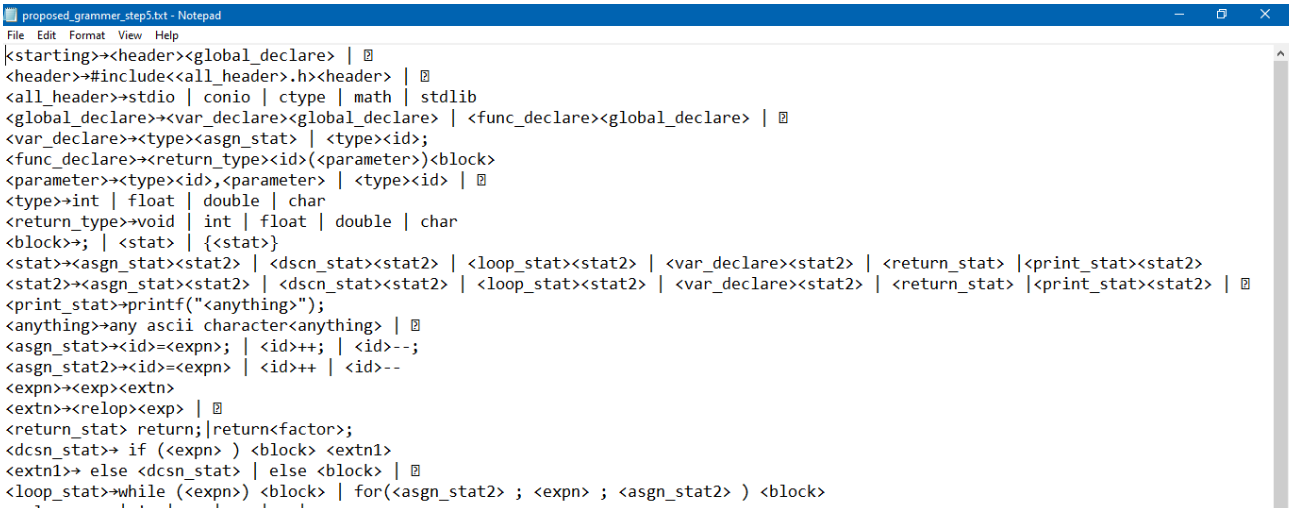
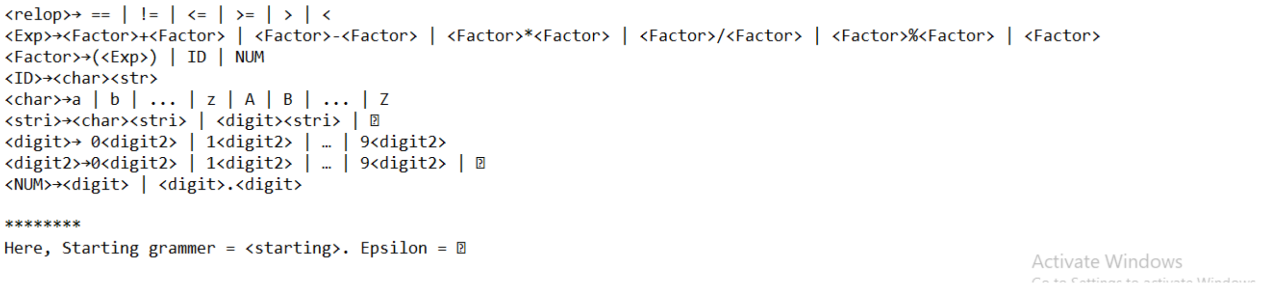


Figure 6: Step 4 Showing Error

7 Step 5: Use of CFGs for Parsing

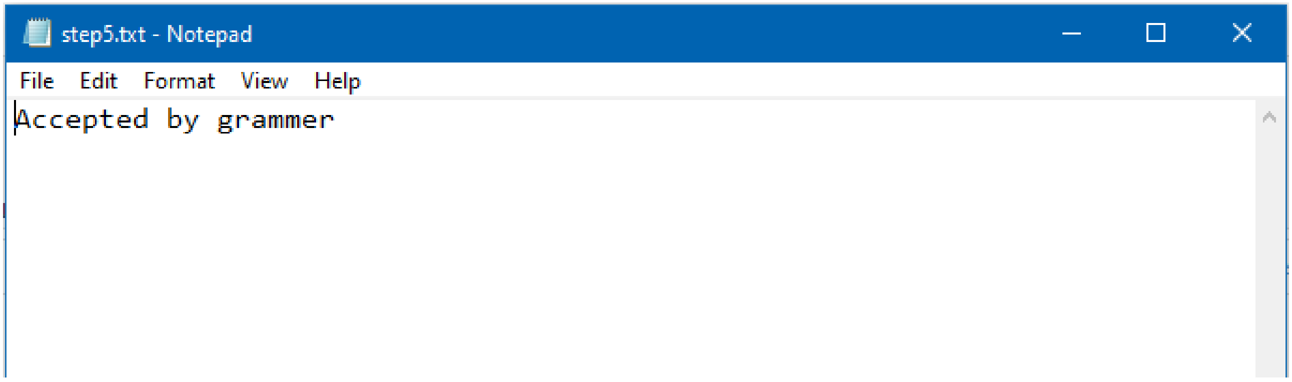


```
<starting>→<header><global_declare> | ⍰
<header>→#include<<all_header>.h><header> | ⍰
<all_header>→stdio | conio | ctype | math | stdlib
<global_declare>→<var_declare><global_declare> | <func_declare><global_declare> | ⍰
<var_declare>→<type><asgn_stat> | <type><id>;
<func_declare>→<return_type><id>(<parameter>)<block>
<parameter>→<type><id>,<parameter> | <type><id> | ⍰
<type>→int | float | double | char
<return_type>→void | int | float | double | char
<block>→; | <stat> | {<stat>}
<stat>→<asgn_stat><stat2> | <dscn_stat><stat2> | <loop_stat><stat2> | <var_declare><stat2> | <return_stat> |<print_stat><stat2>
<stat2>→<asgn_stat><stat2> | <dscn_stat><stat2> | <loop_stat><stat2> | <var_declare><stat2> | <return_stat> |<print_stat><stat2> | ⍰
<print_stat>→printf("<anything>");
<anything>→any ascii character<anything> | ⍰
<asgn_stat>→<id>=<expn>; | <id>++; | <id>--;
<asgn_stat2>→<id>=<expn> | <id>++ | <id>--
<expn>→<exp><extn>
<extn>→<relop><exp> | ⍰
<return_stat> return;|return<factor>;
<dcsn_stat>→ if (<expn> ) <block> <extn1>
<extn1>→ else <dcsn_stat> | else <block> | ⍰
<loop_stat>→while (<expn>) <block> | for(<asgn_stat2> ; <expn> ; <asgn_stat2> ) <block>
```



```
<relop>→ == | != | <= | >= | > | <
<Exp>→<Factor>+<Factor> | <Factor>-<Factor> | <Factor>*<Factor> | <Factor>/<Factor> | <Factor>%<Factor> | <Factor>
<Factor>→(<Exp>) | ID | NUM
<ID>→<char><str>
<char>→a | b | ... | z | A | B | ... | Z
<stri>→<char><stri> | <digit><stri> | ⍰
<digit>→ 0<digit2> | 1<digit2> | … | 9<digit2>
<digit2>→0<digit2> | 1<digit2> | … | 9<digit2> | ⍰
<NUM>→<digit> | <digit>.<digit>

********
Here, Starting grammer = <starting>. Epsilon = ⍰
```

Figure 7: Step 5 The Ultimate Grammar



```
Accepted by grammer
```

Figure 8: Output of Step 5