**APPENDIX FOR THE PAPER TITLED AS**

**"A New Cohesion Metric and Restructuring Technique for Object Oriented Paradigm"**

by (author names removed)

## 1. First Example Class – Class1

### a. Original Class to be Restructured

```
1   class Class1
2   {
3   private:
4     int stk[100];
5     int top;
6     string funinvokes[100];
7     int topInvok;
8     time_t rawtime;
9     int x1, y1, x2, y2;
10    void ErrorInSizeFunInvok()
11    {
12      cout<<"Index out of range!\n";
13      cout<<"The Array has "<<topInvok
14          <<" elements.\n";
15    }
16    void ErrorInSize()
17    {
18      cout<<"Index out of range!\n";
19      cout<<"The Array has "<<top
20          <<" elements.\n";
21    }
22  public:
23    Class1(int left=0,int up=0,
24           int right=0,int bottom=0)
25    {
26      topInvok=0;
27      time ( &rawtime );
28      string t=string(ctime(&rawtime));
29      string temp="Class1 invoked: ";
30      temp+=t;
31      PushFunInvok(temp);
32      top=0;
33      x1=left;
34      y1=up;
35      x2=right;
36      y2=bottom;
37    }
38    ~Class1() {}
39    int Height()
40    {
41      time ( &rawtime );
42      string t=string(ctime(&rawtime));
43      string temp="Height invoked: ";
44      temp+=t;
45      PushFunInvok(temp);
46      return (y2-y1);
47    }
48    int Width()
49    {
50      time ( &rawtime );
51      string t=string(ctime(&rawtime));
52      string temp="Width invoked: ";
53      temp+=t;
54      PushFunInvok(temp);
55      return (x2-x1);
56    }
57    int Area()
58    {
59      time ( &rawtime );
60      string t=string(ctime(&rawtime));
61      string temp="Area invoked: ";
62      temp+=t;
63      PushFunInvok(temp);
64      int w=x2-x1;
65      int h=y2-y1;
66      int a=w*h;
67      return a;
68    }
69    int Perimeter()
70    {
71      time ( &rawtime );
72      string t=string(ctime(&rawtime));
73      string temp="Perimeter invoked:";
74      temp+=t;
75      PushFunInvok(temp);
76      int w=x2-x1;
77      int h=y2-y1;
78      return 2*w+2*h;
79    }
80    void Clear()
81    {
82      time ( &rawtime );
83      string t=string(ctime(&rawtime));
84      string temp="Clear invoked: ";
85      temp+=t;
86      PushFunInvok(temp);
87      top=0;
88    }
89    void printAllInvoks()
90    {
91      for(int i=0; i<topInvok; i++)
92      {
93        string temp=funinvokes[i];
94        cout<<temp;
```

```
95      }
96    }
97    void PushFunInvok(std::string str)
98    {
99      if (topInvok < 100)
100     {
101        funinvokes[topInvok]=str;
102        topInvok++;
103     }
104     else
105        ErrorInSizeFunInvok();
106   }
107   void Push(int i)
108   {
109     time ( &rawtime );
110     string t=string(ctime(&rawtime));
111     string temp="Push invoked: ";
112     temp+=t;
113     PushFunInvok(temp);
114     if (top < 100)
115     {
116        stk[top]=i;
117        top++;
118     }
119     else
120        ErrorInSize();
121   }
122   int Pop()
123   {
124     time ( &rawtime );
125     string t=string(ctime(&rawtime));
126     string temp="Pop invoked: ";
127     temp+=t;
128     PushFunInvok(temp);
129     if (top > 0)
130     {
131        top--;
132        int temp_int=stk[top];
133        return temp_int;
134     }
135     else
136     {
137        ErrorInSize();
138        return -1;
139     }
140   }
141   int Size()
142   {
143     time ( &rawtime );
144     string t=string(ctime(&rawtime));
145     string temp="Size invoked: ";
146     temp+=t;
147     PushFunInvok(temp);
148     return top;
149   }
150 };
```

b. **Restructuring Process**

$DM_{Class1}$ = {stk, top, funinvokes, topInvok, rawtime, x1, y1, x2, y2}

stk

$ST_{stkxClass1}$ ={116, 132}
$SL_{116xClass1}$ ={114, 116, 117, 120, 18, 19}
$SL_{132xClass1}$ ={129, 131, 132, 133, 137, 138, 18, 19}
$SL_{stkxClass1}$ = $SL_{116xClass1}$ U $SL_{132xClass1}$
={114,116,117,120,18,19,129, 131,132,133,137,138}

top

$ST_{topxClass1}$ ={19,32,87,114,116,117,129, 131,132,148}
$SL_{19xClass1}$ ={19}
$SL_{32xClass1}$ ={32}
$SL_{87xClass1}$ ={87}
$SL_{114xClass1}$ ={114,116,117,120,18,19}
$SL_{116xClass1}$ ={114,116,117,120,18,19}
$SL_{117xClass1}$ ={114,116,117,120,18,19}
$SL_{129xClass1}$ ={129,131,132,133,137,18,19,138}
$SL_{131xClass1}$ ={129,131,132,133,137,18,19,138}
$SL_{132xClass1}$ ={129,131,132,133,137,18,19,138}
$SL_{148xClass1}$ ={148}
$SL_{topxClass1}$ = $\bigcup_{s \in ST topxClass1} SLsxClass1$

= {19,32,87,114,116,117,120,18, 129,131,132,133,137,138,148}

funinvok

$ST_{funinvokesxClass1}$ ={93, 101}
$SL_{93xClass1}$ ={91,93,94}
$SL_{101xClass1}$ ={99,101,102,105,12,13}
$SL_{funinvokxClass1}$ = {91,93,94,99,101,102,105,12,13}

topInvok

$ST_{topInvokxClass1}$ ={13,26,91,99,101,102}
$SL_{13xClass1}$ ={13}
$SL_{26xClass1}$ ={26}
$SL_{91xClass1}$ ={91,93,94}
$SL_{99xClass1}$ ={99,101,102,105,12,13}
$SL_{101xClass1}$ ={99,101,102,105,12,13}
$SL_{102xClass1}$ ={99,101,102,105,12,13}
$SL_{topInvokxClass1}$ = {13,26,91,93,94, 99,101,102,105,12}

rawtime

$ST_{rawtimexClass1}$ ={27,28,41,42,50,51,59,60,71,72, 82,83,109,110,124,125,143,144}
$SL_{27xClass1}$ ={27}
$SL_{28xClass1}$ ={28,29,30,31,101,102,99,105,12,13}
$SL_{41xClass1}$ ={41}
$SL_{42xClass1}$ ={42,43,44,45,101,102,99,105,12,13}
$SL_{50xClass1}$ ={50}
$SL_{51xClass1}$ ={51,52,53,54,101,102,99,105,12,13}
$SL_{59xClass1}$ ={59}
$SL_{60xClass1}$ ={60,61,62,63,101,102,99,105,12,13}
$SL_{71xClass1}$ ={71}
$SL_{72xClass1}$ ={72,73,74,75,101,102,99,105,12,13}
$SL_{82xClass1}$ ={82}
$SL_{83xClass1}$ ={83,84,85,86,101,102,99,105,12,13}
$SL_{109xClass1}$ ={109}
$SL_{110xClass1}$ ={110,111,112,113,101, 102,99,105,12,13}
$SL_{124xClass1}$ ={124}
$SL_{125xClass1}$ ={125,126,127,128,101, 102,99,105,12,13}
$SL_{143xClass1}$ ={143}
$SL_{144xClass1}$ ={144,145,146,147,101, 102,99,105,12,13}
$SL_{rawtimexClass1}$ ={27,28,29,30,31, 41,42,43,44,45,50, 51,52,53,54,59, 60,61,62,63,71,72, 73,74,75,82, 83,84,85,86, 109, 110, 111,112,113, 124,125,126,127,128, 143, 144,145,146,147, 101,102,99, 105,12,13}

x1

$ST_{x1xClass1}$ ={33,55,64,76}
$SL_{33xClass1}$ ={33}
$SL_{55xClass1}$ ={55}
$SL_{64xClass1}$ ={64,65,66,67}
$SL_{76xClass1}$ ={76,77,78}
$SL_{x1xClass1}$ =={33,55,64,65,66,67,76,77,78}

x2

$ST_{x2xClass1}$ ={35,55,64,76}
$SL_{35xClass1}$ ={35}
$SL_{55xClass1}$ ={55}
$SL_{64xClass1}$ ={64,65,66,67}
$SL_{76xClass1}$ ={76,77,78}
$SL_{x2xClass1}$ =={35,55,64,65,66,67,76,77,78}

y1

$ST_{y1xClass1}$ ={34,46,65,77}
$SL_{34xClass1}$ ={34}
$SL_{46xClass1}$ ={46}
$SL_{65xClass1}$ ={64,65,66,67}
$SL_{77xClass1}$ ={76,77,78}
$SL_{y1xClass1}$ =={34,46,64,65,66,67,76,77,78}

| y2 |
|---|
| $ST_{y2xClass1}$ ={36,46,65,77} |
| $SL_{36xClass1}$ ={36} |
| $SL_{46xClass1}$ ={46} |
| $SL_{65xClass1}$ ={64,65,66,67} |
| $SL_{77xClass1}$ ={76,77,78} |
| $SL_{y2xClass1}$ =={36,46,64,65,66,67,76,77,78} |

$SL_{stkxClass1}$ ={114,116,117,120,18,19,129,131,132,133,137,138}
$SL_{topxClass1}$={19,32,87,114,116,117,120,18,129,131,132,133,137,138,148}
$SL_{funinvokxClass1}$ ={91,93,94,99,101,102,105,12,13}
$SL_{topInvokxClass1}$ ={13,26,91,93,94,99,101,102,105,12}
$SL_{rawtimexClass1}$ ={27,28,29,30,31, 41,42,43,44,45, 50,51,52,53,54,59,60,61,62,63,71,72,73,74,75,82,83,84, 85,86,109, 110,111,112,113, 124,125,126,127,128,143, 144,145,146,147,101,102,99,105, 12,13}
$SL_{x1xClass1}$ ={33,55,64,65,66,67,76,77,78}
$SL_{x2xClass1}$ ={35,55,64,65,66,67,76,77,78}
$SL_{y1xClass1}$ ={34,46,64,65,66,67,76,77,78}
$SL_{y2xClass1}$ ={36,46,64,65,66,67,76,77,78}

| ∩ | stk | top | funinvok | topInvok | rawtime | x1 | y1 | x2 | y2 |
|---|---|---|---|---|---|---|---|---|---|
| stk | ⊃ | ⊃ | Ø | Ø | Ø | Ø | Ø | Ø | Ø |
| top | ⊃ | ⊃ | Ø | Ø | Ø | Ø | Ø | Ø | Ø |
| funinvok | Ø | Ø | ⊃ | ⊃ | ⊃ | Ø | Ø | Ø | Ø |
| topInvok | Ø | Ø | ⊃ | ⊃ | ⊃ | Ø | Ø | Ø | Ø |
| rawtime | Ø | Ø | ⊃ | ⊃ | ⊃ | Ø | Ø | Ø | Ø |
| x1 | Ø | Ø | Ø | Ø | Ø | ⊃ | ⊃ | ⊃ | ⊃ |
| y1 | Ø | Ø | Ø | Ø | Ø | ⊃ | ⊃ | ⊃ | ⊃ |
| x2 | Ø | Ø | Ø | Ø | Ø | ⊃ | ⊃ | ⊃ | ⊃ |
| y2 | Ø | Ø | Ø | Ø | Ø | ⊃ | ⊃ | ⊃ | ⊃ |

This table shows the intersections of slices of pairs of data members. Ø means the intersection is the empty set and ∩ means there are some elements in the intersection of the slices of two data members. From this table, we generated the DSG in Figure 8 of our paper.

**c. Restructured Version of The Code and**

Restructured Original Class - Class1

```
1   class Class1
2   {
3    private:
4     New1* n1;
5     New2* n2;
6     New3* n3;
7     void ErrorInSizeFunInvok()
8     {
9          n2->fun2_1();
10    }
11    void ErrorInSize()
12    {
13         n1->fun1_1();
14    }
15   public:
16    Class1(int left=0,int up=0,
17           int right=0,int bottom=0)
18    {
19     n1=new New1();
20     n2=new New2();
21     n3=new New3(left,up,right,bottom);
22    }
23    ~Class1() {}
24    int Height()
25    {
26      n2->fun2_2();
27      return n3->fun3_1();
28    }
29    int Width()
30    {
31      n2->fun2_3();
32      return n3->fun3_2();
33    }
34    int Area()
35    {
36      n2->fun2_4();
37      return n3->fun3_3();
38    }
39    int Perimeter()
40    {
41      n2->fun2_5();
42      return n3->fun3_4();
43    }
44    void Clear()
45    {
46      n2->fun2_6();
47      n1->fun1_2();
48    }
49    void printAllInvoks()
50    {
51      n2->fun2_7();
52    }
53    void PushFunInvok(std::string str)
54    {
55      n2->fun2_8(str);
56    }
57    void Push(int i)
58    {
59      n2->fun2_9();
60      n1->fun1_3(i);
61    }
62    int Pop()
63    {
64      n2->fun2_10();
65      return n1->fun1_4();
66    }
67    int Size()
68    {
69      n2->fun2_11();
70      return n1->fun1_5();
71    }
72  };
```

Extracted Class 1

```
1   class New1
2   {
3    private:
4     int stk[100];
5     int top;
6    public:
7     New1()
8     {
9       top=0;
10    }
11    void fun1_1()
12    {
13      cout<<"Index out of range!\n";
14      cout<<"The Array has "<<top
15              <<" elements.\n";
16    }
17    void fun1_2()
18    {
19         top=0;
20    }
21    void fun1_3(int i)
22    {
23         if (top < 100)
24         {
25                stk[top]=i;
26                top++;
27         }
28         else
29                fun1_1();
30    }
31    int fun1_4()
32    {
33      if (top > 0)
34      {
35         top--;
36         int temp_int=stk[top];
```

```
37          return temp_int;
38       }
39       else
40       {
41          fun1_1();
42          return -1;
43       }
44    }
45    int fun1_5()
46    {
47      return top;
48    }
49 };
```

Extracted Class 2

```
1  class New2
2  {
3    private:
4      string funinvokes[100];
5      int topInvok;
6      time_t rawtime;
7    public:
8    New2()
9    {
10     topInvok=0;
11     time ( &rawtime );
12     string t=string(ctime(&rawtime));
13     string temp="Class1 invoked: ";
14     temp+=t;
15     fun2_8(temp);
16   }
17   void fun2_1()
18   {
19      cout<<"Index out of range!\n";
20      cout<<"The Array has "<<topInvok
21          <<" elements.\n";
22   }
23   void fun2_2()
24   {
25     time ( &rawtime );
26     string t=string(ctime(&rawtime));
27     string temp="Height invoked: ";
28     temp+=t;
29     fun2_8(temp);
30   }
31   void fun2_3()
32   {
33     time ( &rawtime );
34     string t=string(ctime(&rawtime));
35     string temp="Width invoked: ";
36     temp+=t;
37     fun2_8(temp);
38   }
39   void fun2_4()
40   {
41     time ( &rawtime );
42     string t=string(ctime(&rawtime));
43     string temp="Area invoked: ";
44     temp+=t;
45     fun2_8(temp);
46   }
47   void fun2_5()
48   {
49     time ( &rawtime );
50     string t=string(ctime(&rawtime));
51     string temp="Perimeter invoked:";
52     temp+=t;
53     fun2_8(temp);
54   }
55   void fun2_6()
56   {
57     time ( &rawtime );
58     string t=string(ctime(&rawtime));
59     string temp="Clear invoked: ";
60     temp+=t;
61     fun2_8(temp);
62     }
63   void fun2_7()
64   {
65     for(int i=0; i<topInvok; i++)
66     {
67        string temp=funinvokes[i];
68        cout<<temp;
69     }
70   }
71   void fun2_8(string str)
72   {
73     if (topInvok < 100)
74     {
75        funinvokes[topInvok]=str;
76        topInvok++;
77     }
78     else
79        fun2_1();
80   }
81   void fun2_9()
82   {
83     time ( &rawtime );
84     string t=string(ctime(&rawtime));
85     string temp="Push invoked: ";
86     temp+=t;
87     fun2_8(temp);
88   }
89   void fun2_10()
90   {
91     time ( &rawtime );
92     string t=string(ctime(&rawtime));
93     string temp="Pop invoked: ";
94     temp+=t;
95     fun2_8(temp);
96   }
97   void fun2_11()
98   {
99     time ( &rawtime );
100    string t=string(ctime(&rawtime));
101    string temp="Size invoked: ";
```

```
102     temp+=t;
103     fun2_8(temp);
104   }
105 };
```

Extracted Class 3

```
1   class New3
2   {
3     private:
4       int x1, y1, x2, y2;
5     public:
6     New3(int left,int up,
7          int right,int bottom)
8     {
9       x1=left;
10      y1=up;
11      x2=right;
12      y2=bottom;
13    }
14    int fun3_1()
15    {
16      return (y2-y1);
17    }
18    int fun3_2()
19    {
20      return (x2-x1);
21    }
22    int fun3_3()
23    {
24      int w=x2-x1;
25      int h=y2-y1;
26      int a=w*h;
27      return a;
28    }
29    int fun3_4()
30    {
31      int w=x2-x1;
32      int h=y2-y1;
33      return 2*w+2*h;
34    }
35  };
```

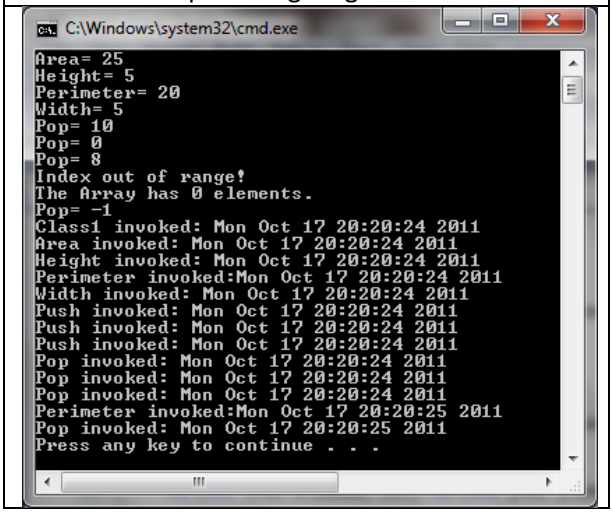
Notice that these three classes are not dependent on the original class that they were extracted from, therefore they are reusable. Following is the client code that we run on both of the versions of the code to verify that our restructuring does not alter program's functional behavior.

Client Code

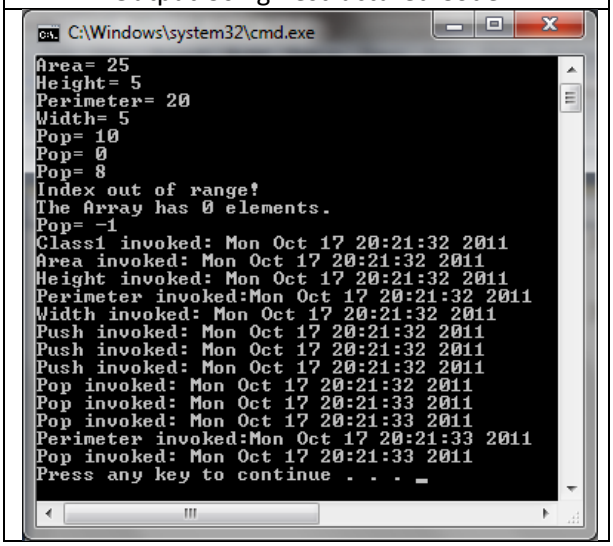
```
#include "Original.h"
int main()
{
 Class1 c(10,10,15,15);
 for(int i=0; i<1000; i++)
    for(int j=0; j<10000; j++);
cout<<"Area= "<<c.Area()<<"\n";
cout<<"Height= "<<c.Height()<<"\n";
cout<<"Perimeter= "<<c.Perimeter()<<"\n";
cout<<"Width= "<<c.Width()<<"\n";
  for(int i=0; i<10000; i++)
    for(int j=0; j<10000; j++);
c.Push(8);
c.Push(0);
  for(int i=0; i<10000; i++)
    for(int j=0; j<1000; j++);
c.Push(10);
cout<<"Pop= "<<c.Pop()<<"\n";
   for(int i=0; i<10000; i++)
    for(int j=0; j<10000; j++);
cout<<"Pop= "<<c.Pop()<<"\n";
cout<<"Pop= "<<c.Pop()<<"\n";
   for(int i=0; i<10000; i++)
    for(int j=0; j<10000; j++);
c.Perimeter();
cout<<"Pop= "<<c.Pop()<<"\n";
   for(int i=0; i<10000; i++)
    for(int j=0; j<10000; j++);
c.printAllInvoks();
   return 0;
}
```

Please verify that the following two outputs generated from running this client code using both original and restructured versions are functionally identical.

Output Using Original Code



Output Using Restructured Code




## 2. Second Example Class – Toker

```
1   class Toker
2   {
3     public:
4     enum mode { code, xml, custom };
5     Toker(const std::string& src = "", bool isFile = true)
6     {
7        prevprevChar=prevChar=currChar=numLines=braceCount=nextChar=0;
8        doReturnComments=EndQuoteCounter=false;
9        doReturnSingleQuotesAsToken=true;
10       pIn=0;
11       _state=default_state;
12      if(src.length() > 0)
13      {
14        if(!attach(src, isFile))
15        {
16          std::string temp = std::string("can't open ") + src;
17          throw std::exception(temp.c_str());
18        }
19      }
20      scTok = "()[]{};.\n";
21      if(_mode == xml)
22      scTok = "<>!" + scTok;
23    }
24    ~Toker()
25    {
26      if(pIn)
27      {
28        pIn->clear();
29        std::ifstream* pFs = dynamic_cast<std::ifstream*>(pIn);
30        if(pFs)
31        {
32          pFs->close();
33        }
```

```
34        delete pIn;
35      }
36    }
37    void setMode(mode md)
38    {
39      _mode = md;
40      scTok = "()[]{};.\n";
41      if(_mode == xml)
42        scTok = "<>!" + scTok;
43    }
44    void setSingleCharTokens(std::string tokChars)
45    {
46      _mode = custom;
47      scTok = tokChars;
48    }
49    bool attach(const std::string& filename, bool isFile = true)
50    {
51      if(pIn && isFile)
52      {
53        pIn->clear();
54        std::ifstream* pFs = dynamic_cast<std::ifstream*>(pIn);
55        if(pFs)
56        {
57          pFs->close();
58        }
59      }
60      if(isFile)
61        pIn = new std::ifstream(filename.c_str());
62      else
63        pIn = new std::istringstream(filename.c_str());
64      return pIn->good();
65    }
66    std::string getTok()
67    {
68      std::string tok = "";
69      stripWhiteSpace();
70      if(isSingleCharTok(nextChar))
71      {
72        getChar();
73        tok.append(1,currChar);
74        return tok;
75      }
76      do
77      {
78        if(isFileEnd())
79          return tok;
80        getChar();
81        if(isBeginComment())
82        {
83          if(tok.length() > 0)
84          {
85            this->putback(currChar);
86            return tok;
87          }
88          tok = eatComment();
89          if(doReturnComments)
90            return tok;
91          else
```

```
92          {
93            tok = "";
94            continue;
95          }
96        }
97        if(isBeginQuote())
98        {
99          if(tok.length() > 0)
100         {
101           this->putback(currChar);
102           return tok;
103         }
104         tok = eatQuote();
105         return tok;
106       }
107       if(!isspace(currChar))
108         tok.append(1,currChar);
109     }  while(!isTokEnd() || tok.length() == 0);
110     return tok;
111   }
112   void returnComments(bool doReturn = true)
113   {
114     doReturnComments = doReturn;
115   }
116   void returnSingleQuotedStringAsToken(bool doCollect=true)
117   {
118     doReturnSingleQuotesAsToken = doCollect;
119     scTok += "\'";
120   }
121   int& lines()
122   {
123     return numLines;
124   }
125   int  braceLevel()
126   {
127     return braceCount;
128   }
129   bool isFileEnd()
130   {
131     return (nextChar == -1);
132   }
133   int  peek()
134   {
135     if(putbacks.size() > 0)
136       return putbacks[putbacks.size()-1];
137     else
138       return pIn->peek();
139 }// peek at next char (not token)
140   void putback(int ch)
141   {
142     putbacks.push_back(ch);
143     nextChar = ch;
144     currChar = prevChar;
145     prevChar = prevprevChar;
146   }  // put back char on stream (not token)
147   enum state { default_state, comment_state, quote_state };
148
149 private:
```

```
150   std::istream* pIn;
152   char prevprevChar, prevChar, currChar, nextChar;
152   std::string scTok;
152   std::vector<char> putbacks;
153   int numLines;
154   int braceCount;
155   bool doReturnComments;
156   bool doReturnSingleQuotesAsToken;
157   bool aCppComment;
158   enum state _state;
159   mode _mode;
160   bool EndQuoteCounter;
161   bool aSingleQuote;
162   // private helper functions
163   int  get()
164   {
165     if(putbacks.size() > 0)
166     {
167       char ch = putbacks.front();
168       putbacks.pop_back();
169       return ch;
170     }
171     return pIn->get();
172   }
173   bool getChar()
174   {
175     char oldNext = nextChar;
176     prevprevChar = prevChar;
177     prevChar = currChar;
178     currChar = this->get();
179     nextChar = this->peek();
180     _ASSERT(currChar == oldNext || oldNext == 0);
181     if(currChar == '\n')
182       ++numLines;
183     if(currChar == '{' && _state == default_state)
184       ++braceCount;
185     if(currChar == '}' && _state == default_state)
186       --braceCount;
187     return !pIn->eof();
188   }
189   bool isSingleCharTok(char ch)
190   {
191     if(scTok.find(ch) < scTok.length())
192       return true;
193     return false;
194   }
195   bool isTokEnd()
196  {
197     if(isspace(nextChar))
198       return true;
199     if(isSingleCharTok(nextChar) || isSingleCharTok(currChar))
200       return true;
201     if(isIdentifierChar(currChar) && !isIdentifierChar(nextChar))
202       return true;
203     if(!isIdentifierChar(currChar) && isIdentifierChar(nextChar))
204       return true;
205     if(isFileEnd())
206       return true;
```

```
207       return false;
208     }
209     void stripWhiteSpace()
210     {
211       if(nextChar == '\n')
212         return;
213       while(isspace(nextChar) && nextChar != '\n')
214       {
215         getChar();
216       }
217     }
218     bool isIdentifierChar(char ch)
219     {
220       if(isalpha(ch) || ch == '_' || isdigit(ch))
221         return true;
222       return false;
223     }
224     bool isBeginComment()
225     {
226       if(prevChar != '\\' && currChar == '/' && nextChar == '*')
227       {
228         aCppComment = false;
229         return true;
230       }
231       if(prevChar != '\\' && currChar == '/' && nextChar == '/')
232       {
233         aCppComment = true;
234         return true;
235       }
236       return false;
237     }
238     bool isEndComment()
239     {
240       if(aCppComment && currChar != '\\' && nextChar == '\n')
241           return true;
242       if(!aCppComment && prevChar != '\\' && currChar == '*' && nextChar == '/')
243         return true;
244       return false;
245     }
246     std::string eatComment()
247     {
248       _state = comment_state;
249       std::string tok(1,currChar);
250       while(!isEndComment() && pIn->good())
251       {
252         getChar();
253         tok.append(1,currChar);
254       }
255       if(!aCppComment)
256       {
257         getChar();
258         tok.append(1,currChar);
259       }
260       _state = default_state;
261       return tok;
262     }
263     bool isBeginQuote()
264     {
```

```
265     if(prevChar != '\\' && currChar == '\'' && doReturnSingleQuotesAsToken)
266     {
267       aSingleQuote = true;
268       return true;
269     }
270     if(prevChar != '\\' && currChar == '\"')
271     {
272       aSingleQuote = false;
273       return true;
274     }
275     return false;
276   }
277   bool isEndQuote()
278   {
279     if(currChar == '\\')
280     {
281       if(prevChar=='\\')
282         EndQuoteCounter = !EndQuoteCounter;
283      else
284         EndQuoteCounter = false;
285     }
286     else
287       EndQuoteCounter = true;
288     if(prevChar == '\\' || currChar != '\\')
289     {
290       if(aSingleQuote && nextChar == '\'' && EndQuoteCounter)
291       {
292         EndQuoteCounter = false;
293         return true;
294       }
295       if(!aSingleQuote && nextChar == '\"' && EndQuoteCounter)
296       {
297         EndQuoteCounter = false;
298         return true;
299       }
300     }
301     return false;
302   }
303   std::string eatQuote()
304   {
305     _state = quote_state;
306     std::string tok(1,currChar);
307     while(!isEndQuote())
308     {
309       getChar();
310       tok.append(1,currChar);
311     }
312     getChar();
313     tok.append(1,currChar);
314     _state = default_state;
315     return tok;
316   }
317   // prohibit copying and assignment
318   Toker(const Toker &tkr);
319   Toker& operator=(const Toker&);
320 };
```