```
1
2
3  //          Course: CS3820-01 Programming Languages
4  //            Name: Sajan, Kefin
5  //      Assignment: Programming Assignment 3
6  //   Date assigned: 12/13/18
7  //        Date due: 12/20/18
8  //  Date handed in: 12/20/18
9  //          Remark: This is a program created to simulate a
…  recursive descent parser
10
11 #include <iostream>
12 #include <stdio.h>
13 #include <ctype.h>
14 #include <string.h>
15
16 using namespace std;
17
18 //Buffer definition//
19 #define BUFFERSIZE 80
20 char tokenBuffer[BUFFERSIZE];
21 bool needToken = true;
22
23 enum tokenType { AND, BEGIN, END, FOR, IF, NOT, OR, READ, WHILE,
…  WRITE, COMMENT, ID, REAL, STRING, PLUS, MULTIPLICATION,
…  ASSIGNMENT, EQUAL, GREATERTHAN, LESSTHAN, LEFTP, COMMA, RIGHTP,
…  SEMICOLON, INVALID, DIVISION, INTEGER };
24
25 //Scanner Functions Headers//
26 tokenType lexical_error(void);
27 //  Return Invaild if the input is not in the system
28
29 void skipSpaces();
30 //  Skips over space characters in the input stream.
31
32 void clearBuffer(void);
33 //  Sets all the elements of the buffer tokenBuffer[] to the
…  null character.
34
35 tokenType getId(void);
36 //  Looks for identifiers.
37
38 tokenType getComment(void);
39 //  Looks for comment, or division char depending on the next
…  character.
40
41 tokenType getReal(void);
42 //  Looks for a real const, or integer.
```

```
43
44 tokenType getStrings(void);
45 //  Looks for a string const
46
47 void displayToken(tokenType code);
48 //  Get an argument as a token code, then displays the
… appropriate message, also prints the contents of the buffer.
49
50 //New function implemented for Assignment 3//
51
52 tokenType getnexttoken(void);
53 //  Calls the function int scanner (void) from the scanner
… assignment
54 //  to get the code of the next token from the input source file
55
56 tokenType peektoken(void);
57 //  Calls the function getnexttoken(void)  and then returns the
… code
58 //  returned by getnexttoken(void)
59
60 tokenType readtoken(void);
61 // Calls the function getnexttoken(void), sets the global
… variable
62 // needToken to true and returns the code returned by
… getnexttoken(void)
63
64 void match(tokenType token);
65 // Receives as argument a token code (to be matched with the
… next input token)
66
67 void syntaxerror(tokenType token);
68 // Receives as argument a token code and then prints an
… appropriate
69 // error message and then skip to the next line.This also make
… sures
70 // that errors in your input source program are only missing
… tokens.
71
72 //Parcer Function Headers//
73 void program(void);
74 void stmntlist(void);
75 void stmnt(void);
76
77 //Scanner Functions//
78
79 tokenType lexical_error(void)
80 // Return Invaild if the input is not in the system
81 {
```

```
82      tokenType userInput;
83      int num = cin.get();
84
85      tokenBuffer[0] = num;
86
87      return INVALID;
88  }
89  void skipSpaces()
90  // Skips over space characters in the input stream.
91  {
92      int usrInp;
93      usrInp = cin.get();
94
95      while (isspace(usrInp))
96          usrInp = cin.get();
97
98      cin.putback(usrInp);
99  }
100 void clear_buf(void)
101 {
102     for (int i = 0; i < BUFFERSIZE; i++)
103         tokenBuffer[i] = '\0';
104 }
105 tokenType getId(void)
106 // Looks for identifiers.
107 {
108     static char reservedWords[][10] =
109     { "AND", "BEGIN", "END", "FOR", "IF", "NOT", "OR", "READ",
… "WHILE", "WRITE" };
110     tokenType userInput;
111     int usrInp;
112     int i = 0;
113
114     usrInp = cin.get();
115
116     if (isalpha(usrInp))
117     {
118         tokenBuffer[i++] = usrInp;
119         usrInp = cin.get();
120         while (isalnum(usrInp))
121         {
122             tokenBuffer[i++] = usrInp;
123             usrInp = cin.get();
124         }
125         cin.putback(usrInp);
126
127
128         int first = 0, mid, last = 9;
```

```
129         bool ntFd = true;
130         while (first <= last && ntFd)
131         {
132             mid = (first + last) / 2;
133             int answer = strcmp(tokenBuffer,
… reservedWords[mid]);
134             if (answer == 0)
135                 ntFd = false;
136             else if (answer > 0)
137                 first = mid + 1;
138             else
139                 last = mid - 1;
140         }
141         if (ntFd)
142             userInput = ID;
143         else
144             userInput = (tokenType)mid;
145     }
146     else
147     {
148         cin.putback(usrInp);
149         return INVALID;
150     }
151     return userInput;
152 }
153
154 tokenType getComment(void)
155 // Finds comment, or divide char depending on the next
… character.
156 {
157     tokenType userInput;
158     int usrInp;
159     int i = 0;
160
161     usrInp = cin.get();
162
163     if (cin.peek() == '*' && usrInp == '/')
164     {
165         tokenBuffer[i++] = usrInp;
166
167         usrInp = cin.get();
168         //tokenBuffer[i++] = usrInp;
169
170         usrInp = cin.get();
171
172         while (!(usrInp == '*' && cin.peek() == '/') &&
… cin.peek() != EOF)
173         {
```

```
174             tokenBuffer[i++] = usrInp;
175             usrInp = cin.get();
176             //Buffer Increase
177         }
178 
179         if (cin.peek() == EOF)
180             userInput = INVALID;
181         else
182         {
183             tokenBuffer[i++] = usrInp;
184             usrInp = cin.get();
185             tokenBuffer[i++] = usrInp;
186             return COMMENT;
187         }
188     }
189     else
190     {
191         cin.putback(usrInp);
192         return INVALID;
193     }
194     return userInput;
195 }
196 
197 tokenType getReal(void)
198 // Looks for a real constant, or integer.
199 {
200     tokenType userInput;
201     int usrInp;
202     int i = 0;
203 
204     if (isdigit(usrInp))
205     {
206         tokenBuffer[i++] = usrInp;
207         usrInp = cin.get();
208 
209         while (isdigit(usrInp))
210         {
211             tokenBuffer[i++] = usrInp;
212             usrInp = cin.get();
213             //Buffer Increase
214         }
215 
216         if (usrInp == '.')
217         {
218             tokenBuffer[i++] = usrInp;
219             usrInp = cin.get();
220             if (isdigit(usrInp))
221             {
```

```
                tokenBuffer[i++] = usrInp;
                usrInp = cin.get();
                while (isdigit(usrInp))
                {
                    tokenBuffer[i++] = usrInp;
                    usrInp = cin.get();
                }
                cin.putback(usrInp);
                return REAL;
            }
            else
            {
                cin.putback(usrInp);
                userInput = INVALID;
            }
        }
        else
        {
            cin.putback(usrInp);
            userInput = INVALID;
        }
    }
    else
    {
        cin.putback(usrInp);
        return INVALID;
    }
    return userInput;
}

tokenType getStrings(void)
// Looks for a string constant.
{
    tokenType userInput;
    int usrInp;
    int i = 0;

    usrInp = cin.get();
    if (usrInp == '\"')
    {
        tokenBuffer[i++] = usrInp;

        usrInp = cin.get();
        while (!(usrInp == '\"') && usrInp != EOF)
        {
            tokenBuffer[i++] = usrInp;

            usrInp = cin.get();
```

```
270         }
271         if (usrInp == EOF)
272             userInput = INVALID;
273         else
274         {
275             tokenBuffer[i++] = usrInp;
276             userInput = STRING;
277         }
278     }
279     else
280     {
281         cin.putback(usrInp);
282         return INVALID;
283     }
284     return userInput;
285 }
286 
287 tokenType getPlus(void)
288 {
289     int usrInp;
290     int i = 0;
291 
292     usrInp = cin.get();
293     if (usrInp == '+')
294     {
295         tokenBuffer[i++] = usrInp;
296         return PLUS;
297     }
298     else
299     {
300         cin.putback(usrInp);
301         return INVALID;
302     }
303 }
304 
305 tokenType getMul(void)
306 {
307     int usrInp;
308     int i = 0;
309 
310     usrInp = cin.get();
311     if (usrInp == '*')
312     {
313         tokenBuffer[i++] = usrInp;
314         return MULTIPLICATION;
315     }
316     else
317     {
```

```
318         cin.putback(usrInp);
319         return INVALID;
320     }
321 }
322 
323 tokenType getAssign(void)
324 {
325     int usrInp;
326     int usrInp2;
327     int i = 0;
328 
329     usrInp = cin.get();
330     usrInp2 = cin.peek();
331     if (usrInp == ':' && usrInp2 == '=')
332     {
333         tokenBuffer[i++] = usrInp;
334         usrInp = cin.get();
335         tokenBuffer[i] = usrInp;
336         return ASSIGNMENT;
337     }
338     else
339     {
340         cin.putback(usrInp);
341         return INVALID;
342     }
343 }
344 
345 tokenType getEqual(void)
346 {
347     tokenType userInput;
348     int usrInp;
349     int i = 0;
350 
351     usrInp = cin.get();
352     if (usrInp == '=')
353     {
354         tokenBuffer[i++] = usrInp;
355         userInput= EQUAL;
356     }
357     else
358     {
359         cin.putback(usrInp);
360         userInput= INVALID;
361     }
362     return userInput;
363 }
364 
365 tokenType getGreater(void)
```

```
{
    tokenType userInput;
    int usrInp;
    int i = 0;

    usrInp = cin.get();
    if (usrInp == '>')
    {
        tokenBuffer[i++] = usrInp;
        return GREATERTHAN;
    }
    else
    {
        cin.putback(usrInp);
        return INVALID;
    }
}

tokenType getLess(void)
{
    int usrInp;
    int i = 0;

    usrInp = cin.get();
    if (usrInp == '<')
    {
        tokenBuffer[i++] = usrInp;
        return LESSTHAN;
    }
    else
    {
        cin.putback(usrInp);
        return INVALID;
    }
}


tokenType getLP(void)
{
    tokenType userInput;
    int usrInp;
    int i = 0;

    usrInp = cin.get();
    if (usrInp == '(')
    {
        tokenBuffer[i++] = usrInp;
        return LEFTP;
```

```
414     }
415     else
416     {
417         cin.putback(usrInp);
418         return INVALID;
419     }
420 }
421 
422 
423 tokenType getRP(void)
424 {
425     tokenType userInput;
426     int usrInp;
427     int i = 0;
428 
429     usrInp = cin.get();
430     if (usrInp == ')')
431     {
432         tokenBuffer[i++] = usrInp;
433         return RIGHTP;
434     }
435     else
436     {
437         cin.putback(usrInp);
438         return INVALID;
439     }
440 }
441 
442 
443 tokenType getComma(void)
444 {
445     int usrInp;
446     int i = 0;
447 
448     usrInp = cin.get();
449     if (usrInp == ',')
450     {
451         tokenBuffer[i++] = usrInp;
452         return COMMA;
453     }
454     else
455     {
456         cin.putback(usrInp);
457         return INVALID;
458     }
459 }
460 
461 
```

```
462 tokenType getSColon(void)
463 {
464     int usrInp;
465     int i = 0;
466 
467     usrInp = cin.get();
468     if (usrInp == ';')
469     {
470         tokenBuffer[i++] = ';';
471         return SEMICOLON;
472     }
473     else
474     {
475         cin.putback(usrInp);
476         return INVALID;
477     }
478 }
479 
480 tokenType scanner(void)
481 {
482     skipSpaces();
483     int usrInp = cin.get();
484 
485     if(usrInp == EOF)
486         return(tokenType)EOF;
487 
488     if (usrInp == '/')
489     {
490         cin.putback(usrInp);
491         return getComment();
492     }
493 
494     if (isalpha(usrInp))
495     {
496         cin.putback(usrInp);
497         return getId();
498     }
499 
500     if (isdigit(usrInp))
501     {
502         cin.putback(usrInp);
503         return getReal();
504     }
505 
506     if (usrInp == '\"')
507     {
508         cin.putback(usrInp);
509         return getStrings();
```

```
510     }
511
512     if (usrInp == '+')
513     {
514         cin.putback(usrInp);
515         return getPlus();
516     }
517
518     if (usrInp == '*')
519     {
520         cin.putback(usrInp);
521         return getMul();
522     }
523
524     if (usrInp == '=')
525     {
526         cin.putback(usrInp);
527         return getEqual();
528     }
529
530     if (usrInp == ':')
531     {
532         cin.putback(usrInp);
533         return getAssign();
534     }
535
536     if(usrInp == '>')
537     {
538         cin.putback(usrInp);
539         return getGreater();
540     }
541
542     if (usrInp == '<')
543     {
544         cin.putback(usrInp);
545         return getLess();
546     }
547
548     if (usrInp == '(')
549     {
550         cin.putback(usrInp);
551         return getLP();
552     }
553
554     if (usrInp == ')')
555     {
556         cin.putback(usrInp);
557         return getRP();
```

```
558     }
559
560     if (usrInp == ',')
561     {
562         cin.putback(usrInp);
563         return getComma();
564     }
565
566     if (usrInp == ';')
567     {
568         cin.putback(usrInp);
569         return getSColon();
570     }
571
572     // Can be only a lexical error //
573
574     cin.putback(usrInp);
575     return lexical_error();
576
577 }
578
579 void display_token(tokenType code)
580 // Get an argument as a token code, then displays the
 … appropriate message, also prints the contents of the buffer.
581 {
582     const char *MESS[] = { "and", "begin", "end", "for", "if",
 … "not", "or", "read", "while", "write", "comment", "identifier",
 … "real constant", "string", "plus", "multiplication",
 … "assignment", "equal", "greater than", "less than", "left
 … parenthesis", "comma", "right parenthesis", "semicolon",
 … "invalid", "division", "integer" };
583     cout << " \n\t\t" << MESS[(int)code] << "\t" << tokenBuffer;
584 }
585
586 //New function implemented for Assignment 3
587 tokenType getnexttoken(void)
588 {
589     tokenType nexttoken;
590     if (needToken)
591     {
592         nexttoken = scanner();
593         needToken = false;
594     }
595     return nexttoken;
596 }
597
598 tokenType peektoken(void)
599 {
```

```
600     return getnexttoken();
601 }
602 
603 tokenType readtoken(void)
604 {
605     tokenType tok = getnexttoken();
606     needToken = true;
607     return tok;
608 }
609 
610 void syntaxerror(tokenType token)
611 {
612     static char message[][20] =  { "AND", "BEGIN", "END", "FOR",
… "IF", "NOT","OR", "READ", "WHILE", "WRITE", "COMMENT",
… "IDENTIFIER",  "REAL CONSTANT","STRING", "PLUS",
… "MULTIPLICATION", "ASSIGNMENT", "EQUAL", "GREATER THAN",
613         "LESS THAN", "LEFT PARENTHESIS", "COMMA", "RIGHT
… PARENTHESIS", "SEMICOLON","INVALID", "DIVISION", "INTEGER"
614     };
615 
616         cout<< " token \"" << message[(int)token] << "\"\t"<< "
… # "
617             << (int)token << endl;
618 
619 }
620 
621 void match(tokenType token)
622 {
623     int tok = readtoken();
624 
625     if (token == tok)
626     { if (tok != SEMICOLON)
627         {
628             cout << tokenBuffer << endl;
629             clear_buf();
630         }
631     }
632         else
633             syntaxerror(token);
634 }
635 
636 //End of Scanner Functions//
637 
638 //Parser Functions//
639 void program(void)
640 {
641     match(BEGIN);
642     stmntlist();
```

```
    match(END);
}

void stmntlist(void)
{
    tokenType nexttoken = peektoken();

    if(nexttoken){

        stmnt();
        match(SEMICOLON);
        stmntlist();
    }
    else
        syntaxerror(nexttoken);
}

void stmnt(void)
{
    tokenType nexttoken = peektoken();

    if (nexttoken)
    {
        match(peektoken());
        match(ASSIGNMENT);
        stmntlist();
        match(SEMICOLON);
    }
    else
        syntaxerror(nexttoken);
}
//End of Parser Functions//

//Main Function//
int main()
{
    program();
    return 0;
}
```