```
/*
    Andrew Maclean
    March 15th, 2017
    CS 4110 Compiler

    program     -> blockst
    blockst     -> BEGINTOK stats ENDTOK
    stats       -> statmt ';' stats | empty
    decl        -> BASICTYPETOK IDTOK
    statmt      -> decl | ifstat | assstat |  blockst | loopst | iostat | empty
    assstat     -> idref ASTOK expression
    ifstat      -> IFTOK  expression THENTOK statmt
    loopst      -> WHILETOK expression DOTOK statmt
    iostat      -> READTOK ( idref ) | WRITETOK (expression)
    expression  -> term expprime
    expprime    -> ADDOPTOK  term expprime | empty
    term        -> relfactor termprime
    termprime   -> MULOPTOK  relfactor termprime | empty
    relfactor   -> factor factorprime
    factorprime -> RELOPTOK factor | empty
    factor      -> NOTTOK factor | idref | LITTOK | '(' expression ')'
    idref       -> IDTOK


    READ / WRITE doens't appear to work correctly
    I'm not sure what write line does, exactly
    Assembly code needs optimization

*/


#ifndef PARSER_H
#define PARSER_H

#include <iostream>
#include <fstream>
#include <string>
#include <queue>
#include "SymbolTable.h"

using namespace std;

struct token
{
    string tokenName;
    int lineNo;
    int tokenNumber;
};

class Parser
{
public:
```

```
53      Parser(queue<token> tokenList)
54      {
55          tokens = tokenList;
56          identifiers = new HashTable();
57          assem.open("assemblyCode.txt");
58          err.open("errors.txt");
59          currentAddr = 0;
60          program();
61          assem.close();
62          err.close();
63          currTemp = 0;
64      }
65
66      // makes sure there is at least one token left to pop
67      void countCheck()
68      {
69          if (tokens.size() == 0)
70          {
71              err << "\nNot enough tokens" << endl;
72              system("pause");
73              exit(EXIT_FAILURE);
74          }
75      }
76
77      //does prints the rule number somewhere
78      void print(int tok)
79      {
80          //cout << tok << " ";
81      }
82      //idref -> IDTOK
83      void idref(bool left)
84      {
85          print(31);
86
87          //as long as the symbol table finds the identifier...
88          if (identifiers->find(tokens.front().tokenName)->name != "Not Found")
89          {
90              //doesn't write any assembly for lefthand identifiers
91              if (!left)
92              {
93                  //pulls an identifiers and puts it in the next spot in the stack
94                  assem << "lw $t" << currTemp << " " << identifiers->find
                        (tokens.front().tokenName)->address << " # load value\n";
95                  assem << "sw $t" << currTemp << " " << currentAddr << endl;
96                  currentAddr -= 4;
97              }
98
99              tokens.pop(); //pop IDTOK
100         }
101
102         //give an error otherwise
103         else
```

```
104         {
105             err << "\nIdentifier " << tokens.front().tokenName << " was not found
                    on line: " << tokens.front().lineNo << endl;
106             tokens.pop();
107         }
108
109         countCheck();
110     }
111
112     //factor -> NOTTOK factor | idref | LITTOK | '(' expression ')'
113     void factor()
114     {
115         // token 17 is NOTTOK
116         if (tokens.front().tokenNumber == 17)
117         {
118             print(27);
119             tokens.pop(); //pop NOTTOK
120             factor();
121         }
122         //token 1 is ID
123         else if (tokens.front().tokenNumber == 1)
124         {
125             print(28);
126             idref(false);
127         }
128         //token 2 is LITERAL
129         else if (tokens.front().tokenNumber == 2)
130         {
131             currTemp++;
132             print(29);
133             //don't follow this with strings
134             if (tokens.front().tokenName[0] != '"')
135             {
136                 // if the token is a number
137                 if (isdigit(tokens.front().tokenName[0]))
138                     assem << "li $t" << currTemp << ", " << tokens.front
                         ().tokenName << " # assinging an int value" << endl;
139
140                 // if it's false
141                 else if (tokens.front().tokenName[0] == 'F')
142                     assem << "li $t" << currTemp << ", 0 # assigning a false
                         value\n";
143
144                 // if it's true
145                 else if (tokens.front().tokenName[0] == 'T')
146                     assem << "li $t" << currTemp << ", 1 # assigning a true value
                         \n";
147
148                 // store in next available spot in stack
149                 assem << "sw $t" << currTemp << ", " << currentAddr << endl;
150             }
151
```

```
152             tokens.pop(); //pop LITTOK
153             countCheck();
154             currentAddr -= 4;
155         }
156         //token 14 is (
157         else if (tokens.front().tokenNumber == 14)
158         {
159             print(30);
160 
161             tokens.pop(); //pop (
162 
163             countCheck();
164             expression();
165 
166             //token 15 is )
167             if (tokens.front().tokenNumber != 15) // missing token error
168                 err << "\nExpected \')\' on line: " << tokens.front().lineNo <<
                      endl;
169             else
170                 tokens.pop(); //pop )
171         }
172         countCheck();
173 
174     }
175 
176     //factorprime -> RELOPTOK factor | empty
177     void factorprime()
178     {
179         string temp;
180         //token 6 is RELOPTOK
181         if (tokens.front().tokenNumber == 6)
182         {
183             print(25);
184             // saves the token for later
185             temp = tokens.front().tokenName;
186             tokens.pop(); //pop RELOPTOK
187             countCheck();
188             factor();
189 
190             currentAddr += 4;
191             currTemp = 0;
192 
193             // compares and goes to skip
194             assem << "lw $t" << currTemp << " " << currentAddr << endl;
195             currentAddr += 4;
196             currTemp++;
197             assem << "lw $t" << currTemp << " " << currentAddr << endl;
198 
199             if (temp == "<")
200                 assem << "blt $t" << currTemp << " $t" << currTemp - 1 << " Skip"
                        << endl;
201 
```

```
202             else if (temp == ">")
203                 assem << "bgt $t" << currTemp << " $t" << currTemp - 1 << " Skip" ↵
                       << endl;
204
205             else if (temp == "!=")
206                 assem << "bne $t" << currTemp << " $t" << currTemp - 1 << " Skip" ↵
                       << endl;
207
208             assem << "sw $t" << currTemp << " " << currentAddr << endl;
209
210             currTemp = 0;
211         }
212         else
213             print(26);
214     }
215
216     //relfactor -> factor factorprime
217     void relfactor()
218     {
219         print(24);
220         factor();
221
222         factorprime();
223     }
224
225     //termprime -> MULOPTOK relfactor termprime | empty
226     void termprime()
227     {
228         string temp;
229         //token 5 is MULOPTOK
230         if (tokens.front().tokenNumber == 5)
231         {
232             print(22);
233             temp = tokens.front().tokenName;
234             tokens.pop(); //pop MULOPTOK
235             countCheck();
236             relfactor();
237             currentAddr += 4;
238             currTemp = 0;
239
240             // loads two registers with the numbers to be operated
241             assem << "lw $t" << currTemp << " " << currentAddr << endl;
242             currentAddr += 4;
243             currTemp++;
244             assem << "lw $t" << currTemp << " " << currentAddr << endl;
245             currTemp++;
246
247             // computes the operation
248             if (temp == "*")
249                 assem << "mult $t" << currTemp << " $t" << currTemp - 1 << " $t"  ↵
                       << currTemp - 2 << endl;
250
```

```
251             else if (temp == "AND")
252                 assem << "and $t" << currTemp << " $t" << currTemp - 1 << " $t"
                      << currTemp - 2 << endl;
253
254             else if (temp == "/" || temp == "DIV")
255                 assem << "div $t" << currTemp - 1 << " $t" << currTemp - 2 <<
                      endl;
256                 assem << "mflo $t" << currTemp << endl;
257
258             // stores the result
259             assem << "sw $t" << currTemp << " " << currentAddr << endl;
260
261             currentAddr -= 4;
262             currTemp = 0;
263             termprime();
264
265         }
266         else
267             print(23);
268     }
269
270     //term -> relfactor termprime
271     void term()
272     {
273         print(21);
274         relfactor();
275         termprime();
276     }
277
278     //expprime -> ADDOPTOK term expprime | empty
279     void expprime()
280     {
281         string temp;
282         //token 4 is ADDOPTOK
283         if (tokens.front().tokenNumber == 4)
284         {
285             print(19);
286             temp = tokens.front().tokenName;
287             tokens.pop(); //pop ADDOPTOK
288             countCheck();
289             term();
290
291             //div1 and div2 are the two temps being worked on
292             currTemp = 0;
293             currentAddr += 4;
294
295             assem << "lw $t" << currTemp << " " << currentAddr << endl;
296             currTemp++;
297             currentAddr += 4;
298             assem << "lw $t" << currTemp << " " << currentAddr << endl;
299             currTemp++;
300
```

```
301             // mostly the same as termprime
302             if (temp == "+")
303                 assem << "add $t" << currTemp << " $t" << currTemp - 1 << " $t"
                      << currTemp - 2 << endl;
304
305             else if (temp == "-")
306                 assem << "sub $t" << currTemp << " $t" << currTemp - 1 << " $t"
                      << currTemp - 2 << endl;
307
308             else if (temp == "OR")
309                 assem << "or $t" << currTemp << " $t" << currTemp - 1 << " $t" <<
                       currTemp - 2 << endl;
310
311             assem << "sw $t" << currTemp << " " << currentAddr << endl;
312
313             currentAddr -= 4;
314             currTemp = 0;
315             expprime();
316
317         }
318         else
319             print(20);
320     }
321
322     //expression -> term expprime
323     void expression()
324     {
325         print(18);
326         term();
327         expprime();
328     }
329
330     //iostat -> READTOK '(' idref ')' | WRITETOK '(' expression ')'
331     void iostat()
332     {
333         string key;
334
335         key = tokens.front().tokenName;
336         tokens.pop(); //pop READTOK or WRITETOK
337         countCheck();
338
339         if (tokens.front().tokenNumber != 14) // missing token error, (
340             err << "\nExpected \'(\' in iostat on line: " << tokens.front
                      ().lineNo << endl;
341         else
342             tokens.pop(); //pop open parenthesis
343
344         countCheck();
345
346         if (key == "READ")
347         {
348             // reads an input
```

```
349             print(16);
350             idref(false);
351 
352         }
353         else
354         {
355             print(17);
356             // writeline function
357             if (key == "WRITELN")
358             {
359                 // writes a string
360                 if (identifiers->find(tokens.front().tokenName)->type ==
                      "string")
361                 {
362                     assem << "output: .asciiz \"\\n" << tokens.front().tokenName
                       << "\\n\"" << endl;
363                     assem << "la $a0 output" << endl;
364                     assem << "li $v0 4" << endl;
365                 }
366                 // writes a variable from memory
367                 else
368                 {
369                     assem << "lw $a0 " << currentAddr << endl;
370                     assem << "li $v0 1" << endl;
371                 }
372                 expression();
373                 assem << "syscall" << endl;
374             }
375             // regular write function
376             else
377             {
378                 if (identifiers->find(tokens.front().tokenName)->type ==
                      "string")
379                 {
380                     assem << "output: .asciiz \"" << tokens.front().tokenName <<
                       "\"" << endl;
381                     assem << "la $a0 output" << endl;
382                     assem << "li $v0 4" << endl;
383                 }
384                 else
385                 {
386                     assem << "lw $a0 " << currentAddr << endl;
387                     assem << "li $v0 1" << endl;
388                 }
389                 expression();
390                 assem << "syscall" << endl;
391             }
392 
393         }
394 
395         if (tokens.front().tokenNumber != 15) // missing token error on )
396             err << "\nExpected \')\' in iostat on line: " << tokens.front
```

```
                ().lineNo << endl;
397         else
398             tokens.pop(); //pop closed parentheis
399
400         countCheck();
401     }
402
403     //loopst -> WHILETOK expression DOTOK statmt
404     void loopst()
405     {
406         print(15);
407         // WHILE is token 11
408         tokens.pop(); //pop WHILETOK
409         assem << "Loop: ";
410         expression();
411         //DO is token 12
412         if (tokens.front().tokenNumber != 12) // missing token error
413             err << "\nExpected \"DO\" token, got " << tokens.front().tokenName << ↵
                    " on line: " << tokens.front().lineNo << endl;
414         else
415             tokens.pop(); //pop DOTOK
416         countCheck();
417
418         statmt();
419         assem << "j Loop " << endl;
420         assem << "Skip: ";
421     }
422
423     //ifstat -> IFTOK expression THENTOK statmt
424     void ifstat()
425     {
426         print(14);
427
428         tokens.pop(); //pop IFTOK
429
430         countCheck();
431         expression();
432
433         if (tokens.front().tokenNumber != 10) // missing token error, token 10 is ↵
                THEN
434             err << "\nExpected \"THEN\" token, got " << tokens.front().tokenName  ↵
                    << " on line: " << tokens.front().lineNo << endl;
435         else
436             tokens.pop(); //pop THENTOK
437
438         countCheck();
439         statmt();
440         // the position to skip to
441         assem << "Skip: ";
442     }
443
444     //assstat -> idref ASTOK expression
```

```
445     void assstat()
446     {
447         // loc is the offset of a variable
448         int loc;
449         //as long as the symbol table finds the identifier...
450         if (identifiers->find(tokens.front().tokenName)->name != "Not Found")
451         {
452             Data *data = identifiers->find(tokens.front().tokenName);
453             loc = data->address;
454             bool left = true;
455             idref(left);
456         }
457         // clears the code and declares an error
458         else
459         {
460             err << "\nUninitialized variable " << tokens.front().tokenName << "
                  on line: " << tokens.front().lineNo << endl;
461             assem.close();
462             assem.open("assemblyCode.txt");
463             assem << "";
464             assem.close();
465             exit(EXIT_FAILURE);
466         }
467         if (tokens.front().tokenNumber != 19) // missing token error ASTOK
468             err << "\nExpected :=, got " << tokens.front().tokenName << " on
                  line: " << tokens.front().lineNo << endl;
469         else
470             tokens.pop(); // pops assignment token
471         //countCheck makes sure that there is at least one token left
472         countCheck();
473         expression();
474         currentAddr += 4;
475         // stores the last spot in memory to a variable
476         assem << "lw $t" << currTemp << " " << currentAddr << endl;
477         assem << "sw $t" << currTemp << " " << loc << " # assign to variable" <<
              endl;
478         currTemp = 0;
479     }
480
481     //stmt -> decl | ifstat | assstat | blockst | loopst | iostat | empty
482     void statmt()
483     {
484         if (tokens.front().tokenNumber == 3)
485         {
486             print(6);
487             decl();
488         }
489         else if (tokens.front().tokenNumber == 9)
490         {
491             print(7);
492             ifstat();
493         }
```

```
494         else if (tokens.front().tokenNumber == 1)
495         {
496             print(8);
497             assstat();
498         }
499         else if (tokens.front().tokenNumber == 7)
500         {
501             print(9);
502             blockst();
503         }
504         else if (tokens.front().tokenNumber == 11)
505         {
506             print(10);
507             loopst();
508         }
509         else if (tokens.front().tokenNumber == 13)
510         {
511             print(11);
512             iostat();
513         }
514         else
515             print(12);
516     }
517
518     //decl -> BASICTYPETOK IDTOK
519     void decl()
520     {
521         print(5);
522         // the token type before it gets popped
523         string type = tokens.front().tokenName;
524         tokens.pop(); //pop TYPETOK
525
526         countCheck();
527
528         if (tokens.front().tokenNumber != 1) // missing token error IDENT
529             err << "\nMissing identifier on line: " << tokens.front().lineNo <<
                  endl;
530         else
531         {
532             // fills the symbol table with the new identifier
533             if (identifiers->find(tokens.front().tokenName)->name == "Not Found")
534             {
535                 Data inserted = { type, tokens.front().tokenName, currentAddr};
536                 identifiers->insert(inserted);
537                 assem << "# stored a new variable in the next offset" << endl;
538                 tokens.pop(); // pop IDTOK
539             }
540             //doesn't make two identifiers
541             else
542             {
543                 err << "\nIdentifier already exists, on line: " << tokens.front
                      ().lineNo << endl;
```

```
544                 tokens.pop();
545             }
546         }
547         countCheck();
548         currentAddr -= 4;
549     }
550
551     //stats -> statmt ';' stats | empty
552     void stats()
553     {
554         if (tokens.front().tokenNumber != 8)
555         {
556             print(3);
557             statmt();
558             if (tokens.front().tokenNumber != 16) // missing token error
559                 err << "\nMissing \';\' on line: " << tokens.front().lineNo <<
                      endl;
560             else //pop ;
561             {
562                 tokens.pop();
563                 countCheck();
564                 assem << "\n# new statement" << endl;
565                 stats();
566             }
567         }
568         else
569             print(4);
570     }
571
572     //blockst -> BEGINTOK stats ENDTOK
573     void blockst()
574     {
575         print(2);
576         if (tokens.front().tokenNumber != 7) // missing token error
577             err << "\nMissing begin token on line: " << tokens.front().lineNo <<
                  endl;
578         else
579         {
580             //pop BEGIN
581             tokens.pop();
582             identifiers->newScope();
583         }
584         countCheck(); // makes sure that the token count is greater than zero
585         stats();
586         if (tokens.front().tokenNumber != 8) // missing token error
587             err << "\nMissing end token on line: " << tokens.front().lineNo <<
                  endl;
588         else
589         {
590             //pop END
591             tokens.pop();
592             identifiers->closeScope();
```

```
593         }
594         countCheck();
595     }
596
597     //program -> blockst '.'
598     void program()
599     {
600         assem << ".text\n.globl main\nmove $fp $sp\nla $a0 ProgStart\nli $v0 4
              \nsyscall\n\n";
601         blockst();
602         assem << "\nla $a0 ProgEnd\nli $v0 4\nsyscall\nli $v0 10\nsyscall\n.data
              \nProgStart: .asciiz \"Program Start\\n\"\nProgEnd: .asciiz \"Program
              End\\n\"";
603         if (tokens.empty()) // missing token error
604             err << "\nNo end of program token\n";
605         cout << "\n";
606         //prints the symbol table
607         identifiers->print();
608     }
609
610 private:
611         queue<token> tokens;
612         HashTable* identifiers;
613         int currentAddr;
614         ofstream assem;
615         ofstream err;
616         int currTemp;
617 };
618
619 #endif
```