```
88          /// <summary>
89          /// Runs the program using the data from the given Quad Table and Symbol Table.
90          /// Trace mode will print each quad code that the interpretter executes.
91          /// </summary>
92          /// <param name="quadTable">Quad Table containing all the necessary Quad Codes</param>
93          /// <param name="symbolTable">Symbol Table containing all the necessary Symbols</param>
94          /// <param name="TraceOn">Toggles Trace Mode on and off</param>
95          public void InterpretQuads(QuadTable quadTable, SymbolTable symbolTable, bool TraceOn = false)
96          {
97              QuadTable = quadTable;
98              SymbolTable = symbolTable;
99              ProgramCounter = 0;
100             while (ProgramCounter < QuadTable.NextQuad())
101             {
102                 CurrentQuad = QuadTable.GetQuad(ProgramCounter);
103                 if (QuadTable.ReserveTable.isValidOpCode(CurrentQuad.OpCode))
104                 {
105                     try
106                     {
107                         switch (CurrentQuad.OpCode)
108                         {
109                             // STOP
110                             // Terminate program
111                             case STOP:
112                                 if (TraceOn)
113                                 {
114                                     PrintTrace(CurrentQuad.OpCode);
115                                 }
116                                 ProgramCounter = QuadTable.NextQuad();
117                                 break;
118                             // DIV
119                             // Compute op1 / op2, place result into op3
120                             case DIV:
121                                 if (TraceOn)
122                                 {
123                                     PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op1, CurrentQuad.Op2,
                                     CurrentQuad.Op3);
124                                 }
125                                 double quotient = Convert.ToDouble(SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue
```

```
                                ()) / SymbolTable.GetSymbol(CurrentQuad.Op2).GetValue();
126                                 SymbolTable.UpdateSymbol(CurrentQuad.Op3, SymbolKind.Variable, quotient);
127                                 ProgramCounter++;
128                                 break;
129                             // MUL
130                             // Compute op1 * op2, place result into op3
131                             case MUL:
132                                 if (TraceOn)
133                                 {
134                                     PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op1, CurrentQuad.Op2,
                                CurrentQuad.Op3);
135                                 }
136                                 SymbolTable.UpdateSymbol(CurrentQuad.Op3, SymbolKind.Variable,
137                                     (SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue() * SymbolTable.GetSymbol
                                (CurrentQuad.Op2).GetValue()));
138                                 ProgramCounter++;
139                                 break;
140                             // SUB
141                             // Compute op1 - op2, place result into op3
142                             case SUB:
143                                 if (TraceOn)
144                                 {
145                                     PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op1, CurrentQuad.Op2,
                                CurrentQuad.Op3);
146                                 }
147                                 SymbolTable.UpdateSymbol(CurrentQuad.Op3, SymbolKind.Variable,
148                                     (SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue() - SymbolTable.GetSymbol
                                (CurrentQuad.Op2).GetValue()));
149                                 ProgramCounter++;
150                                 break;
151                             // ADD
152                             // Compute op1 + op2, place result into op3
153                             case ADD:
154                                 if (TraceOn)
155                                 {
156                                     PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op1, CurrentQuad.Op2,
                                CurrentQuad.Op3);
157                                 }
```

```
158                                 SymbolTable.UpdateSymbol(CurrentQuad.Op3, SymbolKind.Variable,
159                                     (SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue() + SymbolTable.GetSymbol
                                    (CurrentQuad.Op2).GetValue()));
160                                 ProgramCounter++;
161                                 break;
162                             // MOV
163                             // Assign the value in op1 into op3 (op2 is ignored here)
164                             case MOV:
165                                 if (TraceOn)
166                                 {
167                                     PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op1, CurrentQuad.Op3);
168                                 }
169                                 SymbolTable.UpdateSymbol(CurrentQuad.Op3, SymbolKind.Variable, SymbolTable.GetSymbol
                                    (CurrentQuad.Op1).GetValue());
170                                 ProgramCounter++;
171                                 break;
172                             // STI
173                             // Store indexed - Assign the value in op1 into op2 + offset op3
174                             case STI:
175                                 if (TraceOn)
176                                 {
177                                     PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op1, CurrentQuad.Op2,
                                    CurrentQuad.Op3);
178                                 }
179                                 SymbolTable.UpdateSymbol((CurrentQuad.Op2 + CurrentQuad.Op3), SymbolKind.Variable,
                                    SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue());
180                                 ProgramCounter++;
181                                 break;
182                             // LDI
183                             // Load indexed- Assign the value in op1 + offset op2, into op3
184                             case LDI:
185                                 if (TraceOn)
186                                 {
187                                     PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op1, CurrentQuad.Op2,
                                    CurrentQuad.Op3);
188                                 }
189                                 SymbolTable.UpdateSymbol(CurrentQuad.Op3, SymbolKind.Variable, SymbolTable.GetSymbol
                                    (CurrentQuad.Op1 + CurrentQuad.Op2).GetValue());
```

```
190                                 ProgramCounter++;
191                                 break;
192                             // BNZ
193                             // Branch Not Zero; if op1 value <> 0, set program counter to op3
194                             case BNZ:
195                                 if (TraceOn)
196                                 {
197                                     PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op3);
198                                 }
199                                 if (SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue() != 0)
200                                 {
201                                     ProgramCounter = CurrentQuad.Op3;
202                                 }
203                                 else
204                                 {
205                                     ProgramCounter++;
206                                 }
207                                 break;
208                             // BNP
209                             // Branch Not Positive; if op1 value <= 0, set program counter to op3
210                             case BNP:
211                                 if (TraceOn)
212                                 {
213                                     PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op3);
214                                 }
215                                 if (SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue() <= 0)
216                                 {
217                                     ProgramCounter = CurrentQuad.Op3;
218                                 }
219                                 else
220                                 {
221                                     ProgramCounter++;
222                                 }
223                                 break;
224                             // BNN
225                             // Branch Not Negative; if op1 value >= 0, set program counter to op3
226                             case BNN:
227                                 if (TraceOn)
228                                 {
```

```
229                                     PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op3);
230                                 }
231                                 if (SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue() >= 0)
232                                 {
233                                     ProgramCounter = CurrentQuad.Op3;
234                                 }
235                                 else
236                                 {
237                                     ProgramCounter++;
238                                 }
239                                 break;
240                             // BZ
241                             // Branch Zero; if op1 value = 0, set program counter to op3
242                             case BZ:
243                                 if (TraceOn)
244                                 {
245                                     PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op3);
246                                 }
247                                 if (SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue() == 0)
248                                 {
249                                     ProgramCounter = CurrentQuad.Op3;
250                                 }
251                                 else
252                                 {
253                                     ProgramCounter++;
254                                 }
255                                 break;
256                             // BP
257                             // Branch Positive; if op1 value > 0, set program counter to op3
258                             case BP:
259                                 if (TraceOn)
260                                 {
261                                     PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op3);
262                                 }
263                                 if (SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue() > 0)
264                                 {
265                                     ProgramCounter = CurrentQuad.Op3;
266                                 }
267                                 else
268                                 {
```

```
269                                     ProgramCounter++;
270                                 }
271                                 break;
272                             // BN
273                             // Branch Negative; if op1 value < 0, set program counter to op3
274                             case BN:
275                                 if (TraceOn)
276                                 {
277                                     PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op3);
278                                 }
279                                 if (SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue() < 0)
280                                 {
281                                     ProgramCounter = CurrentQuad.Op3;
282                                 }
283                                 else
284                                 {
285                                     ProgramCounter++;
286                                 }
287                                 break;
288                             // BR
289                             // Branch (unconditional); set program counter to op3
290                             case BR:
291                                 if (TraceOn)
292                                 {
293                                     PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op3);
294                                 }
295                                 ProgramCounter = CurrentQuad.Op3;
296                                 break;
297                             // BINDR
298                             // Branch (unconditional); set program counter to op3 value contents (indirect)
299                             case BINDR:
300                                 if (TraceOn)
301                                 {
302                                     PrintTrace(CurrentQuad.OpCode, SymbolTable.GetSymbol(CurrentQuad.Op3).GetValue
                                 ());
303                                 }
304                                 ProgramCounter = SymbolTable.GetSymbol(CurrentQuad.Op3).GetValue();
305                                 break;
306                             // PRINT
```

```
307                             // Write symbol table name and value of op 1
308                             case PRINT:
309                                 if (TraceOn)
310                                 {
311                                     PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op1);
312                                 }
313                                 // Console.WriteLine($"{ SymbolTable.GetSymbol(CurrentQuad.Op1).Name} =
                                     {SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue()}");
314                                 Console.WriteLine($"{SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue()}");
315                                 ProgramCounter++;
316                                 break;
317                             default:
318                                 Console.WriteLine($"Invalid Opcode {CurrentQuad.OpCode}");
319                                 break;
320                         }
321                     }
322                     // Catches any exception, prints the appropriate error message, and stops running the current
                       program.
323                     catch (Exception e)
324                     {
325                         Console.WriteLine("FATAL ERROR: " + e.Message + "\n");
326                         ProgramCounter = QuadTable.NextQuad();
327                     }
328                 }
329             }
330         }
```