```csharp
        /// <summary>
        /// Runs the program using the data from the given Quad Table and Symbol Table.
        /// Trace mode will print each quad code that the interpretter executes.
        /// </summary>
        /// <param name="quadTable">Quad Table containing all the necessary Quad Codes</param>
        /// <param name="symbolTable">Symbol Table containing all the necessary Symbols</param>
        /// <param name="TraceOn">Toggles Trace Mode on and off</param>
        public void InterpretQuads(QuadTable quadTable, SymbolTable symbolTable, bool TraceOn = false)
        {
            QuadTable = quadTable;
            SymbolTable = symbolTable;
            ProgramCounter = 0;
            while (ProgramCounter < QuadTable.NextQuad())
            {
                CurrentQuad = QuadTable.GetQuad(ProgramCounter);
                if (QuadTable.ReserveTable.isValidOpCode(CurrentQuad.OpCode))
                {
                    try
                    {
                        switch (CurrentQuad.OpCode)
                        {
                            // STOP
                            // Terminate program
                            case STOP:
                                if (TraceOn)
                                {
                                    PrintTrace(CurrentQuad.OpCode);
                                }
                                ProgramCounter = QuadTable.NextQuad();
                                break;
                            // DIV
                            // Compute op1 / op2, place result into op3
                            case DIV:
                                if (TraceOn)
                                {
                                    PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op1, CurrentQuad.Op2,
                             CurrentQuad.Op3);
                                }
                                double quotient = Convert.ToDouble(SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue
                ()) / SymbolTable.GetSymbol(CurrentQuad.Op2).GetValue();
                                SymbolTable.UpdateSymbol(CurrentQuad.Op3, SymbolKind.Variable, quotient);
                                ProgramCounter++;
                                break;
                            // MUL
                            // Compute op1 * op2, place result into op3
                            case MUL:
                                if (TraceOn)
                                {
                                    PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op1, CurrentQuad.Op2, CurrentQuad.Op3);
                                }
                                SymbolTable.UpdateSymbol(CurrentQuad.Op3, SymbolKind.Variable,
                                    (SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue() * SymbolTable.GetSymbol(CurrentQuad.Op2).GetValue()));
                                ProgramCounter++;
                                break;
                            // SUB
                            // Compute op1 - op2, place result into op3
                            case SUB:
                                if (TraceOn)
                                {
                                    PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op1, CurrentQuad.Op2, CurrentQuad.Op3);
                                }
                                SymbolTable.UpdateSymbol(CurrentQuad.Op3, SymbolKind.Variable,
                                    (SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue() - SymbolTable.GetSymbol(CurrentQuad.Op2).GetValue()));
                                ProgramCounter++;
                                break;
                            // ADD
                            // Compute op1 + op2, place result into op3
                            case ADD:
                                if (TraceOn)
                                {
                                    PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op1, CurrentQuad.Op2, CurrentQuad.Op3);
                                }
```

```
                                SymbolTable.UpdateSymbol(CurrentQuad.Op3, SymbolKind.Variable,
                                    (SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue() + SymbolTable.GetSymbol(CurrentQuad.Op2).GetValue()));
                                ProgramCounter++;
                                break;
                            // MOV
                            // Assign the value in op1 into op3 (op2 is ignored here)
                            case MOV:
                                if (TraceOn)
                                {
                                    PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op1, CurrentQuad.Op3);
                                }
                                SymbolTable.UpdateSymbol(CurrentQuad.Op3, SymbolKind.Variable, SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue());
                                ProgramCounter++;
                                break;
                            // STI
                            // Store indexed - Assign the value in op1 into op2 + offset op3
                            case STI:
                                if (TraceOn)
                                {
                                    PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op1, CurrentQuad.Op2, CurrentQuad.Op3);
                                }
                                SymbolTable.UpdateSymbol((CurrentQuad.Op2 + CurrentQuad.Op3), SymbolKind.Variable, SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue());
                                ProgramCounter++;
                                break;
                            // LDI
                            // Load indexed- Assign the value in op1 + offset op2, into op3
                            case LDI:
                                if (TraceOn)
                                {
                                    PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op1, CurrentQuad.Op2, CurrentQuad.Op3);
                                }
                                SymbolTable.UpdateSymbol(CurrentQuad.Op3, SymbolKind.Variable, SymbolTable.GetSymbol(CurrentQuad.Op1 + CurrentQuad.Op2).GetValue());
                                ProgramCounter++;
                                break;
                            // BNZ
                            // Branch Not Zero; if op1 value <> 0, set program counter to op3
                            case BNZ:
                                if (TraceOn)
                                {
                                    PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op3);
                                }
                                if (SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue() != 0)
                                {
                                    ProgramCounter = CurrentQuad.Op3;
                                }
                                else
                                {
                                    ProgramCounter++;
                                }
                                break;
                            // BNP
                            // Branch Not Positive; if op1 value <= 0, set program counter to op3
                            case BNP:
                                if (TraceOn)
                                {
                                    PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op3);
                                }
                                if (SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue() <= 0)
                                {
                                    ProgramCounter = CurrentQuad.Op3;
                                }
                                else
                                {
                                    ProgramCounter++;
                                }
                                break;
                            // BNN
                            // Branch Not Negative; if op1 value >= 0, set program counter to op3
                            case BNN:
                                if (TraceOn)
                                {
```

```
                        PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op3);
                    }
                    if (SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue() >= 0)
                    {
                        ProgramCounter = CurrentQuad.Op3;
                    }
                    else
                    {
                        ProgramCounter++;
                    }
                    break;
                // BZ
                // Branch Zero; if op1 value = 0, set program counter to op3
                case BZ:
                    if (TraceOn)
                    {
                        PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op3);
                    }
                    if (SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue() == 0)
                    {
                        ProgramCounter = CurrentQuad.Op3;
                    }
                    else
                    {
                        ProgramCounter++;
                    }
                    break;
                // BP
                // Branch Positive; if op1 value > 0, set program counter to op3
                case BP:
                    if (TraceOn)
                    {
                        PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op3);
                    }
                    if (SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue() > 0)
                    {
                        ProgramCounter = CurrentQuad.Op3;
                    }
                    else
                    {
```

```
                                ProgramCounter++;
                            }
                            break;
                        // BN
                        // Branch Negative; if op1 value < 0, set program counter to op3
                        case BN:
                            if (TraceOn)
                            {
                                PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op3);
                            }
                            if (SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue() < 0)
                            {
                                ProgramCounter = CurrentQuad.Op3;
                            }
                            else
                            {
                                ProgramCounter++;
                            }
                            break;
                        // BR
                        // Branch (unconditional); set program counter to op3
                        case BR:
                            if (TraceOn)
                            {
                                PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op3);
                            }
                            ProgramCounter = CurrentQuad.Op3;
                            break;
                        // BINDR
                        // Branch (unconditional); set program counter to op3 value contents (indirect)
                        case BINDR:
                            if (TraceOn)
                            {
                                PrintTrace(CurrentQuad.OpCode, SymbolTable.GetSymbol(CurrentQuad.Op3).GetValue
                         ());
                            }
                            ProgramCounter = SymbolTable.GetSymbol(CurrentQuad.Op3).GetValue();
                            break;
                        // PRINT
```

```csharp
                            // Write symbol table name and value of op 1
                            case PRINT:
                                if (TraceOn)
                                {
                                    PrintTrace(CurrentQuad.OpCode, CurrentQuad.Op1);
                                }
                                // Console.WriteLine($"{ SymbolTable.GetSymbol(CurrentQuad.Op1).Name} = {SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue()}");
                                Console.WriteLine($"{SymbolTable.GetSymbol(CurrentQuad.Op1).GetValue()}");
                                ProgramCounter++;
                                break;
                            default:
                                Console.WriteLine($"Invalid Opcode {CurrentQuad.OpCode}");
                                break;
                        }
                    }
                    // Catches any exception, prints the appropriate error message, and stops running the current program.
                    catch (Exception e)
                    {
                        Console.WriteLine("FATAL ERROR: " + e.Message + "\n");
                        ProgramCounter = QuadTable.NextQuad();
                    }
                }
            }
        }
```