

```
793     /// <summary>
794     /// Implements CFG Rule: <simple expression> -> [<sign>] <term> {<addop> <term>}*
795     /// </summary>
796     /// <returns>The result of the expression.</returns>
797     private int SimpleExpression()
798     {
799         if (IsError)
800             return -1;
801
802         Debug(true, "SimpleExpression()");
803
804         int left, right, temp, opcode;
805         int signVal = 0;
806
807         if (isSign())
808         {
809             signVal = Sign();
810         }
811
812         left = Term();
813
814         if (signVal == -1)
815         {
816             Quads.AddQuad(MULTIPLY, left, Minus1Index, left);
817         }
818
819         while (isAddOp() && !IsError)
820         {
821             opcode = AddOp();
822             right = Term();
823             temp = GenSymbol();
824             Quads.AddQuad(opcode, left, right, temp);
825             left = temp;
826         }
827
828         Debug(false, "SimpleExpression()");
829         return left;
830     }
831
832     /// <summary>
```

```
833     /// Implements CFG Rule: <term> -> <factor> {<mulop> <factor> }*
834     /// </summary>
835     /// <returns></returns>
836     private int Term()
837     {
838         if (IsError)
839             return -1;
840         int left, right, opCode, temp;
841
842         Debug(true, "Term()");
843         left = Factor();
844
845         while (isMulOp() && !IsError)
846         {
847             opCode = MulOp();
848             right = Factor();
849             temp = GenSymbol();
850             try
851             {
852                 Quads.AddQuad(opCode, left, right, temp);
853             }
854             catch (DivideByZeroException e)
855             {
856                 Console.WriteLine(e.Message);
857             }
858             left = temp;
859         }
860
861         Debug(false, "Term()");
862         return left;
863     }
864
865     /// <summary>
866     /// Implements CFG Rule: <factor> -> <unsigned constant> | <variable> | $LPAR <simple expression> $RPAR
867     /// </summary>
868     /// <returns></returns>
869     private int Factor()
870     {
871         if (IsError)
```

```
872         return -1;
873
874         Debug(true, "Factor()");
875
876         int index = 0;
877
878         if (isUnsignedConstant())
879         {
880             index = UnsignedConstant();
881         }
882         else if (isVariable())
883         {
884             index = Variable();
885         }
886         else if (Scanner.TokenCode == LPAR)
887         {
888             GetNextToken();
889             index = SimpleExpression();
890             if (Scanner.TokenCode == RPAR)
891                 GetNextToken();
892             else
893                 UnexpectedTokenError("RPAR");
894         }
895         else
896             UnexpectedTokenError("UNSIGNED CONSTANT or VARIABLE or LPAR");
897
898         Debug(false, "Factor()");
899         return index;
900     }
```