

```

bool CheckExpr (String E, int &i, NPType &LO, NPType &RO)
// check if the substring between E[i] (an opening bracket)
// and the corresponding closing bracket is a DNA expression;
// pre: E satisfies conditions related to the opening brackets
//      and the closing brackets;
//      E[i] is an opening bracket
// post: if return value is true, then i is position past the closing bracket
//      mentioned above
{ Symbol Oper;
  int Operi, // position of the operator
      n; // number of arguments of the operator
        // consecutive N-word-arguments are counted as 1 argument
  bool OK;
  NPType L1, R1; // types of leftmost and rightmost nucleotide pair
                // of an argument of the operator

  i = NextIndexUsed[i]; // go to next symbol (past the opening bracket)
  Oper = E[i]; // by assumption, Oper is indeed an operator
  Operi = i; // remember this position

  i = NextIndexUsed[i]; // go to next symbol (past the operator)
  OK = true;
  n=0;
  while (OK && (E[i] != Closebr))
    // invariant: E[i-1] is not an opening bracket
    // this is true at first iteration (as E[i-1] is an operator then)
    // this is true at subsequent iterations (as E[i-1] is an N-word
    // or a closing bracket then)
    // consequently, E[i] is not an operator
  { n++; // another argument

    if ((Oper==Uda) && (n>1))
    { cout << "Operator Uda with more than one argument at position "
        << Operi << ".\n";
      OK = false;
    }
    else // this argument is in principle allowed

    { if (WordElt(E[i])) // this argument is an N-word
      { do
        i = NextIndexUsed[i];
        while (WordElt(E[i]));

        switch (Oper)
        { case Upa: L1 = Pp;
              R1 = Pp;
              break;
          case Doa: L1 = Pm;
              R1 = Pm;
              break;
          case Uda: L1 = P;
              R1 = P;
        } // switch
      }
    else // E[i] is not a closing bracket (by while condition),
        // E[i] is not an operator (by invariant)
        // E[i] is not an N-word (by this else)
        // consequently, E[i] is an opening bracket:
        // this argument is a DNA expression
      OK = CheckExpr (E, i, L1, R1);
    }
  }
}

```

```

if (OK) // nothing went wrong yet
  if (Oper==Uda) // apparently, n==1
  { LO = P;
    RO = P;
  }
else // operator = Upa or Doa
  if (n==1) // first argument
  { LO = L1;
    RO = R1;
  }
else // n>1
  switch (Oper)
  { case Upa: if ((R0!=Pm) && (L1!=Pm)) // upper prefit
              RO = R1;
            else // not an upper prefit
            { cout << n-1
              << "-th Argument is not an upper prefit for "
              << n
              << "-th argument of operator Upa\n"
              << " at position "
              << Operi << ".\n";
            }
            OK = false;
          }
        break;
      case Doa: if ((R0!=Pp) && (L1!=Pp)) // lower prefit
                RO = R1;
              else // not a lower prefit
              { cout << n-1
                << "-th Argument is not a lower prefit for "
                << n
                << "-th argument of operator Doa\n"
                << " at position "
                << Operi << ".\n";
              }
            OK = false;
          }
    } // switch

  } // else: argument in principle allowed

} // while

if (OK) // E[i] is a closing bracket
  if (n==0)
  { cout << "Operator with no arguments at position " << Operi << ".\n";
    OK = false;
  }

if (OK)
  i = NextIndexUsed[i]; // go to next symbol (past the closing bracket)

return OK;

} // CheckExpr

```