```mql4
#property copyright ""
#property link ""
#property strict

#define RETRY_COUNT 3


enum timeframe {
   CURRENT = 0, // チャートの時間足
   M1      = 1,
   M5      = 5,
   M15     = 15,
   M30     = 30,
   H1      = 60,
   H4      = 240,
   D1      = 1440,
   W1      = 10080,
   MN1     = 43200
};

input int Magic1 = 1;
input int Magic2 = 2;

input double Lots       = 0.1;
input double StopLoss   = 0;
input double TakeProfit = 0;
input int    Slippage   = 10;
input string comment    = "PIVOT ナンピンマーチン";

input bool isNanpin = true;
input int NanpinCount = 5;
input double NanpinInterval = 20;
input double NanpinMult = 2;
input double NanpinTP = 20;
input double NanpinSL = 150;
input double NanpinAdd = 0;
input int Pivot_method_1_1_1 = 1; //PIVOTロング


input int Pivot_method_2_1_1 = 4; //PIVOTショート



bool Trade = true;
datetime lastAlertTime;
color ArrowColor[2] = {clrBlue, clrRed};
int magic_array[2];
int bars1;
int bars2;


//+------------------------------------------------------------------+
//初期化処理

//+------------------------------------------------------------------+
int OnInit()
{
   if(IsTradeAllowed() == false) {
      Alert("Enable the setting 'Allow live trading' in the Expert Properties!");
   }

   magic_array[0] = Magic1;
   magic_array[1] = Magic2;
   bars1 = getBars(Magic1);
   bars2 = getBars(Magic2);

   return(INIT_SUCCEEDED);
}


//+------------------------------------------------------------------+
//メイン処理

//+------------------------------------------------------------------+
void OnTick()
```

```
{

   int signal = 0;
   double lots = Lots;
   Trade = true;

   if(isNanpin == true) NanpinLogic(NanpinCount, NanpinInterval, NanpinMult, NanpinTP,
NanpinSL, NanpinAdd, magic_array);


   if(Trade == false) return;
   signal = 0;
   if((iHigh(Symbol(), PERIOD_M1, 1) + iLow(Symbol(), PERIOD_M1, 1) + iClose(Symbol(),
PERIOD_M1, 1)) / 3 * 2 - iHigh(Symbol(), PERIOD_M1, 1)  >=  iClose(Symbol(), 0, 1)+
PipsToPrice(0)) signal = 1;
   if(bars1 == Bars) signal = 0;


   if(signal != 0 && getOpenLots(Magic1) == 0) {

      if(openPosition(signal, lots, TakeProfit, StopLoss, Magic1)) {
         bars1 = Bars;
      }
   }
   signal = 0;
   if((iHigh(Symbol(), PERIOD_M1, 1) + iLow(Symbol(), PERIOD_M1, 1) + iClose(Symbol(),
PERIOD_M1, 1)) / 3 * 2 - iLow(Symbol(), PERIOD_M1, 1)  <=  iClose(Symbol(), 0, 1)+
PipsToPrice(0)) signal = -1;
   if(bars2 == Bars) signal = 0;


   if(signal != 0 && getOpenLots(Magic2) == 0) {

      if(openPosition(signal, lots, TakeProfit, StopLoss, Magic2)) {
         bars2 = Bars;
      }
   }

}


//+------------------------------------------------------------------------------------------------+
//|ロット数取得関数
//|    処理:引数に指定した単数ポジションのロット数を取得
//|    引数:ロット数を取得するポジションのマジックナンバー,売買種別
//|    戻り値:ポジションのロット数

//+------------------------------------------------------------------------------------------------+
double getOpenLots(int magic, int type = -1)
{
   double lots = 0;

   for(int i = OrdersTotal() - 1; i >= 0; i--) {
      if(OrderSelect(i, SELECT_BY_POS) == false) return(-1);
      if(OrderSymbol() != Symbol() || OrderMagicNumber() != magic) continue;

      if(type == -1 || OrderType() == type) lots += OrderLots();
   }

   return(lots);
}


//+------------------------------------------------------------------------------------------------+
//|ロット数取得関数
//|    処理:引数に指定した複数ポジションのロット数を取得
//|    引数:ロット数を取得するポジションのマジックナンバー(,区切りで指定),売買種別
//|    戻り値:指定したポジションのロット数

//+------------------------------------------------------------------------------------------------+
double getOpenMultipleLots(string magicstring, int type = -1)
{
   double lots = 0;
   string magics[];
   int    sep_num;
```

```
   sep_num = StringSplit(magicstring , ',' , magics);

   if(sep_num>0){
      for(int i = OrdersTotal() - 1; i >= 0; i--) {
         if(OrderSelect(i, SELECT_BY_POS) == false) return(-1);
         if(OrderSymbol() != Symbol() || !checkMagicNumbers(magics,OrderMagicNumber()))
continue;

         if(type == -1 || OrderType() == type) lots += OrderLots();
      }
   }

   return(lots);
}


//+------------------------------------------------------------------------------------------------------------------+
//|ロット数調整用関数
//|    処理:ロット数が最大値・最小値の範囲から外れていれば最大値・最小値にし、ステップ幅に応じて小数点以下を丸め、変更後のロット数を返す
//|    引数:ロット数
//|    戻り値:調整されたロット数

//+------------------------------------------------------------------------------------------------------------------+
double checkLots(double lots)
{
   double min_lots = MarketInfo(Symbol(), MODE_MINLOT);
   double max_lots = MarketInfo(Symbol(), MODE_MAXLOT);
   double step     = MarketInfo(Symbol(), MODE_LOTSTEP);

   if(lots < min_lots) lots = min_lots;
   if(lots > max_lots) lots = max_lots;

   if(step == 1) {
      lots = NormalizeDouble(lots, 0);
   }
   else if(step == 0.1) {
      lots = NormalizeDouble(lots, 1);
   }
   else {
      lots = NormalizeDouble(lots, 2);
   }

   return(lots);
}


//+------------------------------------------------------------------------------------------------------------------+
//|エントリー注文送信用関数
//|    処理:エントリー注文に必要な数値が正しいかを確認し、エントリー注文を送信する。
//|    引数:売買種別(買い:正の整数, 売り:負の整数),ロット数,テイクプロフィット,ストップロス,ポジションのマジックナンバー
//|    戻り値:処理結果(true:注文成功, false:注文失敗)

//+------------------------------------------------------------------------------------------------------------------+
bool openPosition(int signal, double lots, double tp_pips, double sl_pips, int magic)
{
   int    type = 0, ticket = 0;
   double price = 0, sl = 0, tp = 0;

   lots = checkLots(lots);

   switch(signal) {
      case 1:
         type = OP_BUY;
         price = Ask;
         if(tp_pips > 0) tp = Ask + PipsToPrice(tp_pips);
         if(sl_pips > 0) sl = Ask - PipsToPrice(sl_pips);
         break;
      case -1:
         type = OP_SELL;
         price = Bid;
         if(tp_pips > 0) tp = Bid - PipsToPrice(tp_pips);
         if(sl_pips > 0) sl = Bid + PipsToPrice(sl_pips);
         break;
      default:
```

```
            return(false);
            break;
      }

      if(marginCheck(type, lots, price) == false) return(false);
      for(int i = 0; i < RETRY_COUNT; i++) {
         if(IsTradeAllowed() == true) {
            RefreshRates();
            ticket = OrderSend(Symbol(), type, lots, price, Slippage, sl, tp, comment, magic, 0
, ArrowColor[type]);

            int err = GetLastError();
            if(err == 0) return(true);
            if(err > 0){
               Print("[OrderSend Error] : ", err, " ", ErrorDescription(err));
            }
            if(IsTesting() == true) break;
            if(err == 129 || err == 130 || err == 131 || err == 134) break;
         }

         if(i + 1 == RETRY_COUNT && ticket == -1) {
            Alert("[OrderSend Error] : Order timeout. Check the experts log.");
            return(false);
         }

         Sleep(200);
      }

      return(false);
}



//-------------------------------------------------------------------------------------------------
//|ポジション情報（テイクプロフィット，ストップロス）変更用関数
//|    処理：オープンポジションのテイクプロフィット、ストップロスの変更注文を送信する。
//|    引数：テイクプロフィット，ストップロス，変更するポジションのマジックナンバー
//|    戻り値：処理結果（true：注文成功， false：注文失敗）

//-------------------------------------------------------------------------------------------------
bool setTPSL(double tp, double sl, int ticket)
{
      if(OrderSelect(ticket, SELECT_BY_TICKET) == false) return(false);

      if(tp == 0) tp = OrderTakeProfit();
      if(sl == 0) sl = OrderStopLoss();

      if(MathAbs(tp - OrderTakeProfit()) < 0.0000001 && MathAbs(sl - OrderStopLoss()) <
0.0000001) return(false);

      for(int i = 0; i < RETRY_COUNT; i++) {
         if(IsTradeAllowed() == true) {
            if(OrderModify(ticket, OrderOpenPrice(), sl, tp, 0, clrNONE) == true) return(true);

            int err = GetLastError();
            Print("[OrderModify Error] : ", err, " ", ErrorDescription(err));
            if(IsTesting() == true) break;
            if(err == 1 || err == 130) break;
         }

         if(i + 1 == RETRY_COUNT && ticket == -1) {
            Alert("[OrderModify Error] : Order timeout. Check the experts log.");
            return(false);
         }

         Sleep(200);
      }

      return(false);
}


//-------------------------------------------------------------------------------------------------
//ポジション決済用関数
//    処理：決済注文に必要な数値が正しいかを確認し決済注文を送信する。
```

```
//    引数:決済するポジションのマジックナンバー,売買種別
//    戻り値:処理結果(true:決済成功, false:決済失敗)

//+------------------------------------------------------------------------------------------------------------------+
bool closePosition(int magic, int type = -1)
{
   int result_count = 0;
   int close_signal = 0;

   if(IsTradeAllowed() == true) {
      for(int i = OrdersTotal() - 1; i >= 0; i--) {
         if(OrderSelect(i, SELECT_BY_POS) == false) continue;
         if(OrderSymbol() != Symbol() || OrderMagicNumber() != magic) continue;
         if(type >= 0 && OrderType() != type) continue;
         RefreshRates();

         if(OrderClose(OrderTicket(), OrderLots(), OrderClosePrice(), Slippage, ArrowColor[
OrderType()]) == true) {

            continue;
         }else{
            result_count = 0;
            for(int j = 0; j < RETRY_COUNT; j++) {
               Sleep(200);

               if(OrderClose(OrderTicket(), OrderLots(), OrderClosePrice(), Slippage,
ArrowColor[OrderType()]) == true) {
                  break;
               }else{
                  int err = GetLastError();
                  if(err > 0){
                     result_count++;
                     Print("[OrderClose Error] : ", err, " ", ErrorDescription(err));
                  }
               }
               if(result_count == RETRY_COUNT) {
                  close_signal++;
               }
            }
         }
      }
      if(close_signal == 0) return(true);
      Alert("[OrderClose Error] : Close timeout. Check the experts log.");
   }
   return(false);
}


//+------------------------------------------------------------------------------------------------------------------+
//価格換算用関数
//    処理:pipsを価格に換算する。
//    引数:換算する値
//    戻り値:調整された価格

//+------------------------------------------------------------------------------------------------------------------+
double PipsToPrice(double value)
{
   int mult = (Digits == 3 || Digits == 5) ? 10 : 1;
   return(value * Point * mult);
}


//+------------------------------------------------------------------------------------------------------------------+
//価格換算用関数
//    処理:価格の小数点値に換算する。
//    引数:換算する値
//    戻り値:調整された価格

//+------------------------------------------------------------------------------------------------------------------+
double PointToPrice(int value)
{
   return(value * Point);
}
```

```
//--------------------------------------------------------------------------------------------------------
//pips換算用関数
//    処理：価格をpipsに換算する。
//    引数：換算する値
//    戻り値：調整されたpips

//--------------------------------------------------------------------------------------------------------
double PriceToPips(double value)
{
   int mult = (Digits == 3 || Digits == 5) ? 10 : 1;
   return(value / Point/ mult);
}


//--------------------------------------------------------------------------------------------------------
//pips換算用関数
//    処理：指定した通貨ペアの小数点以下桁数をもとに価格をpipsに換算する。
//    引数：換算する値，換算する通貨ペア
//    戻り値：調整されたpips

//--------------------------------------------------------------------------------------------------------
double PriceToPipsWithSymbol(double value, string symbol)
{
   double Symbol_Digits = MarketInfo(symbol,MODE_DIGITS);
   int mult = (Symbol_Digits == 3 || Symbol_Digits == 5) ? 10 : 1;
   return(value / MarketInfo(symbol,MODE_POINT) / mult);
}


//--------------------------------------------------------------------------------------------------------
//損益取得関数
//    処理：ポジションの損益を取得。

//    引数：損益を取得するポジションのマジックナンバー(,区切りで指定)，売買種別(0：すべて，1：買いのみ，2：売りのみ)，is_all値(true：すべて

//    戻り値：価格で計算したポジションの損益

//--------------------------------------------------------------------------------------------------------
double getOrderProfit(string magicstring, int method, bool is_all)
{
   double profit = 0;
   string magics[];
   int    sep_num;

   sep_num = StringSplit(magicstring , ',' , magics);

   if(sep_num>0) {
      for(int i = OrdersTotal() - 1; i >= 0; i--) {
         if(OrderSelect(i, SELECT_BY_POS) == false) continue;
         if(is_all == false && !checkMagicNumbers(magics,OrderMagicNumber())) continue;
         if(method == 0){
            if(OrderType() == OP_BUY || OrderType() == OP_SELL) {
               profit += OrderProfit();
            }
         }else{
            if(method == 1 && OrderType() == OP_BUY) {
               profit += OrderProfit();
            }
            if(method == 2 && OrderType() == OP_SELL) {
               profit += OrderProfit();
            }
         }
      }
   }

   return(profit);
}


//--------------------------------------------------------------------------------------------------------
//損益取得関数
//    処理：引数に指定した複数ポジションのロット数を取得

//    引数：損益を取得するポジションのマジックナンバー(,区切りで指定)，売買種別(0：すべて，1：買いのみ，2：売りのみ)，is_all値(true：すべて
```

```
//    戻り値:pipsで計算したポジションの損益

//+------------------------------------------------------------------------------------------------------------------+
double getOrderProfitPips(string magicstring, int method, bool is_all)
{
   double profit = 0;
   string magics[];
   int    sep_num;

   sep_num = StringSplit(magicstring , ',' , magics);

   if(sep_num>0) {
      for(int i = OrdersTotal() - 1; i >= 0; i--) {
         if(OrderSelect(i, SELECT_BY_POS) == false) continue;
         if(is_all == false && !checkMagicNumbers(magics,OrderMagicNumber())) continue;
         if(method == 0){
            if(OrderType() == OP_BUY) {
               profit += PriceToPipsWithSymbol(MarketInfo(OrderSymbol(),MODE_BID) -
OrderOpenPrice(),OrderSymbol());
            }
            if(OrderType() == OP_SELL) {
               profit += PriceToPipsWithSymbol(OrderOpenPrice() - MarketInfo(OrderSymbol(),
MODE_ASK),OrderSymbol());
            }
         }else{
            if(method == 1 && OrderType() == OP_BUY) {
               profit += PriceToPipsWithSymbol(MarketInfo(OrderSymbol(),MODE_BID) -
OrderOpenPrice(),OrderSymbol());
            }
            if(method == 2 && OrderType() == OP_SELL) {
               profit += PriceToPipsWithSymbol(OrderOpenPrice() - MarketInfo(OrderSymbol(),
MODE_ASK),OrderSymbol());
            }
         }
      }
   }

   return(profit);
}


//+------------------------------------------------------------------------------------------------------------------+
//マジックナンバー比較用関数
//    処理:入力したマジックナンバーと保有中ポジションのマジックナンバーを複数調べる関数
//    引数:確認するポジションのマジックナンバー(配列),比較するマジックナンバー
//    戻り値:処理結果(true:一致, false:不一致)

//+------------------------------------------------------------------------------------------------------------------+
bool checkMagicNumbers(string& magics[], int OrderMagic)
{
   for(int i = ArraySize(magics) - 1; i >= 0; i--) {
      if(OrderMagic == magic_array[StrToInteger(magics[i])-1]) return (true);
   }
   return (false);
}


//+------------------------------------------------------------------------------------------------------------------+
//最終エントリーしたバー数取得処理
//    処理:最終エントリーしたバー数を取得する処理
//    引数:マジックナンバー
//    戻り値:最終エントリーしたバー数

//+------------------------------------------------------------------------------------------------------------------+
int getBars(int magic)
{
   datetime open_time = 0, close_time = 0, time;
   int bars = Bars, i;

   for(i = OrdersTotal() - 1; i >= 0; i--) {
      if(OrderSelect(i, SELECT_BY_POS) == false) continue;
      if(OrderMagicNumber() != magic || OrderSymbol() != Symbol()) continue;

      if(open_time < OrderOpenTime()) {
         open_time = OrderOpenTime();
```

```
      }
   }

   for(i = OrdersHistoryTotal() - 1; i >= 0; i--) {
      if(OrderSelect(i, SELECT_BY_POS, MODE_HISTORY) == false) continue;
      if(OrderMagicNumber() != magic || OrderSymbol() != Symbol()) continue;

      if(close_time < OrderOpenTime()) {
         close_time = OrderOpenTime();
      }
   }

   if(close_time == 0 && open_time == 0) return(0);

   time = (close_time > open_time) ? close_time : open_time;

   for(i = 0; i < Bars; i++) {
      if(time < Time[i]) {
         bars = Bars - i;
      }
      else {
         break;
      }
   }

   return(bars);
}


//-------------------------------------------------------------------------------------------------
//ポジション数取得関数
//    処理：引数に指定したマジックナンバー（単一）のポジション数を取得。
//    引数：ポジションのマジックナンバー，売買種別
//    戻り値：ポジションの合計数

//-------------------------------------------------------------------------------------------------
int getOpenPositions(int magic, int type = -1)
{
   int positions = 0, i;

   for(i = OrdersTotal() - 1; i >= 0; i--) {
      if(OrderSelect(i, SELECT_BY_POS) == false) continue;
      if(OrderMagicNumber() != magic) continue;

      if(type == -1 || OrderType() == type) positions++;
   }

   return(positions);
}


//-------------------------------------------------------------------------------------------------
//ポジション数取得関数
//    処理：引数に指定したマジックナンバー（複数）のポジション数を取得。
//    引数：ポジションのマジックナンバー（，区切りで指定），売買種別
//    戻り値：ポジションの合計数

//-------------------------------------------------------------------------------------------------
int getOpenMultiplePositions(string magicstring, int type = -1)
{
   int positions = 0, i;
   string magics[];
   int    sep_num;

   sep_num = StringSplit(magicstring , ',' , magics);

   if(sep_num>0){
      for(i = OrdersTotal() - 1; i >= 0; i--) {
         if(OrderSelect(i, SELECT_BY_POS) == false) continue;
         if(!checkMagicNumbers(magics,OrderMagicNumber())) continue;

         if(type == -1 || OrderType() == type) positions++;
      }
   }
```

```
   return(positions);
}


//+------------------------------------------------------------------------------------------------------------------+
//最終エントリーしたバーシフト数取得処理
//    処理：最終エントリーしたバーシフトを取得する
//    引数：通貨ペア，時間足，マジックナンバー
//    戻り値：最終エントリー時のバーシフト数

//+------------------------------------------------------------------------------------------------------------------+
int getLastEntryBar(string symbol, int period, int magic)
{
   int i;
   for(i = OrdersTotal() - 1; i >= 0; i--) {
      if(OrderSelect(i, SELECT_BY_POS) == false) continue;
      if(OrderMagicNumber() != magic) continue;

      if(OrderType() == OP_BUY || OrderType() == OP_SELL) {
         int shift = iBarShift(symbol, period, OrderOpenTime(), false);
         return(shift);
      }
   }
   for(i = OrdersHistoryTotal() - 1; i >= 0; i--) {
      if(OrderSelect(i, SELECT_BY_POS, MODE_HISTORY) == false) continue;
      if(OrderMagicNumber() != magic) continue;

      if(OrderType() == OP_BUY || OrderType() == OP_SELL) {
         int shift = iBarShift(symbol, period, OrderOpenTime(), false);
         return(shift);
      }
   }
   return(-1);
}


//+------------------------------------------------------------------------------------------------------------------+
//指定した時間のバーシフト数取得処理
//    処理：指定した時間のバーシフトを取得する
//    引数：通貨ペア，時間足，バーシフト数を取得する時間
//    戻り値：指定した時間のバーシフト数

//+------------------------------------------------------------------------------------------------------------------+
int getTimeSelectBar(string symbol, int period, string selectTime)
{
   string currentTime = TimeToStr(TimeCurrent(), TIME_MINUTES);
   string nowDate = TimeToStr(TimeCurrent(), TIME_DATE);
   datetime selectDateTime = StringToTime(nowDate + " " + selectTime);
   int backDayCount = 4;

   if(currentTime <= selectTime) {
      for(int i = 0; i < backDayCount; i++) {
         selectDateTime = selectDateTime - 86400;
         int shift = iBarShift(Symbol(), Period(), selectDateTime, true);
         if(shift == -1)continue;
         return(shift);
      }
   }
   for(int i = 0; i < backDayCount; i++) {
      selectDateTime = selectDateTime - 86400;
      int shift = iBarShift(Symbol(), Period(), selectDateTime, true);
      if(shift == -1)continue;
      return(shift);
   }
   return(-1);
}


//+------------------------------------------------------------------------------------------------------------------+
//ローソク足の構成要素(ヒゲ)算出処理
//    処理：入力されたローソク足の構成要素を算出する処理
//    引数：通貨ペア，時間足，ローソク足の位置，ローソク足の構成(0：上ヒゲ，1：実体，2：下ヒゲ，3：値幅)
//    戻り値：指定したローソク足の構成から算出したpips

//+------------------------------------------------------------------------------------------------------------------+
```

```
double getCandleStickPips(string symbol, int period, int shift, int hige)
{
   int digits   = (int)MarketInfo(symbol, MODE_DIGITS);
   double open  = iOpen(symbol,period,shift);
   double close = iClose(symbol,period,shift);
   double high  = iHigh(symbol,period,shift);
   double low   = iLow(symbol,period,shift);

   // 陽線の場合
   if (open < close) {
      if (hige == 0) {
         return (NormalizeDouble(PriceToPips(MathAbs(high - close)), digits));
      }
      else if (hige == 1) {
         return (NormalizeDouble(PriceToPips(MathAbs(close - open)), digits));
      }
      else if (hige == 2) {
         return (NormalizeDouble(PriceToPips(MathAbs(open - low)), digits));
      }
      else if (hige == 3) {
         return (NormalizeDouble(PriceToPips(MathAbs(high - low)), digits));
      }
      else {
         return (NormalizeDouble(PriceToPips(MathAbs(high - close)), digits));
      }
   }
   // 陰線の場合
   else {
      if (hige == 0) {
         return (NormalizeDouble(PriceToPips(MathAbs(high - open)), digits));
      }
      else if (hige == 1) {
         return (NormalizeDouble(PriceToPips(MathAbs(open - close)), digits));
      }
      else if (hige == 2) {
         return (NormalizeDouble(PriceToPips(MathAbs(close - low)), digits));
      }
      else if (hige == 3) {
         return (NormalizeDouble(PriceToPips(MathAbs(high - low)), digits));
      }
      else {
         return (NormalizeDouble(PriceToPips(MathAbs(high - open)), digits));
      }
   }
   return 0;
}


//--------------------------------------------------------------------------------------------------
//テクニカル指標RCI算出処理
//    処理:RCIを算出する処理
//    引数:通貨ペア,時間足,ローソク足の本数,ローソク足の位置
//    戻り値:RCI

//--------------------------------------------------------------------------------------------------
double iRCI(string symbol, int period, int number, int shift)
{
    int rank;
    double d = 0;
    double close_price[];
    ArrayResize(close_price, number);

    for (int i = 0; i < number; i++) {
        close_price[i] = iClose(symbol, period, i+shift);
    }

    ArraySort(close_price, WHOLE_ARRAY, 0, MODE_DESCEND);

    for (int i = 0; i < number; i++) {
        rank = ArrayBsearch(close_price,iClose(symbol, period, i+shift),WHOLE_ARRAY,0,
MODE_DESCEND);
        d += MathPow(i-rank,2);
    }

    return((1 - 6 * d / (number * (number * number - 1))) * 100);
```

```
}


//+------------------------------------------------------------------------------------------------------------------+
//注文送信時の余剰証拠金確認
//    処理:指定した引数でエントリーした場合の、余剰証拠金を確認する
//    引数:売買種別,ロット数,価格
//    戻り値:処理結果(true:問題なし, false:余剰証拠金不足)

//+------------------------------------------------------------------------------------------------------------------+
bool marginCheck(int type, double lots, double price)
{
   if(AccountFreeMarginCheck(Symbol(),type,lots) <= 0) {
      if(lastAlertTime != Time[0]) {
         Alert("[OrderSend Error] : Not enough money");
         lastAlertTime = Time[0];
      }
      return(false);
   }
   return(true);
}



//+------------------------------------------------------------------------------------------------------------------+

//ナンピン関数
//    処理:損益(pips単位)に応じてポジションの追加を行う
//    引数:ナンピンを行うポジションのマジックナンバー

//    戻り値:なし

//+------------------------------------------------------------------------------------------------------------------+
void NanpinLogic(
   int max,
   double interval,
   double mult,
   double tp,
   double sl,
   double add,
   int &magic[])
{
   double price , total, lots;
   int count, type;

   for(int i = 0; i < ArrayRange(magic, 0); i++) {
      getNanpinInfo(magic[i], type, price, count, total, lots);

      if(price > 0) {
         if((total > tp && tp != 0) || (total * -1 > sl && sl != 0)) {
            if(closePosition(magic[i], type) == true) return;
         }
      }

      if(count >= max) return;

      if(type == OP_BUY && Ask <= (price - PipsToPrice(interval))) {
         if(add == 0){
            openPosition(1, lots * MathPow(mult, count + 1), 0, 0, magic[i]);
         }
         if(add > 0){
            if(mult == 0 || mult == 1) {
               openPosition(1, lots + (add * (count + 1)), 0, 0, magic[i]);
            }
            else {
               openPosition(1, (lots * MathPow(mult, count + 1) + (add * (count + 1))), 0, 0
, magic[i]);
            }
         }
      }

      if(type == OP_SELL && Bid >= (price + PipsToPrice(interval))) {
         if(add == 0){
            openPosition(-1, lots * MathPow(mult, count + 1), 0, 0, magic[i]);
         }
```

```
         if(add > 0){
            if(mult == 0 || mult == 1) {
               openPosition(-1, lots + (add * (count + 1)), 0, 0, magic[i]);
            }
            else {
               openPosition(-1, (lots * MathPow(mult, count + 1) + (add * (count + 1))), 0,
0, magic[i]);
            }
         }
      }
   }
}


//-------------------------------------------------------------------------------------------------

//ナンピンの状態を取得する
//    処理：ナンピン中のポジション情報を取得する
//    引数：マジックナンバー、ポジションの種別、最終エントリー時の価格、
//          ナンピン数、合計損益、初期ロット

//    戻り値：なし

//-------------------------------------------------------------------------------------------------
void getNanpinInfo(
   int    magic,
   int    &type,
   double &entry_price,
   int    &count,
   double &total,
   double &lots)
{
   datetime first_time = 0;
   datetime last_time = 0;
   int i = 0;
   type = -1;
   entry_price = 0;
   count = 0;
   total = 0;
   lots = 0;

   for(i = 0; i < OrdersTotal(); i++) {
      if(OrderSelect(i, SELECT_BY_POS) == false) break;
      if(OrderSymbol() != Symbol() || OrderMagicNumber() != magic) continue;

      if(first_time == 0 || OrderOpenTime() < first_time) {
         type = OrderType();
         entry_price = OrderOpenPrice();
         first_time = OrderOpenTime();
         lots = OrderLots();
      }
   }

   if(first_time == 0) return;

   for(i = 0; i < OrdersTotal(); i++) {
      if(OrderSelect(i, SELECT_BY_POS) == false) break;
      if(OrderSymbol() != Symbol() || OrderMagicNumber() != magic) continue;
      if(OrderType() != type) continue;

      if(OrderOpenTime() > first_time) {
         if(OrderOpenPrice() < entry_price && OrderType() == OP_BUY) {
            entry_price = OrderOpenPrice();
         }
         if(OrderOpenPrice() > entry_price && OrderType() == OP_SELL) {
            entry_price = OrderOpenPrice();
         }
         count++;
      }

      if(OrderType() == OP_BUY) {
         total += PriceToPips(Bid - OrderOpenPrice());
      }
      if(OrderType() == OP_SELL) {
         total += PriceToPips(OrderOpenPrice() - Ask);
```

```
      }
   }
}


//+------------------------------------------------------------------+
//エラー内容確認
//    処理:エラー内容を文字列に変換する。
//    引数:エラーコード
//    戻り値:エラー内容

//+------------------------------------------------------------------+
string ErrorDescription(int error_code)
{
   string error_string;

   switch(error_code) {
      case 0:   error_string="no error";
break;
      case 1:   error_string="no error, trade conditions not changed";
break;
      case 2:   error_string="common error";
break;
      case 3:   error_string="invalid trade parameters";
break;
      case 4:   error_string="trade server is busy";
break;
      case 5:   error_string="old version of the client terminal";
break;
      case 6:   error_string="no connection with trade server";
break;
      case 7:   error_string="not enough rights";
break;
      case 8:   error_string="too frequent requests";
break;
      case 9:   error_string="malfunctional trade operation (never returned error)";
break;
      case 64:  error_string="account disabled";
break;
      case 65:  error_string="invalid account";
break;
      case 128: error_string="trade timeout";
break;
      case 129: error_string="invalid price";
break;
      case 130: error_string="invalid stops";
break;
      case 131: error_string="invalid trade volume";
break;
      case 132: error_string="market is closed";
break;
      case 133: error_string="trade is disabled";
break;
      case 134: error_string="not enough money";
break;
      case 135: error_string="price changed";
break;
      case 136: error_string="off quotes";
break;
      case 137: error_string="broker is busy (never returned error)";
break;
      case 138: error_string="requote";
break;
      case 139: error_string="order is locked";
break;
      case 140: error_string="long positions only allowed";
break;
      case 141: error_string="too many requests";
break;
      case 145: error_string="modification denied because order is too close to market";
break;
      case 146: error_string="trade context is busy";
break;
      case 147: error_string="expirations are denied by broker";
break;
```

```
      case 148: error_string="amount of open and pending orders has reached the limit";
break;
      case 149: error_string="hedging is prohibited";
break;
      case 150: error_string="prohibited by FIFO rules";
break;

      case 4000: error_string="no error (never generated code)";
break;
      case 4001: error_string="wrong function pointer";
break;
      case 4002: error_string="array index is out of range";
break;
      case 4003: error_string="no memory for function call stack";
break;
      case 4004: error_string="recursive stack overflow";
break;
      case 4005: error_string="not enough stack for parameter";
break;
      case 4006: error_string="no memory for parameter string";
break;
      case 4007: error_string="no memory for temp string";
break;
      case 4008: error_string="non-initialized string";
break;
      case 4009: error_string="non-initialized string in array";
break;
      case 4010: error_string="no memory for array' string";
break;
      case 4011: error_string="too long string";
break;
      case 4012: error_string="remainder from zero divide";
break;
      case 4013: error_string="zero divide";
break;
      case 4014: error_string="unknown command";
break;
      case 4015: error_string="wrong jump (never generated error)";
break;
      case 4016: error_string="non-initialized array";
break;
      case 4017: error_string="dll calls are not allowed";
break;
      case 4018: error_string="cannot load library";
break;
      case 4019: error_string="cannot call function";
break;
      case 4020: error_string="expert function calls are not allowed";
break;
      case 4021: error_string="not enough memory for temp string returned from function";
break;
      case 4022: error_string="system is busy (never generated error)";
break;
      case 4023: error_string="dll-function call critical error";
break;
      case 4024: error_string="internal error";
break;
      case 4025: error_string="out of memory";
break;
      case 4026: error_string="invalid pointer";
break;
      case 4027: error_string="too many formatters in the format function";
break;
      case 4028: error_string="parameters count is more than formatters count";
break;
      case 4029: error_string="invalid array";
break;
      case 4030: error_string="no reply from chart";
break;
      case 4050: error_string="invalid function parameters count";
break;
      case 4051: error_string="invalid function parameter value";
break;
      case 4052: error_string="string function internal error";
break;
```

```
      case 4053: error_string="some array error";
break;
      case 4054: error_string="incorrect series array usage";
break;
      case 4055: error_string="custom indicator error";
break;
      case 4056: error_string="arrays are incompatible";
break;
      case 4057: error_string="global variables processing error";
break;
      case 4058: error_string="global variable not found";
break;
      case 4059: error_string="function is not allowed in testing mode";
break;
      case 4060: error_string="function is not confirmed";
break;
      case 4061: error_string="send mail error";
break;
      case 4062: error_string="string parameter expected";
break;
      case 4063: error_string="integer parameter expected";
break;
      case 4064: error_string="double parameter expected";
break;
      case 4065: error_string="array as parameter expected";
break;
      case 4066: error_string="requested history data is in update state";
break;
      case 4067: error_string="internal trade error";
break;
      case 4068: error_string="resource not found";
break;
      case 4069: error_string="resource not supported";
break;
      case 4070: error_string="duplicate resource";
break;
      case 4071: error_string="cannot initialize custom indicator";
break;
      case 4072: error_string="cannot load custom indicator";
break;
      case 4073: error_string="no history data";
break;
      case 4074: error_string="not enough memory for history data";
break;
      case 4075: error_string="not enough memory for indicator";
break;
      case 4099: error_string="end of file";
break;
      case 4100: error_string="some file error";
break;
      case 4101: error_string="wrong file name";
break;
      case 4102: error_string="too many opened files";
break;
      case 4103: error_string="cannot open file";
break;
      case 4104: error_string="incompatible access to a file";
break;
      case 4105: error_string="no order selected";
break;
      case 4106: error_string="unknown symbol";
break;
      case 4107: error_string="invalid price parameter for trade function";
break;
      case 4108: error_string="invalid ticket";
break;
      case 4109: error_string="trade is not allowed in the expert properties";
break;
      case 4110: error_string="longs are not allowed in the expert properties";
break;
      case 4111: error_string="shorts are not allowed in the expert properties";
break;
      case 4200: error_string="object already exists";
break;
```

```
      case 4201: error_string="unknown object property";
break;
      case 4202: error_string="object does not exist";
break;
      case 4203: error_string="unknown object type";
break;
      case 4204: error_string="no object name";
break;
      case 4205: error_string="object coordinates error";
break;
      case 4206: error_string="no specified subwindow";
break;
      case 4207: error_string="graphical object error";
break;
      case 4210: error_string="unknown chart property";
break;
      case 4211: error_string="chart not found";
break;
      case 4212: error_string="chart subwindow not found";
break;
      case 4213: error_string="chart indicator not found";
break;
      case 4220: error_string="symbol select error";
break;
      case 4250: error_string="notification error";
break;
      case 4251: error_string="notification parameter error";
break;
      case 4252: error_string="notifications disabled";
break;
      case 4253: error_string="notification send too frequent";
break;
      case 4260: error_string="ftp server is not specified";
break;
      case 4261: error_string="ftp login is not specified";
break;
      case 4262: error_string="ftp connect failed";
break;
      case 4263: error_string="ftp connect closed";
break;
      case 4264: error_string="ftp change path error";
break;
      case 4265: error_string="ftp file error";
break;
      case 4266: error_string="ftp error";
break;
      case 5001: error_string="too many opened files";
break;
      case 5002: error_string="wrong file name";
break;
      case 5003: error_string="too long file name";
break;
      case 5004: error_string="cannot open file";
break;
      case 5005: error_string="text file buffer allocation error";
break;
      case 5006: error_string="cannot delete file";
break;
      case 5007: error_string="invalid file handle (file closed or was not opened)";
break;
      case 5008: error_string="wrong file handle (handle index is out of handle table)";
break;
      case 5009: error_string="file must be opened with FILE_WRITE flag";
break;
      case 5010: error_string="file must be opened with FILE_READ flag";
break;
      case 5011: error_string="file must be opened with FILE_BIN flag";
break;
      case 5012: error_string="file must be opened with FILE_TXT flag";
break;
      case 5013: error_string="file must be opened with FILE_TXT or FILE_CSV flag";
break;
      case 5014: error_string="file must be opened with FILE_CSV flag";
break;
```

```
      case 5015: error_string="file read error";
break;
      case 5016: error_string="file write error";
break;
      case 5017: error_string="string size must be specified for binary file";
break;
      case 5018: error_string="incompatible file (for string arrays-TXT, for others-BIN)";
break;
      case 5019: error_string="file is directory, not file";
break;
      case 5020: error_string="file does not exist";
break;
      case 5021: error_string="file cannot be rewritten";
break;
      case 5022: error_string="wrong directory name";
break;
      case 5023: error_string="directory does not exist";
break;
      case 5024: error_string="specified file is not directory";
break;
      case 5025: error_string="cannot delete directory";
break;
      case 5026: error_string="cannot clean directory";
break;
      case 5027: error_string="array resize error";
break;
      case 5028: error_string="string resize error";
break;
      case 5029: error_string="structure contains strings or dynamic arrays";
break;
      default:   error_string="unknown error";
   }
   return(error_string);
}
```