Listing 2. The Poker unit from Tahoe 5.

```pascal
Unit Poker;

interface

uses Cards;

const
 nothing       = 0;
 jacks         = 1;
 twoPair       = 2;
 threeKind     = 3;
 straight      = 4;
 flush         = 5;
 fullhouse     = 6;
 fourKind      = 7;
 straightflush = 8;
 royalflush    = 9;

type

{ The poker hand game as an object }
TPokerHand = Object
 deck   : TDeck;
 Balance : integer;
 cards : array[1..5] of TCard;
 held   : array[1..5] of Boolean;
 val    : 0..9;
 bet    : integer;
 procedure init;
 procedure deal;
 procedure hold(card:integer);
 procedure eval;
end;

implementation

{ Deal 5 cards face down from new deck}
procedure TPokerHand.init;
var i : integer;
begin
 deck.init;
 bet := 10;
 for i := 1 to 5 do
 begin
   cards[i].init;
```

```
     held[i] := false;
  end;
end;

{ Deal new cards for those not held }
procedure TPokerHand.deal;
var i : integer;begin
 for i := 1 to 5 do
    if not(held[i]) then
       cards[i].setVal(deck.nextCard);
end;

{ Hold or discard a particular card }
procedure TPokerHand.hold(card:integer);
begin
 held[card] := not(held[card]);
end;

{ Evaluate the hand }
procedure TPokerHand.eval;
var i,j,k,m : integer;
    diff,acesHigh : integer;
    flushed, straighted : boolean;
    Locard, HiCard : TCard;
    diffs : array[0..4] of boolean;
    MatchFace : array[1..2] of integer;
    MatchLen  : array[1..2] of integer;
    match : integer;
    matchedYet : boolean;
begin
 val := nothing;
 { Check for flush }
 flushed := true;
 for i := 1 to 5 do
    flushed := (cards[i].suit = cards[1].suit) and flushed;
 if flushed then
    val := flush;

 { Check for straight }
 HiCard.init;
 LoCard.init;
 LoCard.setval(52);
 for i := 1 to 5 do
 begin
    if cards[i].face > HiCard.face then
       HiCard := cards[i];
    if cards[i].face < LoCard.face then
```

```
      LoCard := cards[i];
  end;
  fillChar(diffs, sizeOf(diffs), 0);
  straighted := true;
  if (LoCard.face = 1) and (HiCard.face = 13) then
    AcesHigh := 1
  else
    AcesHigh := 0;
  for i := 1 to 5 do
  begin
    diff := AcesHigh + HiCard.face - cards[i].face;
    { Treat Ace as high card if necessary }
    if (cards[i].face = 1) and (Hicard.face = 13) then
diff := 0;
    if diff > 4 then
        straighted := false
    else
        if not diffs[diff] then
          diffs[diff] := true
        else
          straighted := false;
  end;

  if straighted then
    val := straight;
  { Check for straight flush, royal flush }
  if straighted and flushed then
    val := straightflush;
  if (val = straightflush) and (AcesHigh = 1) then
    val := royalflush;

  { Check for 2, 3 or four of a kind }
  match := 0;
  fillChar(matchLen, sizeOf(matchLen), 0);
  fillChar(matchFace, sizeOf(matchFace), 0);
  for i := 1 to 5 do
  begin
    for j := 1 to 5 do
    if (cards[i].face = cards[j].face) and (i <> j) then
    begin
      matchedYet := False;
      k := 1;
      while (k <= match) and not(MatchedYet) do
      begin
        if matchFace[k] = cards[i].face then
          matchedYet := True;
        inc(k);
```

```
        end;
        if not(MatchedYet) then
        begin
          inc(match);
          matchFace[match] := cards[i].face;
          inc(matchLen[match]);
        end
        else
          inc(matchLen[match]);
    end;

  end;

  { flush is better than 2 pairs }
  { Check for two pair, full house }
  { MatchLen will be n*(n-1) for any sequence of n }
  if (match=2) then
     if matchLen[1] + matchLen[2] = 8 then
       val := fullhouse
     else      if not flushed then val := twopair;

  if (match=1) then
  begin
     if (matchLen[1] = 2) and
        (matchFace[1] = 1) or (matchFace[1] > 10) then
       val := jacks;
     if (matchLen[1] = 6) then
       val := threekind;
     if (matchLen[1] = 12) then
       val := fourkind;
  end;
end;

begin
end.
```