3 Hand Ranking

3.1 Class Definition

A class has to be defined that handles the hand recognition.
An array of up to 7 cards is passed to the get-hand-value-function (pocket + common cards) and it returns the hand type (1-9) plus a unique value so it can be compared to other hands (the better the hand, the higher the value)

```
#include <stdio.h>
#include <iostream>
#include "SCHelpFunctions.h"

class SCRanking
{
    //v003, 2015.04.11 M.Buchty
    //      cleaned and switched to use help class
    //v002, 2014.12.02 M.Buchty
    //      version used to create SCLookUpTables
    //v001, 2014.11.11 M.Buchty
    //      first published version

public:
    int  GetHandValue(int Cards[7], int CardCountInput);

    //global data
    int HandType;            //9 = Royal-/StraightFlush, 8 = Four-of-a-Kind, 7 = FullHouse,
..., 2 = Pair, 1 = HighCard
    int HandValue;           //9xxxxxx = Royal-/StraightFlush, 8xxxxxx = Four-of-a-Kind, ...

private:
    //functions for GetHandValue
    void AnalyseInputData(int Cards[7], int CardCountInput);
    void AnalyseInputData2(int Cards[7]);
    int  CheckStraightFlush(int Length);
    int  CheckFourKind(int ValueOfQuad);
    int  CheckFullHouse(int ValueOfTriple);
    int  CheckFlush();
    int  CheckStraight();
    int  CheckThreeKind(int ValueOfTriple);
    int  CheckTwoPairs(int ValueOfFirstPair);
    int  CheckPair(int ValueOfPair);
    int  CheckHighCard();

    //help functions
    int  SortCompare(const void *a, const void *b);
    static void SortArray (int *Array, size_t Length);
    static int  Maximum (int Cards[], int Length);
    static int  Search(int Array[], int Length, int Entry);
    static int  SearchX(int Array[], int Length, int Entry, int FilteredPos);
    static int  Clean(int CardsValues[], int CardsColors[], int Length, int TargetColor, int
CardsCleaned[]);

    //global data
    int CardCount;              //number of cards
    int CardsSorted[7];         //up to 7 cards (value 0-51) sorted (highest first)
    int CardsValue[7];          //the value of these cards (0-12)
    int CardsColor[7];          //the color of these cards (0-3)
    int CardsPocketValue[2];    //the value of the pocket cards (0-12)
    int CardsPocketColor[2];    //the color of the pocket cards (0-3)
    int CardsCommonValue[5];    //the value of the common cards (0-12)
    int CardsCleaned[8];        //the value of the cards (0-12) that correspond to the color
that exists most
//                              //this color is indicated in "CountColorsMaxPos"
//                              //this array has an additional position at the end that will
be used on "CheckStraightFlush"
```

```
    int CountValues[13];        //array with count for each card value 0-12
    int CountColors[4];         //array with count for each card color 0-3
    int CountValuesMax;         //highest count of card values
    int CountColorsMax;         //highest count of card colors
    int CountValuesMaxPos;      //card value with highest count
//                              //if two max positions exist, the higher card value is
returned
    int CountValuesSecPos;      //card value with highest count if CountValuesMaxPos is
disregarded
    int CountColorsMaxPos;      //card color with highest count
//                              //as with card values, the higher value is returned (but it
does not matter, which)
    int CountCommonValues[13];  //array with count for each card value 0-12
    int CountCommonColors[4];   //array with count for each card color 0-3

    struct                      //help structure for std::sort
    {
        bool operator()(int a, int b) {return a > b;}
    } SortGreater;

};
```

3.2 Get Hand Value Function

This is the only public function (being called from outside the class), all the others are privat (used internally after the main function is called).

```
//converts the input card array into a defined hand value
//this value can be used to check who has the best hand (highest value wins)
int SCRanking::GetHandValue(int Cards[], int CardCountInput)
{
    //IN:  Cards[7]          = 0-51            (local)
    //     CardCountInput    = 5-7
    //     CountColorsMax    = 2-7             (class global)
    //     CountValuesMax    = 1-4
    //     CountValuesMaxPos = 0-12
    //OUT: return            = 1211629-9000012 (local)
    //     HandValue         = 1211629-9000012 (class global)

    //v003, 2015.09.15 M.Buchty
    //      cleaned and switched to use help class
    //v002, 2014.12.02 M.Buchty
    //      version used to create SCLookUpTables
    //v001, 2014.11.08 M.Buchty
    //      first published version

    //check prepare input data
    AnalyseInputData(Cards, CardCountInput);

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

    //initialize return value
    //this function will end after the first check program detects anything and overwrites
this value
    HandValue = 0;

    if (CountColorsMax >= 5)
    {
        HandValue = CheckStraightFlush(CountColorsMax);
        if (HandValue != 0) { return HandValue; }                //stop and return if Royal
Flush or Straight Flush was found
        //                                                      //continue if trigger detected
Flush only
    } //check Royal Flush or Straight Flush

    if (CountValuesMax >= 4)
    {
        HandValue = CheckFourKind(CountValuesMaxPos);
        return HandValue;                                       //stop and return as a Four-
of-a-Kind must have been found
    } //check Four-of-a-Kind

    if (CountValuesMax >= 3)
    {
        HandValue = CheckFullHouse(CountValuesMaxPos);
        if (HandValue != 0) { return HandValue; }                //stop and return if Full
House was found
        //                                                      //continue if trigger detected
Three-of-a-Kind only
    } //check Full House

    if (CountColorsMax >= 5)
    {
        HandValue = CheckFlush();
        return HandValue;                                       //stop and return as a Flush
must have been found
    } //check Flush

    HandValue = CheckStraight();
    if (HandValue != 0) { return HandValue; }                    //stop and return if
Straight was found
    //                                                          //continue with anything else
```

```
    if (CountValuesMax >= 3)
    {
        HandValue = CheckThreeKind(CountValuesMaxPos);
        return HandValue;                                   //stop and return as a Three-
of-a-Kind must have been found
    } //check Three-of-a-Kind

    if (CountValuesMax >= 2)
    {
        HandValue = CheckTwoPairs(CountValuesMaxPos);
        if (HandValue != 0) { return HandValue; }             //stop and return if Two
Pairs were found
        //                                                  //continue if trigger detected
Pair only

        HandValue = CheckPair(CountValuesMaxPos);
        return HandValue;                                   //stop and return as a Pair
must have been found
    } //check TwoPair and Pair

    HandValue = CheckHighCard();
    return HandValue;                                       //if nothing was found, return
High Card
}
```

3.3 Analyse Input Data

```
//checks for correct input value on "Cards"
//breaks down card code 0-51 into values 0-12, colors 0-3
//counts values and colors, stores maximum counts
void SCRanking::AnalyseInputData(int Cards[], int CardCountInput)
{
    //IN:  Cards[7]              = 0-51 (local)
    //     CardCountInput        = 5-7
    //     SortGreater           = 0-1  (class global)
    //OUT: CardCount             = 5-7  (class global)
    //     CardsSorted[7]        = 0-51
    //     HandType              = 0
    //     HandPointer           = 0
    //     FlushDrawPointer      = 0
    //     StraightDrawpointer   = 0
    //     CardsValue[7]         = 0-51
    //     CountValues[13]       = 0-4
    //     CardsColors[7]        = 0-3
    //     CountColors[13]       = 0-7
    //     CountValuesMax        = 1-4
    //     CountValuesMaxPos     = 0-12
    //     CountValuesSecPos     = 0-12
    //     CountColorsMax        = 2-7
    //     CountColorsMaxPos     = 0-12
    //     CardsCleaned[7]       = 0-12

    //v003, 2015.09.15 M.Buchty
    //      cleaned and switched to use help class
    //v002, 2014.12.02 M.Buchty
    //      version used to create SCLookUpTables
    //v001, 2014.11.08 M.Buchty
    //      first published version

    // ------------------------------------------------------------------
    //check and sort input cards

    //copy number of cards in global variable
    CardCount = CardCountInput;

    if (SCHelp::CheckDoubleCards(Cards,CardCount))
        { SCHelp::ErrorMessage("SCRanking", "AnalyseInputData", "CheckDouble"); }

    //copy values to new array as the original array should not be changed (by sorting)
    for (int i=0; i < CardCount; i++)
    {
        //check if card-array contains valid entries only
        if (Cards[i] < 0 or Cards[i] > 51)
        {
            SCHelp::ErrorMessage("SCRanking", "AnalyseInputData", "Cards");
        } //only for debugging

        CardsSorted[i] = Cards[i];
    }

    //sort complete card array (highest first)
    //some detections later on require a sorted array
    std::sort(CardsSorted, CardsSorted + CardCount, SortGreater);

    // ------------------------------------------------------------------
    //main arrays

    //reset values (from previous hands)
    HandType         = 0;
    HandPointer      = 0;
    FlushDrawPointer = 0;
    XtraPointer      = 0;
    for (int i=0; i < 13; i++) { CountValues [i] =  0; }
    for (int i=0; i <  4; i++) { CountColors [i] =  0; }
    for (int i=0; i <  7; i++) { CardsCleaned[i] = -1; }
```

```
    //convert card code 0-51 into individual arrays with values, colors, etc
    for (int i=0; i < CardCount; i++)
    {
        int ActCard           = CardsSorted[i];
        int ActValue          = ActCard / 4;
        CardsValue[i]         = ActValue;
        CountValues[ActValue] ++;
        int ActColor          = ActCard - ActValue * 4;
        CardsColor[i]         = ActColor;
        CountColors[ActColor] ++;
    }
    //set not existing cards to -1 (easier for debugging when reviewing arrays)
    for (int i=CardCount; i < 7; i++)
    {
        CardsValue[i]         = -1;
        CardsSorted[i]        = -1;
    }

    // ------------------------------------------------------------------
    //calculate maxima

    //analyze the height of maxima (values and colors) and where they are
    CountValuesMax                = Maximum(CountValues, 13);
    CountValuesMaxPos             = Search(CountValues, 13, CountValuesMax);
    CountValuesSecPos             = -1; //will be written at CheckFullHouse or CheckTwoPair
    CountColorsMax                = Maximum(CountColors, 4);
    CountColorsMaxPos             = Search(CountColors, 4, CountColorsMax);

    //extract array of colors only (required for flush hands)
    int CountColorsMax2  = Clean(CardsValue, CardsColor, CardCount, CountColorsMaxPos,
CardsCleaned);
    if (CountColorsMax2 != CountColorsMax)
    {
        SCHelp::ErrorMessage("SCRanking", "AnalyseInputData", "CountColorsMax");
    } //only for debugging
}
```

3.4 Straight Flush

```
//check if the cards form a Royal or Straigth Flush
int SCRanking::CheckStraightFlush(int Length)
{
    //IN:  Length            = 0-7               (local)
    //     CardsCleaned[5-7] = 0-12              (class global)
    //     CountColorsMax    = 2-7
    //OUT: return            = 9000000 - 9000012 (local)
    //     HandType          = 9                 (class global)

    //v003, 2015.09.15 M.Buchty
    //      cleaned and switched to use help class
    //v002, 2014.12.02 M.Buchty
    //      version used to create SCLookUpTables
    //v001, 2014.11.08 M.Buchty
    //      first published version

    //start with first card
    int StartValue = CardsCleaned[0];           //pointer to highest card in the street
    int Count      = 1;                         //count number of cards in a row (this value
must reach 5 to form a straight)

    //duplicate Ace (copy to the end) as it can count as "One" as well
    if (StartValue == 12)
    {
        CardsCleaned[Length]  = -1;             //as "Two" corresponds number 0, the Ace as
"One" corresponds -1
        Length               +=  1;
    }

    //go through all other cards
    for(int i=1; i < CountColorsMax; i++)       //start at 1 as first card was already read
out
    {
        int ActValue = CardsCleaned[i];

        if (ActValue == CardsCleaned[i-1]-1)
        {
            Count += 1;
            if (Count >= 5)                     //if 5 cards in a row are found, return result
            {
                HandType         = 9;           //"9" means Straight Flush or Royal Flush
                int ReturnValue  = 9000000;
                ReturnValue     += StartValue;  //adds the value-code of the highest card
                return ReturnValue;             //return with success
            }
        }
        else
        {
            StartValue = ActValue;              //restart new search from this card on
            Count      = 1;
        }
    }

    return 0;                                   //return without success (no Straight, must be
Flush)
}
```

3.5 Four-of-a-Kind

```
//return code of the Four-of-a-Kind
int SCRanking::CheckFourKind(int ValueOfQuad)
{
    //IN:  ValueOfQuad     = 0-12              (local)
    //     CardsValue[5-7] = 0-12              (class global)
    //OUT: return          = 8000000 - 8xxxxxx (local)
    //     HandType        = 8                 (class global)

    //v003, 2015.09.15 M.Buchty
    //      cleaned and switched to use help class
    //v002, 2014.12.02 M.Buchty
    //      version used to create SCLookUpTables
    //v001, 2014.11.08 M.Buchty
    //      first published version

    HandType         = 8;                           //"8" means Four-of-a-Kind
    int ReturnValue  = 8000000;
    ReturnValue     += ValueOfQuad * 13;            //adds the value-code of the quad (on
position 13^1)

    for(int i=0; i < 5; i++)                        //the quad uses 4 cards, so (at latest)
the 5th must be the kicker
    {
        if (CardsValue[i] != ValueOfQuad)           //search kicker (highest card that does
not belong to the quad)
        {
            ReturnValue += CardsValue[i];           //adds the value-code of the kicker (on
position 13^0)
            return ReturnValue;                     //return with success
        }
    }

    SCHelp::ErrorMessage("SCRanking", "CheckFourKind", "End");
    return 0;                                       //return without success (should never
happen)
}
```

3.6 Full House

```
//check if the cards form a Full House and return code
int SCRanking::CheckFullHouse(int ValueOfTriple)
{
    //IN:  ValueOfTriple     = 0-12              (local)
    //     CountValues[13]   = 0-4               (class global)
    //OUT: return            = 7000000 - 7xxxxxx (local)
    //     HandType          = 7                 (class global)
    //     CountValuesSecPos = 0-12

    //v003, 2015.09.15 M.Buchty
    //      cleaned and switched to use help class
    //v002, 2014.12.02 M.Buchty
    //      version used to create SCLookUpTables
    //v001, 2014.11.08 M.Buchty
    //      first published version

    //search if a pair exists (could be taken out of a minor triple)
    int ValueOfPair = SearchX(CountValues, 13, 2, ValueOfTriple);
    if (ValueOfPair > -1)
    {
        HandType           = 7;                     //"7" means Full House
        CountValuesSecPos  = ValueOfPair;           //store value of pair for later use

        int ReturnValue    = 7000000;
        ReturnValue       += ValueOfTriple * 13;    //adds the value-code of the triple (on
position 13^1)
        ReturnValue       += ValueOfPair;           //adds the value-code of the pair (on
position 13^0)
        return ReturnValue;                         //return with success
    }

    return 0;                                       //return without success (no Full Fouse,
must be Three-of-a-Kind)
}
```

3.7 Flush

```
//return code of the Flush
int SCRanking::CheckFlush()
{
    //IN:  CardsCleaned[5-7] = 0-12              (class global)
    //OUT: return            = 6000000 - 6xxxxxx (local)
    //     HandType          = 6                 (class global)

    //v003, 2015.09.15 M.Buchty
    //      cleaned and switched to use help class
    //v002, 2014.12.02 M.Buchty
    //      version used to create SCLookUpTables
    //v001, 2014.11.08 M.Buchty
    //      first published version

    HandType        = 6;                        //"6" means Flush
    int ReturnValue = 6000000;

    int exp = 28561;                            //start with position 13^4
    for(int i=0; i <= 4; i++)
    {
        ReturnValue += CardsCleaned[i] * exp;   //adds the value-code of the card
        exp         /= 13;                      //decrease position (from 13^4 to 13^3 to 13^2
etc)
    }

    return ReturnValue;                         //return with success
}
```

3.8 Straight

```
//check if the cards form a Straight and return code
int SCRanking::CheckStraight()
{
    //IN:  CountValues[13]   = 0-4               (class global)
    //OUT: return            = 5000000 - 5xxxxxx (local)
    //     HandType          = 5                 (class global)

    //v003, 2015.09.15 M.Buchty
    //      cleaned and switched to use help class
    //v002, 2014.12.02 M.Buchty
    //      version used to create SCLookUpTables
    //v001, 2014.11.08 M.Buchty
    //      first published version

    int Count = 0;
    for(int i=12; i >= 0; i--)
    {
        if (CountValues[i] >= 1) { Count ++; }
        else                     { Count = 0; }       //count consecutive cards

        if (Count == 5)                               //write result, if count was successful
        {
            HandType         = 5;                     //"5" means Straight
            int ReturnValue  = 5000000;
            ReturnValue     += i+4;                   //adds the value-code of the highest card
(on position 13^0)
            return ReturnValue;                       //return with success
        }
    }

    if (Count == 4 and CountValues[12] >= 1)          //if the end was reached with cards 2-5
plus "Ace"
    {
        HandType = 5;                                 //"5" means Straight
        return 5000003;                               //return with Straight "Ace" to "Five"
    }

    return 0;                                         //return without success (no Straight,
could be anything lower)
}
```

3.9 Three-of-a-Kind

```
//return code of the Three-of-a-Kind
int SCRanking::CheckThreeKind(int ValueOfTriple)
{
    //IN:  ValueOfTriple   = 0-12              (local)
    //     CardsValue[5-7] = 0-7               (class global)
    //OUT: return          = 4000000 - 4xxxxxx (local)
    //     HandType        = 4                 (class global)

    //v003, 2015.09.15 M.Buchty
    //      cleaned and switched to use help class
    //v002, 2014.12.02 M.Buchty
    //      version used to create SCLookUpTables
    //v001, 2014.11.08 M.Buchty
    //      first published version

    HandType         = 4;                        //"4" means Triple
    int ReturnValue  = 4000000;
    ReturnValue     += ValueOfTriple * 169;      //adds the value-code of the Triple (on the
13^2 position)

    int exp = 13;
    for(int i=0; i < 5; i++)                     //the Triple uses 3 cards, so (at the latest)
        //                                               //                          the 4th and
5th must be the kicker
    {
        if (CardsValue[i] != ValueOfTriple)      //search kicker (that do not belong to the
quad)
        {
            ReturnValue += CardsValue[i] * exp; //adds the value-code of the kicker (on the
13^1 to 13^0 positions)
            exp         /= 13;                   //decrease position (from 13^4 to 13^3 to 13^2
etc)

            if (exp < 1)
            {
                return ReturnValue;              //return with success
            }
        }
    }

    SCHelp::ErrorMessage("SCRanking", "CheckThreeKind", "End");
    return 0;                                    //return without success (should never happen)
}
```

3.10 Two Pairs

```
//check if the cards form two pair and return code
int SCRanking::CheckTwoPairs(int ValueOfFirstPair)
{
    //IN:  ValueOfFirstPair  = 0-12              (local)
    //     CountValues[13]   = 0-4               (class global)
    //OUT: return            = 3000000 - 3xxxxxx (local)
    //     HandType          = 3                 (class global)
    //     CountValuesSecPos = 0-12

    //v003, 2015.09.15 M.Buchty
    //      cleaned and switched to use help class
    //v002, 2014.12.02 M.Buchty
    //      version used to create SCLookUpTables
    //v001, 2014.11.08 M.Buchty
    //      first published version

    //search if second pair exists
    int ValueOfSecondPair = SearchX(CountValues, 13, 2, ValueOfFirstPair);

    if (ValueOfSecondPair > -1)
    {
        HandType           = 3;                         //"3" means Two Pair
        CountValuesSecPos  = ValueOfSecondPair;         //store value of second pair for later
use

        int ReturnValue    = 3000000;
        ReturnValue       += ValueOfFirstPair * 169;    //adds the value-code of the first
pair (on the 13^2 position)
        ReturnValue       += ValueOfSecondPair * 13;    //adds the value-code of the second
pair (on the 13^1 position)

        for(int i=12; i >= 0; i--)                      //search the remaining kicker
        {
            if (CountValues[i] >= 1 and i != ValueOfFirstPair and i != ValueOfSecondPair)
            {
                ReturnValue += i;                       //adds the value-code of the kicker
(on the 13^0 position)
                return ReturnValue;                     //return with success
            }
        }
    }

    return 0;                                           //return without success (no Two Pair,
must be Pair)
}
```

3.11 Pair

```
//return code of the Pair
int SCRanking::CheckPair(int ValueOfPair)
{
    //IN:  ValueOfPair     = 0-12              (local)
    //     CardsValue[5-7] = 0-7               (class global)
    //OUT: return          = 2000000 - 2xxxxxx (local)
    //     HandType        = 2                 (class global)

    //v003, 2015.09.15 M.Buchty
    //      cleaned and switched to use help class
    //v002, 2014.12.02 M.Buchty
    //      version used to create SCLookUpTables
    //v001, 2014.11.08 M.Buchty
    //      first published version

    HandType         = 2;                       //"2" means Pair
    int ReturnValue  = 2000000;
    ReturnValue     += ValueOfPair * 2197;      //adds the value-code of the Pair (on the 13^3
position)

    int exp = 169;                              //first kicker starts on the 13^2 position
    for(int i=0; i < 5; i++)                    //the Pair uses 4 cards, so (at latest) the
3rd to 5th must be the kicker
    {
        if (CardsValue[i] != ValueOfPair)       //search kicker (that do not belong to the
pair)
        {
            ReturnValue += CardsValue[i] * exp; //adds the value-code of the kicker (on the
13^2 to 13^0 positions)

            exp         /= 13;                  //decrease position (from 13^4 to 13^3 to 13^2
etc)

            if (exp < 1)                        //if no kicker is left to be found
            {
                return ReturnValue;             //return with success
            }
        }
    }

    SCHelp::ErrorMessage("SCRanking", "CheckPair", "End");
    return 0;                                   //return without success (should never happen)
}
```

3.12 High Card

```
//return code of the High Card
int SCRanking::CheckHighCard()
{
    //IN:  CardsValue[5-7] = 0-7               (class global)
    //OUT: return          = 1000000 - 1xxxxxx (local)
    //     HandType        = 1                 (class global)

    //v003, 2015.09.15 M.Buchty
    //      cleaned and switched to use help class
    //v002, 2014.12.02 M.Buchty
    //      version used to create SCLookUpTables
    //v001, 2014.11.08 M.Buchty
    //      first published version

    HandType        = 1;                    //"1" means High Card
    int ReturnValue = 1000000;

    int exp = 28561;
    for(int i=0; i <= 4; i++)
    {
        ReturnValue += CardsValue[i] * exp; //adds the value-code of the card
        exp         /= 13;                  //decrease position (from 13^4 to 13^3 to 13^2
etc)
    }

    return ReturnValue;                     //return with success
}
```