

Contents

1	Introduction	1
2	Grading	1
3	Pictures Project	2
4	Poker Game Project	7
4.1	Example Program Runs	7
4.2	Project Specifications	10
4.3	Determining the Type of Hand	13
4.4	Determining the Winning Hand	13

1 Introduction

In this project you have a choice of one of two programs to write. One is on images that can be produced using complex numbers and the other is a multi-player game for 5-card draw poker (without betting). **You have the choice of one of two projects, do one and only one of these.** As it states in the syllabus.

Projects are to be done strictly on your own and as with all assignments the sharing of files and code is strictly prohibited and constitutes an act of Academic Misconduct. Furthermore the use of any electronic medium, such as code repositories, forums, blogs, message boards, email, etc. is strictly prohibited and constitutes an act of Academic Misconduct.

The only person you may discuss this with in any form is me. You may use the textbook, the textbook example code, and the class example code that is posted on the MyClasses site.

When you are ready to submit your work create a folder called `Project01`. Put all the code files needed for the project in the folder along with a makefile. Zip the entire `Project01` folder up into a single zip file and submit it.

2 Grading

The program itself should, of course, be nicely formatted and commented and should follow all the other rules of good programming style. Make sure you are following all the coding and documentation standards of the class that are published on the MyClasses site for this class.

The grading of the project will take two forms, a sample run and an inspection of the code. If the program does not run you will receive a zero for that portion. So even if the program is not complete you will get a better grade for a partial program that runs versus a program that does not run. So I would suggest a completion in stages approach. The run portion of the grading will test the user interface for usability and conforming to the specifications I have outlined above. The code inspection portion of the grade will involve commenting, readability, correct indentation, variable names, structure and style, correctness, and conforming to specifications.

I am looking for clean, easy to read, code. A good use of functions without overuse. Commenting that gets the point across concisely. An easy to use interface with nice looking output.

**You have a choice of one of two projects,
do one and only one of these.**

3 Pictures Project

This project is for you to build a Complex number class, that will handle the basic arithmetic functions, streaming, constructions, and some comparison functions for complex numbers.

A complex number is a number of the form $a + bi$ where a and b are real numbers and i is the imaginary unit, $i = \sqrt{-1}$. a is called the real part and b is called the imaginary part, note that b is the imaginary part and not bi . If we have two complex numbers $a + bi$ and $c + di$ then the four arithmetic functions are defined as,

$$\begin{aligned}(a + bi) + (c + di) &= (a + c) + (b + d)i \\(a + bi) - (c + di) &= (a - c) + (b - d)i \\(a + bi) \cdot (c + di) &= (ac - bd) + (bc + ad)i \\(a + bi)/(c + di) &= \frac{a + bi}{c + di} = \frac{ac + bd}{c^2 + d^2} + \frac{bc - ad}{c^2 + d^2}i\end{aligned}$$

In addition, the modulus of a complex number $a + bi$ is $\sqrt{a^2 + b^2}$.

Create the Complex class that has the following functions and overloads. The class must have both header and implementation files with all implementation code in the cpp file, that is, no inline code. Since complex numbers are not a native data type in C++ your class will store two doubles, one for the real part and one for the imaginary part. This is the only data you will need to store. The operations defined above will simply be manipulations of these values.

- Default constructor setting the value of the complex number to 0.
- Constructor taking the real and imaginary parts as parameters.
- Constructor taking just the real part and setting the imaginary part to 0.
- Destructor.
- Functions to get and set the real parts and imaginary parts.
- A function to set both the real and imaginary parts.
- Overload the $+$, $-$, $*$, and $/$ operators. Each operation will require three overloads. Complex and Complex, Complex and double, and finally double and Complex. So expressions like $z + z$, $z + 5$, and $5 + z$ will all work with each operation.
- Overload $==$ and $==$ for Complex and Complex.
- Overload the $^$ symbol for exponentiation with integer exponents. So z^5 or z^{-2} will work but you do not need to overload it for other types of exponents. If you look at the testing programs that were supplied you will notice that we put parentheses around the exponent expressions, for example, we write $(z^2)+c$ instead of just z^{2+c} . This is needed because the order of presidents of these symbols in C++ do not match the order of presidents in mathematics. In C++ the symbol $^$ is the logical XOR, logical connectors have a very low order of presidents, far below the other arithmetic operators (even below the streaming operators). Although C++ allows you to change the meaning of the arithmetic operators it does not allow you to change the presidents order of them. That is a good thing, think about the havoc that could result if you made $+$ a higher president than $*$. Although the reason we just gave for

not being able to alter the presidents order is most compelling, the real reason is tied up in computational theory and compiler/language design.

- Overload the logical operators == and != between two complex numbers. You do not need to overload any of the others since there is not a standard ordering of the complex number system.
- Overload the assignment operator, this is really not needed for this data type but is good practice.
- Create a mod function to return the modulus of the complex number. That is, z.mod() would be the modulus of z.
- Overload the input and output streaming operators. The output operator should print out $a + bi$ in a nice form. As in the examples below. The input operator should take two doubles with a space between them to load into the real and imaginary parts respectively.

Once your class is written, test it with the following program. You may not alter this program. Make sure that all of your functions are named correctly so that this testing program runs without editing. In the files distributed with this project you will find the `ComplexClassTester.cpp` file, which is the code below.

```
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <string>

#include "Complex.h"

void print(Complex z) {
    cout << z << endl;
}

int main() {
    Complex z;
    Complex x(3, 7);
    Complex y(-4);
    Complex w(0, 2);

    cout << x << ", " << y << ", " << z << ", " << w << endl;
    cout << x.getReal() << ", " << x.getImag() << endl;

    z.setReal(-5);
    z.setImag(19);
    cout << z << endl;

    z.set(-12345, 471);
    cout << z << endl;

    cout << endl;
    z.set(5, -8);
    cout << z << endl;
    cout << z + x << endl;
    cout << z + 4 << endl;
    cout << 4 + z << endl;
    cout << endl;
    cout << z - x << endl;
    cout << z - 4 << endl;
    cout << 4 - z << endl;
    cout << endl;
    cout << z * x << endl;
    cout << z * 4 << endl;
```

```

cout << 4 * z << endl;
cout << endl;
cout << z / x << endl;
cout << z / 4 << endl;
cout << 4 / z << endl;
cout << endl;

w = z;
cout << w << ", " << x << endl;
w += x;
print(w);

cout << endl;

w = z;
cout << w << ", " << x << endl;
w -= x;
print(w);

cout << (w ^ 2) << endl;
cout << (w ^ 3) << endl;
cout << endl;

cout << z.mod() << endl;
cout << endl;

if (w == z)
    cout << "w == z" << endl;
else
    cout << "w != z" << endl;

if (w != z)
    cout << "w != z" << endl;
else
    cout << "w == z" << endl;

cout << endl;

cout << "Input a complex number by simply entering " << endl;
cout << "the real and imaginary parts with a space " << endl;
cout << "between them. " << endl;
cout << "c = ";
cin >> z;
cout << "c = " << z << endl;

return 0;
}

```

Your output should look exactly like the run below.

```

3 + 7i, -4, 0, 2i
3, 7
-5 + 19i
-12345 + 471i

5 - 8i
8 - 1i
9 - 8i
9 - 8i

2 - 15i
1 - 8i
-1 + 8i

```

```

71 + 11i
20 - 32i
20 - 32i

-0.706897 - 1.01724i
1.25 - 2i
0.224719 + 0.359551i

5 - 8i, 3 + 7i
8 - 1i

5 - 8i, 3 + 7i
2 - 15i
-221 - 60i
-1342 + 3195i

9.43398

w != z
w != z

```

Input a complex number by simply entering the real and imaginary parts with a space between them.

```
c = 2 -7
c = 2 - 7i
```

Once you have this working, create a new directory, copy your complex class files over to the new directory, also copy the `bitmap_image.hpp` file to the new directory. This file is contained in the project files that accompany the project. For the main program use the following code with no alterations. This is also contained in the project files that accompany the project, it is in `ComplexImageCreator.cpp`.

```

#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <string>

#include "Complex.h"

#include "bitmap_image.hpp"

int main() {
    int size;
    double cr;
    double ci;
    unsigned int maxiter = 1000;
    double borderRatio;
    double colorExponent;
    double bailout;

    cout << "Input the image size (in pixels): ";
    cin >> size;
    cout << "Input the real value of c: ";
    cin >> cr;
    cout << "Input the imaginary value of c: ";
    cin >> ci;
    cout << "Input the maximum iteration (100-1,000): ";
    cin >> maxiter;
    cout << "Input the border ratio (0-1): ";
    cin >> borderRatio;
    cout << "Input the color exponent (0-1): ";
    cin >> colorExponent;
}

```

```

cout << "Input the bailout radius (>= 4): ";
cin >> bailout;

const unsigned int width = size;
const unsigned int height = size;

bitmap_image pic(width, height);
pic.clear();

Complex z, c(cr, ci);

cout << "Creating Image" << endl;
for (unsigned int y = 0; y < height; ++y) {
    for (unsigned int x = 0; x < width; ++x) {
        double zr = (4.0 * x / width - 2.0);
        double zi = (4.0 * y / height - 2.0);
        z.set(zr, zi);

        unsigned int n = 0;
        while (n < maxiter) {
            z = (z ^ 2) + c;

            if (z.mod() > bailout) {
                int index = 0;

                if (n > maxiter * borderRatio)
                    index = 255;
                else
                    index = pow(1.0 * n / maxiter, colorExponent) * 255;

                pic.set_pixel(x, y, index, index, index);
                n = maxiter;
            }
            n++;
        }
    }
}

pic.save_image("image.bmp");

cout << "Done" << endl;
return 0;
}

```

Compile the program. Note that the `bitmap_image.hpp` file is a header file with all the needed code for the image processing. Hence it simply needs to be included in the main, as above, and not compiled. So you will not add it to the makefile listing.

Run the program, and when asked for input, type in the following values.

```

Input the image size (in pixels): 800
Input the real value of c: -.8
Input the imaginary value of c: .156
Input the maximum iteration (100-1,000): 1000
Input the border ratio (0-1): .5
Input the color exponent (0-1): .5
Input the bailout radius (>= 4): 4

```

The result will be a bitmap image file stored in your project's working directory. Open the image file in an image viewer and if it is not a really cool image then you may have something wrong with the Complex class.

Some other interesting values of c you may wish to try are,

- $c = -0.52 + 0.57i$
- $c = 0.295 + 0.55i$
- $c = -0.624 + 0.435i$
- $c = i$
- $c = -1.25$
- $c = 0.285 + 0.01i$
- $c = -0.70176 - 0.3842i$
- $c = -0.835 - 0.2321i$
- $c = -0.8 + 0.156i$
- $c = -0.8i$

4 Poker Game Project

The objective of this project is to create a program that will play a single round of the poker game five card draw for between 2 and 6 players. In the game of five card draw each player is dealt 5 cards then each player is allowed to throw away up to three cards and have them replaced with new cards from the top of the deck. We will not play the version where you can draw 4 cards if you have an Ace. So when the players get the initial 5 cards these are dealt in the normal manner, one card being dealt to player 1, then one to player 2, then one to player 3, and so on cycling back to player 1 for the second card of each hand and so on. When the player throws away any cards the cards are replaced all at once from the top of the deck. So if player 1 throws away 2 cards these are replaced by the top two cards from the deck.

The program should work as follows. The user is first asked how many players will be playing the game. This input should be between 2 and 6 and the program should catch any errors in the input and if there are, ask the user for another input. The program will then deal the cards to each player (this will not be shown on the screen). Then one by one each player will be asked how many cards they would like to draw. This input must be between 0 and 3 cards, again catching any errors in the input. If the number they want to draw is greater than 0 the program should ask for the card positions to be thrown away, one by one. These inputs should be between 1 and 5, 1 representing the first card, 2 the second and so on. Again catch all input errors. Here there could be another error that is not simply inputting an integer between 1 and 5. If the user inputs a card number that they have already input the program should not replace that card again. This would be like being dealt a card and then throwing that same card away for another card, this is not legal in poker. After each player has had the chance to draw the program should print out the hands of all the players and declare a winner. At this point the program ends.

4.1 Example Program Runs

```
Input number of players 2-6: 3
```

```
Player 1: AD 2S 4S 4D 6C --- One Pair
Number of cards to draw: Input the number of cards to draw 0-3: 2
Input the number of the card to replace 1-5: 2
Input the number of the card to replace 1-5: 5
```

```
Player 2: 5H 10S QD QS KH --- One Pair
Number of cards to draw: Input the number of cards to draw 0-3: 2
Input the number of the card to replace 1-5: 1
Input the number of the card to replace 1-5: 2
```

```
Player 3: 3S 3C 6S 7D 8C --- One Pair
Number of cards to draw: Input the number of cards to draw 0-3: 3
Input the number of the card to replace 1-5: 3
```

Input the number of the card to replace 1-5: 4
Input the number of the card to replace 1-5: 5

Player 1: AD 2H 4S 4D KS --- One Pair
Player 2: 3H 4C QD QS KH --- One Pair
Player 3: 3S 3C 8D 9H KC --- One Pair

Player 2 won this hand.

Input number of players 2-6: 1
Input was not between 2 and 6, try again
Input number of players 2-6: 8
Input was not between 2 and 6, try again
Input number of players 2-6: 3

Player 1: 6H 7S 8S JH QH --- High Card
Number of cards to draw: Input the number of cards to draw 0-3: 3
Input the number of the card to replace 1-5: 1
Input the number of the card to replace 1-5: 1
This card was already replaced, please select another card.
Input the number of the card to replace 1-5: 6
Input was not between 1 and 5, try again
Input the number of the card to replace 1-5: 2
Input the number of the card to replace 1-5: 2
This card was already replaced, please select another card.
Input the number of the card to replace 1-5: 1
This card was already replaced, please select another card.
Input the number of the card to replace 1-5: 3

Player 2: 6C 8H 9D 10C JS --- High Card
Number of cards to draw: Input the number of cards to draw 0-3: 1
Input the number of the card to replace 1-5: 1

Player 3: AD 2H 3D 9S KH --- High Card
Number of cards to draw: Input the number of cards to draw 0-3: 3
Input the number of the card to replace 1-5: 2
Input the number of the card to replace 1-5: 3
Input the number of the card to replace 1-5: 4

Player 1: 4S 4D 7C JH QH --- One Pair
Player 2: AS 8H 9D 10C JS --- High Card
Player 3: AD 5D 8C JD KH --- High Card

Player 1 won this hand.

Input number of players 2-6: 4

Player 1: 5S 5D 6C JH JD --- Two Pair
Number of cards to draw: Input the number of cards to draw 0-3: 1

Input the number of the card to replace 1-5: 3

Player 2: AH 3D 5H 8C 9S --- High Card

Number of cards to draw: Input the number of cards to draw 0-3: 3

Input the number of the card to replace 1-5: 2

Input the number of the card to replace 1-5: 3

Input the number of the card to replace 1-5: 4

Player 3: AS 3S 4S QH KD --- High Card

Number of cards to draw: Input the number of cards to draw 0-3: 2

Input the number of the card to replace 1-5: 2

Input the number of the card to replace 1-5: 3

Player 4: 2H 9D 9C JC KH --- One Pair

Number of cards to draw: Input the number of cards to draw 0-3: 2

Input the number of the card to replace 1-5: 1

Input the number of the card to replace 1-5: 4

Player 1: AC 5S 5D JH JD --- Two Pair

Player 2: AH 3H 9S 10D 10C --- One Pair

Player 3: AS 2D QS QH KD --- One Pair

Player 4: 4C 6S 9D 9C KH --- One Pair

Player 1 won this hand.

Input number of players 2-6: 3

Player 1: 2D 3S 3D 7D 8C --- One Pair

Number of cards to draw: Input the number of cards to draw 0-3: 3

Input the number of the card to replace 1-5: 1

Input the number of the card to replace 1-5: 4

Input the number of the card to replace 1-5: 5

Player 2: 4D 5H 6D 7H 9D --- High Card

Number of cards to draw: Input the number of cards to draw 0-3: 1

Input the number of the card to replace 1-5: 5

Player 3: AD 6C 6H 8S 10C --- One Pair

Number of cards to draw: Input the number of cards to draw 0-3: 2

Input the number of the card to replace 1-5: 4

Input the number of the card to replace 1-5: 5

Player 1: AS 3C 3S 3D QC --- Three of a Kind

Player 2: 4D 5H 6D 7H 8D --- Straight

Player 3: AD 2H 6C 6H JH --- One Pair

Player 2 won this hand.

In the example below, I also printed the deck of cards after the deck was shuffled so that you can better see how the cards are dealt to each player at the beginning and also during the drawing phase of the game.

```
6S 9S AH 8D 10C 6C 5S KS 2H QS AD 5C 9H 2C 8S 6D AC 10S 4H 9C 7H 3D JH
2S 3S 6H 2D 7D 9D 5H 4C 3C 10H 8H KH 8C KD JS 7C 4D AS QH 7S QC JD KC
4S QD 10D JC 5D 3H
```

Input number of players 2-6: 3

Player 1: 5S 6S 8D 9H QS --- High Card

Number of cards to draw: Input the number of cards to draw 0-3: 1

Input the number of the card to replace 1-5: 5

Player 2: AD 2C 9S 10C KS --- High Card

Number of cards to draw: Input the number of cards to draw 0-3: 3

Input the number of the card to replace 1-5: 2

Input the number of the card to replace 1-5: 3

Input the number of the card to replace 1-5: 4

Player 3: AH 2H 5C 6C 8S --- High Card

Number of cards to draw: Input the number of cards to draw 0-3: 3

Input the number of the card to replace 1-5: 2

Input the number of the card to replace 1-5: 3

Input the number of the card to replace 1-5: 4

Player 1: 5S 6S 6D 8D 9H --- One Pair

Player 2: AD AC 4H 10S KS --- One Pair

Player 3: AH 3D 7H 8S 9C --- High Card

Player 2 won this hand.

4.2 Project Specifications

In this project you must incorporate the following, at the very least.

1. You must create three class structures for this project, each with their own declaration file (.h) and implementation file (.cpp), the classes and their functions are below. All data members must be private. The Card and Deck classes can be updates of those you created in a previous assignment.
 - (a) Card Class: The card class is an object representing a single card. The data needs to hold a face value and suit. This class will also need accessors and mutators. It should handle the conversion of the data to a string (such as 4H, 2D, AS) so that other functions or classes can print the card value out without the need to redo the conversions. You will need to sort cards so you will want to overload the comparison operators. Specifically, the Card class should store the card face and suit as integers. The value should be 1 for Ace, 2-10, 11 for Jack, 12 for Queen, and 13 for King. It should also have at least the following functionality.
 - `Card()`: Default constructor will set the value and suit both to 0, that is, a card that does not exist.
 - `Card(int v, int s)`: Constructor that sets value to v and suit to s. You may default these values to take care of the default constructor.
 - `string toStringFace()`: Returns a string of the face value of the card, use A, J, Q, and K for the face cards.
 - `string toStringSuit()`: Returns a string of the suit of the card, use D, C, H, and S.

- `string toString()`: Returns a sting of the card in condensed form, for example, 2C, AS, JD, 10H. There should be no space between the value and the suit.
 - `string toString(bool space)`: Returns a sting of the card in condensed form, for example, 2C, AS, JD, 10H. If space is true then there should be one space between the value and the suit, for example, 2 C, A S, J D, 10 H. Again, you may default the parameter to cover the previous function.
 - Overload the six comparison operators that compare the values of the cards. Note that the Ace can be played as either high or low, for comparison, we will consider it low. So the Ace would come before the 2.
 - `bool isEqual(Card card2)`: Returns true is the two cards are identical, both value and suit are the same. Returns false otherwise.
 - Overload the stream out operator to stream the card output in condensed form, for example, 2C, AS, JD, 10H.
- (b) Deck Class: This class is for the deck of cards object. It should hold a set of 52 cards, standard poker deck. An obvious choice is an array of card objects. Its constructor will need to create the cards in the deck. You will also need a way to determine where the top of the deck is so that you know the next card to be dealt. As in the previous assignment, you can include an integer member called `top`. This class should have methods for shuffling, dealing a card, and printing the deck (helps to determine if gameplay is correct). Specifically, at least the following need to be implemented.
- `Deck()`: Constructor that creates the deck of cards.
 - `void PrintDeck()`: Prints out the deck in a single like using no space between the value and suit for each card and one space between the cards.
 - `void ShuffleDeck()`: Shuffles the deck of cards.
 - `Card dealCard()`: Deals the next card off the top of the deck.
 - `Card getCard(int i)`: Gets the card at index `i` in the deck.
 - `void reset()`: Resets the top to 0.
 - Overload the stream out operator to output the entire deck of cards in its current order. Each card should be output in condensed form with a single space between consecutive cards. The output should be like the example above.
- (c) Poker Hand Class: This class will represent one player's hand. Hence it should hold a set of 5 card objects, use an array. You should also want to store the number of cards that are currently in their hand, this will help organize the hand during the dealing of cards from the deck into the hand so you know what slot to put the new card in. You will need functions for adding a card to the hand, printing the hand, sorting the hand, clearing the hand, replacing a card (accessors and mutators), determining the type of hand (algorithm below), determining the value of a hand (algorithm below), and a function to determine if one hand is greater then another (helps for finding the winner). Although there are numerous ways to determine a hand type and winners of a poker hand you are required to implement the algorithms discussed below. Specifically, you will need to implement at least the following.
- `PokerHand()`: Constructor.
 - `clearHand()`: Removes the cards from the hand in order to deal a new hand to the player.
 - `addToHand(Card card)`: Places the parameter card into the hand in the next position in the hand array, and then increment the number of cards in the hand.
 - `void PrintHand()`: Prints out the hand on a single like using no space between the value and suit for each card and one space between the cards.
 - `void PrintHand(int width)`: Prints out the hand on a single like using no space between the value and suit for each card and uses width spaces to print the

card, right justifying the card. The output below uses this function with a width of 4.

5S 6S 6D 8D 10H

- `void sortHand()`: Sorts the cards in the hand in ascending order by face value. Note that this can be done using the overloaded comparison operators of the card class and does not need any knowledge on the method of storage or the value designation used for the card class. Also, the sort that is implemented is to be either, bubble, selection, or insertion. Do not use the algorithms library.
- `void replace(int pos, Card card)`: Replaces the card at position pos with the new card.
- `int* convertHandToArray()`: In the algorithm below on determining the type of hand the person has the hand is converted to an array of integers to make it easier to computationally determine the hand type. This function takes the hand and constructs the corresponding array, dynamically.
- `bool isFlush()`: Returns true if the hand is a flush, false otherwise.
- `bool isStraight()`: Returns true if the hand is a straight, false otherwise. Remember that the Ace can be either high or low.
- `bool isFullHouse()`: Returns true if the hand is a full house, false otherwise.
- `bool isFourOfKind()`: Returns true if the hand is a four of a kind, false otherwise.
- `bool isThreeOfKind()`: Returns true if the hand is a three of a kind, false otherwise.
- `bool isTwoPair()`: Returns true if the hand is two pair, false otherwise.
- `bool isPair()`: Returns true if the hand is one pair, false otherwise.
- `long long getValue()`: In the algorithm below on determining the winner there is a method for converting the hand to a single number. This function does that conversion and returns the hand value.
- Overload the six comparison operators between two poker hands that compare the hand values returned by the `getValue` function.
- `string HandAndTypeToString(int width)`: This will return a string with the hand followed by the hand type, as in the example below. The width is the number of spaces used for each card, 4 in the example below.
AD AC 4H 10S KS --- One Pair

- Overload the stream out operator that will output the hand with each card having 4 spaces. For example,

AD AC 4H 10S KS

2. The main will control the gameplay. It will create a single deck and an array of poker hands (players) that is of the needed size. Deal the cards to each player, then for each player one at a time ask for the number of cards to draw and then which cards to replace. Do the replacement for each player, print the final hands out, their types, and who won the hand. The gameplay output is to look like the examples above, you do not need to print the deck out at the start like I did in the last example, that was just for illustration. You will want to do this while you are coding so that you know the initial deal and draws are being handled correctly, but not in the finished product.
3. There must be error checking on all user inputs. You may assume that they always type in an integer but all ranges and duplications must be checked.
4. The hands must be sorted by card face value before printing them out.
5. The hand type must be displayed when the hand is printed. A description of 5-card poker hand types and which type wins over another type are listed at the end of this handout. I also give you the algorithm for doing this below.

6. The winner of the hand must be displayed. I give you the algorithm for doing this below.

4.3 Determining the Type of Hand

Determining the type of hand is easier than it may seem at first sight. One method is to create an array with 13 cells and then for each face value of the cards in a hand increment the respective cell. So if index 0 represents a 2, index 1 a 3 and so on, we have the following examples.

• 4S 4C 5C KD AC	<table border="1"><tr><td>0</td><td>0</td><td>2</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	2	1	0	0	0	0	0	0	0	1	1
0	0	2	1	0	0	0	0	0	0	0	1	1		
• 9D 9C 9H JD JC	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>3</td><td>0</td><td>2</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0	3	0	2	0	0	0
0	0	0	0	0	0	0	3	0	2	0	0	0		
• 2C 10S JS QH KC	<table border="1"><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr></table>	1	0	0	0	0	0	0	0	1	1	1	1	0
1	0	0	0	0	0	0	0	1	1	1	1	0		
• 9C 10S JS QH KC	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr></table>	0	0	0	0	0	0	0	0	1	1	1	1	0
0	0	0	0	0	0	0	0	1	1	1	1	0		
• 5D 5C 5H 7D QC	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>3</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr></table>	0	0	0	3	0	1	0	0	0	0	1	0	0
0	0	0	3	0	1	0	0	0	0	1	0	0		
• 5D 5C 5H 5S QC	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>4</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr></table>	0	0	0	4	0	0	0	0	0	0	1	0	0
0	0	0	4	0	0	0	0	0	0	1	0	0		

From these examples it should be clear how you can use this counting array to determine the type of hand you have. Note that in the first example there is a 2 in the array and ones elsewhere indicating a single pair. Example 2 has a 3 and 2 in the array hence a full house. Example 3 has all 1's and 0's and the 1's are not in a row, so this is high card and the high card is a King, the last position of a 1. The fourth example has 5 1's in a row, hence a straight. Note that with a straight the Ace can be both high and low. That is A 2 3 4 5 and 10 J Q K A are both straights. The fifth example is three of a kind and the final example is four of a kind. Note that the flush and straight flush would not be incorporated here but if the player has a flush then all of the card suits would be identical.

4.4 Determining the Winning Hand

You will need to be able to compare two hands to see which is better. Note that there are cases in Poker where there is a tie. For example, two straights with the same high card would be a tie. There are, of course, many other cases where ties occur.

This is probably the most involved portion of the program. Since you know the type of hand, you can determine that a full house wins over two pair fairly easily, but what if two players both have a straight? Then you need to look at the high card in each case. Furthermore, in this situation the Ace could be a high card or a low card. Even more interestingly, what if two players have the same two pair, then you need to look at the “kicker”. There are numerous ways to solve this problem but the one you will implement is as follows. This is a scheme that will associate each possible hand with a unique number in such a way that if one hand beats out another then the winning hand will have the larger value.

Note that the maximum value of a long (or long long on Windows) is 9,223,372,036,854,775,807. The most significant digit is a 9 and interestingly enough there are 9 types of poker hands. There are also six blocks of three digits that follow the 9 and your hand has only 5 cards in it. So it seems plausible that you can store all the information about the specifics of the hand in these positions.

Let's look at a few examples. Say we have this hand, which is of course one pair but we need to look at the other cards as well.

Player 1: 5S 6S 6D 8D 9H --- One Pair

Say for a pair we use the type number 1 (since we would probably use 0 for high card) then we would start with a value of

1,000,000,000,000,000,000

The most important part of the pair is the actual pair, here that is a 6 so we will put 6 in the next block of three numbers, now the value is

1,006,000,000,000,000,000

From here we just have single cards left so we will use the next blocks to store their values from highest to lowest, now the value of the hand is

1,006,009,008,005,000,000

Now if we had two players with a pair as below.

Player 1:	5S	6S	6D	8D	9H	---	One Pair
Player 2:	AD	AC	4H	10S	KS	---	One Pair

The value of player 1 is as we calculated 1,006,009,008,005,000,000 and using the same scheme the value of player 2 is 1,014,013,010,004,000,000, using 14 for the Aces since we would use their high value for pairs, as well as three of a kind, etc.. So player 2 has a larger value and hence would win the hand, as they should since the pair of Aces would win over a pair of sixes.

If the pairs were the same in the case below,

Player 1:	5S	6S	6D	8D	9H	---	One Pair
Player 2:	AD	6C	6H	10S	KS	---	One Pair

With these two hand the value of player 1 is 1,006,009,008,005,000,000 and the value of player 2 is 1,006,014,013,010,000,000. So player 2 still has a larger value and hence would win the hand, as they should since the next highest card, the Ace, would win over the nine.

If you had two different types then the most significant digits will be different. For example, say that 2 pair has a type of 2 then the values of the following hands would be 2,011,005,014,000,000,000 and 1,010,014,009,003,000,000 respectively, hence player 1 would win.

Player 1:	AC	5S	5D	JH	JD	---	Two Pair
Player 2:	AH	3H	9S	10D	10C	---	One Pair

At this point you should see how you would proceed to create a scheme for the other hand types. Specifically,

- High Card: Start with 0,000,000,000,000,000,000 (that is 0, but we put in the 0 blocks for illustration). Put the face values of each of the cards in the hand in the 0 blocks in descending order. For example, the following hands have the following values.

2S	5D	6H	7S	JC	11,007,006,005,002,000
AD	3S	7H	10H	KD	14,013,010,007,003,000
2S	3S	9H	QD	KH	13,012,009,003,002,000

- One Pair: Start with 1,000,000,000,000,000,000, put the pair face value in the first 0 block and the other three face values in each subsequent block, in descending order. For example, the following hands have the following values.

5S	6S	6D	8D	9H	1,006,009,008,005,000,000
AD	6C	6H	10S	KS	1,006,014,013,010,000,000
AH	3H	9S	10D	10C	1,010,014,009,003,000,000

- Two Pair: Start with 2,000,000,000,000,000,000, put the highest pair face value in the first 0 block and the lower pair face value in the second block. Finally put the last card face value in the third block. For example, the following hands have the following values.

AC	5S	5D	JH	JD	2,011,005,014,000,000,000
AD	AC	10H	10S	KS	2,014,010,013,000,000,000
3D	3H	9S	10D	10C	2,010,003,009,000,000,000

- Three of a Kind: Start with 3,000,000,000,000,000,000, put the face value of the three of a kind in the first 0 block and the other two cards in the second and third 0 blocks in descending order. For example, the following hands have the following values.

5C	5S	5D	7H	JD	3,005,011,007,000,000,000
AD	10C	10H	10S	KS	3,010,014,013,000,000,000
3D	3H	3S	10D	JC	3,003,011,010,000,000,000

Since you cannot have two hands with the same three of a kind you can simply use just the first block that stores the three of a kind face value. That is,

5C	5S	5D	7H	JD	3,005,000,000,000,000,000
AD	10C	10H	10S	KS	3,010,000,000,000,000,000
3D	3H	3S	10D	JC	3,003,000,000,000,000,000

- Straight: Start with 4,000,000,000,000,000,000, put the face values of each of the cards in the hand in the 0 blocks in descending order. For example, the following hands have the following values.

2S	3H	4H	5S	6D	4,006,005,004,003,002,000
AD	10H	JC	QD	KS	4,014,013,012,011,010,000
7C	8C	9H	10D	JS	4,011,010,009,008,007,000

In this case you really only need the highest card value in the straight since the others would be determined by it. So alternatively you could put the highest card face value in the first 0 block and not worry about the others. With this scheme you would have,

2S	3H	4H	5S	6D	4,006,000,000,000,000,000
AD	10H	JC	QD	KS	4,014,000,000,000,000,000
7C	8C	9H	10D	JS	4,011,000,000,000,000,000

- Flush: Start with 5,000,000,000,000,000,000, put the face values of each of the cards in the hand in the 0 blocks in descending order. For example, the following hands have the following values.

2S	5S	6S	7S	JS	5,011,007,006,005,002,000
AD	3D	7D	10D	KD	5,014,013,010,007,003,000
2H	3H	9H	QH	KH	5,013,012,009,003,002,000

- Full House: Start with 6,000,000,000,000,000,000, put the face value of the three of a kind in the first 0 block and the pair face value in the second block. For example, the following hands have the following values.

2S	2D	7S	7C	7D	6,007,002,000,000,000,000
AD	AS	AH	10D	10C	6,014,010,000,000,000,000
2H	2C	2D	QH	QD	6,002,012,000,000,000,000

As in the case of three of a kind, you cannot have two hands with the same triple value. So you could simply store the triple face value in the first block.

2S	2D	7S	7C	7D	6,007,000,000,000,000,000
AD	AS	AH	10D	10C	6,014,000,000,000,000,000
2H	2C	2D	QH	QD	6,002,000,000,000,000,000

- Four of a Kind: Start with 7,000,000,000,000,000,000, put the face value of the four of a kind in the first 0 block and the fifth card face value in the second block. For example, the following hands have the following values.

2S	7H	7S	7C	7D	7,007,002,000,000,000,000
AD	AS	AH	AC	10C	7,014,010,000,000,000,000
9H	QC	QD	QH	QS	7,012,009,000,000,000,000

Since you cannot have two hands with the same four of a kind you can simply use just the first block that stores the four of a kind face value. That is,

2S	7H	7S	7C	7D	7,007,000,000,000,000,000
AD	AS	AH	AC	10C	7,014,000,000,000,000,000
9H	QC	QD	QH	QS	7,012,000,000,000,000,000

- Straight Flush: Start with 8,000,000,000,000,000,000, then as with the straight, you could put the face values in descending order on the 0 blocks or just the high card in the first 0 block. For example, the following hands have the following values.

2S	3S	4S	5S	6S	8,006,005,004,003,002,000
AD	10D	JD	QD	KD	8,014,013,012,011,010,000
7C	8C	9C	10C	JC	8,011,010,009,008,007,000

Or using the alternative method,

2S	3S	4S	5S	6S	8,006,000,000,000,000,000
AD	10D	JD	QD	KD	8,014,000,000,000,000,000
7C	8C	9C	10C	JC	8,011,000,000,000,000,000

Note that in either case this also takes care of the highest hand possible, the Royal Flush.

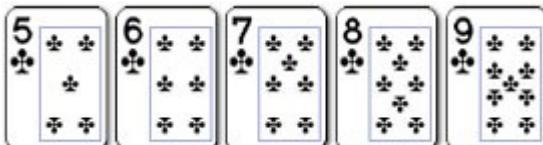
Note that we never use that last set of three zeros, so we could remove them and use values that have a most significant digit of 0–8 and then 5 blocks of zeros to store the information for the hand. For example,

2S	3S	4S	5S	6S	8,006,005,004,003,002
AD	10D	JD	QD	KD	8,014,013,012,011,010
7C	8C	9C	10C	JC	8,011,010,009,008,007

Just make sure that you are consistent with all hand types.

Winning Poker Hands

Straight Flush



Five cards in sequence, of the same suit. In the event of a tie: Highest rank at the top of the sequence wins. The best possible straight flush is known as a **royal flush**, which consists of the ace, king, queen, jack and ten of a suit. A royal flush is an unbeatable hand.

Four of a Kind



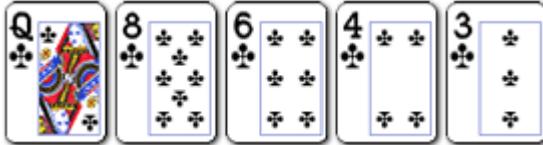
Four cards of the same rank, and one side card or 'kicker'. In the event of a tie: Highest four of a kind wins.

Full House



Three cards of the same rank, and two cards of a different, matching rank. In the event of a tie: Highest three matching cards wins.

Flush



Five cards of the same suit. In the event of a tie: The player holding the highest ranked card wins. If necessary, the second-highest, third-highest, fourth-highest, and fifth-highest cards can be used to break the tie. If all five cards are the same ranks, it is a tie. The suit itself is never used to break a tie in poker.

Straight



Any five consecutive cards of different suits. In the event of a tie: Highest ranking card at the top of the sequence wins. Note: The Ace may be used at the top or bottom of the sequence, and is the only card which can act in this manner. A,K,Q,J,T is the highest (Ace high) straight; 5,4,3,2,A is the lowest (Five high) straight.

Three of a Kind



Any three cards of the same rank. In the event of a tie: Highest ranking three of a kind wins.

Two Pair



Any two cards of the same rank together with another two cards of the same rank. In the event of a tie: Highest pair wins. If players have the same highest pair, highest second pair wins. If both players have two identical pairs, highest side card wins. If side cards are identical it is a tie.

One Pair



Two cards of a matching rank, and three unrelated side cards. In the event of a tie: Highest pair wins. If players have the same pair, the highest side card wins, and if necessary, the second-highest and third-highest side card can be used to break the tie.

High Card



Any hand that does not qualify under a category listed above. In the event of a tie: Highest card wins, and if necessary, the second-highest, third-highest, fourth-highest and smallest card can be used to break the tie.