CIS 278 (CS1) Programming Methods: C++

Assignment 4: Arrays

Skip to Main Content

Learning Objectives

After the successful completion of this learning unit, you will be able to:

- Implement syntactically correct C++ arrays.
- Solve a variety of standard problems using arrays.

Array Practice

I recommend that before you begin the assignment you write as many of these small ungraded programming challenges as you can. You should at least write 2 or 3 of them. They are a good way to gradually build up your confidence and skills. Of course, you'll have to write a program to test each function as well. Note that **none of these functions should include any input or output!**

- Write a function named noNegatives(). It should accept an array of integers and a size argument. It should return true if none of the values are negative. If any of the values are negative it should return false

```
bool noNegatives(const int array[], int size);
```

- Write a function named absoluteValues(). It should accept an array of integers and a size argument. It should replace any negative values with the corresponding positive value.

```
void absoluteValues(int array[], int size);
```

- Write a function named eCount. It should accept an array of chars and a size argument. It should return the number of times that the character 'e' shows up in the array.

```
int eCount(const char array[], int size);
```

- Write a function named charCount. It should be similar to eCount, but instead of counting 'e's it should accept a third argument, a target char. The function should return the number of times the target char shows up in the array.

```
int charCount(const char array[], int size, char targetChar);
```

- Write a method named isSorted. It should accept an array of integers and a size argument. It should return true if the values are sorted in ascending order. False if they are not.

```
bool isSorted(const int array[], int size);
```

- Write a method named equalNeighbors. It should accept an array of chars and a size argument. It should return true if there are two adjacent elements in the array with equal values. If there are not, it should return false.

```
bool equalNeighbors(const char array[], int size);
```

This is not a homework assignment, so feel free to post your code to one of these (not more than one) to the forum at any time.

For Credit

Assignment 4.1 [45 points]

Write a program that reads five (or more) cards from the user, then analyzes the cards and prints out the category of hand that they represent.

Note: This assignment is a great array-practice assignment, if written in a certain way. There are clever algorithms that you can discover yourself or find online that allow you to skip over most of the array processing. You aren't allowed to do something like that, since that would defeat the objectives of the assignment. Don't worry, if you follow all of the requirements below you're in no danger of accidentally falling into the "clever algorithm" approach. This paragraph is just advance warning of the situation. Some of you may be frustrated that you aren't allowed to be creative with your solution. The purpose of this paragraph is to explain why.

Poker hands are categorized according to the following labels: Straight flush, four of a kind, full house, straight, flush, three of a kind, two pairs, pair, high card.

To simplify the program we will ignore card suits, and face cards. The values that the user inputs will be integer values from 2 to 9. When your program runs it should start by collecting five integer values from the user and placing the integers into an array that has 5 elements. It might look like this:

```
Enter five numeric cards, no face cards. Use 2 - 9.
Card 1: 8
Card 2: 7
Card 3: 8
Card 4: 2
Card 5: 3
```

(This is a pair, since there are two eights)

Since we are ignoring card suits there won't be any flushes. Your program should be able to recognize the following hand categories, listed from least valuable to most valuable:

Hand Type	**Description**	**Example**
High Card	There are no matching cards, and the hand is not a straight	2, 5, 3, 8, 7
Pair	Two of the cards are identical	2, 5, 3, 5, 7
Two Pair	Two different pairs	2, 5, 3, 5, 3
Three of a kind	Three matching cards	5, 5, 3, 5, 7
Straight	5 consecutive cards	3, 5, 6, 4, 7
Full House	A pair and three of a kind	5, 7, 5, 7, 7
Four of a kind	Four matching cards	2, 5, 5, 5, 5

(A note on straights: a hand is a straight regardless of the order. So the values 3, 4, 5, 6, 7 represent a straight, but so do the values 7, 4, 5, 6, 3).

Your program should read in five values and then print out the appropriate hand type. If a hand matches more than one description, the program should print out the most valuable hand type.

Here are three sample runs of the program:

```
Enter five numeric cards, no face cards. Use 2 - 9.
Card 1: 8
Card 2: 7
Card 3: 8
Card 4: 2
Card 5: 7
Two Pair!
```

```
Enter five numeric cards, no face cards. Use 2 - 9.
Card 1: 8
```

```
Card 2: 7
Card 3: 4
Card 4: 6
Card 5: 5
Straight!
```

```
Enter five numeric cards, no face cards. Use 2 - 9.
Card 1: 9
Card 2: 2
Card 3: 3
Card 4: 4
Card 5: 5
High Card!
```

Additional Requirements

1) You must write a function for each hand type. Each function must accept a const int array that contains five integers, each representing one of the 5 cards in the hand, and must return "true" if the hand contains the cards indicated by the name of the function, "false" if it does not. The functions should have the following signatures.

Note: in the below, a pair is defined as ***exactly*** two of the same card. If there are three of the same card, that is not a pair. Similarly, a three-of-a-kind is defined as ***exactly*** three of the same card. If there are four of the same card, that is not a three-of-a-kind. Similarly for four-of-a-kind.

```
// post:  returns true if and only if there are one or more pairs in the hand.  Note that
// this function returns false if there are more than two of the same card (and no other pairs).
bool  containsPair(const int hand[]);

// post: returns true if and only if there are two or more pairs in the hand.
bool  containsTwoPair(const int hand[])

// post: returns true if and only if there are one or more three-of-a-kinds in the hand.
bool  containsThreeOfaKind(const int hand[])

// post: returns true if there are 5 consecutive cards in the hand.
bool  containsStraight(const int hand[])

// post: returns true if there is a pair and a three-of-a-kind in the hand.
bool  containsFullHouse(const int hand[])

// post: returns true if there is a four-of-a-kind in the hand.
bool  containsFourOfaKind(const int hand[])
```

2) Of course, as a matter of good practice, you should use a constant to represent the number of cards in the hand, and everything in your code should still work if the number of cards in the hand is changed to 4 or 6 or 11 (for example).

3) Note that it is not acceptable to rely on the order in which you call the functions from main() to make up for any incorrect results in the functions themselves. That means that you should test the functions independently to make sure that they return the correct value.

Some examples:

- A hand that contains three-of-a-kind should return "false" for "containsPair()"
- A hand that contains four-of-a-kind should return "false" for "containsPair()" and "containsThreeOfaKind()"
- A hand that contains a full-house should return "true" for containsThreeOfaKind() and containsPair().
- A hand that contains two-pair should return "true" for containsPair().

Here is a table with some examples that I hope will help clear up any confusion. If there are additional hands that you are unsure about, please ask in the discussion, and I will consider adding rows to this table for further clarification.

hand	pair?	two-pair?	three-of-a-kind?	full-house?	four-of-a-kind?
2, 2, 2, 3, 4	F	F	T	F	F
2, 3, 3, 3, 3	F	F	F	F	T
2, 2, 3, 3, 3	T	F	T	T	F
2, 2, 3, 3, 4	T	T	F	F	F

4) You do not need to write a containsHighCard function. All hands contain a highest card. If you determine that a particular hand is not one of the better hand types, then you know that it is a High Card hand.

5) Do not sort the cards in the hand. Also, do not make a copy of the hand and then sort that.

6) It's possible to simply traverse the array and count up all of the matches you find, and then use that result to determine the type of hand. Don't use this technique. If this is confusing, don't worry. You would know if this is what you are doing.

7) An important objective of this assignment is to have you practice creating excellent decomposition. **Don't worry about efficiency on this assignment. Focus on excellent decomposition, which results in readable code.** This is one of those programs where you can rush and get it done but end up with code that is really difficult to read, debug, modify, and re-use. If you think about it hard, you can think of really helpful ways in which to combine the tasks that the various functions are performing. **5 extra credit points on this assignment will be awarded based on the following criteria:** no function may have nested loops in it. If you need nested loops, the inner loop must be turned into a separate function, hopefully in a way that makes sense and so that the separate function is general enough to be re-used by the other functions. Also, no function other than main() may have more than 5 lines of code. (This is counting declarations, but not counting the function header, blank lines, or lines that have only a curly brace on them.) In my solution I was able to create just 3 helper functions, 2 of which are used repeatedly by the various functions.

These additional criteria are intended as an extra challenge and may be difficult for many of you. If you can't figure it out, give it your best shot, but don't be too discouraged. It's just 5 points. And be sure to study the posted solution carefully.

Suggestions

Test these function independently. Once you are sure that they all work, the program logic for the complete program will be fairly straightforward.

Here is code that tests a containsPair function:

```
int main() {
        int hand[] = {2, 5, 3, 2, 9};

        if (containsPair(hand)) {
                cout << "contains a pair" << endl;
        }
}
```

Submit Your Work

Name your source code file according to the assignment number (a1_1.cpp, a4_2.cpp, etc.). Execute the program and copy/paste the output that is produced by your program into the bottom of the source code file, making it into a comment. Use the Assignment Submission link to submit the source file. When you submit your assignment there will be a text field in which you can add a note to me (called a "comment", but don't confuse it with a C++ comment). In this "comments" section of the submission page let me know whether the program works as required.

Keep in mind that if your code does not compile you will receive a 0.