Assignment 6: Reversal with Arrays

Submit all the deliverables to CourSys: https://courses.cs.sfu.ca/

10% penalty per day late; max 2 days late allowable.

This assignment may be done individually or in pairs (2). Do not show non-group members your code, do not copy code found online or from previous course offerings, and do not post questions about the assignment online. Direct all questions to the instructor and TA.

See the marking guide for details on how each part will be marked.

1. Reversal Game

The reversal game is played with permutations of the numbers from 1 to n. A permutation is a set of numbers arranged in a particular order. Each number can appear only once in a set, for example the possible permutations of the numbers from 1 to 3 are: {1,2,3}, {1,3,2}, {2,1,3}, {2,3,1}, {3,1,2} and {3,2,1}. Notice that there are n factorial (n! = 3! = 6) such permutations.

In the reversal game you start with any permutation of 1 to n you like. For example, when n = 5 one possible permutation is: 3 4 5 1 2

Playing the game consists of repeatedly reversing the order of the first m digits, where m is always the first number of the permutation.

In the permutation given above the first number is a 3, so you reverse the first 3 numbers of the permutation:

\[
\begin{align*}
3 & \quad 4 & \quad 5 & \quad 1 & \quad 2 - \text{reverse the first three numbers} \\
5 & \quad 4 & \quad 3 & \quad 1 & \quad 2
\end{align*}
\]

Now the first number is a 5, and so you reverse the first 5 numbers:

\[
\begin{align*}
5 & \quad 4 & \quad 3 & \quad 1 & \quad 2 - \text{reverse the first five numbers} \\
2 & \quad 1 & \quad 3 & \quad 4 & \quad 5
\end{align*}
\]

2 is the first number, and so you reverse the first 2 numbers:

\[
\begin{align*}
2 & \quad 1 & \quad 3 & \quad 4 & \quad 5 - \text{reverse the first two numbers} \\
1 & \quad 2 & \quad 3 & \quad 4 & \quad 5
\end{align*}
\]

The game ends when the 1 is at the start of the permutation (1 is guaranteed to always end up at the start). In this case, we performed three reversals to get 1 to the start, and so we say that the "score" of the permutation 3 4 5 1 2 is three.

In general, the reversal game is played by repeatedly reversing the first m elements of the permutation, where m is the first element of the permutation each time. The game ends when m is 1.

The objective of your program is to find a permutation of 1, 2, 3, ..., n which maximizes the score (i.e., it needs the greatest number of reversals to get 1 to the start of the permutation). This is a hard problem to solve since there are n! = 1 * 2 * ... * n different permutations of 1 to n, and there is no obvious pattern to how high-scoring permutations are distributed.

1 Assignment created by John Edgar; modified by Brian Fraser.
1.1 Check Permutation
Create a C++ file named `reversing.cpp`. Start with the sample code in the template on the course website. Write a function which accepts an int-array parameter and checks to see if it is a permutation, i.e. does the array contains the values 1 to \( n \), where \( n \) is also the size of the array. Your solution must use arrays, not vectors.

```cpp
// Returns true if arr is a permutation of 1, ..., n.
bool isPermutation(int arr[], int n);
```

You should perform enough tests so that you are confident `isPermutation()` works correctly in all cases. In the sample template code online there are a few sample tests to get you started.

*Hint: For \( arr[] \) to be a permutation of 1 to \( n \), it must contain each value between 1 and \( n \).*

1.2 Initialize the Permutation
Write a function that initializes an array to be the first permutation of size \( n \) (i.e., the numbers, in order, 1, 2, 3, ..., \( n \)). For example, given an array of size 5, it populates it with \{1, 2, 3, 4, 5\}.

```cpp
// Initialize arr[] to hold the values 1 to n, in order.
void initializePermutation(int arr[], int n)
```

1.3 Permutation Score
Write a function that computes and returns the score for a permutation, i.e. the number of reversals required to make \( arr[0] == 1 \).

```cpp
// Returns the number of reversals needed to make arr[0] == 1
// when the reversal game is played on arr.
// The array (arr) is unchanged by this function.
// Note: If arr[0] == 1 initially, then score(arr, n) returns 0
// Note: arr[] must be a permutation.
int scorePermutation(int arr[], int n)
```

You should perform enough tests so that you are confident it works correctly in all cases. The following sub-sections provide more details on how to implement this function.

1.3.1 Reversals
Compute the score by playing the reversal game, which needs a reverse function to reverses the first \( m \) values in the permutation array, where \( m \) is the first element of the permutation.

```cpp
// Reverse m elements in the array arr, changing the original array.
void reverseArray(int arr[], int m)
```

The pseudo-code for a simple reversal algorithm is as follows:

```cpp
void reverseArray(arr, m)
    for i = 0 through m/2
        p = m - 1 - i
        tmp = arr[i]
        arr[i] = arr[p]
        arr[p] = tmp
```
1.3.2 Array Copies
One thing to watch out for is changing your permutation when you don't want to. For example, the permutation you pass to scorePermutation() must not be changed. You'll need to make a copy of the permutation at the start of scorePermutation(), and then play the reversal game on the copy instead of the original. This will change the copy but leave the original unchanged.

You can create a copy by creating a new array as a local variable inside the function. Make this new array's size equal to the number of elements you were given (n), and copy the contents of arr into the new array by writing and calling the following function:

```c
// Copy all elements from source[] to destination[]
void copyArray(int destination[], int source[], int size)
```

Your scorePermutation() function might look something like this pseudo-code:

```c
int scorePermutation(arr, size)
    score = 0
    create new array: copy[size]
    copyArray(copy, arr, size)
    until done
        reverse(copy)
        adjust score
    return score
```

1.4 Permutation Print
Write a function that prints a permutation and its score.

```c
// Prints the permutation, its size, and score in the form:
//    "[n = <size>] Score of {1, 2, 3, ..., n} = <score>"
//    For example, if arr = {3, 4, 5, 1, 2}
//    then printPermutationAndScore(arr, 5) outputs:
//    [n = 5] Score of {3, 4, 5, 1, 2} = 3
void printPermutationAndScore(int arr[], int n)
```

Note that there is no trailing ',' in the list. i.e., it should be of the form “{3, 1, 2}”, not “{3, 1, 2,}”.
1.5 Generating Permutations

It is a challenging problem to come up with different permutations. Your program must check all possible permutations of a given size and find one that generates the highest score.

Since you need to test all possible permutations for a given size, you need a way to generate the permutations. A good way to do this is to generate them in sequence. The sample code online has the `nextPermutation()` function which generates permutations in lexicographic order (i.e., the “smallest” one first).

Given a permutation, it is possible to generate the permutation which lexicographically follows the current permutation. For example, given \{1, 2, 3, 4\}, the next permutation is \{1, 2, 4, 3\}. An algorithm for this is described here (stackoverflow.com), and implemented in the function:

```c
bool nextPermutation(int arr[], int n);
```
in the sample code online.

Call `nextPermutation()` from the functions you implement in the next section. Note that `nextPermutation()` calls the `reverseArray()` function which you have already written: if it contain bugs then it is likely `nextPermutation()` will not work correctly.

To call `nextPermutation()`, the input array must already be a valid permutation. See `demoPermutations()` in the code online for an example of how to create and initialize a permutation, and then use `nextPermutation()`. The output for this demo function, for a permutation size of 4, is shown below.

|   | 1: 1, 2, 3, 4, |
|---|---|---|
| 2: | 1, 2, 4, 3, |
| 3: | 1, 3, 2, 4, |
| 4: | 1, 3, 4, 2, |
| 5: | 1, 4, 2, 3, |
| 6: | 1, 4, 3, 2, |
| 7: | 2, 1, 3, 4, |
| 8: | 2, 1, 4, 3, |
| 9: | 2, 3, 1, 4, |
| 10: | 2, 3, 4, 1, |
| 11: | 2, 4, 1, 3, |
| 12: | 2, 4, 3, 1, |
| 13: | 3, 1, 2, 4, |
| 14: | 3, 1, 4, 2, |
| 15: | 3, 2, 1, 4, |
| 16: | 3, 2, 4, 1, |
| 17: | 3, 4, 1, 2, |
| 18: | 3, 4, 2, 1, |
| 19: | 4, 1, 2, 3, |
| 20: | 4, 1, 3, 2, |
| 21: | 4, 2, 1, 3, |
| 22: | 4, 2, 3, 1, |
| 23: | 4, 3, 1, 2, |
| 24: | 4, 3, 2, 1, |
1.6 Finding Optimal Permutations with Brute Force

Write a function that uses "brute force"¹ to find and return the permutation with the highest score for small values of \( n \). The function should find the permutation with the highest score by testing every possible permutation. This function outputs nothing to the screen: it writes the best permutation into the \( \text{arr} \) parameter. If there is more than one best permutation, it chooses just one of them (any one of them).

```
// Find an optimal permutation (of size \( n \)) and copy it into \( \text{arr}[] \).
// If more than one permutation maximizes the score, then any
// one of the maximizing permutations may be selected.
// The contents of \( \text{arr}[] \) do not matter when called; it will
// contain the maximizing permutation when done (it is output only).
void findOptimalPermutation(int \( \text{arr}[] \), int \( n \))
```

Write a function to find the optimal permutation for different sizes. This function can do most of its work by simply calling \( \text{findOptimalPermutation()} \) for all the different sizes. It prints to the screen.

```
// Test all permutations of sizes between \( \text{low} \) and \( \text{high} \) (inclusive).
// For each size, print one permutation which maximizes it score.
void printOptimalPermutations(int \( \text{low} \), int \( \text{high} \))
```

Call this function with the start size (\( \text{low} \)) of 1 and end size (\( \text{high} \)) of 11. It should generate an output similar to what is shown below. Note that it may take a while to run for \( n=10 \) and \( n=11 \).

```
\[
\begin{array}{l}
\text{[n = 1]} \text{ Score of \{1\} = 0} \\
\text{[n = 2]} \text{ Score of \{2, 1\} = 1} \\
\text{[n = 3]} \text{ Score of \{3, 1, 2\} = 2} \\
\text{[n = 4]} \text{ Score of \{2, 4, 1, 3\} = 4} \\
\text{[n = 5]} \text{ Score of \{3, 1, 4, 5, 2\} = 7} \\
\text{[n = 6]} \text{ Score of \{3, 6, 5, 1, 4, 2\} = 10} \\
\text{[n = 7]} \text{ Score of \{4, 7, 6, 2, 1, 5, 3\} = 16} \\
\text{[n = 8]} \text{ Score of \{6, 1, 5, 7, 8, 3, 2, 4\} = 22} \\
\text{[n = 9]} \text{ Score of \{6, 1, 5, 9, 7, 2, 8, 3, 4\} = 30} \\
\text{[n = 10]} \text{ Score of \{5, 9, 1, 8, 6, 2, 10, 4, 7, 3\} = 38} \\
\text{[n = 11]} \text{ Score of \{4, 9, 11, 6, 10, 7, 8, 2, 1, 3, 5\} = 51}
\end{array}
\]
```

For fun, try it with \( n=12 \) and \( n=13 \) to see how long it takes to run. The program you submit should only calculate between 1 and 11.

It is possible that your particular permutations might not be the same as the ones in this example because there might be more than one permutation with a maximum score and \( \text{findOptimalPermutation()} \) is allowed to return any one of them. However, the scores for each value of \( n \) should match. You may create additional functions if you wish.

---

¹ Brute force means computing all possibilities without any clever algorithm to consider only the most promising cases. For example, using brute force to play chess would be to compute all possible moves without regard for how generally promising it may normally be. Most intelligent systems don’t use just brute force: they use powerful algorithms, known as heuristics, to limit the number of things that need to be checked.
2. Representation

Complete each of the conversions listed below. You must show your work, not just the answer. You can use Google to check some answers: search “105 in hex” returns “105 = 0x69”.

You may do your work in a text document named conversions.txt, or by hand on paper and then scan it. The exact format of how you show your work is unimportant; you just need to demonstrate you did the work vs just using a calculator/Google/program.

You may use a calculator for the individual calculation steps, such as 12 * 16 = ???.

Convert the following from...

1. Decimal to hexadecimal:
   a) 239

2. Decimal to unsigned 8-bit binary:
   a) 5
   b) 175

3. Hexadecimal to binary (need not show work):
   a) 0x8e
   b) 0x7B41 09FC

4. Binary to hexadecimal (all unsigned values)
   a) 1101
   b) 0110 1111
   c) 0001 1001 1100 0011 1011 0111 1010 0000

5. Decimal to 2's complement binary, 8-bit notation:
   a) 124
   b) -7
   c) -110

6. 2's complement binary notation to decimal:
   a) 1101 0010
   b) 0110 1101
   c) 0000 0100
   d) 1011 0101
   e) 1111 1111 1111 1111 1111 1111 1111 1111
      (hint: Think first! No need to show work for part e.)

3. Deliverables

Submit a zip file of the items listed below to the CourSys: https://courses.cs.sfu.ca/

1. Your C++ source code: reversing.cpp
2. Number conversion calculations which show your work for the conversions. Must be in an electronic form: a PDF, a TXT or JPEG file, for example. It may have be created on the computer, or be a scan/photograph of a hand-written document.

To submit, create a group in CourSys (even if you are working alone). You can do this under this assignment's activity, look for the link in the corner. Only one member of your group submits the assignment to CourSys.

Please remember that all submissions will automatically be compared for unexplainable similarities. This comparison will also include similar assignments from previous semesters and the internet. Please review the notes from lecture on the expectations for academic honesty.