CS 31 Worksheet 7

This worksheet is entirely **optional**, and meant for extra practice. Some problems will be more challenging than others and are designed to have you apply your knowledge beyond the examples presented in lecture, discussion or projects. All exams will be done on paper, so it is in your best interest to practice these problems by hand and not rely on a compiler.

Concepts: Pointers

1) What does the following program output?

```
int main() {
      int a = 100, b = 30;
      cout << a + b << endl;           // (1)

      int* ptr = &a;
      cout << *ptr + b << endl;        // (2)

      *ptr = 10;
      cout << *ptr + b << endl;        // (3)

      ptr = &b;
      *ptr = -12;
      cout << *ptr + 2*b << endl;      // (4)

      int c = a + *ptr;
      cout << c << endl;               // (5)

      b = -5;
      cout << a + b << endl;           // (6)

      int arr[5] = {4, 5, 10, 11, -1};
      ptr = arr + 1;
      cout << *arr + *ptr << endl;     // (7)

      int cs;
      int& pic = cs;
      ptr = &pic;
      pic = 31;
      cs++;
      cout << *ptr << endl; // (8)
}
```

2) Write statements that declare a variable of each of the following types:
   - pointer to char
   - array of 10 pointers to char
   - pointer to const int

3) Consider the following function that loops through the characters of a C-string and prints them one by one. What are some possible inputs to the function that could break it?

```
void printChars(const char* str) {
  int i = 0;
  while (*(str + i) != '\0') {
    cout << *(str + i) << endl;
  }
}
```

4) Write a function that reverses a C string without using the `[]`. It takes in a pointer to the beginning of the array holding the C string.

   Function header: `void reverse(char* arr)`

```
char arr[6] = "hello";
reverse(arr);
// now reverse should be "olleh"

char arr[5] = "ucla";
reverse(arr);
// now reverse should be "alcu"

char arr[6] = "kayak";
reverse(arr);
// now reverse should be "kayak"
```

5) Write a function with the following header:

```
void descSort(int* nums, int len)
```

   Given an unsorted array of integers *nums* of length *len*, this function should sort the elements of nums in ascending order. Avoid using square brackets in the *descSort* definition.

   Example:

```
int a[10] = {3, 1, 4, 0, -1, 2, 3, 4, 1, 2};
descSort(a, 10);
//a = [4, 4, 3, 3, 2, 2, 1, 1, 0, -1]
```

6) Write a function minMax that takes in an int array `arr`, size of array `n`, and two int pointers, `min` and `max`. The function should set the min and max pointers to point to the min and max numbers in the array. Every number i in the array is constrained as follows: -1000 < i < 1000. Try to write this function without accessing any element of the array more than once.

Function header: `void minMax(int arr[], int n, int* min, int* max)`

```
int arr[5] = {0, 5, 7, -10, 2};
int* min;
int* max;
minMax(arr, 5, min, max);
// min should point to the -10
// max should point to the 7
```

7) Write a function with the following header:

```
void mergeAndSort(int* nums1, int* nums2, int* merged,
                  int len1, int len2)
```

Given two unsorted arrays of integers *nums1* and *nums2* of lengths *len1* and *len2* and an array *merged* of length *len1* + *len2*, this function should set the elements of merged to the numbers of *nums1* and *nums2* in ascending order. Avoid using square brackets in the *mergeAndSort* definition.

Example:

```
int a[3] = {3, 1, 4};
int b[4] = {2, 1, 1, 8};
int c[7];
mergeAndSort(a, b, c, 3, 4);
//c = [1, 1, 1, 2, 3, 4, 8]
```

8) Given pointers to two numbers represented by integer arrays, output the corresponding sum.

```
void sum(int* list1, int list1_size, int* list2, int list2_size)
```

Example:

```
list1 = 8 | 5 | 3 | 1      list1_size = 4
list2 = 5 | 3 | 2 | 9      list2_size = 4
```

This should output 13860, the sum of 8531 and 5329.

Example:

```
list1 =     5 | 3 | 1      list1_size = 3
```

```
list2 = 5 | 3 | 2 | 9      list2_size = 4
```

This should output 5860, the sum of 531 and 5329.

9) Write a function, rotate, that takes in an array A of 6 integers and an integer n, and rotates A by n positions to the right. If n is negative, rotate |n| positions to the left. Square brackets "[]" may **not** be used anywhere in your solution. Here's the prototype and some examples:

```
void rotate(int* A, int n);

A = {1,2,3,4,5,6};
B = {1,2,3,4,5,6};
C = {1,2,3,4,5,6};
```

After calling `rotate(A, 4), A = {3,4,5,6,1,2}`
After calling `rotate(B, -1), B = {2,3,4,5,6,1}`
After calling `rotate(C, 8), C = {5,6,1,2,3,4}`