CS 31 Worksheet Week 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.

If you have any questions or concerns please contact your LA or go to any of the LA office hours.

Solutions are written in red. The solutions for programming problems are not absolute, it is okay if your code looks different; this is just one way to solve the specific problem.

If you have any questions or concerns please contact your LA or go to any of the LA office hours.

Concepts: C-strings, Pointers

Reading Problems

1) What does the following program output?

```
#include <iostream>
using namespace std;
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)
    }
```

1. 130
2. 130
3. 40
4. -12+2*(-12) = -36
5. 10-12 = -2
6. 10-5 = 5
7. 4+5 = 9
8. 31+1 = 32

2) What does the following program output?

```
    void wiggum(char* suspect, int mysteryParam1, int
mysteryParam2)
    {
        int i = mysteryParam1;
        for (; i < mysteryParam1 + mysteryParam2; i++)
            cout << *(suspect + i);
        cout << endl;
    }

    void sideshowBob(char* target)
    {
        int surprise = -1;

        *target++;
        wiggum(target, -1, 3);
        surprise += target[0];
```

```
        *(target++);
        wiggum(target, -1, 2);
        surprise -= target[1];

        (*target)++;
        wiggum(target, -2, 4);
        surprise += target[0];

        char whyyoulittle = *target++ - surprise;
        cout << whyyoulittle << "art" << endl;
    }

    int main()
    {
        char homer[] = "D'oh";
        sideshowBob(homer);
    }
```

```
D'o
'o
D'ph
Bart
```

(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;
            i++;
        }
    }
```

```
char* str = null;
printChars(str);
```

Programming Problems

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

```
char* c;
char* c[10];
const int* i;
```

2) Write a function with the following header:

```
void reverse(char* arr);
```

`arr` is a pointer to the beginning of a character array holding a C string that is guaranteed to end with a null character '\0'

The function reverses the C string. **Do not use [ ]**.

Example:

```
char arr1[6] = "hello";
reverse(arr1);
// now arr1 should be "olleh"

char arr2[5] = "ucla";
reverse(arr2);
// now arr2 should be "alcu"

char arr3[6] = "kayak";
reverse(arr3);
// now arr3 should be "kayak"
```

```
void reverse(char *arr) {
        // Initialize second to point to the last char before '\0'
         char *first = arr;
         char *second = arr;
         while (*second != '\0')
             second++;
         second--;
         while (second > first) {
```

```
            int temp = *first;
            *first = *second;
            *second = temp;
            second--;
            first++;
        }
    }
```

3) Write a function with the following header:

```
void minMax(int* arr, int n, int*& min, int*& max);
```

`arr` is an integer array of size `n`
`min` and `max` are integer pointers

`minMax` should set the `min` and `max` pointers to point to the min and max numbers in the array. Try to write this function without accessing any element of the array more than once. If the array is empty or `n` is an invalid value, do not change the pointers. **Do not use [ ]**.

```
int arr[5] = {0, 5, 7, -10, 2};
int* pmin;
int* pmax;
minMax(arr, 5, pmin, pmax);
// pmin should point to the -10
// pmax should point to the 7
void minMax(int* arr, int n, int*& min, int*& max) {
    min = arr;
    max = arr;
    for (int i = 0; i < n; i++) {
        if (*arr > *max) {
            max = arr;
        }

        if (*arr < *min) {
            min = arr;
        }

        arr++;
    }
}
```

4) Write a function with the following header:

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

`nums` is an unsorted array of `len` integers

`descSort` sorts the elements of `nums` in descending order. **Do not use [ ]**.

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};
```

```
void descSort(int* nums, int len) {
    // now sorting the array using selection sort
    // curious about sorting algorithms? google it
    for (int x = 0; x < len; x++) {
        int max_num = *(nums + x);
        int max_ind = x;
        for (int y = x + 1; y < len; y++) {
            int curr_num = *(nums + y);
            if (curr_num > max_num) {
                max_num = curr_num;
                max_ind = y;
            }
        }
        *(nums + max_ind) = *(nums + x);
        *(nums + x) = max_num;
    }
}
```

5) Write a function with the following header:

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

`list1` and `list2` are two integer arrays representing numbers
`list1_size` is the number of digits in the first number
`list2_size` is the number of digits in the second number

`sum` prints the sum of the numbers.

Example:

```
int list1 = { 8, 5, 3, 1 };
int list2 = { 5, 3, 2, 9 };
sum(list1, 4, list2, 4);
// prints 13860, the sum of 8531 and 5329

int list3 = {    5, 3, 1 };
int list4 = { 5, 3, 2, 9 };
sum(list3, 3, list4, 4);
// prints 5860, the sum of 531 and 5329

void sum(int* list1, int list1_size, int* list2, int
list2_size) {
      int num1 = 0;
      int place = 1;
      for (int i = list1_size - 1; i >= 0; i--) {
            num1 += (list1[i] * place);
            place *= 10;
      }

      int num2 = 0;
      place = 1;
      for (int j = list2_size - 1; j >= 0; j--) {
            num2 += (list2[j] * place);
            place *= 10;
      }

      cout << num1 + num2 << endl;
}
```