CS 31 Solutions 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.

Solutions are written in red. The solutions for **programming** problems are not absolute. There are many equally correct solutions and different valid perspectives when it comes to solving Computer Science problems. If your solution is different, we highly recommend you share that with students as well!

Concepts: Pointers

Reading Problems

1) What does the following program output?

```
int main() {
    int a = 100, b = 30;
    cout << a + b << endl;            // (1) 130

    int* ptr = &a;
    cout << *ptr + b << endl;         // (2) 130

    *ptr = 10;
    cout << *ptr + b << endl;         // (3) 40  (a = 10, b =
30)

    ptr = &b;
    *ptr = -12;
    cout << *ptr + 2*b << endl;       // (4) -12 + 2(-12) = -36

    int c = a + *ptr;
    cout << c << endl;                // (5) -2

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

    int arr[5] = {4, 5, 10, 11, -1};
```

```
    ptr = arr + 1;
    cout << *arr + *ptr << endl;           // (7) 9

    int cs;
    int& pic = cs;
    ptr = &pic;
    pic = 31;
    cs++;
    cout << *ptr << endl;              // (8) 32
}
```

SOLUTION:

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) Consider the following function that loops through the characters of a supposed 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++;
    }
}
```

SOLUTION:

```
printChars(nullptr);

char ch = 'A';
printChars(&ch);
```

Programming Problems

3) 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

SOLUTION:

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

4) 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"
```

SOLUTION:

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

	// Move all the way to the end
	while (*second != '\0')
		second++;
	second--;

	// swap the characters at second and first
	while (second > first)
	{
		char temp = *first;
		*first = *second;
		*second = temp;
		second--;
		first++;
	}
}
```

5) 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 a[5] = {0, 5, 7, -10, 2};
int* pmin;
int* pmax;
minMax(a, 5, pmin, pmax);
// pmin should now point to the -10
```

```
// pmax should now point to the 7
SOLUTION:
void minMax(int* arr, int n, int*& min, int*& max)
{
      if (n <= 0)
            return;
      min = arr;
      max = arr;
      for (int i = 0; i < n; i++)
      {
            // Essentially keep comparing what is currently the
            // max or min to whatever int you are looking at
            if (*arr > *max)
                  max = arr;
            if (*arr < *min)
                  min = arr;
            arr++;
      }
}
```

6) Write a function with the following header:

```
bool valid_parenthesis(char* text)
```

`text` is a c-string.

`valid_parenthesis` determines if `text` forms a valid parenthesis string with correct openings and closings. **Do not use []**

You can be assured that text has length in [0,100). In addition, text contains only `'('` and `')'` chars.
Example:

```
char text[1] = {'('};
valid_parenthesis(text) returns false;

char text[2] = {'(', ')'};
valid_parenthesis(text) returns true;

char text[4] = {'(', ')', '(', ')'};
valid_parenthesis(text) returns true;

char text[4] = {'(', '(', ')', ')'};
```

```
char text[2] = {'(', '('};
valid_parenthesis(text) returns false;

char text[] = {};
valid_parenthesis(text) returns true; //you can have empty
string
```

SOLUTION:

```
bool valid_parenthesis(char* text){
      int counter = 0;

      while(*text != '\0')
      {
            if (counter < 0)
                  return false;
            if (*text == '(')
                  // everytime we encounter a left parentheses,
                  // we increment our counter
                  counter++;
            if (*text == ')')
                  // everytime we encounter a right parentheses,
                  // we decrement our counter
                  counter--;
            text++;
      }

      return counter == 0;
}
```

7) 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
```

SOLUTION:

```
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;
}
```