CS 31 Worksheet 2

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, it is okay if your code looks different; this is just one way to solve the specific problem.

Concepts

While Loops, Do While Loops, Functions -- by value and by reference, Switch Statements

Reading Problems

1) What does the following code snippet output?

```
void mystery(int& a, int b) {
      int count = 0;
      while (count < 2) {
            a = a + b/2;
            b = a + 5;
            cout << "a: " << a << " b: " << b << endl;
            count++;
      }
}

int main() {
      int a = 5, b = 10;
      cout << "a: " << a << " b: " << b << endl;
      mystery(a, b);
      cout << "a: " << a << " b: " << b << endl;
}
```

```
a: 5 b: 10
a: 10 b: 15
```

a: 17 b: 22

a: 17 b: 10

2) What does the following code snippet output?

```
void mystery(char code) {
	switch(code) {
		case 'a':
		case 'b':
		case 'c':
			cout << "spooky";
			break;
		case 'd':
			cout << "feeling";
			break;
		case '1':
			cout << " ";
			break;
		case '2':
			cout << "?";
		default:
			cout << endl;
			break;
	}
}

int main() {
	string message = "d1a2c1d#";
	int i = 0;
	do {

	mystery(message[i]);
	i++;
	} while(i < message.length());
}
```

feeling spooky?

```
spooky feeling
```

Programming Problems

1) Create a function *changeString* that accepts two parameters:

- string1: a reference to a string value that does not contain spaces and
- string2: a string consisting of letters that will be used as delimiters. Now, for every character in string2 that appears within string1, replace the letter within string1 with a space.

Note: You may assume that every letter within string2 will be unique.

For example: changeString("Helatelmylcookie", "l") -> "He ate my cookie"
changeString("ShouldeHIstartemylab?", "He") -> "Should Istart mylab?"

```
void changeString(string& word, string delimiters) {
	for (int i = 0; i < word.length(); i++) {
		for (int j = 0; j < delimiters.length(); j++) {
			if (word[i] == delimiters[j])
				word[i] = ' ';
		}
	}
}
```

2) a) Write a function *isPalindrome* that takes in a string and determines if it is a palindrome. A palindrome is a string that reads the same forwards and backwards.

For example: isPalindrome("abcba") returns true isPalindrome("skt_will_win") returns false isPalindrome("z") returns true isPalindrome("") returns true

```
bool isPalindrome(string s) {
	int i = 0;
	int j = s.size() - 1;

	while (i < j) {
		if (s[i] != s[j]) {
			return false;
		}
```

```
            i++;
            j--;
      }
return true;
}
```

b). Now write a function, *isPalindrome2,* that is similar to the function above, except we don't care about spaces in the string.

For example: isPalindrome2("ggnore ero n g g") returns true

```
bool isPalindrome2(string s) {
      int i = 0;
      int j = s.size() - 1;
      while (s[i] == ' ') {
            i++;
      }

      while (s[j] == ' ') {
            j--;
      }

      while (i < j) {
            if (s[i] != s[j])
            return false;
      0

            j--;
      } while (s[j] == ' ');
      }

      return true;
}
```

3) a) Write a function *findRun* that takes in a string of lowercase and uppercase alphabetical characters and returns the character with the longest "run." In other words, return the character that occurs the most times in succession. You may assume that the string is not empty. If two characters have equally long runs, return the first one.

For example: findRun("abbccccdda") returns 'c' findRun("aaaabcbbbbcbcbcbcbcb") returns 'a'

```
char findRun(string s) {
	int maxRun = 0;
	char maxChar = ' ';
	int currRun = 0;
	char currChar = ' ';

	for (char c : s) {
		if (c == currChar)
			currRun++;
		else {
			currChar = c;
			currRun = 1;
		}
		if (currRun > maxRun){
			maxRun = currRun;
			maxChar = currChar;
		}
	}
	return maxChar;
}
```

Note: Another way students may format the `for` loop is `for (int i =0; i<s.length(); i++)` And use `s[i]` instead of `c`, or define another variable c (outside the loop) and let `c =s[i]` (at the beginning of the loop).

b) What would you need to add/change to your previous code so that the case of the string does not matter
For example:
findRun ("aAaabcbbbbcb") returns 'a'

```
c = toLower(c); //at the beginning of the for loop
```

4) Write a function *integerDivide* that does integer division without using the division operator (/). You can assume both parameters will be positive.

For example:

integerDivide(6, 2) returns 3 integerDivide(7, 2) returns 3

```
int integerDivide(int x, int y) {
	int count = 0;
	while (x >= y) {
		x -= y;
		count++;
	}

	return count;
	}
```

5) Write a function findLastLength that takes in a string that consists of uppercase alphabetical characters, lowercase alphabetical characters, and empty space ' ' characters. It returns the length of the last word, unless the last word does not exist, in which case it returns 0.

For example:
findLastLength("Misfits should have won against SKT") returns 3
findLastLength(" ") returns 0

```
int findLastLength(string s) {
	bool foundWord = false;
	int lastLen = 0;
	for (int x = s.length() - 1; x >= 0; x--) {
		bool onAlpha = isalpha(s[x]);
		if (foundWord && !onAlpha)
			break;
		else if (onAlpha) {
			foundWord = true;
			lastLen++;
		}
	}

	return lastLen;
}
```

6) Write a function *intPalindrome* that returns whether or not two positive integers are palindromes. They must be exact palindromes, not 10 and 01, or 1540 and 0451.

For example:

intPalindrome(62, 26) returns true intPalindrome(154, 451) returns true

intPalindrome(10, 1) returns false intPalindrome(25, 56) returns false

```
bool intPalindrome (int first, int second) {
      int smaller, larger;
      int reverse = 0; // calculate the reverse of smaller
      int digit = 0;
// We need this comparison and swap because otherwise,
// trailing zeros won't affect the reverse of first.
// If we leave this out, intPalindrome(10, 1) is true
// but intPalindrome(1, 10) is false!
      if (first < second) {
            smaller = first;
            larger = second;
      }
      else {
            smaller = second;
            larger = first;
      }
      do {
            digit = smaller % 10;
            reverse = (reverse * 10) + digit;
            smaller = smaller / 10;
      } while (smaller != 0);
      return reverse == larger;
}
```

7) Write a function that takes a string representing an english sentence as a parameter and returns a string representing that sentence translated into pig latin. Here are the rules for input and translation:

Input: A string representation of a sentence. The sentence contains words separated by spaces, some of which may be have a capitalized first letter. The sentence ends with a period directly after the last word. You may assume the sentence contains only letters, spaces, and one period at the end.

Ex: "David Smallberg is my favorite professor."

Translation: A sentence can be translated into pig latin word by word, following these rules:

1. If a word in english starts with a vowel, its pig latin translation is simply the

english word with "ay" added at the end.

Ex: "apple" => "appleay" 2. If a word in english starts with a consonant, that consonant is moved to the

end of the word and then "ay" is added at the end.

Ex: "chapter" => "haptercay" 3. If a word has a capitalized first letter, its pig latin translation should also have a

capitalized first letter.

Ex: "David" => "Avidday" Ex: "Eggert" => "Eggertay"

Here is an example translation. Feel free to write any helper functions you may find useful in implementing your pig latin translator function.

Ex: "David Smallberg is my favorite professor." =>

"Avidday Mallbergsay isay ymay avoritefay rofessorpay."

```
#include <cctype>
#include <string>
bool isVowel(char c) {
// Avoids having to write checks for lowercase letters.
      char upper = toupper(c);
      return (upper == 'A' || upper == 'E' || upper == 'I' ||
      upper == 'O' || upper == 'U');
}

string pigLatinWord(string word) {
      string pig_latin_word;
      if (isVowel(word[0])) {
            pig_latin_word = word + "ay";
      }
      else {
// Store the character in a variable because tolower technically
// returns an int, which cannot be added to a string.
            char first_letter = tolower(word[0]);
            pig_latin_word = word.substr(1) + first_letter + "ay";
            if (isupper(word[0])) {
                  pig_latin_word[0] = toupper(pig_latin_word[0]);
            }
```

```
	}

	return pig_latin_word;
}

string pigLatin(string sentence) {
	string translation = "";
	string current_word = "";
	for (int i = 0; i < sentence.length(); i++) {
		if (isalpha(sentence[i])) {
		// Build a word to translate individually.
			current_word += sentence[i];
		}
		else {
		// This will be correct whether sentence[i] is ' ' or '.'
			translation += pigLatinWord(current_word) +
				sentence[i];
			current_word = "";
		}
	}

	return translation;
}
```

8) Write a function that takes a string and (1) reverses the lowercase vowels, then (2) converts all lowercase vowels into uppercase.

Example: reverseLowercaseVowels("begin projects early") should return "bAgEn prEjOcts lErly" reverseLowercaseVowels("hellO world") should return "hOllO wErld" [Notice that the capital O in the original string did not change positions because you only need to reverse all the lowercase vowels. That is, keep the uppercase vowels in the same position.]

```
bool isLowercaseVowel(char c) {
	return (c == 'a' || c == 'e' || c == 'i' ||
	c == 'o' || c == 'u');
}

string reverseLowercaseVowel(string input) {
	int first = 0;
	int last = input.size();
```

```
	// First reverse the lowercase vowels

	while (first < last) {
		if (isLowercaseVowel(input[first])) {
			if (isLowercaseVowel(input[last])) {
			// Swap the vowels
				char temp = input[first];
				input[first] = input[last];
				input[last] = temp;
				first++;
				last--;
			}
			else {
				last--;
			}
		} else {
			first++;
	}
}

// Now convert all lowercase vowels to uppercase

	for (int i = 0; i < input.size(); i++) {
		if (isLowercaseVowel(input[i])) {
			input[i] = toupper(input[i]);
		}
	}
return input;
}
```