Data Structures
BITL3

Worksheet 6

Fall 2022

20 October 2022

---

1. **Warm up:** Answer the following questions.

   (a) What is the difference between functions using *recursion* and using *iteration*?

   (b) What is the difference between *static* arrays and *dynamic* arrays?

   (c) What is the difference between the code `int a[10];` and `int *a; a = new int[10];`?

2. For each of the following tasks, write code that implements the function once using *iteration*, and once using *recursion*. A file `recursion_iteration.cpp` with the function names is given in ORTUS.

   (a) Find the (index of the) larget element of an array of types `double`.

   (b) Compute the greatest common divisor of two positive integers, $m > n$. Use the fact that $\gcd(m, n) = \gcd(m, m\%n)$, where % is the modulo operator.

   (c) Print out an input string, but backwards.

3. This question is about *reading* files, and uses the following uncompiled code. This code is available on ORTUS, as the file `analyze_text.cpp`.

```
1   #include <iostream>
2   #include <fstream>
3   using namespace std;
4
5   int main() {
6       string filename = "text.txt";
7       int word_counter = 0;
8       double average_length = 0;
9
10      ifstream file(filename);
11      string word;
12
13      while (file >> word) {
14          average_length += word.length();
15          word_counter += 1;
16      }
17
18      average_length = average_length/word_counter;
20      cout << "The input file has " << word_counter << " words";
21      cout << "with an average length " << average_length << endl;
22
23      return 1;
24  };
```

   (a) Improve the code so that punctuation is not counted in word length. A function `remove_punctuation` for you to use is defined in the file on ORTUS.

   (b) Define a recursive function `count_e` that counts the number of occurences of the letter `e` in an input string. Adjust the code so that it prints out the average number of times the letter `e` appears in words of the input file.

4. This question is about *recursion*, and uses the following uncompiled code. This code is avialable on ORTUS, as the file `prime.cpp`.

```
1   #include <iostream>
2   using namespace std;
3
4   bool is_composite(int n, int m = 2) {
5       if (m*m > n) { return false; }
6       else if (n % m == 0) { return true; }
7       else { return is_composite(n, m+1); }
8   };
9
10  int main() {
11      int check_this;
12      cout << "Please enter a number: ";
13      cin >> check_this;
14      bool prime_check = is_composite(check_this);
15      if (prime_check) { cout << "The number " << check_this << " is not prime" << endl; }
16      else { cout << "The number " << check_this << " is prime" << endl; }
17      return 0;
18  };
```

(a) How many times will `is_composite` be called if the user inputs 19?

(b) Find two different numbers (one composite and one prime) which, when input by the user, `is_composite` will be called exactly 7 times.

(c) Modify the code of `main` so that the next prime number after the number that the user inputs is printed. If the user has input a prime number, then that same number is printed.

(d) **Bonus 1:** This method is a *"prime number sieve"*, and is very inefficient, because inputting large numbers crashes the program. What is the smallest number that you can input which crashes the program? Is this the same for the program running on different computers?

(e) **Bonus 2:** Continuing task (b), for any positive integer k, is there an input n such that `is_composite` will be called exactly k times, when n is input by the user?