CS 7B - Fall 2018 - take-home problems 1 - Solutions

Turn in your answers to questions 2,4,6 and 7 via email. Questions 1,3 and 5 are paper and pencil. Due Thursday, 10/25

1. Consider the program below:

```
#include <iostream>
int main() {
    int var = 8;
    void* ptr = &var;
    std::cout << "\nptr = " << ptr;
    std::cout << "\n&ptr = " << &ptr;
    std::cout << "\n*ptr = " << *reinterpret_cast<int*>(ptr);
}
```

   (a) This is syntactically correct. Describe what is stored in `ptr`.
   ANS: The variable `ptr` is a pointer that contains the address of the first byte of the `int var` variable.

   (b) Describe what `*reinterpret_cast<int*>(ptr)` represents.
   ANS: The quantity `reinterpret_cast<int*>(ptr)` is a pointer to a type `int`, which, when dereferenced and then inserted into the output stream is interpreted as the value of this `int var`, which is 8.

   (c) Describe what is printed to the console.
   ANS: Literally, when the machine I am presently using (this having likely having happened in the past now) executes this code, it prints

```
ptr = 0026FB38
&ptr = 0026FB2C
*ptr = 8
```

   The pointer `ptr` is declared and initialized so that it contains the address of the variable "`var`". What the output shows is that `ptr` is stored at `0x0026FB2C` and contains the value `0x0026FB38`, which is the address where `var` is stored. When we dereference the pointer with `*ptr`, we get the value stored at `0x0026FB38`, which is the integer value $8 = 0000000000000110_2$, which happens to be less than 8 1/2.

   (d) Assume that memory is stored with the least significant byte first. What would the line
   `std::cout << "*ptr = " << *reinterpret_cast<short*>(ptr);`
   print to the screen, if it replaced the last line?
   ANS: As shown below, the first line gives the address of the the first byte of the `int` var.
   The second line prints the address of the `ptr` itself. Hey–everything has to be somewhere in memory, otherwise it ceases to be a thing.

```
ptr = 0x28ff0c
&ptr = 0x28ff08
*ptr = 8
```

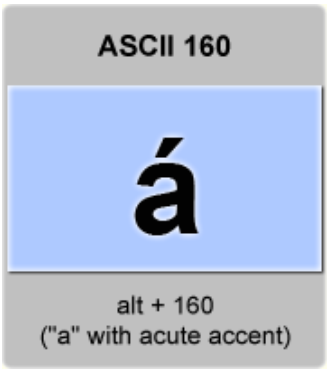
   It's interesting to note that the following code

```
int main() {
int var = 65601;
void* ptr = &var;
std::cout << "\nptr = " << ptr;
std::cout << "\n&ptr = " << &ptr;
std::cout << "\n*ptr = " << *reinterpret_cast<char*>(ptr);
std::cout << "\n*ptr = " << *reinterpret_cast<short*>(ptr);
std::cout << "\n*ptr = " << *reinterpret_cast<int*>(ptr);
}
```

produces

```
ptr = 003CFE94
&ptr = 003CFE88
*ptr = A
*ptr = 65
*ptr = 65601
```

Do you see why?



(e) Would the answer to part (c) above be different if `var = 100000`? Explain.
ANS: In this case the modified code (which is a bit more illuminating) gives us

```
ptr = 0017FAA4
&ptr = 0017FA98
*ptr = á
*ptr = -31072
*ptr = 100000
```

The output of the original code then would have been `*ptr=100000`. Note that $100000(\mod 256) = 160$, which in the extended ascii table is alt+160 ("a" with an acute accent.) To explain by the short version is -31072 requires more deep understanding of how the two's complement comes into play.

2. Write the function `alphabetizeSentences()` to read a text file containing sentences and replace each sentence (a sequence of words ending with a period) with the same words in alphabetical order. For instance, if the file contains this text

   See the dog. Watch the dog chase a cat.

   Then the file will be replaces with this text:

   Dog see the. A cat chase dog the watch.

   Note that first letter of a sentence should be capitalized.
   `void alphabetizeSentences(string fname)`
   ANS:

```
#include <iostream>
using std::cin;
using std::cout;
#include <fstream>
using std::ifstream;
using std::ofstream;
#include <vector>
using std::vector;
#include <string>
using std::string;
#include <algorithm>
using std::sort;

/* Write the function \texttt{alphabetizeSentences()}
to read a text file containing sentences and replace each sentence
(a sequence of words ending with a period) with the same words in
alphabetical order.*/

void alphabetizeSentences(ifstream& ifs) {
  ofstream ofs("alfa01.txt");
  string word;
  vector<string> sentence;
  while (ifs >> word) {
    cout << word << '\n';
    sentence.push_back(word);
    if (word[word.length() - 1] == '.') { //completed sentence
      // Remove period from last word
      sentence[sentence.size()-1] =
        sentence[sentence.size()-1].substr(0, sentence[sentence.size()-1].length() - 1);
      // Make first word lower case
      sentence[0][0] = tolower(sentence[0][0]);
```

```
      // Sort the words into alphabetical order.
      sort(sentence.begin(), sentence.end());
      // Capitalize the first word.
      if (int(sentence[0][0]) > 96) sentence[0][0] -= 32;
      // Put a period at the end.
      sentence[sentence.size()-1] += '.';
      // Write the sentence to the output file
      cout << "sentence.size() = " << sentence.size() << '\n';
      for (string s : sentence) ofs << s << ' ';
      // Clear the decks.
      sentence.clear();
    }
  }
}
int main() {
  string fname;
  cout << "Enter a filename with sentences in it for alphabetizing:'\n'";
  cin >> fname;
  ifstream fs(fname);
  alphabetizeSentences(fs);
  cin.ignore();
  cin.get();
}
```

3. Consider the following complete program:

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include <sstream>
using std::ofstream;
using std::ifstream;
using std::string;
using std::cout;
using std::cin;

void alphabetizeSentences(string fname) {
  string outfile;
  std::ifstream read(fname);
  if (read) {
    /* Get the size of the file  */
    read.seekg(0, std::ios::end);
    std::streampos length = read.tellg();
    read.seekg(0, std::ios::beg);
    /* Use a vector as the buffer.
    * It is exception safe and will be tidied up correctly.
    * This constructor creates a buffer of the correct length.
    From cplusplus.com:
    istream& read (char* s, streamsize n)
    read block of data
    Extracts n characters from the stream and stores them in the array pointed to by s.

    This function simply copies a block of data, without checking its contents nor
    appending a null character at the end.  If the input sequence runs out of characters
    to extract (i.e., the end-of-file is reached) before n characters have been
    successfully read, the array pointed to by s contains all the characters read until
    that point, and both the eofbit and failbit flags are set for the stream.

    Internally, the function accesses the input sequence by first constructing a sentry object
    (with noskipws set to true). Then (if good), it extracts characters from its associated
    stream buffer object as if calling its member functions sbumpc or sgetc, and finally
    destroys the sentry object before returning.

    The number of characters successfully read and stored by this function can be accessed by
    calling member gcount.
    * Then read the whole file into the buffer. */
    std::vector<char> buffer(length);
    read.read(&buffer[0], length);
    //cout << "buffer = ";
    //for(char& ch : buffer) cout << ch;
    /*  Create your string stream.
    * Get the stringbuffer from the stream and set the vector as it source.   */
    std::stringstream localStream;
    localStream.rdbuf()->pubsetbuf(&buffer[0], length);
    cout << "what file would you like to write to? ";
    cin >> outfile;
    ofstream write(outfile);
    localStream << outfile;
    /* Note the buffer is NOT copied, if it goes out of scope
    * the stream will be reading from released memory.    */
  }
}

int main() {
  string fname;
  cout << "\nWhat file would you like to read from?";
  cin >> fname;
  alphabetizeSentences(fname);
  cin.ignore();
  cin.get();
}
```

(a) Describe what's happening on lines 17 and 18.
ANS: The statement `read.seekg(0, std::ios::end);` on line 17 gets the offset of the position at the end of the file relative to the position at the beginning (0). Then line 18 (`std::streampos length = read.tellg();` returns the position in the filestream of the ending (where the `seekg()` function got to) which will then write the length of the file to the variable `length`.

(b) Describe the variable `buffer` created on line 42.
ANS: `vector<char> buffer(length);` declares `buffer` to be a vector of characters initialized to contain `length` characters; that is, enough space for `length` characters is allocated, all initialized to the `NUL` character (ascii value 0).

(c) Describe what happens on line 43.

(d) Line 53 doesn't appear to do anything. Why not?
ANS: No good answer to this yet.

4. Write the function `Similarity()` that computes a measure of similarity between two sorted vectors. The similarity metric is defined as the count of elements the vectors have in common divided by the average size of the vectors. Here are some examples:

v1	v2	Similarity(v1, v2)	Notes
[a, b, c, c, e]	[a, c, c, k, m]	.6	[a, c, c] in common, avg size 5
[a, m]	[w, x, y, z]	0	None in common, avg size 3
[j, k, k]	[k, z]	.4	[k] in common, avg size 2.5

Note that the input vectors are in sorted order. You should use this fact to efficiently implement this operation. Ideally, the function should run in `O(N)` time where `N` is the length of the input vectors.
`double Similarity(vector<char>& v1, vector<char> & v2)`
ANS:

```
#include <iostream>
using std::ostream;
#include <vector>
#include <random>
using std::vector;
using std::cout;
using std::cin;

/*Write the function Similarity() that computes a measure of similarity between two sorted vectors.
The similarity metric is defined as the count of elements the vectors have in common divided by the
average size of the vectors.
*/
double Similarity(vector<char>& v1, vector<char>& v2) {
  double avgsize = ((double)v1.size() + v2.size()) / 2;
  vector<char>::iterator it1 = v1.begin();
  vector<char>::iterator it2 = v2.begin();
  unsigned count{ 0 };
  while(it1!=v1.end() && it2!=v2.end())
    if (*it1 == *it2) {
      ++count;
      ++it1;
      ++it2;
    }
    else if (*it1 < *it2) ++it1;
    else ++it2;
  return count / avgsize;
}

//template<typename T>
ostream& operator<<(ostream& os, const vector<char>& v) {
  os << '{';
  for (unsigned i = 0; i < v.size() - 1; ++i)
    os << v[i] << ", ";
  os << v[v.size() - 1] << '}';
```

```
  return os;
}

int main() {
  vector<char> v1{ 'a', 'b', 'c', 'c', 'e' };
  vector<char> v2{ 'a', 'c', 'c', 'k', 'm' };
  vector<char> v3{ 'a', 'm' };
  vector<char> v4{ 'w', 'x', 'y', 'z' };
  vector<char> v5{ 'j', 'k', 'k' };
  vector<char> v6{ 'k', 'z' };
  //vector<char> v3{ a, m };
  cout << "The similarity between " << v1 << " and " << v2 << " is " << Similarity(v1, v2) << '\n';
  cout << "The similarity between " << v3 << " and " << v4 << " is " << Similarity(v3, v4) << '\n';
  cout << "The similarity between " << v5 << " and " << v6 << " is " << Similarity(v5, v6) << '\n';
  cin.get();
}
```

This produces the output

```
The similarity between {a, b, c, c, e} and {a, c, c, k, m} is 0.6
The similarity between {a, m} and {w, x, y, z} is 0
The similarity between {j, k, k} and {k, z} is 0.4
```

5. Consider the following program:

```
#include <iostream>
#include <string>

template<int N>
class Array {
  int m_Array[N];
public:
  int getSize() const { return N; }
};
int main() {

}
```

(a) What line of code would you write in `main()` that will create an `Array` of 7 `int`s?
ANS: `Array<7> seven;`

(b) How would you modify the class template so you could create an `Array` on `N` objects of any type, `T`?// **ANS:**

```
template<int N, typename T>
class Array {
    T m_Array[N];
public:
    int getSize() const { return N; }
};
int main() {
    Array<7, char> seven;
}
```

(c) How would you implement your modified class definition to create an `Array` of 12 `string`s?
ANS: `Array<12, string> twelve;`

6. Write the function `ExtractStrand()` that extracts a sorted strand from a non-empty vector of integers. The strand is extracted as follows:

   - The first element of the vector is removed and becomes the first element of the strand.
   - The remaining vector elements are considered in order. Each element that can be added to the strand while preserving sorted order is removed from the vector and added to the strand.

   The function returns a new vector containing the sorted strand. The input vector, which was passed by reference, has been modified to remove the extracted numbers, the remaining numbers are preserved in their original order. Your solution can use vectors and primitive variables but no other data structures (no arrays, etc.).
   Here are some examples:

`q`	`ExtractStrand(q)` returns	After call, `q`
`[2, 1, 7, 12, 5, 10, 2]`	`[2, 7, 12]`	`[1, 5, 10, 2]`
`[3, 3, 1, 2, 3, 4, 1]`	`[3, 3, 3, 4]`	`[1, 2, 1]`
`[6, 4, 2, 3, 3]`	`[6]`	`[4, 2, 3, 3]`

ANS:

```
#include <iostream>
using std::cin;
using std::cout;
#include <vector>
using std::vector;
#include <cstdlib>
#include <ctime>

void print(const vector<int>& v) {
  for (int i : v) cout << i << " ";
}

vector<int> ExtractStrand(vector<int>& v) {
  vector<int> strand;
  vector<int>::iterator itv = v.begin();
  strand.push_back(*itv);
  vector<int>::iterator its = strand.begin();
  //v.erase(itv) returns the next valid iterator, if you erase the last element it will point to .
    end()
  itv = v.erase(itv);
  while (itv != v.end())  {
    its = strand.end() - 1;
    if (*itv >= *its) { //move the element from v to strand
      cout << "*itv = " << *itv << ", *its = " << *its << '\n';
      strand.push_back(*itv);
      itv = v.erase(itv);
    }
    else ++itv; //keep that element in v
  }
  return strand;
}
int main() {
  srand(time(0));
  vector<int> v;
  for (int i = 0; i < 10; ++i) v.push_back(rand() % 100);
  cout << "Before ExtractStrand(), v = ";
  print(v);
  cout << '\n';
  vector<int> strand = ExtractStrand(v);
  cout << "After ExtractStrand(), v = ";
  print(v);
  cout << '\n';
  cout << "strand = ";
  print(strand);
  cin.get();
}
```

Here's a typical run of this program

```
Before ExtractStrand(), v = 16 27 15 88 69 10 81 34 64 95
*itv = 27, *its = 16
*itv = 88, *its = 27
*itv = 95, *its = 88
After ExtractStrand(), v = 15 69 10 81 34 64
strand = 16 27 88 95
```

7. The New York Times puzzle page has a new puzzle called the Spelling Bee. Here are the rules:
Create words using letters from the "hive", a collection of 7 letters with one letter at the center.

   - Words must contain at least 4 letters.
   - Words must include the center letter.
   - Words may not be proper nouns or hyphenated.
   - Letters can be used more than once.

   Score points to increase your rating.

   - 4-letter words are worth 1 point each.
   - Longer words earn 1 point per letter.
   - Each puzzle includes at least one "pangram" which uses every letter. These are worth 7 extra points!

   write a program to help you solve the Spelling Bee by putting letters from the hive together and seeing whether or not they're in the ospd.txt file provided on the Calendar. You don't need to bother with the scoring. Use a boolean function `bool in(std::string s, char c)`:

```
bool in(std::string str, char c) {
    const char* s = str.c_str();
    //int i{0};
    while(*s) {
        if(c==*s) return true;
        ++s;
    }
    return false;
}
```

   Here's some pseudocode that may help:

```
while you haven't read all the words from the ospd.txt yet,
read the next word from the ospd.txt
convert your word to a c-string like so:
        const char* cstr = "";
        cstr = word.c_str();
        starting from the first charactor of cstr and while you're not at the end of the cstr
            if that character is not in your hive, break;
            otherwise read the next character in the cstr;

        if at end of cstr then all letters in cstr are in your hive
        if cstr contains the key letter
            your word is a keeper!
```

   ANS:

```cpp
#include <iostream>
#include <fstream>
#include <string>
using std::string;
using namespace std;

// if c is in str return true
bool in(std::string str, char c) {
    const char* s = str.c_str();
    while(*s) {
        if(c==*s) return true;
        ++s;
    }
    return false;
}

int main() {
    std::ifstream ifs("ospd.txt");
    std::ofstream ofs("wordlist0728.txt");
    std::string word;
    std::string letts;
    std::cout << "\nEnter letters: ";
  std::cin >> letts;
  const char* letters = letts.c_str();
    std::cout << "\nEnter the key letter: ";
    char keyLet{65};
    std::cin >> keyLet;
    int i{0}, j{0};
  cout << "letts = " << letts << ", keyLet = " << keyLet << '\n';
    while(ifs>>word) {
        const char* c = "";
        c = word.c_str();
        word = word.c_str();
    while(*c && in(letts, *c)) {
            ++c;
        }
        if (!*c) {
      std::cout << "*c = " << *c << ", in(" << word << ", " << keyLet << ") = " << in(word, keyLet)
    << '\n';
      std::cin.ignore();
      std::cin.get();
    }
        if(!*c && in(word,keyLet)) {
            ofs << word << std::endl;
            std::cout << word << std::endl;
        }
    }
}
```