PIC 10A 1C Problems. Week 10b. TA: Eric Kim [Solutions]

1. strip_char

Write a function `strip_char` that, given a string and a character, returns a new string with the given character removed from the string.
If the user does not provide the character, the function should default to '.':

```
string s1("my.file.txt");
string s2 = strip_char(s1, '.');
string s3 = strip_char(s1);
cout << s2 << endl; // Displays: myfiletxt
cout << s3 << endl; // Displays: myfiletxt
cout << strip_char(s1, 't') << endl; // Displays: my.file.x
```

[Solution]

```
string strip_char(const string& s, const char c = '.') {
  string out;
  for (size_t i = 0; i < s.size(); ++i) {
    if (s[i] != c) {
      out.push_back(s[i]);
    }
  }
  return out;
}
```

2. wds_in_commmon

Write a function wds_in_common that, given two sets of words, returns the words that are in common. Here is the function signature:

```
    set<string> wds_in_common(const set<string>&, const set<string>&);
```

Example:

```
set<string> words1 = { "apple", "pastry", "peach", "turnover", "cobbler" };
set<string> words2 = { "fruit", "apple", "blueberry", "peach" };
set<string> cwords = wds_in_common(words1, words2); // ["apple", "peach"]
```

[Solution]

```
set<string> wds_in_common(const set<string>& wds1,const set<string>& wds2) {
  set<string> out;
  /* First add words from words1 */
  for (string w : wds1) {
    bool is_in_wds2 = (find(begin(wds2), end(wds2), w) != end(wds2));
    if (is_in_wds2) {
      out.insert(w);
    }
  }
  /* Next add words from words2 */
  for (string w : wds2) {
```

```
    if (find(begin(wds1), end(wds1), w) != end(wds1)) {
      out.insert(w);
    }
  }
  return out;
}
```

3. display_vec

Consider the following function display_strings:

```
void display_strings(const vector<string>& vec) {
  cout << "[";
  for (string s : vec) {
    cout << s << " ";
  }
  cout << '\b' << "]"; // '\b' removes final undesired space
}
```

A drawback of this function is that it only works on vector<string>. It won't work for vector<int>, vector<char>, etc.

Write a templated function `display_vec` that will output the elements of the input vector regardless of the type, ie it should work on strings, ints, chars, etc.

In other words, it should work for any type T that has `operator<<` suitably defined:

```
vector<string> vec1 = {"hi", "there"};
vector<int> vec2 = {3, 1, 4, 42};
display_vec(vec1); // Displays: [hi, there]
display_vec(vec2); // Displays: [3, 1, 4, 42]
```

[Solution]

```
template <typename T>
void display_vec(const vector<T>& vec) {
    cout << "[";
    for (T thing : vec) {
       cout << thing << " ";
    }
    cout << '\b' << "]";
}
```

4. count_lines

Write a function `count_lines` that, given a path to a textfile, returns the number of lines in the text file. Suppose the file "mythesis.txt" is a text file containing 12,538 lines of text. Then:

```
    int nblines = count_lines("mythesis.txt"); // nblines is: 12538
```

[Solution]

```
int count_lines(const string& fpath) {
  ifstream ifs(fpath); /// opens file
```

```
  if (ifs.fail()) {
    /* Likely, file does not exist! */
    cout << "ERROR: File could not be opened: " << fpath << endl;
    return -1; // signal that a read error occurred
  }
  int cnt = 0;
  while (!ifs.eof()) {
    string curLine;
    getline(ifs, curLine); // advances ifs to next line
    cnt++;
  }
  return cnt;
}
```

Note: I chose to insert some code that checks to see if the file can even be opened in the first place. You didn't have to insert this check, but this is a really good habit to do in your code, since sometimes you can mistype the name of a file, etc. This is known as "defensive programming", and is an essential technique to create robust programs.

5. int2str

Write a function `int2str` that converts an integer to a string. In particular, there should be two default arguments that allow the user to optionally specify the desired width of the output string, along with the desired fill character. The default values for both should be width=0, and fillchar='0':

```
cout << int2str(42) << endl; // 42
cout << int2str(42, 4, '*') << endl; // **42
cout << int2str(42, 1, '*') << endl; // 42
cout << int2str(42, 4) << endl; // 0042
```

[Solution]

```
string int2str(int x, int width=0, char fillchar='0') {
  ostringstream oss;
  oss << setw(width) << setfill(fillchar) << x;
  return oss.str();
}
```

6. sumprod

Write a function `sumprod` that, given two integers x,y, returns both their sum ***and*** product. It is up to you to define how to return multiple values from a single function.

[Solution]

An easy way is to use the pair class, from <utility>:

```
#include <utility>
pair<int,int> sumprod(int x, int y) {
  return pair<int,int>(x+y, x*y);
}
```

Usage would be:

```
pair<int,int> p1 = sumprod(3, 4);
cout << p1.first; // Displays: 7
cout << p1.second; // Displays: 12
```

Note that the type parameterization pair<int,int> means that pair.first is of type int, and pair.second is of type int. Similarly, pair<int, string> means that pair.first is an int, and pair.second is a string.