Assignment 5

Due. Dec. 1 (Thursday), 17:30.

Programming style (10 marks).

1. (10 marks) Chapter 8, Programming 3, p. 308.
   Suppose you know that all the values in an integer array fall into the range 0 to 9999. Show that it is possible to write a $O(N)$ algorithm to sort arrays with this restriction. Implement your algorithm and evaluate its performance by taking empirical measurements.

   The best way to measure elapsed system time for programs of this sort is to use the ANSI `clock` function, which is exported by the `ctime` interface. The `clock` function takes no arguments and returns the amount of time the processing unit of the computer has used in the execution of the program. The unit of measurement and even the type used to store the result of `clock` differ depending on the type of machine, but you can always convert the system-dependent clock units into seconds by using the following expression:

   ```
   double(clock()) / CLOCKS_PER_SEC;
   ```

   If you record the starting and finishing times in the variable `start` and `finish`, you can use the following code to compute the time required by a calculation:

   ```
   double start, finish, elapsed;

   start = double(clock()) / CLOCKS_PER_SEC;
       <perform some calculation>
   finish = double(clock()) / CLOCKS_PER_SEC;
   elapsed = finish - start;
   ```

   Unfortunately, calculating the time requirements for a program that runs quickly requires some subtlety because there is no guarantee that the system clock unit is precise enough to measure the elapsed time. For example, if you used this strategy to time the process of sorting 10 integers, the odds are good that the time value of `elapsed` at the end of the code fragment would be 0. The reason is that the processing unit on most machines can execute many instructions in the space of a single clock tick–almost certainly enough to get the entire sorting process done for an array of 10 elements. Because the system's internal clock may not tick in the interim, the values recorded for `start` and `finish` are likely to be the same.

   The best way to get around this problem is to repeat the calculation many times between the two calls to the `clock` function. For example, if you want to determine how long it takes to sort 10 numbers, you can perform the sort-10-numbers experiment 1000 times in a row and then divide the total elapsed time by 1000. This strategy gives you a timing measurement that is much more accurate.

   Explain why the performance of the algorithm is so bad for small values of N.

2. (10 marks) Chapter 8, Programming 5, p. 309
   The implementations of the various sort algorithms in Chapter 8 are written to sort a `Vector<int>`, because the `Vector` type is safer and more convenient than arrays. Existing software libraries, however, are more likely to define these functions so that they work with arrays, which means that the prototype for the `Sort` function is

```
void Sort(int array[], int n)
```

where `n` is the effective size of the array.

Revise the implementations of the three sorting algorithms (selection, merge, quick) presented in Chapter 8 so that they use arrays rather than the collection classes from Chapter 4.

3. (10 marks) Chapter 9, Programming 6, p. 336
   Add the necessary code to the scanner package to implement the `setStringOption` extension, which allows the scanner to read quoted strings as a single token. By default, the scanner should continue to treat double-quotation marks just like any other punctuation mark. However, if the client calls

   ```
   scanner.setStringOption(Scanner::ScanQuotesAsStrings);
   ```

   the scanner will instead recognize a quoted string as a single unit. For example, the input line

   ```
   "Hello, world."
   ```

   would be scanned as one single token. Note that the quotation marks are preserved as part of the token so that the client can differentiate a string from other token types.

4. (10 marks) Chapter 10, Programming 12, p. 377.
   Implement the `EditorBuffer` class using the strategy described in the section entitled “Doubly linked lists” in Chapter 10. Be sure to test your implementation as thoroughly as you can. In particular, make sure that you can move the cursor in both directions across parts of the buffer where you have recently made insertions and deletions.