

## CMPT-101, Fall 1999, Assignment 4

### ***The Problem***

We want to print out two paragraphs of English text in side-by-side columns on cout. Here's an example of what we want:<sup>1</sup>

The art of card palming can be brought to a degree of perfection that borders on the wonderful. It is very simple to place one or several cards in the palm and conceal them by partly closing and turning the palm downward, or inward; but it is entirely another matter to palm them from the deck in such a manner that the most critical observer would not even suspect, let alone detect, the action.

Excessive vanity proves the undoing of many experts. The temptation to show off is great. He has become a past master in his profession. He can laugh at luck and defy the law of chance. His fortune is literally at his finger's ends, yet he must never admit his skill or grow chesty over his ability. It requires the philosophy of the stoic to possess any great superiority and refrain from boasting to friend or foe. He must be content to rank with the common herd. In short, the professional player must never slop over. One single display of dexterity and his usefulness is past in that particular company, and the reputation is likely to precede him in many another.

In this example, both columns have the same width, and they are separated by a gap of a few spaces so their lines don't run together. Creating multi-column text like this is tricky since you can only print to cout. You must print the first line of the first paragraph, then some spaces, and then the first line of the second paragraph. Then you must print the second line of the first paragraph, some spaces, and then the second line of the second paragraph, and so on. Both paragraphs are left-justified and "ragged right".

### ***One Solution***

Here is an outline of one solution to the problem of formatting multi-column text. Your job is to write all the specific C++ code, and to follow the given design constraints.

---

<sup>1</sup> The two quotes are from *The Expert at the Card Table*, by S. W. Erdnase, 1902.

The main idea is the use of a class called `Paragraph`. `Paragraph` objects store the text and width of one paragraph. The width is the longest any line can be in the paragraph. As described below, the public interface of `Paragraph` is given, and it is your job to implement it, i.e. provide all the private variables and all the member function bodies. You *can not* add or change any part of the public interface of `Paragraph`, but you can put whatever you need into the private part. `Paragraph` should be totally self-contained, and any non-standard functions that it uses must be declared as private member functions.

For this assignment, a `Paragraph`'s text can be stored as a single `string`. If you prefer, you can use a `vector`, but it's not necessary. Do *not* use C++'s built-in arrays.

The `Paragraph` class has only four public member functions (the actual C++ class is given at the end of the assignment):

- One constructor, that takes a `string` and a positive width. If the input `string` has any `\n` characters, then the constructor replaces each by a space.
- An accessor, named `width()`, that returns the width of the paragraph. There is no accessor for the paragraph's text, and neither the text nor the width can be changed after a `Paragraph` is constructed.
- A mutator, named `nextLine()`, that returns one line of the `Paragraph`'s text. This line should be less than or equal to the width of the paragraph, and the line's final word should be a complete word; i.e. the text must be split only at spaces, never across words. It's possible that a paragraph could contain words that are longer than its width, so in that case the best you can do is to return an overly long word as the entire line.
- An accessor, named `hasMoreLines()`, that returns `true` just when there's text left in the paragraph. For example, in a paragraph with 10 lines, the first call to `nextLine()` will return the first line, the second call will return the second, and so on to the tenth call, which will return the tenth line. Before each of these calls, `hasMoreLines()` returns `true`, but after the tenth call to `nextLine()`, `hasMoreLines()` returns `false`. When `hasMoreLines()` is `false`, `nextLine()` returns an empty string. There is no way to "reset" a `Paragraph`, so once `hasMoreLines()` returns `false`, it will return `false` forever (technically, this means `Paragraphs` are *forward iterators*).

You must implement each of these member functions; the `Paragraph` class interface is given at the end of the assignment. Again: you are not allowed to change, in any way, the public member functions of `Paragraph`. You can not add any public variables or public

functions, nor can you remove or alter any of the existing ones. However, you can add whatever *private* functions or variables you need.

### Using Paragraph Objects

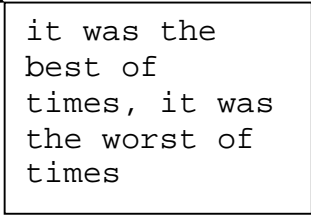
Here's a function that prints out a single paragraph that is left-justified and ragged-right (of course, you must implement Paragraph before this will work!):

```
string left_justified(Paragraph par)
{
    string result;
    while (par.hasMoreLines())
        result = result + par.nextLine() + "\n";

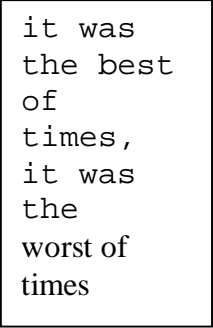
    return result;
}
```

You can call it like this:

```
string text("it was the best of times, it was the worst of times");
cout << left_justified(Paragraph(text,10))
      << "\n\n"
      << left_justified(Paragraph(text,15));
```



it was the  
best of  
times, it was  
the worst of  
times



it was  
the best  
of  
times,  
it was  
the  
worst of  
times

In the first box, each line is at most 10 characters long, and at most 15 in the second box. By changing the width, you can make a paragraph appear as thin or as wide as you like.

After you have written and tested your Paragraph class, write a function called `two_columns` that returns a string (like `left_justified` does) containing the text of the two columns side by side. It should take three parameters: the two Paragraphs, and a positive int representing the size of the gap between the columns. For example, the sample paragraphs at the start were printed like this:

```
cout << two_columns(Paragraph(quote1,30),
                    Paragraph(quote2,30), 4);
```

That is, both columns have a maximum line length of 30, and there's a gap of 4 spaces between the columns for neatness.

### ***What to Hand In***

Write a C++ program that implements the `Paragraph` class, and also the function `two_columns`, as described above. Read in two paragraphs using `getline`; prompt the user to enter the first paragraph, and then prompt them to enter the second. The user indicates the end of a paragraph by putting a “.” (a period) on a line all by itself.

Follow all the usual hand-in instructions for assignments. We want to see all the C++ code you wrote for this assignment. Most importantly, we want to see good implementations of `Paragraph` and `two_columns`.

Hand in *four* test runs of your program, using text of your choice. Your test runs should show the following:

- Two samples showing the *same* text in two side-by-side columns, but with different widths for each test run. These test runs are meant to show off what the `Paragraph` class is capable of doing.
- One test run where the first paragraph has *no* text (i.e. is an empty string).
- One test run showing what happens when you give `two_columns` a paragraph whose width is too small, e.g. it has one or more lines that overflow the width. This should make your output look messy, but it's important to clearly show the limitations of your program.

Make sure to include something in your Features, Limitations, and Bugs page. If your program is perfect in all ways, then at least suggest some features that could be added to make it more useful.

### ***The Paragraph Class --- Do Not Alter the Public Interface!!***

```
class Paragraph {
public:
    Paragraph(string text, int width);  // constructor
    /*
        PURPOSE: constructs a paragraph containing the given text,
                  and with the given width
        RECEIVES: text - a string; every \n in text should be replaced
                  by a space
                  width - a positive number representing the maximum
                  length of any line in the paragraph
    */

    int width() const;  // accessor
    /*
        PURPOSE: returns the width of this Paragraph
        RETURNS: the width of this Paragraph
    */

    bool hasMoreLines();  // accessor
    /*
        PURPOSE: indicates if this Paragraph has any more lines to
                  return in calls to nextLine()
        RETURNS: true if a call to nextLine() will return a line in
                  this Paragraph; false if all lines have been returned
        NOTE: once hasMoreLines() returns false, it will always return
              false; there is no way to "reset" a Paragraph object ---
              once they are done they can't be used any more
    */

    string nextLine();  // mutator
    /*
        PURPOSE: returns the next line of this Paragraph
        RETURNS: the next line of the text; this line is at most width()
                  characters long; if hasMoreLines() is false, then an
                  empty string is returned
    */

private:
    // ... you decide what goes here in the private part ...
}; // class Paragraph
```