Project Strings + Performance

Due: October 18, 2019 at 11:59pm

1 Overview

This assignment was previously two separate projects — Strings and Performance — that for logistical reasons were combined into a single project with two parts.

You should approach each of the two parts as its own project, independent of the other part.

In the Strings portion of this project, you will implement a subset of C's string manipulation library.

In the Performance portion, you will optimize the performance of two functions using techniques you learned in class.

2 Install the Stencil

To get started, run the following command in a terminal:

```
cs0330_install stringsperf
```

This will install the stencil code in your `course/cs0330/stringsperf` directory.

The only files you should edit are `strings/strings.c`, `perf/poly.c`, and `perf/polycol.c`

3 Part 1: Strings

Professor Doeppner is deep in the ocean, and has just come across a couple of shrimp: the Twin-Stripe Crinoid Shrimp and the Peacock Tail Anemone Shrimp. Unfortunately, animals have weird and complicated latin names, and our diver will have to be able to distinguish precisely between Periclimenes affinis and Periclimenes brevicarpalis if he is to be taken seriously as a scientist. Help him preserve his good name within the diving community by implementing C's string manipulation functions!

String manipulation is an important concept in computer science, and it is something that comes up very often in systems programming. Most programming languages have a string library that relieves programmers from writing their own string operations for every program. The C Standard Library has some excellent string manipulation facilities, but we want to assemble one ourselves!

3.1 Assignment

For this part of the project, you will implement a subset of C's standard library string functions. You will use these functions in the upcoming Shell assignments to tokenize and search strings.

You should implement the following functions, all of which are documented extensively in the strings.c stencil file. As a hint, you may need to reuse earlier functions in later functions, so it might help to write these functions in order.

Your implementations should be efficient, but do not optimize your code at the expense of readability. (You will get a chance to write highly optimized code in Performance!) Please comment your code and explain any complicated logic. Your implementations should be at least as fast as the baseline times listed in Section 3.3 for full credit.

Here are the functions you should write, along with example uses of each:

- **`size_t strlen(const char *s);`**
  `strlen` computes the length of a string, excluding the terminating null byte.

```
size_t len = strlen("ALGOT / SKADIS");  // len = 14
```

- **`size_t strspn(const char *s, const char *accept);`**
  `strspn` computes the number of bytes in the largest prefix of s that contains only characters from accept.

```
char *s = "Design your own ELVARLI storage systems";
char *accept = "Design your ELVARLI";
size_t span = strspn(s, accept); // span = 13
```

- **`size_t strcspn(const char *s, const char *reject);`**
  `strcspn` computes the number of bytes in the largest prefix of s that contains only characters not in reject.

```
char *s = "coming up with example strings gets hard";
char *reject = "breakingthe4thwall";
size_t span = strcspn(s, reject); // span = 3
```

- **`int strncmp(const char *s1, const char *s2, size t n);`**
  `strncmp` is similar to the behavior of **`strcmp`**, which you do not have to implement for this assignment. **`strcmp`** compares two strings and returns a number less than, equal to, or greater than 0, if **`s1`** is found to be lexicographically less than, equal to, or greater than **`s2`**.
  `strncmp` is similar, except that it only compares the first (at most) **`n`** bytes of **`s1`** and **`s2`**.

```
int result = strncmp("ABCXYZ", "ABCXYZAAO", 6); // result = 0
```

- **`char *strncpy(char *dest, const char *src, size t n);`**
  `strncpy` is similar to the behavior of **`strcpy`**, which you do not have to implement for this assignment. **`strcpy`** copies the contents of **`src`** into **`dest`**
  `strncpy` is similar, except that it only copies the first (at most) **`n`** bytes from **`src`** into **`dest`**.

```
char dest[] = "MACKAPAR";
strncpy(dest, "TJUSIG", 3); // dest now becomes "TJUKAPAR"
```

Hint: *Think about why we might have used a `char[]` instead of a `char*` for `dest`.*

- **`char *strstr(const char *haystack, const char *needle);`**
  `strstr` finds the first occurrence of the string needle in the string haystack.

```
char *needle = "NYMANE";
char *haystack = "---HOLJES---NYMANE---NYMANE---";
char *location = strstr(haystack, needle);
// location = "NYMANE---NYMANE---"
```

- **`char *strtok(char *str, const char *delim);`**
  `strtok` returns a pointer to the first segment of str that does not contain any characters in delim. **`strtok`** should return non-empty token strings or **`NULL`** if there are no more tokens in **`str`**. Note that **`strtok`** is stateful; to extract a sequence of tokens from the same string, you must make multiple calls by specifying the corpus string in the first call and **`NULL`** each time after.
  Hint: **`strtok`** makes no promise to leave str untouched. You may overwrite characters with null terminators if you'd like.
  Hint: It may be helpful to think of delim as a set of characters that happens to be stored as a list, rather than as a string itself.
  Hint: The **`static`** keyword may be useful when writing **`strtok`**.

```
char *delim = "-";
// string is a char * that equals "---I---Love---33---"
char *token1 = strtok(string, delim); // token1 = "I"
char *token2 = strtok(NULL, delim); // token2 = "Love"
```

Remember that in C, you can think of the type **`char *`** as any of three things: a pointer to a character, a string, or an array of characters. These are all the same! Because of this, you can index into a string like an array.

3.2 Allowed Library Functions and Resources

Read the man pages for more information about each of the string functions if you're confused by the explanation, or come to hours and discuss the functions in more detail. All functions that you'll need to use are defined in **`strings.c`**. **There are no external functions allowed**.

3.3 Testing

We have included a few tests in the main function of the **`tests.c`** file. These tests will test the expected functionality of this string library. While they do cover basic functionality, we encourage you to write more of your own tests so that you can be sure that your functions work correctly before using them to implement subsequent functions.

Please do not modify the tests that are already in the file, as this will only make it harder for you to confirm that everything is working. Also, we will test your code with no compiler optimizations, so do not use a compiler flag to improve your times.

To build the test executable, run:

```
make
```

To test your work against the entire test suite, run:

```
./run_tests [number of repetitions] all
```

To test your work against a specific test or list of tests, run:

```
./run_tests [number of repetitions] <test name(s)>
```

We will be testing your code's efficiency with one million repetitions. If you do not specify a number when running the tests, it will default to one million.

Here are some reference times you should shoot for. Each time is in milliseconds.

	Baseline	Optimized	System
`strlen`	824ms	199ms	86ms
`strspn`	2s 339ms	124ms	93ms
`strcspn`	2s 365ms	141ms	99ms
`strncmp`	1s 956ms	199ms	177ms
`strncpy`	490ms	302ms	287ms
`strstr`	487ms	149ms	32ms
`strtok`	371ms	224ms	177ms

For full credit, we are looking for times that run faster than the baseline. If you want a challenge, see if you can reach the optimized times! The system column shows the times of the actual system functions.

4 Part 2: Performance

4.1 Poly

The number of sharks in the beach at any given time is dependent on a variety of factors, including water temperature, depth, and number of swimmers. In order to plan the optimal beach day, it's important to take these circumstances into account, so Professor Doeppner developed a formula to calculate the number of sharks chillin' at the beach at any moment. However, Tom neglected to include compiler optimizations, causing surfers to forgo the lengthy calculations and just hit the waves. Help speed up Tom's program and save the surfers from the sharks!

4.1.1 Assignment

The purpose of this assignment is to utilize various techniques to optimize the speed of C code. In particular, you will be optimizing the performance of a function, **`poly`**, and explaining the techniques that you used.

The **`poly`** function evaluates a polynomial whose coefficients are given by a, i.e.

$$a[0] + a[1]x + a[2]x^2 + a[3]x^3 + \ldots + a[n]x^n$$

where *n* is the degree of the polynomial. A baseline implementation of **`poly`** is provided for you and replicated below. Your task is to identify and implement ways to improve this function's performance:

```
double poly(double a[], double x, int degree) {
      long i;
      double result = a[0];
      double xpwr = x;
      for (i=1; i<=degree; i++) {
            result += a[i] * xpwr;
            xpwr = x * xpwr;
      }
      return result;
}
```

Please note that to implement a performant `poly` function, you may need to restrict the range of values you can pass in for variable `degree`. You do not have to worry about the result overflowing for extremely large values for `degree`, but you must at a minimum support degrees greater than or equal to 15 (you don't need to worry about the result overflowing when the number of degrees is sufficiently high).

4.1.2 Horner's Method

Horner's Method is an algorithm for polynomial evaluation that reduces the number of operations performed. Given a polynomial

$$p(x) = \sum_{i=0}^{n} a_i x^i$$

Horner's Method factors the polynomial into the new expression

$$p(x) = a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + a_n x)...))$$

Intuitively, this solution halves the number of multiplications necessary for solving a polynomial. Take the following polynomial for example:

$$p(x) = x + 3x^2 + 2x^3$$

To apply Horner's Method to this polynomial, we would rearrange *p(x)* as follows:

$$p(x) = 0 + x(1 + x(3 + 2x))$$

This technique should serve as a starting point for your improved **`poly`** implementation.

4.2 PolyCol

After implementing the appropriate optimizations, Tom's Shark Calculator™ gained immense popularity among the surfin' community, and the people want more! To satisfy demand, Tom has cleverly expanded his program to calculate the shark density at a number of beaches at the same time. However, our hero still remains woefully ignorant of compiler optimizations, and his program has been a commercial failure due to its slowness, bringing hard times upon his company and his family. Help save Tom's business by optimizing his program!

4.2.1 Assignment

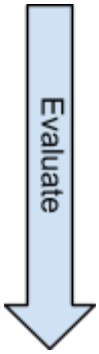
In this part of the performance assignment, you are responsible for optimizing the performance of **`polycol`**. This is a similar problem to `poly`, except that you must simultaneously compute *multiple* polynomials, the coefficients of each making up a column in an *n x n* matrix. For example, given polynomials

$$p_a(x) = a[0] + a[1]x + a[2]x^2 + a[3]x^3 + \ldots + a[n]x^n$$
$$p_b(x) = b[0] + b[1]x + b[2]x^2 + b[3]x^3 + \ldots + b[n]x^n$$
$$p_c(x) = c[0] + c[1]x + c[2]x^2 + c[3]x^3 + \ldots + c[n]x^n$$
$$\ldots$$

The given and resulting matrices would look like this:

	$p_a(x)$	$p_b(x)$	$p_c(x)$	$p_d(x)$	$p_e(x)$	$p_f(x)$

j=0	a[0]	b[0]	c[0]	d[0]	e[0]	f[0]
j=1	a[1]	b[1]	c[1]	d[1]	e[1]	f[1]
j=2	a[2]	b[2]	c[2]	d[2]	e[2]	f[2]
j=3	a[3]	b[3]	c[3]	d[3]	e[3]	f[3]
j=4	a[4]	b[4]	c[4]	d[4]	e[4]	f[4]
j=5	a[5]	b[5]	c[5]	d[5]	e[5]	f[5]



res[0]	res[1]	res[2]	res[3]	res[4]	res[5]

Where `res[0]` = $a[0] + a[1]x + a[2]x^2 + a[3]x^3 + a[4]x^4 + a[5]x^5$ for a given value of x.

In other words, each element `[j][i]` of the matrix is the j^{th} coefficient of polynomial `i`. Your `res` matrix, then, should be a *1 x i* matrix with the computed polynomials in their respective columns.

Please note that the values of x in this computation are $x = i / N$, where `i` is the current column and `N` is the total number of columns. We are using these values of x instead of the standard x=0,1,2,...n to prevent overflow.

The naive approach of this computation, provided in the stencil and replicated below, goes through the columns one at a time and evaluates the contents to find the final *1 x n* matrix.

```
void polycol(int n, double *res, double mat[n][n]) {
    memset(res, '\0', sizeof(double) * n);
i
      for(int i = 0; i < n; i++) {
            res[i] = mat[0][i];
            // `i` must be cast to a double or the result will be rounded to
0
            double x = ((double) i / n);
            double xpwr = x;

            for(int j = 1; j < n; j++) {
                  res[i] += mat[j][i] * xpwr;
                  xpwr *= x;
            }
      }
}
```

This solution, however, does not take advantage of any of the optimizations we learned in class, and is therefore pretty slow. All the surfers will be eaten alive if you don't figure out how to make this run faster!

Hint: You can assume there are an even number of rows and columns in mat (i.e. that the polynomials calculated are of an odd order).

4.3 Getting Started

Before tackling this problem, try to consider what makes this computation so slow. This will help you determine how to improve efficiency. Although **`poly`** and **`polycol`** are separate projects that will be graded separately, we strongly recommend fully completing **`poly`** before beginning **`polycol`**.

4.4 Testing

There are several files in the *performance/poly* directory, but you only need to worry about filling in **`poly.c`.** Similarly, the only file in *performance/polycol* that you need to fill in is `polycol.c`. Each file contains the naive implementations of the respective codes. After building your code, you can run

`./poly` or **`./polycol`**

To test the respective codes. Running this code will print the following:

- `Reference Real time` and `Reference CPU time`, which represent your target values. You should try to get your performance time to be as close to the reference as possible
- `Your Real time` and `Your CPU time`. These values tell you the efficiency of your own code.
- Either "`Your result is correct!`" or "`Results differ`". Please note that you cannot get *any* points for performance if your result is not correct.
- Either "`Your implementation was ___ % slower than the reference`" or "`Your implementation was ___% faster than the reference! Congratulations!!!`"

You will be graded on this project based on the following categories.

- **Functionality**
  - Your code should correctly compute the **`poly`** and **`polycol`** functions' output. A correct output is necessary to receive performance points!
  - Note: you don't have to perform error-checking on the arguments passed to **`poly`** and **`polycol`**: this is done for you by the support code.
- **Performance**

  - **`poly` and `polycol`** run quickly. The faster your program runs, the more points you will receive. Your code should aim to be as fast as the reference code.
  - To get **full performance points** your code should be **no more** than **10% slower than** the **reference solution** for both **`poly` and `polycol`.**
  - You will not receive any performance points if your code does not produce correct output.
- **Explanation**
  - You should explain all optimizations that you made in **`poly` and `polycol`** in your README. This will be a significant portion of your grade, so make sure to explain what optimizations you made and why they make your program faster. If you considered any alternatives, explain why you chose to implement your function the way you did and what tradeoffs you may have made.

5 Grading

See the table below for **guaranteed** grade cutoffs. If you do not meet the threshold for a given letter grade, you may still receive that grade after Professor Doeppner applies a curve (you will only ever be curved up).

Because this is a two part project, we are splitting grade guarantees in the parts. If you reach a different guarantee requirement in each section, you are guaranteed the lower of the two. This does not prevent you from reaching a higher grade, depending on the distribution - You are simply guaranteed to reach the lower of the two. *For example, if you reach an A on Strings, but a B on Performance, you will receive AT LEAST a B on the assignment.*

Grade	Strings	Performance
A	Full functionality, no segfaults, and performance faster than our baseline. Explanation of strstr and strtok in README.	Multiple large optimizations & runtimes within 10% of reference for both poly and polycol, with full explanations. "**`./poly`**" and "**`./polycol`**" should return "**`Your result is correct!`**"
B	Full functionality on all but one, no segfaults on those with full functionality, and performance under our baseline. Explanation of strstr and strtok in README (unless one of those is the one missed).	More than one large optimization on at least one problem, and runtimes within 50% of reference for both poly and polycol. "**`./poly`**" and "**`./polycol`**" should return "**`Your result is correct!`**"
C	Full functionality on all but two, no segfaults on those with full functionality (except on strtok), and	At least one large optimization and runtimes within 150% of reference for both poly and polycol. "**`./poly`**" and "**`./polycol`**" should

	performance under our baseline.	return **`"Your result is correct!"`**
D	You can get your project checked off for a C up until the next project deadline	

For these cutoffs to apply, you must meet the requirements of our style guide 😜

6 Handing In

To hand in your project, run the command:

`cs0330_handin stringsperf`

from your **`stringsperf`** directory.

Your handin must be contain two directories — **`strings`** and **`perf`** — that separate your work for the two parts of this project. The directories should be organized as follows:

- **`strings`** directory
    - All **`.c`** and **`.h`** files containing code you have written for the Strings portion of this assignment.
    - A ***`README`*** explaining your approaches to **`strstr`** and **`strtok`**. You should also document any known bugs and collaborators.
- **`perf`** directory
    - All **`.c`** and **`.h`** files containing code you have written for the Performance portion of this assignment.
    - A *README* explaining your optimizations, as described in the Performance section of the handout. You should also document any known bugs and collaborators.

Ensure that your code is properly formatted before handin.
Consult the **C Style document** (which is on the website) for some pointers on C coding style. Note that you can run a style formatting script in order to make your code match some of the style specifications. To use the script, run the command

```
cs0330_reformat <file1> <file2> ...
```

Check the style guide for more information.

Note: *the reformat script should only be used on .c and .h files*

If you wish to change your handin, you can do so by re-running the handin script. Only the most recent handin will be graded.

Important note: *If you have already handed in your assignment, but plan to hand in again after the TAs start grading at noon on Tuesday, October 23rd, in addition to running the regular handin script (*`cs0330_handin stringsperf`*), you must run* `cs0330_grade_me_late stringsperf` *to inform us not to start grading you yet. You must run the script by noon on 10/19(earlier than usual because it's due friday)*. **If you run this script, you will get grades back later than other students.**

If you do not run this script, the TAs will proceed to grade whatever you have already handed in, and you will receive a grade report with the rest of the class that is based on the code you handed in before we started grading.

If something changes, you can run the script with the `--undo` flag (before noon on 10/19) to tell us to grade you on-time and with the `--info` flag to check if you're currently on the list for late grading.

These instructions apply to all projects unless otherwise stated on the handout.