



Gear Up

Strings + Performance

PROJECT OVERVIEW

- Return to the land of C
- Learn how to manipulate strings at the character level and how basic tokenization works
- Learn techniques to optimize code performance

TOPICS YOU SHOULD REVIEW

- C arrays and pointers
 - What does it mean for strings to be “null-terminated”?
- Performance optimization techniques (stay tuned)

QUICK MATH REVIEW: POLY


- The first half of performance is improving a function we call `poly`
- `poly` is a function for calculating polynomials for a given value of `x`
- Given polynomial $x + 3x^2 + 2x^3$, $x = 2$, `poly` should return

$$0(2)^0 + 1(2)^1 + 3(2)^2 + 2(2)^3 = \mathbf{30}$$

ROADMAP


cs0330_install stringsperf

- Strings
 - Read the stencil file and handout carefully.
 - Fill in each function and test using the provided tester.
 - Tweak the performance to beat the goal times.
- Performance
 - Read the handout and think about how you can add to the provided stencil code.
 - Apply techniques used in class to crank up the efficiency!



NOTE: *You shouldn't need advanced performance techniques in Strings.*

We'd like you to consider only runtime and space efficiency for this assignment.



STRING MANIPULATION

- Strings are just arrays!
 - You can insert into them.
- Strings end with a null character, `'\0'`.
- What happens when you insert a null character into a string?

```
char *str = "hey, 33!"
```

```
str[3] = '\0';
```

```
// str now points to "hey"
```

Data is lost! No one will ever see it again...

PERFORMANCE OPTIMIZATION TECHNIQUES

- Pipelining
- Branch prediction
- Loop unrolling
- Data dependencies
- Cache-friendly code
- Iteration vs recursion
- Inline functions

EXAMPLE OF LOOP UNROLLING

Unoptimized Version:

```
for (int i = 0; i < n; ++i) {  
    A(i) ;  
    B(i) ;  
    C(i) ;  
}
```

Unrolled Version:

```
for (int i = 0; i < (n - 2); i+=3){  
    A(i);  
    A(i+1);  
    A(i+2) ;  
    B(i) ;  
    B(i+1);  
    B(i+2) ;  
    C(i) ;  
    C(i+1);  
    C(i+2);  
}
```

TESTING STRINGS

We give you a suite to test your functionality, however we encourage you to write your own unit tests when applicable.

When in doubt of what a function does, check the man pages.

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

Your times should be faster than the “baseline” column of the table in the handout.

TESTING PERFORMANCE

To test the speed of your poly.c or your addcol.c run your programs normally:

```
./poly
```

```
./polycol
```

TIPS

- Strings
 - Read the man pages for the string functions carefully to understand what the outputs should be.
 - Write the string functions in order. You may be able to use early functions in the later ones.
- Performance
 - Write out “slow” solution first.
 - *Then* use common performance optimization techniques to speed it up.

So WHY Am I DOING This ?

- String manipulation is an important CS concept and has many applications.
 - Code compilation:
tokenize > parse > intermediate representation > etc.
- You'll be working with arbitrary strings in the Shell projects and it will be helpful to start thinking about text processing.

SO WHY AM I DOING THIS ? (CONT...)

- As a programmer, optimization techniques are an important part of your toolkit.
- System code is run much more often across platforms, and having good performance is important for users.
- Working with large amounts of data, these tools will be handy in saving tons of computation time and resources.
- There's usually a fine balance between readability and performance.

DO YOU HAVE ANY QUESTIONS?

