TiMRS

Timers, Made Readable and Simple

Jeff Kline
Faisal Rahman
Daniel Rindone
Eric Webb

Software Development Tools

Languages Used

Code written in:

- C, LLVM, OCaml

Project Management

- Managed and organized events through Slack
- Code sharing through Github

Resources Utilized

- Linux Programming Interface, Ch. 23: Timers
- Brute force programming and cynical humor

Language Overview / Motivation

A language designed to be accessible through a clearly defined set of parameters so that timers can be made by both experienced and entry-level programmers alike.

Usability, Simplicity, Customization, Efficiency

- Imperative
- Statically scoped
- Monomorphic

Reflects an emulation of C-like syntax

Language Overview and Comparison

Where we started

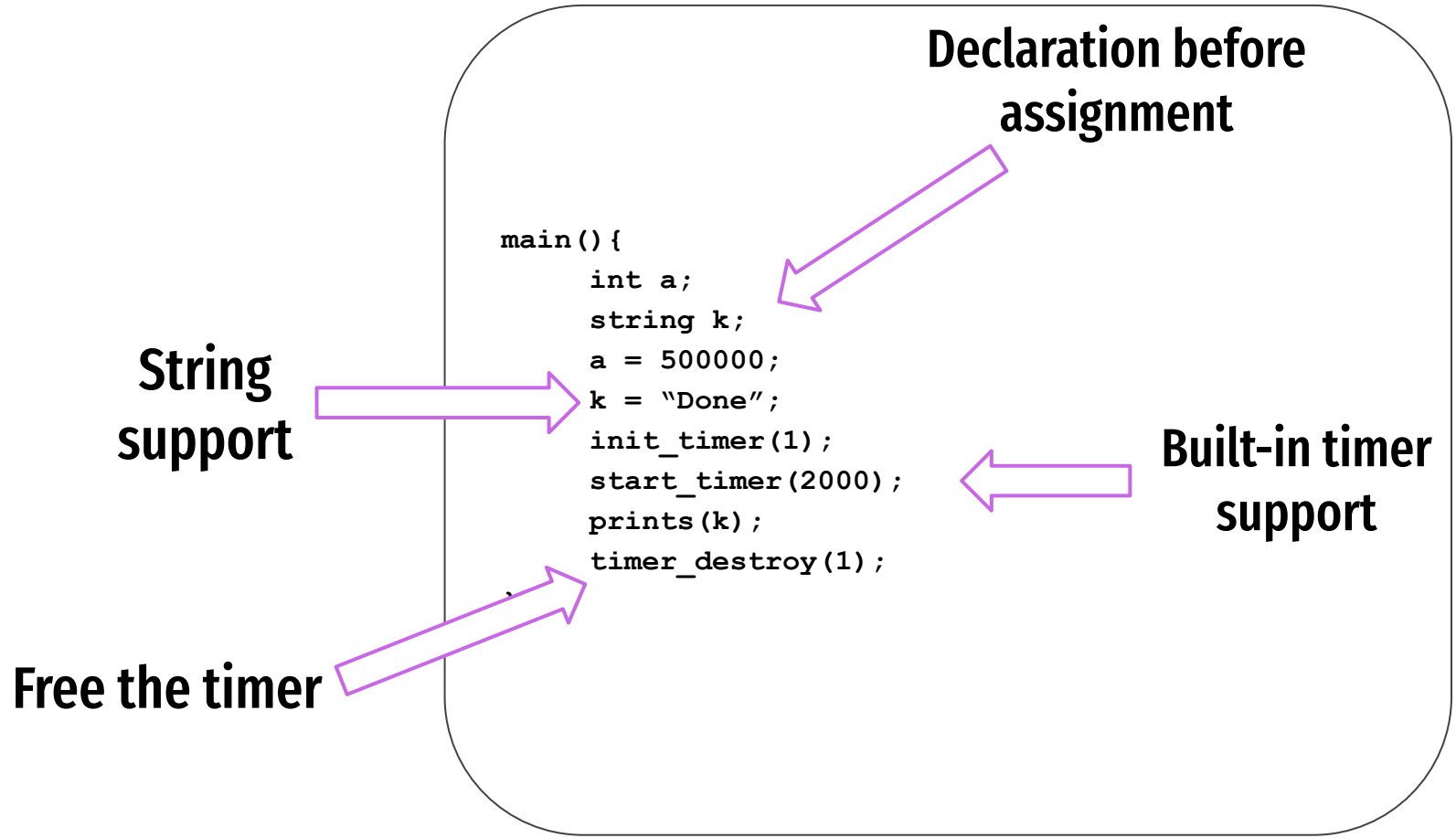
```
5 rounds:
      5 min 30 secs
      2 min then print “Done”
      5 min then alert “STOP”
      30 sec then 2 rounds: 10 sec
2 rounds of:
           30 sec
           15 sec
30 min
30 sec
```

How it’s going

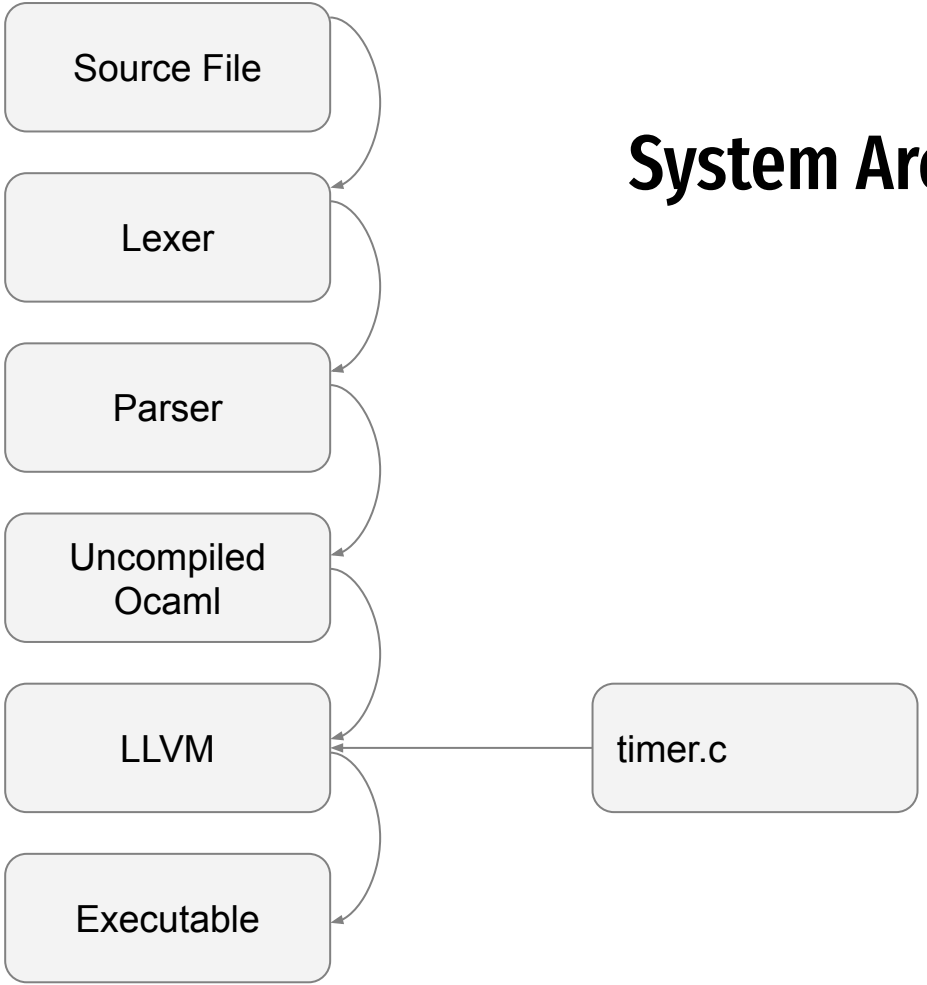
```
main(){
      int a;
      string k;
      a = 5000;
      k = “Done”;
      init_timer(1);
      start_timer(a);
      prints(k);
      timer_destroy(1);
}
```

TiMRS in a Slide

Example



System Architecture/Pipeline



Syntax Basics

Control Flow `if, elif, else, while`

Arithmetic Operators

`+, -, *, /, and ()`	Standard operations for mathematical arithmetic

Assignment Operator

`=`	Assigns the variable on the right hand side to the variable on the left

Comments `/* */`

Syntax Basics

Boolean Operators

||	Example: `a || b`
&&	Example: `b && a`
!	Example: `!b && a`

Conditional Operators

!=	Inequality
==	Equality
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Language Features

Data Types

`int, float, string, bool, void`

Function Calling

Syntax:

```
/* function declaration */
int name(list of parameters)
{
     statement;
}

/* function call */
name(list of parameters)
```

Example:

```
/* user-defined function
declaration */
int countdown(int x, string msg)
{
     if (x <= 30)
     return msg;
}

/* function call */
countdown(10, “done”)
```

Syntax Basics

TiMRS-Specific Commands

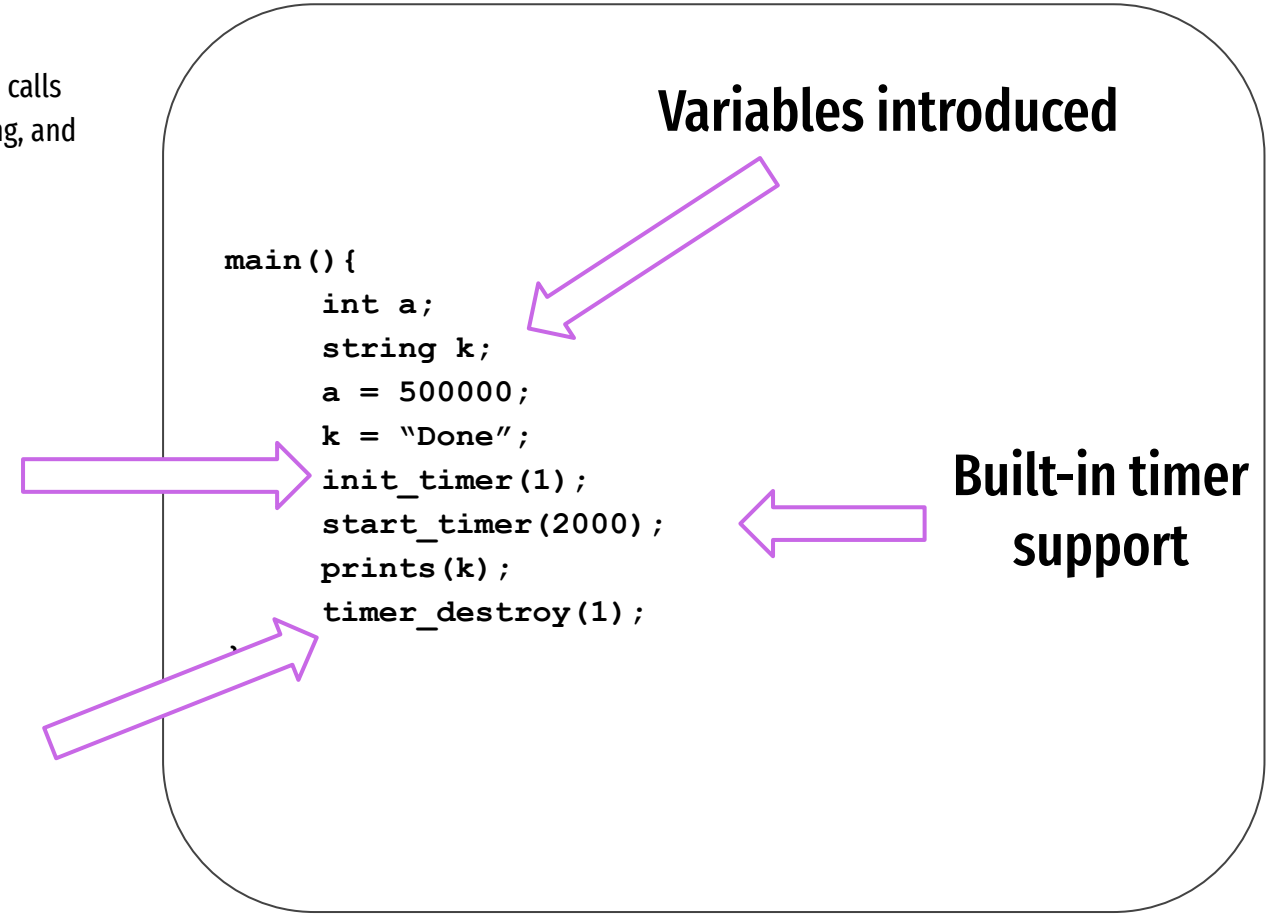
`init_timer()`	Create an instance of a timer
`start_timer()`	Starts a timing event
`timer_destroy()`	Frees the timer from memory
`prints()`	String printing function

Additionally supports all MicroC functions

Timer Functions

There are unique function calls related to creating, running, and clearing our timers.

Example



Testing Process

Overview

- Similar to microc test suite
- Allows labeling and instantiation of a customizable timer

Testing a Timer

```
timer_init(1);
timer1 = start_timer(<microseconds>);
timer_destroy(1);
```

Example

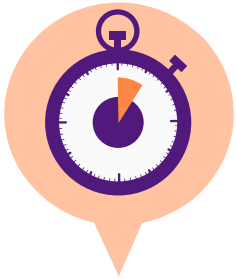
Example of a failed timer:

```
timer_init(1);
timer1 = start_timer(true);
timer_destroy(1);
```

Testing Currently...

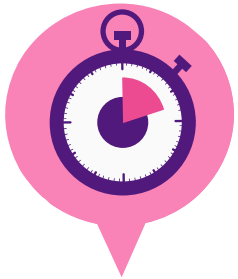
```
-n fail-return1...
OK
-n fail-return2...
OK
-n fail-string-assign1...
FAILED
 fail-string-assign1.err differs
-n fail-string-assign2...
FAILED
 fail-string-assign2.err differs
-n fail-string-assign3...
FAILED
 fail-string-assign3.err differs
-n fail-string-assign4...
FAILED
```

Takeaways/Lessons Learned



TIME

...Is everything! Start early!

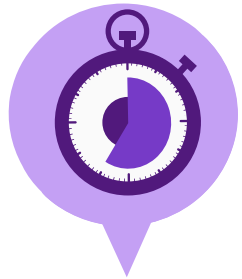


Incremental

Don't try to add too many features at once, start small and work up

Be honest with yourself

Don't try to accomplish the impossible.



Communicate

Be communicative with teammates about progress

Project Demo