What does this C code do?

```
int foo(char *s) {
  int L = 0;
  while (*s++) {
     ++L;
  }
  return L;
}
```

strlen

Exam review tonight 7pm 1404 Siebel

Bring your questions

Machine Language and Pointers

- Array Indexing vs. Pointers
  - Pointer arithmetic, in particular

Representing strings

- A C-style string is represented by an array of bytes.
  - Elements are one-byte ASCII codes for each character.
  - A 0 value marks the end of the array.

32	space	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	”	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	’	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	I	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	del

Null-terminated Strings

+0

- For example, “Harry Potter” can be stored as a 13-byte array.

72	97	114	114	121	32	80	111	116	116	101	114	0
H	a	r	r	y		P	o	t	t	e	r	\0

- Since strings can vary in length, we put a 0, or null, at the end of the string.
  - This is called a null-terminated string

- Computing string length
  - We’ll look at two ways.

p++

Array Indexing Implementation of strlen

$a0

```
int strlen(char *string) {
    int len = 0;
    while (string[len] != 0) {
        len ++;
    }
    return len;
}
```

$v0

```
strlen:   li    $v0, 0          # ~~t0~~ len
sl_loop:  add   $t0, $a0, $v0
          lb    $t1, 0($t0)
          beq   $t1, 0, sldone
          add   $v0, $v0, 1
          j     sl_loop
sldone:   jr    $ra
```

Pointers & Pointer Arithmetic

- Many programmers have a vague understanding of pointers
  - Looking at assembly code is useful for their comprehension.

```
int strlen(char *string) {
    int len = 0;
    while (string[len] != 0) {
        len ++;
    }
    return len;
}
```

```
int strlen(char *string) {
    int len = 0;
    while (*string != 0) {
        string ++;
        len ++;
    }
    return len;
}
```

What is a Pointer?

int *p, *q;

if (p == q)

- A pointer is an address.
- Two pointers that point to the same thing hold the same address
- Dereferencing a pointer means loading from the pointer’s address
- A pointer has a type; the type tells us what kind of load to do
  - Use load byte (lb) for char *
  - Use load half (lh) for short *
  - Use load word (lw) for int *
  - Use load single precision floating point (l.s) for float *
- Pointer arithmetic is often used with pointers to arrays
  - Incrementing a pointer (i.e., ++) makes it point to the next element
  - The amount added to the point depends on the type of pointer
    - pointer = pointer + sizeof(*pointer’s type)*
      - 1 for char *, 4 for int *, 4 for float *, 8 for double *

p+=4;

store load

*q = *p;

What is really going on here…

$a0

```
int strlen(char *string) {
    int len = 0;

    while (*string != 0) {
        string ++;
        len ++;
    }


    return len;
}
```

$v0

```
strlen:   ~~li $v0, 0~~
          move $t1, $a0
sl_loop:  lb $t0, 0($a0)
          beq $t0, 0, sl_done
          add $a0, $a0, 1
          ~~add $v0, $v0, 1~~
          j sl_loop

sl_done:  sub $v0, $a0, $t1
          jr $ra
```

Structs

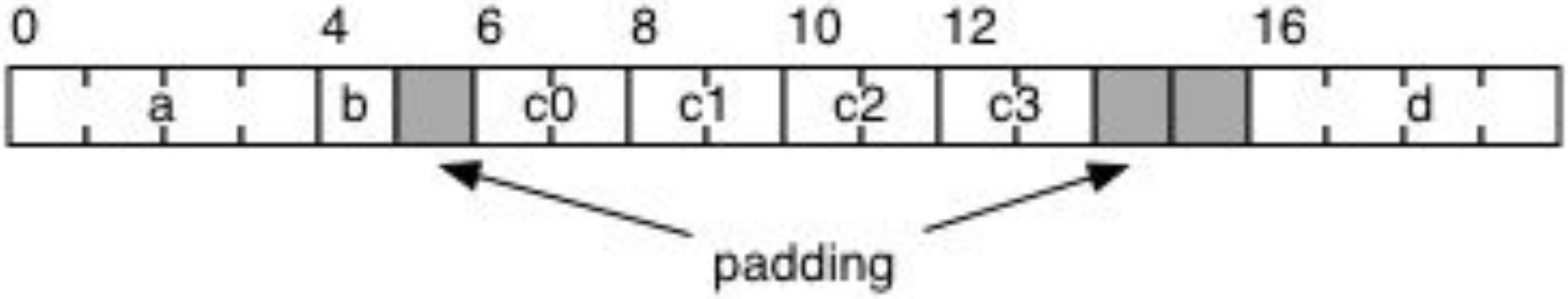
- Structs are like arrays, but the elements can be different types.
  - Same with objects

- Compiler/assembler inserts padding to “naturally align” data
  - Sometimes you can reorganize fields to eliminate padding.

C#   class

- Consider:

```
struct {
  int a;
  char b;
  half c[4];
  int d;
}
```



Summary

- Machine language is the binary representation of instructions:
  - The format in which the machine actually executes them
- MIPS machine language is designed to simplify processor implementation
  - Fixed length instructions
  - 3 instruction encodings: R-type, I-type, and J-type
  - Common operations fit in 1 instruction
    - Uncommon (e.g., long immediates) require more than one
- Pointers are just addresses!!
  - “Pointees” are locations in memory
- Pointer arithmetic updates the address held by the pointer
  - “string ++” points to the next element in an array
  - Pointers are typed so address is incremented by sizeof(pointee)