



wap_add: ↙

```
pushl %ebp
```

```
movl %esp, %ebp
```

```
pushl %ebx
```

```
movl 8(%ebp), %edx
```

```
movl 12(%ebp), %ecx
```

```
movl (%edx), %ebx
```

```
movl (%ecx), %eax
```

```
movl %eax, (%edx)
```

```
movl %ebx, (%ecx)
```

```
addl %ebx, %eax
```

```
popl %ebx
```

```
popl %ebp
```

```
ret
```

```
int isPalindrome (char * Strings, int len) {  
    if (len < 2) return 1;  
    return s[0] == s[len-1] &&  
        isPalindrome (&s[1], len-2);  
}
```

}

caller:

```
pushl %ebp
movl %esp, %ebp
subl $24, %esp

movl $534, -4(%ebp)
movl $1057, -8(%ebp)

leal -8(%ebp), %eax
movl %eax, 4(%esp)
leal -4(%ebp), %eax
movl %eax, (%esp)
```

~~0x80...~~ call swap_add

0x80... movl -4(%ebp), %edx
subl -8(%ebp), %edx
imull %edx, %eax

```
leave
ret
```

```
.text
.globl isPalindrome
.type isPalindrome, @function
```

```
#####
# Recursive function that returns true if the input is a palindrome
# Takes two parameters:
# @param string: a pointer to a char[]
# @param len: an integer that holds the length of the current string
# Returns whether or not the given string is a palindrome
# @return an integer. 1 means true, 0 means false
#
# This function recursively calls itself to determine if the string
# is a palindrome. On each "iteration" the function checks the two end-most
# characters to see if they match. If they do, then it calls itself with
# the address of the second character (&string[1]) and len-2.
# The base case for the recursion is the length of the string is 1 or 0.
# In this case, it IS a palindrome.
#####
```

taco cat

10

eb =>

20

0x8000 isPalindrome: ←

```
# Save the old stack and set up our stack.
pushl %ebp
movl %esp, %ebp
pushl %ebx
```

esp
ebp →

```
# Load the length of the string, and check if it's less than 2
movl 12(%ebp), %eax
cml $2, %eax
```

30

esp →

```
# If it's NOT less than two, skip the following lines
jge compareChars
```

0x5000

```
# If it's less than two, return 1
movl $1, %eax
jmp returnFromFunc
```

taco cat / 0

```
compareChars:
```

40

```
# Load the pointer to the string
movl 8(%ebp), %ebx

# Load the first and last character of the string
# Note: %eax contains the length of the string
movzbl (%ebx), %edx
movzbl -1(%ebx, %eax, 1), %ecx
```

```
# Compare the characters
cml %ecx, %edx
```

50

```
# If they match, set up the recursion
je recurse
```

```
# If they are NOT the same, return 0
xorl %eax, %eax
jmp returnFromFunc
```

```
recurse:
```

60

```
# Subtract 2 from the length and put on the stack
subl $2, %eax
pushl %eax
# Increment the base of the string by 1.
addl $1, %ebx
pushl %ebx
call isPalindrome
```

```
returnFromFunc:
```

0x5100
0x5100

```
leave
ret
```

