CS107 November 18th, 2026

Autumn 2026

CS107 Quiz 7 Solution

This is a closed book, closed note, closed computer quiz. You have 10 minutes to complete it.

SUNet ID (**`@stanford.edu`**): _____

First and Last Name: _____

I accept the letter and spirit of Stanford's Honor Code.

[signed] _____

Do not turn over until instructed to do so.

Problem 1: `assign5` Takeaways [4 points, 2 points each]

The function on the left compiles to the x86-64 on the right, warning of a missing return value. Recall that **`strcmp`** returns the ASCII difference of the first mismatched characters (0 if equal).

```
int stanford(const char *big) {
   char game[14];
   strcpy(game, big);
   if (strcmp(game, "axe") == 0)
      return 0;
}
```

```
stanford:
   sub    $0x18,%rsp
   mov    %rdi,%rsi
   lea    0x2(%rsp),%rdi
   call   strcpy
   mov    $0x402010,%esi # arg2 -> "axe"
   lea    0x2(%rsp),%rdi
   call   strcmp
   add    $0x18,%rsp
   ret
```

a. What does a call to **`stanford("endzone")`** return? Briefly explain why. Remember, whatever ends up in **`%rax`** is treated as the return value.

The call to **`strcmp`** leaves **`'e' - 'a' == 4`** in **`%rax`**, and that's what's left at the time the call to **`stanford("endzone")`** itself returns.

b. What's the longest the C string **`big`** can be—equivalently, the largest **`strlen(big)`** can be—without overwriting any part of the saved return address? Briefly explain.

21 characters! The full stack frame is **`0x18 = 24`** bytes. **`game`** starts at **`%rsp + 2`**, and the stack frame ends at **`%rsp + 24`** when **`strcpy`** is called. That's space for 22 characters (including the **`'\0'`**).

Problem 2: Implicit Allocator [6 points, 2 points each]

An implicit allocator gives each block an 8-byte header, which stores the payload size (always a multiple of 8) with the lowest bit set to 1 if the block is allocated (and left as 0 otherwise).

a. A block's header holds **`0x31`**. What is its payload size, and is it allocated or free?

LSB is clearly 1, so the block is allocated. Mask away the LSB and you're left with **`0x30`**, or 48, bytes.

b. Implement **`get_payload_size`**, which accepts the address of a block's header and returns that block's payload size, in bytes.

```
size_t get_payload_size(size_t *hdr) {

   return *hdr & ~0x7;
}
```

c. Implement **`get_next_header`**, which accepts the address of a block's header and returns the address of the next block's header. You may assume there is, in fact, a next block, and you should call your **`get_payload_size`** from part b.

```
size_t *get_next_header(size_t *hdr) {

   return (size_t *)((char *) hdr + 8 + get_payload_size(hdr));
}
```

alternatively, **`hdr + 1 + (get_payload_size(hdr) >> 3)`**