CS107 November 11th, 2026

Autumn 2026

CS107 Quiz 6 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: `assign4` Takeaways [6 points, 1 point per blank]

a. Each **`char *`** string in **`words`** begins with a punctuation mark, e.g. **`"#purr"`** and **`"!meow"`**. Fill in the blanks so **`qsort`** sorts the strings alphabetically, ignoring first characters. Answer on the placeholder lines. (**`qsort`**'s third argument is the element size.)

```
int cmp_tails(const void *one, const void *two) {

   const char *s = ____*(char **) one____;

   const char *t = ____*(char **) two____;
   return strcmp(s + 1, t + 1);
}
qsort(words, n, ____sizeof(char *)____, cmp_tails);
```

b. **`base`** points to an array of **`n`** elements, each **`width`** bytes, with room for one more. Fill in the blanks so that **`wedge`** inserts a copy of the element addressed by **`elem`** at index **`k`**, shifting later elements over by one. Note: **`memcpy`** and **`memmove`** take the same three parameters in the same order. Place your answers directly on the placeholder lines.

```
void wedge(void *base, size_t n, size_t width,
           size_t k, const void *elem) {

   char *slot = ____(char *) base + k * width____;

   memmove(____slot + width____, slot, ____(n - k) * width____);
   memcpy(slot, elem, width);
}
```

Problem 2: Code Generation and Reverse Engineering [4 points, 2 points per blank]

The function below has two lines of C replaced by blanks, alongside its complete **`gcc`**-generated x86-64 assembly. Recall that **`%rdi`** and **`%rsi`** hold the first two parameters, **`%rax`** holds the return value, and a parameter register left untouched before a **`call`** passes its value along as is. **`pushq`** saves a register to the stack, and **`popq`** restores it. **`game`** takes two **`long`**s and returns a **`long`**. Write your answers on the lines below the boxes, not the ones within.

C Code:

```
long big(long cardinal,
         long bears) {
  long axe = __________;  (i)
  return __________;      (ii)
}
```

x86:

```
big:
   pushq   %rbx
   movq    %rsi, %rbx
   addq    $7, %rdi
   call    game
   subq    %rbx, %rax
   popq    %rbx
   ret
```

i. **`game(cardinal + 7, bears)`**

ii. **`axe - bears`**

Alternatively, **`axe`** could have been backed by **`%rbx`**, in which case **`(i)`** and **`(ii)`** could have been **`bears`** and **`game(cardinal + 7, axe) - axe`**. It's even possible for **`axe`** to store the incremented first argument, so that **`(i)`** and **`(ii)`** are **`cardinal + 7`** and **`game(axe, bears) - bears`**.