*School of Software Engineering*

UNIVERSITYOF SCIENCE AND TECHNOLOGY OF CHINA

**Compilers, Spring 2024**

**Quiz 5: Memory Layout**

Name: Id:

1. In part A of lab 3, you have written a minimal *runtime system*. In order to store objects into the heap, you have filled in code for this function **Tiger\_new()**:

***// "new" an object, do necessary initializations before***

***// returning the pointer (reference).***

**void \*Tiger\_new (void \*vtable, int size){**

***// You should write 3 statements for this function.***

***// #1: "calloc" a chunk of memory of "size":***

***// #2: set up the "vtable" pointer properly:***

***// #3: return the pointer***

**}**

Essentially, the above function uses the C heap as the Java heap, and uses the C allocation facilities (i.e., **calloc()**) to allocate Java objects. Consider another algorithm to do this (with the key idea to manage the Java heap space explicitly):

***// a heap and its head pointer "head", which points to***

***// the next allocatable heap space.***

**char heap[1000000000]={'\0'};**

**char \*head=heap;**

***// "new" an object, do necessary initializations before***

***// returning the pointer (reference).***

**void \*Tiger\_new(void \*vtable, int size){**

***// You should write 4 statements for this function.***

***// #1: allocate a chunk of memory of size "size":***

**char \*p = head;**

**head += size;**

***// #2: clear this chunk of memory (zero off it):***

***// #3: set up the "vtable" pointer properly:***

**\*p = vtable;**

***// #4: return the pointer***

**return p;**

**}**

1. Which one of the two algorithms runs faster? Why?

2. Is the second algorithm better than the first one? Why or why not?