Instructions: This is a closed book, closed note exam. Calculators are not permitted. Please work the exam in dark pencil or pen and do not separate the pages of the exam. Note that this exam is double sided. If you run out of room, please mark on the problem that you will continue on the pages at the end of the exam. Also please identify the problem that is being continued, and draw a box around it. When writing your answer in a given answer box, please just write the answer and do not add extraneous marks etc. which confuse our grading tools.

Read each question over before you start to work.

If you have a question, raise your hand and we will come to you; do not leave your seat.

For maximum credit, show your work.

Good Luck!

Your Name (*please print*) _____

Your Student Number (9 digit) _____

This exam will be conducted according to the Georgia Tech Honor Code. I pledge to neither give nor receive unauthorized assistance on this exam and to abide by all provisions of the Honor Code.

Signed _____

1	2	3		
30	35	35		100

Problem 1 (2 parts, 30 points)   **Singly Linked Lists**

Consider a singly linked list that is implemented using the following `struct` definitions.

```
typedef struct llnode_t {
    int key;
    void* data;
    struct llnode_t* next;
}LLNode;

typedef struct {
    int size;
    LLNode* head;
} SLinkedList;
```

Part A (8 points) Assuming a **64-bit** system and proper memory alignment, consider the following function.

```
SLinkedList* create_slinkedlist(void) {
    SLinkedList* newList = (SLinkedList*)malloc(sizeof(SLinkedList));
    newList->head = NULL;
    newList->size = 0;
    return newList;
}
```

A.1 What integer is passed to `malloc` when this function executes? _____.

A.2 How much space is allocated for the <u>locals</u> in this function's activation frame? _____**bytes.**

Part B (22 points) On the next page, complete the C function `FindAndSplit` that takes a pointer to a list `SLL1` and an integer `key`. It should find the `LLNode` with the matching `key`, if it exists. It should then split the list at the found node, returning a new list from that node to the end. If the key is not found, it should return an empty list. *Do not allocate additional nodes in this function.* Specifically, our function should:

1. Create a new list `SLL2`. //already done
2. Find the `LLNode` with the given `key`. Let's call the node N. Keep track of the number of nodes before node N.
3. Split `SLL1` at node N: `SLL2` holds the nodes from N to the end of list; `SLL1` only holds nodes before N.
4. Update the size of `SLL1` and `SLL2`.
5. Return `SLL2`.

In the space below you might find it helpful to draw pictures of specific examples as you write the code.

```
SLinkedList * FindAndSplit(SLinkedList * SLL1, int key) {
    SLinkedList* SLL2 = create_slinkedlist();
```

```
}
```

Problem 2 (5 parts, 35 points)   **Heap Management**

Consider a memory allocator (malloc and free), such as described in class. Inside the C-code for the allocator, unsigned *heapPtr is the address of the next word that could be allocated in the heap, and unsigned **freePtr is the address of the first block (or object) on the free list (and the word at the address of each block on the free list is a pointer to the next block on the free list). The allocator uses a **best fit** strategy with a free list **sorted by increasing block address**, and never splits or coalesces blocks.

addr	value	addr	value	addr	value	addr	value	addr	value	addr	value
8000	4	8032	20	8064	4	8096	8048	8128	8	8160	45
8004	1234	8036	8016	8068	4	8100	8104	8132	8052	8164	0
8008	12	8040	4	8072	8088	8104	4	8136	4	8168	22
8012	8028	8044	12	8076	4	8108	2	8140	48	8172	7000
8016	4	8048	8	8080	8	8112	20	8144	4	8176	12
8020	8	8052	8072	8084	24	8116	8148	8148	0	8180	41
8024	12	8056	8	8088	8116	8120	4	8152	8	8184	40
8028	8052	8060	4	8092	16	8124	30	8156	0	8188	0

Suppose heapPtr = 8152 and freePtr = 8012. Consider each part below **independently**.

a) (5) How many **blocks and useable bytes** are on the free list? blocks = _____ bytes = _____

b) (5) What value would be returned by the call **malloc(13)** _____

Which (if any) values in the above map would be changed by the call in (b)?

addr _____ value _____ addr _____ value _____ addr _____ value _____ No change (✓) _____
(fill in the address/value pairs above. There may be more pairs than needed.)

Fill in the values at this point: heapPtr = _____ freePtr = _____

c) (5) What value would be returned by the call **malloc(25)** _____

Which (if any) values in the above map would be changed by this call?

addr _____ value _____ addr _____ value _____ addr _____ value _____ No change (✓) _____

Fill in the values at this point: heapPtr = _____ freePtr = _____

d) (5) Which (if any) values in the above map would be changed by the call **free(8080)**
addr _____ value _____ addr _____ value _____ addr _____ value _____ No change (✓) _____

Fill in the values at this point: heapPtr = _____ freePtr = _____

e) Consider the following code, which may be called as a helper function during allocation or freeing blocks to reduce external fragmentation. Assume `freePtr = 8012`.

L1	`void merge(){`
L2	`    unsigned **current = freeptr;`
L3	`    unsigned **next;`
L4	`    unsigned cs, ns, xs;`
L5	`    while (*current) {   // find 2 blocks that can be coalesced`
L6	`        cs = *(current - 1);`
L7	`        next = current + 1 + cs/4;`
L8	`        if (*current == next) break;`
L9	`        current = *current;` `    }`
L10	`    if (*current) {     // coalesce them`
L11	`        ns = *(next - 1);`
L12	`        xs = cs + ns + 4;`
L13	`        *(current - 1) = xs;`
L14	`        *current = *next;` `    }`
L15	`    return;` `}`

e.1) (6 points) What are the values of the following at line **L10**?

current		*current		next		cs	

e.2) (4 points) What are the values of the following at **L13**?

ns		xs	

e.3) (5 points) Which (if any) values in the above map would be changed by lines **L13-L14**?

addr _____ value _____ addr_____ value_____ addr_____ value_____ No change (✓)_____

Problem 3 (2 parts, 35 points)   **Hash Tables**

Part A (21 points) Consider an open hash table composed of a four-bucket table, with each bucket containing a variable length list. Each list entry has three slots <key, value, next> corresponding to the three word groupings in the entries section. The hash function is `key mod four`. Inserted entries are *appended to the end* of a bucket list. The initial state of the hash table is shown. List elements as allocated by malloc are shown in the figure. The symbol to the left of each list element (A, B, C,...) is the address returned by malloc. Entries that are freed are maintained in a last-in-first-out (LIFO) free list which starts out empty. *Be sure to reuse any blocks added to the free list before allocating unused blocks.*

Execute the access trace shown in the table below. For ease of representation, you may use the allocated blocks in any order. Show pointers both by their (symbolic) value, and with an arrow. If a value changes, cross it out and place the new value to the right. If a pointer changes, also place an x on the corresponding arrow, and draw the new arrow.

Buckets (Hash Anchor Table)
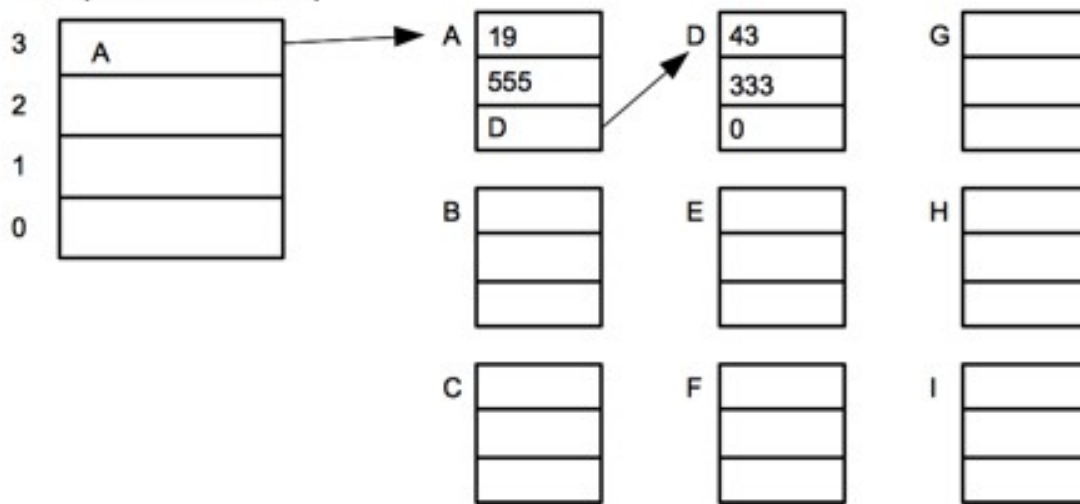
#	op	key	value	#	op	key	value
1	insert	20	777	4	delete	19	
2	insert	16	777	5	insert	32	222
3	insert	20	333				

Part B (14 points) Consider a *different* hash table that uses N **buckets**, each containing a singly linked list of L entries. That is the $N*L$ entries are uniformly distributed among the buckets. An application performs **1000** lookups of various keys: **600** of the lookups find the key in the hash table and **400** lookups fail to find the key and the average number of key comparisons is 7.3. A subsequent test performs **1000** lookups of various keys: **800** of the lookups find the key in the hash table and **200** lookups fail to find the key and the average number of key comparisons is 6.4 In both tests, the keys that are found are distributed throughout the buckets so that each position is equally likely to be where a key is found.

In this hash table the buckets are (bubble in your answer):   ○ sorted   ○ not sorted

What is the value of L ? _____

Extra work space:

If you are continuing a problem here, please identify the problem that is being continued.

Extra work space:

If you are continuing a problem here, please identify the problem that is being continued.