Your Name (please print clearly) _____

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	4	total
25	20	25	30	100

Instructions: This is a closed book, closed note exam. Calculators are not permitted.

If you have a question, raise your hand; do not leave your seat.

Please work the exam in pencil and do not separate the pages of the exam.

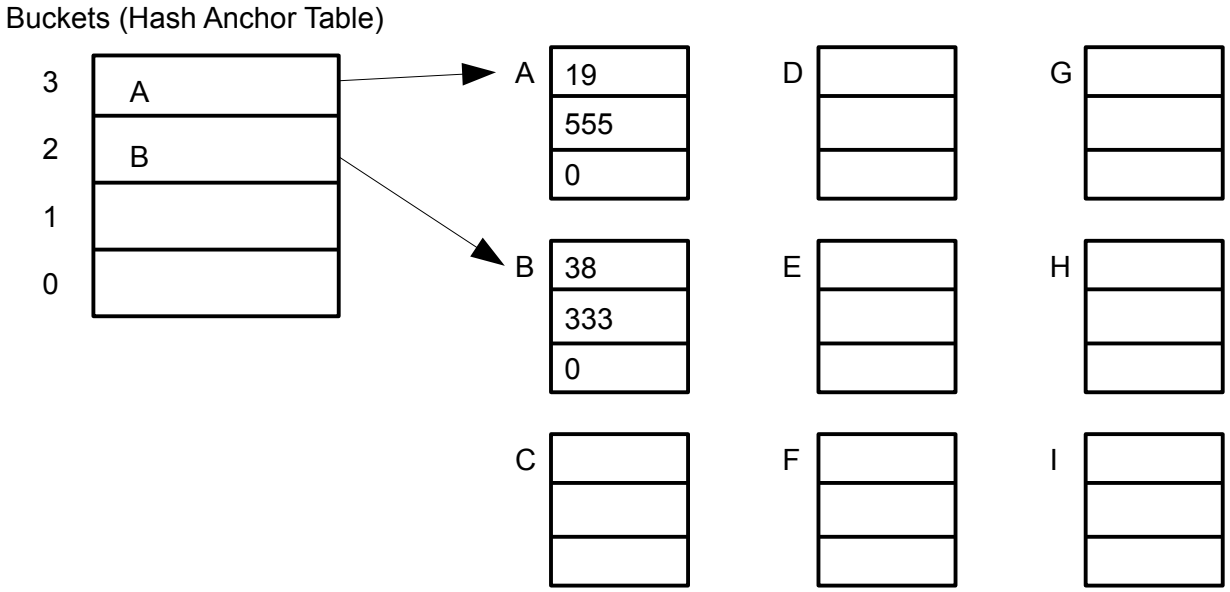
For maximum credit, show your work.

Good Luck!

Problem 1 (25 points) **Hash Tables**

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. Assume the free list is initially empty.

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.



Hash Table Access Trace

#	op	key	value	#	op	key	value
1	insert	20	111	5	remove	19	n/a
2	insert	35	222	6	insert	42	777
3	insert	63	444	7	insert	63	888
4	remove	20	n/a	8	insert	21	999

Problem 2 (3 parts, 20 points) **Associative Set Performance**

Part A (8 points) Suppose we have an associative set of **35** (key, value) pairs implemented as an **unsorted doubly linked list**. An application performs **600** lookups of various keys: **480** of the lookups find the key in the list and **120** lookups fail to find the key. The keys that are found are distributed throughout the list so that each position is equally likely to be where a key is found.

What is the average number of key comparisons that would be needed for a lookup in this list implementation? (Show work. Note: you may not have to use all data provided.)

number of comparisons: _____

Part B (6 points) Suppose the associative set is reimplemented as an **open hash table**. The same **35** (key, value) pairs are stored in the hash table and are evenly distributed across **5** buckets, each implemented as a **sorted doubly linked list**. An application performs the same **600** lookups in which **480** find the key being searched for and **120** do not. The keys that are found are distributed throughout the bucket lists so that each bucket and each position in the bucket lists is equally likely to be where a key is found.

What is the average number of key comparisons that would be needed for a lookup in this hash table implementation? (Show work. Note: you may not have to use all data provided.)

number of comparisons: _____

Part C (6 points) Suppose the hash table of Part B is resized so that the same **35** entries are evenly distributed across **7** buckets. However, the buckets are implemented as ***unsorted*** **doubly linked lists**. As before, the application performs the same **600** lookups in which **480** find the key being searched for and **120** do not. The keys that are found are distributed throughout the bucket lists so that each bucket and each position in the bucket lists is equally likely to be where a key is found.

What is the average number of key comparisons that would be needed for a lookup in this hash table implementation? (Show work. Note: you may not have to use all data provided.)

number of comparisons: _____

Problem 3 (5 parts, 25 points) **Heap Management**

Below is a snapshot of heap storage. The heap has been allocated contiguously beginning at 6000, with no gaps between objects. Heap management maintains a sorted free list to implement a **best-fit** strategy for reusing freed objects. The heap pointer is **6152** and the free pointer is **6072**.

addr	value	addr	value	addr	value	addr	value	addr	value	addr	value
6000	16	6032	12	6064	6020	6096	8	6128	0	6160	0
6004	6024	6036	80	6068	4	6100	6052	6132	4	6164	0
6008	4	6040	330	6072	6100	6104	16	6136	6032	6168	0
6012	16	6044	12	6076	16	6108	12	6140	0	6172	0
6016	36	6048	16	6080	128	6112	152	6144	24	6176	0
6020	8	6052	6128	6084	2604	6116	8	6148	0	6180	0
6024	3080	6056	100	6088	4	6120	16	6152	0	6184	0
6028	1048	6060	4	6092	6080	6124	24	6156	0	6188	0

Part A (4 points) Assuming the heap has been allocated contiguously beginning at 6000, with no gaps between objects, circle all object size words in the map above.

Part B (4 points) List the address of the first data word in each object in the free list in order, starting with the head of the free list.

Addresses of Objects on Free List (in order): _____

Part C (4 points) Based on the free list created in part B, if the following statement were executed, what would be the value of the variable `x`?

```
char* x = (char*)malloc(strlen("GB") + 1);
```

How many bytes of slack (if any) will result? (Hint: `strlen` computes the number of characters in a string up to but not including the terminating null character.)

x _____ **Slack:** _____

Part D (6 points) Suppose the next instruction is **`free(ptr)`**, where **`ptr = 6036`**. List all the changes to memory that will result:

Address	New Value

Part E (7 points) Based on the free list after part D, if an object of size **19 bytes** is allocated, what address will be returned? How many bytes of slack (if any) will result?

Address: _____ **Slack:** _____

List all the changes to memory that will result:

Address	New Value

Problem 4 (6 parts, 30 points) **Dynamic Data Structures: Linked List**

Consider a singly linked list consisting of student records defined by the following struct definition:

```
typedef struct STUDENT
{
  struct STUDENT* next; // Next pointer for linked list
  char* fname;
  char* mname;
  char* lname;
  double average;
  char  letterGrade;
} Student_t;

Student_t* LookupStudent_Sorted(Student_t* h, char* lastname)
{
  int cmp;

  while (_____________________________________)                    /* part A*/
    {

      cmp = ____________________________________________________; /* part B*/


      if (_____________________________________)                   /* part C*/
      {
        ____________________________________________________; /* part C*/
      }
      else if (_____________________________________)              /* part D*/
      {
        ____________________________________________________; /* part D*/
      }
      else

      ____________________________________________________;        /* part E*/
    }
 ____________________________________________________;       /* part F*/
}
```

Assume that the list of student records is sorted by increasing `lname` (i.e., a student record earlier in the list has a `lname` lexicographically less than the `lname` of student records later in the list). The function `LookupStudent_Sorted` takes a pointer `h` to a record at the head of a list of student records and a string `lastname`. Complete this function (following the steps below) so that it searches for a record whose `lname` field matches `lastname`. If such a record is found,return it; otherwise return `NULL`.

Part A (4 points) Continue the while loop if the end of list has not yet been reached.

Part B (5 points) Set the variable `cmp` to the result of comparing `lastname` to the `lname` of the current record `h`. Use `strcmp`, which takes two strings `s1` and `s2` and returns an integer `i` indicating how they are related (`i=0` if `s1 == s2`, `i<0` if `s1<s2`, and `i>0` if `s1>s2`).

Part C (6 points) If the current student record's `lname` matches `lastname`, return the current record.

Part D (6 points) Check whether the search can be ended early and if so, return the appropriate result.

Part E (5 points) Continue searching by moving `h` to the next record in the list.

Part F (4 points) Fill in what to do if the end of the list is reached and we fall out of the loop.