Problem 1 (5 parts, 35 points)	**Linked Lists**

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

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

typedef struct dll_t {
    struct llnode_t* head;
    struct llnode_t* tail;
    struct llnode_t* current;
    int              size
} DLinkedList;
```

Complete the following function by filling in blanks. The function should remove the node the current pointer is pointing to from the list DLL and return the data associated with the node removed. It should move the current pointer to the previous node. If the current node is the head or the only node in the list, the current pointer should be set to `NULL`. The list head and tail pointers should be updated appropriately.

Fill in each blank with the condition under which each statement should execute for `removeBackward` to function correctly. If the statement should always be executed, put a 1 in the blank.

```
void* removeBackward(DLinkedList* DLL){
  if(DLL->current == NULL) return NULL;
  LLNode* removeNode = DLL->current;
  void*   removeData = removeNode->data;
  if (__removeNode != DLL->tail_____________________________){
    (removeNode->next)->previous = removeNode->previous;
  }
  if (__removeNode == DLL->tail_____________________________){
    DLL->tail = removeNode->previous;
  }
  if (__removeNode != DLL->head_____________________________){
    (removeNode->previous)->next = removeNode->next;
  }
  if (__removeNode == DLL->head_____________________________){
    DLL->head = removeNode->next;
  }
  if(__1____________________________________________________){
    DLL->current = removeNode->previous;
  }
  free(removeNode); // Only delete node, not data
  DLL->size--;
  return removeData;
}
```

Problem 2 (2 parts, 30 points) **Doubly Linked Lists**

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

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

typedef struct dll_t {
    struct llnode_t* head;
    struct llnode_t* tail;
    struct llnode_t* current;
    int              size
} DLinkedList;
```

Part A (8 points) Assume a **32-bit** system and consider the following create_dlinkedlist

```
DLinkedList* create_dlinkedlist(void) {
    DLinkedList* newList = (DLinkedList*)malloc(sizeof(DLinkedList));
    newList->head = NULL;
    newList->tail = NULL;
    newList->current = NULL;
    newList->size = 0;
    return newList;
}
```

A.1 What integer is passed to `malloc` when this function executes? **16**.

A.2 How much space is allocated for the locals in this function's activation frame? **4** **bytes.**

Part B (22 points) Complete the C function `Reverse` that takes a pointer to a doubly linked list `DLL` (which might be empty) and reverses the order of the nodes. Each node's previous node becomes its next node and vice versa. The head and tail of the list should be updated as well. The current pointer should not change. *Do not allocate additional nodes in this function.* Specifically, our function should:

1. Run down the list, swapping `next` and `previous` of each element. (Hint: use local variables to temporarily hold pointers to the `next/previous` nodes so that it is easier to swap them.)
2. Swap head and tail. (Hint: use local variables to hold pointers to `head/tail` nodes.)

```
void Reverse (DLinkedList *DLL) {
   LLNode *c = DLL->head;
   LLNode *h = DLL->head;
   LLNode *t = DLL->tail;
   LLNode *n;
  LLNode *p;
  while (c) {
    n = c->next;
    p = c->previous;
    c->previous = n;
    c->next = p;
    C = n;
  }
  DLL->head = t;
  DLL->tail = h;
  return;
}
```

Problem 3 (4 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 **first fit** strategy with a free list **sorted by increasing block size**, and never splits blocks.

addr	value	addr	value	addr	value	addr	value	addr	value	addr	value
8000	16	**8032**	12	**8064**	4	**8096**	8048	**8128**	4	**8160**	0
8004	0	**8036**	0	**8068**	12	**8100**	8104	**8132**	8004	**8164**	0
8008	4	**8040**	8072	**8072**	8088	**8104**	4	**8136**	8	**8168**	22
8012	16	**8044**	12	**8076**	8144	**8108**	2	**8140**	8	**8172**	7000
8016	8072	**8048**	4	**8080**	8	**8112**	12	**8144**	43	**8176**	12
8020	8	**8052**	8116	**8084**	16	**8116**	8004	**8148**	28	**8180**	41
8024	8132	**8056**	8	**8088**	0	**8120**	4	**8152**	8	**8184**	40
8028	116	**8060**	8124	**8092**	16	**8124**	30	**8156**	0	**8188**	0

Suppose `heapPtr` = 8148 and `freePtr` = 8052. Consider each part below **independently**.

a) (5) How many **blocks and useable bytes** are on the free list? blocks = **3** bytes = **32**

b) (10) What value would be returned by the call **`malloc(6);`** **8116**

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

addr **8052** value **8004** 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` = **8148** `freePtr`= **8052**

c) (10) What value would be returned by the call **`malloc(23);`** **8152**

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

addr **8148** value **24** addr_____ value_____ addr_____ value_____ No change (✓) _____

Fill in the values at this point: `heapPtr` = **8176** `freePtr`= **8052**

d) (10) Which (if any) values in the above map would be changed by the call **`free(8140)`**?

addr **8052** value **8140** addr **8140** value **8116** addr_____ value_____ No change (✓)_____

Fill in the values at this point: `heapPtr` = **8148** `freePtr`= **8052**