SCC 120 Introduction to Data Structures

Workshop Three: Linked Lists

1. Consider the C function `malloc(K)`, where K is a positive integer.
   What does `malloc` do?

   a. Allocates *K* bytes of memory in the current stack frame.
   b. Releases *K* bytes of memory to the free space pool.
   c. Attempts to allocate *K* bytes of memory in the heap and, if successful, returns the memory address of the first byte.
   d. Allocates space for a variable at the memory address *K*.

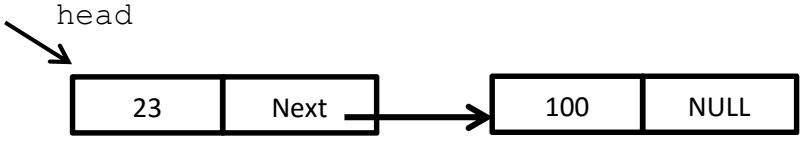
2. Assuming an `int` type occupies 4 bytes and `a char` type occupies 1 byte, what will be printed out by the following `printf` function?

```
int x; int y[10];
char z;
printf("%d %d %d\n", sizeof(x), sizeof(y), sizeof(z));
```

3. Consider a linked list that is made up of nodes of type

```
typedef struct{
   int val;
   struct node* next;
}node;
```

The list has two nodes. You are given a pointer "`head`" of type `node`, which points to the first node of the list; and that the `next` field of the last node is NULL. The list is described in the following diagram:



3.1 Which of the following code statements will remove the second node from the list?

```
a) head->next->val=0;
b) head=NULL;
c) head->next = head->next->next;
d) head->val=0;
```

3.2 Write a few lines of code to add a new node with a value of 17 after the second node of the list. Here we assume initially the list contains only two nodes as depicted in the diagram.

```
void insert(node* head) {
//head points to the first cell of the chain









}
```

3.3 Complete the following function `count()` that takes a linked list as input, and prints out the number of nodes in the linked list. Here we assume the list has at least two nodes

```
int count(node* head) {
//head points to the first cell of the chain









}
```

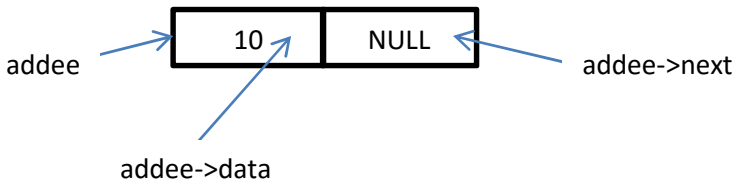
4. Consider the following data structure:

```
typedef  struct _chainCell{
        int data;
        struct _chainCell* next;
} chainCell;
```

The following piece of code creates a chainCell item, `addee`

```
chainCell *addee = malloc(sizeof(chainCell));
addee -> data = 10;
addee -> next = NULL;
```

This creates a `chainCell` item looks like:



The following function, `addInOrderk`, takes in two parameters. The first parameter, `header` points to the first node of the list. The second parameter, `data`, is an integer value. This function creates a new `chainCell` item to store `data`, and inserts the new `chainCell` item, `addee`, to the list. **Pay attention to where of the chain the new item is inserted to.**

```
chainCell* addInOrder(chainCell *header, int data) {
        chainCell *addee= malloc(sizeof(chainCell));
        addee -> data = data;
        addee -> next = NULL;

        if (header == NULL) { //  special case one : adding to an empty chain
            header = addee;
            return header;
        }
         if (data  <  header->data){ // special case two : data to be added
                                     // is added at start of chain
            addee->next = header;
            header = addee;
            return header;
        }

         chainCell *pt = header, *previous = NULL;

         // general case – data added in somewhere of the chain
        while ((pt != NULL) && (data > pt->data)){
            previous = pt;
            pt = pt->next;
        }
        previous->next = addee;
        addee->next = pt;
        return header;
}
```

```
chainCell* header = NULL; //null at this point

//incrementally assemble a list

header = addInOrder(header, 10);

header = addInOrder(header, 5);

header = addInOrder(header, 8);

header = addInOrder(header, 20);

header = addInOrder(header, 6);
```

The above code creates a list with 5 nodes using the `addInOrder`() function. Here `header` points to the first node of the chain. Write down the value of the `data` field for each node of the chain: