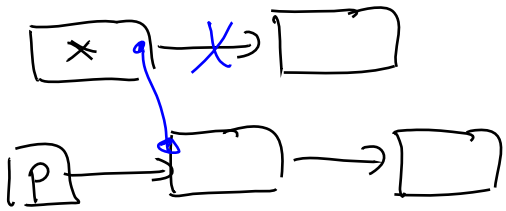
P1) You are given the node class declaration below. The methods are implemented in the same way as in the lecture if not stated otherwise.

```
template <class T>
class Node
{
    private:
            // next is the address of the following node
        Node<T> *next;
    public:
      // the data is public
        T data;

            // constructor
        Node (const T& item, Node<T>* ptrnext = NULL);

        // list modification methods
        void InsertAfter(Node<T> *p);
        Node<T>* AppendAfter(Node<T> *p);
        Node<T> *DeleteAfter(void);

        // obtain the address of the next node
        Node<T> *NextNode(void) const;
};
```

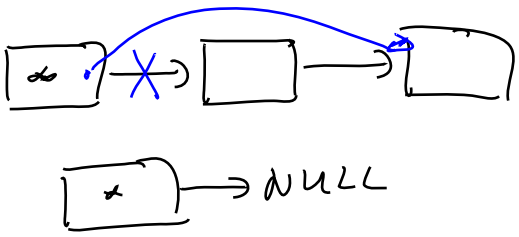
a. Implement the following method that appends a list whose head is pointed to by `p` to the current node and returns the address of the previous next node.

```
Node<T>* Node::AppendAfter(Node<T> *p)
{
    Node<T> * tmpPtr = next;
    next = p;
    return tmpPtr;
}
```



b. Implement the following method that deletes the node following the current node. Do not dellocate the memory of the deleted node, just return its address.

```
Node <T> * Node::DeleteAfter(void){
    Node<T> * tmpPtr = next;
    if(next == NULL)
        return NULL;
    next = tmpPtr -> next;
    return tmpPtr;
}
```



You are given a linked list that consists of integers and a function declared as

```
void Arrange2(Node <T>* &head);
```

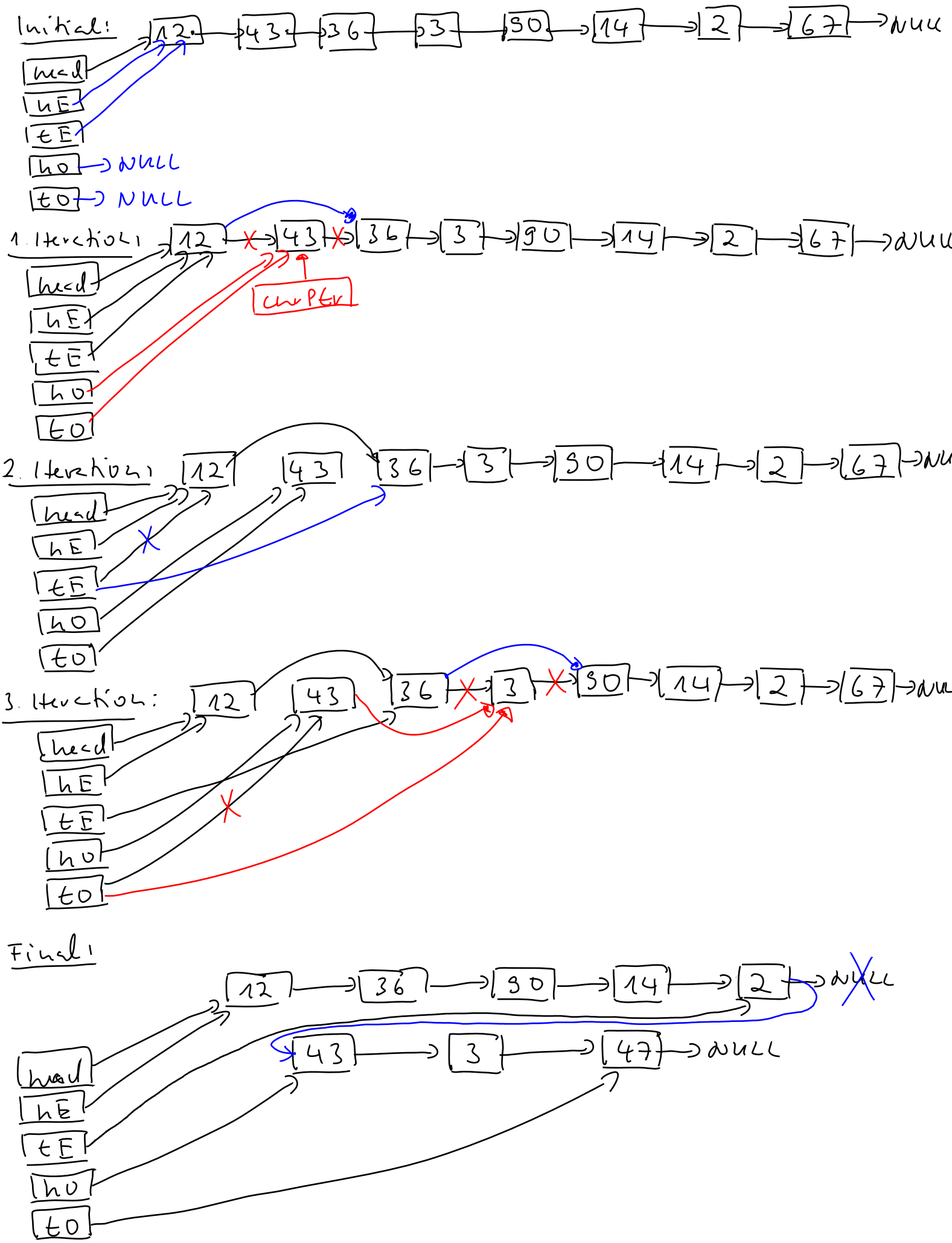
This function rearranges the nodes of a linked list pointed to by `head` such that the nodes with even integer content are at the beginning of the list, followed by the nodes with odd integer content. The order of occurrence among the group of nodes that give the same remainder should be maintained as in the original list. In this question, you can assume that all non-empty lists start with an even number.

Example: Original linked list data: 12, 43, 36, 3, 90, 14, 2, 67

After calling Arrange2: 12, 36, 90, 14, 2, 43, 3, 67

Implement `Arrange2` by filling in the blanks and boxes of the function below according to the given comments and without allocating additional memory for nodes.

```
template <class T>
void Arrange2 (Node <T>* &head)
{
      // return if list is empty
         if( head == NULL)
               return;
      // Declare the necessary variables
      Node<T)* hE=NULL; * tE=NULL; * hO=NULL; tO=NULL;
      Node<T>* curPtr;


      // Separate the nodes into two lists (even and odd)
      hE = head;
      tE = head;
      while(!tE->NextNode() == NULL){
            if( tE->NextNode()->data %2 == 1){
                  curPtr = tE->DeleteAfter();
                  if(hO == NULL){
                        hO = curPtr;
                        tO = curPtr; }
                  else{
                        tO->InsertAfter(curPtr);
                        tO = tO->NextNode(); }
                  tO->AppendAfter(NULL); }
            else tE = tE->NextNode(); }
      // Combine the two lists
      head = hE;
      tE->AppendAfter(hO);

}
```


Initial:
12
43
36
3
90
14
2
67
NULL
head
hE
tE
hO → NULL
tO → NULL
1. Iteration:
curPtr
2. Iteration:
3. Iteration:
Final:
47