CS101 Homework 1

name: ID number:

(2pt+2pt) Problem 1: Reverse a linked list.

Reverse a linked list. Note this linked list is **different** from that in your HW0-Linked list. It does not have a dummy node. e.g.

Before reversion: head is 1

```
1 -> 2 -> 3 -> 4 -> NULL
```

After reversion: head is 4

```
4 -> 3 -> 2 -> 1 -> NULL
```

You are required to use **2** different methods to implement this.

- Iterative: use a loop to implement.
- Recursive: as its name indicates.

Note that the above two methods have different thoughts, if you use a tail recursion but the thought is similar to that of the Iterative method, then it will not be counted as a Recursive method and thus you will not get full points. Actually, tail recursion can be optimized to a loop form so that they are the same indeed.

The answer area is on the next page. You may not use up all the blanks.

Answer:

```
struct node {
   int key;
   node *next;
};
typedef node* linkedlist;
```

Iterative:

```
linkedlist iterative(linkedlist head) {
   _____________________________________;
   _____________________________________;
   _____________________________________;
   _____________________________________;
   _____________________________________;
   _____________________________________;
   _____________________________________;
   _____________________________________;
   _____________________________________;
}
```

Recursive:

```
linkedlist recursive(linkedlist head) {
   _____________________________________;
   _____________________________________;
   _____________________________________;
   _____________________________________;
   _____________________________________;
   _____________________________________;
   _____________________________________;
}
```

(1pt+1pt+2pt) Problem 2: Doubly linked list

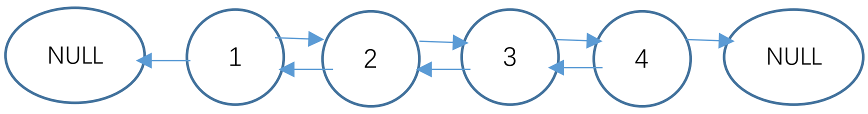
2.1 Given a node, what is the time complexity of **Insert Before** in Doubly linked list? Can the same operation have the same time complexity in Singly linked list? If the answer is yes, explain or draw a picture to show how to realize it with the given example:

I want to insert a node with value 3 before the node with value 4 in the following Singly linked list.

```
1 -> 2 -> 4 -> 5 -> NULL
```

Answer:

2.2 We have a doubly linked list where each node in between has a previous and a next pointer. The `head` of the list node is defined as the node with a `NULL` prev pointer, and the tail of the list is defined as the node with a `NULL` next pointer. e.g. node:1 is `head` and node:4 is `tail`



The class `doubly_ll` has private member `head` and `tail`.

The struct dll_node is given:

```
struct dll_node {
   int val;
   dll_node *prev;
   dll_node *next;
}
```

Now, given a node in the doubly linked list, I want to delete this node, i.e. free its allocated memory, this node is not `NULL`. Fill in the blanks to implement this function.

Answer:

```
void delete_node(struct dll_node *node) {
   if (node == NULL) {
      return;
   }

   //if the node is head
   if(___________________) {
      head = ________________;
      _______________________;
      _______________________;
      return;

   // if the node is tail, similar with head, omit it.
   ... ... ... ...

   // if the node is in between
   ___________________________________;
   ___________________________________;
   ___________________________________;
   return;
}
```

2.3 The same doubly linked list as described in part (b). Given a random node in this list, frees all the reachable nodes from that given node (free the entire doubly linked list). Note that this node may be `head`, or `tail`, or a node in between head and tail. You may not use up all the blanks. Hint: you may want to use Recursive.

Answer:

```
void free_doubly_ll (struct dll_node *node) {
   if (node != NULL) {
      if (node->prev != NULL) {
         ___________________________________;
         ___________________________________;
      }
      if (node->next != NULL) {
         ___________________________________;
         ___________________________________;
      }
      ___________________________________;
      ___________________________________;
   }
}
```

(1pt+1pt+2pt) Problem 3:

In this problem, you will learn how to use linked list to represent a polynomial and make some basic operations on them. A polynomial is an expression which has the form of:

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

Normally, for an order-n polynomial, a size-n array $(a_n,\ a_{n-1}, \cdots, a_0)$ can uniquely determine it. However, when the coefficients are sparse, it will waste a lot of memory. E.g., the polynomial $x^{2018} + 1$ needs a 2019-dimensional array but 2017 zeros in it! That is the motivation to use linked list.

3.1 If we use a linked list to represent a polynomial expression, each node should contain both the coefficient and the index. The structure is defined in the code below. Try to draw the linked list that represents $x^{2018} + 3x^{101} - 7$ (with descending order).

```
typedef struct PolyNode {
   int coef // the coefficient
   int idx  // the index
   struct PolyNode* next
} PolyNode;
```

Answer:

3.2 Now we can use linked list to represent any polynomials we want. Moreover, we also want to do some mathematical operations on them. Try to complete the code following to implement the addition of two polynomials. For convenience, the result will be hold in the first polynomial (i.e. let A = A + B). Your code should maintain the descending order (e.g. the addition of $x^3 + 2x^2 + 9$ and $x^2 + x - 7$ is $x^3 + 3x^2 + x + 2$).

```
Answer:
void insert(PolyNode* A, int coef, int idx)
{ … } // You can assume the insert method is well-defined: a new node
with (idx, coef) will be inserted to the proper place.


void add(PolyNode* A, PolyNode* B) {
   PolyNode* p = B;
   while(p != NULL) {





      p = p->next;
   }
}
```

3.3 Similar to the operation of addition, complete the code of multiplication below. The result will be stored in a new polynomial list. Your code should maintain the descending order as well. Suppose we have redefined `PolyNode` in class and provided a constructor function for `PolyNode`: `PolyNode(coef, idx)`.

```
Answer:

void multiply(PolyNode* A, PolyNode* B) {

   PolyNode* C = new PolyNode(); // Initialize a PolyNode as head for C

   PolyNode* p = B;

   while(p != NULL) {




      p = p->next;

   }

   return C;

}
```

(0.5pt+1.5pt+2pt) Problem 4:

For those arrays whose entries only contain 0 or 1, using a normal array may waste too much space. Notice that any number in computer is stored as binary bits, we then can use them to represent an array. E.g. number 1101001_2 can represents an array 1, 0, 0, 1, 0, 1, 1. (Pay attention to the order)

4.1 There is some subtle thing should be handled. In the computer, the number is of fixed size (e.g. 32 bits for integer), so that the array (the number ***a***) we store has many leading 0s which we do not want when we access the entries. Thus we need a number ***t*** to indicate where the tail is. The method is that we let ***t*** be the number of entries in the array. If $a = 1101001_2$, then t = __________. Your answer should be written in Decimal.

4.2 We then consider some basic operations on the bit array. First of all, an array should support random accesses. Please try to use a simple expression to get the *k*th ($0 \leq k < t$) entries in the array *a*. (Hint: consider the shift and logic operations on digitals given below)

`Answer:`

4.3 At last we consider an operation of insertion. Try to use several expressions to insert a 0 at *k*th ($0 \leq k < t$) index in the array *a*. (Hint: first consider how to separate a bit array into two parts)

`Answer:`

Table The bit operation you can use in Problem 4:

AND	1101_2 AND 1001_2 = 1001_2
OR	1101_2 OR 1001_2 = 1101_2
XOR	1101_2 XOR 1001_2 = 0100_2
NOT	NOT 1101_2 = 0010_2
SHIFT	1101_2 SHIFT 2 = 110100_2

(1pt+3pt) Problem 5:

In this problem, we mainly focus on how to represent data with two dimensional. These will be a first expansion of arrays and linked list.

Normally, a two-dimensional data can be represented as an n by n matrix $\boldsymbol{A}$ mathematically:

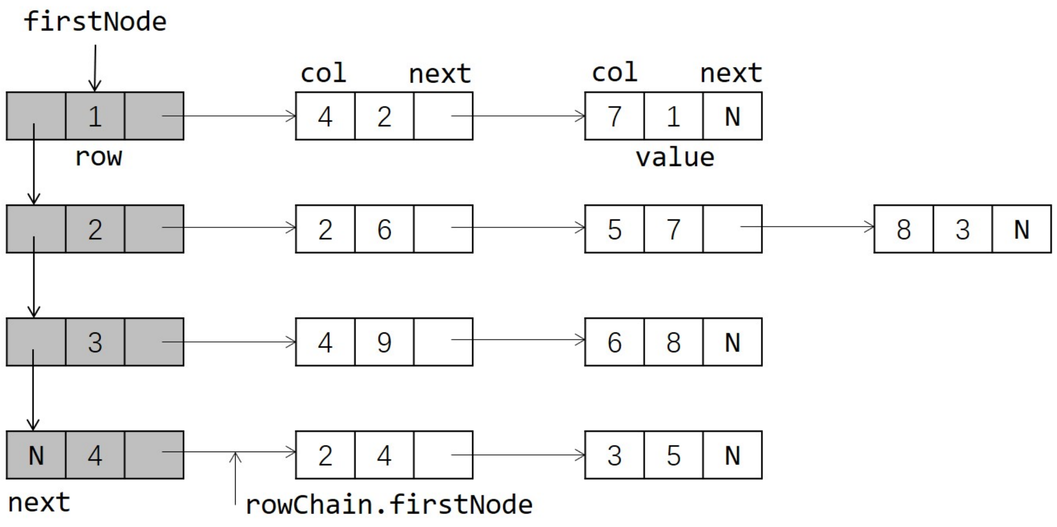
$$A = \begin{pmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{n1} & \cdots & A_{nn} \end{pmatrix}$$

5.1 Firstly we try to use an array to represent such a matrix. A trivial way is to put values in the array one by one in the one-dimension array $\boldsymbol{a}$. Suppose we put values row by row, then which index should we access in $\boldsymbol{a}$ when we want A_{ij}, $(0 < i < n+1, 0 < j < n+1)$? What if we put values column by column?

Answer:

5.2 When the matrix is sparse (i.e. with many 0s in it), using array will waste a lot of memory. Thus we can use linked lists to represent a sparse matrix as show in the figure below (call gray node headerNode and white node matrixNode). Try to complete the code which computes a transpose of a matrix, the definition of transpose is below:

$$A^T = \begin{pmatrix} A_{11} & \cdots & A_{n1} \\ \vdots & \ddots & \vdots \\ A_{1n} & \cdots & A_{nn} \end{pmatrix}$$



Answer:

```
// This function insert a node in a matrix

void insert(Matrix* A, int row, int col, int value) {

{ … } // You can assume that this function is well-defined


// This function compute the transpose of A and

// return the value in B

void transpose(Matrix* A, Matrix* B) {

   B.clear();              // Clear the origin data in B

   headerNode* h = A->firstNode;

   while (h != NULL) {

   }
}
```