Exercise 1:

Use the following declarations to write the file "`ll.c`":

```
struct node{
  char data;
  struct node* next;
};
```

```
struct linked_list{
  struct node* head;
};
```

1. Write a function that adds a new node to the end of a list. The function takes a pointer to the list and the new value. Your function should print feedback messages upon successful/ unsuccessful additions.
   `void add_node(struct linked_list* list, char val)`
2. Write a function that concatenates two linked lists of chars. The function takes pointers to both lists as arguments and concatenates the second list to the first list.
   `void concat(struct linked_list* list1, struct linked_list* list2)`
3. Write a function that prints the content of a list. The function takes a pointer to the list.
   `void print_list(struct linked_list* list)`

Exercise 2:

1. Write a program "`test.c`" that defines two linked lists of char and fills them with the following values:
   a. The first list will have 3 nodes with the following characters: A,B, and C.
   b. The second list will have 2 nodes with the following characters: D and E.
2. In your main also:
   a. Print the contents of list1 and list2
   b. Call the function concat and send your first and second lists as arguments.
   c. Print the content of the concatenated list.
   d. Call the function addNode and send your concatenated list and the value 'F' as arguments.
   e. Print the content of the concatenated list after adding the node.

Sample run:

```
The node (A) was added successfully
The node (B) was added successfully
The node (C) was added successfully
The node (D) was added successfully
The node (E) was added successfully
(A)->(B)->(C)-|
(D)->(E)-|
(A)->(B)->(C)->(D)->(E)-|
The node (F) was added successfully
(A)->(B)->(C)->(D)->(E)->(F)-|
```

Assignment:

1. Write a program "assignment.c" that reads a word from the user and uses a stack that is implemented as a linked list to print the word reversed.
2. You will need the following functions in your program:
   `void push(char info);`   `char pop();`   `int isEmpty();`
3. You will also need to declare:
   a. A global structure (struct stack) to represent the stack
   b. A global pointer of type struct stack

Hmirror , vmirror:

Add to `mybmp.h/mybmp.c` a function `hmirror` that flips the image horizontally. Call your implemented function in `test.c to` test it.

```
void hmirror(Pixel** pixels, int rows, int cols){
  int i, j;
  Pixel tmp;
  for (i=0; i < rows; i++)
    for (j = 0; j < cols/2; j++){
      tmp = pixels[i][j];
      pixels[i][j] = pixels[i][cols-j-1];
      pixels[i][cols-j-1] = tmp;
    }
}
```

Add to `mybmp.h/mybmp.c` a function `vmirror` that flips the image vertically. Call your implemented function in `test.c to` test it.

```
void vmirror(Pixel** pixels, int rows, int cols){
  int i;
  Pixel* tmp = (Pixel*)malloc(sizeof(Pixel) * cols);
  for (i=0; i < rows/2; i++){
      memcpy(tmp, pixels[i], sizeof(Pixel) * cols);
      memcpy(pixels[i], pixels[rows-i-1], sizeof(Pixel) * cols);
      memcpy(pixels[rows-i-1], tmp, sizeof(Pixel) * cols);
  }
}
```