Conclusión Ximena Isaac H:

En un inicio cuando se nos entregó el trabajo revisé el archivo y comparé los contenidos que revisamos en clase para encontrar alguna similitud. Logré darme cuenta que debíamos implementar cosas parecidas a las del trabajo en clase **mi_app** utilizando: *list_contains*, *list_print*, *list_append*, entre otros.

```
int list_contains(List *l, int value_to_find)
{
    Node *current = l ->head;
    while(current){
        if(current->value == value_to_find)
            return 1;
        current = current->next;
    }

    return 0;
}
```

```
List* list_append(List *l, int value_to_add)
{
    //Si es que esta vacio, entonces crear un nuevo nodo en el siguiente
valor
    if(l -> head == NULL)
    {
    l->head = node_new(value_to_add);
    return l;
    }
    //DE AQUI

    Node *current = l -> head;
    while(current -> next != NULL)
    {
        current = current->next;
    }

    current->next = node_new(value_to_add);

    return l;
}
```

Claro, adaptándolo al problema planteado. De igual manera reconocí que era necesaria la implementación de nodos (tema el cual al momento me causaba un poco de conflicto).

En un inicio optamos por hacer cada quién un intento por hacer el código y empezar a desarrollar las funciones que nos permitieran invertir las palabras, iterar en cada nodo, etc.

El planteamiento de la función *WordSep*: Función que permite encontrar en el head espacios o saltos de línea para posteriormente reservar memoria y busca palabra por palabra. Fue de la siguiente manera:

```
Node* WordReverse(FILE* file, FILE* resultfile, Node* head)
{
    if(head->next == NULL)
    {
        return head;
    }
    Node *next = NULL;
    Node *before = NULL;
    Node *newhead=NULL;
    while(head != NULL){ // mientras que head sea distinto a NULL
        next= head->next; // el next va a adquirir el siguiente delprimer
nodo
        head->next=before; //  Este va a ser NULL porque no va a apuntar
a nada
        before=head; // before tomará el primer nodo
        head=next; // el head tomará el siguiente void

    }
    newhead=before; // before tomará el valor de head
    return newhead; // Regresa la nueva cabeza de la lista
}
```

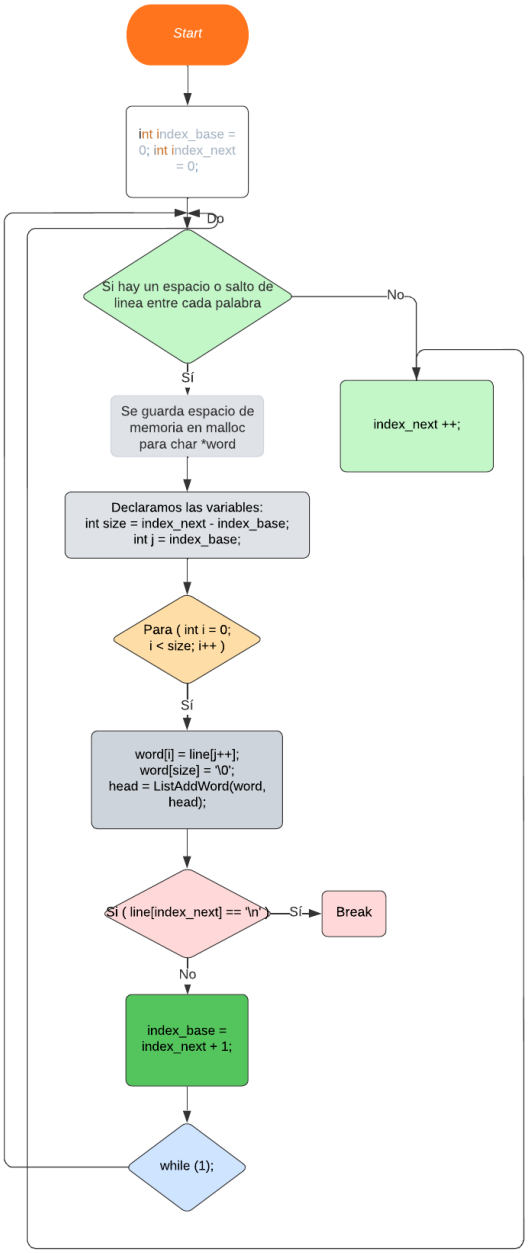
Para realizar la función *ReverseSomeWords* que nos permite identificar las palabras que tienen una longitud mayor a 5 caracteres mi compañero Diego intentó hacer el código y nos dimos cuenta que la implementación era similar a la de *list_contains* (mencionada anteriormente).

Como primera aproximación implementé lo siguiente (algunas variables pueden ser diferentes a las del documento original debido a que lo hice independientemente). Utilize *strrev* como parte de la librería string.h para invertir la palabra una vez cumplida la condición.

```
void reverse_some_words2(List *l)
{
    if (!(l->head)){
        return;}
    Node *current = l->head;
    while (current->next){
        if (current->character >= (char*)5){
            strrev(current->character);}
        current = current->next;}
    return;
}
```

Sin embargo, esta fue la implementación final.

```
Node* ReverseSomeWords(FILE* file, FILE*
resultfile, Node* head) //
{
    Node *current = head; // usa otra variable
    char *new_word;
    while (current != NULL) // mientras current
sea diferente de NULL
    {
        if (strlen(current->word) >= 5)   /
        {
            new_word = strrev(current->word);
            current->word = new_word;
        }
        current = current->next;
    }
    return head;
}
```



Se implementó un struct de *node* (prácticamente igual a los vistos en clase):

```
struct node // se crea la estrctura llamada nodo
{
    char *word;  // apunta a caracter (char)
    struct node* next;//apunta a siguiente nodo
} ;
```

Se creó una función para devolver el apuntador a un archivo, iniciado como null que devuelve mensaje de error en caso de no haber sido abierto correctamente. Regresa un file como resultado.

```
FILE* OpenFile(char *fileName, char *toDo)
{
    FILE *file = NULL; // inicializa el apuntador a archivo como NULL
    file = fopen(fileName, toDo); // se abre el archivo
    if (file == NULL)
    {
        printf("No se pudo abrir su archivo\n");
    }
    return file; // regresa el file (archivo)
}
```

También está la función *ListFillIn* que devuelve un apuntador de nodo a la cabeza de la lista (head)

```
Node* ListFillIn(FILE* file)
{
    Node *head = NULL; // se inicializa en NUll
    char *line = malloc(sizeof(char)*MAX_SIZE); //Se reserva espacio en
el heap (heap = espacio de memoria)
    line = fgets(line, MAX_SIZE, file); // obtiene la primera linea de
    if ( line == NULL ) // si la primera linea no tiene nada (NULL)
    {
        return NULL; // Regresa NULL
    }
    head = WordSep(line); // Pasa a funcion la linea
    return head; // regresa el head
}
```

El WordSep que devuelve el apuntador a la cabeza de la lista donde identifica los saltos o espacios para encontrar cada palabra, va iterando y cuando se encuentra una palabra, esta es agregada a la lista.

```
Node* WordSep(char* line)
{
    int index_base = 0; // se inicializan variables = 0
    int index_next = 0;
    Node *head = NULL;
    do
    {
        if(line[index_next] == ' ' || line[index_next] == '\n' )
            // Lo que hace es identificar espacios o saltos de linea para
así separar por palabras
        {
            char *word = malloc(sizeof(char)*MAX_SIZE); // reserva un
espacio en memoria en donde guardará la palabra
            int size = index_next - index_base;
            int j = index_base;
            for ( int i = 0; i < size; i++ )
            {
                word[i] = line[j++];
            }
            word[size] = '\0';
            // cuando se haya identificado una palabra esta se agrega a
la lista (el apuntador será al primer carácter de la palabra)
            head = ListAddWord(word, head);
            if ( line[index_next] == '\n' ) { // while
                break;
            }
            index_base = index_next + 1;
        }
        index_next ++;
    } while (1);

    return head; // Regresa head ya con la lista llena
}
```

LisrAddWord, agrega la palabra en cada uno de los nodos usando NULL para encontrar el final de la lista.

```
Node* ListAddWord(char *word_to_add, Node *head) // En esta lista se va a agregar palabra por palabra (nodo por nodo)
{
    {
        if (head == NULL) { // si la cabeza de nuestra lista esta vacia
            head = NewNode(word_to_add);// obtine un node de la memoria con malloc  para guardar ahí la palabra (la palabra es el nuevo nodo)
            return head; // regresa head
        } else { // si ya tiene una cabeza la lista entonces ...
            Node *current = head; // Se usa un apuntador a nodo para que head no se modifique
            while(current->next != NULL) // identifica el final de la lista
                current = current->next;
            current->next = NewNode(word_to_add); // agrega el nuevo nodo al final de la lista
            return head; // regresa head
        }
    }
}
```

Lo siguiente agrega un nuevo nodo (se guarda la memoria en malloc)

```
Node* NewNode(char *word_to_add) // devuelve el nuevo nodo
{
    Node *new = malloc(sizeof(Node)); // reserva espacio para el nuevo nodo
    new->word = word_to_add; // se asigna la palabra
    new->next = NULL; // se asigna el siguiente nodo como NULL
    return new; //Regresa el nuevo nodo
}
```

PrintCadenita imprime el resultado de la cadena

```
void PrintCadenita(Node *head) //Función de impresión del resultado
{
    Node *current = head; // usa otra variable para que head no se modifique
    while(current != NULL) // mientras que current sea diferente a NULL
    {
        printf("WORD: %s\n", current->word); //Se imprime la palabra
        current = current->next; // se actualiza el nodo que estamos imprimiendo
    } ;
}
```

```
void WriteResult(FILE* resultsfile, Node * head) { // Función que escribe el resultado en el archivo de result.txt
    char *line = malloc(sizeof(char)*MAX_SIZE); // reserva el espacio en el heap
```

```
    line[0] = '\0'; // inciializa en NULL
    Node * current = head; // Se usa otra variable para que head no se
modifique

    while(current != NULL){ //mientras current ( apuntador anodo) sea
diferente de NULL
//stract =  añade un bloque de memoria a otro (va creando un string con
todas las palabras (nodos) de una linea)
// LOS DOS BLOQUES DEBEN TERMINAR CONUN CARÁCTER NULL
        strcat(line, current->word);
        strcat(line, " ");
        current = current->next;
    }
    strcat(line, "\n"); // Acaba de escribir una linea , y hace saltode
linea
    fputs(line, resultsfile); // escribe la linea enel archivo de
result.txt
}
```

Esta fue una de las implementaciones que traté hacer, sin embargo, cometí varios errores en cuanto a los apuntadores. Lo siguiente invierte el orden de la palabra.

```
Node* WordReverse(FILE* file, FILE* resultfile, Node* head)
{
    if(head->next == NULL)
    {
        return head;
    }
    Node *next = NULL;
    Node *before = NULL;
    Node *newhead=NULL;
    while(head != NULL){ // mientras que head sea distinto a NULL
        next= head->next; // elnext va a adquirir el siguiente delprimer
nodo
        head->next=before; //  Este va a ser NULL porque no va a apuntar
a nada
        before=head; // before tomará el primer nodo
        head=next; // el head tomará el siguiente vodo

    }
    newhead=before; // before tomará el valor de head

    return newhead; // Regresa la nueva cabeza de la lista
}
```

Esto invierte el orden del string, obtiene el texto del file.

```
void reverse_string(int op) // función de elección de lo que queramos
lograr
{
    FILE* file = OpenFile("../input.txt", "r"); // lee el archivo input
    FILE* resultfile = OpenFile("../result.txt", "w"); //escribe en el
archvio result

    do {
```

```
        Node *head = ListFillIn(file);  // llena la lista
        if ( head == NULL ) {  // si el head es igual a NULL sale del
ciclo
            break;
        }
        // ESCOGER CUAL DE LAS DOS FUNCIONES DE ABAJO USAR...
        if (op == 1) // si en el argumento de main(reverse string) se
pone 1
        { // entonces el head realizará el volteo de palabras
            head = WordReverse(file, resultfile, head);

        } else if  (op == 2){
            // entonces el head realizará el volteo de las letras de las
palabras
            head = ReverseSomeWords(file, resultfile, head);

        } else {
            // si no se aplica ninguna de las opciones anteriores se
imprime opcionno valida  ...
            printf("Opcion invalida");
            break;
        }
        WriteResult(resultfile, head); // escribe el resultado en el
archivo result.txt
        PrintCadenita(head); // imrpime la cadena final
    } while ( 1 );
    fclose(file); // cierre del archivo input
    fclose(resultfile); // cierre del archivo result
}
```

Reconozco que mis compañeros realizaron la mayor parte del proyecto y mi aportación fue únicamente para proporcionar opciones de implementación, ideas, tratar de ayudar a solucionar problemas encontrando códigos similares que pudieran ser de ayuda y realizar diagramas de flujo en complemento al documento final. En cuanto a la organización para realizar el proyecto intervine en videollamadas para dar solución a código.

Después de revisar el resultado del trabajo realizado, puedo afirmar que a pesar de que casi no tuve participación en cuanto al código, logro comprender mejor la implementación y el trabajar en conjunto me sirvió para resolver dudas en la utilización de nodos.