Parcial 2

1 Considere la siguiente implementación de Listas Simplemente Enlazadas generales.

```
typedef struct _GNodo {
    void* dato;
    struct _GNodo* sig;
} GNodo;

typedef struct {
    GNodo* primero;
    GNodo* ultimo;
} GList;

typedef void  (*FuncionDestructora)(void *dato);
typedef void* (*FuncionCopia)(void *dato);
typedef void  (*FuncionVisitante) (void* dato);
typedef void* (*FuncionTransformar)(void* dato);

GList glist_crear () {
    GList lista;
    lista.primero = NULL;
    lista.ultimo = NULL;
    return lista;
}

GList glist_agregar_final (GList lista, void* dato, FuncionCopia copia) {
    GNodo* nuevo = malloc(sizeof(GNodo));
    nuevo->dato = copia(dato);
    nuevo->sig = NULL;

    if (lista.primero == NULL)
        lista.primero = nuevo;
    else
        lista.ultimo->sig = nuevo;

    lista.ultimo = nuevo;
    return lista;
}

void glist_destruir (GList lista, FuncionDestructora destroy) {
    GNodo* temp;
    for (GNodo* nodo = lista.primero; nodo != NULL; ){
        temp = nodo;
        nodo = nodo->sig;
        destroy(temp->dato);
        free(temp);
    }
}
```

```
void glist_recorrer (GList lista, FuncionVisitante visit) {
    for (GNodo* nodo = lista.primero; nodo != NULL ; nodo = nodo->sig)
        visit(nodo->dato);
}
```

a) Defina una función que devuelva el resultado de realizar un map a la lista, es decir, que aplique la función f a cada uno de los elementos de la misma. El prototipo de la misma sería:

```
GList glist_map(GList lista, FuncionTransformar f, FuncionCopia c);
```

b) Defina una función *stringMayuscula* que cumpla con el prototipo de FuncionTransformar. La misma debe tomar un string y devolver un nuevo string igual al anterior pero con todos sus caracteres alfabéticos en mayúscula.

c) Muestre cómo utilizaría estas dos funciones para convertir una lista de strings en otra cuyos elementos están todos en mayúscula.

```
void* copia_string (void* dato){
    char* str = (char*) dato;
    char* nuevo = malloc(sizeof(char)*(strlen(str)+1));
    strcpy(nuevo, str);
    return nuevo;
}

void dest_string (void* dato) {
    free(dato);
}

int main() {
    GList lista = glist_crear();
    char* str = "Licenciatura ";
    lista = glist_agregar_final(lista, str, copia_string);
    str = "en ";
    lista = glist_agregar_final(lista, str, copia_string);
    str = "Ciencias ";
    lista = glist_agregar_final(lista, str, copia_string);
    str = "de la ";
    lista = glist_agregar_final(lista, str, copia_string);
    str = "Computacion.";
    lista = glist_agregar_final(lista, str, copia_string);

    // COMPLETAR

    glist_destruir(lista, dest_string);

    return 0;
}
```

2 Considere la siguiente implementación de mergesort:

```c
#include <stdio.h>
#include <stdlib.h>

void combinar(int *array, int inicio, int mitad, int fin) {
  int tam_izq = mitad - inicio + 1;
  int tam_der = fin - mitad;

  int *a_izq = malloc(sizeof(int) * tam_izq);
  int *a_der = malloc(sizeof(int) * tam_der);

  for (int i = 0; i < tam_izq; i++){
   a_izq[i] = array[i + inicio];
  }
  for (int i = 0; i < tam_der; i++){
   a_der[i] = array[i + mitad + 1];
  }

  int min = 0;
  for (int i = 0, j = 0, k = inicio; k <= fin; k++){
    if (i < tam_izq && (j >= tam_der || a_izq[i] <= a_der[j])) {
      min = a_izq[i];
      i++;
    } else {
      min = a_der[j];
      j++;
    }

    array[k] = min;
  }

  free(a_izq);
  free(a_der);
}

void merge_sort_R(int *array, int inicio, int fin) {
  if (inicio >= fin)
    return;

  int mitad = inicio + (fin - inicio) / 2;

  merge_sort_R(array, inicio, mitad);
  merge_sort_R(array, mitad + 1, fin);

  combinar(array, inicio, mitad, fin);
}

void merge_sort(int *array, int tam) {
  merge_sort_R(array, 0, tam - 1);
}
```

Se pide que realice una implementación de mergesort para listas de enteros respetando el nombre de las funciones dadas.