cPigeon

Una empresa está desarrollando "cPigeon", un nuevo sistema de mensajería instantánea para teléfonos móviles del estilo de Whatsapp / Telegram. En la primera fase se realiza el diseño de los TADs básicos para la aplicación que correrá en cada teléfono móvil. Se requiere que la aplicación tenga en cada teléfono móvil la siguiente información:

- **Conjunto de contactos**: cada contacto está definido su nombre y número de teléfono.
- **Conjunto de chats:** cada chat está definido por un conjunto de mensajes. Cada mensaje consta de una cadena de caracteres, un campo "timestamp" (entero que codifica el instante de tiempo en el que se ha enviado) y el número de teléfono del emisor. Nota: la aplicación puede tener varios chats a la vez.
- **Datos del propietario del móvil**: nombre y número de teléfono móvil del propietario.

1. Enumera los TADs necesarios para almacenar los datos de la aplicación, proporciona en C las estructuras de datos para implementar cada uno de los TAD, y especifica el nombre del fichero .c en el que se define cada una de esas estructuras. La aplicación puede gestionar a la vez varios chats.

   Se ha realizado un diseño con seis TADs: DatosContacto, Contactos, Mensaje, Chat, Chats, cPigeon

En datoscontacto.c:

```
#define MAX_NOMBRE 20

struct _DatosContacto {
        char nombre [MAX_NOMBRE];
        int numero;
};
```

En contactos.c:

```
#define MAX_CONTACTOS 500

struct _Contactos {
        DatosContacto *datos[MAX_CONTACTOS];
        int ncontactos;
};
```

En mensaje.c:

```
#define MAX_MENSAJE 1000

struct _Mensaje {
        char cadena[MAX_MENSAJE];
        int timestamp;
        int numRemitente;
};
```

En chat.c:

```
#define MAX_MENSAJES 5000

struct _Chat {
        int nmensajes;
        Mensaje *mensajes[MAX_MENSAJES];
};
```

En chats.c:

```
#define MAX_CHATS 400

struct _Chats {
        int nchats;
        Chat *chats[MAX_CHATS];
}
```

En cpigeon.c:

```
struct cPigeon {
        Contactos *c;
        Chats *ch;
        DatosContacto *datospropietario;
}
```

2. Proporciona en C el código de la primitiva **mensaje_crear**, que reserva memoria para un mensaje, lo inicializa con el valor de timestamp correspondiente, el número del remitente, y copia la cadena con el texto del mensaje.

Mensaje *mensaje_crear (const char *cadena, int timestamp, int numtel);

```
Mensaje *mensaje_crear(const char *cadena, int timestamp, int numtel) {
        Mensaje *pm;

        if(!cadena) return NULL;

        pm = (Mensaje *) malloc(sizeof(Mensaje));
        if ( !pm) return NULL;

        strcpy(pm->cadena, cadena);
        pm->timestamp = timestamp;
        pm->numRemitente = numtel;

        return pm;
}
```

3. Proporciona el código C de la siguiente función:

Status chat_anyade_mensaje (Chat *pc, const char *cadena, int timestamp, int numtel);

que añade un mensaje a un chat. Asume que se ha definido previamente:

```
typedef enum {ERROR, OK} Status;
```

```
Status chat_anyade_mensaje (Chat *pc, const char *cadena, int timestamp, int numtel) {
        if((!pc) || (!cadena)) return ERROR;

        if (pc-> nmensajes == MAX_MENSAJES) return ERROR;

        pc->mensajes[pc->nmensajes] = mensaje_crear(cadena, timestamp, numtel);
        if (! ( pc->mensajes[pc->nmensajes] )) return ERROR;

        pc->nmensajes ++;
        return OK;
 }
```