Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Arquitectura de Computadoras y Ensambladores 1
Primer Semestre 2023
Ing. Álvaro Obrayan Hernández García
Tuto Académico: Ronald Marín

MANUAL TÉCNICO

Nombre	**Carnet**
Gustavo Alejandro Girón Arriola	201900898
Cristian Aramis López Bautista	201904042
Christopher Alexander Acajabon Gudiel	201404278
Erick Estuardo Muñoz Escalante	201602947
Jose Pablo Ceron Urizar	201908251

Inicio del Programa

Estados utilizados para el acceso a las diferentes secciones

```
enum EstadoMenu
{
  ESPERANDO,
  DATOS_INTEGRANTES,
  MENU_PRINCIPAL,
  INICIO_SESION,
  REGISTRO,
  MENU_USUARIO,
  MENU_ADMIN,
  INGRESO_DISPOSITIVO,
  RETIRO_DISPOSITIVO,
  CERRAR_SESION,
  ELMINAR_USUARIO,
  VISTA_LOGS,
  ESTADO_SISTEMA
};
```

Mostrar los datos de los integrantes del grupo:
Función para imprimir en la pantalla lcd los datos de los integrantes del grupo. Esta función se llama para iniciar el programa con la pantalla principal.

```
void mostrarDatosIntegrantes()
{
  // filling teamates
  equipo[0] = "Gustavo Alejandro Giron Arriola - 201900898";
  equipo[1] = "Cristian Aramis Lopez Bautista - 201904042";
  equipo[2] = "Christopher Alexander Acajabon Gudiel - 201404278";
  equipo[3] = "Erick Estuardo Muñoz Escalante - 201602947";
  equipo[4] = "Jose Pablo Ceron Urizar - 201908251";

  for (int i = 0; i < NUM_INTEGRANTES ; i++) // recorrido integrantes
  {
    pantalla.setCursor(0, 0);
    pantalla.print("--INTEGRANTES--"); // tiempo en mostrar los datos de los integrantes

    varTemp = equipo[i];

    pintarCadenaMayorHorizontal(pantalla, varTemp, 1, 16);
    delay(300);

    varTemp = "";
    pantalla.clear();
  }
}
```

```
void pintarCadenaMayorHorizontal(String cadena)
{
  String varMostrar = "";
  for (int j = 0; j < cadena.length(); j++) // recorre cadena
    {
      char caracter = cadena.charAt(j);
      varMostrar = varMostrar + caracter;

      if (varMostrar.length() > CASILLAS_PANTALLA)
      {
        varMostrar = varMostrar.substring(1);
      }

      pantalla.setCursor(0, 1);
      pantalla.print(varMostrar);
      delay(50);
    }
}
```

Menú

Funciones Principales del Flujo Central del Programa

Inicio de Sesión:
Se solicita la verificación de los datos ingresados (sea por vía bluetooth con la aplicación móvil o ya sea por medio del panel) a la memoria.

```
bool verificarLogin(char* nombre, char* contr) {
    int pos_memoria = buscarUsuario(nombre);
    if(pos_memoria == 0) {
      Serial1.println("El usuario no existe");
      return false;
    }
    struct usuario actual;
    actual = leerMemoriaUsuario(pos_memoria);
    bool validar = false;
    for (int i=0; i<12; i++) {
      if(actual.password[i] == contr[i]) {
        validar = true;
      } else {
        validar = false;
        break;
      }
    }

    if (validar) {
      return true;
    } else {
      return false;
    }
}
```

Si las credenciales son correctas se guarda un registro del evento en la memoria y se procede a ejecutar la lógica de las opciones siguientes según el rol del usuario.

```
if(verificarLogin(nombre_temp, contra_temp)){
  Serial1.println("LOGIN CORRECTO");
  //guardar log
  char descripcion[13] = {'L','o','g','i','n',' ','E','x','i','t','o', 's', 'o'};
  char mylog[15];
  memset(mylog, 0, 15);
  memcpy(mylog, descripcion, 13);
  guardarMemoriaLog(mylog);
  return true;
```

Selección de tipo de Rol (administrador o usuario):
Con una función de tipo *bool* se verifica si las credenciales ingresadas coinciden con los datos del administrador y retorna *true*, sino coinciden, se determina que se trata de un usuario normal retornando *false.*

```
bool tipoRol(char nombreUsuario[12]){
  String nombreLogin = "";
  nombreLogin.concat(nombreUsuario[0]);
  nombreLogin.concat(nombreUsuario[1]);
  nombreLogin.concat(nombreUsuario[2]);
  nombreLogin.concat(nombreUsuario[3]);
  nombreLogin.concat(nombreUsuario[4]);
  nombreLogin.concat(nombreUsuario[5]);
  nombreLogin.concat(nombreUsuario[6]);
  nombreLogin.concat(nombreUsuario[7]);
  nombreLogin.concat(nombreUsuario[8]);
  nombreLogin.concat(nombreUsuario[9]);
  nombreLogin.concat(nombreUsuario[10]);

  if(nombreLogin.equals("admin!82871")){
    char descripcion[11] = {'L','o','g','i','n',' ','a','d','m','i','n'};
    char mylog[15];
    memset(mylog, 0, 15);
    memcpy(mylog, descripcion, 11);
    guardarMemoriaLog(mylog);
    return true;
  }

  return false;
}
```

Registro de nuevos usuarios y eliminación de cuenta:
Para hacer el registro de un nuevo usuario y la eliminación de cuentas se utilizan unas de las funciones detalladas en la sección de **Memoria EEPROM.**

Sensores de Temperatura:
Verificar que los sensores de temperatura mantengan un valor aceptable.

```
void lecturaSensores(LedControl ledControl){

  Serial3.print("S1");
  Serial3.readBytes(respuestaSensorTemperatura, 2);
  verificarTemperatura(respuestaSensorTemperatura[0], respuestaSensorTemperatura[1], '1');
```

```
void verificarTemperatura(char valor1, char valor2, char sensor){

  bool tempLeidaCorrecta = false;
  bool tempLeidaCorrecta2 = false;

  if(valor1 == '0' || valor1 == '1' || valor1 == '2' || valor1 == '3' || valor1 == '4' ||
    valor1 == '5' || valor1 == '6' || valor1 == '7' || valor1 == '8' || valor1 == '9'){
    tempLeidaCorrecta = true;
    //Serial1.println(valor1);
  }

  if(valor2 == '0' || valor2 == '1' || valor2 == '2' || valor2 == '3' || valor2 == '4' ||
    valor2 == '5' || valor2 == '6' || valor2 == '7' || valor2 == '8' || valor2 == '9'){
    tempLeidaCorrecta2 = true;
    //Serial1.println(valor2);
  }

  if(tempLeidaCorrecta == true && tempLeidaCorrecta2 == true){ // numero >= 10
    // formar numero leido
    String numeroLeido = "";
    numeroLeido.concat(valor1);
    numeroLeido.concat(valor2);

    int numeroReal = String(numeroLeido).toInt();
    updateLecturaSensorTemperatura(sensor, numeroReal);
```

Compartimentos:
Para implementar las funciones de ingreso y retiro de celular se utilizan las siguientes funciones para acceder y actualizar la matriz que registra el estado de los compartimentos.

```
int buscarCompartimentoLibre(){
  for (int i = 0; i < 9; i++){
    if(respuestaSensorBoton[i] == '0'){ // compartimento en uso
      return i+1;
    }
  }

  return -1; // no encontrado
}
```

```
bool verificarEstadoCompartimentoRetirado(int comp){
  if(respuestaSensorBoton[comp-1] == '0'){ // compartimento libre
    return true;
  }
  return false;
}
```

```
bool verificarEstadoCompartimentoIngresado(int comp){
  if(respuestaSensorBoton[comp-1] == '1'){ // compartimento en uso
    return true;
  }
  return false;
}
```

Mostrar Logs y Estado del Sistema:
Recorren la información almacenada en la memoria EEPROM, accediendo a los espacios reservados correspondientes.

```
void enlistarLogs(LiquidCrystal pantalla)
{
    byte noLogs = obtenerNumeroLogs();
    int logSize = (int)noLogs;
    int contMostrados = 0;
    int contPintar = 0;
    String estado = "";

    while (contMostrados < logSize)
    {
        if (estado == "MOSTRAR")
        {
            while (contPintar < 4)
            {
                struct evento log = buscarLog(contMostrados);
                pintarCadenaMayorHorizontal(pantalla, 16, log.descripcion, contPintar);
                contMostrados++;
                contPintar++;
            }
            estado = "SELECCION";
            break;
        }
        else if (estado == "SELECCION")
        {
            if (digitalRead(2) == LOW) // SIGUIENTE PAGINA
            {                          // ACEPTAR
                estado = "MOSTRAR";
                contPintar = 0;
                break;
            }
            else if (digitalRead(3) == LOW) // SALIR DEL LOG
            {                               // CANCELAR
```

```
void mostrarEstadosSistema(int pagina)
{
  String cadena1 = "Celulares Ingresados: " + String(obtenerCantidadEstado(CELULARES));
  String cadena2 = "Intentos Fallidos: " + String(obtenerCantidadEstado(INTENTOS_FALLIDOS));
  String cadena3 = "Incidentes: " + String(obtenerCantidadEstado(INCIDENTES));
  String cadena4 = "Usuarios Activos: " + String(obtenerCantidadEstado(USUARIOS_ACTIVOS));
  Serial1.println(cadena1);
  Serial1.println(cadena2);
  Serial1.println(cadena3);
  Serial1.println(cadena4);
  Serial1.println("");

  if (!enviado)
  {
    Serial.print(cadena1 + cadena2 + cadena3 + cadena4);
    enviado = true;
  }
  if (pagina == 1)
  {
    pantalla.clear();
    pantalla.setCursor(0, 0);
    pantalla.print("- Celulares");
    pantalla.setCursor(0, 1);
    pantalla.print("  ingresados: " + String(obtenerCantidadEstado(CELULARES)));
    pantalla.setCursor(0, 2);
    pantalla.print("- Intentos");
    pantalla.setCursor(0, 3);
    pantalla.print("  fallidos: " + String(obtenerCantidadEstado(INTENTOS_FALLIDOS)));
  }
  else if (pagina == 2)
  {
    pantalla.clear();
    pantalla.setCursor(0, 0);
    pantalla.print("- Incidentes: " + String(obtenerCantidadEstado(INCIDENTES)));
    pantalla.setCursor(0, 2);
    pantalla.print("- Usuarios");
    pantalla.setCursor(0, 3);
    pantalla.print("  activos: " + String(obtenerCantidadEstado(USUARIOS_ACTIVOS)));
  }
}
```

Bluetooth

Realizar conexión:
Antes de realizar la conexión con la aplicación móvil se procede a vaciar el buffer en caso de que por algún error exista algún dato en tránsito. Luego se solicitan datos a la aplicación para verificar que hay una conexión exitosa.

```
void esperando(LiquidCrystal pantalla){

    limpiarBuffer();
      pantalla.clear();
      pantalla.print(" Esperando  una ");
      pantalla.setCursor(0, 1);
      pantalla.print("   conexion...  ");
      bool alguienPorAhi = false;
          char recibidos[3];
      LOOP {
          while(Serial.available()) {
          Serial.readBytes(recibidos, 2);
          alguienPorAhi = true;
      }
      if (alguienPorAhi && !Serial.available()) break;
      }

}
```

```
void limpiarBuffer() {
    int t0 = millis();
    int t1 = millis();
    LOOP {
        t1 = millis();
        while(Serial.available()) {
        Serial.read();
        }
        if ((t1 - t0 >= 1000) && !Serial.available()) break;
    }
}
```

Enviar y recibir datos:
Para el envío y recepción de datos se utilizan las funciones del puerto Serial del arduino correspondiente. Se envían datos sobre lo que se debe recibir de la aplicación para confirmar el tipo de dato que se está solicitando.

```
void ingresar(int x, LiquidCrystal pantalla){

  if(x == 0){

            memset(nombre_tempp, 0, 11);
            memset(contra_tempp, 0, 11);
            memset(numero_tempp, 0,  9);
            struct usuario nuevo_usuario;
        LOOP {
            limpiarBuffer();
            enviarConfirmar("Nombre:");
                memset(nombre_tempp, 0, 11);
                pantalla.clear();
                pantalla.print("L O G I N");
                pantalla.setCursor(0, 1);
                pantalla.print(" - NOMBRE:");
                pantalla.setCursor(0, 2);
                // OBTENER CADENA DE APLICACIÓN -- Nombre
                bool seEnvioAlgo = false;
                int indiceNombre = 0;
                long int t0 = millis();
                long int t1 = millis();
            limpiarBuffer();
                LOOP {
                    // SI YA SE ENVIO ALGO DESDE LA APLICACION
                    while (Serial.available()) {
                        seEnvioAlgo = true;
                    //   RECIBIRLO
                        nombre_tempp[indiceNombre++] = Serial.read();
```

```
                nombre_tempp[indiceNombre++] = Serial.read();
            }
            // CONTROLAR CUANTO HA PASADO DESDE QUE COME...
            if (seEnvioAlgo) {
                t1 = millis();
                if (t1 - t0 >= 500) break;
            } else {
                t0 = millis();
                t1 = millis();
            }
        }
        pantalla.print(nombre_tempp);
        pantalla.setCursor(0, 3);
        pantalla.print("Correcto?       ");
        delay(500);
        if (entradaAceptada()){
        //estadoActual = MENU_PRINCIPAL;
        break;
      }

  pantalla.setCursor(0, 2);
  pantalla.print(LINEA_VACIA);
  pantalla.setCursor(0, 3);
  pantalla.print(LINEA_VACIA);
    }
```

```
  LOOP {
  limpiarBuffer();
  enviarConfirmar("Contraseña:");
    memset(contra_tempp, 0, 11);
    pantalla.clear();
    pantalla.print("L O G I N");
    pantalla.setCursor(0, 1);
    pantalla.print(" - Contras:");
    pantalla.setCursor(0, 2);
    // OBTENER CADENA DE APLICACIÓN -- Nombre
    bool seEnvioAlgo = false;
    int indiceContra = 0;
    long int t0 = millis();
    long int t1 = millis();
  limpiarBuffer();
    LOOP {
        // SI YA SE ENVIO ALGO DESDE LA APLICACION
        while (Serial.available()) {
            seEnvioAlgo = true;
        //   RECIBIRLO
            contra_tempp[indiceContra++] = Serial.read();
        }
        // CONTROLAR CUANTO HA PASADO DESDE QUE COME...
        if (seEnvioAlgo) {
            t1 = millis();
            if (t1 - t0 >= 500) break;
        } else {
            t0 = millis();
            t1 = millis();
        }
```

```
                }
            }
      imprimirAsteriscos(contra_tempp, pantalla);
              pantalla.setCursor(0, 3);
              pantalla.print("Correcto?       ");
              delay(500);
              if (entradaAceptada()) break;
      pantalla.setCursor(0, 2);
      pantalla.print(LINEA_VACIA);
      pantalla.setCursor(0, 3);
      pantalla.print(LINEA_VACIA);
        }
```

Memoria EEPROM

Manejo de Información en memoria:
Para guardar usuarios, eventos y compartimientos en la memoria EEPROM primero se crea un *struct* para cada elemento, que servirá para almacenar los atributos que contienen la información necesaria de un elemento del sistema.

```
struct usuario {
    char nombre[13];
    char password[13];
    char phone[9];
};
```

```
struct evento {
    byte identificador[1];
    char descripcion[16];
};
```

```
struct compartimiento {
  char posicion[2];
  char pos_memoria[4];
};
```

Antes de almacenar los datos se utiliza un cifrado xor, que se aplica dos veces sobre los datos que se almacenarán en memoria.

```
// Encripta dos veces los datos del array
// Desencripta si se invierten las claves
void encriptar(char* info, int tamano_array, byte clave1, byte clave2) {
    for (int i=0; i<tamano_array; i++) {
        info[i] = info[i] ^ clave1;
        info[i] = info[i] ^ clave2;
    }
}
```

```
void guardarMemoriaUsuario(struct usuario nuevo_usuario) {
    int encontrado = buscarUsuario(nuevo_usuario.nombre);
    if(encontrado == 0) {
      Serial1.println("Guardando usuario en la memoria...");
    } else {
      Serial1.println("Ocupado: Ya existe un usuario con el mismo nombre");
      return;
    }

    EEPROM.get(0, numero_usuarios);
    int siguiente_direccion = 1;
    for (int i=0; i<numero_usuarios; i++) {
        siguiente_direccion += sizeof(struct usuario);
    }

    encriptar(nuevo_usuario.nombre, sizeof(nuevo_usuario.nombre), CLAVE1, CLAVE2);
    encriptar(nuevo_usuario.password, sizeof(nuevo_usuario.password), CLAVE1, CLAVE2);
    encriptar(nuevo_usuario.phone, sizeof(nuevo_usuario.phone), CLAVE1, CLAVE2);
    EEPROM.put(siguiente_direccion, nuevo_usuario);

    // guardar en la primera celda de memoria la cantidad de usuarios existentes
    numero_usuarios++;
    EEPROM.put(0, numero_usuarios);
}
```

Para guardar un usuario se recibe como parámetro el nuevo usuario que se almacenará en la memoria

Antes de guardar el nuevo usuario se verifica si ya existe un usuario con el mismo nombre, en el caso de que exista, no se guardará nada en la memoria.

Para guardar un log/evento se recibe como parámetro una descripción de tipo char[].

```
void guardarMemoriaLog(char* descripcion) {
    struct evento evt;
    EEPROM.get(particion_logs, numero_logs);
    if (numero_logs > 99) {
      numero_logs = 100 - numero_logs;
    }

    evt.identificador[0] = numero_logs;
    memcpy(evt.descripcion, descripcion, sizeof(evt.descripcion));

    encriptar(evt.identificador, 1, CLAVE1, CLAVE2);
    encriptar(evt.descripcion, sizeof(evt.descripcion), CLAVE1, CLAVE2);

    int pos = particion_logs + (numero_logs*sizeof(struct evento)) +1;

    EEPROM.put(pos, evt);

    numero_logs++;
    EEPROM.put(particion_logs, numero_logs);
}
```

El guardado de los compartimentos almacena información sobre la ubicación del compartimiento en el que se encuentra un celular y también la información del dueño del celular correspondiente. Recibe como parámetro un struct compartimiento.

```
void guardarCompartimiento(struct compartimiento cmp) {
    int p = atoi(cmp.posicion);

    encriptar(cmp.posicion, sizeof(cmp.posicion), CLAVE1, CLAVE2);
    encriptar(cmp.pos_memoria, sizeof(cmp.pos_memoria), CLAVE1, CLAVE2);

    int pos = particion_compartimientos + (p*sizeof(struct compartimiento));
    EEPROM.put(pos, cmp);
}
```

Para eliminar información en la memoria se crea un nuevo *struct* del tipo de elemento que se desea eliminar; el nuevo *struct* se utiliza para almacenar ceros en los espacios en memoria necesarios, de esta manera se simula un espacio sin información.

```
void eliminarUsuario(char* nombre) {
    int posicion = buscarUsuario(nombre);
    if (posicion == 0) {
        Serial1.println("No se pudo eliminar el usuario");
        return;
    }

    struct usuario eliminar = {
        "0",
        "0",
        "0"
    };

    EEPROM.put(posicion, eliminar);
    Serial1.println("Usuario eliminado");
}
```

```
void vaciarCompartimiento(char* posicion) {
    int pos = particion_compartimientos + (atoi(posicion)*sizeof(struct compartimiento));

    char ceros[4] = "000";
    struct compartimiento cmp;
    memcpy(cmp.posicion, posicion, sizeof(cmp.posicion));
    memcpy(cmp.pos_memoria, ceros, sizeof(ceros));
    guardarCompartimiento(cmp);
}
```

Sensores

Los compartimentos de los celulares se representan por medio de una matriz de leds, para identificar con qué espacio se está trabajando se utiliza una instrucción *switch-case*.

```
int valorSensorLeido;

char solicitudSensorTemperatura[2];
String valorSensorRetornar;

String valorSensorString(char numeroSensor){
  switch(numeroSensor){
    case '1': {
      valorSensorLeido = analogRead(A0); // sensor 1
      break;
    }
    case '2': {
      valorSensorLeido = analogRead(A1); // sensor 2
      break;
    }
    case '3': {
      valorSensorLeido = analogRead(A2); // sensor 3
      break;
    }
    case '4': {
      valorSensorLeido = analogRead(A3); // sensor 4
      break;
    }
    case '5': {
      valorSensorLeido = analogRead(A4); // sensor 5
      break;
    }
    case '6': {
      valorSensorLeido = analogRead(A5); // sensor 6
      break;
    }
    case '7': {
      valorSensorLeido = analogRead(A6); // sensor 7
      break;
    }
    case '8': {
      valorSensorLeido = analogRead(A7); // sensor 8
      break;
    }
    case '9': {
      valorSensorLeido = analogRead(A8); // sensor 9
      break;
    }
  }

  // calculo en grados C
  String myString = "";
  float mv = (valorSensorLeido/1024.0)*5000;
  int temp = (int)(mv / 10);

  if(temp <= 9){
    myString = "0";
  }

  myString.concat(temp);
  return myString;
}
```

Con la siguiente función se realiza la comunicación entre arduinos y envía los valores solicitados por el arduino controlador o principal.

```
void leerSensoresTemperatura(){

    if (Serial3.available()) {
      Serial3.readBytes(solicitudSensorTemperatura, 2);
      switch (solicitudSensorTemperatura[0]) {
        case 'S': { // sensores temperatura
          valorSensorRetornar = valorSensorString(solicitudSensorTemperatura[1]);
          Serial3.print(valorSensorRetornar);
          Serial1.println(valorSensorRetornar);
          break;
        }
        case 'B': { // botones
          valorSensorRetornar = "";
          for(int i = 2; i <= 10; i++){ // pines botones
            if(digitalRead(i) == LOW){ // presionado
              valorSensorRetornar.concat(1);
            }else{
              valorSensorRetornar.concat(0);
            }
          }
          Serial3.print(valorSensorRetornar);
          Serial1.println(valorSensorRetornar);
          break;
        }
      }
    }
```