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

Al inicio del programa se muestran los datos de los integrantes del grupo en pantalla lcd con la siguiente función

```
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 < 5; i++) // recorrido integrantes
  {
    pantalla.setCursor(0, 0);
    pantalla.print("--INTEGRANTES--"); // tiempo en mostrar los datos de los integrantes

    varTemp = equipo[i];

    pintarCadenaMayorHorizontal(varTemp);
    delay(500);

    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);
    }
}
```

```
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
```

Menú

Después de la pantalla inicial se dará acceso al menú principal, en el cual se tiene la opción de iniciar sesión, donde se hace una verificación con el nombre y la contraseña. Existen dos tipos de usuarios según su rol, administrador y usuario.

```
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;
    }
}
```

```
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;
```

Si las credenciales son correctas se guarda un registro del evento en la memoria.

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;
        }
      }
    }
```