This is the Version 1 software for the Madlab Storage system. The headers are equivalent to new tabs in the Arduino environment.

Madlab Storage

```
#include <LiquidCrystal.h>
#include <NewSoftSerial.h>

//Setup Software Serial port
NewSoftSerial rfid(2,3); //RX 2, TX 3 (from Arduino Perspective)

//LCD variables
LiquidCrystal lcd(14,15,16,17,18,19); //setup LCD on 14 - 19 (analog 0 to 5)
const int lcdBacklight = 4; //backlight enable pin
unsigned long time; // current time in millis
unsigned long prevTime; // previous time that backlight was on
long lcdBackOnTime = 5000; // time in millis() to leave backlight on for
(approximately)

//button variables
const int buttonUp = 8;
const int buttonDown = 9;
const int buttonEnter = 10;
const int buttonBack = 11;
const int buttonRed = 12;
const int lock = 13;

//Menu global variables
int menuTagNo;
int menuLevel;
int prevTagNo;
int prevMenuLevel;

char newName[4];

//testing variables CHANGE WHEN INTRODUCING MORE FEATURES THAT OVERWRITE
NECESSITY
union Header
{
  byte raw[3];
  struct Data
  {
    byte hByte;
    int noTag;
  }
  data;
}
header;
```

```
void setup()
{
  Serial.begin(9600);
  rfid.begin(9600);
  rfid.flush();
  lcd.begin(16,2);
  lcd.clear();
  lcd.print("Initialising");
  pinMode(buttonUp,INPUT);
  pinMode(buttonDown,INPUT);
  pinMode(buttonEnter,INPUT);
  pinMode(buttonBack,INPUT);
  pinMode(buttonRed,INPUT);
  pinMode(lcdBacklight,OUTPUT);
  pinMode(lock,OUTPUT);
  createHead();

  //setup menu and other variables
  menuTagNo = 0;
  menuLevel = 0;
  prevTagNo = -1;
  prevMenuLevel = -1;
  time = millis();
  prevTime = 0;
}

void loop()
{
  buttonRedCheck();
  menu();
  buttonMainCheck();
  lcdBacklightTimer();
  checkRFID();
}

void openDoor()
{
  digitalWrite(lock,HIGH);
  delay(5000);
  digitalWrite(lock,LOW);
}
```

Button

```
//Functions related to buttons


// function changes the menuTagNo and menuLevel depending on buttons for
```

```
menu functionality
void buttonMainCheck()
{
  time = millis();
  if(buttonUpCheck())
  {
    prevTime = time;
    if((menuLevel == 0) && (menuTagNo < (header.data.noTag-1)))
    { menuTagNo++;}
    else if(menuLevel == 0)
    { menuTagNo = 0;}
    else if(menuLevel < 3)
    { menuLevel++;}
    else
    { menuLevel = 1;}
  }
  if(buttonDownCheck())
  {
    prevTime = time;
    if((menuLevel == 0) && (menuTagNo > 0))
    { menuTagNo--;}
    else if(menuLevel == 0)
    { menuTagNo = header.data.noTag-1;}
    else if(menuLevel > 1)
    { menuLevel--;}
    else
    { menuLevel = 3;}
  }
  if(buttonBackCheck())
  {
    prevTime = time;
    if(menuLevel > 0)
    {
      menuLevel = 0;
      prevMenuLevel = -1; //resets the menu level and tag number displayed
to refresh the display
      prevTagNo = -1;
    }
  }
  if(buttonEnterCheck())
  {
    prevTime = time;
    if(menuLevel == 0)
    { menuLevel = 1;}
  }
}

//button debounce and non-repeat functions
boolean buttonUpCheck()
{
  static boolean prevUpVal;
```

```
  boolean curUpVal = digitalRead(buttonUp);
  if(curUpVal!=prevUpVal)
  {
    prevUpVal = curUpVal;
    if(curUpVal)
    { return 1;}
    else
    { return 0;}
  }
  else
  { return 0;}
}

boolean buttonDownCheck()
{
  static boolean prevDownVal;
  boolean curDownVal = digitalRead(buttonDown);
  if(curDownVal!=prevDownVal)
  {
    prevDownVal = curDownVal;
    if(curDownVal)
    { return 1;}
    else
    { return 0;}
  }
  else
  { return 0;}
}

boolean buttonEnterCheck()
{
  static boolean prevEnterVal;
  boolean curEnterVal = digitalRead(buttonEnter);
  if(curEnterVal!=prevEnterVal)
  {
    prevEnterVal = curEnterVal;
    if(curEnterVal)
    { return 1;}
    else
    { return 0;}
  }
  else
  { return 0;}
}

boolean buttonBackCheck()
{
  static boolean prevBackVal;
  boolean curBackVal = digitalRead(buttonBack);
  if(curBackVal!=prevBackVal)
```

```
  {
    prevBackVal = curBackVal;
    if(curBackVal)
    { return 1;}
    else
    { return 0;}
  }
  else
  { return 0;}
}

void buttonRedCheck()
{
  if(digitalRead(buttonRed))
  {
    openDoor();
  }
}
```

DB

```
#include <EXROM.h>    //Necessary for the full functionality of the database

/*
Database Functions:

Entry Specific:
int     entryPointer (int entryID)                  - returns the pointer in
EEPROM of an entry at entryID
void    writeEntry   (byte entryW[], int entryID)   - writes entryW at
entryID
boolean readEntry    (int entryID)                  - reads an entry at
entryID and returns true if an entry is present
boolean compEntry    (byte entryA[], byte entryB[]) - compares entryA and
entryB and returns true if the same
int     findEntry    (byte entryF[])                - finds entryF in
database and returns entryID
boolean addEntry     (byte entryA[])                - adds entryA to
database
void    delEntry     (int entryID)                  - deletes an entry at
entryID and defrags database

Header Specific:
void    writeHead    () - writes a new Header
boolean readHead     () - reads a header, returns true if present, false if
not
boolean updateHead   () - updates the header, returns false if no header
present, true if update successful
boolean createHead   () - creates a header, returns false if one already
```

```
present, true if one created
*/

/************************************************************
These 4 definitions can be changed as needed.

HEADER_P is the pointer for where the header should be placed
in the eeprom

HBYTE is the identifying byte that is used for identifying the
header in the EEPROM - to completely wipe a currently saved
database this number is all that needs to be changed and
uploaded.

EBYTE is the identifying byte that is used for identifying
entries in the EEPROM. Changing this without first wiping the
database will result in loosing entries - especially if data
is already stored within the EEPROM.

MAXMEM is the maximum number of bytes that the database can
take up - keep this equal to or less than the maximum EEPROM
memory of the Arduino.
************************************************************/

#define HEADER_P 0
#define HBYTE 250
#define EBYTE 251
#define MAXMEM 512

/************************************************************
               DO NOT TOUCH - Database Header

This section holds the header structure for the database,
allowing the Arduino to know the size and shape of the database.

************************************************************/

/*union Header
{
  byte raw[3];
  struct Data
  {
    byte hByte;
    int noTag;
  }
  data;
}
header;*/

/************************************************************
```

```
                  END OF DO NOT TOUCH AREA
************************************************************/

union Entry
{
  byte raw[15]; // must be the same size as the Data struct.
  //remember that ints are 2 bytes, floats, longs etc. are 4.
  struct Data
  {
    //Needed for future functionality - rebuilding database
    byte eByte;
    union EData
    {
      byte raw[14]; //must be same size as the PData Struct.
      struct PData
      {
        //Customize this bit!
        byte rfidTag[10];
        char rfidName[4];
        //End of custom area
      }
      pData;
    }
    eData;
  }
  data;
}
entry;

union PData //used for creating new entries
{
  byte raw[14]; //must be same size as the PData Struct above.
  struct Data
  {
    //Customize this bit!
    byte rfidTag[10];
    char rfidName[4];
    //End of custom area
  } data;
}
pData;

/*-----------------------------------------------------
 Entry
 -----------------------------------------------------*/

int entryPointer(int entryID) //returns pointer for specified entryID
{
  return HEADER_P + sizeof(header.raw) + (sizeof(entry.raw)*entryID);
}
```

```
void writeEntry(byte entryW[sizeof(entry.data.eData.raw)], int entryID)
//writes the raw data sent to it - send the whole entry as one array
{
  entry.data.eByte = EBYTE;
  for(int i=0;i<sizeof(entry.data.eData.raw);i++)
  {
    entry.data.eData.raw[i]=entryW[i];
  }
  EXROM.write(entryPointer(entryID), entry.raw, sizeof(entry.raw));
  readHead();
  header.data.noTag++;
  updateHead();
}

boolean readEntry(int entryID) //reads a specified entry of entryID
{
  EXROM.read(entryPointer(entryID), &entry.data.eByte);
  if(entry.data.eByte==EBYTE) //check to see if entry is there
  {
    EXROM.read(entryPointer(entryID), entry.raw, sizeof(entry.raw));
    return true;
  }
  else
    return false;
}

//compares two entries and returns true if identical. useful for searching
boolean compEntry(byte entryA[sizeof(entry.data.eData.raw)], byte
entryB[sizeof(entry.data.eData.raw)])
{
  boolean flag = false;
  for(int i=0;i<sizeof(entry.data.eData.raw);i++)
  {
    if (entryA[i]==entryB[i])
      flag = true;
    else
    {
      flag = false;
      break;
    }
  }
  return flag;
}

//finds an entry in the database, returns entryID if found, -1 if not.
int findEntry(byte entryF[sizeof(entry.data.eData.raw)])
{
  int flag = -1;
  readHead();
  for(int i=0;i<header.data.noTag;i++)
```

```
  {
    readEntry(i);
    if (compEntry(entryF, entry.data.eData.raw))
    {
      flag = i;
      break;
    }
  }
  return flag;
}

// adds an entry to database. returns 1 if successful, 0 if tag already
present, and -1 if database is full
boolean addEntry(byte entryA[sizeof(entry.data.eData.raw)])
{
  int presEntry = findEntry(entryA);
  if(presEntry==-1 && (entryPointer(header.data.noTag) + sizeof(entry.raw))
<= MAXMEM)
  {
    readHead();
    for(int i=0; i <= header.data.noTag; i++)
    {
      if(!readEntry(i)) //if no entry present, write an entry there
      {
        writeEntry(entryA, i);
        return 1;
      }
    }

  }
  else if(presEntry!=-1)
  {
    return 0; // tag already in database
  }
  else
  {
    return -1; // database full
  }
}

//deletes an entry at entryID (fills with 255), and defragments database to
stop having large gaps
void delEntry(int entryID)
{
  readHead();
  byte eEntry[sizeof(entry.raw)]; //create empty entry
  for(int i=0;i<sizeof(eEntry);i++)
  {
    eEntry[i] = 255;
  }
  if(entryID==header.data.noTag) //if entry to be deleted is at end of
```

```
database, just delete
  {
    EXROM.write(entryPointer(entryID), eEntry, sizeof(eEntry));
  }
  else //if entry is not last entry, read the last entry and copy it to
current position, then delete last entry
  {
    readEntry(header.data.noTag-1);
    EXROM.write(entryPointer(entryID), entry.raw, sizeof(entry.raw));
    EXROM.write(entryPointer(header.data.noTag-1), eEntry, sizeof(eEntry));
  }
  header.data.noTag--;
  updateHead();
}

/*------------------------------------------------------
 Header
 ------------------------------------------------------*/

//Writes a new header
void writeHead()
{
  header.data.hByte=HBYTE;
  header.data.noTag=0;
  EXROM.write(HEADER_P, header.raw, sizeof(header.raw));
}

//Reads a header if there is one present at HEADER_P. returns true if found,
false if not.
boolean readHead()
{
  EXROM.read(HEADER_P, &header.data.hByte);
  if(header.data.hByte==HBYTE)
  {
    EXROM.read(HEADER_P, header.raw, sizeof(header.raw));
    return true;
  }
  else
    return false;
}

/*Updates the header if there is one present, with whatever noTag is set.
use readHead() first to get an accurate update, for example:

readHead();
header.data.noTag++;
updateHead();

This will update the header with an increment of one in noTag.*/
boolean updateHead()
```

```
{
  byte hCheck;
  EXROM.read(HEADER_P, &hCheck);
  if(hCheck==HBYTE)
  {
    EXROM.write(HEADER_P, header.raw, sizeof(header.raw));
    return true;
  }
  else
    return false;
}

//Creates a header if there isnt one present. returns false if one present,
true if one is created.
boolean createHead()
{
  if(readHead())
  {
    return false;
  }
  else
  {
    writeHead();
    return true;
  }
}
```

LCD

```
//Extra LCD Functions

void lcdClearSpace(int col, int row, int length) //clear a specific space on
the LCD screen. goes to lcd.home at end.
{
  lcd.setCursor(col,row);
  for(int i=0;i<length;i++)
  {
    lcd.print(" ");
  }
  lcd.home();
}

void lcdBacklightTimer()
{
  if(time - prevTime < lcdBackOnTime)
  {
    digitalWrite(lcdBacklight,HIGH);
  }
  else
```

```
  {
    digitalWrite(lcdBacklight,LOW);
  }
}
```

Menu

```
void menu()
{
  if(menuLevel==0)
  {
    baseMenu();
  }
  else
  {
    mainMenu();
  }
}

void baseMenu()
{
  if(menuTagNo!=prevTagNo)
  {
    lcd.clear();
    lcd.print("RFID Tag    Name");
    lcd.setCursor(9,0);
    lcd.print(menuTagNo,DEC);
    lcd.setCursor(0,1);
    readHead();
    if(readEntry(menuTagNo))
    {
      printTag();
      lcd.print("  ");
      printName();
    }
    else
    {
      lcd.print("No Tag Present");
    }
    prevTagNo = menuTagNo;
  }
}

void mainMenu()
{
  if(menuLevel!=prevMenuLevel)
  {
    prevMenuLevel = menuLevel;
```

```
    if(menuLevel == 1)
    {
      lcdClearSpace(0,0,16);
      lcd.print("< Add New Tag  >");
    }
    else if(menuLevel == 2)
    {
      lcdClearSpace(0,0,16);
      lcd.print("< Del Cur Tag  >");
    }
    else if(menuLevel == 3)
    {
      lcdClearSpace(0,0,16);
      lcd.print("< Change Name  >");
    }
  }
  if(buttonEnterCheck())
  {
    if(menuLevel == 1)
    {
      addRFID();
    }
    else if(menuLevel == 2)
    {
      delMenu();
    }
    else if(menuLevel == 3)
    {
      changeName();
    }
  }
}

void delMenu()
{
  lcdClearSpace(0,0,16);
  lcd.print("Delete?  No  Yes");
  int curState = 0;
  lcd.setCursor(8,0);
  lcd.blink();
  while(1)
  {
    if(buttonBackCheck())
    {
      lcd.noBlink();
      menuLevel = 0;
      prevMenuLevel = -1; //resets the menu level and tag number displayed
to refresh the display
      prevTagNo = -1;
      return;
    }
```

```
    if(buttonEnterCheck())
    {
      if(curState == 0)
      {
        lcd.noBlink();
        menuLevel = 0;
        prevMenuLevel = -1; //resets the menu level and tag number displayed
to refresh the display
        prevTagNo = -1;
        return;
      }
      if(curState == 1)
      {
        delEntry(menuTagNo);
        lcd.noBlink();
        lcd.home();
        lcd.print("Tag Deleted");
        menuLevel = 0;
        menuTagNo = 0;
        prevMenuLevel = -1; //resets the menu level and tag number displayed
to refresh the display
        prevTagNo = -1;
        delay(2000);
        lcd.clear();
        return;
      }
    }
    if(buttonUpCheck())
    {
      if(curState == 0)
      { curState = 1;}
      else if(curState == 1)
      { curState = 0;}
      lcd.setCursor(8+(4*curState),0);
    }
    if(buttonDownCheck())
    {
      if(curState == 0)
      { curState = 1;}
      else if(curState == 1)
      { curState = 0;}
      lcd.setCursor(8+(4*curState),0);
    }
  }
}
```

Name

```
boolean addName()
{
  lcdClearSpace(0,0,16);
  lcd.print("Enter Name");
  lcd.setCursor(12,1);
  lcd.blink();
  int namePos = 0;
  for(int i=0;i<4;i++)
  { newName[i] = 32;}
  while(1)
  {
    if(buttonBackCheck())
    {
      if(namePos == 0)
      {
        lcd.noBlink();
        return false;
      }
      else
      {
        namePos--;
        lcd.setCursor(12+namePos,1);
      }
    }
    if(buttonEnterCheck())
    {
      if(namePos == 3)
      {
        lcd.noBlink();
        return true;
      }
      else
      {
        namePos++;
        lcd.setCursor(12+namePos,1);
      }
    }
    if(buttonUpCheck())
    {
      if(newName[namePos]==32)
      {
        newName[namePos] = 65;
      }
      else if(newName[namePos]==90)
      {
        newName[namePos] = 48;
      }
      else if(newName[namePos]==57)
```

```
      {
        newName[namePos] = 32;
      }
      else
      {
        newName[namePos]++;
      }
      lcd.setCursor(12,1);
      lcd.print(newName);
      lcd.setCursor(12+namePos,1);
    }
    if(buttonDownCheck())
    {
      if(newName[namePos]==32)
      {
        newName[namePos] = 57;
      }
      else if(newName[namePos]==48)
      {
        newName[namePos] = 90;
      }
      else if(newName[namePos]==65)
      {
        newName[namePos] = 32;
      }
      else
      {
        newName[namePos]--;
      }
      lcd.setCursor(12,1);
      lcd.print(newName);
      lcd.setCursor(12+namePos,1);
    }
  }
}

void printName()
{
  for(int i=0; i < 4; i++)
  {
    lcd.print(entry.data.eData.pData.rfidName[i]);
  }
}

boolean changeName()
{
  lcdClearSpace(0,0,16);
  lcd.print("Enter Name");
  lcd.setCursor(12,1);
  lcd.blink();
```

```
  int namePos = 0;
  while(1)
  {
    if(buttonBackCheck())
    {
      if(namePos == 0)
      {
        lcd.noBlink();
        return false;
      }
      else
      {
        namePos--;
        lcd.setCursor(12+namePos,1);
      }
    }
    if(buttonEnterCheck())
    {
      if(namePos == 3)
      {
        lcd.noBlink();
        EXROM.write(entryPointer(menuTagNo), entry.raw, sizeof(entry.raw));
        menuLevel = 0;
        prevMenuLevel = -1; //resets the menu level and tag number displayed
to refresh the display
        prevTagNo = -1;
        return true;
      }
      else
      {
        namePos++;
        lcd.setCursor(12+namePos,1);
      }
    }
    if(buttonUpCheck())
    {
      if(entry.data.eData.pData.rfidName[namePos]==32)
      {
        entry.data.eData.pData.rfidName[namePos] = 65;
      }
      else if(entry.data.eData.pData.rfidName[namePos]==90)
      {
        entry.data.eData.pData.rfidName[namePos] = 48;
      }
      else if(entry.data.eData.pData.rfidName[namePos]==57)
      {
        entry.data.eData.pData.rfidName[namePos] = 32;
      }
      else
      {
        entry.data.eData.pData.rfidName[namePos]++;
```

```
      }
      lcd.setCursor(12,1);
      lcd.print(entry.data.eData.pData.rfidName);
      lcd.setCursor(12+namePos,1);
    }
    if(buttonDownCheck())
    {
      if(entry.data.eData.pData.rfidName[namePos]==32)
      {
        entry.data.eData.pData.rfidName[namePos] = 57;
      }
      else if(entry.data.eData.pData.rfidName[namePos]==48)
      {
        entry.data.eData.pData.rfidName[namePos] = 90;
      }
      else if(entry.data.eData.pData.rfidName[namePos]==65)
      {
        entry.data.eData.pData.rfidName[namePos] = 32;
      }
      else
      {
        entry.data.eData.pData.rfidName[namePos]--;
      }
      lcd.setCursor(12,1);
      lcd.print(entry.data.eData.pData.rfidName);
      lcd.setCursor(12+namePos,1);
    }
  }
}
```

Tag

```
union tagData //union for the ID-12 Innovations RFID reader data
{
  byte tagin[16];
  struct tag
  {
    byte stx;
    byte data[10];
    byte cs[2];
    byte cr;
    byte lf;
    byte etx;
  }
  tag;
}
tagData;
```

```
boolean readRFID() //read an RFID tag from Serial if theres one available,
returns true if one read, false if not.
{
  if(rfid.available())
  {
    delay(100);
    for(int i=0; i < sizeof(tagData.tagin); i++)
    {
      tagData.tagin[i] = rfid.read();
    }
    rfid.flush();
    return true;
  }
  else
  { return false;}
}

void addRFID()
{
  lcd.clear();
  rfid.flush();
  lcd.print("Scan RFID to Add");
  int find = -1;
  while(1)
  {
    if(buttonBackCheck())
    {
      menuLevel = 0;
      prevMenuLevel = -1; //resets the menu level and tag number displayed
to refresh the display
      prevTagNo = -1;
      return;
    }
    if(readRFID() && (find=findRFID(tagData.tag.data))==-1)
    {
      lcd.setCursor(0,1);
      printRFID();
      if(addName())
      {
        for(int j=0;j<4;j++)
        { pData.data.rfidName[j]=newName[j];}
        for(int i=0;i<10;i++)
        { pData.data.rfidTag[i]=tagData.tag.data[i];}
        addEntry(pData.raw);
        lcd.home();
        lcd.print("Tag Added");
        delay(2000);
      }
      menuLevel = 0;
      menuTagNo = 0;
      prevMenuLevel = -1; //resets the menu level and tag number displayed
```

```
to refresh the display
      prevTagNo = -1;
      return;
    }
    if(find > -1)
    {
      lcd.clear();
      lcd.print("Tag In Database");
      delay(2000);
      menuLevel = 0;
      prevMenuLevel = -1; //resets the menu level and tag number displayed
to refresh the display
      prevTagNo = -1;
      return;
    }
  }
}

void printRFID()
{
  for(int i=0; i < sizeof(tagData.tag.data);i++)
  {
    lcd.print(tagData.tag.data[i]);
  }
}

void printTag()
{
  for(int i=0; i < 10;i++)
  {
    lcd.print(entry.data.eData.pData.rfidTag[i]);
  }
}

int findRFID(byte fTag[10])
{
  int flag = -1;
  readHead();
  for(int i=0;i<header.data.noTag;i++)
  {
    readEntry(i);
    if (compRFID(fTag, entry.data.eData.pData.rfidTag))
    {
      flag = i;
      break;
    }
  }
  return flag;
}
```

```
boolean compRFID(byte tagA[10], byte tagB[10])
{
  boolean flag = false;
  for(int i=0;i<10;i++)
  {
    if (tagA[i]==tagB[i])
      flag = true;
    else
    {
      flag = false;
      break;
    }
  }
  return flag;
}

void checkRFID()
{
  if(readRFID())
  {
    int tagNo = findRFID(tagData.tag.data);
    if(tagNo != -1)
    {
      menuTagNo = tagNo;
      readEntry(tagNo);
      lcd.setCursor(0,1);
      printTag();
      lcd.print("  ");
      printName();
      lcdClearSpace(0,0,16);
      lcd.print("Welcome In!");
      openDoor();
      menuLevel = 0;
      prevMenuLevel = -1; //resets the menu level and tag number displayed
to refresh the display
      prevTagNo = -1;
    }
  }
}
```

Category:Madlab projects