Coding of Arduino

The code would take in this Json as shown as input {“lift1”:[3,2,1],”lift2”:[4,5],” lift3”:[2]} where the order of execution is reverse and all the three motors on the circuit would drive in perfect harmony to execute the request.

```
#include <ArduinoJson.h>
#include <SimpleStack.h>
#include <AceSorting.h>

using ace_sorting::insertionSort;

#define TIME_PER_FLOOR_3 1200
#define TIME_PER_FLOOR_2 900
#define TIME_PER_FLOOR_1 800
#define DOWN 50
#define TIME_WAITING_PER_FLOOR 3000
//Ace Sorting - for using sorting function.

//All these varibale as input from JSON
bool flag;
SimpleStack<int> lift1(5), lift2(5), lift3(5);

//All these data structures to keep a track of actual operation being shown
int curr1, curr2, curr3;
struct delayDirection {
  int delay, dir, liftNum, toFloor;
  void clear() {
    liftNum = toFloor = 0;
    delay = dir = -1;
  }
} delaysAndDirections[3];

//Shaft 1 - 10,11
//Shaft 2 - 5,6
//Shaft 3 - 8,9


//Here n is the number of lift. 1 or 2 or 3
void moveLiftUp(int n) {
  switch (n) {
    case 1:
      {
        digitalWrite(10, HIGH);
        digitalWrite(11, LOW);
      }
      break;
```

```
    case 2:
     {
      digitalWrite(5, HIGH);
      digitalWrite(6, LOW);
     }
     break;
    case 3:
     {
      digitalWrite(8, HIGH);
      digitalWrite(9, LOW);
     }
     break;
  }
}
void moveLiftDown(int n) {
  switch (n) {
    case 1:
     {
      digitalWrite(10, LOW);
      digitalWrite(11, HIGH);
     }
     break;
    case 2:
     {
      digitalWrite(5, LOW);
      digitalWrite(6, HIGH);
     }
     break;
    case 3:
     {
      digitalWrite(8, LOW);
      digitalWrite(9, HIGH);
     }
     break;
  }
}
void printCurrentFloors() {

Serial.println("================================================
============================");
  Serial.println("\t\tLift1\tLift2\tLift3");

Serial.println("================================================
============================");
  Serial.print("Current Floors\t");
  Serial.print(curr1);
  Serial.print("\t");
```

```
  Serial.print(curr2);
  Serial.print("\t");
  Serial.print(curr3);
  Serial.println("\t");

Serial.println("==========================================
==========================");
}
void stopLift(int n) {
  Serial.print("Stopping Lift ");
  Serial.println(n);
  switch (n) {
    case 1:
      {
        digitalWrite(10, LOW);
        digitalWrite(11, LOW);
      }
      break;
    case 2:
      {
        digitalWrite(5, LOW);
        digitalWrite(6, LOW);
      }
      break;
    case 3:
      {
        digitalWrite(8, LOW);
        digitalWrite(9, LOW);
      }
      break;
  }
}
void moveLifts(int liftNum, int dir) {

  Serial.print("Moving Lift ");
  Serial.print(liftNum);

  if (dir == 1)  //1 for upward movement
  {
    Serial.println(" Up");
    moveLiftUp(liftNum);
  } else if (dir == -1) {
    Serial.println(" Down");
    moveLiftDown(liftNum);
  }
}
void stopAllAtOnce() {
```

```
  for (int i = 3; i < 10; ++i) {
    if (i != 7)
      digitalWrite(i, LOW);
  }
}

void resetDelayAndDirections() {
  for (int i = 0; i < 3; ++i) {
    delaysAndDirections[i].clear();
  }
}

void printOpenClose() {
  Serial.println("!! GATE OPENING !!");
  delay(TIME_WAITING_PER_FLOOR);
  Serial.println("!! GATE CLOSING !!");
}

void setup() {
  for (int i = 0; i < 12; ++i) {
    if (i != 7) {
      pinMode(i, OUTPUT);
    }
  }
  pinMode(A0, OUTPUT);
  pinMode(A1, OUTPUT);
  flag = 0;
  curr1 = curr2 = curr3 = 1;
  resetDelayAndDirections();
  Serial.begin(9600);
}

void loop() {
  Serial.begin(9600);
  Serial.setTimeout(5000);
  if (!flag) {

Serial.println("==============================================
============================");
    Serial.println("Enter the next schedule in JSON format: ");
    flag = !flag;
    Serial.flush();
  }
  if (Serial.available() > 0) {
    DynamicJsonDocument doc(1024);
    DeserializationError error = deserializeJson(doc, Serial);
```

```
  if (error) {
    Serial.print(F("deserializeJson() failed: "));
    Serial.println(error.f_str());
    // return;
  }

  else {
    //We are assuming that we are getting data in the order of execution only.
    JsonArray jsonLift1 = doc["lift1"];
    JsonArray jsonLift2 = doc["lift2"];
    JsonArray jsonLift3 = doc["lift3"];

    for (int i = 0; i < jsonLift1.size(); ++i) lift1.push(jsonLift1[i].as<int>());
    for (int i = 0; i < jsonLift2.size(); ++i) lift2.push(jsonLift2[i].as<int>());
    for (int i = 0; i < jsonLift3.size(); ++i) lift3.push(jsonLift3[i].as<int>());

    flag = 0;
  }
}
if (flag == 0) {
  //Trying to run a Psuedo-simultaneous function.
  while (lift1.getSize() > 0 && lift2.getSize() > 0 && lift3.getSize() > 0) {
    int* top = new int[3];

    lift1.pop(&top[0]);
    lift2.pop(&top[1]);
    lift3.pop(&top[2]);

    resetDelayAndDirections();

    delaysAndDirections[0].liftNum = 1;
    delaysAndDirections[0].toFloor = top[0];
    delaysAndDirections[0].delay = abs(top[0] - curr1);
    if (top[0] - curr1 == 0) {
      delaysAndDirections[0].dir = 0;
    } else
      delaysAndDirections[0].dir = (top[0] - curr1) > 0 ? 1 : -1;

    delaysAndDirections[1].liftNum = 2;
    delaysAndDirections[1].toFloor = top[1];
    delaysAndDirections[1].delay = abs(top[1] - curr2);
    if (top[1] - curr2 == 0) {
      delaysAndDirections[1].dir = 0;
    } else
      delaysAndDirections[1].dir = (top[1] - curr2) > 0 ? 1 : -1;
```

```
    delaysAndDirections[2].liftNum = 3;
    delaysAndDirections[2].toFloor = top[2];
    delaysAndDirections[2].delay = abs(top[2] - curr3);
    if (top[2] - curr3 == 0) {
      delaysAndDirections[2].dir = 0;
    } else
      delaysAndDirections[2].dir = (top[2] - curr3) > 0 ? 1 : -1;

    insertionSort(delaysAndDirections, 3, [](delayDirection a, delayDirection b)
{
      return a.delay < b.delay;
    });

    for (int i = 0; i < 3; ++i) {
      if (delaysAndDirections[i].dir != 0)
        moveLifts(delaysAndDirections[i].liftNum, delaysAndDirections[i].dir);
    }

    int sum = 0;
    for (int i = 0; i < 3; ++i) {
      if(delaysAndDirections[i].dir==-1)
      {
        delay((TIME_PER_FLOOR_3-DOWN) * max((delaysAndDirections[i].delay
- sum), 0));
      }
      delay(TIME_PER_FLOOR_3 * max((delaysAndDirections[i].delay - sum),
0));
      stopLift(delaysAndDirections[i].liftNum);
      switch (delaysAndDirections[i].liftNum) {
        case 1: curr1 = delaysAndDirections[i].toFloor; break;
        case 2: curr2 = delaysAndDirections[i].toFloor; break;
        case 3: curr3 = delaysAndDirections[i].toFloor; break;
      }
      printCurrentFloors();
      sum = sum + delaysAndDirections[i].delay;
    }
    printOpenClose();

    delete[] top;
  }
  while (lift1.getSize() > 0 && lift2.getSize() > 0) {
    int* top = new int[2];

    lift1.pop(&top[0]);
    lift2.pop(&top[1]);
```

```
    resetDelayAndDirections();

    delaysAndDirections[0].liftNum = 1;
    delaysAndDirections[0].toFloor = top[0];
    delaysAndDirections[0].delay = abs(top[0] - curr1);
    if (top[0] - curr1 == 0) {
      delaysAndDirections[0].dir = 0;
    } else
      delaysAndDirections[0].dir = (top[0] - curr1) > 0 ? 1 : -1;

    delaysAndDirections[1].liftNum = 2;
    delaysAndDirections[1].toFloor = top[1];
    delaysAndDirections[1].delay = abs(top[1] - curr2);
    if (top[1] - curr2 == 0) {
      delaysAndDirections[1].dir = 0;
    } else
      delaysAndDirections[1].dir = (top[1] - curr2) > 0 ? 1 : -1;

    insertionSort(delaysAndDirections, 2, [](delayDirection a, delayDirection b)
{
      return a.delay < b.delay;
    });

    for (int i = 0; i < 2; ++i) {
      if (delaysAndDirections[i].dir != 0)
        moveLifts(delaysAndDirections[i].liftNum, delaysAndDirections[i].dir);
    }

    int sum = 0;
    for (int i = 0; i < 2; ++i) {
      if(delaysAndDirections[i].dir==-1)
      {
        delay((TIME_PER_FLOOR_2-DOWN) * max((delaysAndDirections[i].delay
- sum), 0));
      }
      delay(TIME_PER_FLOOR_2 * max((delaysAndDirections[i].delay - sum),
0));
      stopLift(delaysAndDirections[i].liftNum);
      switch (delaysAndDirections[i].liftNum) {
        case 1: curr1 = delaysAndDirections[i].toFloor; break;
        case 2:
          curr2 = delaysAndDirections[i].toFloor;
          break;
          // case 3: curr3 = delaysAndDirections[i].toFloor;break;
      }
      printCurrentFloors();
```

```
      sum = sum + delaysAndDirections[i].delay;
    }

    printOpenClose();
    delete[] top;
   }
   while (lift1.getSize() > 0 && lift3.getSize() > 0) {
    int* top = new int[2];

    lift1.pop(&top[0]);
    lift3.pop(&top[1]);

    resetDelayAndDirections();

    delaysAndDirections[0].liftNum = 1;
    delaysAndDirections[0].toFloor = top[0];
    delaysAndDirections[0].delay = abs(top[0] - curr1);
    if (top[0] - curr1 == 0) {
     delaysAndDirections[0].dir = 0;
    } else
     delaysAndDirections[0].dir = (top[0] - curr1) > 0 ? 1 : -1;

    delaysAndDirections[1].liftNum = 3;
    delaysAndDirections[1].toFloor = top[1];
    delaysAndDirections[1].delay = abs(top[1] - curr3);
    if (top[1] - curr3 == 0) {
     delaysAndDirections[1].dir = 0;
    } else
     delaysAndDirections[1].dir = (top[1] - curr3) > 0 ? 1 : -1;

    insertionSort(delaysAndDirections, 2, [](delayDirection a, delayDirection b)
{
     return a.delay < b.delay;
    });

    for (int i = 0; i < 2; ++i) {
     if (delaysAndDirections[i].dir != 0)
      moveLifts(delaysAndDirections[i].liftNum, delaysAndDirections[i].dir);
    }

    int sum = 0;
    for (int i = 0; i < 2; ++i) {
     if(delaysAndDirections[i].dir==-1)
     {
      delay((TIME_PER_FLOOR_2-DOWN) * max((delaysAndDirections[i].delay
- sum), 0));
     }
```

```
      delay(TIME_PER_FLOOR_2 * max((delaysAndDirections[i].delay - sum),
0));
      stopLift(delaysAndDirections[i].liftNum);
      switch (delaysAndDirections[i].liftNum) {
        case 1: curr1 = delaysAndDirections[i].toFloor; break;
        // case 2: curr2 = delaysAndDirections[i].toFloor;break;
        case 3: curr3 = delaysAndDirections[i].toFloor; break;
      }
      printCurrentFloors();
      sum = sum + delaysAndDirections[i].delay;
    }
    printOpenClose();
    delete[] top;
  }
  while (lift2.getSize() > 0 && lift3.getSize() > 0) {
    int* top = new int[2];

    lift2.pop(&top[0]);
    lift3.pop(&top[1]);

    resetDelayAndDirections();

    delaysAndDirections[0].liftNum = 2;
    delaysAndDirections[0].toFloor = top[0];
    delaysAndDirections[0].delay = abs(top[0] - curr2);
    if (top[0] - curr2 == 0) {
      delaysAndDirections[0].dir = 0;
    } else
      delaysAndDirections[0].dir = (top[0] - curr2) > 0 ? 1 : -1;

    delaysAndDirections[1].liftNum = 3;
    delaysAndDirections[1].toFloor = top[1];
    delaysAndDirections[1].delay = abs(top[1] - curr3);
    if (top[1] - curr3 == 0) {
      delaysAndDirections[1].dir = 0;
    } else
      delaysAndDirections[1].dir = (top[1] - curr3) > 0 ? 1 : -1;

    insertionSort(delaysAndDirections, 2, [](delayDirection a, delayDirection b)
{
      return a.delay < b.delay;
    });

    for (int i = 0; i < 2; ++i) {
      if (delaysAndDirections[i].dir != 0)
        moveLifts(delaysAndDirections[i].liftNum, delaysAndDirections[i].dir);
    }
```

```
    int sum = 0;
    for (int i = 0; i < 2; ++i) {
      if(delaysAndDirections[i].dir==-1)
      {
        delay((TIME_PER_FLOOR_2-DOWN) * max((delaysAndDirections[i].delay
- sum), 0));
      }
      delay(TIME_PER_FLOOR_2 * max((delaysAndDirections[i].delay - sum),
0));
      stopLift(delaysAndDirections[i].liftNum);
      switch (delaysAndDirections[i].liftNum) {
        // case 1: curr1 = delaysAndDirections[i].toFloor;break;
        case 2: curr2 = delaysAndDirections[i].toFloor; break;
        case 3: curr3 = delaysAndDirections[i].toFloor; break;
      }
      printCurrentFloors();
      sum = sum + delaysAndDirections[i].delay;
    }
    printOpenClose();
    delete[] top;
  }
  while (lift1.getSize() > 0) {
    int top;
    lift1.pop(&top);
    if (top - curr1 > 0) {
      moveLifts(1, 1);
      delay(TIME_PER_FLOOR_1 * (top - curr1));
      stopLift(1);
      curr1 = top;
    } else if (top - curr1 < 0) {
      moveLifts(1, -1);
      delay((TIME_PER_FLOOR_1-DOWN) * abs(top - curr1));
      stopLift(1);
      curr1 = top;
    } else {
      stopLift(1);
      curr1 = top;
    }
    printOpenClose();
    printCurrentFloors();
  }
  while (lift2.getSize() > 0) {
    int top;
    lift2.pop(&top);
    if (top - curr2 > 0) {
      moveLifts(2, 1);
```

```
      delay(TIME_PER_FLOOR_1 * (top - curr2));
      stopLift(2);
      curr2 = top;
    } else if (top - curr2 < 0) {
      moveLifts(2, -1);
      delay((TIME_PER_FLOOR_1-DOWN) * abs(top - curr2));
      stopLift(2);
      curr2 = top;
    } else {
      stopLift(2);
      curr2 = top;
    }
    printOpenClose();
    printCurrentFloors();
  }
  while (lift3.getSize() > 0) {
    int top;
    lift3.pop(&top);
    if (top - curr3 > 0) {
      moveLifts(3, 1);
      delay(TIME_PER_FLOOR_1 * (top - curr3));
      stopLift(3);
      curr3 = top;
    } else if (top - curr3 < 0) {
      moveLifts(3, -1);
      delay((TIME_PER_FLOOR_1-DOWN) * abs(top - curr3));
      stopLift(3);
      curr3 = top;
    } else {
      stopLift(3);
      curr3 = top;
    }
    printOpenClose();
    printCurrentFloors();
  }
  Serial.end();
 }
 delay(1000);
}
```