```
#include "log.h"
#include <fstream>
#include <iostream>
//using namespace std;

int saveInFile(std::string filename, int* tab, unsigned int tabSize){
        std::fstream file;
        file.open(filename.c_str(),std::fstream::out | std::ios::app);

        if (file.is_open()){
                for(unsigned int i=0; i < tabSize - 1;i++){
                file << tab[i] << " ";}
                file << tab[tabSize-1];
                file << std::endl;
                file.close();
        }
        else {
                std::cerr << "Unable to open the file " << filename << std::endl;
                return -1;
        }
        return 0;
}
```

```
#include "awps.h"
#include <wiringPi.h>
#include <iostream>
#include <unistd.h>     //select()
//#include <sys/select.h>
#include <sys/time.h>   //gettimeofday()
#include <string>
#include <signal.h>     //Unix signals
#include "mcp3008Reading.h"
#include "ioManager.h"
#include "log.h"
#include "PlantIO.h"
//#include <vector>
//#include "archives.h"

using namespace std;

void intHandler(int signum) {
        cout << "Caught signal " << signum << endl;
        cout << "Shutting down the AWPS software" << endl;
        initGPIO();
        exit(signum);
}

void initGPIO(){
        initLed(GLEDPIN);
        initLed(YLEDPIN);
        initLed(RLEDPIN);
}

int init() {
        //Dealing signal like ^C
        signal(SIGINT, intHandler);
        wiringPiSetup();        //Initialise the wiringPi Library functions.
        initGPIO();             //Init the GPIO used in this project
        initMCP3008();          //Init the DAC component
        return 1;
}

void checkAndSetState(PlantIO* p) {
        cout << "checking state" << endl;
        unsigned int x = readMCP3008(p->getMoistureChannel());
        cout << "Sensor value is : " << x << endl;
        if (x <= p->getMoistureLimit()) {
                p->setState(wet);
        } else if (x > p->getMoistureLimit() && x <= p->getDryLimit()) {
                p->setState(moist);
        } else if (x > p->getDryLimit()) {
                p->setState(dry);
        }
}

void waterPlant(PlantIO* p) {
        time_t current_time,beg_time;
        time(&beg_time);
        time(&current_time);
        unsigned int wateringTime = p->getWaterTime();
        activateRelay(p->getRelayPin());
        while (difftime(current_time,beg_time) < wateringTime){
                blinkSeveral(RLEDPIN,YLEDPIN,GLEDPIN);
                time(&current_time);
                }
```

```
        desactivateRelay(p->getRelayPin());
        turnOn(GLEDPIN); // Plant was poured so green light till the next check.
}

void work(PlantIO* p) {
        switch (p->getState()) {
        case wet: {
                //Light the Green Led
                turnOn(GLEDPIN);
                turnOff(YLEDPIN);
                turnOff(RLEDPIN);
                break;
        }
        case moist: {
                //Light the Yellow Led
                turnOff(GLEDPIN);
                turnOn(YLEDPIN);
                turnOff(RLEDPIN);
                break;
        }
        case dry: {
                //Light the Red Led
                turnOff(GLEDPIN);
                turnOff(YLEDPIN);
                turnOn(RLEDPIN);
                // Use the pump for WATERTIME s
                waterPlant(p);
                break;
        }
        default:
                cout << "ERROR switch reached default case " << endl;
        }
}

void log(int state, int temper) {
        //Path for the logFILE
        string filename("~/Documents/AWPS/awps.data");
        //Tab to log in a file

        int tab[3];
        struct timeval tod;
        gettimeofday(&tod, NULL);
        tab[0] = tod.tv_sec;
        tab[1] = state;
        tab[2] = temper;
        cout << "time : " << tab[0] << ", state is: " << tab[1] << ", and TÂ° is : " << tab[2]
 << " Â°C" << endl;
        //      saveInFile(filename, tab,(unsigned int)( sizeof(tab)/sizeof(*tab
) ));
}

void hibernate(int s) {
        struct timeval t;
        t.tv_sec = s;
        t.tv_usec = 0;
        select(0, NULL, NULL, NULL, &t);
}

int checkTemperature() {
        double t = readMCP3008(TEMPERATURECHANNEL);
        //cout << "temper = " << temper << endl;
        //temper = temper / 1024 * 3.3 * 10;
        return t;
}
```

```
#include "ioManager.h"
#include <wiringPi.h>
#include <ctime>

void initRelay(int pin){
        pinMode(pin, OUTPUT);
        digitalWrite(pin, LOW);
}

void activateRelay(int pin){
        digitalWrite(pin, HIGH);
}

void desactivateRelay(int pin){
        digitalWrite(pin, LOW);
}

void initLed(int pin){
        pinMode(pin, OUTPUT);
        digitalWrite(pin, LOW);
}
void turnOn(int pin){
        digitalWrite(pin, HIGH);
}

void turnOff(int pin){
        digitalWrite(pin, LOW);
}

void blink(int pin, int timer){
        time_t current_time,beg_time;
        time(&beg_time);
        time(&current_time);
        while (difftime(current_time,beg_time) < timer){
                digitalWrite (pin, HIGH) ;  // On
                delay (500) ;               // mS
                digitalWrite (pin, LOW) ;   // Off
                delay (500) ;
                time(&current_time);
        }
}

void blinkSeveral(int pin1, int pin2, int pin3){
        turnOn(pin1);
        delay(100) ;               // mS
        turnOff(pin1);
        turnOn(pin2);
        delay(100);
        turnOff(pin2);
        turnOn(pin3);
        delay(100) ;
        turnOff(pin3);
}
```

```
#include "awps.h"
#include <cstdlib>      //atoi()
#include <iostream>
static volatile int keepRunning = 1;

int main(int argc, char* argv[]) {
        int cycle(0);
        if (argc == 2) {
                cycle = atoi(argv[1]);
        } else
                cycle = 86400;

        if (init()) {
      PlantIO* basil = new PlantIO("basil",2,150,450,5,8,cycle);
        //todo Loop on all plants.
        //vector<PlantIO> plantGroup;
        //plantGroup.push_back(basil);
                while (keepRunning) {

                        checkAndSetState(basil);
                        if (basil->getState()<0)
                        std::cout << "ERROR on the state process" << std::endl;
                        int temper = checkTemperature();
                        work(basil);
                        std::cout << "name = " << basil->getName() << std::endl;
                        hibernate(basil->getCycleTime());
                }
        }

        std::cout << "End of program ! " << std::endl;
        return 0;

}
```

```
#include "PlantIO.h"
#include "ioManager.h"
#include <string>

PlantIO::PlantIO(std::string n, unsigned int rp,unsigned int ml, unsigned int dl
, unsigned int mc, unsigned int wt=8, unsigned int ct= 86400){
        name = n;
        relayPin=rp;
        moistureLimit=ml;
        dryLimit = dl;
        moistureChannel=mc;
        waterTime=wt;
        cycleTime=ct;
        state=0;
        }


std::string PlantIO::getName(){
        return name;
}
unsigned int PlantIO::getRelayPin(){
return relayPin;
}
unsigned int PlantIO::getMoistureLimit(){
return moistureLimit;
}
unsigned int PlantIO::getDryLimit(){
return dryLimit;
}
unsigned int PlantIO::getMoistureChannel(){
return moistureChannel;
}
unsigned int PlantIO::getWaterTime(){
return waterTime;
}
unsigned int PlantIO::getCycleTime(){
return cycleTime;
}
unsigned int PlantIO::getState(){
return state;
}

void PlantIO::setName(std::string n){
name = n;
}
void PlantIO::setRelayPin(unsigned int rp){
relayPin = rp;
}
void PlantIO::setMoistureLimit(unsigned int ml){
moistureLimit = ml;
}
void PlantIO::setDryLimit(unsigned int dl){
dryLimit = dl;
}
void PlantIO::setMoistureChannel(unsigned int mc){
moistureChannel = mc;
}
void PlantIO::setWaterTime(unsigned int wt){
waterTime = wt;
}
void PlantIO::setCycleTime(unsigned int ct){
cycleTime = ct;
```

```
}
void PlantIO::setState(unsigned int st){
state = st;
}

void PlantIO::initGPIO(){
initRelay(relayPin);
}
```

```
#include <wiringPi.h>
#include <mcp3004.h>
#include "mcp3008Reading.h"
using namespace std;

#define BASE 100
#define SPI_CHAN 1

int initMCP3008(){
        return mcp3004Setup(BASE,SPI_CHAN);
}

int readMCP3008(int channel){
//      wiringPiSetup();
//      mcp3004Setup(BASE, SPI_CHAN) ;
        int value = analogRead(BASE+channel);
        return value;
}
```

```
#ifndef LOG_H
#define LOG_H
#include <string>

int saveInFile(std::string, int*, unsigned int);



#endif
```

```
#ifndef AWPS_H
#define AWPS_H


#include "PlantIO.h"

//Defining the pin used and parameters -> Transfer this in a config file?
#define TEMPERATURECHANNEL 6
#define GLEDPIN 3
#define YLEDPIN 4
#define RLEDPIN 5

//Different states for the state machine
enum state {
        wet, moist, dry
};

void initGPIO();
void intHandler(int signum);
int init();
void checkAndSetState(PlantIO*);
void waterPlant(PlantIO* p);
void work(PlantIO* p);
void log(int state, int temper);
void hibernate(int s);
int checkTemperature();

#endif
```

```
#ifndef LEDMANAGER_H
#define LEDMANAGER_H
void activateRelay(int pin);
void desactivateRelay(int pin);
void initLed(int pin);
void initRelay(int pin);
void turnOn(int pin);
void turnOff(int pin);
void blink(int pin,int timer);
void blinkSeveral(int pin, int pin2, int pin3);


#endif
```

```
#ifndef PLANTIO_H
#define PLANTIO_H

#include <string>
class PlantIO{

        std::string name;
        unsigned int relayPin;
        unsigned int moistureLimit;
        unsigned int dryLimit;
        unsigned int moistureChannel;
        unsigned int waterTime;
        unsigned int cycleTime;
        unsigned int state;
public:

PlantIO(std::string,unsigned int,unsigned int, unsigned int, unsigned int, unsig
ned int, unsigned int);
std::string getName();
unsigned int getRelayPin();
unsigned int getMoistureLimit();
unsigned int getDryLimit();
unsigned int getMoistureChannel();
unsigned int getWaterTime();
unsigned int getCycleTime();
unsigned int getState();


void setName(std::string);
void setRelayPin(unsigned int);
void setMoistureLimit(unsigned int);
void setDryLimit(unsigned int);
void setMoistureChannel(unsigned int);
void setWaterTime(unsigned int);
void setCycleTime(unsigned int);
void setState(unsigned int);

void initGPIO();
};

#endif
```

```
#ifndef MCP3008READING_H
#define MCP3008READING_H

#define BASE 100
#define SPI_CHAN 1

int initMCP3008();

int readMCP3008(int channel);

#endif
```

```
CXXFLAGS=-Wall #Define CXXFLAGS to automatically add them in the command

SRC=$(wildcard *.cc)
DEPS=$(wildcard *.h)
LDFLAGS=-lwiringPi
OBJ=$(SRC:.cc=.o)

# Redefine the default command to create executable without suffix (use g++ inst
ead of gcc)
awps: $(OBJ)
	g++ $(LDFLAGS) -o $@ $^

all: awps awps.pdf

awps.pdf: $(SRC) $(DEPS) Makefile
	a2ps -o - $^ | ps2pdf - docs/$@

clean:
	rm -f *~ *.o *.bak

mrproper: clean
	rm -f awps

depend:
	makedepend $(sources)
```