

```
// MBED PROGRAM
```

```
/*-----
```

```
File Name: SRWork2.c
```

```
Author:  SRaj
```

```
Date:    08/04/2020
```

```
Modified: Name or None
```

```
copyright Fanshawe College, 2019
```

```
Description: Receive command sentence from PIC microcontroller and execute the sentence
```

```
-----*/
```

```
// Preprocessor -----
```

```
#include "mbed.h"
```

```
#include "Timer.h"
```

```
#include<string.h>
```

```
Serial pc(USBTX,USBRX,9600);
```

```
Serial fvs(p13,p14,19200);
```

```
DigitalIn pb1(p5);
```

```
DigitalIn pb2(p6);
```

```
DigitalIn pb3(p7);
```

```
DigitalIn pb4(p8);
```

```
Timer timer;
```

```
// Constants -----
```

```
#define TRUE      1
```

```
#define FALSE     0
```

```
#define HIGH      1
```

```
#define LOW       0
```

```
#define MAXSAMPLES 30
```

```
#define SIZE       4
```

```
#define BUFFERSIZE 20
```

```
#define CONTROLLER 1
```

```
#define FVS        7
```

```
#define TOKENSIZE  10
```

```
// Global Variables -----
```

```
char flag_I = FALSE;
```

```
char flag_D = FALSE;
```

```
int currentTemperature=0;
```

```
int currentPressure=0;

int currentCo2=0;

char timerCount=0;

char buf[BUFFERSIZE];

char index=0;

char sentRdy=FALSE;

char rxBuf[BUFFERSIZE];

char *ptr=rxBuf;

char *token[TOKENSIZE];

char string1[]="AVGUPD";

char string2[]="T";

char string3[]="P";

char string4[]="C";

typedef struct motor

{

    char position;

    int currentPosition;

    char pattern;

    char patternCounter;

}mtr_t;
```

```
typedef struct sensorChannel
```

```
{
```

```
    int currentSample;
```

```
    int samples[MAXSAMPLES];
```

```
    int avgSample;
```

```
    char insertAt;
```

```
    int highLimit;
```

```
    int lowLimit;
```

```
    char state;
```

```
}sensorCh_t;
```

```
typedef struct fermatationVat
```

```
{
```

```
    char address[SIZE];
```

```
    sensorCh_t temperature;
```

```
    sensorCh_t pressure;
```

```
    sensorCh_t co2;
```

```
    char indicator;
```

```
    char heater;
```

```
    mtr_t motor1;
```

```
    mtr_t motor2;
```

```
char userLimit;
```

```
char userSensor;
```

```
}FVS_t;
```

```
FVS_t FVS007;
```

```
/*--- initialize_FVS -----
```

```
Author:   SRaj
```

```
Date:    27/03/2020
```

```
Modified: Name or None
```

```
Desc:    Set the data member to initial value
```

```
Input:   Type and purpose of input arguments
```

```
Returns: Type and purpose of returning argument
```

```
-----*/
```

```
void initialize_FVS (void)
```

```
{
```

```
    int i;
```

```
    FVS007.address[0]='0';
```

```
    FVS007.address[1]='0';
```

```
FVS007.address[2]='7';

FVS007.temperature.currentSample=0;

for(i=0;i<MAXSAMPLES;i++)

{

    FVS007.temperature.samples[i]=0;

}

FVS007.temperature.avgSample=0;

FVS007.temperature.insertAt=0;

FVS007.pressure.currentSample=0;

for(i=0;i<MAXSAMPLES;i++)

{

    FVS007.pressure.samples[i]=0;

}

FVS007.pressure.avgSample=0;

FVS007.pressure.insertAt=0;

FVS007.temperature.highLimit=65;

FVS007.pressure.highLimit=17;

FVS007.temperature.lowLimit=12;

FVS007.pressure.lowLimit=103;

FVS007.pressure.state=0;

FVS007.temperature.state=0;

FVS007.indicator=FALSE;
```

```
FVS007.heater=FALSE;

FVS007.motor1.position=0;

FVS007.motor1.currentPosition=0;

FVS007.motor1.pattern=0;

FVS007.motor1.patternCounter=0;

FVS007.motor2.position=0;

FVS007.motor2.currentPosition=0;

FVS007.motor2.pattern=0;

FVS007.motor2.patternCounter=0;

FVS007.userLimit=0;

FVS007.userSensor=0;

FVS007.co2.currentSample=0;

for(i=0;i<MAXSAMPLES;i++)
{
    FVS007.co2.samples[i]=0;
}

FVS007.co2.avgSample=0;

FVS007.co2.insertAt=0;

FVS007.co2.highLimit=1200;

FVS007.co2.lowLimit=350;

} // eo initialize_FVS
```

/\*---userMode-----

Author: SRaj

Date: 27/03/2020

Modified: None

Desc: User can select the mode.

Input: None

Returns: None

-----\*/

void userMode (void)

{

if(pb1==0)

{

FVS007.userLimit=!FVS007.userLimit;

}

} // eo userMode

/\*---userChannel-----

Author: SRaj

Date: 27/03/2020

Modified: None



Desc: User can select the Channel.

Input: None

Returns: None

-----\*/

```
void userChannel(void)
```

```
{
```

```
    if(pb2==LOW)
```

```
    {
```

```
        FVS007.userSensor++;
```

```
    }
```

```
    if(FVS007.userSensor>=3)
```

```
    {
```

```
        FVS007.userSensor=0;
```

```
    }
```

```
} // eo userChannel
```

```
/*---calChecksum -----
```

Author: SRaj

Date: 27/03/2020

Modified: Name or None

Desc: Calculate the check sum of the string

Input: Pointer to the string

Returns: Return checksum value

-----\*/

```
char calChecksum(char *ptr)
```

```
{
```

```
    char cs=0;
```

```
    while(*ptr)
```

```
    {
```

```
        cs^=*ptr;
```

```
        ptr++;
```

```
    }
```

```
    return cs;
```

```
} // eo calCheckSum
```

```
/*---increment -----
```

Author: SRaj

Date: 27/03/2020

Modified: None

Desc: User can increase the limit of sensor

Input: None

Returns: None

-----\*/

```
void increment(void)
```

```
{
```

```
    if(pb3==LOW)
```

```
    {
```

```
        if (FVS007.userLimit==HIGH && FVS007.userSensor==1)
```

```
        {
```

```
            FVS007.temperature.highLimit++;
```

```
            flag_I=TRUE;
```

```
        }
```

```
    else if (FVS007.userLimit==LOW && FVS007.userSensor==1)
```

```
    {
```

```
        FVS007.temperature.lowLimit++;
```

```
        flag_I=TRUE;
```

```
    }
```

```
    else if (FVS007.userLimit==HIGH && FVS007.userSensor==0)
```

```
    {
```

```
        FVS007.pressure.highLimit++;
```

```
        flag_I=TRUE;
```

```

    }

    else if(FVS007.userLimit==LOW && FVS007.userSensor==0)

    {

        FVS007.pressure.lowLimit++;

        flag_I=TRUE;

    }

    else if(FVS007.userLimit==HIGH && FVS007.userSensor==2)

    {

        FVS007.co2.highLimit++;

        flag_I=TRUE;

    }

    else

    {

        FVS007.co2.lowLimit++;

        flag_I=TRUE;

    }

} //eo if

} //eo increment

/*---decrement -----

```

Author: SRaj

Date: 27/03/2020

Modified: None

Desc: User can decrease the limit of sensor.

Input: None

Returns: None

-----\*/

```
void decrement(void)
```

```
{
```

```
    if (pb4==0)
```

```
    {
```

```
        if (FVS007.userLimit==HIGH && FVS007.userSensor==1)
```

```
        {
```

```
            FVS007.temperature.highLimit--;
```

```
            flag_D=TRUE;
```

```
        }
```

```
    else if (FVS007.userLimit==LOW && FVS007.userSensor==1)
```

```
    {
```

```
        FVS007.temperature.lowLimit--;
```

```
        flag_D=TRUE;
```

```
    }
```

```
    else if (FVS007.userLimit==HIGH && FVS007.userSensor==0)
```

```
{  
  
    FVS007.pressure.highLimit--;  
  
    flag_D=TRUE;  
  
}  
  
else if(FVS007.userLimit==LOW && FVS007.userSensor==0)  
  
{  
  
    FVS007.pressure.lowLimit--;  
  
    flag_D=TRUE;  
  
}  
  
else if(FVS007.userLimit==HIGH && FVS007.userSensor==2)  
  
{  
  
    FVS007.co2.highLimit--;  
  
    flag_D=TRUE;  
  
}  
  
else  
  
{  
  
    FVS007.co2.lowLimit--;  
  
    flag_D=TRUE;  
  
}  
  
} //eo if  
  
} //eo decrement
```

/\*---sentFormation -----

Author: SRaj

Date: 27/03/2020

Modified: Name or None

Desc: Formation of the string with checksum if user changes the limit

Input: None

Returns: None

-----\*/

void sentFormation()

{

if(flag\_l==TRUE)

{

if(FVS007.userLimit==1 && FVS007.userSensor==1)

{

timerCount++;

printf(buf,"\$CONLIM,%i,%i,T,H,%i\0",CONTROLLER,FVS,FVS007.temperature.highLimit);

printf(buf,"%s,%i#\0",buf,calChecksum(buf));

if(timerCount<=5)

{

```

    fvs.puts(buf);

    pc.printf("\r%s",buf);

}

if(timerCount>5)

{

    pc.printf("\033[K\033[H\033[0");

    flag_l=FALSE;

    timerCount=0;

}

}

if(FVS007.userLimit==0 && FVS007.userSensor==1)

{

    timerCount++;


    sprintf(buf,"$CONLIM,%i,%i,T,L,%i\0",CONTROLLER,FVS,FVS007.temperature.lowLimit);

    sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));


    if(timerCount<=5)

    {

        fvs.puts(buf);

        pc.printf("\r%s",buf);

```



```

    }

    if(timerCount>5)

    {

        pc.printf("\033[K\033[H\033[0");

        flag_l=FALSE;

        timerCount=0;

    }

}

if(FVS007.userLimit==1 && FVS007.userSensor==0)

{

    timerCount++;

    sprintf(buf,"$CONLIM,%i,%i,P,H,%i\0",CONTROLLER,FVS,FVS007.pressure.highLimit);

    sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));


    if(timerCount<=5)

    {

        fvs.puts(buf);

        pc.printf("\r%s",buf);

    }

    if(timerCount>5)

    {

        pc.printf("\033[K\033[H\033[0");

```

```

        flag_l=FALSE;

        timerCount=0;

    }

}

if(FVS007.userLimit==0 && FVS007.userSensor==0)

{

    timerCount++;


    sprintf(buf,"$CONLIM,%i,%i,P,L,%i\0",CONTROLLER,FVS,FVS007.pressure.lowLimit);

    sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));


    if(timerCount<=5)

    {

        fvs.puts(buf);

        pc.printf("\r%s",buf);

    }

    if(timerCount>5)

    {

        pc.printf("\033[K\033[H\033[0");

        flag_l=FALSE;

        timerCount=0;

    }

```

```

}

if(FVS007.userLimit==1 && FVS007.userSensor==2)

{

    timerCount++;


    sprintf(buf,"$CONLIM,%i,%i,C,H,%i\0",CONTROLLER,FVS,FVS007.co2.highLimit);

    sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));


    if(timerCount<=5)

    {

        fvs.puts(buf);

        pc.printf("\r%s",buf);

    }

    if(timerCount>5)

    {

        pc.printf("\033[K\033[H\033[0");

        flag_l=FALSE;

        timerCount=0;

    }

}

if(FVS007.userLimit==0 && FVS007.userSensor==2)

{

```

```

timerCount++;

sprintf(buf,"$CONLIM,%i,%i,C,L,%i\0",CONTROLLER,FVS,FVS007.co2.lowLimit);

sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));

if(timerCount<=5)

{

    fvs.puts(buf);

    pc.printf("\r%s",buf);

}

if(timerCount>5)

{

    pc.printf("\033[K\033[H\033[0");

    flag_l=FALSE;

    timerCount=0;

}

}

}

if(flag_D==TRUE)

{

    if(FVS007.userLimit==1 && FVS007.userSensor==1)

```

```

{

timerCount++;


sprintf(buf,"$CONLIM,%i,%i,T,H,%i\0",CONTROLLER,FVS,FVS007.temperature.highLimit);

sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));


if(timerCount<=5)

{

fvs.puts(buf);

pc.printf("\r%s",buf);

}

if(timerCount>5)

{

pc.printf("\033[K\033[H\033[0");

flag_D=FALSE;

timerCount=0;

}

}

if(FVS007.userLimit==0 && FVS007.userSensor==1)

{

timerCount++;

```

```

    sprintf(buf,"$CONLIM,%i,%i,T,L,%i\0",CONTROLLER,FVS,FVS007.temperature.lowLimit);

    sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));

if(timerCount<=5)

{

    fvs.puts(buf);

    pc.printf("\r%s",buf);

}

if(timerCount>5)

{

    pc.printf("\033[K\033[H\033[0");

    flag_D=FALSE;

    timerCount=0;

}

}

if(FVS007.userLimit==1 && FVS007.userSensor==0)

{

    timerCount++;

    sprintf(buf,"$CONLIM,%i,%i,P,H,%i\0",CONTROLLER,FVS,FVS007.pressure.highLimit);

    sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));

```

```

if(timerCount<=5)

{

    fvs.puts(buf);

    pc.printf("\r%s",buf);


}

if(timerCount>5)

{

    pc.printf("\033[K\033[H\033[0");

    flag_D=FALSE;

    timerCount=0;

}

}

if(FVS007.userLimit==0 && FVS007.userSensor==0)

{

    timerCount++;


    sprintf(buf,"$CONLIM,%i,%i,P,L,%i\0",CONTROLLER,FVS,FVS007.pressure.lowLimit);

    sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));


    if(timerCount<=5)

```

```

{

    fvs.puts(buf);

    pc.printf("\r%s",buf);


}

if(timerCount>5)

{

    pc.printf("\033[K\033[H\033[0");

    flag_D=FALSE;

    timerCount=0;

}

}

if(FVS007.userLimit==1 && FVS007.userSensor==2)

{

    timerCount++;


    sprintf(buf,"$CONLIM,%i,%i,C,H,%i\0",CONTROLLER,FVS,FVS007.co2.highLimit);

    sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));


    if(timerCount<=5)

    {

        fvs.puts(buf);
    }
}

```



```

        pc.printf("\r%s",buf);

    }

    if(timerCount>5)

    {

        pc.printf("\033[K\033[H\033[0");

        flag_D=FALSE;

        timerCount=0;

    }

}

if(FVS007.userLimit==0 && FVS007.userSensor==2)

{

    timerCount++;


    sprintf(buf,"$CONLIM,%i,%i,C,L,%i\0",CONTROLLER,FVS,FVS007.co2.lowLimit);

    sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));


    if(timerCount<=5)

    {

        fvs.puts(buf);

        pc.printf("\r%s",buf);

```

```

    }

    if(timerCount>5)

    {

        pc.printf("\033[K\033[H\033[0");

        flag_D=FALSE;

        timerCount=0;

    }

}

}

} //eo sentFormation

```

```

/*--- display -----

```

Author: SRaj

Date: 27/03/2020

Modified: Name or None

Desc: Display the output to terminal window

Input: Type and purpose of input arguments

Returns: Type and purpose of returning argument

```

-----*/

```

```

void display (void)

```

```

{

```

```
pc.printf("\033[2J\033[H\033[0");

pc.printf("\r MMBED System Properties\n");

pc.printf("\r\n");

if (FVS007.userLimit==1 && FVS007.userSensor==1)

{

    pc.printf("\rMODE: HIGH \t\t CHANNEL: TEMPERATURE \n ");

}


if (FVS007.userLimit==0 && FVS007.userSensor==1)

{

    pc.printf("\rMODE: LOW \t\t CHANNEL: TEMPERATURE\n");

}


if (FVS007.userLimit==1 && FVS007.userSensor==0)

{

    pc.printf("\rMODE: HIGH \t\t CHANNEL: PRESSURE\n");

}


if (FVS007.userLimit==0 && FVS007.userSensor==0)

{

    pc.printf("\r MODE: LOW \t\t CHANNEL: PRESSURE\n");

}
```

```

if(FVS007.userLimit==1 && FVS007.userSensor==2)

{

    pc.printf("\rMODE: HIGH \t\t CHANNEL: CO2\n");

}

if(FVS007.userLimit==0 && FVS007.userSensor==2)

{

    pc.printf("\rMODE: LOW \t\t CHANNEL: CO2\n");

}

pc.printf("\n\rTemperature\t\t\tPressure\t\tCO2\n");


pc.printf("\rCurrent:\t%i%cC\t\tCurrent:  %i kPa \tCurrent:  %d ppm\n",
currentTemperature,248,currentPressure,currentCo2);


pc.printf("\rHighLimit:\t%i%cC\t\tHighLimit: %i kPa\tHighLimit:  %i ppm\n",
FVS007.temperature.highLimit,248,FVS007.pressure.highLimit,FVS007.co2.highLimit);    // for printing
high limit of all the sensors


pc.printf("\rLowLimit:\t%i%cC\t\tLowLimit: %i kPa\tLowLimit:  %i ppm\n",
FVS007.temperature.lowLimit,248,FVS007.pressure.lowLimit,FVS007.co2.lowLimit);    // for printing
low limit of both the sesnsors


if(((currentTemperature>FVS007.temperature.highLimit) || (currentTemperature<FVS007.temperature.l
owLimit)) &&
((currentPressure<FVS007.pressure.highLimit) || (currentPressure>FVS007.pressure.lowLimit)) &&
((currentCo2>FVS007.co2.highLimit) || (currentCo2<FVS007.co2.lowLimit)))

```

```

{

    pc.printf("\r\033[0;37mStatus : \033[0;31mAlarm\t\t\t \033[0;37mStatus:
\033[0;31mAlarm\033[0;37m\t\t \033[0;37mStatus: \033[0;31mAlarm\033[0;37m\n ");    // for
printing temperature and pressure status

}

else
if(((currentTemperature>FVS007.temperature.highLimit) || (currentTemperature<FVS007.temperature.l
owLimit)) &&
((currentPressure>FVS007.pressure.highLimit) || (currentPressure<FVS007.pressure.lowLimit)) &&
((currentCo2>FVS007.co2.highLimit) || (currentCo2<FVS007.co2.lowLimit)))

{

    pc.printf("\r\033[0;37mStatus : \033[0;31mAlarm\t\t\t \033[0;37mStatus:
\033[0;32mSafe\033[0;37m\t\t \033[0;37mStatus: \033[0;31mAlarm\033[0;37m\n");

}

else
if(((currentTemperature<FVS007.temperature.highLimit) || (currentTemperature>FVS007.temperature.l
owLimit)) &&
((currentPressure<FVS007.pressure.highLimit) || (currentPressure>FVS007.pressure.lowLimit)) &&
((currentCo2>FVS007.co2.highLimit) || (currentCo2<FVS007.co2.lowLimit)))

{

    pc.printf("\r\033[0;37mStatus : \033[0;32mSafe\t\t\t \033[0;37mStatus:
\033[0;31mAlarm\033[0;37m\t\t \033[0;37mStatus: \033[0;31mAlarm\033[0;37m\n");

}

else if ((currentTemperature<FVS007.temperature.highLimit &&
currentTemperature>FVS007.temperature.lowLimit) && (currentPressure>FVS007.pressure.highLimit
&& currentPressure<FVS007.pressure.lowLimit) && ((currentCo2>FVS007.co2.highLimit) ||
(currentCo2<FVS007.co2.lowLimit)))

{

    pc.printf("\r\033[0;37mStatus : \033[0;32mSafe\t\t\t \033[0;37mStatus:

```

```
\033[0;32mSafe\033[0;37m\t\t \033[0;37mStatus: \033[0;31mAlarm\033[0;37m\n");
```

```
}
```

```
    else if(((currentTemperature>FVS007.temperature.highLimit) ||  
(currentTemperature<FVS007.temperature.lowLimit)) &&  
((currentPressure>FVS007.pressure.lowLimit) || (currentPressure<FVS007.pressure.highLimit)) &&  
(currentCo2>FVS007.co2.lowLimit && currentCo2<FVS007.co2.highLimit))
```

```
{
```

```
    pc.printf("\r\033[0;37mStatus : \033[0;31mAlarm\t\t\t \033[0;37mStatus:  
\033[0;31mAlarm\033[0;37m\t\t \033[0;37mStatus: \033[0;32mSafe\033[0;37m\n");
```

```
}
```

```
    else
```

```
if(((currentTemperature>FVS007.temperature.highLimit) || (currentTemperature<FVS007.temperature.l  
owLimit)) &&  
((currentPressure>FVS007.pressure.highLimit) || (currentPressure<FVS007.pressure.lowLimit)) &&  
((currentCo2<FVS007.co2.highLimit && currentCo2>FVS007.co2.lowLimit)))
```

```
{
```

```
    pc.printf("\r\033[0;37mStatus : \033[0;31mAlarm\t\t\t \033[0;37mStatus:  
\033[0;32mSafe\033[0;37m\t\t \033[0;37mStatus: \033[0;32mSafe\033[0;37m\n");
```

```
}
```

```
    else
```

```
if(((currentTemperature<FVS007.temperature.highLimit) || (currentTemperature>FVS007.temperature.l  
owLimit)) &&  
((currentPressure<FVS007.pressure.highLimit) || (currentPressure>FVS007.pressure.lowLimit)) &&  
((currentCo2<FVS007.co2.highLimit && currentCo2>FVS007.co2.lowLimit)))
```

```
{
```

```
    pc.printf("\r\033[0;37mStatus : \033[0;32mSafe\t\t\t \033[0;37mStatus:  
\033[0;31mAlarm\033[0;37m\t\t \033[0;37mStatus: \033[0;32mSafe\033[0;37m\n");
```

```
}
```

```

else

{

    pc.printf("\r\033[0;37mStatus : \033[0;32mSafe\t\t\t \033[0;37mStatus:
\033[0;32mSafe\033[0;37m\t\t \033[0;37mStatus: \033[0;32mSafe\033[0;37m\n");

}

pc.printf("\n");

pc.printf("\rMOTOR1: Discharge\t\tMOTOR2: Dampener\n");

pc.printf("\rPosition: %i\t\t\tPosition:
%i\n",FVS007.motor1.currentPosition,FVS007.motor2.currentPosition);

pc.printf("\rData:\t 0X0%i\t\t\tData:\t 0X0%i\n",FVS007.motor1.pattern,FVS007.motor2.pattern);

sentFormation();

} //eo display

```

/\*---calRxChecksum -----\*/

Author: SRaj

Date: 08/04/2020

Modified: Name or None

Desc: Calculate the checksum value of received string

Input: Input of buffer

Returns: Returns calulated checksum value

-----\*/

```
char calRxChecksum(char *ptr)
```

```
{
```

```
    char cs2=0;
```

```
    while(*ptr)
```

```
    {
```

```
        cs2+=*ptr;
```

```
        ptr++;
```

```
    }
```

```
    return cs2;
```

```
} // eo calCheckSum
```

```
/*---collectSentence -----
```

Author:  SRaj

Date:   08/04/2020

Modified:  Name or None

Desc:    collect the serial communication sentence from the PIC

Input:    Type and purpose of input arguments

Returns:  Type and purpose of returning argument

```
-----*/
```

```
void collectSentence()
```

```
{
```



```
if(fvs.readable())
{
    char hold=fvs.getc();

    if(hold=='$')
    {
        ptr=rxBuf;
    }

    if(hold=='#')
    {
        sentRdy=TRUE;
    }

    *ptr=hold;

    ptr++;

} //eo if
}
```

/\*---validateSentence-----

Author: SRaj

Date: 08/04/2020

Modified: Name or None

Desc: Validate the received command sentence

Input: buffer as input

Returns: TRUE or FALSE

-----\*/

```
char validateSentence (char *ptr)
```

```
{
```

```
    char rcs=0;
```

```
    char ncs=0;
```

```
    char csFlag=FALSE;
```

```
    int count=strlen(ptr);
```

```
    while(!csFlag)
```

```
    {
```

```
        if(*(ptr+count)=='#')
```

```
        {
```

```
            *(ptr+count)=0X00;
```

```
        }
```

```
        if(*(ptr+count)=='')
```

```
        {
```

```
            *(ptr+count)=0X00;
```

```
            rcs=atoi(ptr+count+1);
```

```
            csFlag=TRUE;
```

```
        }
```

```

        count--;

    }

    ncs=calRxChecksum(ptr);

    if(ncs==rcs)

    {

        return TRUE;

    }

    else

    {

        return FALSE;

    }

} //eo validateSentence

/*---purseSentence -----

Author:   SRaj

Date:    08/04/2020

Modified: Name or None

Desc:    purse the sentence and save each command in token

Input:   buffer as input

Returns: Type and purpose of returning argument

-----*/

```

```
void purseSentence(char *ptr)

{

    int tokenCount=0;


    while(*ptr)

    {

        if(*ptr=='$' || *ptr==',')

        {

            *ptr=0X00;

            token[tokenCount]=ptr+1;

            tokenCount++;

        }

        ptr++;

    }

}

} // eo PurseSentence
```

```
/*---executeSentence -----
```

Author: SRaj

Date: 08/04/2020

Modified: Name or None

Desc:    execute the recieved sentence

Input:    Type and purpose of input arguments

Returns:    Type and purpose of returning argument

-----\*/

```
void executeSentence()
```

```
{
```

```
    if(atoi(token[1])==CONTROLLER)
```

```
    {
```

```
        if(atoi(token[2])==FVS)
```

```
        {
```

```
            if(strcmp(token[0],string1)==0)
```

```
            {
```

```
                if(strcmp(token[3],string2)==0)
```

```
                {
```

```
                    currentTemperature=atoi(token[4]);
```

```
                }
```

```

        if(strcmp(token[3],string3)==0)
        {
            currentPressure=atoi(token[4]);
        }

        if(strcmp(token[3],string4)==0)
        {
            currentCo2=atoi(token[4]);
        }
    }
}
}
}
}
}

```

```

/*--- MAIN FUNCTION -----
-----*/

```

```

int main()
{
    initialize_FVS();

```

```
fvs.attach(&collectSentence);
```

```
while(TRUE)
```

```
{
```

```
    timer.start();
```

```
    if(timer.read()>=1)
```

```
    {
```

```
        userMode();
```

```
        userChannel();
```

```
        increment();
```

```
        decrement();
```

```
        display();
```

```
        if(sentRdy)
```

```
        {
```

```
            validateSentence(rxBuf);
```

```
            purseSentence(rxBuf);
```

```
            executeSentence();
```

```
            sentRdy=FALSE;
```

```
        }
```

```
        timer.reset();
```

```
    }// eo if
```

```
}// eo while  
  
}
```

```
// PIC PROGRAM
```

```
/*-----
```

File Name: SRWork2.c

Author: SRaj

Date: 08/04/2020

Modified: Name or None

copyright Fanshawe College, 2019

Description: Average value is send to controller, all the other operation are same.

```
-----*/
```

```
// Preprocessor -----
```



```
#include "pragmas.h"
```

```
#include <adc.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
#include <stdio.h>
```

```
#include <p18f45k22.h>
```

```
#include "usart.h"
```

```
// Constants -----
```

```
#define TRUE          1
```

```
#define FALSE        0
```

```
#define LED_ON        1
```

```
#define LED_OFF       0
```

```
#define HIGH          1
```

```
#define LOW           0
```

```
#define MAXSAMPLES    20
```

```
#define SIZE           4
```

#define TEMP	0
#define PRES	1
#define CO2	2
#define COEFF	0.0285
#define OFFSET	1.4285
#define TMROFLAG	INTCONbits.TMROIF
#define COEFF_P	0.02
#define VOLTAGE	5
#define ADC_B	1024
#define MSIZE	4
#define U_TEMP	1
#define U_PRES	2
#define MOTOR_P	360
#define MAX_CP	357
#define PATTERNS	4
#define HEATER	PORTBbits.RB0
#define MTR_LED1	PORTDbits.RD5
#define MTR_LED2	PORTDbits.RD4
#define MTR_LED3	PORTCbits.RC5
#define MTR_LED4	PORTCbits.RC4
#define OFFSET_C	0.556
#define COEFF_C	0.00278

```

#define MOTOR2_P          30

#define MTR2_LED1          PORTCbits.RC0
#define MTR2_LED2          PORTCbits.RC1
#define MTR2_LED3          PORTCbits.RC2
#define MTR2_LED4          PORTCbits.RC3

#define BUFFERSIZE         20

#define TOKENSIZE          5

#define CMDSTM              0

#define ADDYTO              1

#define ADDYFM              7

#define RCFLAG              PIR3bits.RC2IF

#define FVS                 7

#define CONTROLLER         1

```

```

// Global Variables -----

```

```

char currentTemperature=0;

```

```

char currentPressure=0;

```

```

int currentCo2=0;

```

```

char buf[BUFFERSIZE];

```

```

char txBuf[10];

```

```

char timeCounter=0;

```

```
char *tokens[TOKENSIZE];

char sentenceRdy=FALSE;

char insert=0;

char sentFlag=FALSE;

char count_s=0;

char string1[]="CONLIM";

char string2[]="T";

char string3[]="H";

char string4[]="L";

char string5[]="P";

char string6[]="C";

char mtrPattern[MSIZE]= {0X08, 0X04, 0X02, 0X01};

char mtr2Pattern[MSIZE]= {0X08, 0X04, 0X02, 0X01};


typedef struct motor
{
    char position;

    int currentPosition;

    char pattern;

    char patternCounter;

}mtr_t;
```

```
typedef struct sensorChannel
{
    char currentSample;

    int samples[MAXSAMPLES];

    int avgSample;

    char insertAt;

    int highLimit;

    int lowLimit;
}sensorCh_t;
```

```
typedef struct fvs
{
    sensorCh_t temperature;

    sensorCh_t pressure;

    sensorCh_t co2;

    char indicator;

    char heater;

    mtr_t motor1;

    mtr_t motor2;

    char userLimit;
```

```
char userSensor;
```

```
}FVS_t;
```

```
FVS_t FVS007;
```

```
// Functions -----
```

```
/*--- set_osc -----
```

```
Author:      SRaj
```

```
Date:        26/03/2020
```

```
Modified:     Name or None
```

```
Desc:         Set the oscillation to 4Mhz
```

```
Input:        Type and purpose of input arguments
```

```
Returns:      Type and purpose of returning argument
```

```
-----*/
```

```
void set_osc (void)
```

```
{
```

```
OSCCON=0X52;                                // set the oscillation to 4Mhz
```

```
OSCCON2=0X04;
```

```
OSCTUNE=0X80;
```

```
while(OSCCONbits.HFIOFS != 1);
```

```
}// eo set_osc
```

```
/*--- adcConfig -----
```

```
Author:      SRaj
```

```
Date:        26/03/2020
```

```
Modified:     Name or None
```

```
Desc:         Configure the ADC port.
```

```
Input:        Type and purpose of input arguments
```

```
Returns:      Type and purpose of returning argument
```

```
-----*/
```

```
void adcConfig(void)
```

```
{
```

```
    ADCON0=0X01;
```

```
    ADCON1=0X00;
```

```
    ADCON2=0XA9;
```

```
}// eo adcConfig
```

```
/*--- configPort -----
```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         configure the port of PIC microcontroller

Input:        Type and purpose of input arguments

Returns:      Type and purpose of returning argument

```
-----*/
```

```
void configPort (void)
```

```
{
```

```
    LATA=0X00;
```

```
    ANSELA=0X07;
```

```
    TRISA=0XFF;
```

```
    LATB=0X00;
```

```
    ANSELB=0X00;
```



```
TRISB=0X00;
```

```
LATC=0X00;
```

```
ANSELC=0X00;
```

```
TRISCbits.RC6=1;
```

```
TRISCbits.RC7=1;
```

```
TRISCbits.RC0=0;
```

```
TRISCbits.RC1=0;
```

```
TRISCbits.RC2=0;
```

```
TRISCbits.RC4=0;
```

```
TRISCbits.RC5=0;
```

```
LATD=0X00;
```

```
ANSELD=0X00;
```

```
TRISDbits.RD6=1;
```

```
TRISDbits.RD7=1;
```

```
TRISDbits.RD4=0;
```

```
TRISDbits.RD5=0;
```

```
}// eo configPort
```

Author:	SRaj
Date:	26/03/2020
Modified:	Name or None
Desc:	Configure the serial communication port of controller
Input:	Type and purpose of input arguments
Returns:	Type and purpose of returning argument

```
void serialConfig (void)
```

```
SPBRG1=25; // baudrate to 9600

RCSTA1=0X90;

TXSTA1=0X26;

BAUDCON1=0X40;
```

```
/*--- serialConfig -----
```

Author: SRaj

Date: 26/03/2020

Modified: Name or None

Desc: Configure the serial communication port of controller

Input: Type and purpose of input arguments

Returns: Type and purpose of returning argument

-----\*/

void serial2Config()

{

SPBRG2=12 ;

RCSTA2=0X90;

TXSTA2=0X26;

BAUDCON2=0X40;

}//eo serial2config

/\*--- configTMRO -----

Author: SRaj

Date: 26/03/2020

Modified: Name or None

Desc: Configure TIMER

Input: Type and purpose of input arguments

-----\*/

 $\{$ 

```
TMR0H=0X0B;
```

```
TMROFLAG= FALSE;
```

```
/*--- resetTMR0 -----
```

-----\*/

```
void resetTMR0 (void)
```

```
{  
  
    TMROH= 0X0B;  
  
    TMROL= 0XDC;  
  
    TMROFLAG=FALSE;
```

```
}//eo resetTMRO
```

```
/*--- configInts -----
```

```
Author:      SRaj
```

```
Date:        03/04/2020
```

```
Modified:     Name or None
```

```
Desc:         config the interrupt register
```

```
Input:        Type and purpose of input arguments
```

```
Returns:      Type and purpose of returning argument
```

```
-----*/
```

```
void configInts()
```

```
{  
  
    INTCON=0xC0;  
  
    PIE3bits.RC2IE=1;  
  
}//eo configInts
```

```
/*--- initializeSystem -----
```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Initialize the configuration of controller

Input:        Type and purpose of input arguments

Returns:      Type and purpose of returning argument

```
-----*/
```

```
void initializeSystem (void)
```

```
{
```

```
    set_osc();
```

```
    configPort();
```

```
    adcConfig();
```

```
    serialConfig();
```

```
    serial2Config();
```

```
    configTMR0();
```

```
    resetTMR0();
```

```
    configInts();
```

```
}// eo initializeSystem
```

/\*--- initialize\_FVS -----

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Set the data member to initial value

Input:        Type and purpose of input arguments

Returns:      Type and purpose of returning argument

-----\*/

void initialize\_FVS (void)

{

    int i;

    FVS007.temperature.currentSample=0;

    for(i=0;i<MAXSAMPLES;i++)

    {

        FVS007.temperature.samples[i]=0;

    }

    FVS007.temperature.avgSample=0;

    FVS007.temperature.insertAt=0;

    FVS007.pressure.currentSample=0;

    for(i=0;i<MAXSAMPLES;i++)

    {

```
FVS007.pressure.samples[i]=0;

}

FVS007.pressure.avgSample=0;

FVS007.pressure.insertAt=0;

FVS007.temperature.highLimit=65;

FVS007.pressure.highLimit=17;

FVS007.temperature.lowLimit=12;

FVS007.pressure.lowLimit=103;

FVS007.indicator=FALSE;

FVS007.heater=FALSE;

FVS007.motor1.position=0;

FVS007.motor1.currentPosition=0;

FVS007.motor1.pattern=0;

FVS007.motor1.patternCounter=0;

FVS007.motor2.position=0;

FVS007.motor2.currentPosition=0;

FVS007.motor2.pattern=0;

FVS007.motor2.patternCounter=0;

FVS007.userLimit=0;

FVS007.userSensor=0;

FVS007.co2.currentSample=0;

for(i=0;i<MAXSAMPLES;i++)
```



```

    {
        FVS007.co2.samples[i]=0;
    }

    FVS007.co2.avgSample=0;

    FVS007.co2.insertAt=0;

    FVS007.co2.highLimit=1200;

    FVS007.co2.lowLimit=350;

} // eo initialize_FVS


/*--- getAdcSample -----*/
Author:      SRaj
Date:        26/03/2020
Modified:     Name or None
Desc:         Get the ADC sample from the input pin
Input:        Channel pin to select the channel of ADC
Returns:      Returns ADC sample
-----*/

int getAdcSample(char channel)
{

```

```

        ADCON0&=0X83;                                // getting input from
the input channel

        ADCON0|=(channel<<2);

        ADCON0bits.GO=TRUE;

        while(ADCON0bits.GO);

        return ADRES;

} // eo getAdcSample

```

```

/*---getTemp -----

```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Convert the ADC Sample to temperature value by adc resolution

Input:        Takes adc sample as input from the temperature channel

Returns:      Returns the Temperature Value

```

-----*/

```

```

int getTemp ()

```

```

{

    float adcResultTemp=0;

    adcResultTemp=getAdcSample(TEMP);

    adcResultTemp/=ADC_B;                                // dividing the sample
by 1024

    adcResultTemp*=VOLTAGE;                                // multiply the sample
to 5

    adcResultTemp-=OFFSET;

    adcResultTemp/=COEFF;

    return adcResultTemp;                                // after resolution and solving
temp, result is returning

} // eo getTemp

```

```

/*---getPres -----

```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Convert the adc sample to pressure value by adc resolution

Input:        Takes adc sample as input from the pressure channel

Returns:      Returns the Pressure Value

```

-----*/

```

```

int getPres ()
{
    float adcResultPres=0;

    adcResultPres=getAdcSample(PRES);

    adcResultPres/=ADC_B;           // dividing the sample
    by 1024

    adcResultPres*=VOLTAGE;         // multiply the sample
    to 5

    adcResultPres/=COEFF_P;

    return adcResultPres;

} // eo getPres

```

/\*---getCo2 -----

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Convert the adc sample to co2 value by adc resolution

Input:        Takes adc sample as input from the pressure channel

Returns:      Returns the Co2 Value

-----\*/

```

int getCo2()

```

```

{

```

```

float adcResultCo2=0;

adcResultCo2=getAdcSample(CO2);

adcResultCo2/=ADC_B;

adcResultCo2*=VOLTAGE;

adcResultCo2+=OFFSET_C;

adcResultCo2/=COEFF_C;

return adcResultCo2;

} // eo getCo2 ::

/*---calTxChecksum -----*/

Author:   SRaj

Date:    27/03/2020

Modified: Name or None

Desc:    checksum is created by using + operator for the trasmission string

Input:   Take the input of buffer

Returns: returns the checksum value

-----*/

```

```

char calTxChecksum(char *ptr)

{

    char cs2=0;

    while(*ptr)

```

```

    {

        cs2+=*ptr;

        ptr++;

    }

    return cs2;

} // eo calChecksum

```

```

/*---avgTemp -----

```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Calculate the average of Temp channel

Input:        None

Returns:      Returns average value

```

-----*/

```

```

int avgTemp()

```

```

{

```

```

    int tempAvg=0;

```

```

    FVS007.temperature.currentSample=getTemp();

```

```

    FVS007.temperature.samples[FVS007.temperature.insertAt]=FVS007.temperature.currentSampl
e;

```

```
FVS007.temperature.avgSample+=FVS007.temperature.samples[FVS007.temperature.insertAt];
```

```
FVS007.temperature.insertAt++;
```

```
if(FVS007.temperature.insertAt>=MAXSAMPLES)
```

```
{
```

```
    FVS007.temperature.insertAt=0;
```

```
    FVS007.temperature.avgSample/=MAXSAMPLES;
```

```
    tempAvg=FVS007.temperature.avgSample;
```

```
    return tempAvg;
```

```
}
```

```
// eo avgTemp
```

```
/*---avgPres -----
```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Calculate the average of Pressure channel

Input:        None

Returns:      Returns average value

```
-----*/
```

```

int avgPres()
{
    int presAvg=0;

    FVS007.pressure.currentSample=getPres();

    FVS007.pressure.samples[FVS007.pressure.insertAt]=FVS007.temperature.currentSample;

    FVS007.pressure.avgSample+=FVS007.pressure.samples[FVS007.pressure.insertAt];

    FVS007.pressure.insertAt++;

    if(FVS007.pressure.insertAt>=MAXSAMPLES)
    {
        FVS007.pressure.insertAt=0;

        FVS007.pressure.avgSample/=MAXSAMPLES;

        presAvg=FVS007.pressure.avgSample;

        return presAvg;
    }

}

} // eo avgPres

```

/\*---avgCo2 -----

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None



Desc: Calculate the average of CO2 channel

Input: None

Returns: Returns average value

-----\*/

```
int avgCo2()
```

```
{
```

```
    int co2Avg=0;
```

```
    FVS007.co2.currentSample=getCo2();
```

```
    FVS007.co2.samples[FVS007.co2.insertAt]=FVS007.co2.currentSample;
```

```
    FVS007.co2.avgSample+=FVS007.co2.samples[FVS007.co2.insertAt];
```

```
    FVS007.co2.insertAt++;
```

```
    if(FVS007.co2.insertAt>=MAXSAMPLES)
```

```
    {
```

```
        FVS007.co2.insertAt=0;
```

```
        FVS007.co2.avgSample/=MAXSAMPLES;
```

```
        co2Avg=FVS007.co2.avgSample;
```

```
        return co2Avg;
```

```
    }
```

```
}// eo avyCo2
```

```
/*---motorStart -----
```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         To Start the motor.

Input:        Type and purpose of input arguments

Returns:      Type and purpose of returning argument

```
-----*/
```

```
void motorStart(void)
```

```
{
```

```
    FVS007.motor1.position=MOTOR_P;
```

```
        if(FVS007.motor1.currentPosition!=FVS007.motor1.position ||  
FVS007.motor1.currentPosition<FVS007.motor1.position)
```

```
    {
```

```
        FVS007.motor1.pattern=mtrPattern[FVS007.motor1.patternCounter];
```

```
        FVS007.motor1.currentPosition+=3;
```

```
        FVS007.motor1.patternCounter++;
```

```
        if(FVS007.motor1.patternCounter>=PATTERNS)
```

```
    {
```

```

        FVS007.motor1.patternCounter=0;

    }

    if(FVS007.motor1.currentPosition>=MAX_CP)

    {

        FVS007.motor1.currentPosition=0;

    }

} //eo if

} //eo motorStart

```

```

/*---motor2Start -----

```

```

Author:      SRaj

Date:        26/03/2020

Modified:     Name or None

Desc:         To open the Dampener

Input:        None

Returns:      None

```

```

-----*/

```

```

void motor2Start()

```

```

{

    FVS007.motor2.position=MOTOR2_P;

    if(FVS007.motor2.currentPosition!=FVS007.motor2.position ||
FVS007.motor2.currentPosition<FVS007.motor2.position)

        {

            FVS007.motor2.pattern=mtr2Pattern[FVS007.motor2.patternCounter];

            FVS007.motor2.currentPosition+=3;

            FVS007.motor2.patternCounter++;

            if(FVS007.motor2.patternCounter>=PATTERNS)

                {

                    FVS007.motor2.patternCounter=0;

                }

        }

}

} // eo motor2Start

```

/\*---motor2Stop -----

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         To close the Dampener

Input:           None

Returns:       None

-----\*/

void motor2Stop()

{

    FVS007.motor2.position=0;

    if(FVS007.motor2.currentPosition!=FVS007.motor2.position ||  
FVS007.motor2.currentPosition>FVS007.motor2.position)

    {

        FVS007.motor2.pattern=mtr2Pattern[FVS007.motor2.patternCounter];

        FVS007.motor2.currentPosition-=3;

        FVS007.motor2.patternCounter++;

        if(FVS007.motor2.patternCounter>=PATTERNS)

        {

            FVS007.motor2.patternCounter=0;

        }

    }

}// eo motor2Stop

/\*---co2Operation -----

Author:       SRaj

Date: 26/03/2020

Modified: Name or None

Desc: control the operation of CO2

Input: None

Returns: None

-----\*/

```
void co2Operation()
```

```
{
```

```
    if(currentCo2>FVS007.co2.highLimit)
```

```
    {
```

```
        motor2Start();
```

```
        if(FVS007.motor2.patternCounter==0)
```

```
        {
```

```
            MTR2_LED1=LED_ON;
```

```
            MTR2_LED2=LED_OFF;
```

```
            MTR2_LED3=LED_OFF;
```

```
            MTR2_LED4=LED_OFF;
```

```
        }
```

```
        if(FVS007.motor2.patternCounter==1)
```

```
        {
```

```
        MTR2_LED1=LED_OFF;

        MTR2_LED2=LED_ON;

        MTR2_LED3=LED_OFF;

        MTR2_LED4=LED_OFF;
    }
```

```
    if(FVS007.motor2.patternCounter==2)
    {

        MTR2_LED1=LED_OFF;

        MTR2_LED2=LED_OFF;

        MTR2_LED3=LED_ON;

        MTR2_LED4=LED_OFF;
    }
```

```
    if(FVS007.motor2.patternCounter==3)
    {

        MTR2_LED1=LED_OFF;

        MTR2_LED2=LED_OFF;

        MTR2_LED3=LED_OFF;

        MTR2_LED4=LED_ON;
    }
```

```
}
```

```
if(currentCo2<FVS007.co2.highLimit && currentCo2>FVS007.co2.lowLimit)

{

    motor2Stop();

    if(FVS007.motor2.patternCounter==0)

    {

        MTR2_LED1=LED_ON;

        MTR2_LED2=LED_OFF;

        MTR2_LED3=LED_OFF;

        MTR2_LED4=LED_OFF;

    }

    if(FVS007.motor2.patternCounter==1)

    {

        MTR2_LED1=LED_OFF;

        MTR2_LED2=LED_ON;

        MTR2_LED3=LED_OFF;

        MTR2_LED4=LED_OFF;

    }

    if(FVS007.motor2.patternCounter==2)

    {
```



```
        MTR2_LED1=LED_OFF;

        MTR2_LED2=LED_OFF;

        MTR2_LED3=LED_ON;

        MTR2_LED4=LED_OFF;

    }
```

```
    if(FVS007.motor2.patternCounter==3)

    {

        MTR2_LED1=LED_OFF;

        MTR2_LED2=LED_OFF;

        MTR2_LED3=LED_OFF;

        MTR2_LED4=LED_ON;

    }
```

```
    if(FVS007.motor2.currentPosition==FVS007.motor2.position)

    {

        MTR2_LED1=LED_OFF;

        MTR2_LED2=LED_OFF;

        MTR2_LED3=LED_OFF;

        MTR2_LED4=LED_OFF;

    }
```

```
}
```

```
}// eo co2Operation
```

```
/*---controlOperation -----
```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Controls the operation of motor

Input:        Type and purpose of input arguments

Returns:      Type and purpose of returning argument

```
-----*/
```

```
void controlOperation (void)
```

```
{
```

```
    if (currentTemperature>FVS007.temperature.lowLimit &&  
currentTemperature<FVS007.temperature.highLimit)
```

```
    {
```

```
        FVS007.motor1.position=FVS007.motor1.currentPosition;
```

```
        MTR_LED1=LED_OFF;
```

```
        MTR_LED2=LED_OFF;
```

```
        MTR_LED3=LED_OFF;
```

```
        MTR_LED4=LED_OFF;
```

```

        if(currentPressure<FVS007.pressure.highLimit)

        {

            HEATER=LED_OFF;

        }

        else

        {

            HEATER=LED_ON;

        }

    }//eo if

```

```

        if (((currentTemperature>FVS007.temperature.highLimit ||
(currentPressure<FVS007.pressure.highLimit)) || (currentTemperature>FVS007.temperature.highLimit
&& currentPressure<FVS007.pressure.highLimit)))

        {

            motorStart();

            HEATER=LED_OFF;

            if (FVS007.motor1.patternCounter==0)

            {

                MTR_LED1=LED_ON;

                MTR_LED2=LED_OFF;

```

```
        MTR_LED3=LED_OFF;

        MTR_LED4=LED_OFF;
    }

    else if (FVS007.motor1.patternCounter==1)
    {

        MTR_LED1=LED_OFF;

        MTR_LED2=LED_ON;

        MTR_LED3=LED_OFF;

        MTR_LED4=LED_OFF;
    }

    else if (FVS007.motor1.patternCounter==2)
    {

        MTR_LED1=LED_OFF;

        MTR_LED2=LED_OFF;

        MTR_LED3=LED_ON;

        MTR_LED4=LED_OFF;
    }

    else
    {
```

```

        MTR_LED1=LED_OFF;

        MTR_LED2=LED_OFF;

        MTR_LED3=LED_OFF;

        MTR_LED4=LED_ON;

    }

} //eo if


if (currentTemperature<FVS007.temperature.lowLimit)
{

    FVS007.motor1.position=FVS007.motor1.currentPosition;


    MTR_LED1=LED_OFF;

    MTR_LED2=LED_OFF;

    MTR_LED3=LED_OFF;

    MTR_LED4=LED_OFF;

    if(currentPressure<FVS007.pressure.highLimit)
    {

        HEATER=LED_OFF;

    }

    else

    {

        HEATER=LED_ON;

```

```

    }

} //eo if

if (currentPressure<FVS007.pressure.highLimit && currentPressure>FVS007.pressure.lowLimit)
{
    FVS007.motor1.position=FVS007.motor1.currentPosition;

    MTR_LED1=LED_OFF;

    MTR_LED2=LED_OFF;

    MTR_LED3=LED_OFF;

    MTR_LED4=LED_OFF;

    if (HEATER==LED_ON)
    {
        HEATER=LED_ON;
    }
    else
    {
        HEATER=LED_OFF;
    }
} // eo if

if (currentPressure>FVS007.pressure.lowLimit)

```

```

{

    FVS007.motor1.position=FVS007.motor1.currentPosition;

    MTR_LED1=LED_OFF;

    MTR_LED2=LED_OFF;

    MTR_LED3=LED_OFF;

    MTR_LED4=LED_OFF;


    if (HEATER==LED_ON)

    {

        HEATER=LED_ON;

    }

    else

    {

        HEATER=LED_OFF;

    }

} //eo if

```

```

} // eo controlOperation

```

```

/*---ledControl -----

```

Author:       SRaj

Date: 26/03/2020

Modified: Name or None

Desc: Controls the led of the sensors

Input: Type and purpose of input arguments

Returns: Type and purpose of returning argument

-----\*/

```
void ledControl()
```

```
{
```

```
    if (currentTemperature>FVS007.temperature.highLimit &&  
currentPressure<FVS007.pressure.highLimit)
```

```
    {
```

```
        PORTBbits.RB1=LED_ON;                                // when pressure is  
high
```

```
        PORTBbits.RB2=LED_OFF;
```

```
        PORTBbits.RB3=LED_OFF;
```

```
        PORTBbits.RB4=LED_ON;                                // when temperature is  
high
```

```
    }
```

```
    else if (currentTemperature<FVS007.temperature.lowLimit &&  
currentPressure>FVS007.pressure.lowLimit) // when pressure and temperature is low
```



```
{  
  
    PORTBbits.RB1=LED_OFF;  
  
    PORTBbits.RB2=LED_ON;           // when pressure is low  
  
    PORTBbits.RB3=LED_ON;           // when temperature is  
low  
  
    PORTBbits.RB4=LED_OFF;  
  
}
```

```
else if (currentTemperature>FVS007.temperature.highLimit &&  
currentPressure>FVS007.pressure.lowLimit)    // when pressure is low and temperature is high
```

```
{  
  
    PORTBbits.RB1=LED_OFF;  
  
    PORTBbits.RB2=LED_ON;  
  
    PORTBbits.RB3=LED_OFF;  
  
    PORTBbits.RB4=LED_ON;  
  
}
```

```
else if (currentPressure<FVS007.pressure.highLimit &&  
currentTemperature<FVS007.temperature.lowLimit)    // when pressure is high and temperature is low
```

```
{  
  
    PORTBbits.RB1=LED_ON;
```

```
    PORTBbits.RB2=LED_OFF;

    PORTBbits.RB3=LED_ON;

    PORTBbits.RB4=LED_OFF;

}
```

```
    else if (currentTemperature<FVS007.temperature.highLimit &&
currentTemperature>FVS007.temperature.lowLimit && currentPressure<FVS007.pressure.highLimit) //
when pressure is high
```

```
    {

        PORTBbits.RB1=LED_ON;

        PORTBbits.RB2=LED_OFF;

        PORTBbits.RB3=LED_OFF;

        PORTBbits.RB4=LED_OFF;

    }
```

```
    else if (currentTemperature<FVS007.temperature.highLimit &&
currentTemperature>FVS007.temperature.lowLimit && currentPressure>FVS007.pressure.lowLimit) //
when pressure is low
```

```
    {

        PORTBbits.RB1=LED_OFF;

        PORTBbits.RB2=LED_ON;

        PORTBbits.RB3=LED_OFF;

        PORTBbits.RB4=LED_OFF;

    }
```

```
        else if (currentPressure>FVS007.pressure.highLimit &&  
currentPressure<FVS007.pressure.lowLimit && currentTemperature>FVS007.temperature.highLimit) //  
when temperature is high
```

```
{  
  
    PORTBbits.RB1=LED_OFF;  
  
    PORTBbits.RB2=LED_OFF;  
  
    PORTBbits.RB3=LED_OFF;  
  
    PORTBbits.RB4=LED_ON;  
  
}
```

```
        else if (currentPressure>FVS007.pressure.highLimit &&  
currentPressure<FVS007.pressure.lowLimit && currentTemperature<FVS007.temperature.lowLimit) //  
when temperature is low
```

```
{  
  
    PORTBbits.RB1=LED_OFF;  
  
    PORTBbits.RB2=LED_OFF;  
  
    PORTBbits.RB3=LED_ON;  
  
    PORTBbits.RB4=LED_OFF;  
  
}
```

```
else
```

```

    {

        PORTBbits.RB1=LED_OFF;

        PORTBbits.RB2=LED_OFF;

        PORTBbits.RB3=LED_OFF;

        PORTBbits.RB4=LED_OFF;

    }// eo else..if

} // eo ledcontrol


/*--- diaplay -----
Author:      SRaj
Date:        26/03/2020
Modified:     Name or None
Desc:         Display the output to terminal window
Input:        Type and purpose of input arguments
Returns:      Type and purpose of returning argument
-----*/

void display (void)

{

    printf("\033[2J\033[H\033[0");

    printf("\rFFVS007 System Properties\n");

```

```
printf("\r\n");

if (FVS007.userLimit==1 && FVS007.userSensor==1)

{

    printf("\rMODE: HIGH \t\t CHANNEL: TEMPERATURE \n ");

}


if (FVS007.userLimit==0 && FVS007.userSensor==1)

{

    printf("\rMODE: LOW \t\t CHANNEL: TEMPERATURE\n");

}


if (FVS007.userLimit==1 && FVS007.userSensor==0)

{

    printf("\rMODE: HIGH \t\t CHANNEL: PRESSURE\n");

}


if (FVS007.userLimit==0 && FVS007.userSensor==0)

{

    printf("\r MODE: LOW \t\t CHANNEL: PRESSURE\n");

}


if(FVS007.userLimit==1 && FVS007.userSensor==2)

{

    printf("\rMODE: HIGH \t\t CHANNEL: CO2\n");
```

```

    }

    if(FVS007.userLimit==0 && FVS007.userSensor==2)

    {

        printf("\rMODE: LOW \t\t CHANNEL: CO2\n");

    }

    printf("\n\rTemperature\t\t\tPressure\t\tCO2\n");

    printf("\rCurrent:\t%i%cC\t\tCurrent:  %i kPa \tCurrent:  %d ppm\n",
currentTemperature,248,currentPressure,currentCo2); //for printing current value of temperature and
pressure

    printf("\rHighLimit:\t%i%cC\t\tHighLimit: %i kPa\tHighLimit:  %i ppm\n",
FVS007.temperature.highLimit,248,FVS007.pressure.highLimit,FVS007.co2.highLimit);           // for
printing high limit of all the sensors

    printf("\rLowLimit:\t%i%cC\t\tLowLimit:  %i kPa\tLowLimit:  %i ppm\n",
FVS007.temperature.lowLimit,248,FVS007.pressure.lowLimit,FVS007.co2.lowLimit);
    // for printing low limit of both the sesnsors

    if(((currentTemperature>FVS007.temperature.highLimit) || (currentTemperature<FVS007.tempe
rature.lowLimit)) &&
((currentPressure<FVS007.pressure.highLimit) || (currentPressure>FVS007.pressure.lowLimit)) &&
((currentCo2>FVS007.co2.highLimit) || (currentCo2<FVS007.co2.lowLimit)))

    {

        printf("\r\033[0;37mStatus : \033[0;31mAlarm\t\t\t\033[0;37mStatus:
\033[0;31mAlarm\033[0;37m\t\t\033[0;37mStatus: \033[0;31mAlarm\033[0;37m\n ");
        // for printing temperature and pressure status

    }

    else

    if(((currentTemperature>FVS007.temperature.highLimit) || (currentTemperature<FVS007.temperature.l

```

```

owLimit)) &&
((currentPressure>FVS007.pressure.highLimit) || (currentPressure<FVS007.pressure.lowLimit)) &&
((currentCo2>FVS007.co2.highLimit) || (currentCo2<FVS007.co2.lowLimit)))

{

    printf("\r\033[0;37mStatus : \033[0;31mAlarm\t\t\t \033[0;37mStatus:
\033[0;32mSafe\033[0;37m\t\t \033[0;37mStatus: \033[0;31mAlarm\033[0;37m\n");

}

else

if(((currentTemperature<FVS007.temperature.highLimit) || (currentTemperature>FVS007.temperature.l
owLimit)) &&
((currentPressure<FVS007.pressure.highLimit) || (currentPressure>FVS007.pressure.lowLimit)) &&
((currentCo2>FVS007.co2.highLimit) || (currentCo2<FVS007.co2.lowLimit)))

{

    printf("\r\033[0;37mStatus : \033[0;32mSafe\t\t\t \033[0;37mStatus:
\033[0;31mAlarm\033[0;37m\t\t \033[0;37mStatus: \033[0;31mAlarm\033[0;37m\n");

}

else if ((currentTemperature<FVS007.temperature.highLimit &&
currentTemperature>FVS007.temperature.lowLimit) && (currentPressure>FVS007.pressure.highLimit
&& currentPressure<FVS007.pressure.lowLimit) && ((currentCo2>FVS007.co2.highLimit) ||
(currentCo2<FVS007.co2.lowLimit)))

{

    printf("\r\033[0;37mStatus : \033[0;32mSafe\t\t\t \033[0;37mStatus:
\033[0;32mSafe\033[0;37m\t\t \033[0;37mStatus: \033[0;31mAlarm\033[0;37m\n");

}

else if(((currentTemperature>FVS007.temperature.highLimit) ||
(currentTemperature<FVS007.temperature.lowLimit)) &&
((currentPressure>FVS007.pressure.lowLimit) || (currentPressure<FVS007.pressure.highLimit)) &&
(currentCo2>FVS007.co2.lowLimit && currentCo2<FVS007.co2.highLimit))

```

```

{

    printf("\r\033[0;37mStatus : \033[0;31mAlarm\t\t\t \033[0;37mStatus:
\033[0;31mAlarm\033[0;37m\t\t \033[0;37mStatus: \033[0;32mSafe\033[0;37m\n");

}

else
if(((currentTemperature>FVS007.temperature.highLimit) || (currentTemperature<FVS007.temperature.l
owLimit)) &&
((currentPressure>FVS007.pressure.highLimit) || (currentPressure<FVS007.pressure.lowLimit)) &&
((currentCo2<FVS007.co2.highLimit && currentCo2>FVS007.co2.lowLimit)))

{

    printf("\r\033[0;37mStatus : \033[0;31mAlarm\t\t\t \033[0;37mStatus:
\033[0;32mSafe\033[0;37m\t\t \033[0;37mStatus: \033[0;32mSafe\033[0;37m\n");

}

else
if(((currentTemperature<FVS007.temperature.highLimit) || (currentTemperature>FVS007.temperature.l
owLimit)) &&
((currentPressure<FVS007.pressure.highLimit) || (currentPressure>FVS007.pressure.lowLimit)) &&
((currentCo2<FVS007.co2.highLimit && currentCo2>FVS007.co2.lowLimit)))

{

    printf("\r\033[0;37mStatus : \033[0;32mSafe\t\t\t \033[0;37mStatus:
\033[0;31mAlarm\033[0;37m\t\t \033[0;37mStatus: \033[0;32mSafe\033[0;37m\n");

}

else

{

    printf("\r\033[0;37mStatus : \033[0;32mSafe\t\t\t \033[0;37mStatus:
\033[0;32mSafe\033[0;37m\t\t \033[0;37mStatus: \033[0;32mSafe\033[0;37m\n");

}

printf("\n");

```



```

        printf("\rMOTOR1: Discharge\t\tMOTOR2: Dampener\n");

        printf("\rPosition: %i\t\t\tPosition:
%i\n",FVS007.motor1.currentPosition,FVS007.motor2.currentPosition);

        printf("\rData:\t 0X0%i\t\t\tData:\t
0X0%i\n",FVS007.motor1.pattern,FVS007.motor2.pattern);

    } //eo display

```

```

/*---calChecksum -----

```

```

Author:      SRaj

Date:        26/03/2020

Modified:    Name or None

Desc:        Calculate the check sum of the string

Input:       Pointer to the string

Returns:     Return checksum value

```

```

-----*/

```

```

char calChecksum(char *ptr)

```

```

{

    char cs=0;

    while(*ptr)

    {

        cs^=*ptr;

```

```

        ptr++;

    }

    return cs;

} // eo calChecksum

```

```

/*---ISR-----

```

Author:       SRaj

Date:         03/04/2020

Modified:     Name or None

Desc:         when sentence is ready, interrupt will happen and sentence will be collected

Input:        Type and purpose of input arguments

Returns:      Type and purpose of returning argument

```

-----*/

```

```

void ISR();

```

```

#pragma code int_vector=0x008

```

```

void int_vector()

```

```

{

```

```

    _asm

```

```

        GOTO ISR

```

```

        _endasm
    }

#pragma code

/*---collectSentence -----
Author:      SRaj
Date:        03/04/2020
Modified:     Name or None
Desc:        collect the serial communication sentence from the mbed
Input:       Type and purpose of input arguments
Returns:     Type and purpose of returning argument
-----*/

void collectSentence(char *ptr)
{
    char hold=0;

    if(PIR3bits.RC2IF)
    {
        hold=RCREG2;

        if(hold=='$')
        {

```

```

        *ptr=hold;

        while(hold!='#')

        {

            if(PIR3bits.RC2IF)

            {

                hold=RCREG2;

                ptr++;

                *ptr=hold;

                if (hold=='#')

                {

                    ptr++;

                    *ptr = 0x00;

                    sentenceRdy= TRUE;

                }

            }

        }

    }

}

} //eo collectSentence

#pragma interrupt ISR

```

```

void ISR(void)

{

    if(PIR3bits.RC2IF)

    {

        PIE3bits.RC2IE=0;

        collectSentence(buf);

        PIE3bits.RC2IE=1;

    }

} // eo ISR

```

/\*---validateSentence-----

Author:       SRaj

Date:         03/04/2020

Modified:     Name or None

Desc:         Validate the received command sentence

Input:        buffer as input

Returns:      TRUE or FALSE

-----\*/

char validateSentence (char \*ptr)

```

{

```

```
char rcs=0;
```

```
char ncs=0;
```

```
char csFlag=FALSE;
```

```
int count=strlen(ptr);
```

```
while(!csFlag)
```

```
{
```

```
    if(*(ptr+count)=='#')
```

```
    {
```

```
        *(ptr+count)=0X00;
```

```
    }
```

```
    if(*(ptr+count)=='')
```

```
    {
```

```
        *(ptr+count)=0X00;
```

```
        rcs=atoi(ptr+count+1);
```

```
        csFlag=TRUE;
```

```
    }
```

```
    count--;
```

```
}
```

```
ncs=calChecksum(ptr);
```

```
if(ncs==rcs)
```

```
{
```

```

        return TRUE;

    }

    else

    {

        return FALSE;

    }

} //eo validateSentence

```

```

/*---purseSentence -----

```

Author:       SRaj

Date:         03/04/2020

Modified:     Name or None

Desc:         purse the sentence and save each command in token

Input:        buffer as input

Returns:      Type and purpose of returning argument

```

-----*/

```

```

void purseSentence(char *ptr)

```

```

{

```

```

    int tokenCount=0;

```

```

    while(*ptr)

```

```

    {

        if(*ptr=='$' || *ptr==',')

        {

            *ptr=0X00;

            tokens[tokenCount]=ptr+1;

            tokenCount++;

        }

        ptr++;

    }

} // eo ParseSentence

/*---executeSentence -----
Author:      SRaj
Date:        03/04/2020
Modified:     Name or None
Desc:         execute the recieved sentence
Input:        Type and purpose of input arguments
Returns:      Type and purpose of returning argument
-----*/

```



```
void executeSentence()
```

```
{
```

```
    if(atoi(tokens[1])==ADDYTO)
```

```
    {
```

```
        if(atoi(tokens[2])==ADDYFM)
```

```
        {
```

```
            if(strcmp(tokens[0],string1)==0)
```

```
            {
```

```
                if(strcmp(tokens[3],string2)==0)
```

```
                {
```

```
                    if(strcmp(tokens[4],string3)==0)
```

```
                    {
```

```
                        FVS007.temperature.highLimit=atoi(tokens[5]);
```

```
                        FVS007.userLimit=1;
```

```
                        FVS007.userSensor=1;
```

```
    }  
  
    else  
  
    {  
  
        FVS007.temperature.lowLimit=atoi(tokens[5]);  
  
        FVS007.userLimit=0;  
  
        FVS007.userSensor=1;  
  
    }  
  
}  
  
if(strcmp(tokens[3],string5)==0)  
  
{  
  
    if(strcmp(tokens[4],string3)==0)  
  
    {  
  
        FVS007.pressure.highLimit=atoi(tokens[5]);  
  
        FVS007.userLimit=1;  
  
        FVS007.userSensor=0;  
  
    }  
  
    else  
  
    {  
  
        FVS007.pressure.lowLimit=atoi(tokens[5]);  
  
        FVS007.userLimit=0;  
  
        FVS007.userSensor=0;  
  
    }  
  
}
```

```

    }

    if(strcmp(tokens[3],string6)==0)

    {

        if(strcmp(tokens[4],string3)==0)

        {

            FVS007.co2.highLimit=atoi(tokens[5]);

            FVS007.userLimit=1;

            FVS007.userSensor=2;

        }

        else

        {

            FVS007.co2.lowLimit=atoi(tokens[5]);

            FVS007.userLimit=0;

            FVS007.userSensor=2;

        }

    }

}

}

}

} //eo executeSentence

```

/\*---sendAvg -----

Author:       SRaj

Date:         08/04/2020

Modified:     Name or None

Desc:         when average value is calculated, command string is formed and send to TX2

Input:        Type and purpose of input arguments

Returns:      Type and purpose of returning argument

-----\*/

void sendAvg (void)

{

    if(count\_s>=1)

    {

        sprintf(txBuf,"\$AVGUPD,%i,%i,T,%i\0",CONTROLLER,FVS,currentTemperature);

        sprintf(txBuf,"%s,%i#\0",txBuf,calTxChecksum(txBuf));

        puts2USART(txBuf);

    }

    if(count\_s>=2)

    {

        sprintf(txBuf,"\$AVGUPD,%i,%i,P,%i\0",CONTROLLER,FVS,currentPressure);

        sprintf(txBuf,"%s,%i#\0",txBuf,calTxChecksum(txBuf));

        puts2USART(txBuf);

```

    }

    if(count_s>=3)

    {

        sprintf(txBuf,"$AVGUPD,%i,%i,C,%i\0",CONTROLLER,FVS,currentCo2);

        sprintf(txBuf,"%s,%i#\0",txBuf,calTxChecksum(txBuf));

        puts2USART(txBuf);

        count_s=0;

    }

} // eo sendAvg::

```

```

/*--- MAIN FUNCTION -----
-----*/

```

```

void main ()

{

    initializeSystem();

    initialize_FVS();

    while (1)

    {

```

```
if (TMR0FLAG)
```

```
{
```

```
    timeCounter++;
```

```
    co2Operation();
```

```
    if(timeCounter>=4)
```

```
    {
```

```
        timeCounter=0;
```

```
        count_s++;
```

```
        currentTemperature=avgTemp();
```

```
        currentPressure=avgPres();
```

```
        currentCo2=avgCo2();
```

```
        controlOperation();
```

```
        sendAvg();
```

```
        ledControl();
```

```
        display();
```

```
        if(sentenceRdy==TRUE)
```

```
        {
```

```
            sentenceRdy=FALSE;
```

```
printf("\r\n%s\n",buf);
```

```
validateSentence(buf);
```

```
purseSentence(buf);
```

```
executeSentence();
```

```
}//eo if
```

```
}// eo if
```

```
resetTMR0();
```

```
}//eo if
```

```
}// eo while
```

```
}
```

// PIC PROGRAM

//--- LAB 5B ---

/\*-----

File Name: ELNC6007MSLAB5B

Author: SRaj

Date: 16/04/2020

Modified: Name or None

copyright Fanshawe College, 2019

Description: Motor Operation is send to Mbed, all the other operation are same as before.



-----\*/

// Preprocessor -----

#include "pragmas.h"

#include <adc.h>

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <p18f45k22.h>

#include "usart.h"

```
// Constants -----
```

```
#define TRUE          1
```

```
#define FALSE        0
```

```
#define LED_ON        1
```

```
#define LED_OFF       0
```

```
#define HIGH          1
```

```
#define LOW           0
```

```
#define MAXSAMPLES    15
```

```
#define SIZE           4
```

```
#define TEMP           0
```

#define PRES	1
#define CO2	2
#define COEFF	0.0285
#define OFFSET	1.4285
#define TMR0FLAG	INTCONbits.TMR0IF
#define COEFF_P	0.02
#define VOLTAGE	5
#define ADC_B	1024
#define MSIZE	4
#define U_TEMP	1
#define U_PRE	2
#define MOTOR_P	360
#define MAX_CP	357
#define PATTERNS	4
#define HEATER	PORTBbits.RB0

```
#define MTR_LED1          PORTDbits.RD5

#define MTR_LED2          PORTDbits.RD4

#define MTR_LED3          PORTCbits.RC5

#define MTR_LED4          PORTCbits.RC4

#define OFFSET_C          0.556

#define COEFF_C            0.00278

#define MOTOR2_P          30

#define MTR2_LED1         PORTCbits.RC0

#define MTR2_LED2         PORTCbits.RC1

#define MTR2_LED3         PORTCbits.RC2

#define MTR2_LED4         PORTCbits.RC3

#define BUFFERSIZE        30

#define TOKENSIZE         10

#define CMDSTM            0
```

```

#define ADDYTO          1

#define ADDYFM          7

#define RCFLAG          PIR3bits.RC2IF

#define FVS             7

#define CONTROLLER      1

#define MTR1            1

#define MTR2            2

#define MTRON           1

#define MTROFF          0

```

```

// Global Variables -----

```

```

char currentTemperature=0;

```

```

char currentPressure=0;

```

```

int currentCo2=0;

```

```

char buf[BUFFERSIZE];

```

```
char txBuf[BUFFERSIZE];
```

```
char mtrBuf[BUFFERSIZE];
```

```
char timeCounter=0;
```

```
char *tokens[TOKENSIZE];
```

```
char sentenceRdy=FALSE;
```

```
char insert=0;
```

```
char sentFlag=FALSE;
```

```
char count_s=0;
```

```
char string1[]="CONLIM";
```

```
char string2[]="T";
```

```
char string3[]="H";
```

```
char string4[]="L";
```

```
char string5[]="P";
```

```
char string6[]="C";
```

```
char mtr1StartFlag=FALSE;
```

```
char mtr1StopFlag=FALSE;
```

```
char mtr2StartFlag=FALSE;
```

```
char mtr2StopFlag=FALSE;
```

```
char mtrPattern[MSIZE]= {0X08, 0X04, 0X02, 0X01};
```

```
char mtr2Pattern[MSIZE]= {0X08, 0X04, 0X02, 0X01};
```

```
typedef struct motor
```

```
{
```

```
    char position;
```

```
    int currentPosition;
```

```
    char pattern;
```

```
    char patternCounter;
```

```
}mtr_t;
```

```
typedef struct sensorChannel
```

```
{
```

```
    char currentSample;
```

```
    int samples[MAXSAMPLES];
```

```
    int avgSample;
```

```
    char insertAt;
```

```
    int highLimit;
```

```
    int lowLimit;
```

```
}sensorCh_t;
```

```
typedef struct fvs
```

```
{
```



```
sensorCh_t temperature;
```

```
sensorCh_t pressure;
```

```
sensorCh_t co2;
```

```
char indicator;
```

```
char heater;
```

```
mtr_t motor1;
```

```
mtr_t motor2;
```

```
char userLimit;
```

```
char userSensor;
```

```
}FVS_t;
```

```
FVS_t FVS007;
```

// Functions -----

/\*--- set\_osc -----

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Set the oscillation to 4Mhz

Input:        Type and purpose of input arguments

Returns:      Type and purpose of returning argument

-----\*/

void set\_osc(void)

```

{

    OSCCON=0X52;                                // set the oscillation to 4Mhz

    OSCCON2=0X04;

    OSCTUNE=0X80;

    while(OSCCONbits.HFIOFS != 1);

} // eo set_osc

```

```

/*--- adcConfig -----

```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Configure the ADC port.

Input:        Type and purpose of input arguments

Returns:       Type and purpose of returning argument

-----\*/

```
void adcConfig(void)
```

```
{
```

```
    ADCON0=0X01;
```

```
    ADCON1=0X00;
```

```
    ADCON2=0XA9;
```

```
}// eo adcConfig
```

```
/*--- configPort -----
```

Author:       SRaj

Date:         26/03/2020

Modified:      Name or None

Desc:            cofigure the port of PIC microcontroller

Input:           Type and purpose of input arguments

Returns:         Type and purpose of returning argument

-----\*/

```
void configPort (void)
```

```
{
```

```
    LATA=0X00;
```

```
    ANSELA=0X07;
```

```
    TRISA=0XFF;
```

```
    LATB=0X00;
```

```
ANSELB=0X00;
```

```
TRISB=0X00;
```

```
LATC=0X00;
```

```
ANSELB=0X00;
```

```
TRISCbits.RC6=1;
```

```
TRISCbits.RC7=1;
```

```
TRISCbits.RC0=0;
```

```
TRISCbits.RC1=0;
```

```
TRISCbits.RC2=0;
```

```
TRISCbits.RC4=0;
```

```
TRISCbits.RC5=0;
```

```
LATD=0X00;
```

```
ANSELD=0X00;
```

```
TRISDbits.RD6=1;
```

```
TRISDbits.RD7=1;
```

```
TRISDbits.RD4=0;
```

```
TRISDbits.RD5=0;
```

```
}// eo configPort
```

```
/*--- serialConfig -----
```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Configure the serial communication port of controller

Input: Type and purpose of input arguments

Returns: Type and purpose of returning argument

-----\*/

```
void serialConfig (void)
```

$$\{$$

```
SPBRG1=25; // baudrate to 9600
```

```
RCSTA1=0X90;
```

TXSTA1=0X26;

```
BAUDCON1=0X40;
```

```
// eo serialConfig
```



/\*--- serialConfig -----

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Configure the serial communication port of controller

Input:        Type and purpose of input arguments

Returns:      Type and purpose of returning argument

-----\*/

void serial2Config()

{

    SPBRG2=12 ;

    RCSTA2=0X90;

    TXSTA2=0X26;

    BAUDCON2=0X40;

```
}//eo serial2config
```

```
/*--- configTMRO -----
```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Configure TIMER

Input:        Type and purpose of input arguments

Returns:      Type and purpose of returning argument

```
-----*/
```

```
void configTMRO (void)
```

```
{
```

```
TOCON=0X91; // timer set to 1 sec rollover
```

```
TMROH=0X0B;
```

```
TMR0L=0XDC;
```

```
TMROFLAG= FALSE;
```

```
}// eo configTMRO
```

```
/*--- resetTMRO -----
```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         For reseting the timer

Input:        Type and purpose of input arguments

Returns:      Type and purpose of returning argument

```
-----*/
```

```
void resetTMRO (void)
```

```
{
```

```
    TMROH= 0X0B;
```

```
    TMROL= 0XDC;
```

```
    TMROFLAG=FALSE;
```

```
}//eo resetTMRO
```

```
/*--- configInts -----
```

Author:       SRaj

Date:         03/04/2020

Modified:     Name or None

Desc:         config the interrupt register

Input:           Type and purpose of input arguments

Returns:        Type and purpose of returning argument

-----\*/

```
void configInts()
```

```
{
```

```
    INTCON=0xC0;
```

```
    PIE3bits.RC2IE=1;
```

```
}//eo configInts
```

```
/*--- initializeSystem -----
```

Author:        SRaj

Date:          26/03/2020

Modified:      Name or None

Desc:          Initialize the configuration of controller

Input:           Type and purpose of input arguments

Returns:        Type and purpose of returning argument

-----\*/

```
void initializeSystem (void)
```

```
{
```

```
    set_osc();
```

```
    configPort();
```

```
    adcConfig();
```

```
    serialConfig();
```

```
    serial2Config();
```

```
    configTMR0();
```

```
    resetTMR0();
```

```
    configInts();
```

```
}// eo initializeSystem
```

```
/*--- initialize_FVS -----
```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Set the data member to initial value

Input:        Type and purpose of input arguments

Returns:      Type and purpose of returning argument

```
-----*/
```

```
void initialize_FVS (void)
```

```
{
```

```
    int i;
```

```
FVS007.temperature.currentSample=0;
```

```
for(i=0;i<MAXSAMPLES;i++)
```

```
{
```

```
    FVS007.temperature.samples[i]=0;
```

```
}
```

```
FVS007.temperature.avgSample=0;
```

```
FVS007.temperature.insertAt=0;
```

```
FVS007.pressure.currentSample=0;
```

```
for(i=0;i<MAXSAMPLES;i++)
```

```
{
```

```
    FVS007.pressure.samples[i]=0;
```

```
}
```

```
FVS007.pressure.avgSample=0;
```

```
FVS007.pressure.insertAt=0;
```



FVS007.temperature.highLimit=65;

FVS007.pressure.highLimit=17;

FVS007.temperature.lowLimit=12;

FVS007.pressure.lowLimit=103;

FVS007.indicator=FALSE;

FVS007.heater=FALSE;

FVS007.motor1.position=0;

FVS007.motor1.currentPosition=0;

FVS007.motor1.pattern=0;

FVS007.motor1.patternCounter=0;

FVS007.motor2.position=0;

FVS007.motor2.currentPosition=0;

FVS007.motor2.pattern=0;

FVS007.motor2.patternCounter=0;

FVS007.userLimit=0;

```
FVS007.userSensor=0;

FVS007.co2.currentSample=0;

for(i=0;i<MAXSAMPLES;i++)

{

    FVS007.co2.samples[i]=0;

}

FVS007.co2.avgSample=0;

FVS007.co2.insertAt=0;

FVS007.co2.highLimit=1200;

FVS007.co2.lowLimit=350;

} // eo initialize_FVS
```

```
ADCON0bits.GO=TRUE;
```

```
while(ADCON0bits.GO);
```

```
return ADRES;
```

```
}// eo getAdcSample
```

```
/*---getTemp -----
```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Convert the ADC Sample to temperature value by adc resolution

Input:        Takes adc sample as input from the temperature channel

Returns:      Returns the Temperature Value

-----\*/

```
int getTemp ()
```

```
{
```

```
    float adcResultTemp=0;
```

```
    adcResultTemp=getAdcSample(TEMP);
```

```
    adcResultTemp/=ADC_B;                                // dividing the sample  
by 1024
```

```
    adcResultTemp*=VOLTAGE;                                // multiply the sample  
to 5
```

```
    adcResultTemp-=OFFSET;
```

```
    adcResultTemp/=COEFF;
```

```
    return adcResultTemp;                                // after resolution and solving  
temp, result is returning
```

```
}// eo getTemp
```

/\*---getPres -----

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Convert the adc sample to pressure value by adc resolution

Input:        Takes adc sample as input from the pressure channel

Returns:       Returns the Pressure Value

-----\*/

int getPres ()

{

    float adcResultPres=0;

    adcResultPres=getAdcSample(PRES);

```
        adcResultPres/=ADC_B;                                // dividing the sample
by 1024
```

```
        adcResultPres*=VOLTAGE;                                // multiply the sample
to 5
```

```
        adcResultPres/=COEFF_P;
```

```
        return adcResultPres;
```

```
    }// eo getPres
```

```
/*---getCo2 -----
```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Convert the adc sample to co2 value by adc resolution

Input:        Takes adc sample as input from the pressure channel

Returns:      Returns the Co2 Value

-----\*/

int getCo2()

{

float adcResultCo2=0;

adcResultCo2=getAdcSample(CO2);

adcResultCo2/=ADC\_B;

adcResultCo2\*=VOLTAGE;

adcResultCo2+=OFFSET\_C;

adcResultCo2/=COEFF\_C;

return adcResultCo2;

}// eo getCo2 ::

/\*---calTxChecksum -----

Author: SRaj



Date: 27/03/2020

Modified: Name or None

Desc: checksum is created by using + operator for the trasmission string

Input: Take the input of buffer

Returns: returns the checksum value

-----\*/

```
char calTxChecksum(char *ptr)
```

```
{
```

```
    char cs2=0;
```

```
    while(*ptr)
```

```
    {
```

```
        cs2+=*ptr;
```

```
        ptr++;
```

```
}
```

```
return cs2;
```

```
} // eo calChecksum
```

```
/*---avgTemp -----
```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Calculate the average of Temp channel

Input:        None

Returns:      Returns average value

```
-----*/
```

```
int avgTemp()
```

```
{
```

```
int tempAvg=0;

FVS007.temperature.currentSample=getTemp();

FVS007.temperature.samples[FVS007.temperature.insertAt]=FVS007.temperature.currentSampl
e;

FVS007.temperature.avgSample+=FVS007.temperature.samples[FVS007.temperature.insertAt];

FVS007.temperature.insertAt++;

if(FVS007.temperature.insertAt>=MAXSAMPLES)

{

    FVS007.temperature.insertAt=0;

    FVS007.temperature.avgSample/=MAXSAMPLES;

    tempAvg=FVS007.temperature.avgSample;

    return tempAvg;

}
```

```
} // eo avgTemp
```

```
/*---avgPres -----
```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Calculate the average of Pressure channel

Input:        None

Returns:      Returns average value

```
-----*/
```

```
int avgPres()
```

```
{
```

```
    int presAvg=0;
```

```
FVS007.pressure.currentSample=getPres();
```

```
FVS007.pressure.samples[FVS007.pressure.insertAt]=FVS007.temperature.currentSample;
```

```
FVS007.pressure.avgSample+=FVS007.pressure.samples[FVS007.pressure.insertAt];
```

```
FVS007.pressure.insertAt++;
```

```
if(FVS007.pressure.insertAt>=MAXSAMPLES)
```

```
{
```

```
    FVS007.pressure.insertAt=0;
```

```
    FVS007.pressure.avgSample/=MAXSAMPLES;
```

```
    presAvg=FVS007.pressure.avgSample;
```

```
    return presAvg;
```

```
}
```

```
// eo avgPres
```

/\*---avgCo2 -----

Author: SRaj

Date: 26/03/2020

Modified: Name or None

Desc: Calculate the average of CO2 channel

Input: None

Returns: Returns average value

-----\*/

int avgCo2()

{

int co2Avg=0;

FVS007.co2.currentSample=getCo2();

FVS007.co2.samples[FVS007.co2.insertAt]=FVS007.co2.currentSample;

```
FVS007.co2.avgSample+=FVS007.co2.samples[FVS007.co2.insertAt];
```

```
FVS007.co2.insertAt++;
```

```
if(FVS007.co2.insertAt>=MAXSAMPLES)
```

```
{
```

```
    FVS007.co2.insertAt=0;
```

```
    FVS007.co2.avgSample/=MAXSAMPLES;
```

```
    co2Avg=FVS007.co2.avgSample;
```

```
    return co2Avg;
```

```
}
```

```
// eo avyCo2
```

```
/*---motorStart -----
```

Author: SRaj

Date: 26/03/2020

Modified: Name or None

Desc: To Start the motor.

Input: Type and purpose of input arguments

Returns: Type and purpose of returning argument

-----\*/

```
void motorStart(void)
```

```
{
```

```
    FVS007.motor1.position=MOTOR_P;
```

```
        if(FVS007.motor1.currentPosition!=FVS007.motor1.position ||  
FVS007.motor1.currentPosition<FVS007.motor1.position)
```

```
    {
```

```
        FVS007.motor1.pattern=mtrPattern[FVS007.motor1.patternCounter];
```



```
FVS007.motor1.currentPosition+=3;
```

```
FVS007.motor1.patternCounter++;
```

```
if(FVS007.motor1.patternCounter>=PATTERNS)
```

```
{
```

```
    FVS007.motor1.patternCounter=0;
```

```
}
```

```
if(FVS007.motor1.currentPosition>=MAX_CP)
```

```
{
```

```
    FVS007.motor1.currentPosition=0;
```

```
}
```

```
}//eo if
```

```
} //eo motorStart
```

```
void motorStop()
```

```
{
```

```
    FVS007.motor1.position=FVS007.motor1.currentPosition;
```

```
}
```

```
/*---motor2Start -----
```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         To open the Dampener

Input:        None

Returns:      None

```
-----*/
```

```
void motor2Start()
```

```
{
```

```
    FVS007.motor2.position=MOTOR2_P;
```

```
    if(FVS007.motor2.currentPosition!=FVS007.motor2.position ||  
FVS007.motor2.currentPosition<FVS007.motor2.position)
```

```
    {
```

```
        FVS007.motor2.pattern=mtr2Pattern[FVS007.motor2.patternCounter];
```

```
        FVS007.motor2.currentPosition+=3;
```

```
        FVS007.motor2.patternCounter++;
```

```
        if(FVS007.motor2.patternCounter>=PATTERNS)
```

```
        {
```

```
            FVS007.motor2.patternCounter=0;
```

```
        }
```

}

}// eo motor2Start

/\*---motor2Stop -----

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         To close the Dampener

Input:        None

Returns:      None

-----\*/

void motor2Stop()

{

```
FVS007.motor2.position=0;

    if(FVS007.motor2.currentPosition!=FVS007.motor2.position ||
FVS007.motor2.currentPosition>FVS007.motor2.position)

    {

        FVS007.motor2.pattern=mtr2Pattern[FVS007.motor2.patternCounter];

        FVS007.motor2.currentPosition-=3;

        FVS007.motor2.patternCounter++;

        if(FVS007.motor2.patternCounter>=PATTERNS)

        {

            FVS007.motor2.patternCounter=0;

        }

    }

}

} // eo motor2Stop
```

/\*---co2Operation -----

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         control the operation of CO2

Input:        None

Returns:      None

-----\*/

void co2Operation()

{

    if(currentCo2>FVS007.co2.highLimit)

    {

        motor2Start();

        if(FVS007.motor2.patternCounter==0)

```
{  
  
    MTR2_LED1=LED_ON;  
  
    MTR2_LED2=LED_OFF;  
  
    MTR2_LED3=LED_OFF;  
  
    MTR2_LED4=LED_OFF;  
  
}
```

```
if(FVS007.motor2.patternCounter==1)
```

```
{  
  
    MTR2_LED1=LED_OFF;  
  
    MTR2_LED2=LED_ON;  
  
    MTR2_LED3=LED_OFF;  
  
    MTR2_LED4=LED_OFF;  
  
}
```

```
if(FVS007.motor2.patternCounter==2)
```

```
{
```

```
    MTR2_LED1=LED_OFF;
```

```
    MTR2_LED2=LED_OFF;
```

```
    MTR2_LED3=LED_ON;
```

```
    MTR2_LED4=LED_OFF;
```

```
}
```

```
if(FVS007.motor2.patternCounter==3)
```

```
{
```

```
    MTR2_LED1=LED_OFF;
```

```
    MTR2_LED2=LED_OFF;
```

```
    MTR2_LED3=LED_OFF;
```

```
    MTR2_LED4=LED_ON;
```



```
}
```

```
}
```

```
if(currentCo2<FVS007.co2.highLimit && currentCo2>FVS007.co2.lowLimit)
```

```
{
```

```
    motor2Stop();
```

```
    if(FVS007.motor2.patternCounter==0)
```

```
    {
```

```
        MTR2_LED1=LED_ON;
```

```
        MTR2_LED2=LED_OFF;
```

```
        MTR2_LED3=LED_OFF;
```

```
        MTR2_LED4=LED_OFF;
```

```
    }
```

```
if(FVS007.motor2.patternCounter==1)
```

```
{
```

```
    MTR2_LED1=LED_OFF;
```

```
    MTR2_LED2=LED_ON;
```

```
    MTR2_LED3=LED_OFF;
```

```
    MTR2_LED4=LED_OFF;
```

```
}
```

```
if(FVS007.motor2.patternCounter==2)
```

```
{
```

```
    MTR2_LED1=LED_OFF;
```

```
    MTR2_LED2=LED_OFF;
```

```
    MTR2_LED3=LED_ON;
```

```
    MTR2_LED4=LED_OFF;
```

```
}
```

```
if(FVS007.motor2.patternCounter==3)
```

```
{
```

```
    MTR2_LED1=LED_OFF;
```

```
    MTR2_LED2=LED_OFF;
```

```
    MTR2_LED3=LED_OFF;
```

```
    MTR2_LED4=LED_ON;
```

```
}
```

```
if(FVS007.motor2.currentPosition==FVS007.motor2.position)
```

```
{
```

```
    MTR2_LED1=LED_OFF;
```

```
    MTR2_LED2=LED_OFF;
```

```
    MTR2_LED3=LED_OFF;
```

```
MTR2_LED4=LED_OFF;
```

```
}
```

```
}
```

```
}// eo co2Operation
```

```
/*---controlOperation -----
```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Controls the operation of motor

Input:        Type and purpose of input arguments

Returns:      Type and purpose of returning argument

```
-----*/
```

```
void controlOperation (void)
```

```
{
```

```
if (currentTemperature>FVS007.temperature.lowLimit &&  
currentTemperature<FVS007.temperature.highLimit)
```

```
{
```

```
    motorStop();
```

```
        MTR_LED1=LED_OFF;
```

```
        MTR_LED2=LED_OFF;
```

```
        MTR_LED3=LED_OFF;
```

```
        MTR_LED4=LED_OFF;
```

```
if(currentPressure<FVS007.pressure.highLimit)
```

```
{
```

```
    HEATER=LED_OFF;
```

```
}
```

```
else
```

```
{
```

```
    HEATER=LED_ON;
```

```
}
```

```
//eo if
```

```
    if (currentTemperature>FVS007.temperature.highLimit ||  
currentPressure<FVS007.pressure.highLimit)
```

```
{
```

```
    motorStart();
```

```
    HEATER=LED_OFF;
```

```
    if (FVS007.motor1.patternCounter==0)
```

```
{
```

```
        MTR_LED1=LED_ON;
```

```
MTR_LED2=LED_OFF;
```

```
MTR_LED3=LED_OFF;
```

```
MTR_LED4=LED_OFF;
```

```
}
```

```
else if (FVS007.motor1.patternCounter==1)
```

```
{
```

```
MTR_LED1=LED_OFF;
```

```
MTR_LED2=LED_ON;
```

```
MTR_LED3=LED_OFF;
```

```
MTR_LED4=LED_OFF;
```

```
}
```

```
else if (FVS007.motor1.patternCounter==2)
```

```
{  
  
    MTR_LED1=LED_OFF;  
  
    MTR_LED2=LED_OFF;  
  
    MTR_LED3=LED_ON;  
  
    MTR_LED4=LED_OFF;  
  
}
```

else

```
{  
  
    MTR_LED1=LED_OFF;  
  
    MTR_LED2=LED_OFF;  
  
    MTR_LED3=LED_OFF;  
  
    MTR_LED4=LED_ON;  
  
}
```



```
} //eo if
```

```
        if(currentTemperature>FVS007.temperature.highLimit &&  
currentPressure<FVS007.pressure.highLimit)
```

```
{
```

```
    motorStart();
```

```
    HEATER=LED_OFF;
```

```
    if (FVS007.motor1.patternCounter==0)
```

```
{
```

```
    MTR_LED1=LED_ON;
```

```
    MTR_LED2=LED_OFF;
```

```
    MTR_LED3=LED_OFF;
```

```
    MTR_LED4=LED_OFF;
```

```
}
```

```
else if (FVS007.motor1.patternCounter==1)
```

```
{
```

```
    MTR_LED1=LED_OFF;
```

```
    MTR_LED2=LED_ON;
```

```
    MTR_LED3=LED_OFF;
```

```
    MTR_LED4=LED_OFF;
```

```
}
```

```
else if (FVS007.motor1.patternCounter==2)
```

```
{
```

```
    MTR_LED1=LED_OFF;
```

```
    MTR_LED2=LED_OFF;
```

```
    MTR_LED3=LED_ON;
```

```
MTR_LED4=LED_OFF;
```

```
}
```

```
else
```

```
{
```

```
MTR_LED1=LED_OFF;
```

```
MTR_LED2=LED_OFF;
```

```
MTR_LED3=LED_OFF;
```

```
MTR_LED4=LED_ON;
```

```
}
```

```
}//eo if
```

```
if (currentTemperature<FVS007.temperature.lowLimit)
```

```
{
```

```
motorStop();
```

```
    MTR_LED1=LED_OFF;
```

```
    MTR_LED2=LED_OFF;
```

```
    MTR_LED3=LED_OFF;
```

```
    MTR_LED4=LED_OFF;
```

```
if(currentPressure<FVS007.pressure.highLimit)
```

```
{
```

```
    HEATER=LED_OFF;
```

```
}
```

```
else
```

```
{
```

```
    HEATER=LED_ON;
```

```
}
```

```
} //eo if
```

```
if (currentPressure<FVS007.pressure.highLimit && currentPressure>FVS007.pressure.lowLimit)
```

```
{
```

```
    motorStop();
```

```
        MTR_LED1=LED_OFF;
```

```
        MTR_LED2=LED_OFF;
```

```
        MTR_LED3=LED_OFF;
```

```
        MTR_LED4=LED_OFF;
```

```
    if (HEATER==LED_ON)
```

```
    {
```

```
        HEATER=LED_ON;
```

```
    }
```

```
else
```

```
{
```

```
    HEATER=LED_OFF;
```

```
}
```

```
// eo if
```

```
if (currentPressure>FVS007.pressure.lowLimit)
```

```
{
```

```
    motorStop();
```

```
    MTR_LED1=LED_OFF;
```

```
    MTR_LED2=LED_OFF;
```

```
    MTR_LED3=LED_OFF;
```

```
    MTR_LED4=LED_OFF;
```

```
        if (HEATER==LED_ON)

        {

                HEATER=LED_ON;

        }

        else

        {

                HEATER=LED_OFF;

        }

} //eo if


} // eo controlOperation


/*---MtrUpdate -----
```

Author: SRaj

Date: 16/04/2020

Modified: Name or None

Desc: Motor flag will be true if motor will start or stop in any condition

Input: Type and purpose of input arguments

Returns: Type and purpose of returning argument

-----\*/

```
void mtrUpdate(void)
```

```
{
```

```
    if(currentTemperature>FVS007.temperature.highLimit ||  
    currentPressure<FVS007.pressure.highLimit)
```

```
    {
```

```
        if(currentCo2>FVS007.co2.highLimit)
```

```
        {
```



```
mtr1StartFlag=TRUE;
```

```
mtr2StartFlag=TRUE;
```

```
}
```

```
else
```

```
{
```

```
mtr1StartFlag=TRUE;
```

```
mtr2StopFlag=TRUE;
```

```
}
```

```
}
```

```
else
```

```
{
```

```
if(currentCo2>FVS007.co2.highLimit)
```

```
{

    mtr1StopFlag=TRUE;

    mtr2StartFlag=TRUE;

}

else

{

    mtr1StopFlag=TRUE;

    mtr2StopFlag=TRUE;

}

}

}

/*---motorSend -----
```

Author:       SRaj

Date: 16/04/2020

Modified: Name or None

Desc: Motor Sentence will be formed and send it to TX2

Input: Type and purpose of input arguments

Returns: Type and purpose of returning argument

-----\*/

```
void motorSend(void)
```

```
{
```

```
    if(mtr1StartFlag==TRUE && mtr2StartFlag==TRUE)
```

```
    {
```

```
        sprintf(mtrBuf,"$MTRUPD,%i,%i,%i,%i,%i,%i,%i,%i\0",CONTROLLER,FVS,MTR1,MTRON,FVS007.m  
otor1.currentPosition,MTR2,MTRON,FVS007.motor2.currentPosition);
```

```
        sprintf(mtrBuf,"%s,%i#\0",mtrBuf,calTxChecksum(mtrBuf));
```

```

        puts2USART(mtrBuf);

        mtr1StartFlag=FALSE;

        mtr2StartFlag=FALSE;

    }

    else if (mtr1StartFlag==TRUE && mtr2StopFlag==TRUE)

    {

        sprintf(mtrBuf,"$MTRUPD,%i,%i,%i,%i,%i,%i,%i,%i\n",CONTROLLER,FVS,MTR1,MTRON,FVS007.m
otor1.currentPosition,MTR2,MTROFF,FVS007.motor2.currentPosition);

        sprintf(mtrBuf,"%s,%i#\n",mtrBuf,calTxChecksum(mtrBuf));

        puts2USART(mtrBuf);

        mtr1StartFlag=FALSE;

        mtr2StopFlag=FALSE;

    }

    else if (mtr1StopFlag==TRUE && mtr2StartFlag==TRUE)

    {

```

```
    sprintf(mtrBuf,"$MTRUPD,%i,%i,%i,%i,%i,%i,%i,%i\0",CONTROLLER,FVS,MTR1,MTROFF,FVS007.  
motor1.currentPosition,MTR2,MTRON,FVS007.motor2.currentPosition);
```

```
    sprintf(mtrBuf,"%s,%i#\0",mtrBuf,calTxChecksum(mtrBuf));
```

```
    puts2USART(mtrBuf);
```

```
    mtr1StopFlag=FALSE;
```

```
    mtr2StartFlag=FALSE;
```

```
}
```

```
else
```

```
{
```

```
    sprintf(mtrBuf,"$MTRUPD,%i,%i,%i,%i,%i,%i,%i,%i\0",CONTROLLER,FVS,MTR1,MTROFF,FVS007.  
motor1.currentPosition,MTR2,MTROFF,FVS007.motor2.currentPosition);
```

```
    sprintf(mtrBuf,"%s,%i#\0",mtrBuf,calTxChecksum(mtrBuf));
```

```
    puts2USART(mtrBuf);
```

```
    mtr1StopFlag=FALSE;
```

```
    mtr2StopFlag=FALSE;
```

```
}
```

```
}
```

```
/*---ledControl -----
```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Controls the led of the sensors

Input:        Type and purpose of input arguments

Returns:      Type and purpose of returning argument

```
-----*/
```

```
void ledControl()
```

```
{
```

```
if (currentTemperature>FVS007.temperature.highLimit &&  
currentPressure<FVS007.pressure.highLimit)
```

```
{
```

```
PORTBbits.RB1=LED_ON; // when pressure is  
high
```

```
PORTBbits.RB2=LED_OFF;
```

```
PORTBbits.RB3=LED_OFF;
```

```
PORTBbits.RB4=LED_ON; // when temperature is  
high
```

```
}
```

```
else if (currentTemperature<FVS007.temperature.lowLimit &&  
currentPressure>FVS007.pressure.lowLimit) // when pressure and temperature is low
```

```
{
```

```

        PORTBbits.RB1=LED_OFF;

        PORTBbits.RB2=LED_ON;                                // when pressure is low

        PORTBbits.RB3=LED_ON;                                // when temperature is
low
        PORTBbits.RB4=LED_OFF;

    }

    else if (currentTemperature>FVS007.temperature.highLimit &&
currentPressure>FVS007.pressure.lowLimit)    // when pressure is low and temperature is high

    {

        PORTBbits.RB1=LED_OFF;

        PORTBbits.RB2=LED_ON;

        PORTBbits.RB3=LED_OFF;

        PORTBbits.RB4=LED_ON;

    }

```



```
    else if (currentPressure<FVS007.pressure.highLimit &&  
currentTemperature<FVS007.temperature.lowLimit)    // when pressure is high and temperature is low
```

```
{
```

```
    PORTBbits.RB1=LED_ON;
```

```
    PORTBbits.RB2=LED_OFF;
```

```
    PORTBbits.RB3=LED_ON;
```

```
    PORTBbits.RB4=LED_OFF;
```

```
}
```

```
    else if (currentTemperature<FVS007.temperature.highLimit &&  
currentTemperature>FVS007.temperature.lowLimit && currentPressure<FVS007.pressure.highLimit) //  
when pressure is high
```

```
{
```

```
    PORTBbits.RB1=LED_ON;
```

```
PORTBbits.RB2=LED_OFF;
```

```
PORTBbits.RB3=LED_OFF;
```

```
PORTBbits.RB4=LED_OFF;
```

```
}
```

```
    else if (currentTemperature<FVS007.temperature.highLimit &&  
currentTemperature>FVS007.temperature.lowLimit && currentPressure>FVS007.pressure.lowLimit) //  
when pressure is low
```

```
{
```

```
    PORTBbits.RB1=LED_OFF;
```

```
    PORTBbits.RB2=LED_ON;
```

```
    PORTBbits.RB3=LED_OFF;
```

```
    PORTBbits.RB4=LED_OFF;
```

```
}
```

```
    else if (currentPressure>FVS007.pressure.highLimit &&  
currentPressure<FVS007.pressure.lowLimit && currentTemperature>FVS007.temperature.highLimit) //  
when temperature is high
```

```
{
```

```
    PORTBbits.RB1=LED_OFF;
```

```
    PORTBbits.RB2=LED_OFF;
```

```
    PORTBbits.RB3=LED_OFF;
```

```
    PORTBbits.RB4=LED_ON;
```

```
}
```

```
    else if (currentPressure>FVS007.pressure.highLimit &&  
currentPressure<FVS007.pressure.lowLimit && currentTemperature<FVS007.temperature.lowLimit) //  
when temperature is low
```

```
{
```

```
    PORTBbits.RB1=LED_OFF;
```

```
PORTBbits.RB2=LED_OFF;
```

```
PORTBbits.RB3=LED_ON;
```

```
PORTBbits.RB4=LED_OFF;
```

```
}
```

```
else
```

```
{
```

```
PORTBbits.RB1=LED_OFF;
```

```
PORTBbits.RB2=LED_OFF;
```

```
PORTBbits.RB3=LED_OFF;
```

```
PORTBbits.RB4=LED_OFF;
```

```
}// eo else..if
```

```
}// eo ledcontrol
```

/\*--- diaplay -----

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Display the output to terminal window

Input:        Type and purpose of input arguments

Returns:      Type and purpose of returning argument

-----\*/

void display (void)

{

    printf("\033[2J\033[H\033[0");

    printf("\rFFVS007 System Properties\n");

    printf("\r\n");

```
if (FVS007.userLimit==1 && FVS007.userSensor==1)

{

    printf("\rMODE: HIGH \t\t CHANNEL: TEMPERATURE \n ");

}
```

```
if (FVS007.userLimit==0 && FVS007.userSensor==1)

{

    printf("\rMODE: LOW \t\t CHANNEL: TEMPERATURE\n");

}
```

```
if (FVS007.userLimit==1 && FVS007.userSensor==0)

{

    printf("\rMODE: HIGH \t\t CHANNEL: PRESSURE\n");

}
```

```
if (FVS007.userLimit==0 && FVS007.userSensor==0)
```

```

{

    printf("\r MODE: LOW \t\t CHANNEL: PRESSURE\n");

}

if(FVS007.userLimit==1 && FVS007.userSensor==2)

{

    printf("\rMODE: HIGH \t\t CHANNEL: CO2\n");

}

if(FVS007.userLimit==0 && FVS007.userSensor==2)

{

    printf("\rMODE: LOW \t\t CHANNEL: CO2\n");

}

printf("\n\rTemperature\t\t\tPressure\t\tCO2\n");

printf("\rCurrent:\t%i%cC\t\tCurrent:  %i kPa \tCurrent:  %d ppm\n",
currentTemperature,248,currentPressure,currentCo2); //for printing current value of temperature and
pressure

```

```

        printf("\rHighLimit:\t%i%C\t\tHighLimit: %i kPa\tHighLimit: %i ppm\n",
FVS007.temperature.highLimit,248,FVS007.pressure.highLimit,FVS007.co2.highLimit);           // for
printing high limit of all the sensors

```

```

        printf("\rLowLimit:\t%i%C\t\tLowLimit: %i kPa\tLowLimit: %i ppm\n",
FVS007.temperature.lowLimit,248,FVS007.pressure.lowLimit,FVS007.co2.lowLimit);
        // for printing low limit of both the sesnsors

```

```

        if(((currentTemperature>FVS007.temperature.highLimit) || (currentTemperature<FVS007.tempe
rature.lowLimit)) &&
((currentPressure<FVS007.pressure.highLimit) || (currentPressure>FVS007.pressure.lowLimit)) &&
((currentCo2>FVS007.co2.highLimit) || (currentCo2<FVS007.co2.lowLimit)))

```

```

        {

```

```

                printf("\r\033[0;37mStatus : \033[0;31mAlarm\t\t\t \033[0;37mStatus:
\033[0;31mAlarm\033[0;37m\t\t \033[0;37mStatus: \033[0;31mAlarm\033[0;37m\n ");
                // for printing temperature and pressure status

```

```

        }

```

```

        else
        if(((currentTemperature>FVS007.temperature.highLimit) || (currentTemperature<FVS007.temperature.l
owLimit)) &&
((currentPressure>FVS007.pressure.highLimit) || (currentPressure<FVS007.pressure.lowLimit)) &&
((currentCo2>FVS007.co2.highLimit) || (currentCo2<FVS007.co2.lowLimit)))

```

```

        {

```



```
        printf("\r\033[0;37mStatus : \033[0;31mAlarm\t\t\t \033[0;37mStatus:  
\033[0;32mSafe\033[0;37m\t\t \033[0;37mStatus: \033[0;31mAlarm\033[0;37m\n");
```

```
    }
```

```
        else
```

```
if(((currentTemperature<FVS007.temperature.highLimit) || (currentTemperature>FVS007.temperature.l  
owLimit)) &&  
((currentPressure<FVS007.pressure.highLimit) || (currentPressure>FVS007.pressure.lowLimit)) &&  
((currentCo2>FVS007.co2.highLimit) || (currentCo2<FVS007.co2.lowLimit)))
```

```
{
```

```
        printf("\r\033[0;37mStatus : \033[0;32mSafe\t\t\t \033[0;37mStatus:  
\033[0;31mAlarm\033[0;37m\t\t \033[0;37mStatus: \033[0;31mAlarm\033[0;37m\n");
```

```
    }
```

```
        else if ((currentTemperature<FVS007.temperature.highLimit &&  
currentTemperature>FVS007.temperature.lowLimit) && (currentPressure>FVS007.pressure.highLimit  
&& currentPressure<FVS007.pressure.lowLimit) && ((currentCo2>FVS007.co2.highLimit) ||  
(currentCo2<FVS007.co2.lowLimit)))
```

```
{
```

```
        printf("\r\033[0;37mStatus : \033[0;32mSafe\t\t\t \033[0;37mStatus:  
\033[0;32mSafe\033[0;37m\t\t \033[0;37mStatus: \033[0;31mAlarm\033[0;37m\n");
```

```
}
```

```
else if(((currentTemperature>FVS007.temperature.highLimit) ||  
(currentTemperature<FVS007.temperature.lowLimit)) &&  
((currentPressure>FVS007.pressure.lowLimit) || (currentPressure<FVS007.pressure.highLimit)) &&  
(currentCo2>FVS007.co2.lowLimit && currentCo2<FVS007.co2.highLimit))
```

```
{
```

```
printf("\r\033[0;37mStatus : \033[0;31mAlarm\t\t\t \033[0;37mStatus:  
\033[0;31mAlarm\033[0;37m\t\t\t \033[0;37mStatus: \033[0;32mSafe\033[0;37m\n");
```

```
}
```

```
else  
if(((currentTemperature>FVS007.temperature.highLimit) || (currentTemperature<FVS007.temperature.l  
owLimit)) &&  
((currentPressure>FVS007.pressure.highLimit) || (currentPressure<FVS007.pressure.lowLimit)) &&  
((currentCo2<FVS007.co2.highLimit && currentCo2>FVS007.co2.lowLimit)))
```

```
{
```

```
printf("\r\033[0;37mStatus : \033[0;31mAlarm\t\t\t \033[0;37mStatus:  
\033[0;32mSafe\033[0;37m\t\t\t \033[0;37mStatus: \033[0;32mSafe\033[0;37m\n");
```

```
}
```

```
else  
if(((currentTemperature<FVS007.temperature.highLimit) || (currentTemperature>FVS007.temperature.l  
owLimit)) &&  
((currentPressure<FVS007.pressure.highLimit) || (currentPressure>FVS007.pressure.lowLimit)) &&  
((currentCo2<FVS007.co2.highLimit && currentCo2>FVS007.co2.lowLimit)))
```

```

{

    printf("\r\033[0;37mStatus : \033[0;32mSafe\t\t\t \033[0;37mStatus:
\033[0;31mAlarm\033[0;37m\t\t \033[0;37mStatus: \033[0;32mSafe\033[0;37m\n");

}

else

{

    printf("\r\033[0;37mStatus : \033[0;32mSafe\t\t\t \033[0;37mStatus:
\033[0;32mSafe\033[0;37m\t\t \033[0;37mStatus: \033[0;32mSafe\033[0;37m\n");

}

printf("\n");

printf("\rMOTOR1: Discharge\t\tMOTOR2: Dampener\n");

printf("\rPosition: %i\t\t\tPosition:
%i\n",FVS007.motor1.currentPosition,FVS007.motor2.currentPosition);

printf("\rData:\t 0X0%i\t\t\tData:\t
0X0%i\n",FVS007.motor1.pattern,FVS007.motor2.pattern);

mtrUpdate();

motorSend();

```

```
} //eo display
```

```
/*---calChecksum -----
```

Author:       SRaj

Date:         26/03/2020

Modified:     Name or None

Desc:         Calculate the check sum of the string

Input:        Pointer to the string

Returns:      Return checksum value

```
-----*/
```

```
char calChecksum(char *ptr)
```

```
{
```

```
    char cs=0;
```

```
    while(*ptr)
```

```

{

    cs^=*ptr;

    ptr++;

}

return cs;

} // eo calChecksum

/*---ISR -----
```

Returns:       Type and purpose of returning argument

-----\*/

```
void ISR();
```

```
#pragma code int_vector=0x008
```

```
void int_vector()
```

```
{
```

```
    _asm
```

```
        GOTO ISR
```

```
    _endasm
```

```
}
```

```
#pragma code
```

/\*---collectSentence -----

Author:       SRaj

Date:         03/04/2020

Modified:     Name or None

Desc:         collect the serial communication sentence from the mbed

Input:        Type and purpose of input arguments

Returns:      Type and purpose of returning argument

-----\*/

void collectSentence(char \*ptr)

{

    char hold=0;

    if(PIR3bits.RC2IF)

    {

```
hold=RCREG2;
```

```
if(hold=='$')
```

```
{
```

```
    *ptr=hold;
```

```
    while(hold!='#')
```

```
    {
```

```
        if(PIR3bits.RC2IF)
```

```
        {
```

```
            hold=RCREG2;
```

```
            ptr++;
```

```
            *ptr=hold;
```

```
            if (hold=='#')
```

```
            {
```

```
                ptr++;
```

```
                *ptr = 0x00;
```



```
sentenceRdy= TRUE;
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
} //eo collectSentence
```

```
#pragma interrupt ISR
```

```
void ISR(void)
```

```
{
```

```
    if(PIR3bits.RC2IF)
```

```
    {
```

```
PIE3bits.RC2IE=0;
```

```
collectSentence(buf);
```

```
PIE3bits.RC2IE=1;
```

```
}
```

```
} // eo ISR
```

```
/*---validateSentence-----
```

Author:       SRaj

Date:         03/04/2020

Modified:     Name or None

Desc:         Validate the received command sentence

Input:        buffer as input

Returns:      TRUE or FALSE

-----\*/

```
char validateSentence (char *ptr)
```

```
{
```

```
    char rcs=0;
```

```
    char ncs=0;
```

```
    char csFlag=FALSE;
```

```
    int count=strlen(ptr);
```

```
    while(!csFlag)
```

```
    {
```

```
        if(*(ptr+count)=='#')
```

```
        {
```

```
            *(ptr+count)=0X00;
```

```
        }
```

```
    if(*(ptr+count)=='  
  
    {  
  
        *(ptr+count)=0X00;  
  
        rcs=atoi(ptr+count+1);  
  
        csFlag=TRUE;  
  
    }  
  
    count--;  
  
}  
  
ncs=calChecksum(ptr);  
  
if(ncs==rcs)  
  
    {  
  
        return TRUE;  
  
    }  
  
else  
  
    {
```

```
return FALSE;
```

```
}
```

```
} //eo validateSentence
```

```
/*---purseSentence -----
```

Author:       SRaj

Date:         03/04/2020

Modified:     Name or None

Desc:         purse the sentence and save each command in token

Input:        buffer as input

Returns:      Type and purpose of returning argument

```
-----*/
```

```
void purseSentence(char *ptr)
```

```
{
```

```
    int tokenCount=0;
```

```
    while(*ptr)
```

```
    {
```

```
        if(*ptr=='$' || *ptr==',')
```

```
        {
```

```
            *ptr=0X00;
```

```
            tokens[tokenCount]=ptr+1;
```

```
            tokenCount++;
```

```
        }
```

```
        ptr++;
```

```
    }
```

```
} // eo PurseSentence
```

```
/*---executeSentence -----
```

Author:       SRaj

Date:         03/04/2020

Modified:     Name or None

Desc:         execute the recieved sentence

Input:        Type and purpose of input arguments

Returns:      Type and purpose of returning argument

```
-----*/
```

```
void executeSentence()
```

```
{
```

```
if(atoi(tokens[1])==ADDYTO)
```

```
{
```

```
if(atoi(tokens[2])==ADDYFM)
```

```
{
```

```
if(strcmp(tokens[0],string1)==0)
```

```
{
```

```
if(strcmp(tokens[3],string2)==0)
```

```
{
```

```
if(strcmp(tokens[4],string3)==0)
```

```
{
```



```
FVS007.temperature.highLimit=atoi(tokens[5]);
```

```
FVS007.userLimit=1;
```

```
FVS007.userSensor=1;
```

```
}
```

```
else
```

```
{
```

```
FVS007.temperature.lowLimit=atoi(tokens[5]);
```

```
FVS007.userLimit=0;
```

```
FVS007.userSensor=1;
```

```
}
```

```
}
```

```
if(strcmp(tokens[3],string5)==0)
```

```
{
```

```
if(strcmp(tokens[4],string3)==0)

{

    FVS007.pressure.highLimit=atoi(tokens[5]);

    FVS007.userLimit=1;

    FVS007.userSensor=0;

}

else

{

    FVS007.pressure.lowLimit=atoi(tokens[5]);

    FVS007.userLimit=0;

    FVS007.userSensor=0;

}

}

if(strcmp(tokens[3],string6)==0)

{
```

```
if(strcmp(tokens[4],string3)==0)

{

    FVS007.co2.highLimit=atoi(tokens[5]);

    FVS007.userLimit=1;

    FVS007.userSensor=2;

}

else

{

    FVS007.co2.lowLimit=atoi(tokens[5]);

    FVS007.userLimit=0;

    FVS007.userSensor=2;

}

}
```

```
}
```

```
}
```

```
} //eo executeSentence
```

```
/*---sendAvg -----
```

Author:       SRaj

Date:         08/04/2020

Modified:     Name or None

Desc:         when average value is calculated, command string is formed and send to TX2

Input:        Type and purpose of input arguments

Returns:      Type and purpose of returning argument

```
-----*/
```

```
void sendAvg (void)
```

```
{
```

```
if(count_s>=1)

{

    sprintf(txBuf,"$AVGUPD,%i,%i,T,%i\0",CONTROLLER,FVS,currentTemperature);

    sprintf(txBuf,"%s,%i#\0",txBuf,calTxChecksum(txBuf));

    puts2USART(txBuf);

}

if(count_s>=2)

{

    sprintf(txBuf,"$AVGUPD,%i,%i,P,%i\0",CONTROLLER,FVS,currentPressure);

    sprintf(txBuf,"%s,%i#\0",txBuf,calTxChecksum(txBuf));

    puts2USART(txBuf);

}

if(count_s>=3)

{
```

```
    sprintf(txBuf,"$AVGUPD,%i,%i,C,%i\0",CONTROLLER,FVS,currentCo2);
```

```
    sprintf(txBuf,"%s,%i#\0",txBuf,calTxChecksum(txBuf));
```

```
    puts2USART(txBuf);
```

```
    count_s=0;
```

```
}
```

```
// eo sendAvg::
```

```
/*--- MAIN FUNCTION -----
```

```
-----*/
```

```
void main ()
```

```
{
```

```
    initializeSystem();
```

```
    initialize_FVS();
```

```
    while (1)
```

```
    {
```

```
        if (TMROFLAG)
```

```
        {
```

```
            timeCounter++;
```

```
            co2Operation();
```

```
            if(timeCounter>=4)
```

```
            {
```

```
                timeCounter=0;
```

```
                count_s++;
```

```
                currentTemperature=avgTemp();
```

```
currentPressure=avgPres();
```

```
currentCo2=avgCo2();
```

```
sendAvg();
```

```
controlOperation();
```

```
ledControl();
```

```
display();
```

```
if(sentenceRdy==TRUE)
```

```
{
```

```
    sentenceRdy=FALSE;
```

```
    printf("\r\n%s\n",buf);
```

```
    validateSentence(buf);
```

```
    purseSentence(buf);
```

```
    executeSentence();
```



```
}//eo if
```

```
}// eo if
```

```
resetTMR0();
```

```
}//eo if
```

```
}// eo while
```

```
}
```

```
// MBED PROGRAM
```

```
//--- LAB 5B ---
```

```
/*-----
```

File Name: ELNC6007MSLAB5B

Author: SRaj

Date: 08/04/2020

Modified: Name or None

copyright Fanshawe College, 2019

Description: Receive motor command sentence from PIC microcontroller and execute the sentence

-----\*/

// Preprocessor -----

#include "mbed.h"

#include "Timer.h"

#include<string.h>

```
Serial pc(USBTX,USBRX,9600);
```

```
Serial fvs(p13,p14,19200);
```

```
DigitalIn pb1(p5);
```

```
DigitalIn pb2(p6);
```

```
DigitalIn pb3(p7);
```

```
DigitalIn pb4(p8);
```

```
Timer timer;
```

```
// Constants -----
```

```
#define TRUE      1
```

```
#define FALSE     0
```

```
#define HIGH      1
```

```
#define LOW       0
```

```
#define MAXSAMPLES 30
```

```
#define SIZE 4
```

```
#define BUFFERSIZE 30
```

```
#define CONTROLLER 1
```

```
#define FVS 7
```

```
#define TOKENSIZE 15
```

```
// Global Variables -----
```

```
char flag_I = FALSE;
```

```
char flag_D = FALSE;
```

```
int currentTemperature=0;
```

```
int currentPressure=0;
```

```
int currentCo2=0;
```

```
char timerCount=0;
```

```
char buf[BUFFERSIZE];
```

```
char index=0;
```

```
char sentRdy=FALSE;
```

```
char rxBuf[BUFFERSIZE];
```

```
char *ptr=rxBuf;
```

```
char *token[TOKENSIZE];
```

```
char string1[]="AVGUPD";
```

```
char string2[]="T";
```

```
char string3[]="P";
```

```
char string4[]="C";
```

```
char string5[]="MTRUPD";
```

```
char mtr1=1;
```

```
char mtr2=2;
```

```
char mtrOn=1;
```

```
char mtrOff=0;
```

```
typedef struct motor
```

```
{
```

```
    char position;
```

```
    int currentPosition;
```

```
    char pattern;
```

```
    char patternCounter;
```

```
    char status;
```

```
}mtr_t;
```

```
typedef struct sensorChannel
```

```
{
```

```
    int currentSample;
```

```
    int samples[MAXSAMPLES];
```

```
int avgSample;
```

```
char insertAt;
```

```
int highLimit;
```

```
int lowLimit;
```

```
char state;
```

```
}sensorCh_t;
```

```
typedef struct fermentationVat
```

```
{
```

```
char address[SIZE];
```

```
sensorCh_t temperature;
```

```
sensorCh_t pressure;
```

```
sensorCh_t co2;
```

```
char indicator;
```

```
char heater;
```

```
mtr_t motor1;
```

```
mtr_t motor2;
```

```
char userLimit;
```

```
char userSensor;
```

```
}FVS_t;
```

```
FVS_t FVS007;
```

```
/*--- initialize_FVS -----
```

Author: SRaj

Date: 27/03/2020



Modified: Name or None

Desc: Set the data member to initial value

Input: Type and purpose of input arguments

Returns: Type and purpose of returning argument

-----\*/

```
void initialize_FVS (void)
```

```
{
```

```
    int i;
```

```
    FVS007.address[0]='0';
```

```
    FVS007.address[1]='0';
```

```
    FVS007.address[2]='7';
```

```
    FVS007.temperature.currentSample=0;
```

```
    for(i=0;i<MAXSAMPLES;i++)
```

```
    {
```

```
FVS007.temperature.samples[i]=0;
```

```
}
```

```
FVS007.temperature.avgSample=0;
```

```
FVS007.temperature.insertAt=0;
```

```
FVS007.pressure.currentSample=0;
```

```
for(i=0;i<MAXSAMPLES;i++)
```

```
{
```

```
    FVS007.pressure.samples[i]=0;
```

```
}
```

```
FVS007.pressure.avgSample=0;
```

```
FVS007.pressure.insertAt=0;
```

```
FVS007.temperature.highLimit=65;
```

```
FVS007.pressure.highLimit=17;
```

```
FVS007.temperature.lowLimit=12;
```

FVS007.pressure.lowLimit=103;

FVS007.pressure.state=0;

FVS007.temperature.state=0;

FVS007.indicator=FALSE;

FVS007.heater=FALSE;

FVS007.motor1.position=0;

FVS007.motor1.currentPosition=0;

FVS007.motor1.pattern=0;

FVS007.motor1.patternCounter=0;

FVS007.motor2.position=0;

FVS007.motor2.currentPosition=0;

FVS007.motor2.pattern=0;

FVS007.motor2.patternCounter=0;

FVS007.motor1.status=0;

FVS007.motor2.status=0;

```
FVS007.userLimit=0;
```

```
FVS007.userSensor=0;
```

```
FVS007.co2.currentSample=0;
```

```
for(i=0;i<MAXSAMPLES;i++)
```

```
{
```

```
    FVS007.co2.samples[i]=0;
```

```
}
```

```
FVS007.co2.avgSample=0;
```

```
FVS007.co2.insertAt=0;
```

```
FVS007.co2.highLimit=1200;
```

```
FVS007.co2.lowLimit=350;
```

```
}// eo initialize_FVS
```

/\*---userMode-----

Author: SRaj

Date: 27/03/2020

Modified: None

Desc: User can select the mode.

Input: None

Returns: None

-----\*/

void userMode (void)

{

if(pb1==0)

{

FVS007.userLimit=!FVS007.userLimit;

}

```
} // eo userMode
```

```
/*---userChannel-----
```

Author: SRaj

Date: 27/03/2020

Modified: None

Desc: User can select the Channel.

Input: None

Returns: None

```
-----*/
```

```
void userChannel(void)
```

```
{
```

```
    if(pb2==LOW)
```

```
{  
  
    FVS007.userSensor++;  
  
}  
  
if(FVS007.userSensor>=3)  
  
{  
  
    FVS007.userSensor=0;  
  
}  
  
} // eo userChannel  
  
  
/*---calChecksum -----
```

Author: SRaj

Date: 27/03/2020

Modified: Name or None

Desc: Calculate the check sum of the string

Input: Pointer to the string

Returns: Return checksum value

-----\*/

```
char calChecksum(char *ptr)
```

```
{
```

```
    char cs=0;
```

```
    while(*ptr)
```

```
    {
```

```
        cs^=*ptr;
```

```
        ptr++;
```

```
    }
```

```
    return cs;
```

```
} // eo calCheckSum
```



/\*---increment -----

Author: SRaj

Date: 27/03/2020

Modified: None

Desc: User can increase the limit of sensor

Input: None

Returns: None

-----\*/

void increment(void)

{

if(pb3==LOW)

{

```
if (FVS007.userLimit==HIGH && FVS007.userSensor==1)

{

    FVS007.temperature.highLimit++;

    flag_I=TRUE;

}

else if (FVS007.userLimit==LOW && FVS007.userSensor==1)

{

    FVS007.temperature.lowLimit++;

    flag_I=TRUE;

}

else if (FVS007.userLimit==HIGH && FVS007.userSensor==0)

{

    FVS007.pressure.highLimit++;

    flag_I=TRUE;
```

```
}
```

```
else if(FVS007.userLimit==LOW && FVS007.userSensor==0)
```

```
{
```

```
    FVS007.pressure.lowLimit++;
```

```
    flag_I=TRUE;
```

```
}
```

```
else if(FVS007.userLimit==HIGH && FVS007.userSensor==2)
```

```
{
```

```
    FVS007.co2.highLimit++;
```

```
    flag_I=TRUE;
```

```
}
```

```
else
```

```
{
```

```
    FVS007.co2.lowLimit++;
```

```
    flag_I=TRUE;
```

}

}//eo if

}//eo increment

/\*---decrement -----

Author: SRaj

Date: 27/03/2020

Modified: None

Desc: User can decrease the limit of sensor.

Input: None

Returns: None

-----\*/

```
void decrement(void)
```

```
{
```

```
    if (pb4==0)
```

```
    {
```

```
        if (FVS007.userLimit==HIGH && FVS007.userSensor==1)
```

```
        {
```

```
            FVS007.temperature.highLimit--;
```

```
            flag_D=TRUE;
```

```
        }
```

```
        else if (FVS007.userLimit==LOW && FVS007.userSensor==1)
```

```
        {
```

```
            FVS007.temperature.lowLimit--;
```

```
            flag_D=TRUE;
```

```
        }
```

```
        else if (FVS007.userLimit==HIGH && FVS007.userSensor==0)
```

```
{
```

```
FVS007.pressure.highLimit--;
```

```
flag_D=TRUE;
```

```
}
```

```
else if(FVS007.userLimit==LOW && FVS007.userSensor==0)
```

```
{
```

```
FVS007.pressure.lowLimit--;
```

```
flag_D=TRUE;
```

```
}
```

```
else if(FVS007.userLimit==HIGH && FVS007.userSensor==2)
```

```
{
```

```
FVS007.co2.highLimit--;
```

```
flag_D=TRUE;
```

```
}
```

else

{

FVS007.co2.lowLimit--;

flag\_D=TRUE;

}

}//eo if

}//eo decrement

/\*---sentFormation -----

Author: SRaj

Date: 27/03/2020

Modified: Name or None

Desc: Formation of the string with checksum if user changes the limit

Input: None

Returns: None

-----\*/

```
void sentFormation()
```

```
{
```

```
    if(flag_I==TRUE)
```

```
    {
```

```
        if(FVS007.userLimit==1 && FVS007.userSensor==1)
```

```
        {
```

```
            timerCount++;
```

```
            sprintf(buf,"$CONLIM,%i,%i,T,H,%i\0",CONTROLLER,FVS,FVS007.temperature.highLimit);
```

```
            sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));
```

```
        if(timerCount<=5)
```



```
{

    fvs.puts(buf);

    pc.printf("\r%s",buf);

}

if(timerCount>5)

{

    pc.printf("\033[K\033[H\033[0");

    flag_l=FALSE;

    timerCount=0;

}

}

if(FVS007.userLimit==0 && FVS007.userSensor==1)

{

    timerCount++;

}
```

```
sprintf(buf,"$CONLIM,%i,%i,T,L,%i\0",CONTROLLER,FVS,FVS007.temperature.lowLimit);
```

```
sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));
```

```
if(timerCount<=5)
```

```
{
```

```
    fvs.puts(buf);
```

```
    pc.printf("\r%s",buf);
```

```
}
```

```
if(timerCount>5)
```

```
{
```

```
    pc.printf("\033[K\033[H\033[0");
```

```
    flag_l=FALSE;
```

```
    timerCount=0;
```

```
}
```

```
}
```

```
if(FVS007.userLimit==1 && FVS007.userSensor==0)
```

```
{
```

```
    timerCount++;
```

```
    sprintf(buf,"$CONLIM,%i,%i,P,H,%i\0",CONTROLLER,FVS,FVS007.pressure.highLimit);
```

```
    sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));
```

```
    if(timerCount<=5)
```

```
    {
```

```
        fvs.puts(buf);
```

```
        pc.printf("\r%s",buf);
```

```
    }
```

```
    if(timerCount>5)
```

```
    {
```

```
pc.printf("\033[K\033[H\033[0");
```

```
flag_l=FALSE;
```

```
timerCount=0;
```

```
}
```

```
}
```

```
if(FVS007.userLimit==0 && FVS007.userSensor==0)
```

```
{
```

```
timerCount++;
```

```
sprintf(buf,"$CONLIM,%i,%i,P,L,%i\0",CONTROLLER,FVS,FVS007.pressure.lowLimit);
```

```
sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));
```

```
if(timerCount<=5)
```

```
{
```

```
fvs.puts(buf);
```

```
pc.printf("\r%s",buf);
```

```
}
```

```
if(timerCount>5)
```

```
{
```

```
pc.printf("\033[K\033[H\033[0");
```

```
flag_l=FALSE;
```

```
timerCount=0;
```

```
}
```

```
}
```

```
if(FVS007.userLimit==1 && FVS007.userSensor==2)
```

```
{
```

```
timerCount++;
```

```
sprintf(buf,"$CONLIM,%i,%i,C,H,%i\0",CONTROLLER,FVS,FVS007.co2.highLimit);
```

```
sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));
```

```
if(timerCount<=5)
```

```
{
```

```
    fvs.puts(buf);
```

```
    pc.printf("\r%s",buf);
```

```
}
```

```
if(timerCount>5)
```

```
{
```

```
    pc.printf("\033[K\033[H\033[0");
```

```
    flag_l=FALSE;
```

```
    timerCount=0;
```

```
}
```

```
}
```

```

if(FVS007.userLimit==0 && FVS007.userSensor==2)

{

    timerCount++;

    sprintf(buf,"$CONLIM,%i,%i,C,L,%i\0",CONTROLLER,FVS,FVS007.co2.lowLimit);

    sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));


    if(timerCount<=5)

    {

        fvs.puts(buf);

        pc.printf("\r%s",buf);

    }

    if(timerCount>5)

    {

        pc.printf("\033[K\033[H\033[0");

```

```
flag_I=FALSE;
```

```
timerCount=0;
```

```
}
```

```
}
```

```
}
```

```
if(flag_D==TRUE)
```

```
{
```

```
if(FVS007.userLimit==1 && FVS007.userSensor==1)
```

```
{
```

```
timerCount++;
```

```
sprintf(buf,"$CONLIM,%i,%i,T,H,%i\0",CONTROLLER,FVS,FVS007.temperature.highLimit);
```

```
sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));
```



```
if(timerCount<=5)
```

```
{
```

```
    fvs.puts(buf);
```

```
    pc.printf("\r%s",buf);
```

```
}
```

```
if(timerCount>5)
```

```
{
```

```
    pc.printf("\033[K\033[H\033[0");
```

```
    flag_D=FALSE;
```

```
    timerCount=0;
```

```
}
```

```
}
```

```
if(FVS007.userLimit==0 && FVS007.userSensor==1)
```

```
{
```

```
timerCount++;
```

```
sprintf(buf,"$CONLIM,%i,%i,T,L,%i\0",CONTROLLER,FVS,FVS007.temperature.lowLimit);
```

```
sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));
```

```
if(timerCount<=5)
```

```
{
```

```
    fvs.puts(buf);
```

```
    pc.printf("\r%s",buf);
```

```
}
```

```
if(timerCount>5)
```

```
{
```

```
    pc.printf("\033[K\033[H\033[0");
```

```
flag_D=FALSE;
```

```
timerCount=0;
```

```
}
```

```
}
```

```
if(FVS007.userLimit==1 && FVS007.userSensor==0)
```

```
{
```

```
timerCount++;
```

```
printf(buf,"$CONLIM,%i,%i,P,H,%i\0",CONTROLLER,FVS,FVS007.pressure.highLimit);
```

```
printf(buf,"%s,%i#\0",buf,calChecksum(buf));
```

```
if(timerCount<=5)
```

```
{
```

```
fvs.puts(buf);
```

```
pc.printf("\r%s",buf);
```

```
}
```

```
if(timerCount>5)
```

```
{
```

```
pc.printf("\033[K\033[H\033[0");
```

```
flag_D=FALSE;
```

```
timerCount=0;
```

```
}
```

```
}
```

```
if(FVS007.userLimit==0 && FVS007.userSensor==0)
```

```
{
```

```
timerCount++;
```

```
sprintf(buf,"$CONLIM,%i,%i,P,L,%i\0",CONTROLLER,FVS,FVS007.pressure.lowLimit);
```

```
    sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));
```

```
if(timerCount<=5)
```

```
{
```

```
    fvs.puts(buf);
```

```
    pc.printf("\r%s",buf);
```

```
}
```

```
if(timerCount>5)
```

```
{
```

```
    pc.printf("\033[K\033[H\033[0");
```

```
    flag_D=FALSE;
```

```
    timerCount=0;
```

```
}
```

```
}
```

```
if(FVS007.userLimit==1 && FVS007.userSensor==2)
```

```
{
```

```
    timerCount++;
```

```
    sprintf(buf,"$CONLIM,%i,%i,C,H,%i\0",CONTROLLER,FVS,FVS007.co2.highLimit);
```

```
    sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));
```

```
    if(timerCount<=5)
```

```
    {
```

```
        fvs.puts(buf);
```

```
        pc.printf("\r%s",buf);
```

```
    }
```

```
    if(timerCount>5)
```

```
{

    pc.printf("\033[K\033[H\033[0");

    flag_D=FALSE;

    timerCount=0;

}

}

if(FVS007.userLimit==0 && FVS007.userSensor==2)

{

    timerCount++;

    sprintf(buf,"$CONLIM,%i,%i,C,L,%i\0",CONTROLLER,FVS,FVS007.co2.lowLimit);

    sprintf(buf,"%s,%i#\0",buf,calChecksum(buf));

    if(timerCount<=5)

    {
```

```
fvs.puts(buf);
```

```
pc.printf("\r%s",buf);
```

```
}
```

```
if(timerCount>5)
```

```
{
```

```
pc.printf("\033[K\033[H\033[0");
```

```
flag_D=FALSE;
```

```
timerCount=0;
```

```
}
```

```
}
```

```
}
```

```
} //eo sentFormation
```



/\*--- display -----

Author: SRaj

Date: 27/03/2020

Modified: Name or None

Desc: Display the output to terminal window

Input: Type and purpose of input arguments

Returns: Type and purpose of returning argument

-----\*/

void display (void)

{

pc.printf("\033[2J\033[H\033[0");

pc.printf("\r MMBED System Properties\n");

pc.printf("\r\n");

```
if (FVS007.userLimit==1 && FVS007.userSensor==1)

{

    pc.printf("\rMODE: HIGH \t\t CHANNEL: TEMPERATURE \n ");

}
```

```
if (FVS007.userLimit==0 && FVS007.userSensor==1)

{

    pc.printf("\rMODE: LOW \t\t CHANNEL: TEMPERATURE\n");

}
```

```
if (FVS007.userLimit==1 && FVS007.userSensor==0)

{

    pc.printf("\rMODE: HIGH \t\t CHANNEL: PRESSURE\n");

}
```

```
if (FVS007.userLimit==0 && FVS007.userSensor==0)

{

    pc.printf("\r MODE: LOW \t\t CHANNEL: PRESSURE\n");

}

if(FVS007.userLimit==1 && FVS007.userSensor==2)

{

    pc.printf("\rMODE: HIGH \t\t CHANNEL: CO2\n");

}

if(FVS007.userLimit==0 && FVS007.userSensor==2)

{

    pc.printf("\rMODE: LOW \t\t CHANNEL: CO2\n");

}

pc.printf("\n\rTemperature\t\t\tPressure\t\tCO2\n");
```

```
pc.printf("\rCurrent:\t%i%C\t\tCurrent: %i kPa \tCurrent: %d ppm\n",
currentTemperature,248,currentPressure,currentCo2);
```

```
pc.printf("\rHighLimit:\t%i%C\t\tHighLimit: %i kPa\tHighLimit: %i ppm\n",
FVS007.temperature.highLimit,248,FVS007.pressure.highLimit,FVS007.co2.highLimit); // for printing
high limit of all the sensors
```

```
pc.printf("\rLowLimit:\t%i%C\t\tLowLimit: %i kPa\tLowLimit: %i ppm\n",
FVS007.temperature.lowLimit,248,FVS007.pressure.lowLimit,FVS007.co2.lowLimit); // for printing
low limit of both the sesnsors
```

```
if(((currentTemperature>FVS007.temperature.highLimit) || (currentTemperature<FVS007.temperature.l
owLimit)) &&
((currentPressure<FVS007.pressure.highLimit) || (currentPressure>FVS007.pressure.lowLimit)) &&
((currentCo2>FVS007.co2.highLimit) || (currentCo2<FVS007.co2.lowLimit)))
```

```
{
```

```
pc.printf("\r\033[0;37mStatus : \033[0;31mAlarm\t\t\t \033[0;37mStatus:
\033[0;31mAlarm\033[0;37m\t\t \033[0;37mStatus: \033[0;31mAlarm\033[0;37m\n "); // for
printing temperature and pressure status
```

```
}
```

```
else
if(((currentTemperature>FVS007.temperature.highLimit) || (currentTemperature<FVS007.temperature.l
```

```
owLimit)) &&  
((currentPressure>FVS007.pressure.highLimit) || (currentPressure<FVS007.pressure.lowLimit)) &&  
((currentCo2>FVS007.co2.highLimit) || (currentCo2<FVS007.co2.lowLimit)))
```

```
{
```

```
    pc.printf("\r\033[0;37mStatus : \033[0;31mAlarm\t\t\t \033[0;37mStatus:  
\033[0;32mSafe\033[0;37m\t\t \033[0;37mStatus: \033[0;31mAlarm\033[0;37m\n");
```

```
}
```

```
    else  
if(((currentTemperature<FVS007.temperature.highLimit) || (currentTemperature>FVS007.temperature.l  
owLimit)) &&  
((currentPressure<FVS007.pressure.highLimit) || (currentPressure>FVS007.pressure.lowLimit)) &&  
((currentCo2>FVS007.co2.highLimit) || (currentCo2<FVS007.co2.lowLimit)))
```

```
{
```

```
    pc.printf("\r\033[0;37mStatus : \033[0;32mSafe\t\t\t \033[0;37mStatus:  
\033[0;31mAlarm\033[0;37m\t\t \033[0;37mStatus: \033[0;31mAlarm\033[0;37m\n");
```

```
}
```

```
    else if ((currentTemperature<FVS007.temperature.highLimit &&  
currentTemperature>FVS007.temperature.lowLimit) && (currentPressure>FVS007.pressure.highLimit  
&& currentPressure<FVS007.pressure.lowLimit) && ((currentCo2>FVS007.co2.highLimit) ||  
(currentCo2<FVS007.co2.lowLimit)))
```

```
{
```

```
pc.printf("\r\033[0;37mStatus : \033[0;32mSafe\t\t\t \033[0;37mStatus:  
\033[0;32mSafe\033[0;37m\t\t \033[0;37mStatus: \033[0;31mAlarm\033[0;37m\n");
```

```
}
```

```
else if(((currentTemperature>FVS007.temperature.highLimit) ||  
(currentTemperature<FVS007.temperature.lowLimit)) &&  
((currentPressure>FVS007.pressure.lowLimit) || (currentPressure<FVS007.pressure.highLimit)) &&  
(currentCo2>FVS007.co2.lowLimit && currentCo2<FVS007.co2.highLimit))
```

```
{
```

```
pc.printf("\r\033[0;37mStatus : \033[0;31mAlarm\t\t\t \033[0;37mStatus:  
\033[0;31mAlarm\033[0;37m\t\t \033[0;37mStatus: \033[0;32mSafe\033[0;37m\n");
```

```
}
```

```
else  
if(((currentTemperature>FVS007.temperature.highLimit) || (currentTemperature<FVS007.temperature.l  
owLimit)) &&  
((currentPressure>FVS007.pressure.highLimit) || (currentPressure<FVS007.pressure.lowLimit)) &&  
((currentCo2<FVS007.co2.highLimit && currentCo2>FVS007.co2.lowLimit)))
```

```
{
```

```
pc.printf("\r\033[0;37mStatus : \033[0;31mAlarm\t\t\t \033[0;37mStatus:  
\033[0;32mSafe\033[0;37m\t\t \033[0;37mStatus: \033[0;32mSafe\033[0;37m\n");
```

```
}
```

```

else
if(((currentTemperature<FVS007.temperature.highLimit) || (currentTemperature>FVS007.temperature.l
owLimit)) &&
((currentPressure<FVS007.pressure.highLimit) || (currentPressure>FVS007.pressure.lowLimit)) &&
((currentCo2<FVS007.co2.highLimit && currentCo2>FVS007.co2.lowLimit)))

{

    pc.printf("\r\033[0;37mStatus : \033[0;32mSafe\t\t\t \033[0;37mStatus:
\033[0;31mAlarm\033[0;37m\t\t \033[0;37mStatus: \033[0;32mSafe\033[0;37m\n");

}

else

{

    pc.printf("\r\033[0;37mStatus : \033[0;32mSafe\t\t\t \033[0;37mStatus:
\033[0;32mSafe\033[0;37m\t\t \033[0;37mStatus: \033[0;32mSafe\033[0;37m\n");

}

pc.printf("\n");

pc.printf("\rMOTOR1: Discharge\t\tMOTOR2: Dampener\n");

pc.printf("\rPosition: %i\t\t\tPosition:
%i\n",FVS007.motor1.currentPosition,FVS007.motor2.currentPosition);

```

```
if(FVS007.motor1.status==1 && FVS007.motor2.status==1)
```

```
{
```

```
    pc.printf("\rControl: ON\t\tControl: ON\n");
```

```
}
```

```
if(FVS007.motor1.status==1 && FVS007.motor2.status==0)
```

```
{
```

```
    pc.printf("\rControl: ON\t\tControl: OFF\n");
```

```
}
```

```
if(FVS007.motor1.status==0 && FVS007.motor2.status==1)
```

```
{
```

```
    pc.printf("\rControl: OFF\t\tControl: ON\n");
```

```
}
```

```
if(FVS007.motor1.status==0 && FVS007.motor2.status==0)
```

```
{
```

```
    pc.printf("\rControl: OFF\t\tControl: OFF\n");
```



```
}
```

```
sentFormation();
```

```
} //eo display
```

```
/*---calRxChecksum -----
```

Author: SRaj

Date: 08/04/2020

Modified: Name or None

Desc: Calculate the checksum value of received string

Input: Input of buffer

Returns: Returns calulated checksum value

```
-----*/
```

```
char calRxChecksum(char *ptr)
```

```
{  
  
    char cs2=0;  
  
    while(*ptr)  
  
    {  
  
        cs2+=*ptr;  
  
        ptr++;  
  
    }  
  
    return cs2;  
  
} // eo calChecksum
```

/\*---collectSentence -----

Author: SRaj

Date: 08/04/2020

Modified: Name or None

Desc: collect the serial communication sentence from the PIC

Input:   Type and purpose of input arguments

Returns:   Type and purpose of returning argument

-----\*/

```
void collectSentence()
```

```
{
```

```
    if(fvs.readable())
```

```
    {
```

```
        char hold=fvs.getc();
```

```
        if(hold=='$')
```

```
        {
```

```
            ptr=rxBuf;
```

```
        }
```

```
        if(hold=='#')
```

```
{  
  
    sentRdy=TRUE;  
  
}  
  
*ptr=hold;  
  
ptr++;  
  
} //eo if  
  
}  
  
  
/*---validateSentence-----
```

Author: SRaj

Date: 08/04/2020

Modified: Name or None

Desc: Validate the received command sentence

Input: buffer as input

Returns: TRUE or FALSE

-----\*/

```
char validateSentence (char *ptr)
```

```
{
```

```
    char rcs=0;
```

```
    char ncs=0;
```

```
    char csFlag=FALSE;
```

```
    int count=strlen(ptr);
```

```
    while(!csFlag)
```

```
    {
```

```
        if(*(ptr+count)=='#')
```

```
        {
```

```
            *(ptr+count)=0X00;
```

```
}
```

```
if(*(ptr+count)=='')
```

```
{
```

```
    *(ptr+count)=0X00;
```

```
    rcs=atoi(ptr+count+1);
```

```
    csFlag=TRUE;
```

```
}
```

```
    count--;
```

```
}
```

```
ncs=calRxChecksum(ptr);
```

```
if(ncs==rcs)
```

```
{
```

```
    return TRUE;
```

```
}
```

```
else
```

```

{

    return FALSE;

}

} //eo validateSentence

/*---purseSentence -----

Author:  SRaj

Date:    08/04/2020

Modified: Name or None

Desc:    purse the sentence and save each command in token

Input:   buffer as input

Returns: Type and purpose of returning argument

-----*/

```

```
void purseSentence(char *ptr)
```

```
{
```

```
    int tokenCount=0;
```

```
    while(*ptr)
```

```
    {
```

```
        if(*ptr=='$' || *ptr==',')
```

```
        {
```

```
            *ptr=0X00;
```

```
            token[tokenCount]=ptr+1;
```

```
            tokenCount++;
```

```
        }
```

```
        ptr++;
```

```
    }
```



```
}// eo PurseSentence
```

```
/*---executeSentence -----
```

Author: SRaj

Date: 08/04/2020

Modified: Name or None

Desc: execute the recieved sentence

Input: Type and purpose of input arguments

Returns: Type and purpose of returning argument

```
-----*/
```

```
void executeSentence()
```

```
{
```

```
if(atoi(token[1])==CONTROLLER)

{

    if(atoi(token[2])==FVS)

    {

        if(strcmp(token[0],string1)==0)

        {

            if(strcmp(token[3],string2)==0)

            {

                currentTemperature=atoi(token[4]);

            }

            if(strcmp(token[3],string3)==0)

            {

                currentPressure=atoi(token[4]);

            }

            if(strcmp(token[3],string4)==0)
```

```
{

    currentCo2=atoi(token[4]);

}

}

if(strcmp(token[0],string5)==0)

{

    if(atoi(token[3])==mtr1)

    {

        if(atoi(token[4])==mtrOn)

        {

            FVS007.motor1.currentPosition=atoi(token[5]);

            FVS007.motor1.status=1;

        }

        if(atoi(token[4])==mtrOff)
```

```
{

    FVS007.motor1.currentPosition=atoi(token[5]);

    FVS007.motor1.status=0;

}

}

if(atoi(token[6])==mtr2)

{

    if(atoi(token[7])==mtrOn)

    {

        FVS007.motor2.currentPosition=atoi(token[8]);

        FVS007.motor2.status=1;

    }

    if(atoi(token[7])==mtrOff)

    {

        FVS007.motor2.currentPosition=atoi(token[8]);
```

```
FVS007.motor2.status=0;
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
/*--- MAIN FUNCTION -----
```

```
-----*/
```

```
int main()
```

```
{

    initialize_FVS();

    fvs.attach(&collectSentence);


    while(TRUE)

    {

        timer.start();

        if(timer.read()>=1)

        {

            userMode();

            userChannel();

            increment();

            decrement();

            display();

            if(sentRdy)
```

```
{  
  
    if(validateSentence(rxBuf))  
  
    {  
  
        purseSentence(rxBuf);  
  
        executeSentence();  
  
        sentRdy=FALSE;  
  
    }  
  
}  
  
    timer.reset();  
  
} // eo if  
  
} // eo while  
  
}
```