

```

#include <stdlib.h>
#include <dos.h>
#include "driver.h"

void set_timer(int frequency);
void restore_timer();

const int freq=60;
const double ds=16.3/48;

#define lpt_data    lpt_base+0
#define lpt_status  lpt_base+1
#define lpt_control lpt_base+2

enum{clock_int=0x08};

driver *driver::current=0;
const byte driver::left_motor[4]  ={0x08,0x04,0x02,0x01};
const byte driver::right_motor[4] ={0x10,0x20,0x40,0x80};

driver::driver(int lpt, int irq)
{
    installed=false;
    s_left=s_right=0;
    left=right=0;
    last_status=0;
    control=0x13;
    piezo_t=0;
    automode=false;
    install(lpt,irq);
}

driver::~driver()
{
    uninstall();
}

void driver::stop()
{
    disable();
    automode=false;
    s_left=s_right=0;
    enable();
}

void driver::move(double s)
{
    disable();
    automode=false;
    s_left=s_right=int(s/ds);
    enable();
}

void driver::rotate(double angle)
{
    disable();
    automode=false;
    int s=int(160./360 * angle);
    s_left+=s;
    s_right-=s;
    enable();
}

void driver::set_automode()
{
    disable();
    automode=true;
    last_move=none;
    enable();
}

volatile bool playing=false;
bool en;
int p_time;

void interrupt driver::play_int(...)
{
    if(playing && en && current)
        outportb(current->lpt_control, current->control^=0x08);
    if(p_time>0)
        p_time--;
    else playing=false;
}

void driver::play(int frequency, double duration)
{
    if(!installed)
        return;
    if(frequency<20){
        frequency=20;
        en=false;
    }
    else en=true;
    p_time=int(duration*frequency);
    playing=true;
    void interrupt (*old_int)(...)=getvect(clock_int);
    set_timer(frequency);
    setvect(clock_int,play_int);
    while(playing) {}
    set_timer(freq);
    setvect(clock_int,old_int);
}

void driver::step_left(int dir)
{
    left=(left+dir)&3;
    s_left-=dir;
}

void driver::step_right(int dir)
{

```

```

right=(right+dir)&3;
s_right-=dir;
}

void driver::autopilot()
{
    if(last_status)
    if(last_status&0xe0){ //zderzenie z przodu lub utrata podloza
        s_left=s_right=-40;
        last_move=backward;
    }
    else if(last_status&0x10){ //zderzenie z lewej
        s_left=-40;
        s_right=+40;
        last_move=right_rot;
    }
    else{
        s_right=-40;
        s_left=+40;
        last_move=left_rot;
    }
    else while(true)
    switch(random(4)){
        case 0: case 1:
            if(last_move==backward)
                break;
            s_left=s_right=100;
            last_move=forward;
            return;
        case 2:
            if(last_move==right_rot)
                break;
            s_left=+50;
            s_right=-50;
            last_move=left_rot;
            return;
        case 3:
            if(last_move==left_rot)
                break;
            s_left=-50;
            s_right=+50;
            last_move=right_rot;
            return;
    }
}

void driver::step()
{
    if(piezo_t>0){
        outportb(lpt_control,control^=0x08);
        piezo_t--;
    }
    {
        byte status=inportb(lpt_status);
        status^=0x78; // odwrocenie zanegowanych sygnalow
        status&=0xf8; // obcięcie niepotrzebnych bitow
        if(status & ~last_status){
            s_left=s_right=0;
            piezo_t=20; //daj glos
        }
        last_status=status;
    }
    if(s_left==0 && s_right==0)
    if(automode)
        autopilot();
    else outportb(lpt_data,0);
    else{
        if(s_left>0) step_left(+1);
        if(s_left<0) step_left(-1);
        if(s_right>0) step_right(+1);
        if(s_right<0) step_right(-1);
        outportb(lpt_data,left_motor[left] | right_motor[right]);
    }
}

bool driver::install(int lpt, int irq)
{
    if(installed)
        return true;
    if(driver::current)
        return false;
    if(lpt<1 || 4<lpt || irq<2 || 15<irq)
        return false;
    lpt_base=((word far*)0x00000408 + (lpt-1));
    if(irq<8){
        pic_base=0x20; // adres ukkladu 8259 (Master)
        irq_mask=1<<irq;
        lpt_int=0x08+irq;
    }
    else{
        pic_base=0xa0; // adres ukkladu Slave
        irq_mask=1<<(irq-8);
        lpt_int=0x68+irq;
    }
    current=this;
    set_timer(freq);
    backup.old_clock_int=getvect(clock_int);
    backup.old_lpt_int=getvect(lpt_int);
    setvect(clock_int,clock_int_handler);
    setvect(lpt_int,lpt_int_handler);
    outportb(lpt_control,control);
    outportb(pic_base+1,inportb(pic_base+1) & ~irq_mask); /* od
przerwania - wyzerowanie bitu maski w OCW1 8259 */
    installed=true;
    return true;
}

void driver::uninstall()
{
    restore_timer();
    outportb(lpt_control,0x03); /* zgaszenie diod */
    outportb(pic_base+1,inportb(pic_base+1) | irq_mask);
    //zablokowanie przerwania z LPT
    setvect(clock_int,backup.old_clock_int); //odtworzenie wektorow

```

```

setvect(lpt_int,backup.old_lpt_int);
outputb(lpt_data,0); // wyłączenie silników
current=0;
installed=false;
}

```

230

```

void interrupt driver::clock_int_handler(...)
{
    if(current)
        current->step();
}

```

240

```

void interrupt driver::lpt_int_handler(...)
{
    if(current){
        current->s_left=current->s_right=0;
        outputb(current->lpt_control,current->control^=0x4);
    }
    outputb(current->pic_base,0x20); /* end of interrupt
    - powiadomienie 8259 o koncu obsługi przerwania */
}

```

250

```

void set_timer(int frequency)
{
    const double ticks=1193180; //częstotliwość taktowania licznika - 1.19MHz
    const word divisor=ticks/frequency;
    enum{timer=0x0040};
    outputb(timer+3,0x34); //zapis do licznika
    outputb(timer+0,divisor&0xff); //mniej znaczący bajt
    outputb(timer+0,divisor>>8); //bardziej znaczący
}

```

260

```

void restore_timer()
{
    enum{timer=0x0040};
    outputb(timer+3,0x34);
    outputb(timer+0,0xff);
    outputb(timer+0,0xff);
}

```