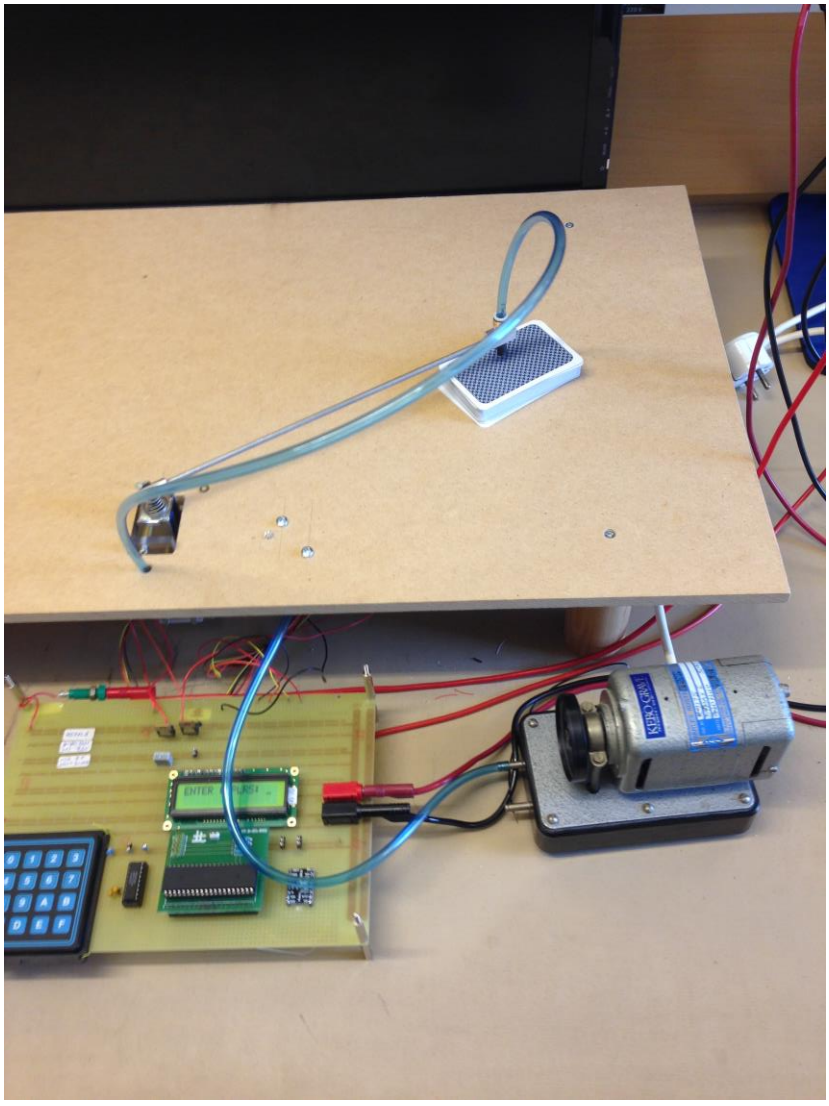
Digitala projekt, EITF11 - Rapport
Handledare: Bertil Lindvall

Fredrik Nylund (ine12fny)
Johan Ekman (ine11jek)
Olof Wikman (ine11owi)



Innehållsförteckning

Inledning
Denna rapport är resultatet av konstruktionskursen Digitala projekt EITF11. Syftet med kursen är att ge ett begrepp om hur konstruktionsarbete går till genom att ta fram en fungerande prototyp med tillhörande dokumentation. Detta innefattar planering och konstruktion av hårdvara och implementering av den programkod som utgör mjukvaran samt integration av båda dessa.

Kravspecifikation
Gruppen har valt att konstruera en kortdelarmaskin som minimerar det fruktansvärda arbete som uppstår i samband med utdelning av kort vid kortspel. Nedan presenteras ett antal krav som ursprungligen ställdes upp:

1. Kortutdelaren ska kunna dela ut ett antal kort till ett antal spelare enligt användarens val.
2. Antalet spelare och kort ska anges av spelaren via en numerisk knappsats. Max spelare skall vara 5. Max antal kort skalla vara en kortlek, dvs. 52.
3. En skärm skall skriva ut de val som användaren förväntas göra och sedan bekräfta dessa val.
4. Kortutdelaren ska plocka kort från en korthög med hjälp av en arm och en kompressor som möjliggör att kortet sugs fast med munstycke längst ut på armen.
5. Korten delas ut genom att de plockas ett och ett från en korthög och delar ut till spelarna.
6. Delararmen styrs i en cirkelbåge i horisontalplanet av en motor som är placerad i origo.
7. Armen kan röras i vertikalled med en motor.

Hårdvara
De huvudsakliga komponenter som användes presenteras nedan. Hur hårdvarukomponenterna kopplades återfinns i kopplingsschemat, se bilaga.

Processor
Prototypen använder en ATMega16 som processor. Denna har ett minne på 16kb och fyra portar A, B, C och D fördelade över 32 programmeringsbara 1/0 pinnar, varav 8 med möjlighet till A/D-omvandling. Processorn är även kompatibel med J-TAG vilket medför enkel implementering av programkod genom AVR-studio.

Display
För att kommunicera med användaren används en LCD skärm av modell SHARP Dot-Matrix, LCD Alfanumerisk teckendisplay. Denna anger enligt kravspecifikationen vad användaren förväntas göra och bekräftar de val av antal spelare och kort som görs.

Knappsats och encoder
För att användaren skall kunna ange antal spelare och kort används en 4x4 matris knappsats. De numeriska knapparna används för att göra detta och bokstäverna A-F används inte i prototypen förutom då man initialt kan trycka valfri tangent för att starta

programmet. Knappsatsen kopplas via en encoder till mictrocontrollern för att underlätta mjukvaruprogrammeringen.

Stegmotor och driver carrier

En bipolär 4 lead stegmotor används för att röra armen i horisontalplanet. Denna är kopplad via en driver carrier för att låta mikroprocessorn kontrollera stegmotorns antal steg och rikting.

Motor för vertikal rörelse

En motor används för att justera armen i vertikalled. Den använda motorn möjliggör att placera armen i två lägen, ett undre och ett övre med en differens på ca 10mm.

Kompressor och ventil

En kompressor av märket Charles Austen Pumps används. Denna går konstant och ger upphov till ett sug i munstycket längs ut på armen. En ventil som seriekopplas mellan kompressorn och munstycket möjliggör att suga fast och släppa kort genom kontroll från mikroprocessorn.

Övrigt

För att få systemet att fungera som önskat används även några ytterligare komponenter:

- Två st kondensatorer, 1u och u1(), föra att få encodern att fungera korrekt.
- Två st NPN-transistorer för att kunna låta mikroprocessorn hantera en 13V spänning till vertikalmotorn och ventilen samt en 9,6V spänning till stegmotorn.
- En potentiometer för att justera kontrasten på displayen.
- Tre st spänningsaggregat på 5V, 9,6V respektive 13V används för strömförsörjning.

Mjukvara

Efter att all nödvändig setup har utförts, portar satts input eller output, displayen initierats etc, går programmet in i en while-loop. I while-loopen anropas två olika funktioner, ett som sköter knappsatsen och ett som sköter motorernas rörelse, beroende på vilken fas man är i.

Under menyfasen, när antal spelare och antal kort knappas in, anropas funktionen keypad_control() i main loopen. Här utförs olika funktioner beroende på vilken flagga, keypad_flag, som är uppe. Flaggan ändras när en knapp trycks in och ISR(VECT1_OVF) anropas. När antalet spelare och kort har bestämts och alla värden är validerade startas motorn.

Under motorfasen, när armen plockar upp och delar ut kort, anropas funktionen motor_control() i main loopen. På samma sätt som med keypad_control() bestäms operation beroende på flagga. Flaggan, motor_flag, sätts här medhjälp av en timer (INT1), interupten triggas var 0.03sek, och medhjälp av variablerna timer och step_count kan korten delas ut med rätt delay och avstånd. När utdelningen är klar återgår programmet till menyfasen.

För koden, se bilaga.

Metod

Planering

Efter att ha diskuterat fram den idé som skulle förverkligas, ställdes ett antal funktionella krav upp för prototypen. Efter detta gjordes en grov skiss av systemet vilket resulterade i en lista med nödvändiga hårdvarukomponenter. Denna lista uppdaterades något under arbetets gång. Ett första kopplingsschema konstruerade i programmet PowerLogic. Även denna uppdaterades något under arbetets gång då ytterligare komponenter upptäcktes vara nödvändiga.

Hårdvarukonstruktion

Utefter kopplingsschemat sattes systemet samman. Därefter testades varje komponent för sig. Testen genomfördes med AVR studio. De felkopplingar som upptäcktes åtgärdades.

Programmering

Programmet kodades i C i programmet AVR studios. Koden testades kontinuerligt med hjälp av en J-TAG så att den på ett effektivt sätt hanterade hårdvaran.

Kalibrering

Efter att de funktionella kraven uppfyllts skedde kalibrering av hårdvaran för att prototypen skulle dela korten på ett så smidigt sätt som möjligt. Vi justerade antalet steg i stegmotorn för att få rätt avstånd mellan alla spelare. Vi justerade även armens grundläge för att den ska befinna sig i rätt höjd för att kunna plocka kort från en 52korts kortlek. Även kompressorns hastighet och munstycket justerades.

Problem vid konstruktion

Vissa problem uppstod under arbetets gång pga. okunskap från vår sida och som efter vidare efterforskningar kunde lösas. Problemen med lösningar presenteras nedan:

- Armens ursprungsläge i horisontalplanet var svårt att ställa in vid start av systemet. Detta löses genom att man låter användaren göra detta innan användning.
- Begränsning av armens rörelse i vertikalled. Detta löstes genom att begränsa antal kort som kan delas ut till 10 st.
- Vertikalmotorn fastnade ibland i nedre läget. Smörjning ledde till bättre resultat.
- Tre olika spänningar är inblandade. Detta beror på att mikroprocessorn använder 5V och ventilen, vertikalmotorn och stegmotorn använder 12V, 24V respektive 9,6V. Vi lyckades dock hitta en kompromiss för ventilen och vertikalmotorn som båda fungerade på drygt 13V. Dock krävdes fortfarande tre olika spänningar. Genom att använda grundspänningen 5V kunde de övriga två spänningarna kontrolleras via NPN-tranistorerna.
- Stegmotorn blev inledningsvis för varm. Detta justerades genom att reglera strömstyrkan genom motorn genom potentiometern på driver carrier chipet.
- Ett problem uppstod när hastigheten på stegmotorn ökades då detta ledde till hackiga rörelser då armen är något svajig. Detta medför en viss begränsning i delarhastighet.

- En önskvärd funktion var att armen skulle återgå och stanna i sitt ursprungsläge under tiden spelet genomfördes. Detta omöjliggjordes dock av att stegmotorn blir för varm då ström förs igenom den under för lång tid.
- För att få allt att fungera som önskat krävs att man justerar armens läge vertikalled innan man kör.

Resultat

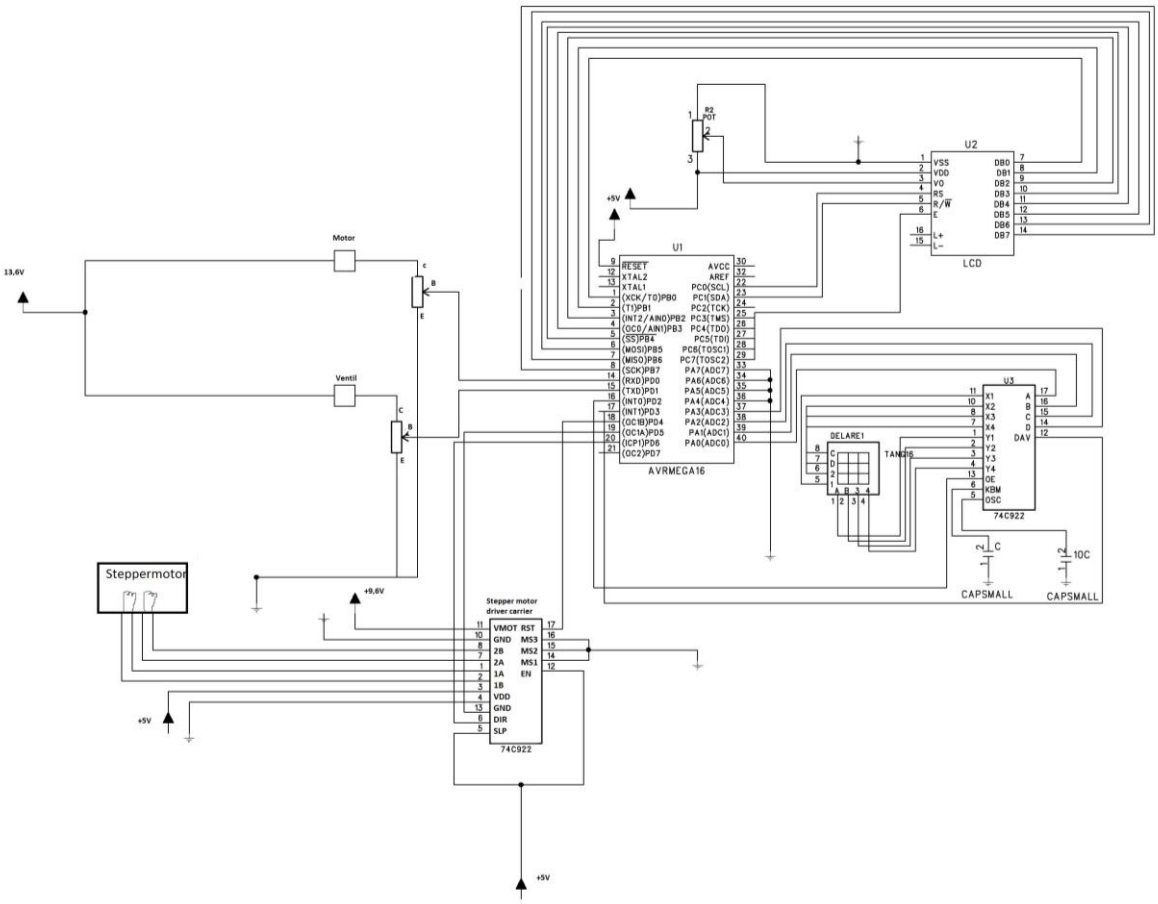
Alla krav i specifikationen uppfylldes förutom andra delen av krav 2. Antalet spelare och kort kan visserligen anges av spelaren via den numerisk knappsats, men systemet fungerar inte med upp till 52 kort. Det begränsades till totalt 2 kort kort per spelare dvs totalt 10 kort. Detta beror på begränsningen i motorn som justerar armen i vertikalled. Armen når inte ner till korthögen då fler än ca 10 kort redan plockats. Pga. Armen inte är helt fix så fungerar utdelaren ej som önskat även då endast 10 kort ska delas.

Slutsatser

Projektet gav oss lärdomar i planering, konstruktion och felsökning av hårdvara samt konstruktion av mjukvara och hur dessa två integrerades för att erhålla ett fungerande system. Vi lyckades i teorin uppfylla i stort sätt alla krav. I realiteten fungerade dock slutprodukten inte helt som önskat, något som kanske inte var helt oväntat. Även om den hade fungerat som i teorin hade det kanske inte varit någon avsevärd förbättring av kortutdelande jämfört med manuell delning.

Bilaga

Kopplingsschema



Källkod

```
/*
 * Carddealer.c
 *
 * Created: 2014-03-20 11:09:22
 *  Author: digpi01
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 8000000UL // 8 MHz

//Input variables
volatile char players;
volatile char cards;
volatile char key;

//Variables used to control keypad
volatile uint16_t menu_flag=0;
volatile uint16_t menu_displayed;
volatile uint16_t menu_finished=0;

//Variables used to control motor
volatile uint16_t step_count=0;
volatile uint16_t steps_players=1;
volatile uint16_t steps_cards=1;
volatile uint16_t motor_flag=0;
volatile uint16_t timer=0;
volatile uint16_t motor_finished=0;

//Interrupt Service Routine for when a key is pressed
ISR(INT1_vect)
{
            key = PINA;
            if (menu_flag==0){
                        motor_finished=0;
                        menu_flag++;
            }else if (menu_flag==1){
                        if (check_key(key)!='E'){
                                    //check if input is valid
                                    menu_flag++;
                        }else{
                                    menu_flag=4;
                        }
            }else if (menu_flag==2){
                        if (check_key(key)!='E'){
                                    //check if input is valid
                                    menu_flag++;
                        }else{
                                    menu_flag=4;
                        }
            }else if (menu_flag==4){
                        //invalid input
                        menu_flag=1;

            }
}

int keypad_control(){
            if (menu_flag==1){
                        //Make arm hard
```

```
                        PORTD=PORTD&0b11101111;
                        //Write "Enter hashtag player:"
                        disp_write_players();
                        menu_displayed=1;
                }else if (menu_flag==2){
                        //Check pressed value
                        players=check_key(key);
                        disp_write_letter(players);
                        disp_write_cards();
                        menu_displayed=2;
                }else if (menu_flag==3){
                        //Check pressed value
                        cards = check_key(key);
                        disp_write_letter(cards);
                        //disable_button();
                        enable_timer();
                        menu_finished=1;
                        menu_displayed=3;
                }else if (menu_flag==4){
                        //If invalid input clear display and write "Invalid entry"
                        disp_clear();
                        disp_write_invalid();
                        menu_displayed=4;
                }
}


int main(void)
{
                setup();
                disp_setup();

    while(1)
    {
                        if (menu_finished==0 && menu_displayed!=menu_flag){
                                keypad_control();
                        }
                        if (motor_finished==0 && menu_finished==1){
                                motor_control();
                        }
                }
}


ISR(TIMER0_OVF_vect)
{
                if (motor_flag==0 || motor_flag==1 || motor_flag==4||motor_flag==7){
                        motor_flag++;
                }else if (motor_flag==2){
                        //(40*0.03)s delay
                        timer++;
                        if(timer>40){
                                timer=0;
                                motor_flag=3;
                        }
                }else if(motor_flag==3){
                        timer++;
                        if (timer>40){
                                timer=0;
                                motor_flag=4;
```

```
                        }
                }else if(motor_flag==5){
                        //Will move steps_players*20 steps before continueing.
                        if (step_count<(steps_players*15)){
                                motor_flag=4;
                                step_count++;
                        }else{
                                motor_flag=6;
                                step_count=0;
                        }
                }else if (motor_flag==6){
                        //(40*0.03)s delay.
                        timer++;
                        if(timer>40){
                                timer=0;
                                motor_flag=7;
                        }
                }else if(motor_flag==8){
                        //Will move steps_players*20 steps before continueing.
                        if (step_count<(steps_players*15)){
                                motor_flag=7;
                                step_count++;
                        }else{
                                step_count=0;
                                if (steps_players<players-48){
                                        //Will increase distances moved by
20 if there still are more players.
                                        motor_flag=1;
                                        steps_players++;
                                }else{
                                        if (steps_cards<cards-48){
                                        //Will start new cycle with 20 steps
if there are more cards to deal.
                                                steps_cards++;
                                                steps_players=1;
                                                motor_flag=1;

                                        }else{
                                                motor_flag=9;
                                        }
                                }
                        }
                }

}

int motor_control(){
                        if (motor_flag==1){
                                //Arm down
                                PORTD=PORTD|0b00000001;
                        }else if(motor_flag==2){
                                //Port open
                                PORTD=PORTD&0b11111101;
                        }else if(motor_flag==3){
                                //Arm up
                                PORTD=PORTD&0b11111110;
                        }else if(motor_flag==4){
                                //Set Direction
                                PORTD=PORTD|0b00100000;
                                //Toggle step
                                PORTD=PORTD|0b01000000;
                        }else if(motor_flag==5){
```

```
                                        //Untoggle step
                                        PORTD=PORTD&0b10111111;

                                }else if(motor_flag==6){
                                        //Port close
                                        PORTD=PORTD|0b00000010;
                                }else if (motor_flag==7){
                                        //Set Direction
                                        PORTD=PORTD&0b11011111;
                                        //Toggle step
                                        PORTD=PORTD|0b01000000;
                                }else if (motor_flag==8){
                                        //Untoggle step
                                        PORTD=PORTD&0b10111111;
                                }else if (motor_flag==9){
                                        //Dealing is complete. Back to menu
                                        reset();
                                }
}

int reset(){
            //Variables used to control keypad
            menu_flag=0;
            menu_displayed=10;
            menu_finished=0;

            //Variables used to control motor
            step_count=0;
            steps_players=1;
            steps_cards=1;
            motor_flag=0;
            timer=0;
            motor_finished=0;

            disable_timer();
            enable_button();
            main();
}
int enable_timer()
{
            cli();
                        //Disable Global Interrupts


            TCCR0 = (1<<CS02)|(1<<CS00);          //Timer clock (255*1024/8M)

            TIMSK|=(1<<TOIE0);                                                  //Enable
Timer0

            sei();
                        //Enable Global Interrupts
}
int disable_timer()
{

                                        //Clear pending interrupts
            TIMSK &= ~(1 << TOIE0);
}


/*
 * commands.c
```

```
 *
 * Created: 2014-03-20 17:21:29
 *  Author: digpi01
 */

#include <avr/io.h>
#include <avr/interrupt.h>


int setup()
{
		DDRA = 0x00;
		DDRB = 0xFF;
		DDRC = 0xFF;
		DDRD = 0xF7;
		TCCR1B |= (1 << CS10);
		PORTD=PORTD&0b11111110;
		PORTD=PORTD|0b00010000;
		PIND=PIND|00000100;
		enable_button();
}

int enable_button()
{
		cli(); //Disable Global Interrupts
		GICR =(1<<INT1); //Set Bit6 of GICR to unmask INT0 interrupt.
		MCUCR =(3<<ISC10); //Configuring MCUCR for Rising Edge interrupt for INT0
		sei(); //Enable Global Interrupts


}

int disable_button(){
		GICR &= ~(1 << INT0);

}

char check_key(char key)
{

		if (key==0b00001000){
				return '1';
		}else if(key==0b00000100){
				return '2';
		}else if(key==0b00000000){
				return '3';
		}else if(key==0b00001101){
				return '4';
		}else if(key==0b00001001){
				return '5';
		}else{
				return 'E';
		}



}


int delay(int delay){
		for(int j=0;j<delay;j++){
				for(int i=0;i<4000;i++){
```

```
                        }
                }
}


/*
 * LCD_commands.c
 *
 * Created: 2014-03-26 17:11:29
 *  Author: digpi01
 */
/*
 * CFile1.c
 *
 * Created: 2014-03-20 17:18:23
 *  Author: digpi01
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/delay.h>

int disp_setup(void)
{
                //Setup
                PORTC=0x80;

                //Clear disp
                PORTB=0x1;
                disp_enable_setup();
                //send_command();
                //Sending Function Set
                PORTB = 0x38;
                disp_enable_setup();
                //send_command();

                //Sending Display ON/OFF
                PORTB=0xE;
                disp_enable_setup();
                //send_command();

                //Sending Entry Mode Set
                PORTB=0x6;
                disp_enable_setup();
                //send_command();

                disp_clear();
                disp_write_position();
}
int disp_enable(void)
{
                //Switching enable
                PORTC=0x1;
                delay(1);
                PORTC=0x81;
}

int disp_enable_setup(void)
{
                //Switching enable
                PORTC=0x0;
                delay(1);
```

```
            PORTC=0x80;
}

int disp_clear(void)
{
            //Clear disp
            PORTC=0x80;
            PORTB=0x1;
            disp_enable_setup();
            //Sending Function Set
            PORTB = 0x2;
            disp_enable_setup();
}

void disp_write_letter(char val)
{
            //writing char val
            PORTC=0x81;
            PORTB=val;
            disp_enable();
}

void disp_new_row()
{
            //Changing row
            PORTC=0x80;
            PORTB=0xC0;
            disp_enable_setup();
}

int disp_write_players(void)
{
            //Writing "Enter # Plyrs:
            disp_clear();
            disp_write_letter('E');
            disp_write_letter('N');
            disp_write_letter('T');
            disp_write_letter('E');
            disp_write_letter('R');
            disp_write_letter(' ');
            disp_write_letter('#');
            disp_write_letter(' ');
            disp_write_letter('P');
            disp_write_letter('L');
            disp_write_letter('R');
            disp_write_letter('S');
            disp_write_letter(':');
            disp_write_letter(' ');

}

int disp_write_invalid(void)
{
            //Writing "Enter # Plyrs:
            disp_clear();
            disp_write_letter('I');
            disp_write_letter('N');
            disp_write_letter('V');
            disp_write_letter('A');
            disp_write_letter('L');
            disp_write_letter('I');
            disp_write_letter('D');
```

```
            disp_write_letter(' ');
            disp_write_letter('E');
            disp_write_letter('N');
            disp_write_letter('T');
            disp_write_letter('R');
            disp_write_letter('Y');
}


int disp_write_position(void)
{
            //Writing "Enter # Plyrs:
            disp_clear();
            disp_write_letter('A');
            disp_write_letter('R');
            disp_write_letter('M');
            disp_write_letter(' ');
            disp_write_letter('O');
            disp_write_letter('V');
            disp_write_letter('E');
            disp_write_letter('R');
            disp_write_letter(' ');
            disp_write_letter('C');
            disp_write_letter('A');
            disp_write_letter('R');
            disp_write_letter('D');
            disp_write_letter('S');
            disp_new_row();
            disp_write_letter('&');
            disp_write_letter(' ');
            disp_write_letter('P');
            disp_write_letter('R');
            disp_write_letter('E');
            disp_write_letter('S');
            disp_write_letter('S');
            disp_write_letter(' ');
            disp_write_letter('A');
            disp_write_letter('N');
            disp_write_letter('Y');
            disp_write_letter(' ');
            disp_write_letter('K');
            disp_write_letter('E');
            disp_write_letter('Y');
}

int disp_write_restart(void)
{
            //Writing "Enter # Plyrs:
            disp_clear();
            disp_write_letter('P');
            disp_write_letter('R');
            disp_write_letter('E');
            disp_write_letter('S');
            disp_write_letter('S');
            disp_write_letter(' ');
            disp_write_letter('A');
            disp_write_letter('N');
            disp_write_letter('Y');
            disp_write_letter(' ');
            disp_write_letter('K');
            disp_write_letter('E');
            disp_write_letter('Y');
```

```
            disp_new_row();
            disp_write_letter('T');
            disp_write_letter('O');
            disp_write_letter(' ');
            disp_write_letter('R');
            disp_write_letter('E');
            disp_write_letter('S');
            disp_write_letter('T');
            disp_write_letter('A');
            disp_write_letter('R');
            disp_write_letter('T');
}
int disp_write_cards(void)
{
            //Writing "Enter # Crds:"
            disp_new_row();
            disp_write_letter('E');
            disp_write_letter('N');
            disp_write_letter('T');
            disp_write_letter('E');
            disp_write_letter('R');
            disp_write_letter(' ');
            disp_write_letter('#');
            disp_write_letter(' ');
            disp_write_letter('C');
            disp_write_letter('R');
            disp_write_letter('D');
            disp_write_letter('S');
            disp_write_letter(':');
            disp_write_letter(' ');
}
```