Embedded C

Week 2: Write to Serial - Read Input – Pointers - Interrupts

KdG Karel de Grote Hogeschool

2020-2021

Overzicht

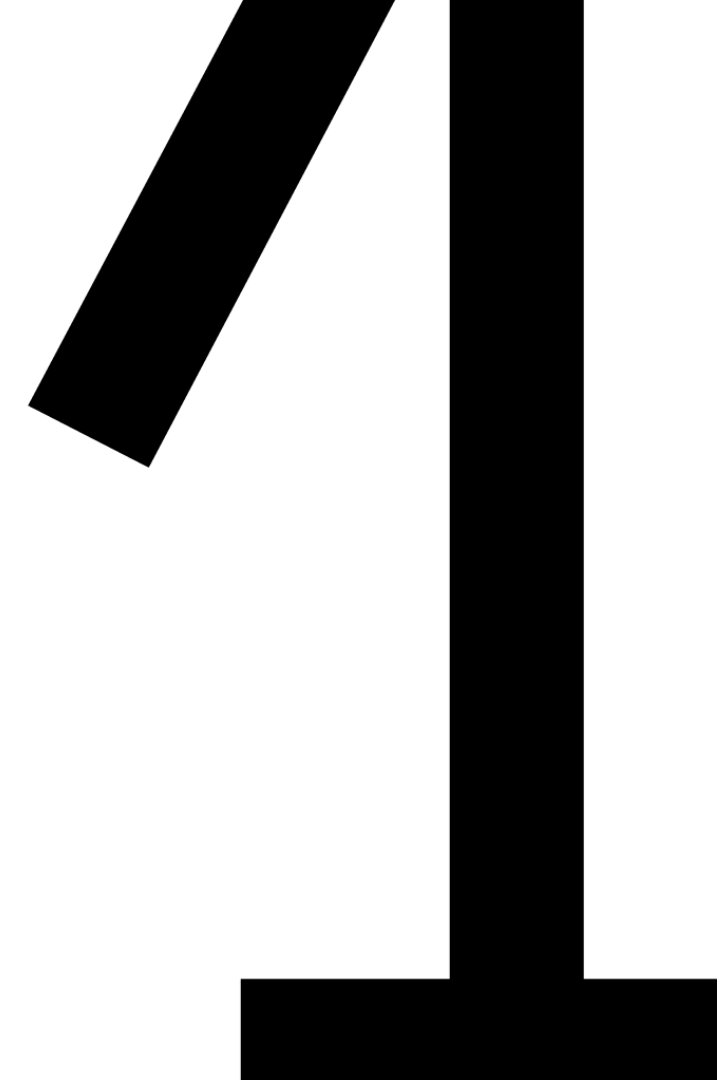
- Week 1: Hello Arduino - Basics - LED
- Week 2: Buttons - Interrupts
- Week 3: Potentiometer – the LED Screen
- Week 4: Timers - Sound
- Week 5: Integratie en individueel project
- Week 6: Integratie en individueel project

Content

- Overzicht Week 2
- Demo 1 – Write to serial
- Demo 2 – Read input from buttons
- Demo 3 – Read input from buttons with macros
- Button library
- Demo 4 – Pointers
- Demo 5 – Interrupts and buttons
- Extra oefening
- Week project

Week 2 - Embedded

- Overzicht
- Tutorials
- Demos
- Oefeningen
- Weekproject



Week 2 - Overzicht

Overzicht

- Schrijven naar de seriële poort:
  - de USART lib

- Buttons:
  - hoe lees ik een pin uit?
  - Efficiëntere buttons met Interrupts

- De programmeertaal C:
  - Macro's
  - Pointers - Call by Value - Call by Reference

Week 2 - Tutorials

Overzicht

1 – Interfacing met de Hardware
- 1.1 – Lezen van een pin
- 1.2 – Interrupts

2 – De Programmeertaal C
- 2.1 – Preprocessor en macro’s
- 2.2 – Pointers – ByVal & ByRef

3 – Overige Tutorials
- 3.1 – Wat is een pull-up weerstand
- 3.2 – Wat is een seriële poort

Week 2 - Demos

Overzicht

1 – **Write to serial**

Door de externe library usart te gebruiken kunnen we communiceren over de seriële poort

2 – **Read input from buttons**

Hoe kan je detecteren of een knop ingedrukt wordt?

3 – **Read input from buttons with macros**

Idem, maar nu door gebruik te maken van macro's

4 – **Pointers**

Hoe gebruik je pointers in C?

5 – **Interrupts and buttons**

Het gebruik van het interrupt systeem van de AVR om te reageren op de druk op een knop

Week 1 - Oefeningen


Oefening


Overzicht

Button library maken

Schrijf een kleine bibliotheek (button.h/button.c) die een aantal handige button functies implementeert die je in toekomstige projecten kan hergebruiken

Oefeningen op het gebruik van pointers in C

Kleine basisoefeningen

Timing en chronologie van ISR

Maak een programma waarmee je de timing en chronologie van ISR's (Interrupt Service Routines) kunt aantonen.

Extra: Combineer LED’s met buttons

LEDs aan of uit op basis van wat welke button(s) je indrukt.

Week 2 - Weekproject

Opdracht Overzicht

Simon Says

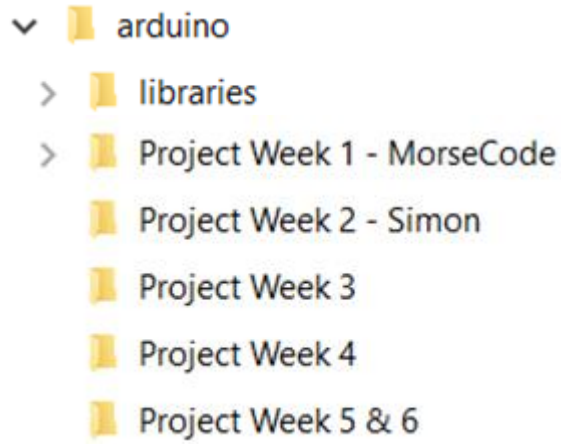
We werken het bekende spel Simon Says uit. In onze versie "speelt" de Arduino een telkens langer wordend patroon van lichtjes af, de speler moet proberen dat patroon na te spelen met de knopjes. Het patroon heeft een maximale lengte van 10. Wanneer de speler erin slaagt om dat te halen is hij de Simon Master.

Je maakt gebruik van functies en ook #define kan handig zijn om dit project goed leesbaar te houden. Uiteraard kan je ook je zelfgeschreven libraries (led.h en button.h) gebruiken!

Pak de oefening adhv volgende stappen aan:

- Opgave 1: De opstart
- Opgave 2: Seeden van de randomgenerator
- Opgave 3: Aanmaken van de random reeks
- Opgave 4: Afspelen van de puzzle
- Opgave 5: Uitlezen van de input van de gebruiker
- Opgave 6: Het volledige spel

Niet vergeten: Portfolio



Demo 1 – Write to serial

Demo 1 – Write to serial - C-code

We laten het eerste LED-lampje (LED nummer 0) aan en uit pinken en we tellen hoeveel keer het lampje aan- en uitging. Hierbij schrijven we het aantal uit op de serial monitor (via de seriële poort -USB kabel-)

```
#include <util/delay.h>
#include <avr/io.h>
#include <LED.h>
#include <usart.h>

int main()
{
    enableOneLed(1); // DDRB |= (1 << PB2+ (4-1)); in register B wordt de 3de bit nl PB2 op 1 –output- gezet
    initUSART
    int counter = 0;
    while (1)
    {
        lightDownOneLed(1); // PORTB = (1 << PB2 + (4-1));
        _delay_ms(100);
        lightUpOneLed(1);   // PORTB = (0 << PB2 + (4-1));
        _delay_ms(100);
        printString("Debugging!!");
        printf("Countertje: %d\n", counter);
        counter++;
    }
    return 0;
}
```

Demo 1 – Write to serial - Achtergrond

Demo 1 – Blinking LED - Achtergrond

LED Library

Demo 1 – Blinking LED - Achtergrond

Week 1 – Weekproject - usart

Demo 1 – Blinking LED - Achtergrond

Week 1 – Weekproject - array

De microcontroller ATMega328 communiceert via zijn pins met de buitenwereld. Dit gebeurt door een spanning op één van de pins aan of af te zetten. De pins van de microcontroller zijn opgedeeld in 3 poorten: PORTB, PORTC en PORTD.
Elk van deze poorten bevat een aantal pins. Zo heeft
- PORTB beschikking over 8 pins (PB0-PB7),
- PORTC heeft 7 pins (PC0-PC6) en
- PORTD heeft er weer 8 (PD0-PD7).

De ATMega328 heeft per PORT drie registers ter beschikking:

- Het **DDRx** register (**D**ata **D**irection **R**egister): *hierin kunnen we aangeven of we een bepaalde pin voor input of voor output willen gebruiken.*
- Het **PORTx** register (**P**in **O**utput **R**egis**t**er): *hierin kunnen we effectief dan spanning aan of af zetten op de juiste pins.*
- *Het **PINx** register* (**P**in **In**put Register): *dit gebruiken we om input uit te lezen.*

Array van char

Zie Canvas Tutorials C-taal

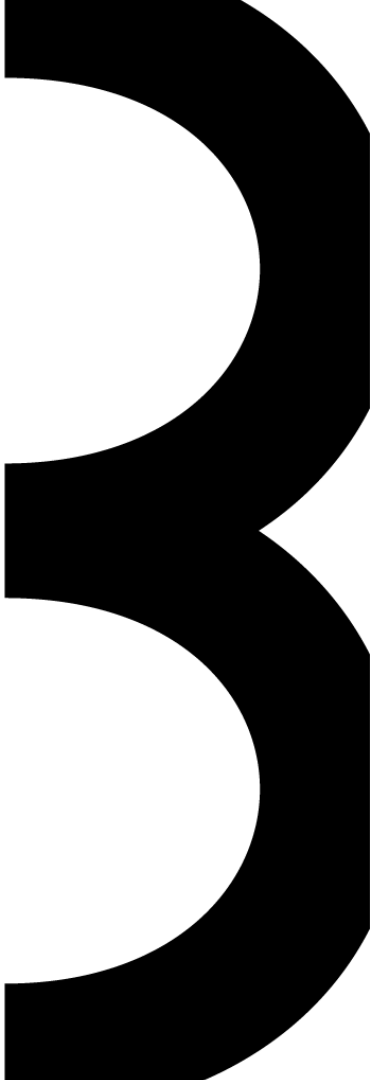
Tip1:

```
char stringArray[2];
stringArray[1] = '\0';
```

Tip 2:

Serial monitor is terug te vinden op de bank onderaan VSCode:

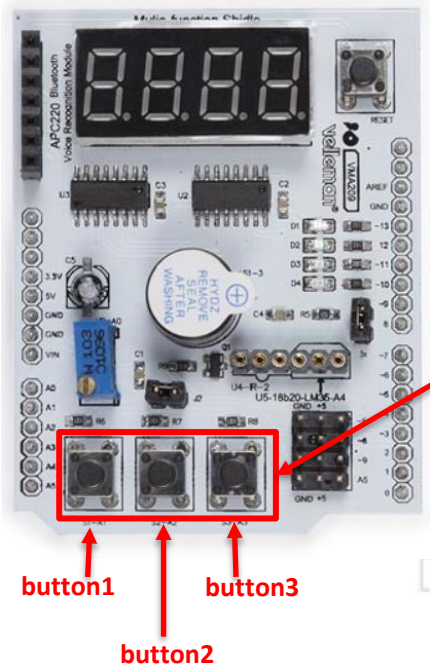
Demo 2 – Read input from buttons



Demo 2 – Read input from buttons - C-code

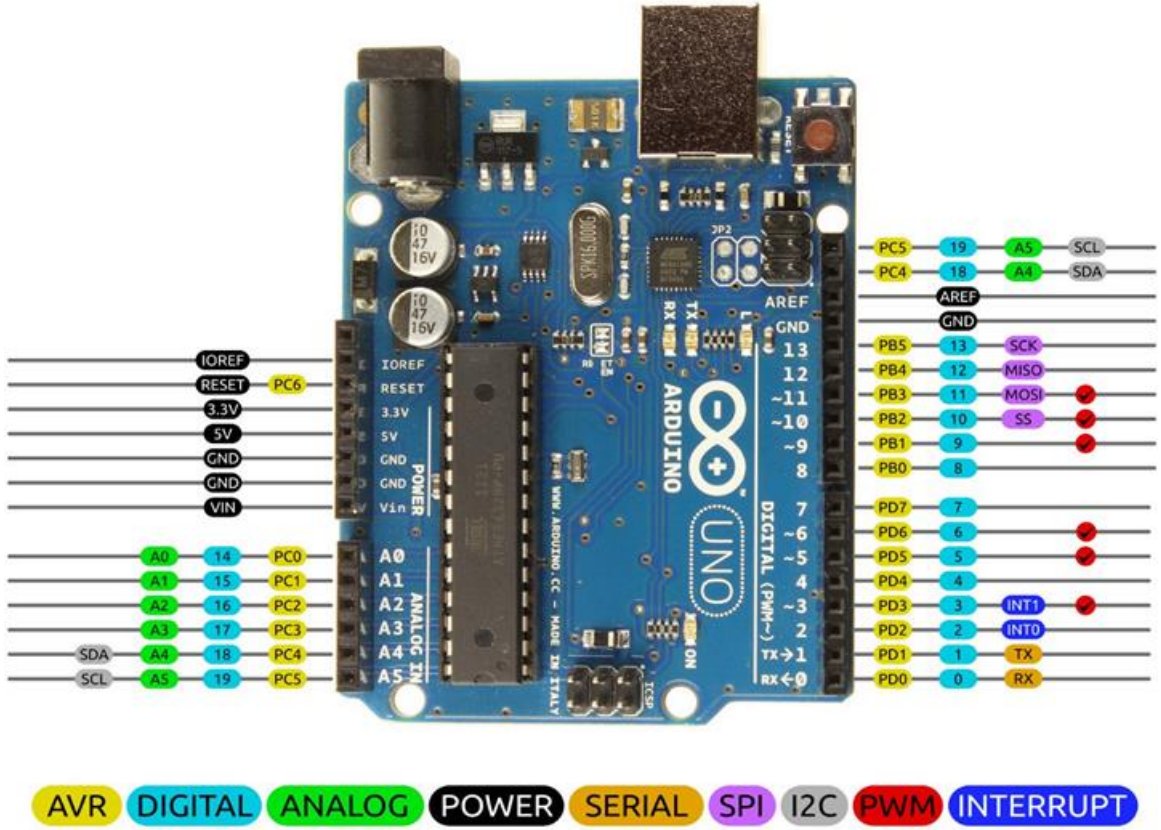
```
#include <util/delay.h>
#include <avr/io.h>
#include <usart.h>

int main() {
  initUSART();
  DDRC &= ~(1 << PC1);              // Button 1 hangt aan C1, zetten de tweede bit –PC1- in het DD Register op 0 –input-
  printf("DDRC: ");  printBinaryByte(DDRC); // usart functies
  PORTC |= (1 << PC1);                      // Pull up aanzetten van C1: PINC komt op high te staan –zie tutorial-
  printf("\nPORTC: ");  printBinaryByte(PORTC);
  while (1) {
    printf("\nPINC: ");  printBinaryByte(PINC);
    if ((PINC & (1 << PC1)) == 0) {          // Testen of de PC1-e bit van PINC op 0 staat -knop ingedrukt-.
      printf(" - Button 1 pushed!\n");
    }
    else {
      printf(" - Button 1 NOT pushed!!\n");
    }
    _delay_ms(1000);
  }
  return 0;
}
```

Demo 2 – Read input from buttons - Achtergrond

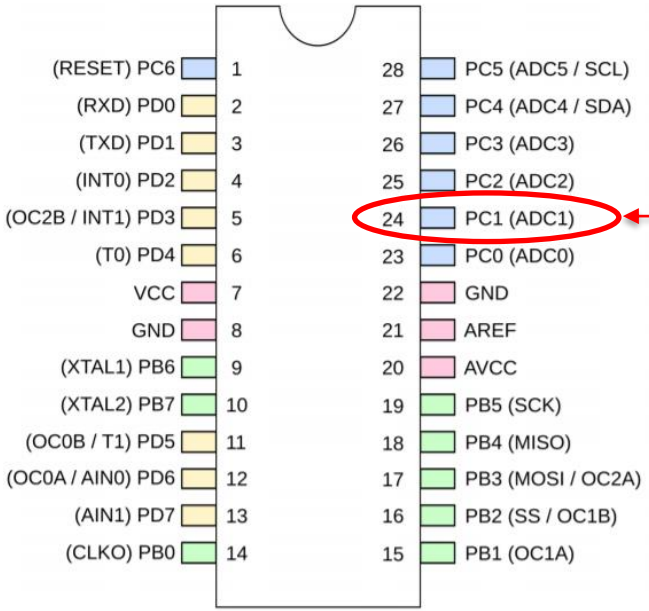


4 red LEDs	10, 11, 12, 13
3 buttons + reset button	A1, A2, A3
potentiometer (10 kΩ)	A0
4-digit, 7-segment LED tube driven by 74HC595	latch 4, clock 7, data 8
buzzer	3 (digital on-off)
socket for IR receiver (remote control)	2
socket for temperature sensor LM35 or DS18B20 (polarity!)	A4
header for APC220 shield	GND, +5V, 0, 1 (RX/TX)
free pins (PWM)	5, 6, 9, A5

Demo 2 – Read input from buttons - Achtergrond



Demo 2 – Read input from buttons - Achtergrond

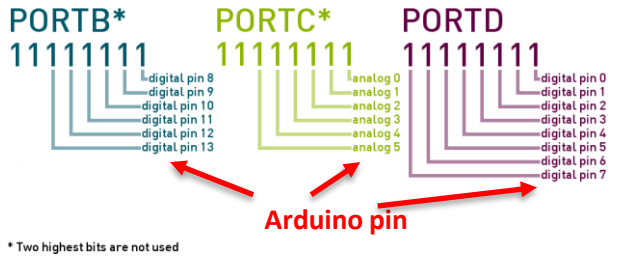


Multi-Function shield V2	Digital pin (Dn)	Analog pin (An)	AVR pin	AVR port	AVR function(s)	AVR PWM
Bluetooth header - tx	0		2	PD0	RxD	
Bluetooth header - rx	1		3	PD1	TxD	
IR receiver	2		4	PD2	INT0	
Buzzer	3		5	PD3	INT1, OC2B	Yes
7-seg display Pin latch	4		6	PD4	T0, XCK	
header	5		11	PD5	T1	Yes
header	6		12	PD6	AIN0	Yes
7-seg display clock	7		13	PD7	AIN1	
7-seg display data	8		14	PB0	CLK0, ICP1	
	9		15	PB1	OC1A	Yes
Red LED	10		16	PB2	OC1B, SS	Yes
Red LED	11		17	PB3	OC2A, MOSI	Yes
Red LED	12		18	PB4	MISO	
Red LED	13		19	PB5	SCK	
Variable resistor	14	0	23	PC0		
Switch 1	15	1	24	PC1		
Switch 2	16	2	25	PC2		
Switch 3	17	3	26	PC3		
DS18B20 header	18	4	27	PC4	SDA	
header	19	5	28	PC5	SCL	

Demo 2 – Read input from buttons - Achtergrond

De microcontroller ATMega328 communiceert via zijn pins met de buitenwereld. Dit gebeurt door een spanning op één van de pins aan of af te zetten. De pins van de microcontroller zijn opgedeeld in 3 poorten: PORTB, PORTC en PORTD.
Elk van deze poorten bevat een aantal pins. Zo heeft

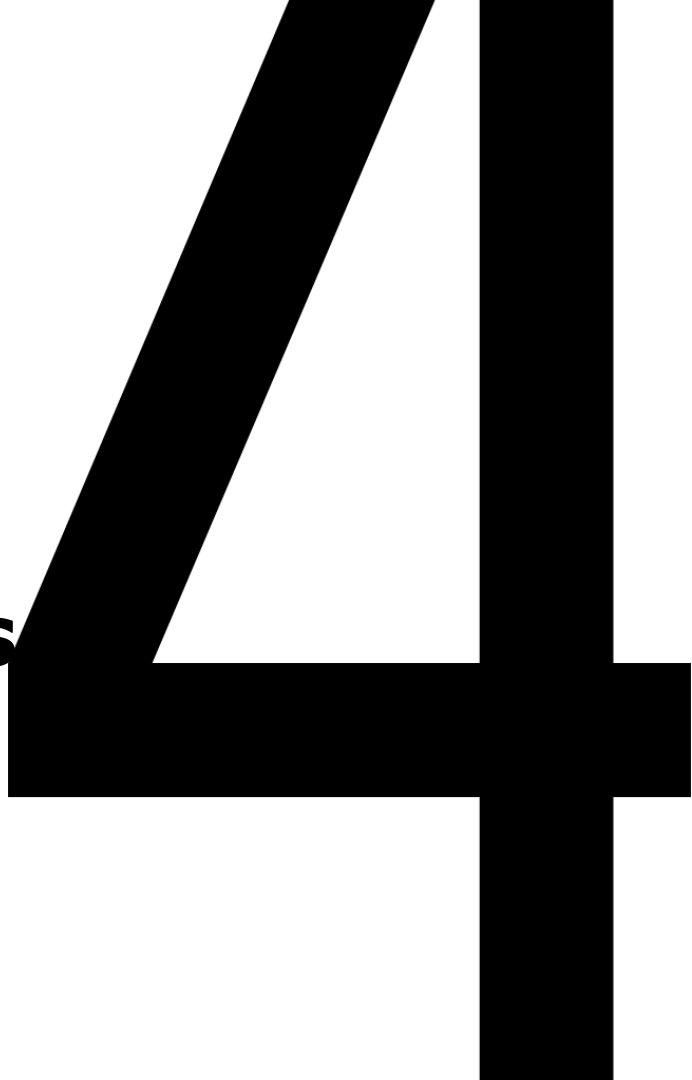
- PORTB beschikking over 8 pins (PB0-PB7),
- PORTC heeft 7 pins (PC0-PC6) en
- PORTD heeft er weer 8 (PD0-PD7).



De ATMega328 heeft per PORT drie registers ter beschikking:

- Het **DDRx** register (**D**ata **D**irection **R**egister): *hierin kunnen we aangeven of we een bepaalde pin voor input (0) of voor output (1) willen gebruiken.*
- Het **PORTx** register (**P**in **O**utput **R**egis**t**er): *hierin kunnen we effectief dan spanning aan of af zetten op de juiste pins.*
- *Het **PINx** register* (**P**in **In**put Register): *dit gebruiken we om input uit te lezen.*

Demo 3 – Read input from buttons with macros



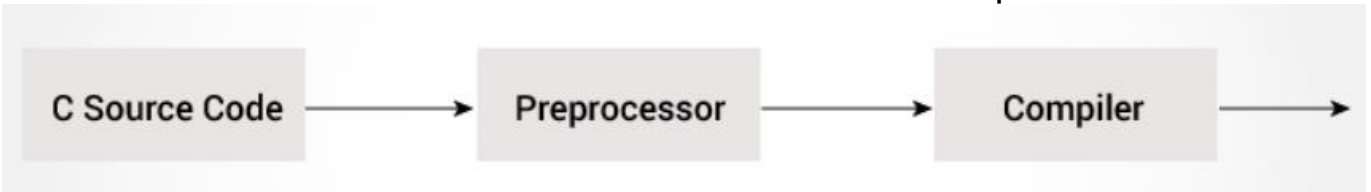
Demo 3 – Read input from buttons with macros (*) - C-code

```
#include <util/delay.h>
#include <avr/io.h>
#include <usart.h>
//gebruiken voorgedefinieerde macro's, o.a. _BV en bit_is_clear
//Ze zijn gedefinieerd in avr/sfr_defs.h en worden ge-include in avr/io.h
int main() {
  initUSART();
  DDRC &= ~ _BV(PC1);                          // _BV-macro ipv (1 << PC1) (BV- staat voor "Bit Value")
  printf("DDRC: "); printBinaryByte(DDRC);
  PORTC |= _BV(PC1);                           //Pull up aanzetten van C1 met _BV macro ipv (1 << PC1)
  printf("\nPORTC: "); printBinaryByte(PORTC);
  while (1) {
    printf("\nPINC: "); printBinaryByte(PINC);
    if (bit_is_clear(PINC, PC1)) {             //bit_is_clear macro ipv (!(PINC & (1 << PC1)))
      printf(" - Button 1 pushed!\n");
    }
    else {
      printf(" - Button 1 NOT pushed!!\n");
    }
    _delay_ms(1000);
  }
  return 0;
}
```

-(*) een **macro** is een stukje code waar een naam werd aan gegeven

Demo 3 – Read input from buttons with macros - Achtergrond

Voor de C-code gecompileerd wordt, wordt er een preprocessor op toegepast. De preprocessor zal de C-code 'transformeren' vooraleer de compiler er mee aan de slag gaat.



Het 'transformeren' bestaat o.a. uit het toevoegen van header files, constanten invullen,…

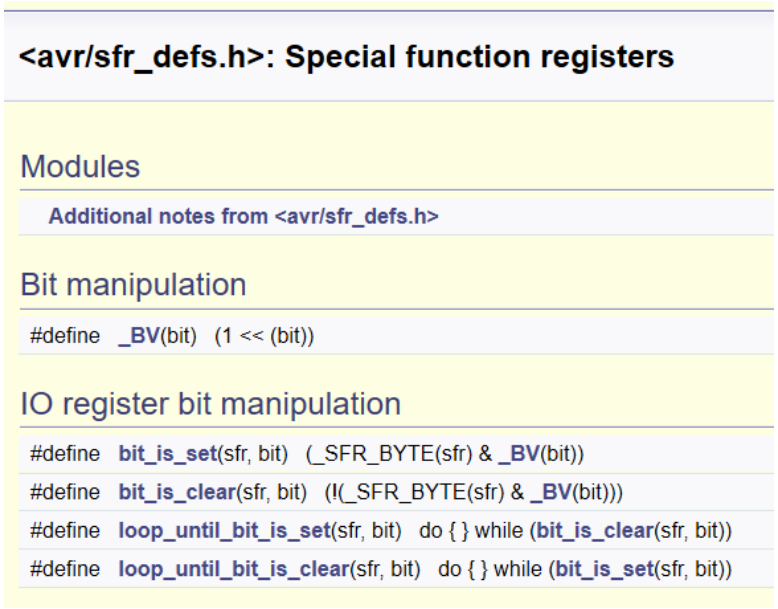
Met een #-symbol duid je in je C code aan wat de preprocessor moet doen. Vbn:

`#include <usart.h>` ⇨ toevoegen van de inhoud van de header-file usart.h

`#define NUMBER_OF_LEDS 4` ⇨ vervangt overal in de code NUMBER_OF_LEDS door 4

Demo 3 – Read input from buttons with macros - Achtergrond

Gebruikte macro's: zie: http://www.nongnu.org/avr-libc/user-manual/group__avr__sfr.html



<avr/sfr_defs.h>: Special function registers

Modules

Additional notes from <avr/sfr_defs.h>

Bit manipulation

```
#define _BV(bit) (1 << (bit))
```

IO register bit manipulation

```
#define bit_is_set(sfr, bit) (_SFR_BYTE(sfr) & _BV(bit))
#define bit_is_clear(sfr, bit) (!(_SFR_BYTE(sfr) & _BV(bit)))
#define loop_until_bit_is_set(sfr, bit) do { } while (bit_is_clear(sfr, bit))
#define loop_until_bit_is_clear(sfr, bit) do { } while (bit_is_set(sfr, bit))
```

Button Library

Week 2 - Oefening

Oefening

Button library maken

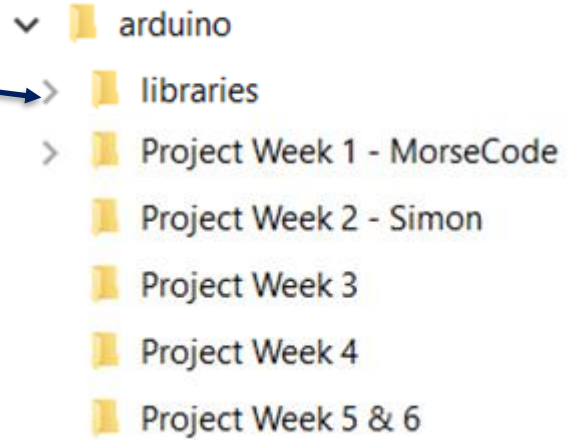
Schrijf een kleine bibliotheek (button.h/button.c) die een aantal handige button functies implementeert die je in toekomstige projecten kan hergebruiken

```
void enableOneButton(int);          // het meest linkse knopje op je shield = button 1
void enableMultipleButtons(uint8_t);
void enableAllButtons ();

void disableAllButtons ();

int buttonPushed(int);    // het meest linkse knopje op je shield = button 1
int multipleButtonsPushed (uint8_t);
int allButtonsPushed ();

int buttonReleased(int);  // het meest linkse knopje op je shield = button 1
int multipleButtonsReleased (uint8_t);
int allButtonsReleased ();
```

Week 2 - Oefening

Button Library aanmaken:

- Maak de subfolder *button* aan in de folder *..\arduino\libraries*
- Plaats twee bestanden in de subfolder *button*:
  - ✓ Button.h – bevat enkel de headers voor de button-functies
  - ✓ Button.c – bevat de implementatie van de button-functies



Demo 4 – Pointers

Demo 4 – Pointers - C-code

```
#include <util/delay.h>
#include <avr/io.h>
#include <usart.h>

void increment_byref(int* a){
   printf("Incrementing a (%d)\n", *a);
   (*a)++;
}

void increment_byval(int a){
   printf("Incrementing a (%d)\n",a);
   a++;
}

void array_as_parameter(int* p){
  //hoe groot is de array in bytes?
  printf("sizeof array as parameter:%d\n",sizeof(p));
}
```

```
int main(){
  initUSART();
  int a = 10;
  int* b = &a;
  int c[] = {1,2,3};
  int d[] = {1,2,3,4,5,6};
  printf("Value of a: %d\n",a);
  printf("sizeof a:%d\n",sizeof(a));  //aantal bytes voor een int (ATmega328)
  printf("Value of &a: %p\n",&a);     // adres waar is a opgeslagen
  printf("Value of b: %p\n",b);       //b bevat het adres van a
  printf("sizeof b:%d\n",sizeof(b)); //hoe groot is een int pointer?
  printf("Value of &b: %p\n",&b);   //en wat is het adres van b zelf?

  printf("Value of c: %p\n",c);       //array is een pointer naar 1° element
  printf("Value of &c[0]: %p\n",&c[0]); //hetzelfde als adres 1° element
  printf("sizeof c:%d\n",sizeof(c)); //hoe groot is de array in bytes?

  array_as_parameter(c);            //geef array als parameter mee
  printf("Value of d: %p\n",d);
  printf("sizeof d:%d\n",sizeof(d));
  increment_byval(a);
  printf("Value of a after by val increment: %d\n",a);
  increment_byref(&a);
  printf("Value of a after by ref increment: %d\n",a);
  return 0;
}
```

```
Value of a: 10
sizeof a:2
Value of &a: 0x8f2
Value of b: 0x8f2
sizeof b:2
Value of &b: 0x8f4
Value of c: 0x8ec
Value of &c[0]: 0x8ec
sizeof c:6
sizeof array as parameter:2
Value of d: 0x8e0
sizeof d:12
Incrementing a (10)
Value of a after by val increment: 10
Incrementing a (10)
Value of a after by ref increment: 11
```

Demo 4 – Pointers - Achtergrond

Pointer = variabele die een geheugenadres kan bevatten

- Declaratie: met de * operator (Indirection operator)
- Voorbeeld: `int* b;`
  - → b kan het geheugenadres van een `int` variabele bevatten
  - → We zeggen: **"b is een pointer naar een int"**
- Gebruik:

```
int a = 10;
int* b;
b = &a;
a++;      // wat is de inhoud van a en b?
(*b)++;   // wat is de inhoud van a en b?
```

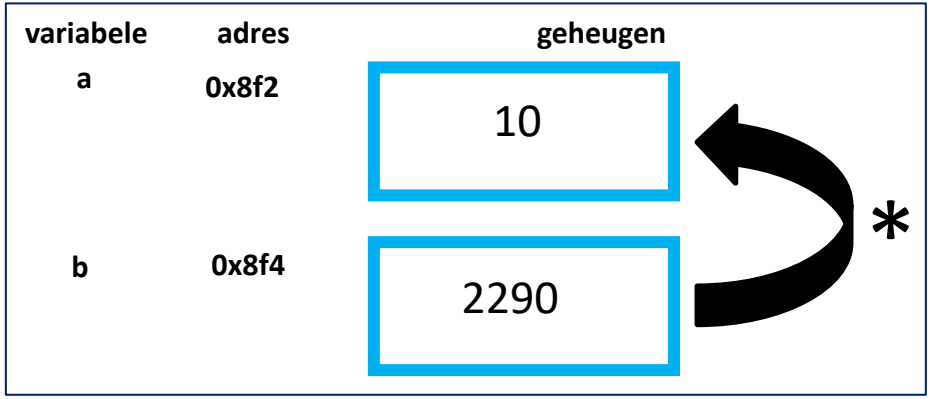
Pointer-operatoren:

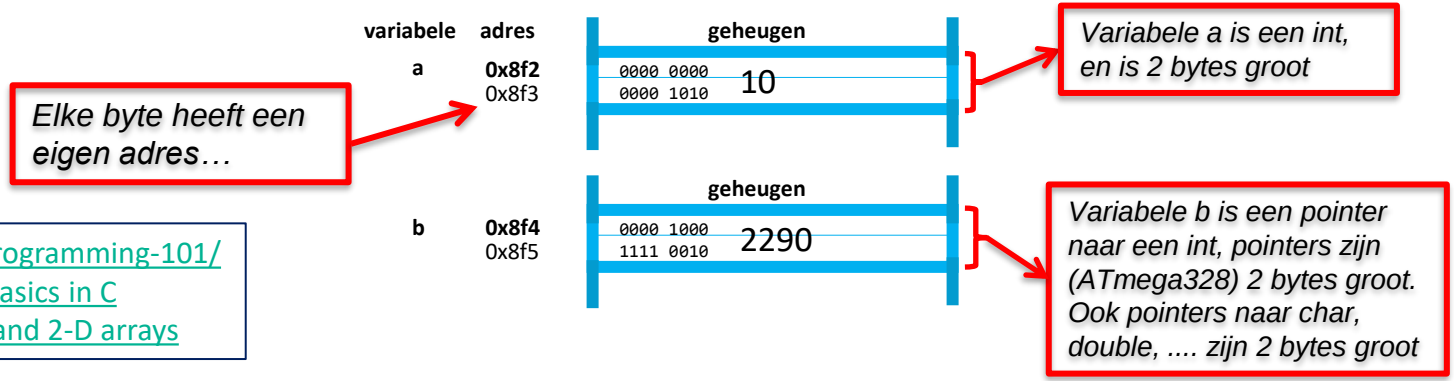
&: "adres van variabele"
&a: het adres van variabele a

***:** "variabele met adres"
*b: de variabele met als adres b

Demo 4 – Pointers - Achtergrond

```
int a = 10;
int* b;
b = &a;
```

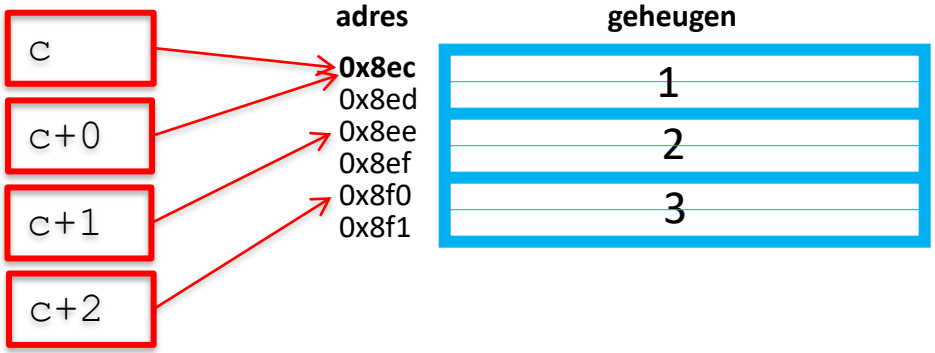




Tip: https://overiq.com/c-programming-101/
⇨ Pointers ⇨ Pointer Basics in C
⇨ Pointers and 2-D arrays

Demo 4 – Pointers – Achtergrond – 1 dim arrays

```
int c[] = {1, 2, 3};
```



c = naam van array (variabele...)

c = **constante pointer** naar het begin van de array

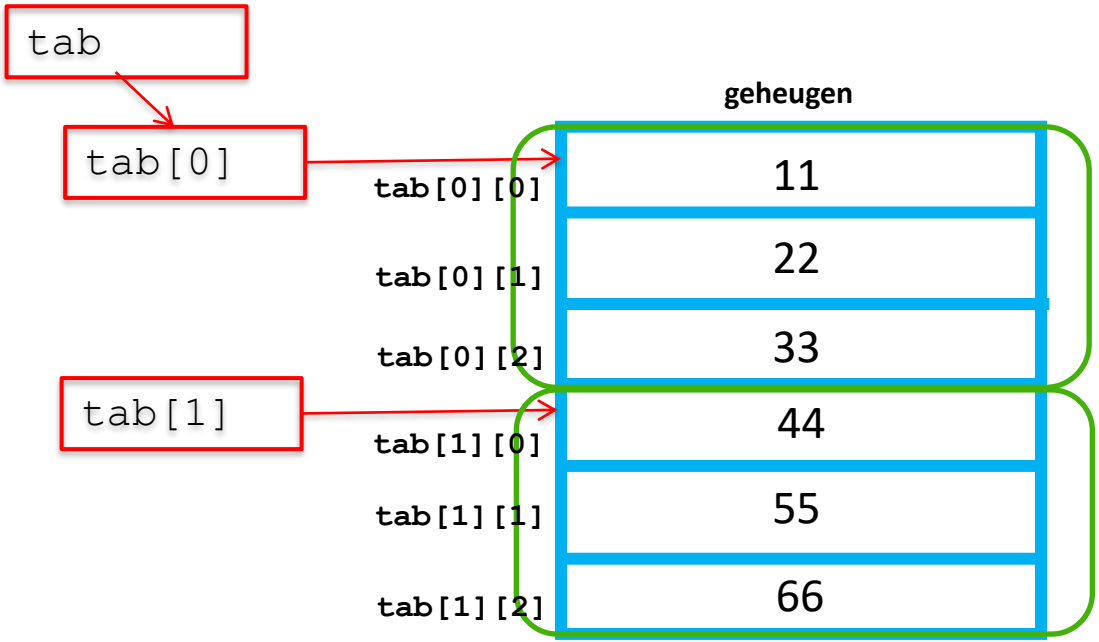
c *kan dus niet veranderen van waarde!*

c[0]	1	*(c+0)=*c	1
c[1]	2	*(c+1)	2
c[2]	3	*(c+2)	3

Tip: https://overiq.com/c-programming-101/
⇨ Pointers ⇨ Pointers and 1-D arrays

Demo 4 – Pointers – Achtergrond – 2 dim arrays

```
int tab[2][3] = { {11, 22, 33}, {44, 55, 66} };
```




```
tab  = array van 2 arrays van int

tab[0] = array van 3 int
       = *(tab+0)

tab[1] = array van 3 int
       = *(tab+1)
```

1	tab[0][0]	*tab[0]	**tab	*(*tab+0)
2	tab[0][1]	*(tab[0]+1)	*(*tab+1)	*(*tab+1)
3	tab[0][2]	*(tab[0]+2)	*(*tab+2)	*(*tab+2)
4	tab[1][0]	*tab[1]	*(*(tab+1))	*(*tab+3)
5	tab[1][1]	*(tab[1]+1)	*(*(tab+1)+1)	*(*tab+4)
6	tab[1][2]	*(tab[1]+2)	*(*(tab+1)+2)	*(*tab+5)

tab[i][j] == *(tab[i]+j) == *(*(tab+i)+j)

Tip: https://overiq.com/c-programming-101/
⇨ Pointers ⇨ Pointers and 2-D arrays

Demo 4 – Pointers - Achtergrond

```
void increment_byval(int a){
    printf("Incrementing a (%d)\n",a);
    a++;
}

int main() {
  int a = 10;
  increment_byval(a);
  printf("Value of a after by val increment: %d\n",a);
  return 0;
}
```

*De parameter **a** is een locale kopie van de variabele a: enkel de inhoud wordt doorgegeven.*

*We noemen dit: "**call-by-value**"*

*Bij het doorgeven van waardes via parameters, wordt er op de 'stack' (volgende week meer hierover) een copy van de variabele a gezet. Je verkrijgt in de functie increment_byval dus een locale variabele **a** met als (begin) waarde 10.*

Demo 4 – Pointers - Achtergrond

```
void increment_byref(int* a){
    printf("Incrementing a (%d)\n",*a);
    (*a)++;
}

int main() {
  int a = 10;
  increment_byref(&a);
  printf("Value of a after by ref increment: %d\n",a);
  return 0;
}
```

*De parameter **a** bevat het geheugenadres van de variabele a.*

*We noemen dit: "**call-by-reference**"*

*Bij het doorgeven van waardes via parameters, wordt er op de 'stack' een copy van &a gezet. Je verkrijgt in de functie increment_byref dus een locale variabele **a** (=pointer naar een int) dat een kopie van het adres van **a** bevat.*

Tip: https://overiq.com/c-programming-101/
⇨ Pointers ⇨ Call by Value and Call by Reference in C

Week 2 - Oefening

Oefeningen op het gebruik van pointers in C

1) Maak een nieuw project "*pointerOef_1*"
   - Definieer 2 macro's: **MAX** om het aantal elementen van een array vast te leggen (bijvoorbeeld op 5) en **ZEVENVOUD** om van een bepaald getal het zevenvoud te berekenen.
   - In de **main** maak je een array van MAX gehele getallen, die je meteen met 0 initialiseert.
   - Roep vanuit de main de functie **printArray** op en geef de array als parameter mee. Je drukt van de array alle elementen af, én ook de adressen waar die elementen zich bevinden (zie voorbeeldafdruk hieronder).
   - Roep vanuit de main de functie **maakArray** op. Daar vul je de array met zevenvouden.
   - Roep opnieuw de functie **printArray** op om de gewijzigde inhoud van de array te controleren.

2) Maak een nieuw project "*StringOef2*"
   - Doe een include van **string.h** (voor stringfuncties)
   - Declareer in de main een **array van 7 strings van 10 karakters** en zet daarin de namen van je vrienden. Liefst als volgt:
     char namen[MAX][10] = {"Hans", "Anniek", "Wim", ... };
   - Maak een functie **drukEersteLetters**, die de 7 eerste letters van de namen op één regel afdrukt.
   - Maak een functie **drukLaatsteLetters**, die de 7 laatste letters van de namen op één regel afdrukt.
   - Schrijf een functie **zoekKleinsteNaam**, die de alfabetisch kleinste naam als returnvalue teruggeeft. De oproep vanuit de main doe je als volgt: `char* kleinste = zoekKleinsteNaam(namen);`
     Druk deze naam in de main af.

Tip: https://overiq.com/c-programming-101/
⇨ Pointers ⇨ Passing 2-D Array to a Function in C

Demo 5 – Interrupts & buttons

Demo 5 – Interrupts & buttons - C-code

```
#include <avr/interrupt.h>
#include <avr/io.h>
#include <util/delay.h>
#include <LED.h>


/* Deze ISR wordt aangeroepen als Pin Change Interrupt 1 afgaat
   (PCINT1_vect)
   Dit is de interrupt voor PORTC, die waarop de knopjes hangen...
*/
ISR(PCINT1_vect) {
   if (bit_is_clear(PINC, PC1)) {// button 1 is ingedrukt (bit op 0)?
      _delay_us(1000);          // We wachten 1000 µseconden en
                               // checken dan nog eens (debounce!)
   if (bit_is_clear(PINC, PC1)) { // button 1 is ingedrukt (bit op 0)?
      lightInverseOneLed(2);
   }
  }
}
```

```
int main() {
  enableOneLed (1);      // we gaan led1 & led2 gebruiken
  enableOneLed (2);
  lightDownOneLed (1);  // de 2 leds uitzetten
  lightDownOneLed (2);
  DDRC &= ~_BV(PC1); // we gaan knop 0 gebruiken
  PORTC |= _BV(PC1);   // pull up aanzetten
  PCICR |= _BV(PCIE1); /* in Pin Change Interrupt Control
          Register: geef aan welke interrupt(s) je wil activeren
          (PCIE0: poort B, PCIE1: poort C, PCIE2: poort D) */
  PCMSK1 |= _BV(PC1); /* in overeenkomstig Pin Change
           Mask Register: geef aan welke pin(s) van die poort
           de ISR activeren */
  sei();                    /* Set Enable Interrupts --> globaal
                               interrupt systeem aanzetten */

  while (1) {
    lightUpOneLed (1);
    _delay_ms(100);
    lightDownOneLed (1);
    _delay_ms(100);
   }
  return 0;
}
```

Demo 5 – Interrupts & buttons - Achtergrond

Wat is een interrupt?

Een ‘interrupt’ (letterlijk vertaald: ‘onderbreking’) is een verzoek om aandacht van een hardwarecomponent aan een andere component, meestal een processor.
De processor stopt bij een interrupt met de uitvoering van het programma, voert de code geassocieerd met de interrupt uit en gaat daarna verder met de uitvoering van het programma

Hoe wordt een interrupt behandeld?

Interrupts worden opgevangen door een ***interrupt handler***, ook wel ***Interrupt Service Routine*** (ISR) genoemd. Dit is een stukje code dat gekoppeld wordt aan een specifieke interrupt conditie.

Hoe verlopen interrupts voor de ATmega328?

In het geheugen wordt een ***Interrupt Vector Table*** bijgehouden waarin voor elk van de (meer dan 20) interrupts staat welke routine (ISR) er uitgevoerd moet worden zodra de interrupt wordt geactiveerd.

Demo 5 – Interrupts & buttons - Achtergrond

Welke interrupts zijn van belang?

In dit vak zijn we voornamelijk geïnteresseerd in de zogenaamde ***Pin Change Interrupts*** (PCINTx_vect) die op poort B, C en D kunnen plaats vinden.

Hoe wordt een interrupt (op Arduino/ATmega328) geconfigureerd?

1. Configureer de volgende registers
   - In het ***Pin Change Interrupt Control Register***(PCICR) geef je aan in welke poort je geïnteresseerd bent. Voor poort B zet je bit PCIE0 op 1, voor poort C, PCIE1 en voor poort D, PCIE2.
   - In het ***Pin Change Mask Register*** (PCMSKx) geven we aan in welke pin van die poort we geïnteresseerd zijn. Voorbeeld= button 1 hangt op PC1
2. Schrijf de bijpassende code (ISR). Dit is in feite een macro met als naam ISR en met de pin change interrupt als parameter:

```
ISR(PCINTx_vect) {
   //hier komt de code
}
```

3. Zet het globaal interrupt systeem aan mbv de macro sei() (Set Enable Interrupt bit)

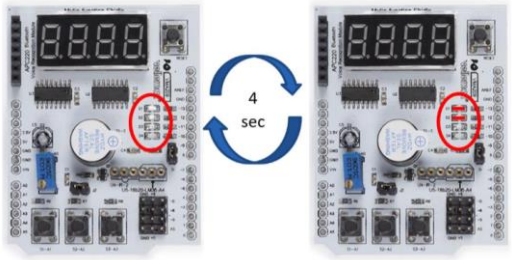
Week 2 - Oefening

Oefening

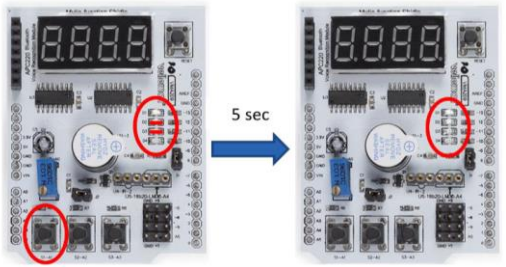
Timing en chronologie van ISR

Maak een programma waarmee je de timing en chronologie van ISR's (Interrupt Service Routines) kunt aantonen.

Bij het starten van dit programma beginnen de LEDs D1 en D2 samen aan en uit knipperen met een interval van 4”
(2 seconden gezamenlijk aan, 2 seconden gezamenlijk uit).

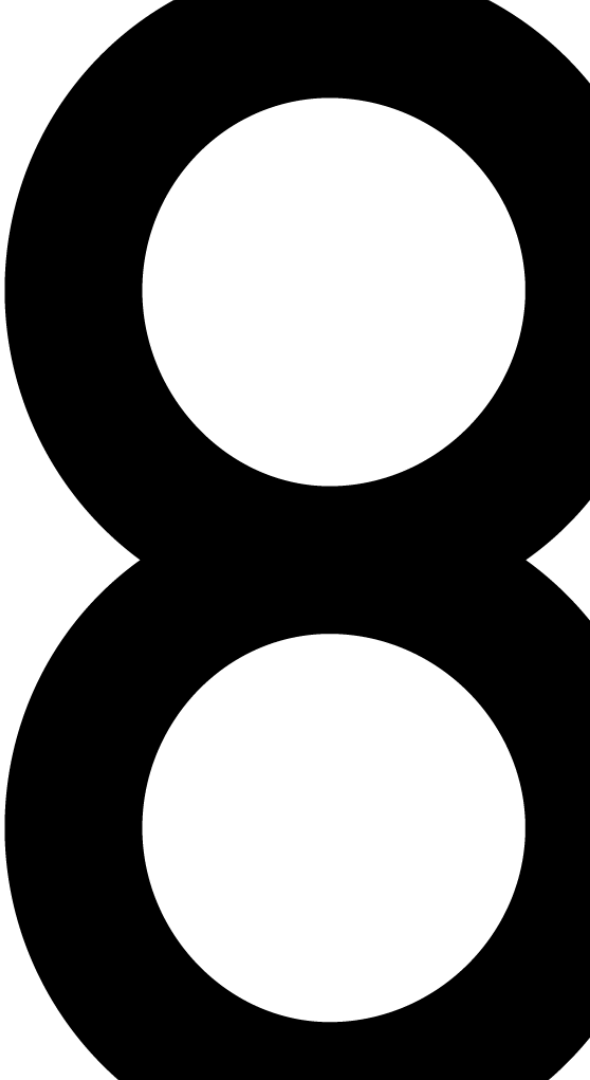


Wanneer de gebruiker op knop S1 drukt, start een bijbehorende ISR. In de ISR staat code die ervoor zorgt dat LED's D2 en D3 samen éénmalig aan en uitknipperen met een interval van 10” (5 seconden uit, 5 seconden aan)



Beantwoord de vragen (zie opgave Canvas).
Maak ook de uitbreiding.

Extra oefening



Week 2 - Oefening

Oefening

Combineer LED’s met buttons

Voor deze oefening werk je verplicht met interrupts. Maak ook zoveel mogelijk gebruik van je zelfgeschreven LED-library en Button-library.

Je schrijft een programmaatje waarbij de 3 knoppen een ander resultaat op de leds genereren:

- Druk je op knop 1, dan begint led 1 te knipperen, druk je nogmaals dan gaat het uit.
- Druk je op knop 2, dan begint led 2 te knipperen, druk je nogmaals dan gaat het uit.
- Druk je op knop 3, dan begint led 3 te knipperen, druk je nogmaals dan gaat het uit.

Opgelet: druk ik bijvoorbeeld op knop 1 en op knop 2, dan verwacht ik dat zowel led 1 als led 2 knipperen!

Week project

Week 2 - Weekproject

Opdracht

Simon Says

We werken het bekende spel Simon Says uit. In onze versie "speelt" de Arduino een telkens langer wordend patroon van lichtjes af, de speler moet proberen dat patroon na te spelen met de knopjes. Het patroon heeft een maximale lengte van 10. Wanneer de speler erin slaagt om dat te halen is hij de Simon Master.

Je maakt gebruik van functies en ook #define kan handig zijn om dit project goed leesbaar te houden. Uiteraard kan je ook je zelfgeschreven libraries (led.h en button.h) gebruiken!

Pak de oefening adhv volgende stappen aan:

- Opgave 1: De opstart
- Opgave 2: Seeden van de randomgenerator
- Opgave 3: Aanmaken van de random reeks
- Opgave 4: Afspelen van de puzzle
- Opgave 5: Uitlezen van de input van de gebruiker
- Opgave 6: Het volledige spel

(zie de opgave op Canvas voor meer details)

Tips:

- Geen oneindige lus nodig, laat het spel 1 x spelen en geef op het einde de boodschap dat men op de reset-knop moet drukken om het spel opnieuw te spelen
- Laat bij het indrukken van een knop ook het overeenkomstige LED-lampje even oplichten.
- Geef een “get ready”-sein (bv een paar keer alle LEDs laten flikkeren) voor je een volgorde toont
- Geef bij het einde een “eind”-sein (bv bij succes een LEDchaos, bij falen het één voor één uitdoven van de 4 LEDs)

KdG Karel de Grote Hogeschool