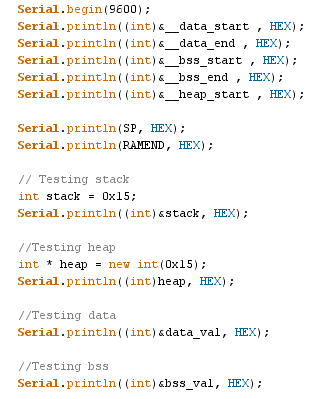
# Memory

Kasper Rijksen

Joery van den Hoff

## Geheugen locaties

Als eerste hebben we een aantal geheugen locaties uitgeprint. Zie hieronder voor code en output.



|  |  |
| --- | --- |
| \_\_data\_start | 0x100 |
| \_\_data\_end | 0x130 |
| \_\_bss\_start | 0x130 |
| \_\_bss\_end | 0x1DC |
| \_\_heap\_start | 0x1DC |
| SP | 0x8F3 |
| RAMEND | 0x8FF |

Bovenstaande waarden zijn allemaal als verwacht (en gespecificeerd over onze arduino).

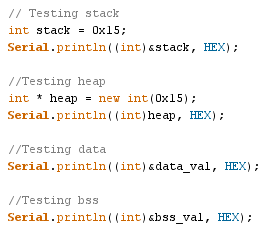
## Variabelen

Daarna hebben we adressen van verschillende variabelen uitgeprint, zie hieronder voor code en output.

Globale variable:

D:\Dropbox\ES31\opdracht6\code2.PNG

In setup:

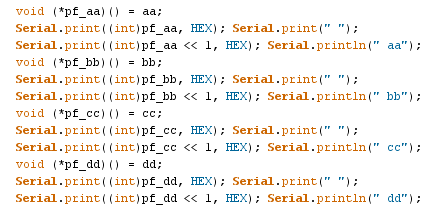


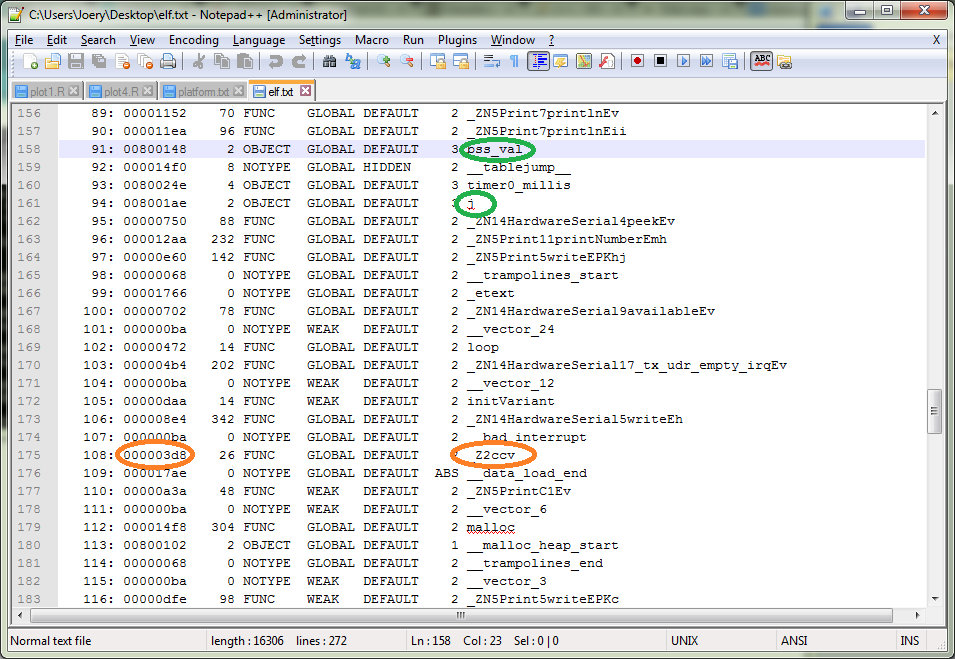
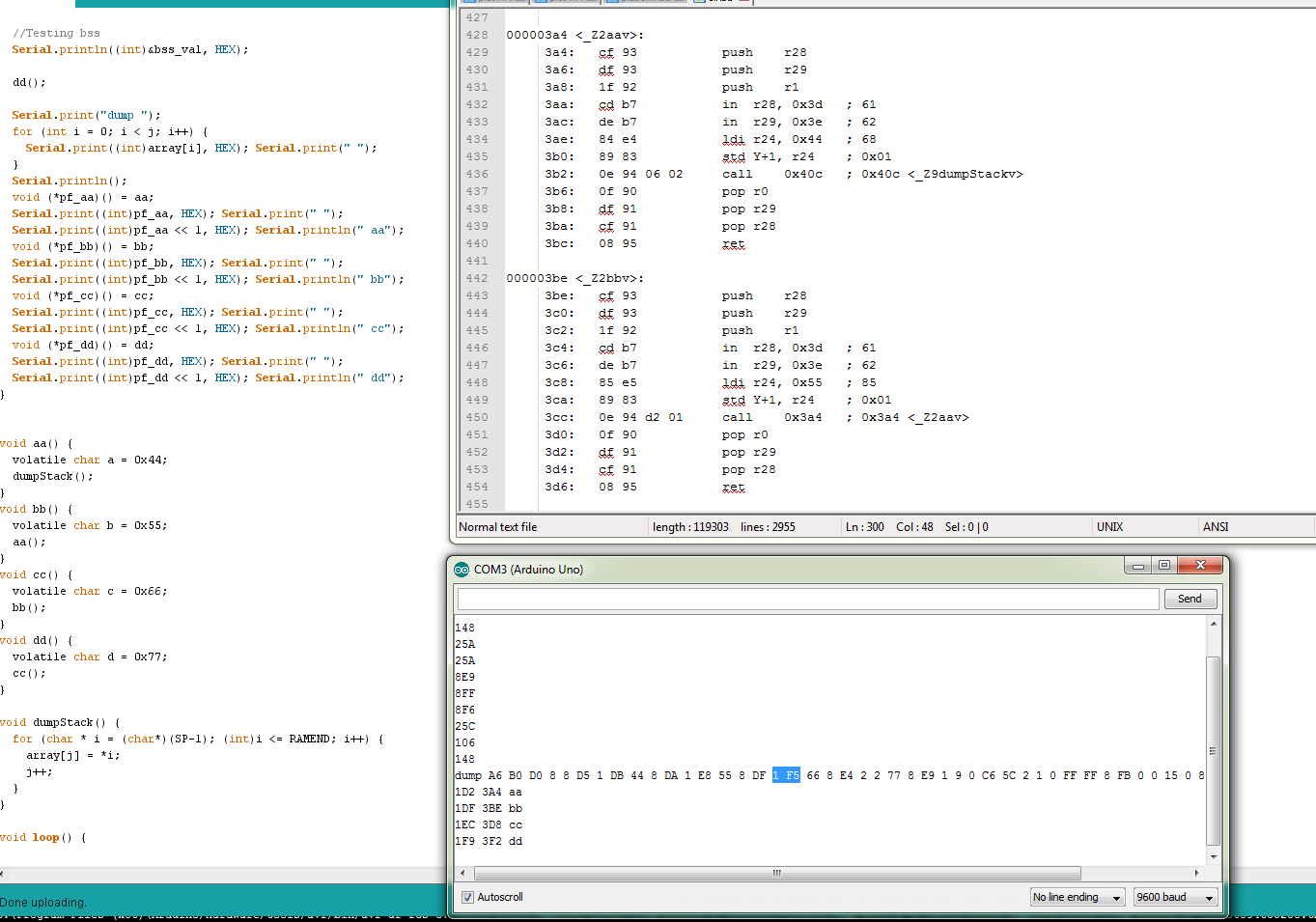
|  |  |
| --- | --- |
| stack | 0x8F6 |
| heap | 0x1DE |
| data\_val | 0x106 |
| bss\_val | 0x130 |

Ook deze waarden komen overeen met wat we verwacht hadden. Als je ze vergelijkt met de vorige waarden kan je zien dat de stack variabele op de stack staat, etc.

## Functies

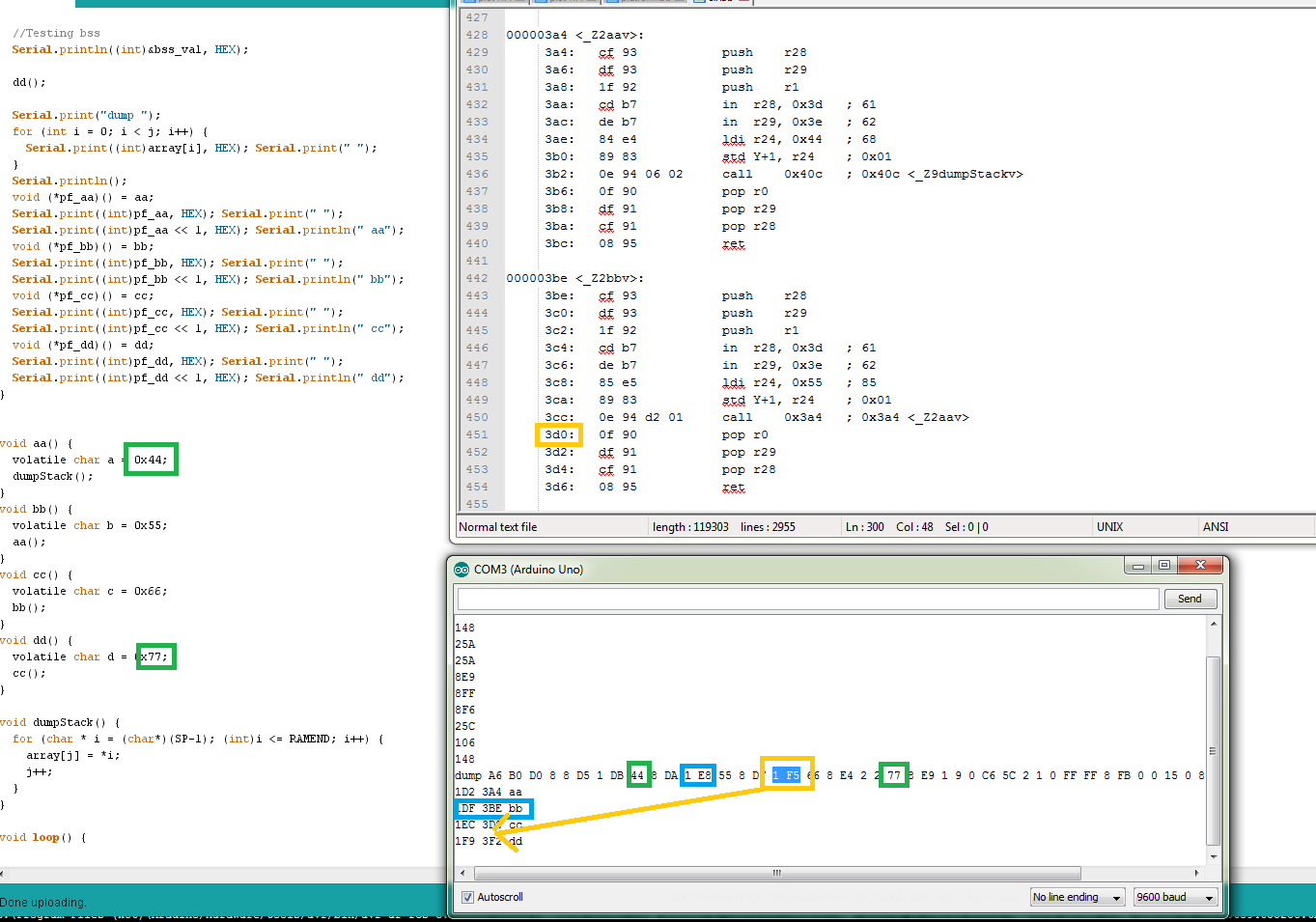
Vervolgens hebben we een aantal functies uitgeprint, zie code en output hieronder.





Zoals je kan zien bij de oranje cirkels, komt het adres dat wij terugkregen in de seriële monitor overeen met wat er in de elf file staat.

## Stack dump

Wij hebben 4 methode gemaakt die elkaar aanroepen. Hierboven hebben we al gezien waar ze in het geheugen staan. Hier zien we de stackdump.

Groen is de lokale variabelen van de methoden. Blauw is de return adres naar methode bb(). Geel is de return adres naar methode cc(), en we kunnen zien in de obj-dump welke instructie dat precies is.

# Bonus

## Stack implementation

Hier is de code van de compiler:

116: 80 e1 ldi r24, 0x10 ; 16

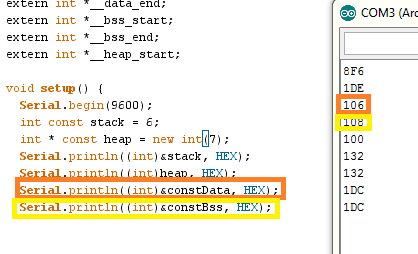
118: 89 83 std Y+1, r24 ; 0x01

Het volgende commando had ook gebruikt kunnen worden:

ldi r24, 0x10

PUSH r24

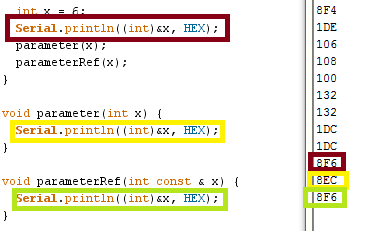
## Const variables



Hier kan je zien dat de variabele die normaal in de Bss staat, nu bij de data staat omdat de variabele geinitialiseerd moet zijn, en dus altijd 0 blijft. Hij kan dus niet bij de ongeinitialiseerde data staan, want const variabelen kunnen niet aangepast worden.

## Passing of parameters

Hier een uitwerking hoe parameters worden doorgegeven, dit is zoals verwacht. Het adres blijft hetzelfde als je by reference meegeeft, maar anders wordt er een kopie gemaakt.



## Architecture

Ik weet niet hoe ik deze vraag kan toelichten met een code voorbeeld, maar de atmega heeft een Harvard architectuur. Harvard architectuur is dat er 2 memory locaties zijn, een voor de programmacode en een voor de data. Onze arduino heeft zelfs 3 memory locaties:

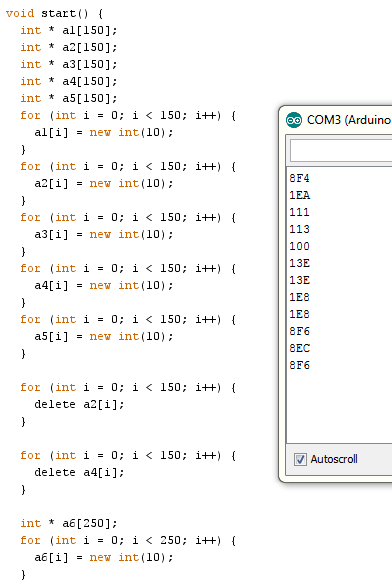
* Flash, hier staat de code op (en dus de start variabelen)
* SRAM, hier is de stack, de heap etc.
* EEPROM, dit is geheugen dat kan worden gebruikt voor permanente opslag omdat de arduino zo weinig SRAM heeft, kan het ook worden gebruikt om ruimte te besparen in de SRAM.

## Memory fragmentation

Ik laat hier zien hoe memory fragmentatie werkt. Eerst proppen we het geheugen helemaal vol. Vervolgens gooien we positie 2 en 4 weg. Je zou denken dat er nu 300 (x2) ruimte is (en dat is ook zo). Maar deze ruimtes zitten niet naast elkaar. Dat betekent dus dat er geen objecten aangemaakt kunnen worden die groter zijn dan 150 (x2).

Zoals je kan zien slaagt het programma wel als ik weer een array van 150 (x2) maak. Maar niet met een array van 250 (x2). Dit werkt precies hetzelfde als met harde schijf fragmentatie.

HHier asd

Hier crasht het programma:

Hier lukt het wel om het programma te starten:

