Verslaglegging

LPA



|  |  |  |
| --- | --- | --- |
| **Assessor** | **:** | **Bakx, René** |
| Naam | : | Thom van Hout |
| Student Nummer | : | 384449 |
| Datum | : | 1-18-2019 |

Inhoud

[Inleiding 1](#_Toc535590941)

[Doelstelling 1](#_Toc535590942)

[Behaalde Resultaten 2](#_Toc535590943)

[Gemaakte Keuzes 2](#_Toc535590944)

[Reflectie 2](#_Toc535590945)

[Dag 1 2](#_Toc535590946)

[Doelstelling voor deze iteratie 2](#_Toc535590947)

[Behaalde resultaten 2](#_Toc535590948)

[Verantwoording gemaakte keuzes 2](#_Toc535590949)

[Reflectie 3](#_Toc535590950)

[Demo 3](#_Toc535590951)

[Feedback 3](#_Toc535590952)

[Dag 2 8](#_Toc535590953)

[Doelstelling voor deze iteratie 8](#_Toc535590954)

[Behaalde resultaten 8](#_Toc535590955)

[Verantwoording gemaakte keuzes 8](#_Toc535590956)

[Reflectie 8](#_Toc535590957)

[Demo 8](#_Toc535590958)

[Feedback 8](#_Toc535590959)

[Dag 3 11](#_Toc535590960)

# Inleiding

Dit project heeft als onderwerp stenografie, dit is vergelijkbaar met cryptografie. Maar in tegenstelling tot de data onleesbaar maken voor iemand zonder een sleutel, is het doel van stenografie het feit verbergen dat er geheime data aanwezig is.

De vorm van stenografie waar dit project om draait is stenografie toegepast op afbeeldingen. Door data te overschrijven op lagere bits van de originele afbeelding, kan data verstopt worden. Deze data kan vervolgens weer uitgelezen worden en worden gescheiden van de verstopte data.

Visueel (voor ons mensen) zal dit niet veel aan de afbeelding veranderen.

De voortgang van het project wordt per dag omschreven, eerst volgen echter algemene punten over dit project.

# Doelstelling

Mijn doel is in dit project is een toepassing maken van stenografie. Om dit doel te bereiken, bestaat dit project ook uit onderzoek doen.

In onderstaande tabel zijn gewenste doelen te zien voor de uiteindelijke applicatie:

|  |  |
| --- | --- |
| **Doel** | **Must have, Should have, Could have** |
| Een afbeelding lezen. | Must have |
| Data encoden in de afbeelding. | Must have |
| Data decoden uit de afbeelding. | Should have |
| Data op andere volgorde encoden. | Could have |
|  |  |
|  |  |
|  |  |

# Dag 1

## Doelstelling voor deze iteratie

1. Welke programmeertaal is geschikt voor afbeelding steganografie?
2. Welk image type is geschikt voor steganografie?
3. Uitzoeken hoe stenografie werkt.

## Proces

Met behulp van de volgende bronnen concludeer ik dat het meest geschikte file format een bitmap is:

* <https://www.ijsr.net/archive/v4i4/29031501.pdf>
* <https://www.sav.sk/journals/uploads/0317153109jo-mo.pdf>
* <https://pdfs.semanticscholar.org/ea32/dbcb2a1027af4c9c0676d08ee6d470aa6a2d.pdf>

Mijn voorkeur ging eerst uit naar PNG, sinds dit een format is dat ik vaak gebruik, vanwege transparantie, fijne bestandsgrootte en wereldwijd grootschalig gebruik.   
Ook brengen veel zoekopdrachten voor toepassingen van stenografie resultaten met gebruik van PNG.

Welk file format gebruiken voor steganografie? => BMP

Flowchart van Hide- en Extract-operatie.

Hoe te controlleren of file in BMP past?

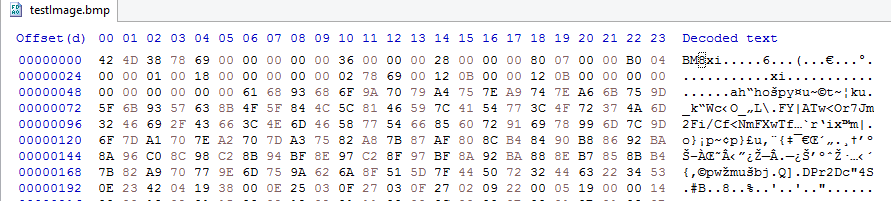
1. BMP opbouw:

//aantekening op papier

**Voor testen is het handig de BMP uit te kunnen lezen. => Hex editor**

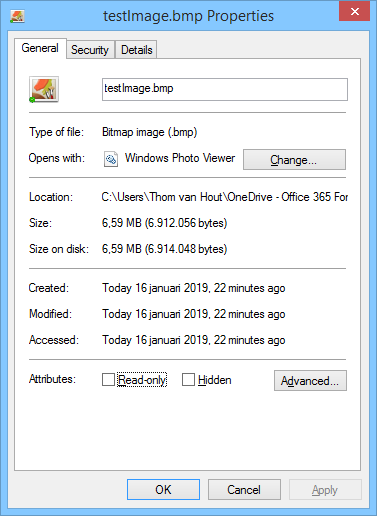
JPG gedownload, convert naar BMP met photoshop.

Als test, hoe groot is de filesize?



Filesize staat volgens <http://www.dragonwins.com/domains/GetTechEd/bmp/bmpfileformat.htm> in bytes nr.2 t/m 5 (vanaf 0).

Volgens Windows (file properties), is de filesize 6.912.048 bytes.



Op bytes 2 t/m 5 staat:

Hier staat: 38 78 69 00, dit is 947.415.296 in dec.

Ook zonder de twee nullen op plek 5 komt er niet het juiste getal uit. Om te kijken of Little Endian, of Big Endian misschien invloedt had, maar de hex waarden omgekeerd levert ook geen juist resultaat op. Omdat ik geen x86 CPU heb, gebruikt mijn pc Little Edian.

96 87 83(16) = 9.865.091 en ook niet de juiste filesize.

Om te controlleren of het niet aan photoshop lag (<https://forums.adobe.com/thread/2220073>), deed ik ook een export van de originele image naar BMP met paint. Ook hier hetzelfde probleem.

Uiteindelijk kwam ik bij de juiste manier de waarde uit te lezen, de LSB van een reeks bytes staat rechts (Little Endian), de individuele byte waardes moeten echter wel van links naar rechts uitgelezen worden.

=> 00 69 78 38 (16) = 6.912.056(10) Dit is correct en komt overeen met “Size” en niet “Size on Disk”.

- MinGW installeerd + toegevoegd aan path. VSCode herkent gcc niet, cmd wel. Nu wel ineens...

- Met stdio.h kan ik nu userinput lezen. Misschien is een losse klasse hiervoor handig.

De file die verstopt gaat worden in de bitmap, moet worden geschreven in de data, niet in de header want dit kan visuele artifacts of corruptie veroorzaken.

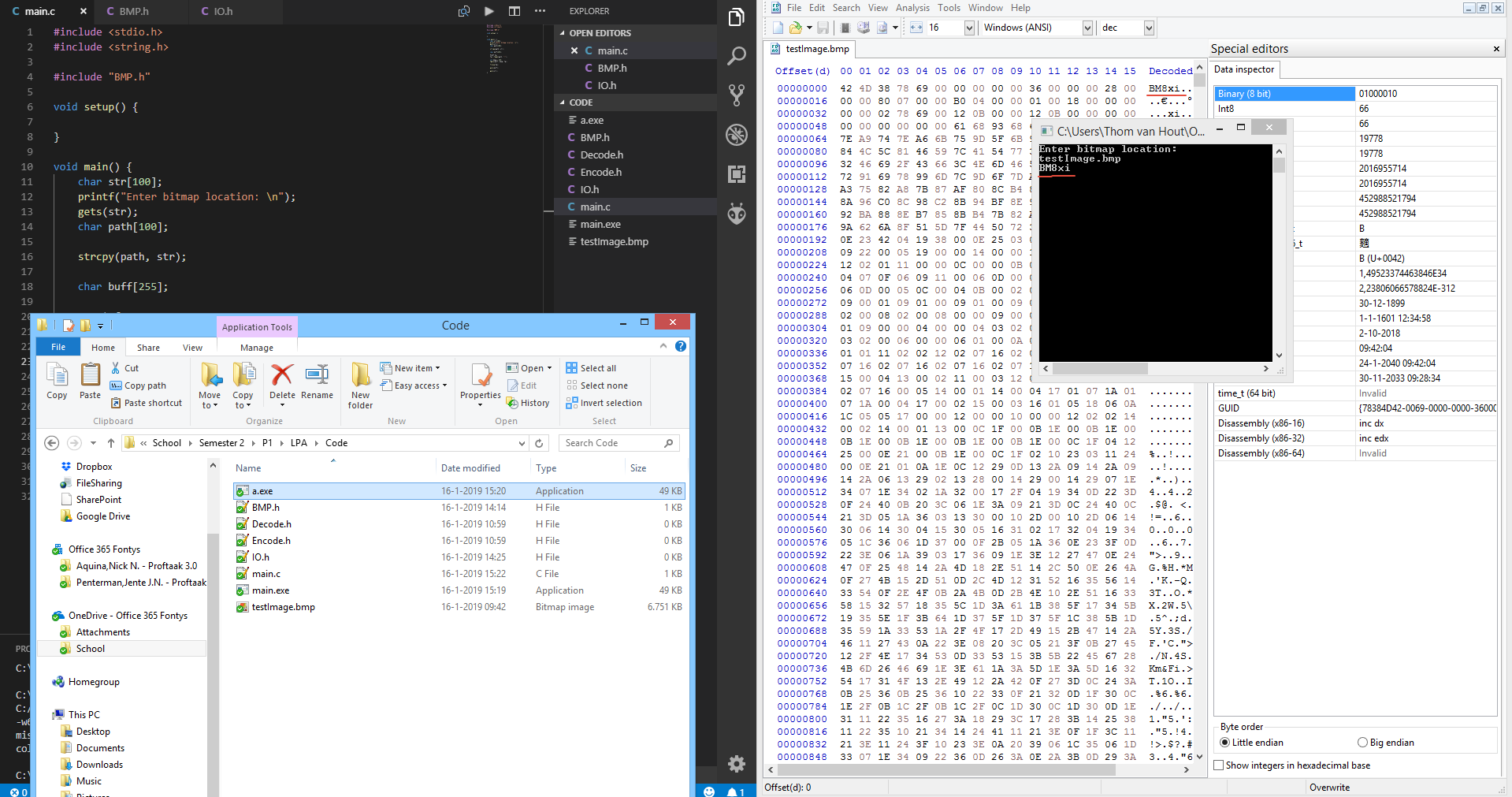
Voordat de file verstopt kan worden, moeten eerst de volgende dingen gebeuren:

* Lees bitmap filesize om zeker te zijn dat het te verbergen bestand past in de pixels.  
  Om aan het aantal bytes voor de kleuren te komen, kan gebruik worden gemaakt van **biWidth** en biHeight uit de ImageHeader. (**biHeight** is negatief wanneer de rijen pixels van boven naar onder gesorteerd zijn.)

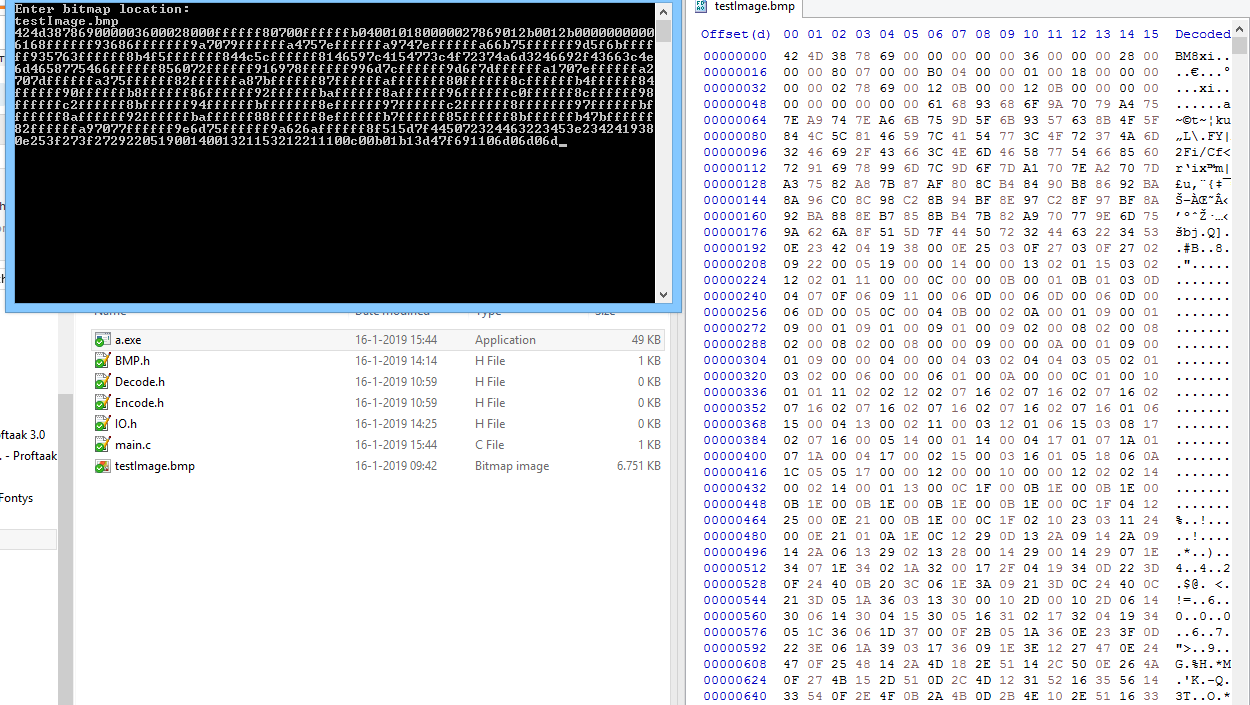
**biSizeImage** geeft ook het aantal pixel bytes, maar wordt bij uncompressed files op 0 gezet.

* Aantal pixels = biHeight \* biWidth

Image grootte in bytes is afhankelijk van hoeveelheid bits per kleur.

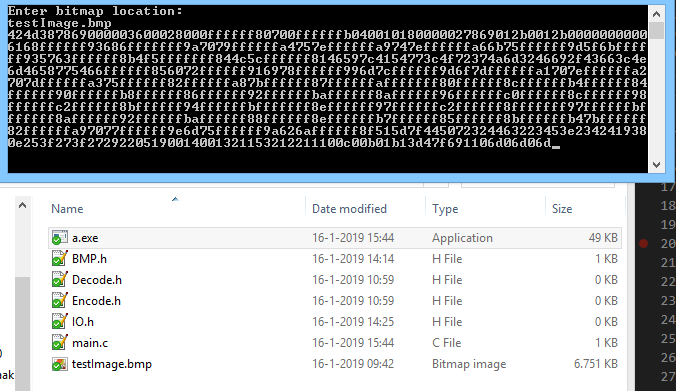


Niet alle data wordt nog weergegeven, wel komt de data overeen.



Door gebruik te maken van **fread** i.p.v. **fgets**, kunnen bytes worden uitgelezen. Deze worden geprint met printf(“%x”, {}) als HEX.

(<https://www.tutorialspoint.com/cprogramming/c_file_io.htm>)



256 Bytes van een ingevoerde bitmap kunnen geprint worden. Nog geen detectie of bestand wel/niet bestaat of afhandeling.

## Behaalde resultaten

1. Bekend welk image format het meest toepasselijk is.
2. Eerste opzet code:
   1. Console input en feedback.
   2. Lezen van een bestand en de data opslaan in een array (nu nog gelimiteerd tot 256 bytes).
   3. De 256 bytes printen als HEX waarden.
   4. Structs aangemaakt die alle header informatie van de bitmap kunnen opslaan. Informatie hieruit is vitaal om te kunnen encoden.
3. Hex viewer/editor geïnstalleerd (HxD).
4. MinGW geïnstalleerd en geconfigureerd om te kunnen compilen met VSCode.
5. Opbouw bitmap file onderzocht.

## Verantwoording gemaakte keuzes

1. Gekozen voor een bitmap file:

Compressie is niet vereist bij het schrijven van een bitmap file. Hierdoor is het mogelijk veel data te verbergen (grote filesize) en gaat verborgen data niet verloren tijdens compressie.

1. Taal:

Als programmeertaal kies ik voor C, mijn comfort ligt bij C#. Daarbuiten is het een lage taal, dit is toepasselijk voor de bitmanipulatie die gedaan wordt.

Ook geef ik voorkeur aan een command line toepassing boven een met een UI.

Als laatst is C sneller, bij gebruik van grote bestanden is dit verschil wellicht te merken.

* + - Feedback René -> Eerst tekst encoden, daarna grooter.

## Reflectie

Vandaag heb ik veel tijd besteed aan onderzoek. Dit heeft me een beter beeld gegeven over hoe de applicatie er uit gaat zien.

Als eerste zal ik gaan focussen op een message geëncode krijgen. Een ander bestand zou geen groot verschil moeten zijn, sinds het allebei bytes zijn. Het kunnen encoden en decoden van een bestand is wat ik wil bereiken.

Op basis van wat ik vandaag geleerd heb, kwamen deze punten voor de richting van het project in me op:

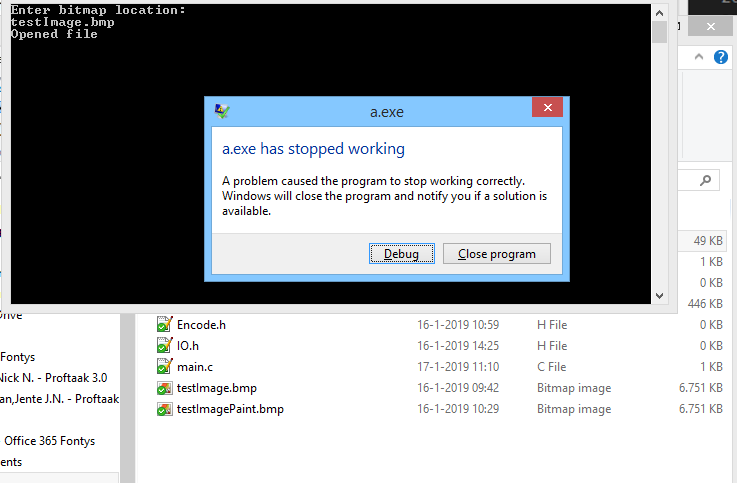
* Een Color Palette table is soms aanwezig, is dit iets waar rekening mee moet worden gehouden?
* De bitmap die ik als test gemaakt heb is 24-bit, heeft het meerwaarde om voor alle mogelijkheden te kunnen encoden/decoden? Dit veranderd namelijk de plekken waar data geschreven kan worden en de hoeveelheid data die encoded kan worden.
* Hoeveel bits kunnen overschreden worden voordat het zichtbaar wordt?
* Ga ik de mogelijkheid implementeren de afbeelding waarin de data verstopt moet worden te groeien, als de gewenste encode data te groot is?

# Dag 2

## Doelstelling voor deze iteratie

Het is nodig meer uit te kunnen lezen dan 256 bytes, ik verwacht dat dit opgelost kan worden door bytes 1 voor een uit te lezen, of door het gehele bestand te kopieëren naar het geheugen. In het laatste geval zal dan ook moeten worden afgehandeld wanneer het geheugen vol is.

Door de filepointer te checken op NULL, kan worden voorkomen dat een niet bestaand bestand wordt geprobeerd uitgelezen te worden. Dit veroorzaakt nu echter een crash bij het openen van mijn originele test image en niet bij de andere.  
fgets Werkt wel nog op dit bestand gek genoeg.



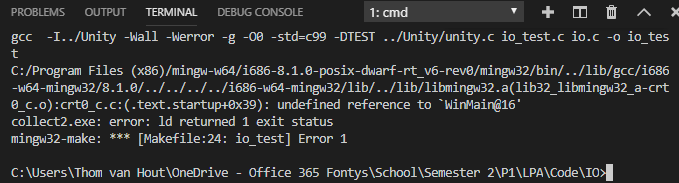
De code crashed vlak tijdens de fclose(). Volgens <https://stackoverflow.com/questions/17449917/fclose-causing-a-crash-in-windows> ligt dit aan een andere fout in de code en niet fclose.

De fout zat in de buffer waarin de data werd opgeslagen, hier was geen plaats voor de extra escape char die wordt geplaatst op het eind na het uitlezen. Het probleem is echter nog niet opgelost.

fread was ook niet in orde, ik paste elementSize aan i.p.v. elementCount.

De fout zat uiteindelijk in het 2e argument van fread: elementSize. Hier gaf ik sizeof(buffer), hierdoor werd veel te veel data uitgelezen. Het moest namelijk sizeof(char) zijn. Deze website heeft me geholpen dit te realiseren. <https://www.codingunit.com/c-tutorial-binary-file-io>

Er kunnen veel dingen over het hoofd gaan, daarom vind ik unit tests maken belangrijk. Helaas krijg ik make van mingw niet aan de praat. In verband met tijd ga ik voor nu verder zonder unit tests.



Eerste implementatie Unittests voor BMP en IO klasses.

Begonnen aan parsen header van BMP file. 

* + Doelstelling voor deze iteratie

Feedback René:

* + Eerste stap is 0 bits encoden i.p.v. data, vervolgens door naar volgende stap (messages encoden).
  + Maak de verslaglegging in een vloeiend lopend verhaal.
  + Behaalde resultaten
* File IO crashes gefixed, ook was er nog geen afhandeling van niet bestaande bestanden.
* Unit test integratie opgezet en unit tests gemaakt.
* Makefiles gemaakt voor de klasses IO en BMP.
* Struct initialisatie en 1e dataparsing.
  + Verantwoording van de gemaakte keuzes

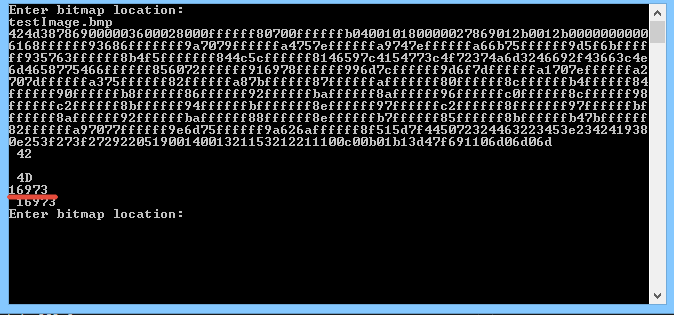
1. Voor nu focus ik me op 24-bit kleur.
2. Keuzes als klassen, unit tests en makefiles maken, zijn gekozen voor de goede structuur die ze geven. Ik verwacht dat dit uiteindelijk tijd zal besparen door de overzichtelijkheid en controlleerbaarheid.
   * Reflectie
3. Van de ene kant vind ik mijn tijd besteed aan unit testen opzetten nuttig, maar aan de andere kant zou ik liever meer tijd willen besteden aan features. Voorheen wist ik niet hoe Unit tests opzetten moest dus ik heb er van geleerd.  
   Vandaag heb ik ontzettend vaak “testImage.bmp” uitgetypt, unit tests geven me de mogelijkheid zulke situtaties te voorkomen.
4. Net als in de proftaak met het onderzoek heb ik vandaag ook het probleem, dat browser management chaotisch is. Het project heeft verscheidene taken en vaak is het daarom nuttig een tabblad, waar bruikbare informatie gevonden is, open te laten. Maar veel tabbladen openhouden is chaotisch en gaat tenkoste van tijd.

Een oplossing is om beter mijn vondsten te documenteren tijdens het werken, zodat ik op één plek snel bronnen terug kan vinden zonder de gehele bron opnieuw te doorzoeken.

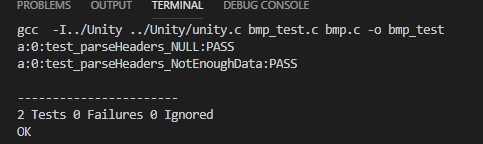




1. Vandaag had ik graag meer afgekregen, veel tijd is helaas verloren gegaan aan bugs en onderzoek doen.
   * Indien van toepassing een demonstratie   
     van geproduceerde artefacten.



2 Bytes uit de header worden opgeslagen in- en uitgelezen van een struct. Dit is de eerste keer dat bitmanipulatie toegepast is, om 2 bytes te combineren tot een ushort.



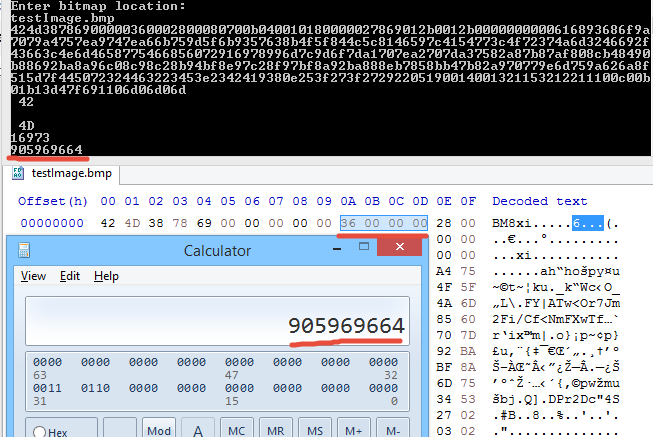
2 Tests voor BMP klasse af.

  
Afhandeling van niet bestaande bestanden.

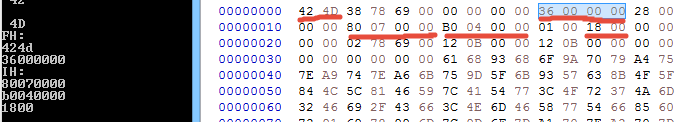
  
Klassen hebben hun eigen mapje, wist voorheen niet hoe je kon includen uit een ander directory.

# Dag 3

In plaats van de data in een char array opslaan, lijkt me een uint8\_t meer gepast. Ookal zal dit geen verschil uitmaken.



De offset uit de header wordt goed geparsed. Ik heb besloten alleen de data te gaan parsen die vereist is om tijd te besparen.

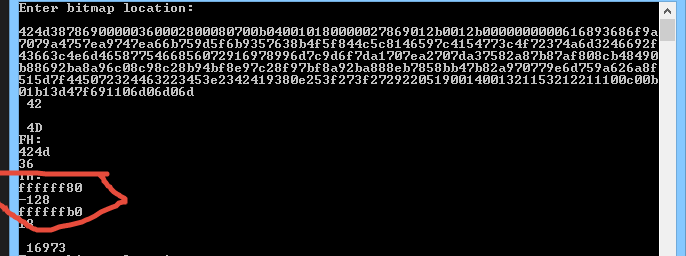


Essentiele data uit de fileheader en imageheader kan geparsed worden.

Een probleem is dat deze data little endian is, dit maakt uit omdat als de getallen zo worden uitgelezen de values niet kloppen. Dit betekent dat in code de values in de struct terug geconvert zouden moeten worden. De offset van 36.000.000(16) zou 54(10) moeten zijn, geen 905969664(10).

Omdat er alleen gelezen wordt uit de header, is het handig de data meteen goed op te slaan want er moet mee gerekend worden en dit voorkomt dat het later steeds gedaan moet worden.

Nu de data als big-edian geparsed wordt, kloppen de width en height gek genoeg niet meer.



De waarde zetten op precies dezelfde manier in main werkt wel. Zelfde includes als main en andersom fixt het probleem ook niet, net als het eerst initialiseren van de structs.

Om te weten waar data encoded gaat worden, zijn de start en het eind van de data nodig.  
De start is uit te lezen uit offset. Het eind is te berekenen met: bitCount/8 \* width \* height + start.  
Hiermee wordt het aantal bytes nodig voor de data berekend en opgeteld bij de start van de data.

De laatste 2 bytes van de image zijn 0, wellicht is het belangrijk deze ook te schrijven. Waarschijnlijk zijn deze er omdat een meervoud van 4 vereist is.

54/4 is niet een geheel getal.

56/4 is dit wel.

Het is nodig omdat elke kleur (RGB) één byte heeft (vandaar 24-bit color), op het eind van deze data worden 0 bytes geschreven om te zorgen dat het aantal bytes een veelvoud van 4 is.

Na een aantal fouten in de argumenten van de read en write functies te hebben opgelost, werkt 0 bits encoden op de LSB. De code zit nog niet in een eigen klasse, dit ga ik nu doen. Wel sla ik unit tests voor nu over, sinds ik de tijd beter kan gebruiken.  
Het process om de file te schrijven duurt lang, dit komt denk ik omdat ik byte-voor-byte uitlees en schrijf.

Voor de encode functie zit ik te twijfelen tussen een filepointers meegeven, of paths. Ik ga voor paths zodat de encode klasse controle heeft over het openen en sluiten van de filestreams.

* + Doelstelling voor deze iteratie

1. 0 Bytes kunnen encoden.
2. Documentatie op orde stellen.
3. Message encoden.
4. Message decoden.