Aan:	Alle HoGent studenten pbati[1] Tin die het olod[2] IT Fundamentals volgen in semester 1 van AJ 2020 - '21
Van:	Lectoren IT Fundamentals
Datum:	do 10 december 2020
Ref.:	209lmBF1j.doc:hpz
Betreft:	**Belangrijk! Correcte informatie over CRC (Cyclic Redundancy Check) berekening als onderdeel van hoofdstuk 8 Codes en zoals te kennen voor het examen.**

Lees (ook) de overeenkomstige .pdf bij dit bericht, die is doorgaans vollediger.

Beste student,

De enige correcte werking van het CRC algoritme werd voor de allereerste keer in 1961 beschreven: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.680.3771&rep=rep1&type=pdf .
Een Nederlandstalige correcte uitleg kan je lezen in het begeleidend boek 'Wiskunde voor IT' vanaf pagina 379, maar de Nederlandstalige en de Engelstalige wikipediasite leggen ook de enige juiste werking van CRC uit.
Opgelet, in de slides van IT Fun zoals via Chamilo downloadable, staat in hoofdstuk 8, genaamd 'Codes' een foute werkwijze! Ook de on line CRC-berekening in de begeleidende website https://itfundamentals.azurewebsites.net/CRC is fout! En ook de oplossingen van de oefeningen in de slides van hoofdstuk 8 zijn fout!

Over de correcte werking zoals te kennen voor het komende examen vind je illustrerende informatie in volgende youtube filmpjes van playlist IT FUN van het kanaal van Andre Ooit:
IT-FUN 11 (CRC (Cyclic Redundancy Check) By Example - Oef. 1 van sl. 19 Part 1 - Sender)
https://www.youtube.com/watch?v=97ozedZVR4w&list=PLuhWuD-eOibinLbklK6ALPfK49BaB2KD9&index=11
IT-FUN 12 (CRC (Cyclic Redundancy Check) By Example - Oef. 1 sl. 19 - Part 2 - Receiver)
https://www.youtube.com/watch?v=SumUi623pIQ&list=PLuhWuD-eOibinLbklK6ALPfK49BaB2KD9&index=12
IT-FUN 23 (CRC (Cyclic Redundancy Check) Example 2 - Part 1 - Sender)
https://www.youtube.com/watch?v=moGk7gDh-30&list=PLuhWuD-eOibinLbklK6ALPfK49BaB2KD9&index=23
IT-FUN 24 (CRC (Cyclic Redundancy Check) Example 2 - Part 2 - Receiver)
https://www.youtube.com/watch?v=bgx9afsEJ7A
[ITFUN - 025] - Codes - CRC (Cyclic Redundancy Check) Example 2.1 p. 19 (Sender & Receiver)
https://www.youtube.com/watch?v=FNNBNpGSyZs&list=PLuhWuD-eOibinLbklK6ALPfK49BaB2KD9&index=25
en in dui zen den andere youtube filmpjes en op nog meer websites.

[1] Professionele BAchelor Toegepaste Informatica
[2] olod: OpLeidingsOnDerdeel

Illustratief correct voorbeeld:

Het toevoegen van het aantal nullen aan het bericht is essentieel maar die nullen voegen niets toe aan het bericht zelf, ze zijn redundant.
Het geheel wordt dan op de gekende binaire manier gedeeld. De uiteindelijk bekomen rest komt in de plaats van de nullen en zó verzonden.
De ontvanger doet precies dezelfde deling als de verzender heeft gedaan en komt uit op het oorspronkelijk bericht met al de overbodige redundante nullen er weer achter en de cirkel is rond: cyclic. En dat kan je zo blijven doen, ten minste, dat controleer je: check: cyclic redundancy check

bericht: 1101 / deler (polynoom): 111 / uitgebreid bericht: 1101 00 (het aantal toegevoegde, overbodige nullen is steeds één minder dan het aantal bits van de polynoom)

```
pol.    uitgebreid bericht van de verzender
111     1101 00 (= begin)
         111
         0011
          000
          011 0
           11 1
           00 10
            0 00
            0 10
```

```
pol.    uitgebreid bericht van de ontvanger
111     1101 10
         111
         0011
          000
          011 1
           11 1
           00 00
            0 00
            0 00 (<- dit aanhechten aan het oorspr. bericht en terug bij het begin, ten minste dat check je)
```

De Cyclic Redundantie Check werkt als volgt:

1) verzender A wil een bericht (ook message genoemd) M in bitformaat naar ontvanger B sturen.

Beiden spreken een gezamenlijke zogenaamde polynoom P af (ook deler of divisor genoemd) die moet 'helpen' om te controleren of B het bericht correct ontvangen heeft.

P is L bits lang (L is ten minste 2 en zeker msb= 1)

2) Aan het oorspronkelijk te verzenden binair bericht voegt A (L - 1) nulbits toe, zogenaamde **REDUNDANTe** nullen. Ze verhogen de complexiteit van het oorspronkelijk bericht niet, ze zijn met andere woorden 'overbodig' (= redundant) máàr ze laten wél toe om de juistheid nà verzending door A en ontvangst door B van M te **CHECKen**, te controleren (zie verder)

3) A breidt het oorspronkelijk bericht M met (L - 1) nulbits uit = Message with added zero's, genoteerd MWAZ.

4) A voert met P een x-or deling uit op MWAZ.
De dàn overblijvende (L - 1) bits vormen de zogenaamde CRC, of CRC code, of CRC woord. Merk op: alle bits van het oorspronkelijk bericht M zijn na x-or deling 'weggedeeld'.

5) A vervangt de (redundante) oorspronkelijk toegevoegde nulbits door de bits van de CRC code en vormt zó het uitgebreid te verzenden bericht (= extended message to send) genoteerd EMTS.
A ***zendt*** het naar B.

6) Als alles goed gaat, ***ontvangt*** B het uitgebreide bericht (= extended message received) genoteerd EMR dat exact gelijk moet zijn aan het EMTS.

7) B controleert dit door met polynoom P de x-or deling uit te voeren op het uitgebreid ontvangen bericht EMR.
Indien op het einde van de uitgevoerde x-or deling allemaal nullen verschijnen, is het ganse proces van verzenden en ontvangen foutlloos verlopen. B vervangt de laatste (L - 1) bits van het EMR door de nullen van de x-or deling en het oorspronkelijk te verzenden bericht aangevuld met de redundante nullen verschijnt weer, dit is het MWAZ en 'de cirkel is rond' (= **CYCLIC**) Het bericht werd correct ontvangen (*) .

8) B haalt de (L - 1) nulbits van het MWAZ en verkrijgt de pure, nuttige gegevens zijnde het bericht M.

9) Indien de laatste (L - 1) bits niet alle nul zijn, is er één, of zijn er meerdere fouten. In regel kan de CRC code deze niet exact bepalen en opnieuw laten verzenden is aangewezen.

(*) volledigere: er zitten geen bitfouten in het bericht detecteerbaar door de gebruikte polynoom P

Met vriendelijke groet, lectoren IT FUN, HoGent, DIT, pbati

———————

A real computer doesn't have no eyes. - © A. Ooit