Introductie tot de cursus

Introductie tot de cursus

In deze introductie willen wij u informeren over de bedoeling van de cursus, de opzet van het cursusmateriaal en over de manier waarop u de cursus kunt bestuderen. U vindt in deze introductie dus nog geen echte studiestof, maar praktische en studietechnische informatie die u inzicht geeft in de aard en opzet van de cursus en die u kan helpen bij het studeren. Ook treft u informatie aan over de bij de cursus gebruikte programmatuur.

1 De functie van de cursus

1.1 MOTIVERING VAN DE CURSUS

In de onderbouw van een informaticacurriculum zal altijd plaats zijn ingeruimd voor één of meer programmeercursussen. In deze cursussen staat in het algemeen het leren oplossen van problemen centraal, waarbij men gebruikmaakt van een gekozen programmeertaal, waarmee de oplossingen kunnen worden geïmplementeerd. De keuze voor een eerste programmeertaal is voor een informaticaopleiding niet eenvoudig, gezien het enorm grote aantal programmeertalen dat op dit moment in gebruik is. Elk van deze talen heeft zo zijn eigen voor- en nadelen op uiteenlopende aspecten als uitdrukkingskracht, gemak om de taal te leren, vraag van de arbeidsmarkt, beschikbaarheid en prijs van implementaties op verschillende platforms, betrouwbaarheid, efficiëntie, gemak om uitgaande van kennis van een taal ook andere programmeertalen te leren en theoretisch fundament van de taal.
In het verleden is bij de meeste opleidingen de keuze gevallen op een zogenaamde *imperatieve programmeertaal* als eerste kennismaking met programmeren. Hierbij waren – in de jaren tachtig van de vorige eeuw – vooral talen als Pascal, C en Modula-2 gebruikelijke keuzes, aangezien deze talen allemaal hoog scoren op de meeste van de genoemde aspecten.
Een relatief klein deel van de informaticaopleidingen koos in deze jaren voor een geheel andere introductie tot programmeren. Bij opleidingen waarbinnen kunstmatige intelligentie (AI: artificial intelligence) een belangrijke rol speelt, wil nog wel eens de keuze vallen op Lisp (een klassieke *functionele programmeertaal*) of Prolog (een *logische programmeertaal*) als eerste programmeertaal, omdat deze talen geschikter blijken te zijn om juist AI-problemen mee te beschrijven en op te lossen.
Andere opleidingen kozen ervoor om *naast* (of na) de imperatief gerichte programmeercursussen de studenten ook kennis te laten maken met het programmeren in een niet-imperatieve taal als Lisp of Prolog.
Ook om andere redenen – zoals de toenemende nadruk op het op formele wijze redeneren over programma's – is vanaf de tweede helft van de jaren tachtig bij een aantal academische informaticaopleidingen de overstap gemaakt naar een functionele programmeertaal voor een eerste kennismaking met programmeren. In deze gevallen werd meestal

gekozen voor moderne functionele talen als ML, Scheme (een moderne versie van Lisp), Miranda of – in het begin van de jaren negentig – voor Haskell. In het curriculum van de faculteit Informatica is een aparte cursus gewijd aan functioneel programmeren.

Naast de toenemende aandacht voor de logische en functionele programmeerstijl – men spreekt ook wel van programmeerparadigma's – zijn ook twee andere belangrijke ontwikkelingen van invloed geweest op de curricula op het gebied van programmeren.

De eerste ontwikkeling is het ontstaan van de *objectgeoriënteerde programmeertalen*. Hier is niet echt sprake van een fundamenteel ander programmeerparadigma, maar meer van een verdere ondersteuning van de ideeën over hergebruik van software-onderdelen en flexibiliteit bij het aanpassen van software, die al was ingezet bij de ontwikkeling van imperatieve talen als Ada en Modula-2. Belangrijke objectgeoriënteerde programmeertalen zijn Smalltalk, C++ (een uitbreiding van de imperatieve taal C), Eiffel en sinds 1995 de taal Java. Java is een taal die lijkt op C++, maar wel beter is: eenvoudiger om te leren, betrouwbaar, klein (de taal is gemaakt om op kleine computers te kunnen draaien) en zo goed als zuiver objectgeoriënteerd. Om deze redenen heeft de faculteit Informatica in 1997 gekozen voor Java als centrale programmeertaal in het informaticacurriculum. Dit heeft geleid tot de ontwikkeling van een aantal cursussen en een integrerend practicum op het gebied van programmeren met Java en op het gebied van algoritmen en datastructuren.

Een tweede ontwikkeling heeft te maken met het beschikbaar komen van computers met meer processoren, waardoor berekeningen parallel kunnen worden uitgevoerd. Om deze parallelle systemen te kunnen programmeren, dienen we te beschikken over aparte taalconcepten. Veel van de ontwikkelde *parallelle programmeertalen* zijn varianten van populaire imperatieve programmeertalen waaraan een aantal voor het parallellisme noodzakelijke concepten zijn toegevoegd, bijvoorbeeld Concurrent Pascal en Concurrent C. De imperatieve taal Ada kent een aantal parallelle concepten. Steeds meer objectgeoriënteerde talen zoals Java en Scala nemen standaardconstructies op om parallelle processen ('threads') te creëren en te laten samenwerken. De taal Occam is een voorbeeld van een taal die speciaal is ontworpen voor het programmeren van parallelle toepassingen.

Welke keuze een informaticaopleiding ook maakt voor de talen waarmee studenten leren programmeren, in het vervolg van de opleiding zal toch ook aandacht besteed moeten worden aan de andere programmeerstijlen en zal een student een bredere kennis en een dieper inzicht moeten krijgen in de geschetste ontwikkelingen op het gebied van programmeertalen. Vandaar dat naast programmeercursussen ook vrijwel altijd aandacht besteed wordt aan programmeertaalconcepten. Hierbij geeft men expliciet een overzicht van de verschillende programmeerparadigma's, hun essentiële kenmerken en hun onderlinge verschillen. De inhoud en aanpak van een cursus over programmeertaalconcepten en -paradigma's zal uiteraard sterk afhankelijk zijn van de gemaakte keuzes bij eerdere programmeercursussen.

In deze cursus is de volgende doelstelling centraal gesteld: het geven van een overzicht van en inzicht in programmeertaalconcepten, waarbij we proberen te tonen welke concepten karakteristiek zijn voor de verschillende programmeerstijlen en welke concepten gemeenschappelijk zijn.

Aspecten als implementatie en ontwerp van programmeertalen hebben veel minder een rol gespeeld bij de doelstellingen voor deze cursus.

Met deze cursus geven we een afronding van het programmeeronderwijs van onze bacheloropleiding. U zult na het bestuderen van deze cursus een breed overzicht hebben van de huidige stand van zaken en goed voorbereid zijn om ontwikkelingen op het gebied van programmeertalen te kunnen volgen en beoordelen. Het is duidelijk dat de universele programmeertaal niet bestaat en er waarschijnlijk ook nooit zal komen. Wel zien we dat objectgeoriënteerde taalconcepten opgenomen worden in veel van de bestaande programmeertalen en andere softwaregereedschappen. Ook concepten uit de functionele en logische talen zullen naar alle waarschijnlijkheid hun plaats verwerven in de programmeertalen van de komende decennia, maar ook in specificatietalen, in vraagtalen, enzovoort. Bovendien is er een trend naar integratie van functionele en objectgeoriënteerde talen.
Niet alleen voor de software-specialist is kennis van de verschillende taalconcepten en -paradigma's een must, ook voor de hardware-specialist zal vanwege de toenemende convergentie tussen hardware en software – denk aan de ontwikkelingen op het gebied van ontwerptalen voor hardware – deze cursus van belang kunnen zijn.

1.2 PLAATS VAN DE CURSUS

De cursus is een verplicht onderdeel van de bacheloropleiding Informatica. De cursus is ook geschikt als losse cursus voor personen die behoefte hebben aan een op zichzelf staande cursus over programmeertaalconcepten.

2 De inhoud van de cursus

2.1 LEERDOELEN

Na het bestuderen van deze cursus wordt verwacht dat u
– een goed inzicht hebt in de basisconcepten van programmeertalen, zoals waarden, typen, expressies, variabelen, opdrachten, bindingen en abstractiemechanismen
– kunt aangeven in hoeverre een concrete programmeertaal voldoet aan de vier in deze cursus geformuleerde kwaliteitscriteria voor programmeertalen: het type-volledigheidsprincipe, het kwalificatieprincipe, het abstractieprincipe en het correspondentieprincipe
– een goed inzicht hebt in inkapselingstechnieken, typesystemen en manieren om de programmaverwerking te onderbreken
– een goed inzicht hebt in de concepten van parallel en gedistribeerd programmeren
– met eigen woorden kunt beschrijven welke taalconcepten kenmerkend zijn voor respectievelijk de imperatieve, de objectgeoriënteerde, de functionele, de parallelle, de logische en de scripting programmeertalen.

2.2 VOORKENNIS

Om de cursus met succes te kunnen volgen, dient u over een goede vaardigheid te beschikken in het programmeren in een objectgeoriënteerde programmeertaal (bijvoorbeeld Java) en in een functionele taal (bijvoorbeeld Haskell) op het niveau van de cursussen *Objectgeoriënteerd programmeren in Java 2* en *Functioneel programmeren*. Verder gaan we in de cursus uit van een vertrouwdheid met elementaire gegevensstructuren als lijsten en bomen en een vaardigheid in recursief programmeren, zodat ook de cursus *Datastructuren en algoritmen* als voorkennis sterk wordt aanbevolen. Hoewel niet per se noodzakelijk, is het met name voor blok 3 (Parallel programmeren) aan te raden dat u vooraf ook de cursus *Besturingssystemen* hebt gevolgd.
Door het abstractieniveau van de cursus verwachten we verder dat u beschikt over voldoende wiskundige scholing. Kennis op het niveau van de cursus *Discrete wiskunde A* is daarbij een minimumeis.
Omdat het tekstboek in het Engels is geschreven, is een goede leesvaardigheid van de Engelse taal een vereiste.

Voor wie onvoldoende kennis heeft van Haskell staat op de cursussite een aantal leereenheden met een introductie op Haskell uit een voorgaande versie van de cursus.

2.3 PROGRAMMEERTALEN

De cursus is voor een belangrijk deel gebaseerd op het tekstboek *Programming Language Design Concepts* van David A. Watt. De inhoud van dit boek komt goed tegemoet aan de doelstelling van de cursus: een benadering vanuit de taalconcepten.

Regelmatig worden nieuwe programmeertalen ontwikkeld. Daarnaast evolueren bestaande programmeertalen snel. Het tekstboek representeert een momentopname in de ontwikkeling van programmeertalen. Wat in het tekstboek staat met betrekking tot de voorzieningen in een bepaalde programmeertaal kan nu al achterhaald zijn, maar dat is voor het bestuderen van concepten niet belangrijk. In het bijbehorende werkboek hebben we modernere onderwerpen aan de orde gebracht. Maar ook voor het werkboek geldt dat het een momentopname is. Concepten en paradigma's zijn in de cursus belangrijk, niet de opsomming van de eigenschappen van de besproken programmeertalen.

In het voorwoord van het tekstboek schrijft de auteur dat u het meeste voordeel uit het bestuderen van het tekstboek haalt als u al vooraf vertrouwd bent met ten minste twee verschillende typen programmeertalen. In het cursusboek gaan wij ervan uit dat u een goede beheersing van de objectgeoriënteerde taal Java en van de functionele taal Haskell heeft. Omdat een objectgeoriënteerde taal als Java voortbouwt op concepten uit de imperatieve talen, zal het met alleen voorkennis van Java weinig moeite kosten de voorbeelden in C, C++ of Ada te begrijpen. Bovendien zijn bij elk voorbeeld in het tekstboek in de cursusboeken corresponderende voorbeelden in Java opgenomen.

In de cursus en bij het tentamen wordt ervan uitgegaan dat u een actieve kennis van de talen Java en Haskell heeft. Het betekent dat u gegeven code kunt begrijpen en desgevraagd codefragmenten kunt opstellen. Bij het schriftelijke tentamen zijn syntactische fouten in programmacode niet relevant.
Van de andere programmeertalen wordt slechts passieve kennis verwacht. Hiermee wordt bedoeld dat u gegeven code moet kunnen interpreteren, niet dat u zelf code hoeft op te stellen.

2.4 OPBOUW VAN DE CURSUS

De cursus is opgebouwd uit vier blokken:
1 Basisconcepten
2 Geavanceerde concepten
3 Parallel programmeren
4 Programmeerparadigma's.

Van elk van deze blokken geven we een korte omschrijving van de inhoud en de omvang.

Blok 1: Basisconcepten

Inhoud

Dit blok bestaat uit vier leereenheden die zijn gebaseerd op de hoofdstukken 2 t/m 5 van het tekstboek. In dit eerste deel van het boek worden de basisconcepten behandeld, zoals die in vrijwel alle programmeertalen wel op een of andere manier voorkomen: waarden en typen (hoofdstuk 2), variabelen en geheugen (hoofdstuk 3), bindingen en scope (hoofdstuk 4) en abstractie en parametrisering (hoofdstuk 5).
De nadruk ligt in deze hoofdstukken op het afzonderlijk beschrijven en bestuderen van deze basisconcepten, dat wil zeggen: zoveel mogelijk los van de context van een concrete programmeertaal. Wel passen we de concepten steeds toe aan de hand van concrete talen. Bij elk van deze vier hoofdstukken is een werkboekeenheid ontwikkeld. Samen met de hoofdstukken uit het tekstboek vormen zij de leereenheden 1 t/m 4.
Op de functie van de werkboekeenheden gaan we in paragraaf 3 nader in.

Omvang

Blok 1 bestaat uit vier leereenheden en wordt afgesloten met een samenvatting over de leerstof van het gehele blok.
De totale studielast van dit blok schatten we op 43 uur.

Blok 2: Geavanceerde concepten

Inhoud

Dit blok is gebaseerd op de eerste vier hoofdstukken van het derde deel van het tekstboek (Chapters 6 t/m 9). Hierin wordt een aantal belangrijke taalconcepten van een wat hoger niveau bestudeerd: inkapselingstechnieken (Chapters 6 en 7), typesystemen (Chapter 8) en verschillende methoden om de programmaverwerking te onderbreken (Chapter 9). Ook bij deze vier hoofdstukken zijn werkboekeenheden ontwikkeld. Samen met de genoemde hoofdstukken uit het tekstboek vormen zij de leereenheden 5 t/m 8.

Omvang

Blok 2 bestaat uit vier leereenheden en wordt afgesloten met een samenvatting. De totale studielast van dit blok schatten we op 30 uur.

Blok 3: Parallel programmeren

Inhoud

Dit blok is gebaseerd op hoofdstuk 10 van het tekstboek (Chapter 10: Concurrency). Ook bij dit hoofdstuk zijn werkboekeenheden ontwikkeld. Samen met het hoofdstuk uit het tekstboek vormen zij leereenheid 10.

Die leereenheid gaat in op parallelle concepten op hoger niveau en op gestructureerd parallel programmeren. In de tweede leereenheid (11) van dit blok worden als casussen de talen Java en Haskell nader beschouwd. Bij deze laatste leereenheid maken we geen gebruik van het tekstboek.

Omvang

Blok 3 bestaat uit twee leereenheden en wordt afgesloten met een samenvatting. De totale studielast van dit blok schatten we op 12 uur.

Inhoud

Blok 4: Programmeerparadigma's
In dit laatste, afsluitende blok gaan we de verschillende programmeerparadigma's nog eens nader beschouwen en op relevante aspecten met elkaar vergelijken. Zo kijken we in leereenheid 12 – gebaseerd op hoofdstuk 11 van het tekstboek – naar het imperatieve programmeerparadigma, waarbij we vooral de twee voorbeeldtalen C en Ada nog eens nader tegen het licht houden. In leereenheid 13 wordt het objectgeoriënteerde paradigma vergeleken met het imperatieve paradigma. Hierbij wordt hoofdstuk 12 van het tekstboek gebruikt.
De karakteristieken van het functionele paradigma komen in leereenheid 14 aan bod. Ook wordt in deze leereenheid het functionele paradigma vergeleken met het objectgeoriënteerde paradigma. Leereenheid 14 maakt gebruik van hoofdstuk 14 uit het tekstboek.
Logisch programmeren komt in leereenheid 15 voor het eerst aan de orde, waarbij we gebruikmaken van hoofdstuk 15 van het tekstboek. Dat is ook het moment waarop de drie paradigma's imperatief, functioneel en logisch tegen elkaar worden afgezet.
In dit blok komt verder scripting aan de orde, in leereenheid 16 die gebaseerd is op hoofdstuk 16 van het tekstboek.
Leereenheid 17 maakt geen gebruik van het tekstboek. In deze leereenheid worden aspectgeoriënteerd programmeren en de multiparadigmataal Scala besproken.
In de laatste leereenheid van dit blok – gedeeltelijk gebaseerd op de hoofdstukken 17 en 18 van het tekstboek – worden enkele conclusies geformuleerd.

Omvang

Blok 4 bestaat uit zeven leereenheden en bevat geen samenvatting. De totale studielast van dit blok schatten we op 40 uur.

Een overzicht van de opbouw van de cursus en de verdeling van de studielast over de verschillende onderdelen van de cursus zien we in tabel1.

TABEL 1 Opbouw van de cursus en studielast onderdelen

onderdeel	studielast
Introductie tot de cursus	1 uur
Blok 1: Basisconcepten	43 uur
Blok 2: Geavanceerde concepten	30 uur
Blok 3: Parallel programmeren	12 uur
Blok 4: Programmeerparadigma's	40 uur
Eindtoets	2 uur
totaal	128 uur

Tekstboek: voorwoord en hoofdstuk 1

Wij raden u aan om nu het voorwoord (Preface) en het eerste hoofdstuk van het tekstboek (Programming languages) door te lezen.

De relatie tussen de leereenheden van het werkboek en de hoofdstukken van het tekstboek kunt u ook terugvinden op de structuurpagina van deze cursus.

2.5 ACHTERGRONDLITERATUUR

Het tekstboek bevat tal van literatuurverwijzingen. Zie de paragrafen 'Further reading' na elk hoofdstuk.

3 Aanwijzingen voor het bestuderen van de cursus

3.1 OVERZICHT VAN HET CURSUSMATERIAAL

Het cursusmateriaal bestaat uit de volgende onderdelen:

Het *tekstboek: Programming Language Design Concepts* van David A. Watt.

Een werkboek in twee delen. Het werkboek heeft vier functies:
– het splitsen van de leerstof in leereenheden
– het geven van aanvullende leerstof op het tekstboek
– het structureren van de leerstof
– het helpen met het verwerken van de stof door middel van opgaven en het toetsen of u de stof op voldoende niveau beheerst.

Op de *cursussite* op yOUlearn vindt u de meest actuele informatie over de cursus en alle informatie die niet of moeilijk statisch is vast te leggen in de cursusboeken. In het bijzonder gaat het daarbij om de volgende zaken:
– informatie over de cursus en de organisatie van de studie
– informatie over de begeleiding en tentaminering
– informatie over de installatie van de programmatuur
– de bouwstenen bij de verschillende leereenheden
– cursusnieuws, errata en discussiegroep.

De bouwstenen bevat programma's in Haskell, Java en Prolog. Alle Haskell-programma's uit een bepaalde leereenheid staan in een bestand Lexx.hs in directory Examples, waarbij xx het nummer van de leereenheid is. Alle Java-programma's uit een bepaalde leereenheid staan in een directory Lexx, waarbij xx het nummer van de leereenheid is.

Belangrijk

U bestudeert de cursus *vanuit het werkboek*. Elk van de genoemde vier functies van het werkboek nemen we in de volgende paragrafen apart door.

3.2 DE LEEREENHEDEN

De totale stof van de cursus is verdeeld in leereenheden. Elke leereenheid bestaat uit een afgeronde hoeveelheid stof met een omvang die varieert van 3 tot 13 uur.

Introductie

Iedere leereenheid begint met een *introductie* waarin we een kader scheppen voor de stof die we in de leereenheid behandelen. In de introductie zijn ook de *leerdoelen* opgenomen. Zij geven aan welke kennis, inzichten en vaardigheden u zich eigen moet maken bij het bestuderen van de leereenheid. Tot slot bevat de introductie *studeeraanwijzingen*, waarin onder andere de studielast van de leereenheid is aangegeven en de relatie met het tekstboek staat vermeld.

Studeeraan-wijzingen

Leerkern

De *leerkern* beschrijft de wijze van studeren. Er wordt per paragraaf duidelijk aangegeven wanneer u welke theorie uit het tekstboek moet bestuderen. Ook de aanvulling op de leerstof van het tekstboek staat in de leerkern. Daarnaast bevat de leerkern ook opgaven; de uitwerking daarvan treft u aan in de *terugkoppeling*.
Na de leerkern volgt een *zelftoets*. Ook de uitwerkingen van de opgaven in de zelftoets zijn opgenomen in de terugkoppeling. De functie van opgaven, zelftoets en terugkoppeling bespreken we in paragraaf 3.4.

Terminologie

We gebruiken het woord 'hoofdstuk' om een onderdeel in het tekstboek aan te duiden en het woord 'leereenheid' voor een onderdeel van het werkboek.
In het tekstboek zijn de voorbeelden (examples) voorzien van een nummering bestaande uit een hoofdstuknummer en een volgnummer. In het werkboek wordt dezelfde nummering gebruikt om de voorbeelden aan te duiden. Het eerste deel van het nummer betreft het hoofdstuk in het tekstboek. Bijvoorbeeld: voorbeeld 4.6 betreft example met volg-nummer 6 in hoofdstuk 4 van het tekstboek van Watt, ongeacht de leereenheid van het werkboek waarin het voorbeeld ter sprake komt.

3.3 STRUCTURERING VAN DE STOF

Hoewel wij van u verwachten dat u de eerdergenoemde hoofdstukken uit het tekstboek allemaal volledig doorneemt, is het niet zo dat u alle daarin voorkomende onderwerpen even goed hoeft te beheersen. Het werkboek helpt u om belangrijke en minder belangrijke zaken te onderscheiden.

Leerdoelen

Een van de manieren waarop dit gebeurt is het formuleren van *leerdoelen*. Wat hierin vermeld staat, wordt u geacht te weten, te begrijpen of te kunnen. Omgekeerd hoeft u datgene wat niet expliciet in de leerdoelen staat, ook niet te weten of te kunnen, voor het tentamen althans. Als u erover twijfelt hoe belangrijk een passage is, kunnen de leerdoelen u dus helpen.

Kernbegrippen

Het eerste leerdoel van veel leereenheden somt de kernbegrippen uit die leereenheid op. Van deze begrippen moet u precies weten wat ze betekenen. Van een aantal van deze kernbegrippen zult u dan ook in de zelftoets de betekenis moeten uitleggen. Deze begrippen vindt u ook in het register achterin deel 2.

Verklarende woordenlijst

Een tweede hulpmiddel dat de werkboekdelen bieden is een uitgebreide *verklarende woordenlijst*. In het tekstboek en in het werkboek worden regelmatig termen gebruikt uit andere takken van de informatica of uit de wiskunde, die niet expliciet worden uitgelegd. Of u die termen kent, hangt sterk af van uw vooropleiding of ervaring. We verwachten bijvoorbeeld dat veel van onze studenten bekend zijn met termen als bijvoorbeeld *argument* of *lexicografische ordening*, maar dat zal niet voor

iedereen gelden. Verklaringen voor dergelijke termen hebben we echter niet in de leereenheden zelf opgenomen, maar bij elkaar gezet in een verklarende woordenlijst. Eventueel voorzien we de termen ook van gangbare Nederlandstalige of Engelstalige equivalenten. Ook van alle kernbegrippen uit de cursus bevat de verklarende woordenlijst een korte omschrijving.

3.4 OPGAVEN EN ZELFTOETSEN

Opgaven

De opgaven en de zelftoetsen vervullen in een leereenheid elk een aparte rol. De *opgaven* zijn bedoeld om u de stof te laten verwerken door er actief mee om te gaan. U krijgt het grootste rendement van deze opgaven als u ze tijdens het bestuderen van de leerkern eerst zelf zo volledig mogelijk probeert uit te werken en pas daarna in de terugkoppeling uw uitwerking vergelijkt met de daar gegeven uitwerking.

Zelftoets

De vragen en opdrachten uit de *zelftoets* zijn daarentegen vooral bedoeld om u te helpen beoordelen of u de leerstof uit de betreffende leereenheid in voldoende mate beheerst. Als u nog veel moeite heeft met de opgaven van de zelftoets, is het verstandig eerst bepaalde delen van de leereenheid opnieuw te bestuderen alvorens verder te gaan met de volgende leereenheid. De zelftoetsopgaven zijn overigens niet geheel representatief voor het tentamen, aangezien bij het tentamen geprobeerd wordt vragen te stellen die de leerstof van één leereenheid overstijgen. Ook zullen we in de zelftoets vaak eenvoudige kennisvragen stellen, terwijl die op het tentamen minder vaak voorkomen.
In die zin is de eindtoets meer representatief voor het niveau van het tentamen dan de zelftoetsen per leereenheid (zie ook paragraaf 6.2).

4 Programmatuur en documentatie

4.1 PROGRAMMATUUR

Systeem-specificaties

De programmatuur bij de cursus kan draaien op pc's onder Windows. Om de cursus te bestuderen verwachten we dat u over een recente versie van Java beschikt. De Java-programma's van de bouwstenen zijn getest met versie SDK 1.6 van Java.

In deze cursus wordt gebruikgemaakt van de volgende open-licentiesoftware:
– Tekstverwerker Notepad++
– Haskell-interpretator GHCI of WinGHCi
– Prolog-interpretator GNU Prolog.
Deze software kunt u van internet ophalen.

Informatie over de installatie van de programmatuur vindt u op de *cursussite* op Studienet.

4.2 DOCUMENTATIE

Documentatie voor het gebruik van de tekst-editor Notepad++ en de interpretatoren (Win)GHCI en GNU Prolog vindt u via de *helpfunctie* van deze programma's.

5 Studiebegeleiding

Concepten van programmeertalen is een verplichte cursus van de bacheloropleiding Informatica. Informatie over de *groepsbegeleiding* van de cursus kunt u vinden op de *cursussite* op Studienet. Buiten de periode waarin groepsbegeleiding plaatsvindt, is er alleen standaardstudiebegeleiding via Studienet.

6 Tentaminering

6.1 TENTAMEN

Schriftelijk tentamen

De cursus wordt afgesloten met een *schriftelijk tentamen* van 3 uur. De opgaven zijn qua aantal en moeilijkheidsgraad vergelijkbaar met de opgaven van de eindtoets van de cursus. Tijdens het tentamen mag *geen* cursusmateriaal worden geraadpleegd. Meer informatie over het tentamen kunt u vinden in de *Studiegids Informatica* en op de *cursussite* op Studienet.

6.2 EINDTOETS

Eindtoets

Aan het eind van het tweede werkboekdeel is een *eindtoets* opgenomen die representatief is voor het schriftelijk gedeelte van het tentamen. We raden u sterk aan deze eindtoets pas te maken als u klaar bent met de tentamenvoorbereiding.

7 Oorspronkelijk cursusteam

Het werkboek is gebaseerd op drie eerdere versies van de cursus.

De auteurs van de eerste versies van de cursus zijn in alfabetische volgorde:

drs. J.D. Fokker
prof. dr. J.Th. Jeuring
ir. H. Koppelman
C.A. Nolet
drs. Th. F. de Ridder
prof. dr. S.D. Swierstra
ir. F.J. Wester
M. Witsiers-Voglet