Gevorderd Programmeren

Gevorderd Programmeren	
Richting	Informatica
Jaar	2BINF

Bespreking

Theorie

De theorie van dit vak wordt gegeven door Professor Broeckhove. Het is ten zeerste aangeraden om naar de lessen te gaan van dit vak aangezien deze interessant en leerrijk zijn. Om de theorie van dit vak te leren krijg je zowel een cursus als een relatief moeilijk handboek, ook worden de slides van tijdens de lessen beschikbaar gesteld. Zoals U misschien al gemerkt hebt is dit dus een hele brok, ook bij dit vak geldt echter dat het belangrijk is om de informatie uit het handboek en de cursus te begrijpen. Er worden weinig puur theoretische vragen gesteld en de meeste vragen zijn dus ook doordenkvragen.

De evaluatie van het vak bestaat uit

- een Trace-opdracht: code wordt gegeven en men vraagt de student om de output te redeneren.
- een schriftelijk examen op de pc, waarvan de vragen gebaseerd zijn op de C++ FAQs.

Praktijk

Er worden elke week een aantal opdrachtjes gegeven die niet verplicht ingediend moeten worden, toch is het zeker interessant om de laatste paar lessen de praktijk te volgen. Tijdens deze lessen wordt er namelijk uitleg gegeven in verband met de Boost library en de nieuwe C++ standaard. Naast het semesterproject wordt er eveneens ook altijd een examenopdracht van vorige jaren gegeven, ter voorbereiding van het prakijkexamen.

Puntenverdeling

Het examen gevorderd programmeren bestaat uit 4 onderdelen:

1. Programmeerproject doorheen het semester
2. Praktisch examen programmeren
3. Trace-opdracht
4. Theoretisch examen

Een minimum score van 10/20 op elk onderdeel is nodig om een slagend cijfer te behalen.

Examenvragen

Academiejaar 2022-2023 - 1ste zittijd

Media:AP_theory_2022-2023.pdf

Academiejaar 2021 - 2022 - 1ste zittijd

Theorie

Media:GevProg-Examen-21-22-1ezit.pdf

Academiejaar 2020 - 2021 - 1ste zittijd

Theorie

Media:GevProg-Examen-20-21-1ezit.pdf

Tracing

Media:Trace_opdracht_2020_2021_1ezit.pdf

Praktijk

Academiejaar 2019 - 2020 - 2de zittijd

Theorie

Media:GevProg-Examen-19-20-2ezit.pdf

Tracing

Media:Trace_opdracht_2019_2020_augustus.pdf

Oefeningen

Media:Programmeeropdracht_2019_2020_2.pdf

Academiejaar 2019 - 2020 - 1ste zittijd

Theorie

Media:GevProg-Examen-19-20-1ezit.pdf

Tracing

Media:Trace_opdracht_2019_2020_january.pdf

Oefeningen

Media:Programmeeropdracht_2019_2020.pdf

Academiejaar 2018 - 2019 - 2de zittijd

Tracing

Media:TraceGP1819Z2.pdf

Academiejaar 2018 - 2019 - 1ste zittijd

Theorie

Media:TheorieGP1819Z1.pdf

Tracing

Media:TraceGP1819Z1.pdf

Oefeningen

Media:ProgrammeerGP1819Z1.pdf

Academiejaar 2017 - 2018 - 1ste zittijd

Theorie

Media:TheorieGP201720181EZIT.pdf

Tracing

Media:TraceGP201720181EZIT.pdf

Oefeningen

Media:ProgrammeeropdrachtGP201720181EZIT.pdf

Bonus

De boys (en girl) van dit jaar hebben geprobeerd zo veel mogelijk examenvragen pre 2017-2018 uit te werken.

Dit heeft ons enorm geholpen, dus hopelijk jullie ook :)

Media:BonusGP2017-20181EZIT.pdf

Academiejaar 2016 - 2017 - 2de zittijd

Theorie

Media:GevorderProgrammeren_2dezit.pdf

Oefeningen

Media:GP_OEF_2016_2017_2DE.pdf

Tracing

Media:Trace20162017_augustus.pdf

Academiejaar 2016 - 2017 - 1ste zittijd

Theorie

Media:Examenvragen_GP_2016_2017_stezit.pdf

Oefeningen

Media:Examenvragen_oefenigen_GP_2016_2017_stezit.pdf

Academiejaar 2015 - 2016 - 2de zittijd

Theorie

Media:Examenvragen_GP_2015_2016_2dezit.pdf

Academiejaar 2015 - 2016 - 1ste zittijd

Theorie

Media:Examenvragen_GP_2015_2016_1stezit.pdf

Academiejaar 2013 - 2014 - 1ste zittijd

Theorie

Media:Examenvragen_GP_2013_2014_stezit.pdf

Praktijk

Media:Gp1314.pdf

Academiejaar 2008 - 2009 - 1ste zittijd

Theorie

Aangezien het theoretische gedeelte mondeling afgenomen wordt, hier enkele reconstructies door enkele studenten:

-
  - Welk stuk van de leerstof ken je het beste? (Functies en klassen) Vertel.
  - Wat is een aggregatie en compositie relatie en leg uit wat shallow en deep copy hiermee te maken hebben?
  - Wat is closed behaviour?
  - Leg uit: overerving, hiding en overloading tussen een afgeleide klasse en zijn basis.
-
  - Leg het stuk da je het best beheerst uit de cursus uit. Start direct! Als je alles evengoed beheerst zal ik er iets uitnemen.
  - Generisch Programmeren: wat is het? hoe kan men in c++ gebruik maken van generisch programmeren?
-
  - Leg een deel uit dat je het tofste vindt, of het beste beheerst, ... (Abstracte Base klassen)
  - Wat is een interface exact en hoe verhoudt deze zich in de verzameling van Abstracte base classes?
  - Leg wat meer uit met betrekking tot de 3 sectoren in een classe definitie, en leg de link met overerving van abstracte base klassen uit.
  - Wat uitleg geven over co, contra-variantie van de pre- en postcondities.
  - Voorbeeld geven van Closed Behaviour principle.
-
  - Leg verschil uit tussen: instance-datamembers en klasse-datamembers
  - Zijn er dan ook instance-memberfuncties en klasse-memberfuncties?
  - Hoe roep ik deze op voor een klasse-memberfunctie?
-
  - Stel functie a.f()); en b.f(); Hoe weet de compiler welke functie aan welk object gebonden is?
  - Bij de code onderaan, welke functie wordt er uitgevoerd? Vervolgens enkele bijvragen over varianties op de code.

```
class A{
void f()
};

class B : public A {
void f()
};

B* b = new(B);
b.f();
```

- Welk stuk van de leerstof ken je het beste? (Principle of closed behaviour) Vertel.
- Vervolgens een uitwijking naar de formele OO-leer.
- Enkele vragen in verband met inheritance, polymorfisme en zelfs wat doorvragen over hoe polymorfisme gerealiseerd wordt door de compiler.
- Oppervlakkige vragen over templates.

Praktijk

Aangezien deze opgaven beschikbaar zijn op aanvraag bij de praktijkassistent van dienst, volgt hieronder een verkorte opgave.

Inleiding & uitweiding

Britse wiskundige John Horton Conway vond in 1970 een zeer bekend cellulaire automaat uit, "The Game of Life". Het "spel" bestaat uit een 2D rooster van cellen die al dan niet "leven" (denk: ze hebben de waarde 1 of 0). Gegeven een begintoestand, mogelijk random, wordt de simulatie stap per stap uitgevoerd (generatie per generatie) en is onderworpen aan de volgende regels:

- Een levende cel blijft alleen leven in de volgende generatie als ze 2 of 3 leven buren heeft in de huidige generatie. (anders sterf ze door onder- of overpopulatie)
- Een dode cel komt tot leven in de volgende generatie als ze exact 3 levende buren heeft in de huidige generatie. (reproductie)

Afhankelijk van de beginsituaties kunnen tijdens de evolutie verschillende patronen voorkomen: min of meer chaotische evoluties, invariante objecten, objecten met terugkerende periode, ... Heel wat onderzoek is intussen gebeurd naar cellulaire automaten en hun toepasbaarheid. Conway's Game of Life is dus zeker niet de enige set van regels die een interessante cellulaire automaat oplevert. Denk maar aan:

- Het "speelveld" hoeft niet 2D te zijn en hoeft geen triviale geometrie te hebben
- De regels van overleven en reproductie kunnen veel gecompliceerder zijn met verschillende fitness waarden
- Verschillende species kunnen samenleven

Bij het ontwerpen van code die zulke cellulaire automaten kan simuleren is het dus nuttig om een zo flexibel mogelijke structuur te ontwikkelen.

De opgave

De opgave bestaat uit het schrijven van een cellulaire automaat-simulator die o.a. de Conway's Game of Life kan simuleren; maar ook andere cel-soorten moeten kunnen worden gebruikt. Het ontwerp moet het conceptueel mogelijk maken om cellen op willekeurige manieren aan elkaar te koppelen en dus ook niet noodzakelijk op een regelmatig 2D rooster. Gebruik de ontwikkelde structuur om een voorbeeld-implementatie van een 2D automaat te maken met verschillende cel-soorten.

Academiejaar 2008 - 2009 - 1ste zittijd

1. Syntax

    1. Declaratie-syntax: Stel de volgende C++ declaraties op, zonder gebruik te maken van typedef statements:

        1. Een const pointer naar array van 3 pointers naar double, geïnitializeerd op nul.

        2. Een pointer naar een const pointer naar een functie zonder terugkeerparameteren met reference naar double als oproepparameter, geïnitializeerd op nul.

        3. Een reference naar een functie met integer als terugkeerwaarde en integer als argument, geïnitializeerd door een gegeven functie g.

        4. Een array van tien pointers naar functie met reference naar integer als argument en pointer naar double als return.

    2. Template code: Omschrijf, gebruik makend van de correcte termen, de elementen in onderstaande template code:

```
template< typename It>
It
fff(It first, It last)
{
        typename It::value_type val = typename It::value_type();
        for (It it = first; it != last; ++it) {
                if (value == *it) return it;
        }
        return last;
}
```

    Schrijf een zo kort mogelijk stukje code waarin je dit template gebruikt (het stukje code hoeft niet zinvol te zijn als het maar toont dat je begrijpt hoe het template gebruikt kan worden) waarbij je via een include directive de bovenstaande definitie includeert.

2. Definities en begrippen:

    1. Leg zo kort mogelijk de volgende begrippen uit:

        1. automatic storage duration

        2. namespace name lookup (ook wel Koenig lookup genoemd)

        3. const member functie

        4. exception matching in een catch clause

    2. Geef telkens zo kort mogelijk de verbanden/verschillen uit tussen de twee begrippen:

        1. static versus object member van een klasse

        2. static versus dynamic type in C++

        3. co-variantie versus contra-variantie in een afleidingshierarchie

3. Taalconstructen

    Naast de default en geparametriseerde constructoren zijn de copy constructor, de assignment operator en de destructor essentiële methoden in een klasse. Ze worden soms ook wel de Grote Drie (Big Three in C++ FAQ's) genoemd. Gegeven een klasse X met als enig datamember een (niet const) pointer naar object van klasse Y. De geparametriseerde constructer heeft een Y& argument (met de voor de hand liggende bedoeling dat het argument gebruikt wordt als referent of pointee van het Y* datamember).

    1. Geef de klasse definitie van X zonder method bodies

    2. Schrijf de methode definities (buiten de klasse) voor het shallow copy scenario (pointer is geen eigenaar van de pointee of referent)

    3. Schrijf de methode definities (buiten de klasse) voor het deep copy scenario (pointer is eigenaar van de pointee of referent)

    4. Wat is het synthetiseren van de hogervermelde methodes; voor welke van de methodes gebeurd het en in welke omstandigheden?

4. Taalconstructen

   Leg, in de context van publieke inheritance, zo kort mogelijk de volgende aspecten van polymorphe klassen uit:

   1. Definitie van een polymorphe klasse
   2. Het onderscheid tussen polymorphe, abstracte en interface klassen?
   3. Aan welke voorwaarden moet er voldaan zijn opdat een memberfunctie polymorph ageert? Schrijf een zo kort mogelijk codefragment met een polymorphe oproep
   4. Wat is upcasting versus downcasting? Geef een zo kort mogelijk codefragment dat upcasting uitvoert. Geef een zo kort mogelijk codefragment dat downcasting uitvoert.
   5. Voor de destructor van polymorphe klasse geldt een dwingende codeer richtlijn. Wat is deze? Illustreer de noodzaak van de voorgaande richtlijn met een zo kort mogelijk code-fragment.

5. Programma-interpretatie

   Geef de output van het bijgevoegde programma ’exam9.cpp’ weer. Daar waar adressen worden uitgeprint mag je <<adres-van variabele naam>> als symbolische aanduidingen gebruiken. De klassen die voorkomen werden ook in de voorbeeldprogramma’s gebruik en in geen enkel opzicht gewijzigd. *De programmacode is niet in deze tuyeaux opgenomen, maar het idee achter de vraag telt hier.*

6. Programma-interpretatie: Analoog aan vorige vraag maar voor een ander bestand.

Academiejaar 2007 - 2008 - 1ste zittijd

1. Syntax

    1. Declaratie-syntax: Stel de volgende C++ declaraties op, zonder gebruik te maken van typedef statements:

        1. Een array van twee array's van drie doubles, alle geïnitializeerd op nul.

        2. Een pointer naar een const pointer naar een functie zonder terugkeerparameter en met een reference naar een double als oproepparameter, geïnitializeerd op nul.

        3. Een pointer naar een array van drie pointers naar functies void pointer als oproepparameter en zonder terugkeerparameter.

        4. Een functie zonder terugkeerparameter en als oproepparameter een pointer naar functie met als terugkeerparameter een int en als oproepparameter een pointer naar double.

        5. Een klasse C met in de private sectie een const double als datamember en in de publieke sectie een constructor, met oproepparameter double x, die de datamember initialiseert met x.

    2. Template code: Schrijf een functie template dat alle elementen van eens equence doorloopt en telt hoeveel keer de defaultwaarde van het waardetype van de sequence optreedt en dat aantal als unsigned int teruggeeft.

2. Programma-interpretatie: Geef de output van het bijgevoegde programma 'exam5.cpp' weer. Daar waar adressen worden uitgeprint mag je <<adres-van variabele naam>> als symbolische aanduidingen gebruiken. De klassen die voorkomen werden ook in de voorbeeldprogramma's gebruik en in geen enkel opzicht gewijzigd. *De programmacode is niet in deze tuyeaux opgenomen, maar het idee achter de vraag telt hier.*

3. Programma-interpretatie: Analoog aan vorige vraag maar voor een ander bestand.

4. Definities en begrippen:
    1. Leg zo kort mogelijk de volgende begrippen uit:
        1. Storage duration (en in het bijzonder de drie categorieën in C++)
        2. Linkage (en in het bijzonder de drie categorieën in C++)
        3. Virtual inheritance
        4. Type conformiteit
    2. Geef telkens zo kort mogelijk de verbanden/verschillen uit tussen de twee begrippen:
        1. Dynamisch type en late binding.
        2. Publieke versus private inheritance.
        3. Liskov substitutie principe versus methode precondities.
5. Taalconstructen: Leg zo kort mogelijk de volgende aspecten uit van exception handling in C++:
    1. basisfilosofie
    2. try-catch structuur
    3. stack frame unwinding mechanisme
    4. matching in catch clause
6. Taalconstructen: ;eg zo kort mogelijk de volgende aspecten van operator overloading uit:
    1. doel
    2. memberfunctie versus helper implementatie
    3. interpretatieschema van uitdrukkingen als operator oproep
    4. beperkingen

    Geef een voorbeeld van een klassen met één double als datamember en de vier operatoren == en +=+= en ++ en <<<< (alle met hun gebruikelijke betekenis).

Academiejaar 2005 - 2006 - 1ste zittijd

Mondeling, open boek. De vragen vormen hoe dan ook slechts een startspunt van discussie. Een selectie, van zeer open tot redelijk gesloten vraagstelling:

1. Leg me het stuk C++ uit dat U het best kent.
2. Becommentarieer een van de stukken uit de C++ standaard die in de cursus gerefereerd worden.
3. Welke beperkingen worden er aan inheritance opgelegd in C++.
4. Hoe interageren de fenomenen van overriding en hiding. Illustreer met een zo beknopt mogelijk voorbeeld.
5. Wat is polymorphisme. Hoe activeert met het in C++.
6. Wat zijn de implicaties van het gebruik van de keywords public, protected en private.