# 

**Faculteit Natuur en Techniek**

|  |  |
| --- | --- |
| Collegejaar | 2015-2016 |
| Periode | D |
| Opleiding | ICT |
| Osiris Cursusnaam | Tentamen OOP C++ |
| Osiris Cursuscode | TICT-V1OOPC-15 |
| Tijdsduur tentamen | 90 minuten |
| Tentamendatum | proeftentamen |
| Tijdstip | proeftentamen |
|  |  |
| **Invullen door de docent** |  |
| Meerdere versies | Nee |
| Docent(en) | Wouter van Ooijen, Arno Kamphuis |
| Aantal pagina’s (incl. voorblad) | 10 |
| Rekenmachine | Geen rekenmachine |
| Andere toegestane hulpmiddelen | [Andere toegestane hulpmiddelen] |
| Aantal vragen | 3 open vragen, 10 multiple-choice vragen |
| Soort vragen | Open en MC vragen |
| Antwoorden uitwerken op | Uitwerkpapier (lijntjes) |
|  | Multiple Choice antwoorden invullen op de laatste blz. van het tentamen |
| Kladpapier toegestaan | *Ja* |
|  |  |
|  |  |

**Invullen door de student**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Studentnummer en naam student |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| Naam docent |  | | | | | | | |

|  |  |
| --- | --- |
|  | **~~Tentamenopgave mag NIET worden meegenomen, gefotografeerd, overgeschreven of op andere wijze gedupliceerd worden!~~**  **(Dit geldt wel voor het echte tentamen)** |

|  |
| --- |
| Bij iedere open vraag is aangegeven hoeveel punten u met die vraag kunt behalen. Uw cijfer is het totaal aantal punten gedeeld door 10, met een mimum van 1.0.  Als uw antwoord code bevat zal dit niet beoordeld worden op ‘kleine’ syntaxfouten, d.w.z. die fouten die door de compiler gesignaleerd zouden worden, en waarvan de verbetering vanzelf spreekt.  Mogelijk zijn delen van de vraag **vet** gedrukt of gemarkeerd met \*sterretjes\*. Het is aan te raden daar extra op te letten.  De 10 multiplechoice vragen kunnen maximaal 25 punten opleveren. Een kwart van de MC vragen goed beantwoord levert 0 punten op, dit stijgt lineair tot 25 punten bij 10 MC vragen goed beantwoord.  Een goed antwoord verhoogt uw score, een fout (of geen) antwoord verlaagt uw score niet. Vul dus altijd een antwoord in. De antwoorden op de multiple choice vragen vult u in in de tabel achteraan. Als u een multiple choice antwoord wilt verbeteren kruist u het foute antwoord door en vult u daarachter het goede antwoord in.  Als een vraag (of bij multiple choice ook: een of meer antwoorden) volgens u een fout bevatten die u hinderen bij het geven van uw antwoord schrijf dit dan op. |

1. [25 punten]

|  |
| --- |
|  |

In een GIS (Geografisch Informatie Systeem) wordt gebruik gemaakt van een ADT voor een coordinaat (x en y positie), en er worden knooppunten (plekken waar wegen samenkomen) en verbindingen (wegen die knooppunten met elkaar verbinden) opgeslagen. In een verbinding wordt ook opgeslagen hoe lang de weg echt is, wat in het algemeen langer zal zijn dan de rechte lijn tussen de twee knooppunten.

1. Schrijf de 3 klasse definities, inclusief de constructors.
2. De functie efficiency in de klasse verbinding moet teruggeven hoe efficient (waarde 0..1) een verbinding is tov. een rechte lijn. Schrijf deze functie en geef aan wat er eventueel moet veranderen of worden toegevoegd om dit mogelijke te maken (er zijn meerdere mogelijkheden, je hoeft er maar 1 te noemen).

Je mag aannemenen dat er functies beschikbaar zijn voor het berekenen van een kwadraat en een wortel.

1. [25 punten]

Een integer queue (wachtrij, buffer) kan een reeks (maximaal 10) integers opslaan. De functie put(x) voegt de integer x achteraan de wachtrij toe (behalve als x gelijk aan 0, dan gebeurt er niets), de functie get() haalt de voorste integer uit de wachtrij en geeft die terug als resultaat. Als de wachtrij leeg is dan geeft get() de waarde 0 terug.

Schrijf hiervoor een ADT (Abstract Data Type) (klasse definitie, met de implementatie van de functies zo veel mogelijk in de klasse definities) die de volgende operaties ondersteunt:

1. Een constructor zonder parameters (de wachtrij is initieel leeg)
2. Een constructor met als argument een wachtrij (de nieuw geconstrueerde wachtrij bevat daarna dezelfde waarden als het argument).
3. De get() en put() functies
4. De += operator die de elementen uit het rechter argument toevoegt achteraan het linker argument, zonder het rechter argument te veranderen. (Dit kan je eenvoudig doen door eerst een kopie van het rechter argument te maken.)
5. De + operator op twee wachtrijen die als resultaat een wachtrij teruggeeft met daarin eerste de elementen van de linker wachtrij, en dan de elementen van de rechter wachtrij. (Je kan gebruik maken van de += operator).

Let op:

Waar dat redelijkerwijs door een gebruiker verwacht mag worden, mogen de argumenten van een functie door het aanroepen van die functie niet veranderd worden, dus alles wat const kan zijn, moet ook const zijn.

1. [25 punten]

|  |  |
| --- | --- |
|  | class Pizza {  protected:  double prijs;  public:  Pizza( double prijs ): prijs( prijs ){}  virtual double getPrijs(){ return prijs; }  virtual std::string naam(){ return "pizza"; }  }; |

Gegeven de klasse Pizza. Van deze Pizza kan je de prijs (zoals opgegeven bij de constructor aanroep) en de naam (“pizza”) opvragen. Standaard is een pizza alleen de bodem met tomatensaus er op.

We willen nu pizza’s samen kunnen stellen door er verschillende toppings te doen (bijvoorbeeld kaas, ham, ui, pepperoni, etc...). Deze toppings hebben een naam en een prijs die bovenop de prijs voor de pizza komt. Als je aan een topping object zijn prijs opvraagt krijg zijn topping prijs, plus de prijs van alle onderliggende toppings, plus de prijs van de basis pizza. Als je aan een topping object zijn naam vraagt dan krijg je de volledige naam van de pizza, dus bv. “pizza met ham met ananas”.

Ontwerp in UML en schrijf in C++ een decorator klasse Topping. Geef dus zowel het UML klassendiagram (inclusief de relatie met Pizza) als de C++ code, inclusief de implementatie (mag in de klasse) van de constructor en de beide functies.

Geef tevens een stukje voorbeeld code om met de Pizza en de Topping klassen een pizza te maken met kaas (€ 0,49) en pepperoni (€ 0,79). De basis pizza kost € 4,99.

🡺 Voer je antwoorden in op de laatste bladzijde, dat is de lijst die wordt nagekeken.

|  |
| --- |
| 1) Wat is de relatie tussen een struct en een class? |
| A: een struct kan geen functies bevatten. B: een struct kan geen constructor hebben. C: de default toegankelijkheid is anders. D: er is geen verschil. |

|  |
| --- |
| 2) Hoe geef je bij de \*aanroep\* van een functie een parameter (in dit geval de persoon kees) 'by reference' door? |
| A: f( kees ); B: f( & kees ); C: f( const kees ); D: f( \* kees ); |

|  |
| --- |
| 3) Wat is waar voor een functie declaratie? |
| A: Alle parameters van een functie moeten defaults hebben. B: Een functie moet ten minste 1 default parameter hebben. C: Een functie mag maar (ten hoogste) 1 default parameter hebben. D: Een functie mag nul default parameters hebben. |

|  |
| --- |
| 4) Een methode wordt aangeroepen met het volgende statement. Wat kan de declaratie van deze methode zijn? |
| object.method( 15 ); |
| A: void method( int x ) const B: void method( int x[] )  C: void method( int & x ) D: void method( int \* x ) const |

|  |
| --- |
| 5) Wat is waar? Onder een 'nieuw operator symbool' wordt bv. \*\* voor machtsverheffen verstaan. |
| A: Je kan in C++ een nieuwe betekenis geven aan een bestaande operator, en een nieuwe operator symbolen definieeren. B: Je kan in C++ een nieuw operator symbool definieeren, maar de prioriteit kan je niet zelf aangeven. C: Je kan in C++ een nieuwe betekenis geven aan een bestaande operator, maar geen nieuwe operator symbolen definieeren. D: Je kan in C++ de prioriteiten van operatoren wijzigen, maar alleen voor de al bestaande operatoren. |

|  |
| --- |
| 6) Een operator<< die gebruikt wordt voor text uitvoer moet zijn linker argument 'by reference' meekrijgen. Wat is de reden hiervan? |
| A: Om een keten van << expressies mogelijk te maken. B: Omdat anders de wijzigingen in een copie van het argument worden gemaakt. C: Om te voorkomen dat er onnodig een copie wordt gemaakt. D: Anders kan de aanroeper niet checken of de operatie geslaagd is. |

|  |
| --- |
| 7) Wanneer maak je een methode virtual? |
| A: Altijd, want dit geeft meer flexibilieit dan een niet-virtuele methode. B: Als via (een pointer of een reference naar) de superklasse een methode in een subklasse aangeroepen moet worden. C: Als de implementatie in de superklasse leeg zou zijn. D: Als hij in een subklasse hergedefinieerd moet kunnen worden (aangegevn met 'override'). |

|  |
| --- |
| 8) Wat zal dit programma printen? |
| #include<iostream>  class Base { public:  virtual void show() { std::cout << " In Base "; } };  class Derived: public Base { public:  void show() override { std::cout << "In Derived "; } };  int main(){  Derived d;  Base \*bp = &d;  bp->show();   Base &br = \*bp;  br.show(); } |
| A: In Base In Base B: In Base In Derived C: In Derived In Derived D: In Derived In Base |

|  |
| --- |
| 9) Let goed op 'virtual'. Wat zal deze code printen? |
| #include <iostream> class fruit { public:  virtual const char \* naam(){ return "onbekend"; }  const char \* vorm(){ return "geen"; } };   class appel : public fruit { public:  const char \* naam(){ return "appel"; }  const char \* vorm(){ return "rond"; }  };   class peer : public fruit { public:  const char \* naam(){ return "peer"; }  const char \* vorm(){ return "druppelvormig"; } };   void print( fruit & f ){  std::cout << " een " << f.naam() << " is " << f.vorm(); }    . . .    appel a;  peer p;  print( a );  print( p ); |
| A: een appel is geen een peer is geen B: een appel is rond een peer is druppelvormig C: een onbekend is rond een onbekend is peervormig D: een onbekend is geen een onbekend is geen |

|  |
| --- |
| 10) Welke (implementatie) code kan gebruik maken van de protected attributen van een klasse? |
| A: De methoden van die klasse, en afgeleide klassen. B: De methoden van die klasse. C: De methoden van die klasse, en afgeleide klassen, en van superklassen. D: De methoden van die klasse, en van superklassen. |