# 

**Faculteit Natuur en Techniek**

|  |  |
| --- | --- |
| Collegejaar | 2016-2017 |
| Periode | B |
| Opleiding | ICT |
| Osiris Cursusnaam |  |
| Osiris Cursuscode | TCTI-V2CPSE2-16 |
| Tijdsduur tentamen |  |
| Tentamendatum | proeftentamen |
| Tijdstip | Tijdsduur: 90 minuten |
|  |  |
| **Invullen door de docent** |  |
| Meerdere versies | nee |
| Docent(en) | Wouter van Ooijen |
| Aantal pagina’s (incl. voorblad) | 10 |
| Rekenmachine | nee |
| Andere toegestane hulpmiddelen | nee |
| Aantal vragen | 2 open vragen en 20 gesloten (4-keuze) vragen |
| Soort vragen |
| Antwoorden uitwerken op | Open vragen op uitwerkpapier; gesloten (multiple-choice) op de invulbladzijde. |
| Kladpapier toegestaan | Ja |
|  |  |
|  |  |

**Invullen door de student**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Studentnummer en naam student |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| Naam docent | Wouter van Ooijen | | | | | | | |

|  |  |
| --- | --- |
|  | **Tentamenopgave mag NIET worden meegenomen, gefotografeerd, overgeschreven of op andere wijze gedupliceerd worden!** |

|  |
| --- |
| Bij iedere open vraag is aangegeven hoeveel punten je met die vraag kunt behalen. Je cijfer is het totaal aantal punten gedeeld door 10, met een mimum van 1.0.  Als je antwoord code bevat zal dit niet beoordeld worden op ‘kleine’ syntaxfouten, d.w.z. die fouten die door de compiler of assembler gesignaleerd zouden worden, en waarvan de verbetering vanzef spreekt.  Als je kladpapier gebruikt en inlevert, maarkeer dit dan duidelijk, bv. met een streep er doorheen.  De 20 multiple-choice vragen kunnen maximaal 50 punten opleveren. Een goed antwoord verhoogt je score, een fout (of geen) antwoord verlaagt je score niet. Vul dus altijd een antwoord in. De antwoorden op de multiple choice vragen vult je in in de tabel achteraan. Als je een multiple-choice antwoord wilt verbeteren kruist dan het foute antwoord door en vult daarachter het goede antwoord in. Let op dat je antwoord letters duidelijk zijn: in geval van twijfel wordt je antwoord fout gerekend.  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. [30] Gegeven de onderstaande code. In de klassen zijn allerlei details weggelaten. De string operator>> kan je gebruiken om het eerste woord uit een string te lezen (en te verwijderen).
2. Schrijf een combo klasse, met zijn constructor en lokale variabelen. Een combo is een scherm\_object dat is samengesteld uit (verwijzingen naar) twee andere scherm objecten. De constructor krijgt een object map mee (met daarin de al bestaande objecten), en een string met de namen van de twee onderliggende objecten. Zorg dat er geen dangling references of dangling pointers kunnen ontstaan, ook niet als er later elementen uit de map worden verwijderd.
3. Schrijf een factory functie die (op een effciente manier) een object\_map\_type teruggeeft. De functie krijgt als parameter een string\_vector waarvan ieder element een object beschrijft. Alle objecten moeten in de map terecht komen. Een voorbeeld van een parameter waarde is

{

“rechthoek stam bruin (50,50) (100,200)”,

“cirkel kruin (100,200) 60”,

“combo boom stam kruin”

}

|  |
| --- |
| class scherm\_object { . . . };  class rechthoek : pubic scherm\_object {  public:  // bv. s == “bruin (50,50) (100,200)”  rechthoek( std::string s ){ . . . }  };  class cirkel : pubic scherm\_object {  public:  // bv s == “groen (100,200) 60”  cirkel( std::string s ){ . . . }  };  // lees en verwijder separators uit source.  // lees en verwijder tekens uit source en stop die in dest,  // totaan de eerste separator.  std::string & operator>>( std::string & source, std::string & dest ){  ... }  typedef std::shared\_ptr< scherm\_object > scherm\_object\_ptr;  typedef std::map< std::string, scherm\_object\_ptr > object\_map\_type;  typedef std::vector< std::string > string\_vector; |

[15]

|  |
| --- |
| class combo : public scherm\_object {  private:  scherm\_object\_ptr subjects[ 2 ];  public:  combo( const object\_map\_type & map, std::string & s ){  std::string s1, s2;  s >> s1 >> s2;  subjects[ 0 ] = map[ s1 ];  subjects[ 1 ] = map[ s2 ];  }  } |

[15]

|  |
| --- |
| typedef unique\_ptr< object\_map\_type > map\_ptr;  map\_ptr make\_map(string\_vector &v ){  p = make\_unique< object\_map\_type >();  for( auto & s : v ){  std::string name;  s >> item >> name;  if( item == “rechthoek” ){  \*p[ name ] = rechthoek( s );  } else if(item == “cirkel” ){  \*p[ name ] = cirkel( s );  } else if( item == “combo” ){  \*p[ name ] == combo( \*p, s );  }  }  return p;  } |

1. [20] Gegeven een integer container a (a is een forward iterator), en de functies
   1. apply( start, end, functie ) die de lambda functie toepast op alle elementen van de start iterator tot de end iterator.
   2. find( start, end, predicate ) die de eerste iterator (van start tot end) teruggeeft waarvoor het predicaat waar is.

Schrijf de stukjes code, gebruikmakend van apply en lambda’s (dus zonder zelf lussen of functies te schrijven). Schrijf de lambda’s met de meest beperkte capture list.

* 1. Zet alle elementen van a op 0
  2. Zet de elementen van a op de waarden 1, 2, 3, 4, …
  3. Zet de waarden van a van element n (inclusief) tot element m (exclusief) op 42 (je mag er van uitgaan dat a de elementen n en m bevat, en dat n <= m)
  4. Tel op hoeveel plekken in a een element kleiner is dan het voorgaande element. Je mag er van uitgaan dat a ten minste 1 element bevat.

[20]

|  |
| --- |
| apply( a.begin(), a.end(), []( int & x ){ x = 0; } );  int c = 0;  apply( a.begin(), a.end(), [&c]( int & x ){ x = ++c; } );  apply(  find( a.begin(), a.end(), [=n]( int & x ){ return x == n;},  find( a.begin(), a.end(), [=m]( int & x ){ return x == m; },  []( int & x ){ x = 42; }  );  int last = \*a.begin();  count = 0;  apply( ++a.begin(), a.end(),  [&last,&count]( int & x ){  if( x < last ){ count++; }  last = x;  }  ); |

Voer je antwoorden in op de laatste bladzijde, dat is de lijst die wordt nagekeken. Je mag antwoorden hier omcirkelen oid., maar dat wordt niet nagekeken.

|  |
| --- |
| 1) Wat is waar als een (C++) programma dat undefined behaviour vertoont? |
| A: Het effect van dat programma kan varieren, afhankelijk van de compiler. B: Het effect van dat programma is niet gedefinieerd door de taal, maar wel door de (documentatie van) de compiler. C: Het effect van dat programma zal niet corect zijn. D: Het effect van dat programma kan alles zijn. |

|  |
| --- |
| 2) Welke lambda verhoogt zijn parameter met 1? |
| A: [](){ ++n; } B: []( int & n ){ ++n; } C: [ &n ]( int &n ){ ++n; } D: [ &n ](){ ++n; } |

|  |
| --- |
| 3) Wat geeft de = in deze lambda aan? |
| [=]( . . . ){ . . . } |
| A: Alle parameters worden by reference doorgegeven. B: Alle parameters worden by value doorgegeven. C: Alle niet-parameters worden by value doorgegeven. D: Alle niet-parameters worden by reference doorgegeven. |

|  |
| --- |
| 4) Als een klant precies heeft aangegeven wat hij wil, hij wil dat allemaal, en hij is verder niet beschikbaar totaan de acceptatie van het product, wat is dan een goede keuze qua ontwikkelmethodiek? |
| A: Waterval en agile methode kunnen beide even goed werken. B: Een waterval-achtige methode. C: Hier is geen enkele methode geschikt voor. D: Scrum, of een andere agile methode. |

|  |
| --- |
| 5) Waar wordt een (niet statische) lokale variabele geallokeerd? |
| A: op de heap, en (door de gebruiker) gedeallokeerd B: op de stack, en automatisch gedeallokeerd C: op de stack, en expliciet (door de gebruiker) gedeallokeerd D: op de heap, en automatisch gedeallokeerd |

|  |
| --- |
| 6) Wat is waar voor iets dat je met 'new vogel[ 100 ]' hebt geallokeert en vervolgens met delete (dus zonder [ ]) vrijgeeft? |
| A: Dit maakt geen verschil met vrijgeven met delete[]. B: Dan krijg je een foutmelding tijdens het 'builden' (tijdens compilen of linken) van je applicatie. C: Dan \*zal\* je programma zich 'raar' gaan gedragen. D: Dan \*kan\* je programma zich 'raar' gaan gedragen. |

|  |
| --- |
| 7) wat is waar? |
| A: Een destructor kan overloaded zijn maar een constructor niet. B: Een constructor geeft een return waarde terug maar een destructor niet. C: Een constructor kan een parameter hebben maar een destructor niet. D: Een destructor kan een parameter hebben maar een constructor niet. |

|  |
| --- |
| 8) Wat is waar over garbage collection en C++? |
| A: Garbage collection is in C++ niet nodig, want smart pointers bereiken precies het zelfde. B: De taal C++ biedt geen garbage collection. C: Het concept van garbage collection is voor C++ niet relevant, omdat het een probleem oplost dat in C++ niet bestaat. D: Garbage collection in C++ is standaard niet ingeschakeld maar kan voor (niet real-time) applicaties worden ingeschakeld. |

|  |
| --- |
| 9) Wat is het doel van de 'regel van drie'? |
| A: Dat een object verantwoordelijk is voor het allokeren en deallokeren van zijn sub-objecten. B: Dat er geen undefined behaviour ontstaat als een object aan zichzelf wordt toegekend. C: Dat de copy constructor geen onnodige copien maakt. D: Dat de destructor ook zorgt voor het destructen van het superklasse deel van een subklasse. |

|  |
| --- |
| 10) Wat is (als het goed is) waar over een unique\_ptr<...>? |
| A: Die wijst naar een object dat maar een keer wordt aangemaakt (een singleton). B: Die wijst gedurende zijn bestaan altijd naar het zelfde opject. C: Die wijst als enige unique\_ptr naar een bepaald object op de heap. D: Die houdt bij hoeveel pointers er naar het geallokeerde object wijzen, en geeft het object vrij zodra dat aantal 0 is. |

|  |
| --- |
| 11) Wat zal dit programma printen? |
| #include <iostream> void PrintSequence( int StopNum ){  int Num = 1;  while( true ){  if( Num >= StopNum ){ throw Num; }  std::cout << Num << " ";  Num++;  } }  int main(){  try {  PrintSequence( 10 )  } catch( int ExNum ){  std::cout << "Caught : " << ExNum;  } } |
| A: 1 2 3 4 5 6 7 8 9 10 B: 1 2 3 4 5 6 7 8 9 Caught : 10 C: 1 2 3 4 5 6 7 8 9  D: 1 2 3 4 5 6 7 8 9 10 Caught : 10 |

|  |
| --- |
| 12) Wat is (in de context van C++ en STL) een iterator? |
| A: Een functie die je kan aanroepen om alle waarden in een container te benaderen. B: Een object dat zich gedraagt als een soort pointer. C: Een taal-constructie waarmee je door de elementen van een array-achtige container loopt. D: Een functie-expressie (een anonieme functie die je declareert op de plek waar je hem meegeeft als parameter). |

|  |
| --- |
| 13) Een array is een voobeeld van een |
| A: random-access iterator B: bidirectional iterator C: forward iterator D: backward iterator |

|  |
| --- |
| 14) Wat beschrijft het Flyweight pattern het beste? |
| A: Doe niet meer dan (op dit moment) nodig is. B: Gemeenschappelijke dingen worden gedeeld. C: Veel gebruikte gegevens moeten bij elkaar staan. D: Vermijdt initializatie van (globale) objecten die nooit gebruikt worden. |

|  |
| --- |
| 15) Het (primaire) doel van het Dirty Flag pattern is |
| A: Code leesbaarder maken B: Sneller runnen C: Geheugen besparen D: Herbruikbaarheid van (delen van) de code |

|  |
| --- |
| 16) Wat voor soort schaal is een jaartal in de bij ons gebruikelijk Gregoriaanse kalender? |
| A: nominale schaal B: interval schaal C: ordinale schaal D: ratio schaal |

|  |
| --- |
| 17) Wat is een voorbeeld van een ratio schaal? |
| A: Een lijst van kleuren, bv. groen, geel, blauw, rood. B: Het saldo op een bankrekening. C: De beoordeling van een werkstuk met de termen onvoldoende, voldoende, goed. D: Een temperatuur in graden Celcius. |

|  |
| --- |
| 18) Wat beschrijft het Command pattern het beste? |
| A: De operator() wordt gebruikt om een object te kunnen gebruiken als functie. B: Een actie wordt omgezet in een object. C: Een std::function object wordt gebruikt om een lambda op te slaan. D: De actie die aan een for\_each loop wordt meegegeven is een lambda. |

|  |
| --- |
| 19) Wat is \*geen\* doel van het Observer pattern? |
| A: De actie die gekoppeld is aan een externe gebeurtenis wijzigen tijdens het runnen. B: Wijzigingen van een waarde doorgeven aan de objecten die daar om gevraagd hebben. C: Voorkomen dat een low-level object afhankelijk is van een high-level object. D: Een reeks acties overnieuw uit te kunnen voeren. |

|  |
| --- |
| 20) Wat biedt het adapter pattern? |
| A: Een andere interface voor de zelfde functionaliteit. B: Run-time selectie van de implementatie van een methode.  C: Andere of meer functionaliteit via de zelfde interface. D: Variaties in de functionaliteit, geselecteerd via parameters van de constructor. |

001 : D   
 002 : B   
 003 : C   
 004 : B   
 005 : B

006 : D   
 007 : C   
 008 : B   
 009 : A   
 010 : C

011 : B   
 012 : B   
 013 : A   
 014 : B   
 015 : B

016 : B   
 017 : B   
 018 : B   
 019 : D   
 020 : A

|  |  |
| --- | --- |
| **Muliple Choice Vraag** | **Antwoord (schrijf duidelijk: A, B, C, of D)** |
| 1. |  |
| 2. |  |
| 3. |  |
| 4. |  |
| 5. |  |
| 6. |  |
| 7. |  |
| 8. |  |
| 9. |  |
| 10. |  |
| 11. |  |
| 12. |  |
| 13. |  |
| 14. |  |
| 15. |  |
| 16. |  |
| 17. |  |
| 18. |  |
| 19. |  |
| 20. |  |

**That’s all folks,  /  ?**