Universiteit van Amsterdam

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Tentamen

C++ programmeermethoden

Bachelor Kunstmatige Intelligentie

2e Deeltentamen
Datum: 27 mei 2019
Tijd: 17.00-19.00

Aantal pagina's: 11 (inclusief voorblad)
Aantal vragen: 5
Maximaal aantal te behalen punten: 10

VOORDAT U BEGINT

- **Wacht** tot u de instructie krijgt het tentamen te openen.
- Controleer of uw versie van het tentamen compleet is.
- Schrijf **uw naam en studentnummer en indien van toepassing versienummer** op **elk vel papier** dat u inlevert en **nummer de pagina's**.
- U dient uw **mobiele telefoon** uit te schakelen en te bewaren in uw jas of tas.
  Uw **jas en tas** moeten onder uw tafel liggen.
- **Toegestane hulpmiddelen**: boek & notities & laptop (alleen voor lezen van ebook!).

HUISHOUDELIJKE MEDEDELINGEN

- De eerste 30 minuten en de laatste 15 minuten mag u de zaal niet verlaten, ook niet voor het bezoeken van het toilet.
- Op verzoek van de examinator (of diens vertegenwoordiger) moet u zich kunnen legitimeren met een bewijs van inschrijving of een geldig legitimatiebewijs.
- Tijdens het tentamen is toiletbezoek niet toegestaan, tenzij de surveillant hier toestemming voor geeft.
- 15 minuten voor het eind wordt u gewaarschuwd dat het inlevertijdstip nadert.
- Vul indien van toepassing na afloop van het tentamen alstublieft het evaluatieformulier in.

Succes!

C++ programmeermethoden Deeltoets 2

Maandag 27 Mei, 17:00-19:00, Roeterseiland Campus C1.04

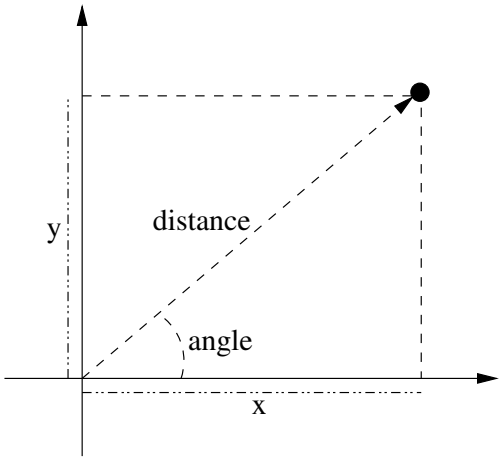
Bas Terwijn

Schakel de Wifi en eventuele andere communicatie-mogelijkheden (bluetooth, telefonie, mobiel internet, etc.) van uw mobiel, laptop, tablet, e-reader, etc. uit.

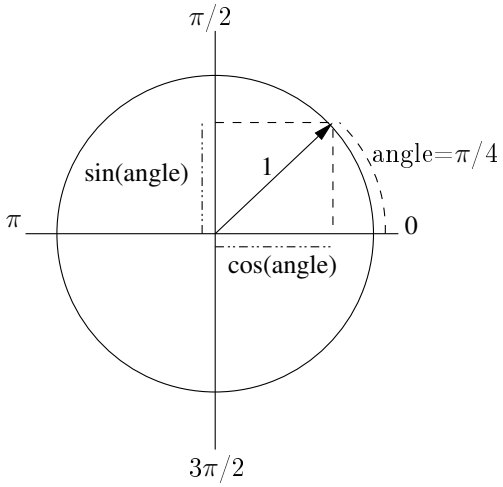
Gebruik uw laptop, tablet of e-reader alleen voor het lezen van uw ebook. Andere windows (bv. een command prompt of IDE) kunnen worden aangezien als fraude zoals beschreven in de "UvA Fraude- en plagiaatsregeling". Fraude kan leiden tot uitsluiting van deelname aan dit vak en in het uiterste geval tot beindiging van de inschijving bij opleidingen van de UvA. Kom dus niet in de verleiding en houd ook de schijn tegen.

Vraag 1 (2 punten)

Een punt in een twee-dimensionaal assenstelsel kan zowel worden gerepresenteerd door een Polar (angle,distance) als door een Cartesian (x,y) coördinaat zoals weergegeven in figuur 1a. De conversie van een Polar coördinaat met distance 1 naar een Cartesian coördinaat kan gebeuren d.m.v. de sinus and cosinus functie zoals weergegeven in figuur 1b.



(a) Punt gerepresenteerd als Polar en Cartesian coördinaat



(b) Eenheidscirkel met conversie van Polar coördinaat met angle=$\pi/4$ en distance=1 naar een Cartesian coördinaat met x=cos(angle) en y=sin(angle)

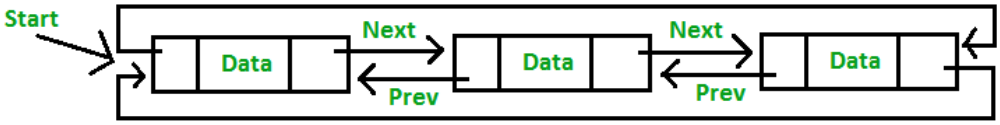
Figuur 1: Polar en Cartesian coördinatenstelsels

Geef van onderstaande programma voor elke regel in de main functie (A t/m D) aan of deze een foutmelding geeft, en zo ja wat de oorzaak daarvan is, en zo niet welke waarde er dan geprint wordt. Neem aan dat regels die leiden tot een foutmelding worden verwijderd zodat de andere regels wel kunnen worden uitgevoerd.

```
#include <iostream>
#include <cmath>
using namespace std;

struct Polar
{
   double angle,distance;
   Polar(double angle,double distance)
      : angle(angle), distance(distance) {}
};

struct Cartesian
{
   double x,y;
   Cartesian(double x,double y) : x(x), y(y) {}

   Cartesian(const Polar& pc) // converting constructor
   {
      x = std::cos(pc.angle) * pc.distance;
      y = std::sin(pc.angle) * pc.distance;
   }

   Cartesian operator+(Cartesian cc2)
   {  return Cartesian( x + cc2.x , y + cc2.y ); }
};

Cartesian operator-(const Cartesian& cc1, const Cartesian& cc2)
{  return Cartesian( cc1.x - cc2.x , cc1.y - cc2.y ); }

ostream& operator<<(ostream& os, const Cartesian& cc)
{  return os<<'('<< cc.x <<','<< cc.y <<")"; }

int main()
{
   cout<<"A:"<<( Cartesian(0,2) + Polar(0,1) )<<'\n';
   cout<<"B:"<<( Cartesian(0,2) - Polar(0,1) )<<'\n';
   cout<<"C:"<<( Polar(0,2) + Cartesian(0,1) )<<'\n';
   cout<<"D:"<<( Polar(0,2) - Cartesian(0,1) )<<'\n';
}
```

Vraag 2 (2 punten)

In het onderstaande programma is een circulaire double-linked list geïmplementeerd. Zie figuur 2 voor een voorbeeld daarvan. Wat is de uitvoer van dit programma?



Figuur 2: Een voorbeeld circulaire double-linked list, iedere node heeft zijn eigen data (i), een next pointer (n), en een previous pointer (p). Nodes verwijzen naar elkaar in een cirkel.

```
#include <iostream>
using namespace std;

struct Node
{
   int i;
   Node *next,*prev;
   Node(int i)                          : i(i), next(this), prev(this) {}
   Node(int i,Node *next,Node *prev) : i(i), next(next), prev(prev) {}

   Node& add(int i)
   {
      Node* n=new Node(i,this,prev);
      prev->next = n;
      prev = n;
      return *this;
   }
};

ostream& operator<<(ostream& os,const Node& n)
{
   const Node *current=&n;
   do
   {
      os<< current->i <<" ";
      current = current->next;
   } while (current != &n);
   return os;
}

int main()
{
   Node* n=new Node(0);
   cout<<"A: "<< *n  <<'\n';
   cout<<"B: "<< n->add(1) <<'\n';
   cout<<"C: "<< n->add(2) <<'\n';
   n = n->prev;
   cout<<"D: "<< n->add(3) <<'\n';
}
```

Vraag 3 (2 punten)

In het onderstaande programma wordt een Sequence met verschillende Number objecten gevuld. Functie 'sum()' wordt vervolgens gebruikt om de som van de Sequence uit te rekenen.

```
#include <iostream>
using namespace std;

struct Number
{
   int i;
   Number()   :i(0) {}
   Number(int i) :i(i) {}
};

Number& operator+=( Number& n1, Number& n2 )
{  n1.i += n2.i; return n1; }
ostream& operator<<( ostream& os, const Number& n )
{ return os<< n.i ; }

struct Sequence
{
   static const int length = 1000;
   Number data[ length ];
   int index;

   Sequence() : index(0) {}
   int get_size() { return index; }
   Number& get_element( int i ) { return data[i]; }

   bool add(Number d)
   {
      if (index < length) { data[ index++ ] = d; return true; }
      return false;
   }
};

Number sum( Sequence& sequence ) // <- ``const'' gets added here
{
   Number s(0);
   for (int i=0; i<sequence.get_size(); ++i)
      s += sequence.get_element(i);
   return s;
}
```

```
int main()
{
   Sequence sequence;
   for (int i=0; i<30; ++i)
      sequence.add(i);
   cout<<" sum: "<< sum(sequence) <<'\n';
}
```

Dit werkt, maar de programmeur wil nu voorkomen dat in de 'sum()' functie de Sequence per ongeluk kan worden aangepast wat bij toekomstige aanpassingen aan sum() tot bugs zou kunnen leiden. De programmeur voegt daarom "const" toe aan de parameter van sum() in bovenstaande programma:

```
Number sum( const Sequence& sequence )
```

Maar doordat sequence nu "const" is moet op andere plaatsen ook "const" worden toegevoegd om foutmeldingen van de compiler te voorkomen. Via sequence moet het immers niet meer mogelijk zijn om direct of indirect aanpassingen te doen. Er moet op nog minimaal 4 plaatsen "const" worden toegevoegd:

- 2x bij een methode
- 1x bij een return type
- 1x bij een parameter

Geef aan op welke plekken "const" moet worden toegevoegd. Schrijf daarbij alleen de (delen van) regels over waarbij je "const" toevoegt om zo veel overschrijfwerk te voorkomen.

Vraag 4 (2 punten)

Geef de uitvoer van onderstaande programma waar 'dynamic binding' en 'slicing' plaatsvindt.

```
#include <iostream>
using namespace std;

struct Base_1
{
   void print() const { cout<<"class Base_1\n"; }
};

class Sub_1 : public Base_1
{
   void print() const { cout<<"class Sub_1\n"; }
};

struct Base_2
{
   virtual void print() const { cout<<"class Base_2 \n"; }
};

class Sub_2 : public Base_2
{
   void print() const { cout<<"class Sub_2\n"; }
};

void function_A(const Base_1 b) { b.print(); }
void function_B(const Base_1* b) { b->print(); }
void function_C(const Base_2 b) { b.print(); }
void function_D(const Base_2* b) { b->print(); }

int main()
{
   const Sub_1 sub1;
   const Sub_2 sub2;
   cout<<"A: "; function_A( sub1 ); cout<<'\n';
   cout<<"B: "; function_B( &sub1 ); cout<<'\n';
   cout<<"C: "; function_C( sub2 ); cout<<'\n';
   cout<<"D: "; function_D( &sub2 ); cout<<'\n';
}
```

Vraag 5 (2 punten)

Beschrijf in je eigen woorden de overeenkomst en het verschil tussen:

- a) een ’struct’ en een ’class’
- b) een ’string’ en een ’char[]’
- c) een ’static array’ en een ’dynamic array’
- d) een ’copy constructor’ en een ’assignment operator’

The End!