Universiteit van Amsterdam

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Tentamen

C++ programmeermethoden

Bachelor Kunstmatige Intelligentie

2e Deeltentamen
Datum: 3 juni 2022
Tijd: 13.00-15.00

Aantal pagina's: 9 (inclusief voorblad)
Aantal vragen: 4
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/slides/notities).

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

Vrijdag 3 Juni, 13:00-15:00, Roeterseilandcampus 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 (2.5 punten)

Onderstaande code is van een spel waar de speler de finish moet zien te bereiken, maar de code-kwaliteit zou beter zijn al er meer abstractie wordt gebruikt.

```
bool contains(int x,int y,int rx1,int ry1,int rx2,int ry2) {
   if (x < rx1)
      return false;
   if (y < ry1)
      return false;
   if (x > rx2)
      return false;
   if (y > ry2)
      return false;
   return true;
}

int main() {
   int player_x = 150;
   int player_y = 160;

   int finish_x1 = 0;
   int finish_y1 = 0;
   int finish_x2 = 100;
   int finish_y2 = 100;

   while ( !contains( player_x, player_y, finish_x1, finish_y1,
      finish_x2, finish_y2) ) {
      // here the player position gets updated (you dont have to
         rewrite this part):
      // player_x += ...
      // player_y += ...
   }
}
```

- **a)** Herschrijf deze code en voeg abstractie toe door met minimaal 2 verschillende classes of structs het grote aantal losse integers te vervangen.
- **b)** Gebruik hierbij ook een “bool operator<(...) ” functie, voorkom code duplicatie, en zorg voor goede leesbaarheid.
- **c)** Leg in het algemeen uit waarom abstractie de kwaliteit van code kan verbeteren.

Vraag 2 (2.5 punten)

Onderstaande code gebruikt “raw for loops” terwijl de C++ Core Guidelines adviseert om in plaats daarvan zoveel mogelijke STL Algorithm functies te gebruiken.

```
#include ...
using namespace std;

double function1(vector<double>& vec) {
   double value = vec[0];
   for (int i=1; i<vec.size(); i++)
      if (vec[i] > value)
         value = vec[i];
   return value;
}

bool function2(vector<double>& vec) {
   double value = 0;
   for (int i=0; i<vec.size(); i++)
      value += vec[i];
   return value/vec.size() >= 5.5;
}

vector<int> function3(vector<int>& vec1,
                      vector<int>& vec2)
{
   vector<int> result;
   for (auto d1 : vec1) {
      bool add = true;
      for (auto d2 : vec2) {
         if (d1 == d2) {
            add = false;
            break;
         }
      }
      if (add)
         result.push_back(d1);
   }
   return result;
}
```

- **a)** Herschrijf deze code en vervang de loops met STL Algorithm functies. Gebruik hierbij de onderstaande STL Algorithm voorbeeld-code ter vervanging van documentatie.
- **b)** Leg in het algemeen uit waarom STL Algorithm functies de kwaliteit van code kunnen verbeteren maar soms ook de kwaliteit kunnen verslechteren. Geef eventueel een voorbeeld van verslechteren ter verduidelijking.

```
void stl_algorithm_examples()
{
   vector<int> v{1,2,3,4};

   int n=3;
   auto iter_find = find(begin(v), end(v), n);
   if (iter_find != v.end())
      cout<<"element "<<n<<" found\n"; // element 3 found

   auto iter_min = min_element(begin(v), end(v));
   auto iter_max = max_element(begin(v), end(v));
   cout<< *iter_min <<" "<< *iter_max <<'\n'; // 1 4

   vector<int> v2{5,6,3,4};
   // the collections need to be sorted before using
   // set_intersection() set_difference()
   sort( begin(v), end(v) );
   sort( begin(v2), end(v2) );

   vector<int> result;
   set_intersection( begin(v), end(v), begin(v2), end(v2),
      back_inserter(result) );
   cout<< result <<'\n'; // 3 4

   result.clear(); // empties the vector
   set_difference( begin(v), end(v), begin(v2), end(v2),
      back_inserter(result) );
   cout<< result <<'\n'; // 1 2

   cout<< accumulate(begin(v), end(v), 0.0) <<'\n'; // 10
}
```

Vraag 3 (2 punten)

Onderstaande code gebruikt een “Logger” class om een recept voor cake te streamen met indentatie door gebruik van het ’-’ teken.

```
#include <iostream>

class Logger {
   ostream& stream;
   int indent=0;
   void make_indent() { for (int i=0; i<indent; i++) stream<<'-'; }
public:
   Logger(ostream& s) : stream{s} {}
   void log(string s) { make_indent(); stream<< s <<'\n'; }
   void change_indentation(int i) { indent+=i; }
};

int main() {
   Logger log{cout};
   log.log("Make Cake");
   {
      log.change_indentation(2);
      log.log("Stir Ingredients");
      {
         log.change_indentation(3);
         log.log("baking soda");
         log.log("butter");
         log.change_indentation(-3); // don't forget!
      }
      log.change_indentation(-2); // don't forget!
   }
   {
      log.change_indentation(2);
      log.log("Bake Ingredients");
      {
         log.change_indentation(3);
         log.log("heat oven to 350C");
         log.log("add pan and set timer to 20min");
         log.change_indentation(-3); // don't forget!
      }
      log.change_indentation(-2); // don't forget!
   }
}
```

Deze code resulteert in de volgende output:

```
Make Cake
--Stir Ingredients
-----baking soda
-----butter
--Bake Ingredients
-----heat oven to 350C
-----add pan and set timer to 20min
```

Een probleem met deze code is dat we na indenteren met bv "`log.change_indentation(3);`" we later niet moeten vergeten deze indentatie weer terug te draaien met "`log.change_indentation(-3);`".

- **a)** Schrijf class "Indent" die door gebruik van het RAII principe (dus met gebruik van een destructor die automatisch wordt aangeroepen als een object out-of-scope gaat) zorgt dat de identatie vanzelf terug wordt gedraaid, zodat we niet meer de negatieve indentatie-veranderingen hoeven te schrijven. Deze class "Indent" moet met onderstaande code dezelfde output geven als hierboven:

```
int main()
{
   Logger log{cout};
   log.log("Make Cake");
   {
      Indent i(log, 2);
      log.log("Stir Ingredients");
      {
         Indent i(log, 3);
         log.log("baking soda");
         log.log("butter");
      }
   }
   {
      Indent i(log, 2);
      log.log("Bake Ingredients");
      {
         Indent i(log, 3);
         log.log("heat oven to 350C");
         log.log("add pan and set timer to 20min");
      }
   }
}
```

- **b)** Beschrijf een ander creatief voorbeeld waar RAII nuttig kan zijn, anders dan de standaard toepassing van vrijgeven van geheugen, files, locks of connections.

Vraag 4 (3 punten)

- **a)** Beschrijf de relatie tussen keywords **friend** en **private**.
- **b)** Beschrijf het effect van het "overriden" van een functie welke wel en niet **virtual** is.
- **C)** Beschrijf een punt wat je interessant/leerzaam/vreemd vond in de video: Kate Gregory "Simplicity: Not Just For Beginners" CppCon 2018.
- **d)** Beschrijf het effect van elk van de 3 **const** in onderstaande code:

```
class Some_Class
{
public:
   const int& get_some_value(const vector<int>& data) const
   {
      // ...
   }
}
```

The End!