Inleiding Programmeren

Examen

1B Informatica,
Academiejaar 2018-2019

Toon Calders Tom Hofkens Stephen Pauwels Len Feremans

25 januari 2019

1 Belangrijke informatie

- Maak deze opdracht individueel. Het gebruik van communicatiemiddelen zoals smartphones, smart watches en internet is niet toegestaan, met uitzondering van Blackboard.
- Het examen begint om 9u. Ten laatste na 30 minuten geef je het theorie examen C++ af. Daarna mag je starten met de programmeeropdracht.
- Als je vrijgesteld bent van Python, krijg je tijd tot 11:30 voor praktijk C++.
- Als je niet vrijgesteld bent voor Python, krijg je tot **ten laatste 11:30** om jouw Python opdracht uit te voeren. Daarna heb je tot 13:00 voor het praktijk examen C++.
- Duid in jouw oplossing duidelijk aan waar welke vraag wordt beantwoord. Dit kan bijvoorbeeld door in de code commentaren te voorzien zoals:
  `// Volgende functie is het antwoord op vraag 1.`
- Voorzie elk bestand dat je aanmaakt of aanpast van een hoofding die jouw naam en studentnummer bevat.
- Je mag gebruik maken van een USB stick met documentatie en ander lesmateriaal die je eventueel bij je hebt. Ook via Blackboard wordt alle lesmateriaal en bijkomende C++ en Python documentatie beschikbaar gesteld.
- Dien je praktijk-oplossing in via Blackboard.
- **Controleer jouw inzending via Blackboard nauwgezet. Werden van alle bestanden die je maakte, de laatste versies toegevoegd ?**
- Indien je bijzondere faciliteiten hebt voor dit examen, breng dan de surveillant op voorhand op de hoogte.

2 Theoretisch examen

1. (5pt) Beschouw volgende C++ code. Geef schematisch de inhoud van het geheugen weer op de gemarkeerde punten in het programma. Er zijn 3 gemarkeerde punten (PUNT1, PUNT2, PUNT3); jouw antwoord omvat dus 3 schema's, 1 per punt. Zorg ervoor dat elk schema:

   (a) volledig is; de inhoud van alle variabelen en alle aangemaakte objecten zijn aanwezig; van elk object is het type gespecificeerd;

   (b) duidelijk van elke bezette geheugenplaats weergeeft of die zich op de stack met lokale variabelen of in het vrije geheugen bevindt;

   (c) voor elke pointer weergeeft of hij `nullptr` is en indien niet, naar welke geheugenplaats die verwijst.

```
1  struct Node {
2    Node() {}
3    Node(Node* n, int c): next(n), content(c) {}
4    ~Node() { delete next; }
5
6    Node* clone() {
7      if (next) return new Node(next->clone(),content);
8      else return new Node(nullptr,content);
9    }
10
11   int content;
12   Node* next=nullptr;
13 };
14
15 class List {
16 public:
17   List(){ root=nullptr; }
18   ~List() { if (root) delete root; }
19
20   void insert (int i) { root=new Node(root,i); }
21
22   List clone() {
23     List L;
24     if (root) L.root=root->clone();
25     return L;
26   }
27
28 private:
29   Node* root;
30 };
31
32
33 int main() {
34   List L1;
35   L1.insert(1); L1.insert(2); L1.insert(3);
36     // PUNT 1
37   List L2=L1.clone();
38   L2.insert(4); L2.insert(5); L2.insert(6);
39     // PUNT 2
40   List& L3=L2;
41   L3.insert(7);
42     // PUNT 3
43   return 0;
44 }
```

2. (3pt) Bekijk volgende definities van de klassen `Class_A`, `Class_B` en `Class_C`. Welke instructies van lijn 37 tot en met lijn 58 zijn niet toegestaan in dit fragment?

```
1  class Class_A {
2  public:
3    int ma;
4
5    int& get_mb() {
6      return mb;
7    }
8
9  protected:
10   int& get_ma() { return ma; }
11
12 private:
13   int mb;
14 };
15
16 class Class_B: protected Class_A {
17 public:
18   int value_of_ma() {
19     return get_ma();
20   }
21
22   int value_of_mb() {
23     return get_mb();
24   }
25 };
26
27 class Class_C: private Class_B {
28 public:
29   int get_both(int& x, int& y) {
30     x=get_ma();
31     y=value_of_mb();
32   }
33 };
34
35 int main() {
36   Class_A a; Class_B b; Class_C c;
37
38   a.ma=3;
39   a.mb=5;
40   a.get_ma();
41   a.get_mb();
42
43   b.ma=3;
44   b.mb=5;
45   b.get_ma();
46   b.get_mb();
47   b.value_of_ma();
48   b.value_of_mb();
49
50   c.ma=3;
51   c.mb=5;
52   c.get_ma();
53   c.get_mb();
54   c.value_of_ma();
55   c.value_of_mb();
56   int x,y; c.get_both(x,y);
57   return 0;
58 }
```

3. (2pt) Geef een minimaal voorbeeld (programmaatje) waarbij het keyword `virtual` gebruikt wordt en waarbij de aanwezigheid van dit keyword een invloed heeft op de uitvoer van het programma. Dit wil zeggen: geef enkele lijnen code die syntactisch correct zijn en waarin het keyword `virtual` gebruikt wordt. Als je het keyword `virtual` weglaat, moet het programma nog steeds syntactisch correct zijn, maar de uitvoer verandert.

1								6
		6		2		7		
7	8	9	4	5		1		3
			8		7			4
				3				
	9				4	2		1
3	1	2	9	7			4	
	4			1	2		7	8
9		8						

Figuur 1: Voorbeeld van een Sudoku puzzel. 9x9 vakjes zijn onderverdeeld in 3x3 blokken van telkens 3x3 vakjes. De lege vakjes in de puzzel dienen zodanig ingevuld te worden met de getallen 1 tot en met 9 zodat er nooit twee maal hetzelfde cijfer voorkomt in een kolom, rij, of blok.

3 Praktische Opdracht

3.1 Inleiding

De opdrachten gaat over Sudoku's. Wikipedia definiëert een Sudoku als volgt:

> Een sudoku is een puzzel bestaande uit negen bij negen vakjes die gegroepeerd zijn als negen blokken van drie bij drie vakjes. In de vakjes moeten de cijfers 1 tot en met 9 ingevuld worden op zo'n manier dat in elke horizontale lijn en in elke verticale kolom en in elk van de negen blokjes de cijfers 1 tot en met 9 één keer voorkomen.

In figuur 1 is een voorbeeld gegeven van een sudoku. De bedoeling van een Sudoku is om de puzzel verder aan te vullen met de cijfers 1-9 zonder de spelregel dat er geen twee maal dezelfde waarde in een rij, kolom, of blok mar staan, te breken. In het voorbeeld is een van de cellen in het rood gemarkeerd (middelste cel van het blok links bovenaan). De rij van deze cel bevat reeds 2, 6 en 7; de kolom van deze cel bevat 1, 4, 8 en 9; het blok van deze cel bevat 1, 6, 7, 8 en 9. Deze cijfers zijn dus uitgesloten voor deze cel. De enige mogelijkheden die overblijven voor deze cel zijn dus 3 en 5. Bij het oplossen van de puzzel kunnen we op dit moment dus nog niet bepalen wat de waarde moet zijn van deze cel. Voor de blauwe cel in ons voorbeeld (rechts onder in het middelste blok bovenaan), echter, is de enige mogelijke invulling een 6, omdat alle andere cijfers reeds voorkomen in de rij, kolom, of blok van de blauwe cel. Bij het oplossen van de Sudoku puzzel kunnen we hier dus een 6 invullen. Daarna gaan we op zoek naar een andere cel die een unieke invulling heeft tot de hele puzzel opgelost is, of we vastlopen.

4 Python (Herkansing)

Maak dit onderdeel **enkel indien je nog geen vrijstelling hebt voor Python**.

4.1 Bestanden

Bij de opdracht heb je slechts 1 bestand gekregen: `sudoku.py`. Dit is het vertrekpunt van jouw examen.

4.2 Debugging (3pt)

Bekijk het bestand `sudoku.py`. Het bestand bevat een hulpfunctie `print_sudoku` die een sudoku als input krijgt en deze vervolgens mooi uitprint. Een sudoku wordt voorgesteld door een 2-dimensionale lijst, waarbij `None` staat voor een niet ingevulde waarde.

Vervolgens zie je 3 functies: `void maak_sudoku()`, `bool dubbels(rij)` en `bool geldig(sudoku)`. Deze drie functies bevatten 1 of meerdere fouten die je in deze opdracht moet corrigeren. De bedoeling van de functie `maak_sudoku` is om een nieuwe sudoku correct te initialiseren met enkel lege velden. De functie `geldig` kijkt dan weer na of een (gedeeltelijk ingevulde) Sudoku correct is ingevuld.

Jouw opdracht is nu: het bestand wordt uitgevoerd zonder problemen, maar als je de uitvoer bekijkt zie je enkele onverwachte resultaten. Ga na wat het probleem is en corrigeer de code.

4.3 Bestaande code uitbreiden (3pt)

Momenteel bevat de functie `geldig` enkel code om na te gaan of er geen duplicaten zijn in kolommen of rijen, maar de 9 blokjes zelf worden nog niet gecontroleerd. Breid de code uit zodat ook de verschillende blokken op duplicaten gecontroleerd worden.

```
s2 = [[1,    None, None, None, None, None, None, None, 6],
      [None, 1,    6,    None, 2,    None, 7,    None, None],
      [7,    8,    9,    4,    5,    None, 1,    None, 3],
      [None, None, None, 8,    None, 7,    None, None, 4],
      [None, None, None, None, 3,    None, None, None, None],
      [None, 9,    None, None, None, 4,    2,    None, 1],
      [3,    None, 2,    9,    7,    None, None, 4,    None],
      [None, 4,    None, None, 1,    2,    None, 7,    8],
      [9,    None, 8,    None, None, None, None, None, None]]
print_sudoku(s2)
print(geldig(s2))
# geldig(s2) geeft foutief "true" het blokje links bovenaan(vierkantje
# opgespannen door[0][0] en[2][2]) bevat twee maal het cijfer 1
```

4.4 Nieuwe functionaliteit (12pt)

In dit onderdeel ga je zelf nieuwe functies moeten toevoegen.

4.4.1 Mogelijkheden opsommen (5p)

Voeg een functie `mogelijk` toe die een Sudoku en een positie als input krijgt en vervolgens een lijst teruggeeft die alle getallen van 1 t.e.m. 9 bevat die voor die positie mogelijk zijn volgens de regels van Sudoku. Dit wil zeggen: alle getallen 1-9 die nog niet voorkomen in de rij, kolom, of blok van de positie.

Enkele voorbeelden:

```
s3 = [[1,    None, None, None, None, None, None, None, 6],
      [None, None, 6,    None, 2,    None, 7,    None, None],
      [7,    8,    9,    4,    5,    None, 1,    None, 3],
      [None, None, None, 8,    None, 7,    None, None, 4],
      [None, None, None, None, 3,    None, None, None, None],
      [None, 9,    None, None, None, 4,    2,    None, 1],
      [3,    1,    2,    9,    7,    None, None, 4,    None],
      [None, 4,    None, None, 1,    2,    None, 7,    8],
      [9,    None, 8,    None, None, None, None, None, None]]

print(mogelijk(s3, 1, 1)) # Geeft: [3 5]
print(mogelijk(s3, 4, 5)) # Geeft: [1 5 6 9]
print(mogelijk(s3, 8, 8)) # Geeft: [2 5]
```

De (inefficiënte) brute-force methode die voor een positie alle mogelijkheden afloopt en `geldig` aanroept, geeft slechts de helft van de punten op dit onderdeel. M.a.w.; om op dit deel alle punten te halen, moet je een correcte versie van `mogelijk` implementeren die `geldig` **niet** aanroept.

4.4.2 Sudoku oplossen (7p)

Maak een functie `vul_aan` die een Sudoku die als input gegeven wordt, verder aanvult door telkens elke lege positie in te vullen waarvoor er maar 1 mogelijkheid bestaat. `vul_aan` blijft dit doen totdat ofwel de Sudoku volledig ingevuld is, ofwel er voor elke lege cel nog meerdere mogelijkheden zijn. Een voorbeeld: de volgende code:

```
s4 = [[None, None, None, None, None, None, None, None, 6],
      [None, None, 6,    None, 2,    None, 7,    None, None],
      [7,    8,    9,    4,    5,    None, 1,    None, 3],
      [None, None, None, 8,    None, 7,    None, None, 4],
      [None, None, None, None, 3,    None, None, None, None],
      [None, 9,    None, None, None, 4,    2,    None, 1],
      [3,    1,    2,    9,    7,    None, None, 4,    None],
      [None, 4,    None, None, 1,    2,    None, 7,    8],
      [9,    None, 8,    None, None, None, None, None, None]]
print_sudoku(s4)
vul_aan(s4)
```

geeft de aangevulde Sudoku op de volgende pagina:

```
+-----------------+-----------------+-----------------+
|  _     _     _  |  7     8     9  |  4     5     6  |
|                 |                 |                 |
|  4     5     6  |  1     2     3  |  7     8     9  |
|                 |                 |                 |
|  7     8     9  |  4     5     6  |  1     2     3  |
+-----------------+-----------------+-----------------+
|  _     _     _  |  8     9     7  |  5     6     4  |
|                 |                 |                 |
|  5     6     4  |  2     3     1  |  8     9     7  |
|                 |                 |                 |
|  8     9     7  |  5     6     4  |  2     3     1  |
+-----------------+-----------------+-----------------+
|  3     1     2  |  9     7     8  |  6     4     5  |
|                 |                 |                 |
|  6     4     5  |  3     1     2  |  9     7     8  |
|                 |                 |                 |
|  9     7     8  |  6     4     5  |  3     1     2  |
+-----------------+-----------------+-----------------+
```

4.5 Programmeerstijl (2pt)

Naast de opdrachten staan er ook 2 punten op programmeerstijl; dit houdt onder andere in: correct gebruik van globale variabelen, duidelijke, leesbare en goed gedocumenteerde code, gestructureerd werken zoals doordacht gebruik maken van functies waar nodig en in mindere mate efficiëntie van de code. Vermijd onnodige duplicatie van code.

5 C++

5.1 Bestanden

Bij de opdracht heb je slechts 1 bestand gekregen: `frame.cpp`. Dit is het vertrekpunt van jouw examen.

1. Maak een nieuw project aan.
2. Open bijgevoegde `frame.cpp` en kopieer de inhoud naar de `main.cpp` van je nieuwe project.

5.2 Debugging (3pt)

Bekijk het bestand `frame.cpp`. Het bestand start met de definitie van de data structuur die gebruikt wordt om een Sudoku op te slaan:

```
typedef vector<vector<int>> Sudoku; // Een Sudoku is een 9x9 matrix die we
                                    // voorstellen als een vector van vectoren
const int LEEG=0;                   // We gebruiken constante "LEEG" om een cel
                                    // aan te duiden waar nog geen waarde in staat
```

Daarna volgt de declaratie van twee hulpfuncties.

```
ostream& operator<<(ostream& s, const vector<int> &v);    // Print een vector
                                                          // uit via cout << v;
void print(Sudoku s);                                     // Nette weergave van
                                                          // een sudoku
```

Vervolgens zie je 3 functies: `void init(Sudoku s)`, `bool dubbels(vector<int> v)` en `bool geldig(Sudoku s)`. Deze drie functies bevatten 1 of meerdere fouten die je in deze opdracht moet corrigeren. De bedoeling van de functie `init` is om een Sudoku correct te initialiseren met enkel lege velden. De functie `geldig` kijkt dan weer na of een (gedeeltelijk ingevulde) Sudoku correct is ingevuld.

Jouw opdracht is nu: het bestand compileert zonder problemen, maar als je het uitvoert krijg je een foutmelding. Ga na wat het probleem is en corrigeer de code.

5.3 Bestaande code uitbreiden (2pt)

Momenteel bevat de functie `geldig` enkel code om na te gaan of er geen duplicaten zijn in kolommen of rijen, maar de 9 blokjes zelf worden nog niet gecontroleerd. Breid de code uit zodat ook de verschillende blokken op duplicaten gecontroleerd worden.

```
Sudoku s2={{1,     LEEG,  LEEG,  LEEG,  LEEG,  LEEG,  LEEG,  LEEG,  6},
           {LEEG,  1,     6,     LEEG,  2,     LEEG,  7,     LEEG,  LEEG},
           {7,     8,     9,     4,     5,     LEEG,  1,     LEEG,  3},
           {LEEG,  LEEG,  LEEG,  8,     LEEG,  7,     LEEG,  LEEG,  4},
           {LEEG,  LEEG,  LEEG,  LEEG,  3,     LEEG,  LEEG,  LEEG,  LEEG},
           {LEEG,  9,     LEEG,  LEEG,  LEEG,  4,     2,     LEEG,  1},
           {3,     LEEG,  2,     9,     7,     LEEG,  LEEG,  4,     LEEG},
           {LEEG,  4,     LEEG,  LEEG,  1,     2,     LEEG,  7,     8},
           {9,     LEEG,  8,     LEEG,  LEEG,  LEEG,  LEEG,  LEEG,  LEEG}};
// geldig(s2) geeft foutief "true"; het blokje links bovenaan (vierkantje
// opgespannen door [0][0] en [2][2]) bevat twee maal het cijfer 1
```

5.4 Nieuwe functionaliteit (10pt)

In dit onderdeel ga je zelf nieuwe functies moeten toevoegen.

5.4.1 Mogelijkheden opsommen (4p)

Voeg een functie `mogelijk` toe die een Sudoku en een positie als input krijgt en vervolgens een vector teruggeeft die alle getallen van 1 t.e.m. 9 bevat die voor die positie mogelijk zijn volgens de regels van Sudoku. Dit wil zeggen: alle getallen 1-9 die nog niet voorkomen in de rij, kolom, of blok van de positie.

Enkele voorbeelden:

```
Sudoku s3={{1,    LEEG,  LEEG,  LEEG,  LEEG,  LEEG,  LEEG,  LEEG,  6},
           {LEEG, LEEG,  6,     LEEG,  2,     LEEG,  7,     LEEG,  LEEG},
           {7,    8,     9,     4,     5,     LEEG,  1,     LEEG,  3},
           {LEEG, LEEG,  LEEG,  8,     LEEG,  7,     LEEG,  LEEG,  4},
           {LEEG, LEEG,  LEEG,  LEEG,  3,     LEEG,  LEEG,  LEEG,  LEEG},
           {LEEG, 9,     LEEG,  LEEG,  LEEG,  4,     2,     LEEG,  1},
           {3,    1,     2,     9,     7,     LEEG,  LEEG,  4,     LEEG},
           {LEEG, 4,     LEEG,  LEEG,  1,     2,     LEEG,  7,     8},
           {9,    LEEG,  8,     LEEG,  LEEG,  LEEG,  LEEG,  LEEG,  LEEG}};
// cout << mogelijk(s3,1,1) << endl;  // Geeft: [ 3 5 ]
// cout << mogelijk(s3,4,5) << endl;  // Geeft: [ 1 5 6 9 ]
// cout << mogelijk(s3,8,8) << endl;  // Geeft: [ 2 5 ]
```

De (inefficiënte) brute-force methode die voor een positie alle mogelijkheden afloopt en `geldig` aanroept, geeft slechts de helft van de punten op dit onderdeel. M.a.w.; om op dit deel alle punten te halen, moet je een correcte versie van `mogelijk` implementeren die `geldig` **niet** aanroept.

5.4.2 Sudoku oplossen (6p)

Maak een functie `vul_aan` die een Sudoku die als input gegeven wordt, verder aanvult door telkens elke lege positie in te vullen waarvoor er maar 1 mogelijkheid bestaat. `vul_aan` blijft dit doen totdat ofwel de Sudoku volledig ingevuld is, ofwel er voor elke lege cel nog meerdere mogelijkheden zijn. Een voorbeeld: de volgende code:

```
Sudoku s4={{LEEG,  LEEG,  LEEG,  LEEG,  LEEG,  LEEG,  LEEG,  LEEG,  6},
           {LEEG,  LEEG,  6,     LEEG,  2,     LEEG,  7,     LEEG,  LEEG},
           {7,     8,     9,     4,     5,     LEEG,  1,     LEEG,  3},
           {LEEG,  LEEG,  LEEG,  8,     LEEG,  7,     LEEG,  LEEG,  4},
           {LEEG,  LEEG,  LEEG,  LEEG,  3,     LEEG,  LEEG,  LEEG,  LEEG},
           {LEEG,  9,     LEEG,  LEEG,  LEEG,  4,     2,     LEEG,  1},
           {3,     1,     2,     9,     7,     LEEG,  LEEG,  4,     LEEG},
           {LEEG,  4,     LEEG,  LEEG,  1,     2,     LEEG,  7,     8},
           {9,     LEEG,  8,     LEEG,  LEEG,  LEEG,  LEEG,  LEEG,  LEEG}};
vul_aan(s4);
print(s4);
```

geeft de aangevulde Sudoku op de volgende pagina:

```
+-----------------+-----------------+-----------------+
|  _     _     _  |  7     8     9  |  4     5     6  |
|                 |                 |                 |
|  4     5     6  |  1     2     3  |  7     8     9  |
|                 |                 |                 |
|  7     8     9  |  4     5     6  |  1     2     3  |
+-----------------+-----------------+-----------------+
|  _     _     _  |  8     9     7  |  5     6     4  |
|                 |                 |                 |
|  5     6     4  |  2     3     1  |  8     9     7  |
|                 |                 |                 |
|  8     9     7  |  5     6     4  |  2     3     1  |
+-----------------+-----------------+-----------------+
|  3     1     2  |  9     7     8  |  6     4     5  |
|                 |                 |                 |
|  6     4     5  |  3     1     2  |  9     7     8  |
|                 |                 |                 |
|  9     7     8  |  6     4     5  |  3     1     2  |
+-----------------+-----------------+-----------------+
```

5.5 Een oplossing met klasse! (3pt)

Reorganiseer jouw code door gebruik te maken van klassen. Maak een klasse Sudoku, voeg data en methods toe en herschrijf de code zodat die gebruik maakt van de nieuwe klasse.

5.6 Programmeerstijl (2pt)

Naast de opdrachten staan er ook 2 punten op programmeerstijl; dit houdt onder andere in: correct gebruik van globale variabelen, duidelijke, leesbare en goed gedocumenteerde code, gestructureerd werken zoals doordacht gebruik maken van functies waar nodig, *const correctness*, *encapsulatie* en *data hiding*, en in mindere mate efficiëntie van de code. Vermijd onnodige duplicatie van code.