Bachelor thesis
Computer Science

Radboud University

Determining the Neo-Piagetian reasoning level of novice programmers

Author:
Rick Lukassen
s4263812

First supervisor:
Prof. dr. Erik Barendsen
e.barendsen@cs.ru.nl

Second supervisor:
dr. Sjaak Smetsers
s.smetsers@science.ru.nl

April 16, 2018

Abstract

Research has shown that one of the reasons novice programmers struggle is a difference in reasoning level between student and teacher. Previous research has proposed a way to determine a students reasoning level using written tests and presented an exercise catalogue. In this paper this categorization methods' reliability is tested in an introductory programming course. In order to find the reliability of the test, it is compared to the results of think-aloud sessions. For these sessions, a method to classify the students was created which purpose was to make the process more objective. This method will also be covered in this paper. Using these methods the reliability of the test was 50% which increases to 80% if we accept an error rate of half a stage.

Contents

1 Introduction **2**
- 1.1 Problem statement . . . 3
- 1.2 Neo-Piagetian stages . . . 3
- 1.3 Stage evaluation . . . 5

2 Methods **7**
- 2.1 The participants . . . 7
- 2.2 Written test . . . 7
  - 2.2.1 Questions . . . 8
  - 2.2.2 Classifying . . . 13
- 2.3 Think-aloud sessions . . . 14
  - 2.3.1 Questions . . . 15
  - 2.3.2 Observable behaviour . . . 17
  - 2.3.3 Classifying . . . 20
  - 2.3.4 Question 1 . . . 21

3 Results **27**
- 3.1 Written test . . . 27
- 3.2 Think-aloud sessions . . . 29
- 3.3 Comparing written test to think-aloud sessions . . . 40
  - 3.3.1 Total classification . . . 40
  - 3.3.2 Variable assignment . . . 41
  - 3.3.3 If and else statements . . . 41

4 Conclusion and Discussion **43**

References **45**

A Appendix **46**
- A.1 Questions written test, Dutch . . . 46
- A.2 Questions used in think-aloud sessions, Dutch . . . 52
- A.3 Transcriptions, Dutch . . . 57

Chapter 1

Introduction

Many novice programmers struggle during their first exposure to programming. A multinational study (McCracken et al., 2001) has concluded that programming language and programming paradigms have little to no effect on the learning succes of novice programmers. If we take into account that programming languages and paradigm have little effect, the issue must lie somewhere else. Lister (2011) has studied the role of abstraction in programming. He found that the abstract reasoning capabilities of novice programmers were fundamentally different from expert programmers.

When these findings are translated to programming education, he concluded that problems arise when instructors explain and guide students. The instructors use a language that represents an expert level of abstraction, however, many students are not yet able to follow the discussion of programming this way because they can not yet follow this level of abstraction. Lister (2011) has proposed a model based on the neo-Piagetian theory. This model describes the various levels of reasoning a novice programmer could traverse. These levels of reasoning have specific characteristics exhibited by students.

Teague, Corney, Ahadi, and Lister (2013) have already done research on categorizing students according to this this model. Teague used think-aloud sessions to determine the reasoning levels of the students. Teague (2015) found empirical evidence supporting Lister's model.

Even though Lister proposed a model and Teague found empirical evidence back this model, they did not formulate any way to incorporate those findings into a classroom setting. Kutscha (2017) proposed a categorization method and an exercise catalogue to enable instructors to incorporate these findings into their classes. His categorization method was based on written tests. Kutscha applied his exercise catalogue and categorization in a more advanced programming course.

The catalogue consists of mostly tracing exercises and "explain in English" exercises. Lopez, Whalley, Robbins, and Lister (2008) have found

a correlation between tracing tasks and code writing, and explaining tasks and code writing. So by determining the reasoning level of students also tells something about their ability to write code themselves.

1.1 Problem statement

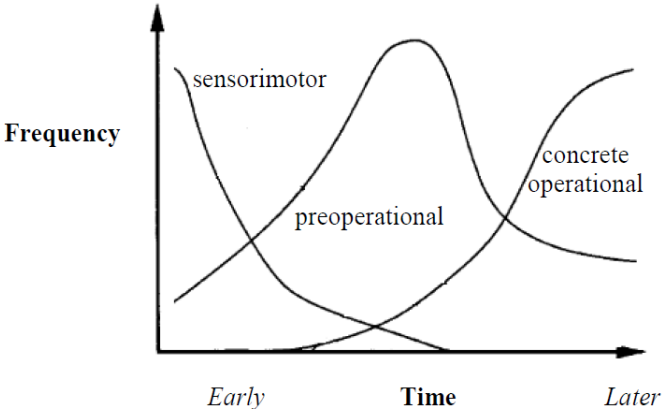
In this paper the categorization method from Kutscha will be revisited and exercises from the catalogue proposed by Kutscha (2017) will be used. These exercises will be applied in an introductory programming course. This will be done in order to validate the method proposed by Kutscha and answer the following question: to what extent it is possible to determine the reasoning level of a student using written tests? In order to answer this question the results from written tests will be compared to the results from thinking-aloud sessions.

1.2 Neo-Piagetian stages

In this section the concepts of Lister's model are explained in more detail.

Overlapping waves

Like neo-Piagetians, Lister does not classify a novice programmer as being at a unique stage at any given moment. He accepts the so-called "overlapping waves model". Within this model, a person exhibits a certain mix of the Piagetian stages. Despite multiple levels of reasoning coexisting, a novice programmer predominantly reasons at a certain level. The image below illustrates this model.



An example to clarify the concept of overlapping waves would be the following. When a person commences to learning a new skill, s/he will reason predominantly at the sensorimotor stage, however as the person progresses in learning this skill s/he will reason less at the sensorimotor stage and more at the pre-operational stage for certain aspects of the skill. While

trying to master a new aspect of a skill, the student might be at the pre-operational level for previously studied aspects, but when reasoning about the new aspect s/he will reason at the sensorimotor level. So multiple stages of reasoning exist at a certain moment, however one of those stages is most dominant. For the purpose of this paper, the student will be classified to be at a certain stage compliant with the most obvious reasoning level.

Tracing and reasoning

Lister uses the skills tracing and reasoning as the core skills to determine the stage the novice is in. Tracing code can be defined as the ability to manually executing the code (in your mind). Reasoning about code is about the ability to abstract from a piece of code. The difference will be explained using the following piece of code.

```
c = a;
a = b;
b = c;
```

Given this code and start state [a = 1, b = 5, c = 100] a student who is able to trace the code could give the correct end state [a = 5, b = 1, temp = 1]. This would be achieved by entering these numbers into the variables and then manually executing each line of code. If the student *can not* reason about the code, s/he would not be able to answer the question "what does the code do on a more abstract level?" correctly. If the student *can* reason about the code, the student could give us the correct answer (swapping values a and b). This piece of code will be used to clarify the different stages.

Sensorimotor

According to Lister (2016) a novice programmer in the sensorimotor stage has an incoherent understanding of program execution. That means a student primarily on this reasoning level is likely to give us a wrong final state, for example [a = 5, b = 5, c = 5]. This student is also unlikely to give us the abstract meaning of the code.

Pre-operational

At the pre-operational stage a novice programmer can reliably manually execute ('trace') several lines of code. The novice often makes inductive guesses about what a piece of code does by performing one or more successful traces, and finding relations between input and output. A student primarily exhibiting this reasoning level would be able to give us the proper final state [a = 5, b = 1, c = 1]. The student would be unable to deduce the meaning

of the code, but could give the purpose of the code using the start states and end states.

Concrete operational

At the concrete operational stage a novice programmer is able to reason deductively about code by reading the code, opposed to the inductive approach used by pre-operational novices. Students on this level of reasoning should be expected to trace the code successfully and deduce the purpose of the code correctly (swapping values a and b).

A student on this reasoning level is also able to work with more abstract concepts. The student should also be able to work with conservation. This means the student can preserve the specification while working on variations of an implementation. The student can also work with reversible processes, so the student can write code which *undoes* the work of another piece of code. The student can also work with transitive inference at this stage. This means that the student can observe the relationship between A and C given the relationships between A and B and between B and C. Despite being able to reason in such an abstract manner, the student is usually only able to do so in familiar and real situations.

Formal operational

The formal operational stage is defined as the ultimate stage of Piagetian reasoning. This is defined as the level at which an expert performs. As most students need guidance traversing the earlier stages, this stage will not be focused on in this paper.

1.3 Stage evaluation

The stage evaluation by Teague et al. (2013) was done using think-aloud sessions in order to investigate the reasoning of the student. In the sessions, the students were given a set of programming exercises and were asked to do the exercises while thinking aloud. The set of exercises consisted of tracing exercises and explaining exercises. During the sessions, the emphasis was on the reasoning of the student: the student should simply articulate thoughts rather than formulate a proper sentence. This was done to minimize the cognitive effort of the novice to produce a verbalization of his/her thoughts. These sessions were recorded and later analyzed. Comparing the behaviour of the student with the neo-Piagetian stage typical behaviour (as found by Lister (2011)) Teague determined the neo-Piagetian stage of the student. (Teague, 2015) also found behaviour which novice programmers from certain reasoning levels exhibited exhibited. The behaviour was linked to the sensorimotor, pre-operational and concrete operational stage in table 1.1.

From this table table 2.4 was distilled. This table is a reduced version of 1.1 in rubrics form. This way it is easier to read and gives a better view of behaviour which is related to other behaviour. For example “Struggles to trace code” is not close to “Can trace code reliably”.

Table 1.1: Linking programming behaviour to Neo-Piagetian stage

-	Sensorimotor	Pre-operational	Concrete operational
Struggles to trace (hand execute) code	x		
Tracing attempts takes considerable cognitive effort	x		
Misconceptions applied inconsistently	x		
Developing language skills in the domain	x		
Developing ability to distinguish between parts of code (e.g., reserved words, built-in names and symbols)	x		
Cognitive effort required to process syntax	x		
Cannot predict cyclic series even if starting at the first element	x		
Predominant strategy is trial and error	x		
Relies on specific values to trace code	x	x	
Cannot reason about code. At best, makes inductive guess	x	x	
Cannot see meaning in code	x	x	
Cannot see relationship between parts of code	x	x	
Egocentric: spatial and temporal centration	x	x	
Reluctant/unable to consider or attempt alternative solutions	x	x	
Reluctant to do multiple traces	x	x	
Cannot predict cyclic series past last element when starting from intermediary	x	x	
Cannot reason with abstractions of code	x	x	
Relies on specific values to write, check and verify code	x	x	
Relies on specific values to (attempt to) reason about code	x	x	
Relies on tracing with specific values for understanding	x	x	
Remaining misconceptions are applied consistently		x	
Traces without shortcuts (because cannot reason about code)		x	
Tracing is a mechanic process, without construing purpose		x	
Cannot extend and examine possibilities (e.g., unable to invent -		x	x
new uses for known constructs – limited to familiar uses and previous experiences)		x	x
Can trace reliably		x	x
Able to generate useful data for trace		x	x
Able to trace with abstractions (”abstract trace”)		x	x
Can write small programs, without extensive trial and error			x
Can work with concept of Conservation (retaining the specification while changing the implementation)			x
Can work with concept of Reversibility (undoing or reversing the effect of code)			x
Can work with concept of Transitive Inference			x
Can short-cut trace (because can reason about code)			x
Spatial and temporal decentration			x
Can see relationship between parts of code			x
No longer bound by perceptions - can infer reality			x
Can reason about code’s purpose			x
Can offer voluntary explanation of code’s purpose			x
Can explain code at an abstract level			x
Can predict and implement cyclic series from an intermediary point			x
Can recognise algorithmic patterns			x

Chapter 2

Methods

In order to determine how reliably a novice programmers' reasoning level can be estimated using written tests, a case study in the setting of a introductory programming course was used. Students were asked to voluntarily participate in a written test. The students were also asked to participate in think-aloud sessions. In order to rate our written test, we will compare these results with each other.

2.1 The participants

All of the participants are students in the introductory programming course "Imperative Programming". This course is the first programming course new computing science students encounter. This research covered topics the students have studied in the course so far and could be expected to understand.

2.2 Written test

For the written test, the students were asked to make several programming exercises which covered the topics they had studied so far. The test was divided into 4 parts having a certain topic. The topics were 'variable assignment', 'if and else-statements', 'while-loops' and 'for-loops and arrays'. The test consists of 2 tracing exercises per topic followed by a question where the student was asked to do one of the following tasks: explain the purpose of a piece of code, reverse the effect of a piece of code or rewrite a piece of code.

The students made the test on a computer using a testing environment they were used to. They were asked to do the test on their own without help of their neighbours. The time limit for the test was 60 minutes, but if they were done sooner they were free to go. The answers given by the students were later corrected and analyzed.

2.2.1 Questions

The questions were based on the questions in the catalogue from Kutscha (2017). The questions were translated from English to Dutch and were translated from Java to C++, and in some cases slightly altered. In the following sections the origin and purpose of every question is explained. These questions can be found in the appendices.

Question 1.1

```
int a;
int b;
a = 1;
b = a;
a = 2;
```

What are the values of *a* and *b* after the above code is executed?

This question is based on VA1M006 (Kutscha, 2017). It was slightly altered because the variable names could steer the students which could influence the students' behaviour. After assigning the value of one variable to another, students might have the misconception that those variables are now linked together like they would be in math. So after executing the code in this exercise, students having this misconception would give the answer *a = 2, b = 2* instead of the correct answer: *a = 2, b = 1.*

Question 1.2

```
int w = 0;
int h = 0;
int a = w * h;
w = 2;
h = 3;
std::cout<<a;
```

What is printed after this code is executed?

The question is based on VA1M008 (Kutscha, 2017). Students may have the misconception that code can have an effect on previous lines of code. This affects students who may not yet understand that code is executed sequentially. A student having this misconception would probably answer this question with *6* while the correct answer would be *0.*

Question 1.3

```
int a = 3;
int b = 4;
int c = 5;
int d;
d = a;
a = b;
b = c;
c = d;
```

Describe the purpose of this code.

Question is based on VA2C001 (Kutscha, 2017). Instead of picking lines of code (while remembering the chosen implementation) the student is asked to explain the purpose of a piece of code. A novice programmer without the ability to reason about code is unlikely to present us the purpose of the code: *swapping the values of a, b and c.* If the student answers this question correctly, s/he is likely to have reached the concrete operational stage.

Question 2.1

```
int a = 1;
if(a == 2){
    std::cout<<"first ";
}
std::cout<<"second ";
a = 2;
```

What is printed after this code is executed?

This question is based on CI1M001 (Kutscha, 2017). It covers the misconception novice programmers might have that an if-statement gets executed whenever the condition is met. Students with this misconception could think that the if-statement is executed after *a = 2;* and during the execution of the code *second first* is printed.

Question 2.2

```
std::cout<<"Enter an integer:";
int a;
std::cin>>a;
if(a > 20){
        std::cout<<"That's a big number!";
}
std::cout<<"That's not a big number!";
```

What is printed after the user enters 30 while executing the code?

Based on CI1M002 (Kutscha, 2017). The question covers the misconception that students might think that the computer understands what a piece of code is supposed to do. In this case a student with this misconception is likely to miss that *That's not a big number!* is also printed after printing *That's a big number!*.

Question 2.3

```
if(a > b){
    e = e + 1;
} else if (a < b){
    e = e - 1;
}
if(b < c){
    e = e - 1;
} else if(b > c){
    e = e + 1;
}
```

***a*, *b*, *c* and *e* were initialized before the above code. Write code that undoes the effect of this code on *e*. That means that after executing the above code, followed by your code, *e* has it's original value.**

This question is essentially the same as CI2R001 (Kutscha, 2017). The student has to write code which reverses the effect of a given piece of code on a variable. If the student answers this question correctly, s/he is likely at the concrete operational level.

Question 3.1

```
int c = 0;
while(c < 2){
    std::cout<<"abc ";
    c = c + 1;
    std::cout<<"def ";
}
```

What is printed after the execution of above code?

This questions was based on CW1M001 (Kutscha, 2017), but altered slightly because the original question could steer the student in a particular direction

which could influence the students behaviour. The exercise tests whether a student has the misconception that a while loop stops immediately when the condition is no longer met. Students having this misconception would likely answer this question with *abc def abc* instead of *abc def abc def*.

Question 3.2

```
int c = 4;
do{
    c = c + 1;
} while(c < 4);
std::cout<<c;
```

What is printed after the above code is executed?

This question is based on CW1M002 (Kutscha, 2017). The question tests whether a student has the misconception that a do-while also checks for the condition before the first iteration. Students having this misconception are likely to answer this question incorrectly with *4*.

Question 3.3

```
void greeting(int a, string name){
    int i = 0;
    a = a * 2;
    while(i < a){
        if(i%2 == 0){
            cout<<"Hello ";
        } else{
            cout<<name<<'\n';
        }
        i = i + 1;
    }
}
```

Write a function that has the same specification as the above code, but has a different implementation.

The question was based on CW2C001 (Kutscha, 2017). The purpose is to target the ability to transform between different implementations while conserving the specification of the code. According to Lister, only novice programmers reasoning at the concrete operational level should be able to do this. Therefore a student answering this question correctly is likely to be at the concrete operational level.

Question 4.1

```
int arr [5] = {10, 20, 30, 40, 50};
std::cout<<arr[1];
```

What is printed after the above code is executed?

Here it is tested whether a student has the misconception that the index counter for arrays and lists starts at 1. The question is essentially the same as LI1M001 (Kutscha, 2017). A student with this misconception is likely to answer this question with *10* instead of *20*.

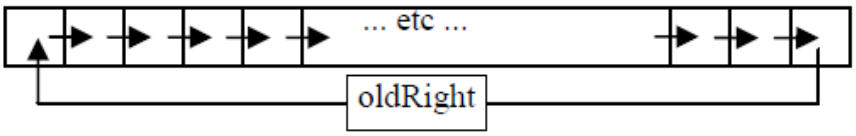
Question 4.2

```
int arr1 [4] = {1, 1, 2, 3};
int arr2 [4] = {8, 4, 2, 1};
int l = -1;
for(int i = 0; i < 4; i++){
    if(arr1[i] < arr2[i]){
        l = i;
    }
}
std::cout<<l;
```

What is printed after of above code is executed?

This question is basically the same as LI1O002 (Kutscha, 2017). The purpose of this question is to see whether the novice programmer can reliably trace code which contains a for-loop and an array.

Question 4.3




```
void f(int values[], int size){
    int oldright = values[size-1];
    for(int j = size - 1; j > 0; j--){
        values[j] = values[j-1];
    }
    values[0] = oldright;
}
```

***values* is a previously initialised array. *size* is equal to the size of *values*. For example: say *values* is equal to [1,2,3] then *size* is**

3. The above code takes the array and shifts every element one position to the right, the rightmost element ends up at the leftmost position. The image above the code also explain this. After executing this code on [1,2,3] the result is [3,1,2].

Write code that undoes the effect of above code. That means write code that shifts every element in the array one spot to the left with the leftmost element ending up at the rightmost position.

The question is based on LI2R001 (Kutscha, 2017). The purpose of the question is to see whether the student can work with reversible processes. A student able to answer this question correctly is likely to operate at the concrete operational level.

2.2.2 Classifying

The different types of questions were graded in 2 different manners as can be seen in table 2.1. Tracing questions were graded as either correct or incorrect, the answer was considered correct if it was the same as in the answer sheet and was considered incorrect otherwise. For the explaining and the coding questions (where a student is asked to explain/reverse/rewrite a piece of code) the 'form' of the answer was also considered. This provides additional information and gives the opportunity to account for students who are very close to the expected answer. A student who can trace could give answers which are correct, but are not in the form we expect. This is explained using the following example: a student is asked to explain a piece of code and gives a correct end-state (e.g. a = 3, b = 1, c = 2) or a correct line-by-line description. It is correct, but not of the expected form (rotates the values in a, b and c one step to the right). Or the other way around: the answer is of the expected form and shows the student is able to understand the question, but the answer is incorrect (e.g. off by one).

Table 2.1: Possible scores students

Type of question	Score	Meaning
Trace	+	Correct answer
Trace	-	Incorrect answer
Explain/reverse/rewrite	−−	Answer is incorrect and also not of the expected form.
Explain/reverse/rewrite	−+	Answer is incorrect but is of the expected form.
Explain/reverse/rewrite	+−	Answer is correct but is not of the expected form.
Explain/reverse/rewrite	++	Answer is correct and is in the expected form.

Classification of the student was done according to table 2.2. The table should be interpreted as follows: in the first column are the questions

from a certain topic; the questions are represented in the following form: *topic.exercise*. In the table *t.1*, *t.2* and *t.3* should be interpreted as question 1, 2 and 3 of a certain topic (e.g. 1.3 for exercise 3 on topic 1, variable assignment). In the first row are the reasoning levels.In the table S stands for sensorimotor, P for pre-operational and C for concrete operational. It is based on the characteristics of novice programmers on a certain level as found by Lister (2016), as such someone acting mostly on the pre-operational level is expected to be able to trace reliably but unable to explain the purpose of a piece of code.

Table 2.2: Classification of student

Question	**S**	**S/P**	**S/P**	**P**	**P/C**	**C**	**C**	**C**
t.1	-	-	+	+	+	-	+	+
t.2	-	+	-	+	+	+	-	+
t.3	Any	$--/-+/+-$	$--/-+/+-$	$--/+-$	$-+$	$++$	$++$	$++$

The students will be classified in general as well as per topic, so a student could be classified as concrete operational on the topic of variable assignment but could be on the stage of sensorimotor with regard to while-loops.

2.3 Think-aloud sessions

Several students also voluntarily participated in one-on-one think-aloud sessions. In these sessions the students were asked to complete 5 programming tasks while thinking aloud in a one-on-one environment. The students agreed that the session was recorded and would be compared to the written test. For these tasks the students had 20 minutes in total. The goal was to have the participants articulate what is going on in their head, so they did not have to formulate coherent sentences or explanations. This way we hope to replicate the thoughts of the student as good as possible. After finishing a task, the student was asked to write down their final answer on the sheet. The sessions were recorded using a smartphone and transcribed at a later moment. Afterwards the transcriptions were coded and these codings were used to classify the student.

2.3.1 Questions

Question 1

```
int a = 0;
int b;
int c = 10 + 5;
int d = 23;
int e = 4;
b = c;
c = a;
a = b;
e = c + 3;
d = c;
c = d;
```

After the above code was executed, what are the values of *a*, *b*, *c*, *d* and *e*?

This question is based on VA1M009 (Kutscha, 2017). It consists of several variable assignments covering multiple topics and possible misconceptions. Students that grasp the basic concepts of variable assignment are not expected to make errors in this exercise.

Question 2

```
a = a + b;
b = b + c;
c = c + a;
```

***a*, *b* and *c* are previously initialized integers. Write code that undoes the effect of the above code. This means that *a*, *b* and *c* have their original values.**

The question is based on VA2R001 (Kutscha, 2017) but is simplified slightly to account for possible longer thinking. The student has to undo the effect of a piece of code by writing other code. If the student answers this question correctly, the student is likely to reason at the concrete operational level.

Question 3

```
int answer;
if(a > b){
    if(b > c){
        answer = c;
    }
    else{
        answer = b;
    }
} else if(a > c){
    answer = c;
}
else {
    answer = a;
}
```

Describe the purpose of this code.

The question is based on CI2T002 (Kutscha, 2017). The questions purpose is to find out whether a novice programmer can reason with transitive inference. If a student answers this question correctly, s/he is likely to reason at the concrete operational level.

Question 4

```
int size = 9;
int array[] = {2, 5, 7, 7, 8, 10, 28, 50, 102};
bool b = true;
int c = 0;
while(c < size -1){
    if(array[c] > array[c+1]){
        b = false;
    }
    c = c + 1;
}
```

Describe the purpose of this code.

This questions is based on CW2T001 (Kutscha, 2017). The student has to reason about the purpose of the comparison of every pair of elements in order to find the purpose of the entire code fragment. In order to successfully do this, the student has to be able to work with transitive inference. If answered correctly, the student is likely to reason at the concrete operational level.

Question 5

```
int size = 10;
int arr [size] = {5 , 3, 6, 2, 1, 6, 3, 1, 22, 3};
int a = arr[0];
int b = arr[0];
for(int i = 1; i < size-1; i++){
    if(arr[i] > a){
        a = arr[i];
    }
    if(arr[i] < b){
        b = arr[i];
    }
}
```

Describe the purpose of this code.

The question is based on LI2C001 (Kutscha, 2017). It is altered in a way that the student now has to explain what the code does instead of being able to conserve a specific way of implementing a piece of code. This way the code is no longer about conservation but is now about transitive inference instead. This way it is more reliable to tell whether a student can reason with transitive inference. If the student is able to answer this question correctly, s/he is likely to reason at the concrete operational level.

2.3.2 Observable behaviour

In order to classify the students, specific behaviours or conclusions in the transcriptions were coded. The basis for the classification will be table the table 1.1 constructed by Teague. The table links programming behaviour to reasoning levels. Given the behaviour of the student, it is then possible to see at which reasoning level this behaviour is found. After coding the entire transcription, it is possible to see what the reasoning level is that is most predominant. This will be the reasoning level the student is classified as. Table 1.1 was difficult to use in practice. Because of this, behaviour which was easier to observe was linked to the higher level descriptions from Teague.

While this table gives an option to classify novice programmers, some of the behaviours are hard to observe. In order to actually use this in practice, these concepts are linked to observable behaviour. Table 2.3 shows codes used on the transcriptions.

Struggles to trace code

From the definition it seems pretty obvious, however, it is tough to determine whether a student is struggling to trace (a piece of) code. To classify a

Table 2.3: Observable behaviour

Incorrectly traces
Correctly traces
Takes several attempts to trace code
Reasons about code using information gained from tracing
Reasons about the abstract meaning from code
Does not see purpose of code within a code fragment
Unable to find meaning in code fragment
Applies misconceptions consistently
Answers question on a topic correctly

student as struggling to trace code, it means the student consistently has issues tracing code. One aspect could be that the student takes a long time to trace code, however it is hard to define *long*. For the purpose of this paper a student is considered to be "struggling to trace code" if s/he:

- made incorrect traces in over 50% of the cases, or
- took several attempts at tracing code in the majority of the cases.

Can trace code reliably

This behaviour is the counterpart of the previous behaviour. Here if a student "can trace code reliably" it means the student behaves opposite of its counterpart. This means the student:

- correctly traces in over 50% of the cases, and
- did not need several attempts at tracing in the majority of the cases.

Cannot reason about code, at best makes inductive guesses

It is hard to determine whether a student is unable to reason about code and whether s/he makes inductive guesses about a piece of code. What can be observed, while a student is thinking aloud, is whether the novice programmer is actually reasoning about the code. It is also possible that a novice programmer traces the program (several times) and then makes an inductive guess regarding the purpose of (a piece of) code. Here defined as "can not reason about code, at best makes inductive guesses" is when a student:

- is unable to find the abstract meaning from the piece of code, and/or
- uses information gained from tracing to reason about code.

Can reason about codes purpose

For a student to be able to reason about a piece of code, s/he needs to be able to do at least 2 things. The student should be able to determine a codes' purpose and should so without using the inductive approach described earlier. This means the student:

– is able to find the abstract meaning from the piece of code, and

– reasons about code without using the combinations of input and output gained from tracing (several times).

Cannot see relationship between parts of code

The inability of a student to see a relationship between parts of code should be observable through the inability from a student to understand the purpose of certain code. It suggests that the student does not understand the relationship between that code and some other code. In order to state that a student "cannot see relationship between parts of code", the student:

– does not see purpose of code within a code fragment, and/or

– cannot find meaning in entire code fragment.

Relies on specific values to (attempt to) reason about code

The student attempts to find the purpose of a piece of code through the usage of specific values for variables. After one or more (successful) traces the student reasons about the purpose of the code using the relationship between in- and outputs. Student "relies on specific values to (attempt to) reason about code" when the student:

– uses information gained from tracing to reason about code.

Remaining misconceptions are applied consistently

In order to determine a student can be observed with this behaviour, it is necessary for a student to work with a specific concepts more than once. If the subject is then found to consistently apply a misconception, this behaviour is observed.

Can work with concept of Reversibility

It is tough to be able to decide whether a student can work with the concept of reversibility. A first indication for this is that student is able to answer questions on this topic correctly. Another important aspect is the reasoning of a student, which makes this behaviour closely related to being able to reason about codes purpose. In this research the decision on whether a student can work with the concept of reversibility was made when:

– the student answered the question(s) on this topic correctly, and

– the student was able to reason about the codes purpose.

This means that if a student was able to answer the question correctly but was unable to reason about the code or made inductive guesses about the purpose was determined to be unable to work with this concept.

Can work with concept of Transitive Inference

It is tough to be able to decide whether a student can work with the concept of transitive inference. A first indication for this is that student is able to answer questions on this topic correctly. Another important aspect is the reasoning of a student, which makes this behaviour closely related to being able to reason about codes purpose. In this research the decision on whether a student can work with the concept of transitive inference was made when:

- the student answered the question(s) on this topic correctly, and
- the student was able to reason about the codes purpose.

This means that if a student was able to answer the question correctly but was unable to reason about the code or made inductive guesses about the purpose was determined to be unable to work with this concept.

2.3.3 Classifying

To classify students the coding of the transcription is used to estimate the reasoning level per question. Every code can be linked to behaviour specified by Teague which belongs to a certain reasoning level. This way the reasoning level per question can be decided by looking at the predominant behaviour. This reasoning can then be linked using a reduced version of 1.1, using the rubric in table 2.4.

Table 2.4: Linking behaviour to reasoning level

Ability/topic	**Sensorimotor**	**Pre-operational**	**Concrete-operational**
Tracing reliability	Struggles to trace code	Traces code reliably	
Reason about code	Cannot reason about codes' purpose, at best makes inductive guess		Can reason about codes' purpose
Relationship between parts of code	Cannot see relationship between parts of code		Can see relationship between parts of code
Concepts	Misconceptions applied inconsistently	Remaining misconceptions applied consistently	
Writing program	Main strategy is trial and error		Can write small programs, without extensive trial and error
Reversibility	Can't work with concept of reversibility		Can work with concept of reversibility
Transitive inference	Can't work with concept of reversibility		Can work with concept of reversibility

For the general level the same method is applied. It is possible for a student to behave in a way that makes it hard to determine the predominant stage of student if a student shows behaviour of 2 reasoning levels close to each other (sensorimotor and pre-operational, or pre-operational and concrete operational). In that case the student is classified as being somewhere between the 2 reasoning levels. For example, a student exhibits behaviour that belongs to the pre-operational stage but also shows behaviour that is linked to the concrete operational stage.

2.3.4 Question 1

Codes used for this question are: *traced correctly*, *traced incorrectly* and *signs of misconception*.

- For a student to be labeled sensorimotor for this question, s/he has to have:
  - *traced incorrectly*,

  and can have
  - *signs of misconception*.
- The student is not labeled between the sensorimotor and pre-operational level for this question.
- The student is labeled pre-operational for this question if the student has:
  - *traced correctly*,
  - *signs of misconception*.
- A student is labeled between pre-operational and concrete operational for this question if the student has:
  - *traced correctly*
- Students are not labeled concrete operational for this question.

Question 2

Codes used for this question are: *can work with reversibility*, *can not work with reversibility*, *correct answer*, *incorrect answer*.

- The student is not labeled sensorimotor for this question.
- The student is labeled between sensorimotor and pre-operational for this question when the student has:
  - *can not work with reversibility*,
  - *incorrect answer*.
- The student is not labeled pre-operational for this question.
- Students are not labeled between pre-operational and concrete operational for this question.
- The student is labeled concrete operational for this question if the student has:
  - *can work with reversibility*,
  - *correct answer*.

Question 3

Codes used for this question are: *can work with transitive inference*, *can not work with transitive inference*, *can see meaning in code*, *can not see meaning in code*, *can reason about code's purpose*, *can not reason about code's purpose*, *incorrect conclusion*, *correct conclusion*, *traces to find answer*.

- The student is not labeled sensorimotor for this question.
- The student is labeled between sensorimotor and pre-operational for this question when the student has:
  - *can not work with transitive inference*,
  - *can not see meaning in code*,
  - *can not reason about code's purpose*

  and can have:
  - *incorrect conclusion*
  - *traces to find answer*.
- A student is labeled pre-operational for this question if student has 1 of the following:
  - *can work with transitive inference*,
  - *can see meaning in code*,
  - *can reason about code's purpose*

  and can have:
  - *incorrect conclusion*
  - *traces to find answer*.
- Students are labeled between pre-operational and concrete operational for this question if the student has (at least) 2 out of the following:
  - *can work with transitive inference*,
  - *can see meaning in code*,
  - *can reason about code's purpose*

  and can have:
  - *incorrect conclusion*
  - *traces to find answer*.
- The student is labeled concrete operational for this question if the student has:

- *can not work with transitive inference*,
- *can see meaning in code*,
- *correct conclusion, can reason about code's purpose*,

and does not have:

- *incorrect conclusion.*
- *traces to find answer*

Question 4

Codes used for this question are: *can work with transitive inference*, *can not work with transitive inference*, *can see meaning in code*, *can not see meaning in code*, *can reason about code's purpose*, *can not reason about code's purpose*, *incorrect conclusion*, *traces to find answer*, *traces code with specific values.*

- The student is not labeled sensorimotor for this question.
- The student is labeled between sensorimotor and pre-operational for this question when the student has:
  - *can not work with transitive inference*,
  - *can not see meaning in code*,
  - *can not reason about code's purpose.*
- A student is labeled pre-operational for this question if student has 1 of the following:
  - *can work with transitive inference*,
  - *can see meaning in code*,
  - *can reason about code's purpose*,

  and can have:
  - *incorrect conclusion*,
  - *traces code with specific values*
  - *traces to find answer.*
- Students are labeled between pre-operational and concrete operational for this question if the student has 2 out of the following:
  - *can work with transitive inference*,
  - *can see meaning in code*,
  - *can reason about code's purpose*

and can have:

  - *incorrect conclusion*,
  - *traces code with specific values*
  - *traces to find answer*.

- The student is labeled concrete operational for this question if the student has:

  - *can not work with transitive inference*,
  - *can see meaning in code*,
  - *correct conclusion*
  - *can reason about code's purpose*

  and does not have:

  - *incorrect conclusion*.

Question 5

Codes used for this question are: *can work with transitive inference*, *can not work with transitive inference*, *can see meaning in code*, *can not see meaning in code*, *can reason about code's purpose*, *can not reason about code's purpose*, *incorrect conclusion*, *traces to find answer*, *traces code with specific values*, *correct trace*.

- The student is not labeled sensorimotor for this question.
- The student is labeled between sensorimotor and pre-operational for this question when the student has:

  - *can not work with transitive inference*,
  - *can not see meaning in code*,
  - *can not reason about code's purpose*.

- A student is labeled pre-operational for this question if student has 1 of the following:

  - *can work with transitive inference*,
  - *can see meaning in code*,
  - *can reason about code's purpose* codes,

  and can have:

  - *correct trace*,
  - *incorrect conclusion*,

  - *traces to find answer* and *traces code with specific values.*

- Students are labeled between pre-operational and concrete operational for this question if the student has (at least) 2 out of the following:

  - *can work with transitive inference*,
  - *can see meaning in code*,
  - *can reason about code's purpose*,

  and can have:

  - *can not work with transitive inference*,
  - *can not see meaning in code*,
  - *can not reason about code's purpose*,
  - *incorrect conclusion*,
  - *traces to find answer*,
  - *traces code with specific values.*

- The student is labeled concrete operational for this question if the student has

  - *can work with transitive inference*,
  - *can see meaning in code*,
  - *correct conclusion*,
  - *can reason about code's purpose*,

  and does not have:

  - *incorrect conclusion*,
  - *traces to find answer* and *traces code with specific values.*

Total

Using the overlapping waves model, the predominant behaviour between the different questions determines the total level of the student. The predominant behaviour is determined by counting the occurrences of a reasoning level. When the student scored a certain reasoning level on a question, it will increase the counter by 1. If a student scores between two reasoning levels, both levels increase their counter by 0.5, so if a student scores P/C (pre-operational/concrete operational) both pre-operational and concrete operational increase their counter by a half. The sum of the scores of the levels equals 5 for every student.

When one of the reasoning levels scores 3.5 or higher, it is certainly the most dominant and the student will be classified as such. An example to

illustrate: if a student has scores *[S: 0, P: 1.5, C: 3.5]* s/he will be classified as concrete operational. When the highest score for a reasoning level is 3 and the second highest score is lower than 2, it is also considered the most dominant level and the student is again classified as such. For instance, if a student scores *[S:0.5, P:3, C:1.5]* s/he is classified as pre-operational, as this is the most dominant level. When the highest score is 3 and the second highest score is 2, the highest score is a (shared) 2.5 or the highest score is 2, it is unclear which reasoning level is the most dominant. In these cases the student will be classified between these stages. For example, if a student has scores *[S:2.5, P: 2, 0.5]* the student is classified as sensorimotor/pre-operational.

Chapter 3

Results

3.1 Written test

The table below 3.1 shows the results of the written test taken by the students. One student was removed from the results, because either s/he did not understand the questions or did not take the test seriously. As can also be seen in the tables, student S19 had some issues with the testing environment and was not able to fill in Q2.3.

The last columns are labeled C1, C2, C3, C4, CT. These stand for classification part 1, classification part 2, classification part 3, classification part 4 and classification in total. Here S stands for sensorimotor, P for pre-operational, C for concrete operational and for example S/P stands for sensorimotor or pre-operational. For each part the classification was done as can be seen in table 2.2. The meaning of the cells was further explained in 2.1. The total classification of a student was based on the global performance of the student. Below every question will be discussed individually.

Some questions require some explaining, this will be done below.

Question 1.3

Unlike questions 1.1 and 1.2, this question checked whether the students could describe the purpose of a piece of code. The question was answered correctly by 68% of the students. All the remaining students gave an answer that was factually correct but not the expected type of answer, which means they did not understand the purpose of the code.

Question 2.3

This question checked whether students could work with the concept of reversibility. 80% of the students answered this question correctly. 16% of the students answered the question incorrectly. One student had issues with the testing environment and was unable to answer the question.

Table 3.1: Results written test

	Q1.1	Q1.2	Q1.3	Q2.1	Q2.2	Q2.3	Q3.1	Q3.2	Q3.3	Q4.1	Q4.2	Q4.3	C1	C2	C3	C4	CT
S1	+	+	+−	+	-	−−	-	-	+−	+	-	−−	P	S/P	S	S/P	S/P
S2	+	+	+−	+	-	++	+	-	++	+	+	−+	P	C	C	P/C	C
S3	+	+	+−	+	+	++	+	-	++	+	+	−+	P	C	C	P/C	P/C
S4	+	+	++	+	+	−−	+	+	+−	+	+	++	C	P	P	C	P/C
S5	+	+	++	+	+	++	+	-	+−	+	-	−−	C	C	S/P	S/P	P
S6	+	+	+−	+	+	++	+	+	+−	+	-	−+	P	C	P	S/P	P
S7	+	+	++	+	-	−−	+	+	+−	+	+	++	C	S/P	P	C	C
S8	+	+	++	+	-	++	+	-	+−	+	+	−−	C	C	S/P	P	P/C
S9	+	+	++	+	+	++	+	+	+−	+	+	++	C	C	P	C	C
S10	+	+	++	+	-	++	+	-	++	-	+	++	C	C	C	C	C
S11	+	+	++	+	+	++	+	+	−−	+	+	++	C	C	P	C	C
S12	+	+	++	+	+	++	+	+	++	+	+	++	C	C	C	C	C
S13	+	+	+−	+	+	++	+	+	+−	+	+	++	P	C	P	C	P/C
S14	+	-	++	+	+	++	+	-	+−	+	-	−−	C	C	S/P	S/P	P
S15	+	+	++	+	+	++	+	+	++	+	+	++	C	C	C	C	C
S16	+	+	+−	+	+	++	+	-	+−	+	+	++	P	C	S/P	C	P/C
S17	+	+	++	+	-	++	+	-	++	+	+	−−	C	C	C	P	C
S18	+	+	++	+	-	−−	+	+	+−	+	-	−+	C	S/P	P	S/P	S/P
S19	-	+	+−	+	+	ERROR	+	+	+−	+	+	−+	S/P	P/C	P	P	P
S20	+	+	++	+	+	++	+	-	+−	+	+	−−	C	C	S/P	P	C
S21	+	+	+−	+	-	++	-	+	−−	+	-	−−	P	C	S/P	S/P	P
S22	+	+	++	+	+	++	+	-	++	+	+	++	C	C	C	C	C
S23	+	+	++	+	-	++	+	-	++	+	-	++	C	C	C	C	C
S24	+	+	++	+	+	++	+	+	++	+	+	++	C	C	C	C	C
S25	+	+	++	+	+	++	+	+	++	+	+	−+	C	C	C	P/C	C
Correct	96%	96%	68%	100%	64%	80%	92%	52%	40%	96%	72%	48%					

Question 3.3

The student was asked to rewrite a function into another function with the same functionality. 40% of the students answered this question correctly. A fraction of 52% had a factually correct answer, but not the expected type of answer which means they did not see the purpose of the code. 8% of the students gave a wrong answer. Given the amount of students that answered this question almost correct, it's interesting to find out whether students did not see the correct answer or whether the question was not formulated properly.

Question 4.3

The question checked whether students could work with the concept of reversibility. 48% of the students answered this question correctly. 24% of the students had an answer close to the expected type of answer, but failed to execute it properly. 28% of the students had a wrong answer to this question.

3.2 Think-aloud sessions

In this section the different questions will be discussed. After that the students are discussed along with their determined reasoning level.

Questions

Question 1

This question was answered correctly by all the students. One student expressed some confusion, but was able to correctly answer the question. It makes sense that all the students were able to properly trace this piece of code as it only involved variable assignment and they had already followed 7 weeks of programming education.

Question 2

The question was answered correctly by 6 students. 2 students gave code that was not in the right order. 2 students solved the issue like a math problem. Instead of giving:

```
c = c - a;
b = b - c;
a = a - b;
```

They tried to solve it in a way as shown below, this answer was not accepted as a correct answer:

```
a' = 0.5*(a - b + c);
b' = 0.5*(b - c + a);
c' = 0.5*(c - a + b);
a = a';
b = b';
c = c';
```

Question 3

The question was answered correctly by 6 students, one of these students was unable to really connect the meaning of the different parts of the code. 3 students ended up giving a line-by-line description of the code. One student appeared to be able to work with transitive inference but was unable to give the purpose of the code and was unable to see the meaning of the different parts of code.

Question 4

The question was answered correctly by 2 students. 4 students incorrectly stated that the code checked whether a given array was in a descending

Table 3.2: Think-aloud sessions

	Q1	Q2	Q3	Q4	Q5	Total
S1	P	S/P	S/P	S/P	S/P	S/P
S4	P/C	C	S/P	S/P	C	P/C
S5	P/C	C	C	P/C	C	C
S6	P/C	C	P	S/P	P/C	P/C
S10	P/C	S/P	P/C	P/C	C	P/C
S15	P/C	C	C	P/C	C	C
S16	P/C	C	C	C	C	C
S18	P/C	S/P	C	P	C	P/C
S21	P/C	C	S/P	S/P	P/C	P
S22	P/C	S/P	C	C	C	C

order, while it should be ascending. Those students showed that they can work with transitive inference as that is a skill necessary to come to that conclusion.

Question 5

This question was answered correctly by 8 students. One student was unable to answer the question correctly. One managed to answer the question correctly, but found it through the usage of tracing the code and was therefor not labeled concrete operational. Given the number of students answering question 3 and question 4 correctly, it is interesting to see the number of students answering this question correctly.

Classifying students

In table 3.2 the results of the students can be seen. Column Q1 covers question 1, Q2 question 2 etc. In the column labeled "Total" the overall reasoning level of the student can be found. S stands for sensorimotor, P for pre-operational and C for concrete operational. Per student a short description of the behaviour the student exhibited while answering the questions and the codes applied to the transcription are given.

S1

Question 1

The student shows signs of confusion when trying to trace the code. "*It's b becomes c and not c becomes b, right?*". Despite this confusion, the student manages to trace the code correctly.
Codes applied: correct trace, signs of misconception.

Question 2

This time the student is unable to give us the proper answer. After giving the incorrect answer, the student mutters "*[...] I guess.*", indicating that

the students' strategy appears to be trial and error.
Codes applied: incorrect answer, can not work with reversibility

Question 3

The student gives a line-by-line interpretation of the code.
Codes applied: can not see meaning in code, incorrect conclusion, can not work with transitive inferene, can not reason about code's purpose.

Question 4

When attempting to find the purpose of this piece of code, the student traces the code. In order to trace the code, the student works with specific values that are available. The student is eventually unable to find meaning in the code.
Codes applied: Traces code with specific values, can not see meaning in code, can not reason about code's purpose, can not work with transitive inference.

Question 5

Again the student traces the code using specific values available. The student is again unable to find the purpose of the code.
Codes applied: Traces code with specific values, can not see meaning in code, can not reason about code's purpose, can not work with transitive inference.

S4

Question 1

The student traces the code correctly and without issues.
Codes applied: correct trace.

Question 2

After struggling a bit and having some doubts, the student successfully produces a correct answer. "*So to get them back, first c has to be returned to the old value*" which is the beginning of the solution produced by the student.
Codes applied: correct answer, can work with reversibility.

Question 3

The student is unable to see the connection between the pieces of code and eventually produces an answer which is basically the same as giving us a line-by-line description of the program.
Codes applied: can not work with transitive inference, can not see meaning in code, can not reason about code's purpose.

Question 4

The student concludes the purpose of the code as follows "*If it occurs at least once that the first number is bigger than the second.*" After which s/he correctly traces the array in the code and states that in the example b would remain true. The student does not see the meaning of pieces of code and does not see how these pieces are connected.
Codes applied: can not work with transitive inference, can not reason about code's purpose, can not see meaning in code, correct trace.

Question 5

After reading the code, the student quickly concludes that *a* will eventually be the biggest value in the array. After which s/he briefly doubts the previous conclusion but is certain afterwards. After this the student also sees that *b* will be the smallest value in the array. The student then traces the code to find out whether the conclusion was correct, which is the case.
Codes applied: can work with transitive inference, can reason about code's purpose, can see meaning in code, traces with specific values.

Other behaviour: Doubt.

S5

Question 1

The student traces the code correctly and without issues.
Codes applied: correct trace.

Question 2

After reading the code, the student is briefly confused "*I feel like you need to add a sort of 'd', so you can save something before overwriting...*". After realizing a can't be 'a - b' directly the student quickly realizes what s/he is supposed to do "*But b is already changed, so we should start at the bottom.*". The student understands how the pieces of code (a = a + b; b = b + c;) are connected and understands how to reverse this code.
Codes applied: can work with reversibility, correct answer.

Question 3

While tracing the piece of code, the student quickly realizes what is going on "*So it's looking at the smallest...*". After this, the s/he continues tracing to find out that was correct.
Codes applied: can work with transitive inference, can reason about code's purpose, can see meaning in code.

Question 4

The student traces the code and appears to have the right idea "*So you want to see for how long they're in order*". However instead of realizing the code checks whether the array is sorted, the student concludes the code checks for how long they're in order. It's not clear where this misconception originates from and the student successfully uses the concept of transitive inference while reasoning about the code.
Codes applied: can work with transitive inference, can not reason about code's purpose, can see meaning in code, correct trace, incorrect conclusion.

Question 5

During the trace, the student mutters "*so you start with [1]: 5[...]*" which means the student appears to trace with specific values. However after this utterance, the student quickly reaches the conclusion without using more values to trace "*So I think you want to find the greatest and smallest value.*"
Codes applied: can work with transitive inference, can reason about code's purpose, can see meaning in code, correct conclusion, traces code with specific values.

S6

Question 1

The student traces the code correctly and without issues.
Codes applied: correct trace.

Question 2

The student shows some signs of doubt while solving the question. Eventually the student presents the correct answer.
Codes applied: can work with reversibility, correct answer.

Other behaviour: Doubt.

Question 3

At the start the student mutters an incorrect interpretation of the code "*As it would appear, it gives us the highest value. Oh, no*". Later on the student shows signs of being able to work with transitive inference "*If a is greater than b [...] whether b is greater than c. That would mean a is also greater than c.*" Later on the student reaches the piece of code that is repeated, but represents another situation, "*So this is the same as here. So this doesn't really matter*". This could indicate that the student doesn't really see the relationship between pieces of code. The student gives the

correct answer, but did not seem completely convinced. After muttering the answer, the student claimed "*I know what's happening. But I don't know what the purpose is.*" So the student was not really convinced he had already found the answer.
Codes applied: can work with transitive inference, can not reason about code's purpose, can not see meaning in code.

Question 4

While tracing the code, the student appears to rely on certain values to trace the code, "*[...] checks whether 2 is greater than 5. That is not the case.*" After reading and tracing the code, the student does not seem to be able to deduce the purpose of the code "*So what the purpose is... Checking whether the next value in the array is higher... Is lower.*"
Codes applied: can not work with transitive inference, can not reason about code's purpose, can not see meaning in code, correct trace, incorrect conclusion, traces with specific values.

Question 5

When trying to solve this exercise. The student again traces the code and seems to rely on certain values to do so "*in this case... 3 is not greater than 5*". While concluding the purpose of the code, it is not clear whether the student deduces the purpose or induces this from the information gained from the trace "*If it eventually reaches the 6 then a is increased. So a is the highest value in the array. Then b will be the lowest I think.*"
Codes applied: can work with transitive inference, can reason about code's purpose, can see meaning in code, correct trace, traces code with specific values.

S10

Question 1

This student correctly traces the code.
Codes applied: correct trace.

Question 2

After reading the code fragment, the student says "*And then you want them back to the way they were, so a should be minus b. A equals a minus b.*" After which s/he continues this way.
Codes applied: can not work with reversibility, incorrect answer.

Question 3

The student takes some time to find a meaning to the code and at some point says "*If a is greater than c then it does c, but that is also here...*". This indicates that the student does not (yet) see the connection between the pieces of code. Eventually s/he finds the purpose of the code "*It displays... the smallest integer of the three values*".
Codes applied: can work with transitive inference, can reason about code's purpose, can not see meaning in code, correct trace.

Question 4

S/he concludes that the purpose of the code is to check whether the code check if an array is descending. This indicates the student can work with transitive inference. The conclusion of the student is incorrect though, it is unclear where this error originates from.
Codes applied: can work with transitive inference, incorrect conclusion, can see meaning in code, can reason about code's purpose.

Question 5

Again the student shows the ability to work with transitive inference: "*If location i is bigger than a, then a changes to array location i. [...] a becomes the greatest value in the array.*" Then after showing signs of doubt, the student concludes that b will become the smallest value.
Codes applied: can work with transitive inference, can see meaning in code, can reason about code's purpose.

Other behaviour: doubting several times.

S15

Question 1

The student traces the code correctly and without any issues.
Codes applied: correct trace.

Question 2

The student correctly reverses the code without issues. This indicates the student can work with the concept of reversibility.
Codes applied: correct answer, can work with reversibility.

Question 3

After reading the code, the student understand that the code is about finding the smallest value out of 3 variables. "*[...] if b is smaller than a you check whether c is also smaller than b and then the answer is c and that is then the smallest.*"
Codes applied: can work with transitive inference, can reason about code, can see meaning in code.

Question 4

S/he reads the piece of code and appears to trace with specific values available in the code "*This is smaller than uhm... 2 is smaller than 5. So uhm, here it's not true.*" The student then appears to be confused by this conclusion and the code, because after this piece of code the boolean value will still be true. The student then claims it becomes false. This incorrect conclusion is still applied when the student shows being able to work with transitive inference, because the student correctly concludes that the code checks whether an array is sorted. The student made an error and concluded the code checks whether the array is descending while it should be ascending.
Codes applied: can work with transitive inference, incorrect conclusion, can see meaning in code, traces with specific values.

Question 5

The student reads the code and quickly sees the purpose of the code: a will be the biggest element in the array and b will be the smallest element. The student concludes with a becoming 1 and b becoming 22 which is promptly fixed to a becoming 22 and b to 1 after realizing it was turned around.
Codes applied: can work with transitive inference, can see meaning in code, can reason about code's purpose.

S16

Question 1

The student traces the code correctly and has no problems doing so.
Codes applied: correct trace.

Question 2

The student has no problems finding that in order to reverse the effect of the code fragment, the operations should be the opposite and in the opposite direction.
Codes applied: correct answer, can work with reversibility.

Question 3

After reading the code, the student quickly realizes the purpose of the code fragment: "*which basically means that it gives the smallest value.*"
Codes applied: can work with transitive inference, can reason about code's purpose, can see meaning in code.

Question 4

While reading the piece of code, the student incorrectly states "*If it is greater than the second value in the array, then b is changed to false. And otherwise c is increased by 1.*" After that the student traces the code using the example array. After that s/he concludes that the code checks whether the array is ordered in ascending order.
Codes apllied: can work with transitive inference, can reason about code's purpose, can see meaning in code, traces code with specific values.

Other behaviour: looks like an inductive guess.

Question 5

S/he reads the piece of code and uses some values from the given array to see what happens, this tells us the student (occasionally) traces using specific values. After this the student finds the purpose of the code.
Codes applied: can work with transitive inference, can see meaning in code, can reason about code's purpose.

S18

Question 1

The student has no issues tracing the code correctly.
Codes applied: correct trace.

Question 2

The student is unable to reverse the effect of the code fragment by reversing the operations. Instead the student attempts to reverse the effect in another way "*if you, for example, subtract b from a you get a minus c. If you add c to that, you get 2a.*"
Codes applied: incorrect answer, can not work with reversibility.

Question 3

After reading the code, the student correctly concludes the purpose of the code fragment.

Codes applied: can work with transitive inference, can reason about code's purpose, can see meaning in code.

Question 4

The student works with transitive inference, but ends up with a wrong answer. The student incorrectly concludes "*The purpose of this code is to see whether it's a strictly descending... Whether the array contains strictly descending integer values.*" The student obviously does not manage to find the purpose of the code, but applies transitive inference without further issues.
Codes applied: can work with transitive inference, incorrect conclusion, can not reason about code's purpose, can not see meaning in code.

Question 5

The student reaches the correct conclusion: "*Here with a you get the maximal value, because every time an element in the array is greater than the value of a it replaces that value.*"
Codes applied: can work with transitive inference, can see meaning in code, can reason about code's purpose.

S21

Question 1

The student traces the code correctly and without any issues.
Codes applied: correct trace.

Question 2

The student has no trouble finding the solution for the answer: "*Let's see, we'll probably have to do it from back to front.*"
Codes applied: can work with reversibility, correct answer.

Question 3

After reading the code fragment, the student presents the answer in the form of a line-by-line description.
Codes applied: can not work with transitive inference, can not see meaning in code, can not reason about code's purpose.

Question 4

The student can not find the purpose of the code. This also indicates that the student is unable to work with transitive inference and that the student

is not able to see the connection between the various pieces of code. While presenting the answer, the student also presents an incorrect trace "*At this position. On the third.. Second position. Because 7 is not greater than 7. So it returns false on position 2*".
Codes applied: can not work with transitive inference, can not reason about code's purpose, can not see meaning in code.

Other behaviour: Incorrect tracing.

Question 5

At first, the student struggles to see meaning in the code: "*It's slightly awkward code if I read it.*" The student then traces the code using some values from the presented array, "*So a is 5 at the start and b is 5. If the array[i] is greater than 5, so at the second position. Oh no, at the sixth position*". After some time the student reaches the correct conclusion.
Codes applied: can not see meaning in code, traces with specific values, can work with transitive inference, can reason about code's purpose, correct conclusion.

S22

Question 1

The student traces the code correctly and without any issues.
Codes applied: correct answer.

Question 2

The student is unable to reverse the effect of the code fragment by reversing the operations. At first s/he struggles and skips the exercise, to finish it after finishing the rest of the exercises. Eventually the student tries to solve the problem in a different way: "*So a plus 2b plus c. Subtract c' which is c plus a. What remains is 2b*"
Codes applied: can not work with reversibility, incorrect answer.

Other behaviour: Struggled finding a solution.

Question 3

The student correctly finds the purpose of this piece of code.
Codes applied: can work with transitive inference, can see meaning in code, can reason about code's purpose, correct conclusion.

Question 4

After reading the code, the student concludes the purpose of the code correctly.
Codes applied: can work with transitive inference, can reason about code's purpose, can see meaning in code, correct conclusion.

Question 5

The student shows the ability to work with transitive inference as s/he found the correct meaning of the code: "*The moment the element it's checking is greater than the element that's currently saved, then the element you are currently checking gets saved in a. So eventually the highest value in the array is what remains.*"
Codes applied: can work with transitive inference, can see meaning in code, can reason about code's purpose.

3.3 Comparing written test to think-aloud sessions

In order to validate the use of the written test, the results from the written test are compared to the results from the thinking aloud sessions. First the total classification is covered. After that the different topics will be compared separately. The first tracing question was not used in determining the reasoning level for a certain topic as it did not add any information.

3.3.1 Total classification

Table 3.3 shows how the participating students scored. In the last column is the difference in stages between the written test and the think-aloud sessions. A 0 means there is no difference, 0.5 means the difference is half a stage and 1 means the difference is one stage, etc. The average difference between the scores is 0.35. The accuracy of the method, compared to the think-aloud, is 50% when only considering the exactly same stages as correct.

Table 3.3: Comparing results

	Written test	Think-aloud sessions	Difference
S1	S/P	S/P	0
S4	P/C	P/C	0
S5	P	C	1
S6	P	P/C	0.5
S10	C	P/C	0.5
S15	C	C	0
S16	P/C	C	0.5
S18	S/P	P/C	1
S21	P	P	0
S22	C	C	0

3.3.2 Variable assignment

For the first topic the result of question 2 of the think-aloud session was used and C1 from the written test, the results can be found in table 3.4. The average difference between the written test and the think-aloud session was 0.75. The accuracy of the written test was 40%.

Table 3.4: Comparing results: variable assignment

	Written test	Think-aloud sessions	Difference
S1	P	P	0
S4	C	C	0
S5	C	C	0
S6	P	C	1
S10	C	S/P	1.5
S15	C	C	0
S16	P	C	1
S18	C	S/P	1.5
S21	P	C	1
S22	C	S/P	1.5

3.3.3 If and else statements

For the second topic the result of question 3 of the think-aloud session was used and C2 from the written test, the results can be found in table 3.5. The average difference between the written test and the think-aloud session was 0.5. The accuracy of the written test was 50%.

Table 3.5: Comparing results: if and else statements

	Written test	Think-aloud sessions	Difference
S1	S/P	S/P	0
S4	P	S/P	0.5
S5	C	C	0
S6	C	P	1
S10	C	P/C	0.5
S15	C	C	0
S16	C	C	0
S18	S/P	C	1.5
S21	C	S/P	1.5
S22	C	C	0

While-loops

For the third topic the result of question 4 of the think-aloud session was used and C3 from the written test, the results can be found in table 3.6. The average difference between the written test and the think-aloud session was 0.5. The accuracy of the written test was 30%.

Table 3.6: Comparing results: while-loops

	Written test	Think-aloud sessions	Difference
S1	S	S/P	0.5
S4	P	S/P	0.5
S5	S/P	P/C	1
S6	P	S/P	0.5
S10	C	P/C	0.5
S15	C	P/C	0.5
S16	S/P	C	1.5
S18	P	P	0
S21	S/P	S/P	0
S22	C	C	0

For-loops and arrays

For the fourth topic the result of question 5 of the think-aloud session was used and C4 from the written test, the results can be found in table 3.7. The average difference between the written test and the think-aloud session was 0.5. The accuracy of the written test was 60%.

Table 3.7: Comparing results: for-loops and arrays

	Written test	Think-aloud sessions	Difference
S1	S/P	S/P	0
S4	C	C	0
S5	S/P	C	1.5
S6	S/P	P/C	1
S10	C	C	0
S15	C	C	0
S16	C	C	0
S18	S/P	C	1.5
S21	S/P	P/C	1
S22	C	C	0

Chapter 4

Conclusion and Discussion

Conclusion

The neo-Piagetian stage for the students determined from the written tests was the same as determined from the think-aloud sessions in 50% of the cases. As novice programmers can exhibit behaviour from several stages at once according to Listers' model, it might make sense that it is only at 50% accuracy. If we accept an error of half a level this increases to 80%.

Discussion

The accuracy of this written test (50%/80%) are lower than the results from Kutscha (2017) where the accuracy of the test was 63%, 91% if half a level difference is allowed. This difference can be attributed to the experience of the students involved. The scope of this research was students taking the introductory programming course. Kutscha (2017) worked with students in a more advanced programming course which was the successor of the programming course used in this research. This could explain the difference in accuracy between this research and the research by Kutscha.

The written test had an accuracy of 50%, which is not at a level where it is useful in practice. The set of students used to validate the results of the written test consisted of only 10 participants. Because of this, it is hard to generalize the outcome of this research. When allowing an error rate of half a reasoning stage, the results are much better. It would be interesting to apply this research on a larger group of students to see if the results hold.

For this research a method was developed to classify the students using the think-aloud sessions in a reliable way. In earlier stages a more interpretive method was used to classify the students which yielded a better match with the test results. This suggests some behaviour is still missing in the list used to classify students. The classification was relatively coarse, but future research could make the classification more fine-grained. In order to

do so, more specific behaviour exhibited by students should be found and linked to corresponding reasoning stages.

When inspecting the questions used in the written test and in the think-aloud sessions it is not clear why students gave incorrect answers to some questions. In cases where the student was unable to find the correct answer, it was not investigated whether the student was unable to find the correct answer, the student did not understand the question or whether the question was not phrased correctly. A telling example is question 3.3 from the written test where only 40% of the students answered the question correctly. Also interesting is why 4 students answered question 4 of the thinking-aloud sessions incorrectly because they confused ascending and descending. In the future time could be invested in finding out why students answered these questions incorrectly.

In this research, the stage determined from the thinking-aloud sessions was considered the correct stage because of earlier research by Teague et al. (2013). But unless we can actually follow the thought process of a novice programmer, it is unsure how accurate this classification is.

The tests were designed to be completed in a limited amount of time by the students. While this was successful and the written test was made under 60 minutes by every student and no student took longer than 20 minutes to complete the think-aloud session, the downside is that this experiment used a low number of questions. The think-aloud session only had 5 questions, where every topic was generally only tested in one question. The limited number of questions could have had an influence on the results and a greater number of questions would certainly increase the accuracy of the classifications.

In this research both the written test and the think-aloud sessions were taken at some point in the middle of the course, at which point most students should be expected to be able to work with the covered topics. For these tests to have meaning in a course context it would be better to conduct these more regularly, for example when a topic was fully covered to see if all the students are up to speed. An interesting extension to this research would be to take the same questions, but conduct the research spread out over a longer time period.

When looking at the concrete operational level, the focus of this research was on *reversibility* and *transitive inference*. According to (Lister, 2016) the students in the concrete operational stage can work with *conservation* as well. It could be worthwhile to see if the results still hold if conservation is taken into account as well.

References

Kutscha, T. (2017). *Towards a practical application of the neo-piagetian theory for novice programmers* (Unpublished master's thesis). Radboud University. (`http://www.ru.nl/publish/pages/769526/timkutschamasterthesis.pdf`)

Lister, R. (2011). Concrete and other neo-piagetian forms of reasoning in the novice programmer. In *Proceedings of the thirteenth australasian computing education conference-volume 114* (pp. 9–18).

Lister, R. (2016). Toward a developmental epistemology of computer programming. In *Proceedings of the 11th workshop in primary and secondary computing education* (pp. 5–16).

Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. In *Proceedings of the fourth international workshop on computing education research* (pp. 101–112).

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., ... Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. *ACM SIGCSE Bulletin*, *33*(4), 125–180.

Teague, D. (2015). *Neo-piagetian theory and the novice programmer* (Unpublished doctoral dissertation). Queensland University of Technology.

Teague, D., Corney, M., Ahadi, A., & Lister, R. (2013). A qualitative think aloud study of the early neo-piagetian stages of reasoning in novice programmers. In *Proceedings of the fifteenth australasian computing education conference-volume 136* (pp. 87–95).

Appendix A

Appendix

A.1 Questions written test, Dutch

Deel 1:

Opgave 1:

```
int a;
int b;
a = 1;
b = a;
a = 2;
```

Wat zijn de waarden van *a* en *b* na het uitvoeren van de bovenstaande code?

a=.....

b=.....

Opgave 2:

```
int w = 0;
int h = 0;
int a = w * h;
w = 2;
h = 3;
std::cout<<a;
```

Wat wordt er getoond nadat de bovenstaande code is uitgevoerd?

.....

.....

Opgave 3:

```
int a = 3;
int b = 4;
int c = 5;
int d;
d = a;
a = b;
b = c;
c = d;
```

Beschrijf wat de bedoeling is van bovenstaande code.

.....

.....

Deel 2:

Opgave 1:

```
int a = 1;
if(a == 2){
    std::cout<<"first ";
}
std::cout<<"second ";
a = 2;
```

Wat wordt er getoond nadat de bovenstaande code is uitgevoerd?

.....

Opgave 2:

```
std::cout<<"Enter an integer:";
int a;
std::cin>>a;
if(a > 20){
        std::cout<<"That's a big number!";
}
std::cout<<"That's not a big number!";
```

Wat wordt er getoond nadat de bovenstaande code is uitgevoerd en de gebruiker het getal 30 in heeft ingevoerd?

.....

Opgave 3:

```
if(a > b){
    e = e + 1;
} else if (a < b){
    e = e - 1;
}
if(b < c){
    e = e - 1;
} else if(b > c){
    e = e + 1;
}
```

***a*, *b*, *c* en *e* zijn eerder geïnitialiseerde integers. Schrijf code die het effect van bovenstaande code op *e* ongedaan maakt. D.w.z. zodanig dat als jouw code wordt uitgevoerd na bovenstaande code, *e* daarna de originele waarde heeft.**

.....

.....

.....

.....

Deel 3:

Opgave 1:

```
int c = 0;
while(c < 2){
    std::cout<<"abc ";
    c = c + 1;
    std::cout<<"def ";
}
```

Wat wordt er getoond nadat de bovenstaande code is uitgevoerd?

.....

Opgave 2:

```
int c = 4;
do{
    c = c + 1;
} while(c < 4);
std::cout<<c;
```

Wat wordt er getoond nadat de bovenstaande code is uitgevoerd?

.....

Opgave 3:

```
void greeting(int a, string name){
    int i = 0;
    a = a * 2;
    while(i < a){
        if(i%2 == 0){
            cout<<"Hello ";
        } else{
            cout<<name<<'\n';
        }
        i = i + 1;
    }
}
```

Schrijf een functie die hetzelfde gedrag vertoont als de bovenstaande functie, maar een andere implementatie heeft.

.....

.....

.....

.....

.....

Deel 4:

Opgave 1:

```
int arr [5] = {10, 20, 30, 40, 50};
std::cout<<arr[1];
```

Wat wordt er getoond na het uitvoeren van bovenstaande code?

.....

.....

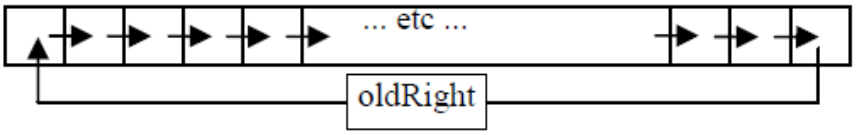
Opgave 2:

```
int arr1 [4] = {1, 1, 2, 3};
int arr2 [4] = {8, 4, 2, 1};
int l = -1;
for(int i = 0; i < 4; i++){
    if(arr1[i] < arr2[i]){
        l = i;
    }
}
std::cout<<l;
```

Wat wordt er getoond nadat de bovenstaande code is uitgevoerd?

.....

Opgave 3:




```
void f(int values[], int size){
    int oldright = values[size-1];
    for(int j = size - 1; j > 0; j--){
        values[j] = values[j-1];
    }
    values[0] = oldright;
}
```

values **is een eerder geïnitialiseerde array.** ***size*** **is gelijk aan de grootte van** ***values*****. Bijvoorbeeld: als** ***value*** **gelijk is aan [1,2,3] dan is** ***size*** **gelijk aan 3. De bovenstaande code neemt een array en schuift ieder element van de array één plekje naar rechts; het meest rechtse element eindigt op de meest linkse plek. Zie de afbeelding boven de code. Na het uitvoeren van deze code op bijvoorbeeld [1,2,3] ziet de array er als volgt uit: [3,1,2].**

Schrijf code die het effect van bovenstaande code ongedaan maakt. D.w.z. schrijf code die alle elementen in de array één plek naar links schuift; het meest linker element eindigt op de meest rechtse plek.

.....

.....

.....

.....

.....

A.2 Questions used in think-aloud sessions, Dutch

Opgave 1:

```
int a = 0;
int b;
int c = 10 + 5;
int d = 23;
int e = 4;
b = c;
c = a;
a = b;
e = c + 3;
d = c;
c = d;
```

Wat zijn de waarden van *a*, *b*, *c*, *d* en *e* na het uitvoeren van bovenstaande code?

.....

.....

Opgave 2:

```
a = a + b;
b = b + c;
c = c + a;
```

***a*, *b* en *c* zijn voor het bovenstaande stuk code geïnitialiseerde integers. Schrijf code die het effect van de bovenstaande code ongedaan maakt, d.w.z. code die er voor zorgt dat *a*, *b* en *c* weer de originele waarden hebben.**

.....

.....

.....

.....

.....

Opgave 3:

```
int answer;
if(a > b){
    if(b > c){
        answer = c;
    }
    else{
        answer = b;
    }
} else if(a > c){
    answer = c;
}
else {
    answer = a;
}
```

Beschrijf wat de bedoeling is van bovenstaande code.

.....

.....

.....

.....

.....

Opgave 4:

```
int size = 9;
int array[] = {2, 5, 7, 7, 8, 10, 28, 50, 102};
bool b = true;
int c = 0;
while(c < size -1){
    if(array[c] > array[c+1]){
        b = false;
    }
    c = c + 1;
}
```

Beschrijf wat de bedoeling is van bovenstaande code.

.....

.....

.....

.....

.....

Opgave 5:

```
int size = 10;
int arr [size] = {5 , 3, 6, 2, 1, 6, 3, 1, 22, 3};
int a = arr[0];
int b = arr[0];
for(int i = 1; i < size-1; i++){
    if(arr[i] > a){
        a = arr[i];
    }
    if(arr[i] < b){
        b = arr[i];
    }
}
```

Beschrijf wat de bedoeling is van bovenstaande code.

.....

.....

.....

.....

.....

A.3 Transcriptions, Dutch

Student 1

Question 1

Wat zijn de waarden van a,b,c,d,e... Uhm, a is 0. c is 15... Uhm, even kijken. B wordt c. Dus b wordt dan 15. En c wordt a, of doe ik het nu precies verkeerdom? Het is toch b wordt c en niet c wordt b, of b krijgt de waarde van c toch... c wordt dan 0... Denk ik. En a wordt b. Dus a is dan 15. e is c plus 3. dus e is 3... d is c, dus d is 0 en dan wordt c is d is c is 0.

Question 2

A, b, c zijn voor bovenstaande code geïnitialiseerd. Schrijf code die het effect van bovenstaande code ongedaan maakt. Dat wil zeggen, code die er voor zorgt dat a, b en c weer de originele waarde hebben. Uhm... a is dan a min b. b is dan b min c. Ik heb echt het gevoel dat ik alles fout doe... En dit wordt dan c min a. Gok ik.

Question 3

Beschrijf wat de bedoeling is. Uhm... Als a groter is dan b. Dan als b groter dan c, dan wordt je antwoord.. is c. En als a wel groter is dan b.. En b is niet groter dan c... Dan is je antwoord b. En als a niet groter dan b en.. maar.. a groter dan c, dan is het antwoord c. En als a niet groter dan... b en a niet groter dan c dan is het antwoord a.

Question 4

Uhmm... Een array met 9 elementen... En een c die is 0. Als c kleiner is dan 9 min 1. Even kijken hoor... Als 0 kleiner dan 8... Dan doet die c is c plus 1. En als... Die... Array 0 wordt dit dus dat is 2, groter is dan 0 plus 1, dus dat is 5... Dat is niet zo. Dus dan doet die... En dan hoe kan je dit ofschrijven... Tot je nieuwe c is kleiner dan 8... Ik heb geen idee.. Ik weet niet hoe ik dit zo moet opschrijven.

Question 5

Uhmm... Wat is de bedoeling van bovenstaande code... Je begint met de eerste waarde van je array.. En die ga je vergelijken.. En dan hier pak je de tweede waarde... Dus... Waarde uit de array vergelijken met de eerste waarde van de array. Uhmm... Als de eerste groter is, dan... en als de eerste waarde kleiner is dan de eerste... Nee... Dit moet uit de array...

Kleiner is dan de eerste uit de array. Dan is b.. Ja en dan dit loopen tot i niet... groter of gelijk is aan deze.

Student 4

Question 1

Uhm ik ga nu eerst kijken wat hier staat voor ik de vraag lees... Dus int a is 0, dus gewoon a is 0. We hebben ook een getal b, geheel getal, uh c is 15. d is 23, e is 4. Dan wordt b 15. c wordt a, dus dat wordt 0. a wordt b, uh, en dat is c nu. Dus dat wordt dan 15. En nu ga ik de vraag lezen. Wat zijn de waarden van a, b, c, d en e na het uitvoeren van bovenstaande code, o ja. Dus uh, b wordt 15, uh c wordt 0, a wordt b wat nu 15 is, dus a wordt 15. Uh, waar ben ik nu... e wordt c plus 3, c is 0 dus e wordt 3. d wordt c. c is 0, dus d wordt 0. En c wordt d dus ook 0. Volgens mij. Dus wat hadden we. A is 15. c wordt 0, d wordt 0 en wordt 3...

Question 2

a is a plus b. b is b plus c. c is c plus a. a, b en c zijn voor het bovenstaande stuk code geïnitialiseerde integers. Schrijf code die het effect van bovenstaande code ongedaan maakt. Dat wil zeggen code die ervoor zorgt dat a, b en c weer de originele waarde hebben. Uhm. Ok. Dus eerst even kijken wat er nou gebeurt. A wordt a plus b, dus zichzelf plus b. b wordt zichzelf plus c. En c wordt zichzelf plus de nieuwe a. Dus om ze terug te krijgen moeten we eerst c weer de oude maken. Dus c wordt c min a. En b wordt gewoon, daar moet nog een puntkomma achter, en b wordt... Oh dat is een probleem. Uh, denk ik. Nee want moet de oude c gebruiken om terug te gaan naar de oude b zeg maar. Dus b wordt b min c. En dit is nu al de oude c. En a wordt a min b volgens mij gewoon. Ook weer de oude b. En die hebben we al de oude b gemaakt, dus dit klopt volgens mij.

Question 3

Beschrijf wat de bedoeling is van bovenstaande code. Uhm, even kijken, we hebben een integer answer. Als a groter is dan b... Ik neem aan dat a, b en c al integers zijn... Uh als b groter dan c en dat is dan wel in de andere if. Dan wordt het antwoord c. Dus, het wordt c als als groter is dan b en b groter is dan c. En dat moet ik denk ik dan opschrijven als antwoord, maar daar wacht ik nog even mee. Het wordt b als a groter is dan b en uh b kleiner of gelijk is aan c. Uh. Het wordt c als a groter is dan c. En het wordt a in alle andere situaties. En ik denk dat het niet nodig is om uit te gaan leggen wat al die andere situaties zijn. Wat ik misschien normaal wel zou doen als ik een echt programma erbij zou moeten schrijven ofzo. Uhm. Dus uh... Verandert de integer c... Uh answer bedoel ik naar: c als a

groter is dan b en b groter dan c. Uh. Naar b als a groter is dan b en uhm. B groter dan c.. Nee, gewoon als a, b en b kleiner gelijk is aan c, dat was hem. En dan c als a groter is dan c en a kleiner of gelijk is aan b. Dat was ik net vergeten, dat moet er nog bij, anders gaat die er niet in. Dat klopt toch... En naar d, nee naar a, als de andere 2 if-jes niet gebeuren, dus als a kleiner of gelijk is aan b en a kleiner gelijk is aan c.

Question 4

Uhm. Dus we hebben een integer size. Die is nu 9. We hebben een array, met allemaal dingetjes erin, allemaal getallen. We hebben een b die waar of niet waar is, we hebben een integer c die 0 is. Nu ga ik even de vraag lezen. Beschrijf wat de bedoeling is van de bovenstaande code. Dus uh, while c is kleiner dan size min 1. Dan ga je ervan uit dat size de grootte is van de array en dat ga ik nu even natellen. 1, 2, 3, 4, 5, 6, 7, 8, 9. Ik begin me wel af te vragen, ik dacht dat in die... tussen die vierkant haakjes van de array de hoeveelheid van de array moest... Op een tentamen kan je het niet nakijken en als ik het zou programmeren zou ik gaan kloten ermee. Uhm. Dus while c kleiner is dan size min 1. Uh dus dat is gewoon terwijl c kleiner... Gewoon, hij gaat dus gewoon het array doorlopen waarschijnlijk. Uh, want hij begint bij 0 want c is 0. Dus hij begint bij het eerste dinges van het array. Als array c groter is dan array c plus 1, dus dan kijkt die naar het plekje er rechts van in het array, dan is b false. En uh, c wordt sowieso c plus 1. Dus dan gaat die gewoon naar de volgende plek in het array kijken. Ik weet niet wat die dan doet als die bij de laatste komt... Uh, maar dat moet hier verder denk ik niet bij. De bedoeling van de code is om te kijken of de code... Kijkt naar elk getal in het array, en vergelijkt dat getal met het volgende getal uit het array. En uh, zet b naar false als dus in van die getallen dus groter is. Zet b naar false als het 1 keer, en dan ook echt 1 keer. Niet echt 1 keer, maar minstens 1 keer. Als het minstens 1 keer voorkomt dat het eerste getal groter is dan de tweede. Dat had ik heel wat korter op kunnen schrijven, maar dit is wel wat die doet. Dus wat wordt het uiteindelijk, ik weet niet of het bij de vraag hoort, maar ik vind het gewoon interessant om te kijken. Nee, hij blijft dus gewoon true. Volgens mij.

Question 5

Beschrijf wat de bedoeling is van de bovenstaande code. De grootte is 10. Dus ik kijk even naar het array en er staan wat mij betreft willekeurige getallen, ik zie niet echt... Oh integers allemaal. En we hebben een integer a en die is 5 want het is de nulste plek van het array. En b is ook 5, want dat is ook de nulste plek van het array. Gaan we een for-loopje in, die begint bij 1. Dus hij begint zo te kijken bij de 3, als ik even vooruit kijk naar waar die naar gaat kijken. Uhm, dan uh, hij gaat door tot het einde van het array,

tot die 3. En hij gaat steeds 1 stapje gewoon. Uhm, if array en dan die plek groter is dan a, wat dus gewoon de eerste plek van die 5 is, de eerste plek in het array. En nu begrijp ik ook waarom die bij 1 begint en niet bij 0, de for-loop. Dan wordt a die plek in het array. Dus als die groter wordt dan wordt a die grotere zeg maar. Uh, en als die kleiner is dan wordt b die kleinere. Dus hij zet a uiteindelijk naar het grootste getal denk ik. Nee... Ja, hij zet a uiteindelijk naar het grootste getal in het array. En b naar het kleinste getal. Denk ik... Ja. De bedoeling van bovenstaande code... Verandert a in het grootste getal in het array. En b naar de kleinste. En dan zou ik nu gaan checken of het klopt. Ik loop hem gewoon nog even door. Hij begint bij array i, wat bij 1 is dus, dan gaat die bij 5, dat is dus 3. Kijken of 3 groter is dan 5, wat niet zo is. Dus dan verandert die b naar uh. Oeh ik bedenk me nu dat er iets fout... Oh nee, dat is niet fout. Ik bedacht me als die een even groot getal tegen komt, wat er dan gaat gebeuren. Maar dan blijft... Dan gebeurt er niks dus dan blijven a en b gewoon hetzelfde, wat er ook moet gebeuren. Dus 3 zou in dit geval... b zou in 3 veranderd worden en dat is tot dan toe ook het kleinste getal.

Student 5

Question 1

Wat zijn de waarden van a, b, c, d, e na het uitvoeren van bovenstaande code. Het laatste wat voor a gedaan is dat die gelijk gesteld wordt aan b. b was gelijk aan c. En c was gelijk aan 10 plus 5 is 15. Dus a is 15. Uh. B is het laatste gezegd dat die gelijk is aan c. Dus die wordt ook 15. c was d, en d was c en c is a en a is 0. En laten we d was c, c was a, a was 0. En e, e is c plus 3, c is a, a was 0. Dus e is 3. Ik zou het normaal nog controleren als het een echt tentamen was...

Question 2

a, b en c zijn voor het bovenstaande stuk code geïnitialiseerde integers. Schrijf code die het effect van bovenstaande code ongedaan maakt. Dat wil zeggen, schrijf code die zorgt dat a, b, c de originele waarde hebben. A is a plus b. b is b plus c. En c is c plus a. We veranderen eerst a in iets maar die a wordt ook weer bij c gebruikt. Dus eigenlijk staat er bij c; a plus b. Voor mijn gevoel moet je er eerst een soort d bij doen, zodat je iets kan opslaan voordat je het overschrijft maar... Dus eigenlijk zou ik zeggen a is a min b. Maar de b is al veranderd dus we moeten onderaan beginnen. Dus c is c min a. En dan b is b min c. En dat is, in die volgorde klopt het weer. En dan a is a min b.

Question 3

Beschrijf wat de bedoeling is van bovenstaande code... Het rekent wat getallen uit. Als a groter is dan b en b is groter dan c dan is het antwoord c. Anders b. En als a groter is dan c dan wordt het ook weer c en anders is het antwoord a. Dus er wordt daar gekeken naar de kleinste.. Als b kleiner is dan c dan is b het kleinste antwoord. En als a kleiner is dan b, dan wordt gekeken of a groter is dan c. Dan geven ze volgens mij elke keer het kleinste antwoord. De kleinste van de 3 getallen.

Question 4

Beschrijf wat de bedoeling is van bovenstaande code... Een rij met een grootte, een boolean die true is. C is 0 en als c kleiner is dan de grootte uhm. Als het getal op die plaats in de rij groter is dan het getal verderop dan is die false. En dan kijk je op de rij dus... Je doet dit net zo lang tot het volgende getal kleiner is dan het getal daarvoor. Dus eigenlijk wil je kijken hoe ver ze op volgorde staan. Te weten welke plaats in de rij het volgende getal niet meer groter is dan het getal daarvoor.

Question 5

Nog een keer een code. Weer met een array en een grootte. En 2 losse integers en een for-loop. En als de... als die plaats groter is dan a dan is... a gelijk aan die plaats. En als die kleiner is dan b dan wordt... Dan krijgt c de waarde van die plaats. Dus je begint bij

1

5 dan is a. a en b dat zijn de eerste van de rij. Dus volgens mij wil je het grootste en het kleinste getal vinden. Zodra die nieuwe groter... a is het grootste getal en b is het kleinste getal.

Student 6

Question 1

Waarden van a, b, c, d, e... Eens zien. Je ziet al meteen dat b geen waarde krijgt aan het begin, maar dat komt daarna meteen al, dus dat is geen probleem. Dus b wordt eerst gezet naar c, dus b is 15. c wordt dan verandert naar a, dus verandert c in 0. Dan wordt a veranderd in b, dus dan is gelijk aan b is a gelijk aan 15. Dan wordt e gelijk aan c plus 3. c is nu 0. e wordt dan 0 plus 3 is 3. Daarna wordt d veranderd in c. En c is 0, dus d is ook 0. En dan verandert c in d en d is 0. Dus c is nog steeds 0.

Question 2

Nou... Ok... a wordt b opgeteld.. bij b wordt c opgeteld en bij c wordt a opgeteld. Uhm.. onderaan beginnen, want hier a heeft een hogere waarde... Toch? Ja. Dus c is dan gelijk aan c – a. b is gelijk aan... Klopt dat? ja. B min c. Want c moet eerst weer omlaag voordat we die b kunnen veranderen. Dan is a gelijk aan a min b.

Question 3

Bedoeling van bovenstaande code... a groter dan b... en c... Zo te zien wordt het hoogste antwoord gegeven. Of nee niet... Hier wordt eerst gekeken welke het grootste is a, b of c. Zo te zien... Als a groter is dan b... Dan wordt gekeken of b groter is dan c. Dus dat zou betekenen dat a ook groter is dan c. En anders zou dus b grootste... Kleinste zijn. Nee wacht. Ok, geen context... een beetje vreemd dan. Dus... Hier was a dus groter dan c. Dan is answer gelijk aan c. Als a groter dan b en b niet groter dan c... Dan is answer gelijk aan b. Dus hier staat eigenlijk hetzelfde als hier. Dus die maakt eigenlijk niet echt uit... Als a kleiner is dan b, dan is answer gelijk aan a. Dus... dan wordt het kleinste antwoord getoond? Ik weet wat er gebeurd... Maar ik weet niet wat de bedoeling is. Ik weet wel wat er gebeurd, maar ik zie de logica er niet echt achter... Hier wordt... Anders ga ik wel gewoon door, ik weet in ieder geval wat er wordt gedaan.

Question 4

Bedoeling van bovenstaande code... 9... Even tellen. Bool b is true. C is 0. Zolang c kleiner is dan size min 1. size min 1 omdat c begint bij 0... De array op positie c dus positie 0, moet groter zijn dan de positie die er na komt. En dan wordt false getoond en anders wordt c gewoon 1 opgehoogd voor de volgende te testen. Dus begint die gewoon hier en kijkt die of 2 groter is dan 5. Dat is niet zo, 5 hoger dan 7... Niks is hoger dus er komt nooit false uit. Dus wat de bedoeling is... Kijken of de volgende waarde in de array hoger is... lager is.

Question 5

Int size is 10.. Array... a is 5. b is 5, want dat is positie 0 in de array. For-loop... Kleiner dan size min 1, omdat die bij 0 begint. En die.. begint op 1. Omdat a en b al gezet zijn op die positie dus.. dan kan die gewoon op 1 beginnen. Als array positie i dus gewoon een volgende positie, hoger is dan a. Dan is a gelijk aan i... De volgende positie gekeken of die hoger is dan de vorige, in dit geval... 3 is niet hoger dan 5. Dus blijft hetzelfde... Als die uiteindelijk aankomt bij de 6 dan wordt a omhoog gedaan. Dus a is de hoogste waarde in de array... Zal b wel de laagste zijn denk ik... Ja.

Student 10

Question 1

Uhm... Ik lees het nu gewoon door... b is niet geïnitialiseerd... b is c. Dus b wordt 15. c is a, dus c wordt 0. a is b, en b was net 15 geworden dus a is ook 15. a is b. b is 15. Oh, dat had ik net al. E is c plus 3. c is 0, dus e is nu 3. d is c, dus... en c is 0 dus d is 0. c is d en d was net c, dus dat doet verder niet zo heel veel. Wat zijn de waarden? B is c, b is 15. c is a, c is 0. a is b, en b was net 15 dus a is 15. e is c plus 3, c is net veranderd in a... dus e is 3. d is c. Dus... en c is 0. Dus d is 0. En c is d, dus c is 0.

Question 2

Ok... Geïnitialiseerd... Dus ze hebben waardes... Schrijf code die het ongedaan maakt. Uhm... Dit zorgt ervoor dat ze bij elkaar op worden geteld. Dus... c plus a... En dan wil je ze weer terug hebben dus a moet min b worden. A is a min b. b is b min c. c is c min a.

Question 3

De bedoeling van deze code... Als a is groter dan b en b is groter dan c dan is het c. Dus... Als a groter dan c dan zegt die c. Anders zegt die b... Dus... Anders als a is groter dan b maar b is kleiner dan c, dan is het antwoord b. Dus, uhmm, dus a is groter dan b. Als a is groter dan c dan doet die c, maar dat heeft die hier ook al... Else answer is a... Wanneer doet die a? Als a is niet groter dan b en als a is kleiner dan b en c. En... a is kleiner dan c... Dan is het a. Als a groter dan c... Als c is de kleinste dan doet die c... Als b is de kleinste dan doet die b. Ja dus... Hij displayed... het kleinste getal van drie waardes.

Question 4

Size is 9... Dat klopt... Uhm, bool b is true... Als dus, hij loopt gewoon het hele ding door waarschijnlijk. Ja. Dit, hiermee loopt die het array door. Als array is groter dan... Dan wordt die false. Dus, uhm, als de plek links in het array groter is dan de plek rechts, dan wordt b false. Dus deze uhm, geeft aan of het een aflopende reeks is. Als die afloopt dan blijft b true. Dus checken of de reeks integers in de array vanaf aflopende grootte zijn.

Question 5

Beschrijf wat de bedoeling is van deze... Array op plek 0.. a.. b is array op plek 0... Size 10.. For int... dit is ook loopt gewoon er doorheen tot die bij het einde is.. Oh nee, niet totdat die bij het einde is. Totdat die 1 voor het einde is. Als het goed is. Ja.. Als plek i groter is dan a, dan wordt

a array plek i... Dus uh, eerst wordt de plek op... a wordt de grootste waarde in 't array. B is kleinste waarde...Toch? Ja. Ik denk het wel... Nou ja, trouwens... Met als uitzondering de laatste dan... Uhm, a wordt de grootste integer in het array...van de laatste waarde in het array. B wordt de kleinste waarde... Klopt die met uitzondering wel... Hij begint bij 1, ah hij begint bij 1. Ja dat klopt dus dan niet. B wordt de kleinste waarde in het array. En hij kan bij 1 beginnen omdat hij hier al standaard bij de eerste plek zet.

Student 15

Question 1

Uhm, b is c is 15. Uhm, c is a is 0. a is b is 15. e is c plus 3 is 0 plus 3 is dan 3. Uhm d is c is 0. c is d is 0.

Question 2

Uhm. C wordt als laatste uitgevoerd. Dus dan is het denk ik handig om dat als eerste weer uh als inverse te doen, dus dan moet het c is c min a. Uhm. Daarna b is b min c. En a is a min b. Omdat ja, het moet uhm, het moet het effect ongedaan maken, dus dan moet je het tegenovergestelde doen van wat daar is gedaan.

Question 3

Uhm... De bedoeling is om, het kleinste antwoord te vinden. Want eerst wordt gekeken welke van a of b het kleinste is en dan als b kleiner is dan a dan ga je kijken of c ook kleiner is dan b en dan is het antwoord c en dat is dan de kleinste. Anders is b de kleinste. Als a niet kleiner is dan b. Ik bedoel als b niet kleiner is dan a, dan is a dus uhm kleiner of gelijk aan b. En dan ga je kijken of c nog kleiner is dan a. Als dat zo is dan is c dus de kleinste en anders is a de kleinste.

Question 4

Je hebt een integer size, die is 9. Je hebt een integer array, met 9 geïnitialiseerde waardes. Size is dan de lengte van de array. Boolean b is true. En nog een integer c die is 0. Zolang de c kleiner is dan de maat van de array min eentje. Moet je uhm, de volgende code uitvoeren. Als uhm... Ja je begint bij de eerste element van de array natuurlijk omdat c eerst 0 is, als die kleiner is dan het volgende element dan is b false. Anders blijft die natuurlijk wat die was. En dan verhoog je de waarde 1. En dan uhm... Even kijken. Deze is kleiner dan uhm... 2 is kleiner dan 5. Dus uhm, hier is die niet waar. Dus dan wordt die false. Uhm. Hij kijkt... Ik denk dat die kijkt of uhm, de

array gestructureerd is. Dat de grootste waarde als eerste in de array staat en de laagste... Dat die aflopend is de array. Maar hier is die oplopend... Dus dan wordt b natuurlijk false, maar als b true is... b blijft true als die aflopend is...

Question 5

Uhm. De size van de array is 10. En uhm alle 10 de elementen zijn geïnitialiseerd. Uhm. En je hebt een a en b die allebei het eerste element van de array bevatten. En dan uhm en for-loop. Zolang.. van 1 tot uhm, de size van de array min eentje moet je gaan kijken als een element van de array groter is dan a. Dan wordt a dat element. En als een element van de array kleiner is dan b dan wordt b dat element. Dus dan zoek je eigenlijk... voor a eigenlijk het grootste element van de array en voor b het kleinste element van het array. Dan wordt a uiteindelijk 1, deze 1, de eerste 1 omdat er niet kleiner gelijk staat uh groter of gelijk bedoel ik. En b wordt uiteindelijk 22. Nu heb ik het omgedraaid per ongeluk...

Student 16

Question 1

Wat zijn de waarden van a,b,c,d en e na het uitvoeren van bovenstaande code. Hij initialiseert eerst a naar 0, b niet. C wordt geïnitialiseerd naar 15. d naar 23, e naar 4. b krijgt de waarde van c, dus b wordt 15. c krijgt de waarde van a, a was 0. dus c is 0. a krijgt de waarde van b. de laatste waarde die b heeft gekregen is 15 dus a wordt 15. e krijgt de waarde c plus 3. de laatste waarde van c was 0 dus e krijgt de waarde van 3. d krijgt de waarde van c, laatste waarde van c is 0 dus d is 0. en c is d dus dat is de laatste waarde van d en dat is 0. Die blijft hetzelfde, dus die kan ik laten staan. Dan heb ik de waarden van a, b, c, d en e.

Question 2

a, b en c zijn voor het bovenstaande stuk code geïnitialiseerde integers. Schrijf code die het effect van bovenstaande code ongedaan maakt. Dat wil zeggen, code die ervoor zorgt dat a, b en c weer de originele waarden hebben. A is a plus b. b is b plus c en c is c plus a. Uhm, even denken. Ze zitten allemaal 2 keer er in. Wat betekent dat ze allemaal een waarde van elkaar hebben gekregen uiteindelijk... En ze komen zo na elkaar. Om dit te moeten doen zou ik eerst c naar zijn oude waarde terug moeten brengen, dan b en dan a. Dus in omgekeerde volgorde. Dus dat betekent dat de nieuwe c, c min a wordt. Dat de b, doordat c weer de originele waarde heeft uh, b is b min c. En de a wordt dan de a min de b. En dan zouden ze allemaal weer terug moeten zijn bij de originele waarde.

Question 3

Beschrijf wat de bedoeling is van bovenstaande code. Uhh er wordt een integer gedefinieerd als answer. Als a groter is dan b dan gaan we een andere if-statement in. En als b groter is dan c dan komt antwoord c naar boven en anders komt antwoord b naar boven. Wat dus eigenlijk inhoudt dat die de kleinste waarde geeft. Hij output... Hij zet de laagste waarde van a, b en c in answer. Want hier is a groter dan b, hier is die groter dan c, dus als die er beide niet is dan is a de kleinste dus dan komt die in a te staan bij de else. Dus, de kleinste waarde van a,b,c wordt in answer gezet.

Question 4

Beschrijf wat de bedoeling is van bovenstaande code. De integer size wordt geïnitialiseerd en op 9 gezet. Er wordt een array, een integer array, gedefinieerd van 1, 2, 3... , 9 waarden. Dus waardes 0 tot en met 8. Boolean b wordt op true gezet. Er wordt een integer c gedefinieerd, die staat op 0. Als c kleiner is dan de integer 9 min 1, dus als die kleiner is dan 8 dan zit je dus in de while-loop. Dan is het als de array c en de c staat op 0, dus de eerste waarde in de array. Als die groter is dan de tweede waarde in de array, dan wordt b naar false gezet. En anders wordt c met 1 opgehoogd. Dat eerste is niet het geval, want... De waarde 0, de eerste waarde in de array, is kleiner dan de tweede waarde in de array. C wordt opgehoogd, dus we blijven in de loop zitten. De tweede waarde wordt met de derde waarde vergeleken, die is ... Dat is nog steeds niet zo, want de tweede waarde is kleiner dan de eerste waarde. En als ik doorkijk is dat eigenlijk voor alle waarden zo. Dus hij controleert of de waardes oplopen of niet, dat is eigenlijk wat deze code doet. Controleert of de waarden in de array oplopen, als dat zo is blijft die op true zitten en anders wordt die false. Dus hij kijkt gewoon of de waardes in de array oplopen.

Question 5

Beschrijf wat de bedoeling is van de bovenstaande code. Initialiseert de integer size met waarde 10, initialiseert een array van 10 waardes, en er zitten er ook 10 in. Uh, initialiseert integer a met de eerste waarde uit de array. Datzelfde doet die met integer b. Dus die worden allebei 5. Voor int i is 1 tot en met, tot dat i size min 1 is, dus dat die 9 is. Wordt i met 1 opgehoogd. En daarbinnen gaat die kijken of de array met positie of waarde 1, dus de tweede positie, of die groter is dan a. En a was gedefinieerd op 5. Dat is in dit geval niet zo, uhm. Dus anders zou die die array-waarde ophogen. Anders gaat die kijken of die groter... Of die kleiner is dan b. Als dat zo is, zet die in die array waarde een 1, verandert die b. Dus wat uiteindelijk de bedoeling is... Is dat de kleinste waarde uit de array in b komt te staan en de grootste waarde in a komt te staan. Dus int a vullen

met grootste waarde uit array en int b vullen met kleinste waarde uit het array.

Student 18

Question 1

Uhm, eens kijken. Een hele hoop integers. En dan... Dit lijkt heel erg op iets wat ik vanochtend ook heb gezien, maar het is net iets anders. Een waarde die hier verwisselt wordt... b krijgt de waarde van iets wat al gedefinieerd was... Vervolgens wordt a... wordt c naar 0 terug gezet. Vervolgens krijgt a de waarde die oorspronkelijk c had, of nouja, die dus nu... Vervolgens verandert er ook nog iets aan e. Nog een keer iets aan d en nog een keer iets aan c. Dus dit ga ik eigenlijk gewoon van boven naar beneden doorwerken, waarbij 1 verandering van b hebt. Wat eigenlijk al meteen 15 wordt. Dus dat wordt b is 15. Vervolgens verandert hier c. Daar verandert nog een keer c dus daar ga ik nog heel even mee wachten. Uhm. A wordt, nou, b was hier al 15 dus a wordt hier ook 15. Dan krijg je e wordt c plus 3, maar daar heeft c oorspronkelijk de waarde 0 gekregen dus dit is e wordt 3. Vervolgens wordt d de waarde van c. Dus nogmaals, c werd 0 dus d wordt ook 0. En dat krijgt vervolgens c de waarde die d heeft gekregen dat is dus ook 0.

Question 2

Uhm, ok. Kijk aan. A,b, en c zijn voor het bovenstaande stuk code geïnitialiseerde integers. Schrijf code die het effect van bovenstaande code ongedaan maakt. Dat wil zeggen code die ervoor zorgt dat a, b en c weer de originele waarde hebben. Ok ja. Uhm, aangezien dit van boven naar beneden gaat. Zou je dat ongedaan willen krijgen door eigenlijk de andere kant op te willen werken... Maar dat gaat ook even niet heel erg makkelijk. Even kijken. Je krijgt hier dan uiteindelijk een waarde van a is a plus b. b is b plus c. En c is c plus a. Ja... Dat is een interessante. In principe, als je een aantal dingen hiervan af gaat halen krijg je natuurlijk weer waardes die er oorspronkelijk uitkomen, dus als je bijvoorbeeld b van a afhaalt krijg je daaruit krijg je... a min c. En als je daar c weer bij optelt krijg je... 2a uit. Dus dat zou a min b plus c zijn. Is 2a. Als je dat dan keer een half doet krijg je de oorspronkelijke waarde van a er weer uit. Dus dit zou vervolgens zijn voor a is een half a min b plus c. En vervolgens is het volgens mij doorrouleren. Dat wil ik wel even voor de zekerheid checken. Als je hier b min c plus a doet dan krijg je... Daar valt een c weg... Daar valt een a weg... Dan krijg je de b er inderdaad uit. Dus dan krijg je b is een half b min c plus a. En vervolgens heb je voor c dan hetzelfde nog een keer doorrouleren als het goed is. Dan krijg je als je daar a vanaf haalt dan vallen de a'tjes weg en als je er b bij optelt dan vallen vervolgens weer de b'tjes weg. En

hou je daar een c over. Dus dan... c min a plus b. Dan zou je op deze manier weer de originele waardes terug moeten krijgen.

Question 3

Dan vervolgens... Een hele hoop if-statements. Beschrijf wat de bedoeling is. Ok, ja. Dus. We kijken hier eerst of a groter is dan b, dan kijken we of b groter is dan c, en als dat is komt er c uit. Dus in dit geval zou c de kleinste waarde van de 3 zijn. En als dit niet het geval is, zou dit betekenen dat b de kleinste waarde van de 3 is. Want op het moment dat a groter is dan b maar dan b niet groter dan c dan is b de kleinste. Uhm, vervolgens gaan we hier natuurlijk kijken of dat als dat niet waar is, of a groter is dan c. Dus op het moment dat a kleiner is dan b en a groter is dan c, dan komen we opnieuw uit bij c is de kleinste. Dan is het b a c in volgorde en als dat niet waar is, is a de kleinste. Dus de bedoeling van de bovenstaande code is om de kleinste waarde van de 3 variabelen te vinden.

Question 4

Dan hebben we een integer en een array, en een boolean. En nog een integer. Vervolgens gaan we een while-loopje doen. Opnieuw natuurlijk met de bedoeling. Uhm, vervolgens is het.. zo lang het nummer van.. Of zo lang we nog niet op de laatste plek van de array zijn aangekomen, dan gaan we iedere keer kijken of... Ah check, de bedoeling van deze code is om te kijken of de nummer geordend zijn op het moment. En dan strikt geordend zijn, dus er mogen ook geen 2 dezelfde waardes achter elkaar komen, want daar zit dat kleiner dan teken... Of het groter dan teken, anders moet het groter of gelijk aan zijn. Dus in dit geval kijkt het array of de eerste waarde iedere keer, of de waarde die je op dat moment checkt groter is dan de waarde die daarna komt. Dus hij wil kijken of het een aflopende array is van grote getallen die steeds kleiner worden. Wat in dit geval natuurlijk vals terugkeert. Want dit is allereerst omgekeerd. Want dit zou dan van 102 tot 2 terug moeten lopen. En we hebben daar een dubbele waarde en de dubbele waarde gaat er natuurlijk sowieso al mis omdat dit een groter dan teken is ... Maar het uiteindelijke doel van deze code is om te kijken of het een strikt aflopende... Of de array strikt aflopende integerwaardes heeft. Dus ook geen dubbele waardes. Eens checken of ik dat wel daadwerkelijk goed heb.. Het is wel ja inderdaad, c en c plus 1 en het loopt op dus ja. Dit zou het moeten zijn.

Question 5

Voor de laatste hebben we nog een keertje een array. Dit keer een niet zo mooi geordende variant. Want we zien hier allemaal getalletjes... Wel allemaal positief. En we hebben a en b die in eerste instantie geïnitialiseerd

worden op het nulde element, dus die zijn hier in ieder geval allebei 5. Vervolgens gaan we weer de hele array door. Want dat loopt van i is 1 tot i is size min 1. i is 1 kan je natuurlijk al beginnen omdat je bij de nulde al de integers al gedefinieerd hebt. Vervolgens kijken we of array i groter is dan a. Waarbij je uiteindelijk dus kijkt... Hier krijg je bij a uiteindelijk de maximale waarde, want iedere keer dat in de array het getalletje groter is dan a wordt die vervangen. En b precies het tegenovergestelde. Daar wordt de minimale waarde van de array uitgehaald, omdat op het moment dat b.. Of het moment dat de array-waarde kleiner is dan b. Ik zeg het volgens mij precies andersom. Maar, ff kijken. Als een waarde kleiner is dan b, dan wordt b wel ja precies. Dus b wordt de maximale waarde en a wordt de minimale waarde. Even kijken, ja. Want als de waarde groter is dan a, dan verandert die a. En als de waarde kleiner is dan b... Dit zou ik heel makkelijk moeten kunnen. Als die groter dan a... Als die groter is dan vervangt die hem, maar dan wordt... Nee het is wel zo, a wordt maximaal en b wordt de minimale waarde. Dus a krijgt de maximale waarde van de array en b de minimale waarde.

Student 21

Question 1

Integer a is nul, even kijken... Wat zijn de waarden van a, b, c, d en e, na het uitvoeren van de bovenstaande code. A is nul. B is .. uh alleen aangeroepen. C is 15. d is 23. e is 4. a is c. c is.. a is b .. e is c plus 3. d is c. c is d. Even kijken. Dus.. Ik mag meeschrijven hè? Even kijken. Dus b is 15. c is 0. a is 15. e is ... uhm, 0 plus 3 is 3. d is.. ook c, en c was 0. c is d en d heb ik net 0 gemaakt. Dus wat zijn de waarden van a, b en c dan. Nou, a is 15. b is ook 15. c heb ik staan op 0. d is 0. e heb ik staan op 3.

Question 2

a, b en c zijn voor het bovenstaande stuk code geïnitialiseerd... Integers.. Schrijf code die het effect van bovenstaande code ongedaan maakt. Derwijze code die ervoor zorgt dat a, b en c weer de originele waarde hebben. A is a plus b. b is b plus c. c is c plus a. Uhm, even kijken. A is a plus b. b is b plus a. c ... Eens kijken, dan zullen we het waarschijnlijk van achter naar voren moeten doen. Uhm a is niet... a is wel veranderd. B is... ook veranderd. Even kijken, we kunnen uhm... Het is moeilijker als je hardop denkt. Uhm. C is c plus a. a is ook al veranderd. B is b plus c. Uhm. Dus dan is b ook veranderd. Dus ik schrijf een code hieronder op die het hierna weer omdraait. Oh, dan is het c... Want die a is nog steeds niet veranderd. Dan c is uhm... even kijken, hier was het c plus a, dus dan c min a. Dan heb je b is b min c, want c is weer de goeie. En dan heb je hetzelfde met a. Dus a min b. Want b is ook weer de goeie waarde.

Question 3

Integer answer... Beschrijf wat de bedoeling is van de bovenstaande code. Dus je hebt een integer als answer. Als a is groter dan b. En als b is groter dan c. Dan is het antwoord c. Als het... Anders is het antwoord b. Als a meteen groter is dan c, dan is het antwoord c. Anders is het antwoord a. Dus hij returned 3 antwoorden, a, b en c... Uhm. Even kijken. Dus, a zal een... a heeft een bepaalde waarde. Als a groter is dan b of als a groter is dan c. Kan a groter zijn dan allebei... Ja want dan is het antwoord c. Ok. Als b groter is dan c. Dan is het antwoord c, anders is het b. Even kijken... Het antwoord is b als b groter is dan ... Als b kleiner is dan a... En groter is dan c. Else... als a groter is dan c, het antwoord is c. even kijken... Het antwoord is c als a groter is dan c... Want hier is a groter dan b, dus ook groter dan c. Als a groter is dan c, dan is het antwoord sowieso c. En het antwoord is a, als a kleiner is dan b en kleiner dan c... Beschrijf wat de bedoeling is van bovenstaande code... Nou, de bovenstaande code die, uhm, geeft een waarde aan integer answer en het answer hangt af van de waarde van a. Als a groter is dan b en groter is dan c, dan returned die a. Als a groter is dan... Als a kleiner is dan b... Ja. Als a groter is dan c, returned die c. En als uh, a groter is dan b en als a... Dan is die eigenlijk ook groter dan c. Maar b is groter dan c, dan returned die b.

Question 4

Integer size 9... Beschrijf wat de bedoeling is van bovenstaande code. Size is 9... Je hebt een array met 1, 2, 3, ... ,8 , 9 waardes.. Je hebt boolean b is true. Integer c is 0. Zolang 0... en dan telt die door tot en met 9 waardes... Hij begint bij 0. Als de array c... Die begint ook op 0. Als die groter is dan de array c plus 1. Als deze waarde groter is dan deze waarde... Dan is het false. Daarna is het c wordt 1 groter... C plus plus. Dus deze code gaat de array af. Hij begint op de nulde positie. Code gaat array af... Vanaf positie 0... Tot en met uhm... positie 9, positie 8 is dat eigenlijk. En hij returned... Uhm, b is false.. Hij returned false... boolean b is true.. b is false... Wanneer het getal, het integer, op positie c groter is dan de positie daarna. Dus c plus 1. In dit geval zou dat niet zo zijn, want 7 is... Nou... Nee in dit geval zou die wel false returnen. Op deze positie. Op de derde.. tweede positie. Want 7 is niet groter dan 7. Dus hij returned false op positie uhm 2. Want 7 is niet groter dan 7.

Question 5

Beschrijf wat de bedoeling is van de bovenstaande code... Integer size is 10. De array heeft 1, 2, 3, ..., 9, 10 waardes. Uhm integer a is positie 0. Integer is ook op die positie. Voor integer i is 1 en i is kleiner dan de size min 1, dus i plus plus... Dus de for-loop gaat alle waardes af. Als die waarde

groter is dan a en a is 5. Dan a is die positie... En als de array i kleiner is dan b, dan b is array i. Uhm, het is een beetje onhandige code als ik het zo zie. Maar wat de code doet is, hij verandert... Alle waardes in de array want a en b is gelijk aan elkaar. Ze zijn allebei dezelfde positie, dus allebei 5. Dus hij verandert alle waardes in de array die niet 5 zijn... Die niet 5 zijn. En als de waarde groter is dan 5, dan wordt die vervangen met 5. Nee dan wordt... Oh... Ah, ik heb hem verkeerd begrepen. Hij verandert wel alle waardes in de array die niet... Die niet ok... Dus hij gaat per positie... Uh, dus a kan veranderen. Dus a is op het begin 5 en b is op het begin 5. Dus als de array i groter is dan 5, dus op de tweede positie. Oh nee op de 6e positie. Of als de array groter is dan 3. Dus het resultaat is.. Uh.. De a wordt de even kijken, als die groter is dan a... a wordt de grootste waarde in de array. Want als die uhm steeds grotere waarde tegenkomt, dan neemt die die over. Dus de grootste waarde van de array. En b doet het tegenovergestelde. En b wordt de kleinste waarde in de array.

Student 22

Question 1

Eens eventjes kijken. Uhm, in eerste instantie ga ik kijken welke... Wat hier allemaal gedeclareerd is... Ik zie dat a de waarde 0 krijgt. B heeft nog geen waarde gekregen. Waarde c heeft 15. d heeft 23 en e is 4. Vanaf dit moment gaat b een waarde krijgen. Dus b krijgt de waarde van c dus b krijgt de waarde 15. En c krijgt de waarde van a krijgt de waarde 0. a krijgt de waarde b. Dus wat we hier zien.. b heeft ondertussen waarde 15 gekregen dus a krijgt de waarde 15. Wat te zien is, is dat deze waardes dus gewoon een beetje geswitched zijn nu. Dat de waardes van a en c omgedraaid zijn met behulp van variabele b. Uh variabele e krijgt hier een andere waarde die krijgt de waarde van c die nu 0 is plus 3. Dus we houden uiteindelijk 3 over. Waarde d krijgt de waarde 0. En c krijgt weer de waarde d dus eventjes kijken... Dat zou gewoon precies hetzelfde moeten zijn dus dat zou 0 moeten zijn. Dus uiteindelijk houden we over... Voor a, b, c, d en e... Als ik me niet vergis. Voor a houden we uiteindelijk over 15. Dus de laatste keer dat a hier wordt gedeclareerd. Uh, de waarde b die hier helemaal onderaan staat ff kijken dat is hier de bovenste regel zeg maar is ook de waarde 15 gegeven. Dan gaan we de waarde c zoeken, die heeft waarde 0. D heeft waarde 0 en e heeft waarde 3.

Question 2

Uhm, even kijken. A,b en c zijn voor het bovenstaande stuk code geïnitialiseerd.. geïnitialiseerde integers, schrijf code die het effect van het bovenstaande stuk code ongedaan maakt. Dat wil zeggen code die ervoor zorgt dat a, b en c de originele waarde hebben. Uhm, eens eventjes kijken. Dit is denk ik wel

een opgave... Uh. Zo... Hoe wil je dit op een fatsoenlijke manier doen. Uhm, eens even kijken. Ik zoek nu dus eigenlijk een verband. Als a, a plus b is. Hoe kun je dan met behulp van de waarden van a plus b, b plus c en c plus a... Hoe kun je dan terug naar een beginwaarde voor a.. Moet eerlijk zeggen, op het moment zie ik het nog niet meteen. Dus ik denk dat het handig is als we deze even voor het laatst bewaren." [Hier maakt de subject eerst 3, 4 en 5.] "Eventjes kijken. Dit is toch een lastige... Als we het element b willen.. Stel we willen element b, hoe kunnen we dit fatsoenlijk doen? Dan zou ik zeggen... Tel a plus b bij elkaar op... Dan heb je a plus b, plus b plus c. Dus dan heb je a plus 2b plus c. Ik schrijf het even hier op... Ik maak het even a' en b'. Dus a plus 2b plus c. Trek je hier c' vanaf dat is c plus a. Dus wat je overhoudt is 2b. Dus wat we kunnen doen om b te bepalen is a' plus b' min c' gedeeld door 2 is b. Dus om b te bepalen, ik zal het even hier opschrijven, b is... a' plus b' ja klopt... Gedeeld door twee. Daar moet nog min c' bij. Ok, hetzelfde voor a. Wat zouden we dan moeten doen? Dan zouden we uh. Zoeken we de elementen waarbij er een a in staan dat zijn in dit geval a en c. Die tellen we bij elkaar op, dus a is a' plus c'. Wat we nu hebben is 2a plus b plus c. Dus we willen die b plus c weghebben dus moeten we er b' van af trekken. Dan houden we over 2a. Dat delen we door 2 en wat overblijft is a. Laatste wat we doen is c. Nu moeten we de elementen opzoeken waar uhm c in staan. Dat is in dit geval de nieuwe b, ofwel b'. En de nieuwe c of c'. Dus tellen we bij elkaar op, krijgen we c is c' plus b'. Dan hebben we a plus b plus 2c. Daar moeten we a plus b vanaf halen, dat doen we door a' ervan af te trekken en dat geheel delen we door 2. Als het goed is blijven zo onze a, b en c over. Dat gaan we kijken. A' is a plus b, plus c plus a. Dus ... 2a plus b plus c. Daar trekken we min b... Of trekken we b plus c vanaf. Dus wat we overhouden is 2a gedeeld door 2 is a. Voor zover ik weet geldt dat voor de andere twee ook dus ik denk dat 'ie zo goed is.

Question 3

Er wordt een integer answer wordt gemaakt, ik neem aan dat dat je output gaat worden. Als a groter is dan b en als b groter is dan c. Wordt waarde c je antwoord... Anders, als a groter is dan b, maar b kleiner of gelijk is aan c. Wordt b je antwoord... Anders als a groter is dan c, dus als uhm, a niet groter is dan b, maar wel groter is dan c, is je antwoord c. Anders is het antwoord a. Dus wat er eigenlijk wordt gekeken is uh, welke het laagste is van dit ding. Want als a groter is dan b dan kunnen we zien dat a geen antwoord gaat worden, dus dan gaat het tussen b en c. Als b groter is dan c, is het antwoord c. Dus de laagste wordt hier out-geput. Anders wordt b hier out-geput. Dus dat is in dat geval het laagste. Anders als a groter is dan c, wordt c out-geput. Dus dan ook de lagere van de 2. En anders wordt a out-geput. Dus wat hier gebeurt.. De laagste integer wordt gegeven als

antwoord. Als answer.

Question 4

Uhm. Er wordt een grootte van een array of ja, althans, int size wordt gegeven dus ik neem aan dat het uiteindelijk invloed gaat hebben op die array... Dan wordt een array gegeven met 2, 4, 6, 8, 9, gelukkig 9 variabelen. Uhm, een boolean die nu op true staat en een integer c die op 0 staat. Zolang c kleiner is dan size min 1. Uhm, dus we gaan in principe door tot het laatste element van de array. Aangezien het van 0 tot size min 1 loopt standaard. Als uhm.. Als element c groter is dan het element wat daarna komt, dan moet b false worden. Dus wat je hier doet is je gaat kijken of de volgorde van de integers die hier staan, of dat dat van klein naar groot is. Op het moment dat, even kijken, op het moment dat element c dus het element voor een ander element dat die groter is dan de ander dan wordt b false gemaakt. En ik neem aan dat die uiteindelijk wordt geoutput of dat dit hetgene is waar het hier om gaat. Dus er wordt gekeken of de element in de array van laag naar hoog lopen. Zo niet, boolean b is false. Dit doet die voor de gehele array... Dus voor de gehele array gaat die elementen controleren.

Question 5

We gaan weer de elementen tellen. 6,8,10. Gelukkig maar, een array van size 10. Uhm, met hierin waardes... We maken 2 nieuwe arrays, array a en array b. Uhm, voor integer i die vanaf element 2 loopt in principe. Want hij gaat van i is 1 tot size min 1. Dus hij slaat het eerste element over. Als het element van array...Oh sorry, ik zie het helemaal mis. Integer a is de beginwaarde, integer b is ook de beginwaarde... Van de array. Integer i is 1. Hij gaat dus lopen van het tweede element tot het laatste element. En hij gaat kijken, zolang het element wat hij gaat controleren groter is dan a. Groter is dan de beginwaarde in feite. Dan wordt a array i. Dus hij gaat elementen vergelijken en op het moment dat het element wat hij controleert groter is dan het element wat je hebt opgeslagen, dan wordt het element wat jij op dat moment aan het controleren bent wordt opgeslagen in integer a. Wat er dus uiteindelijk over blijft als je deze code hebt gerund is dat in integer a de hoogste waarde van de array staat en in integer b de laagste waarde van de array. Als ik me niet vergis... Uhm, ja. Dus dat gaan we opschrijven. Uiteindelijk... zal in int a de hoogste waarde van array van arr staan. In integer b zal de laagste waarde staan. Dit doet die door gewoon elke.. Door elke keer 1 element wat er na komt te vergelijken. En op het moment dat die groter is in het geval van a slaat hij die op als vergelijkingsmateriaal. En dan gaat die elk moment wat daarna komt daar ook weer mee vergelijken tot uiteindelijk de grootste overblijft. Bij b idemdito maar dan gaan we kijken naar de laagste.