Leibniz Universität Hannover

Programmieren 1: Abschlusstest 25.-26.2.2016

Human-Computer Interaction Group

Prof. Dr. Michael Rohs
michael.rohs@hci.uni-hannover.de

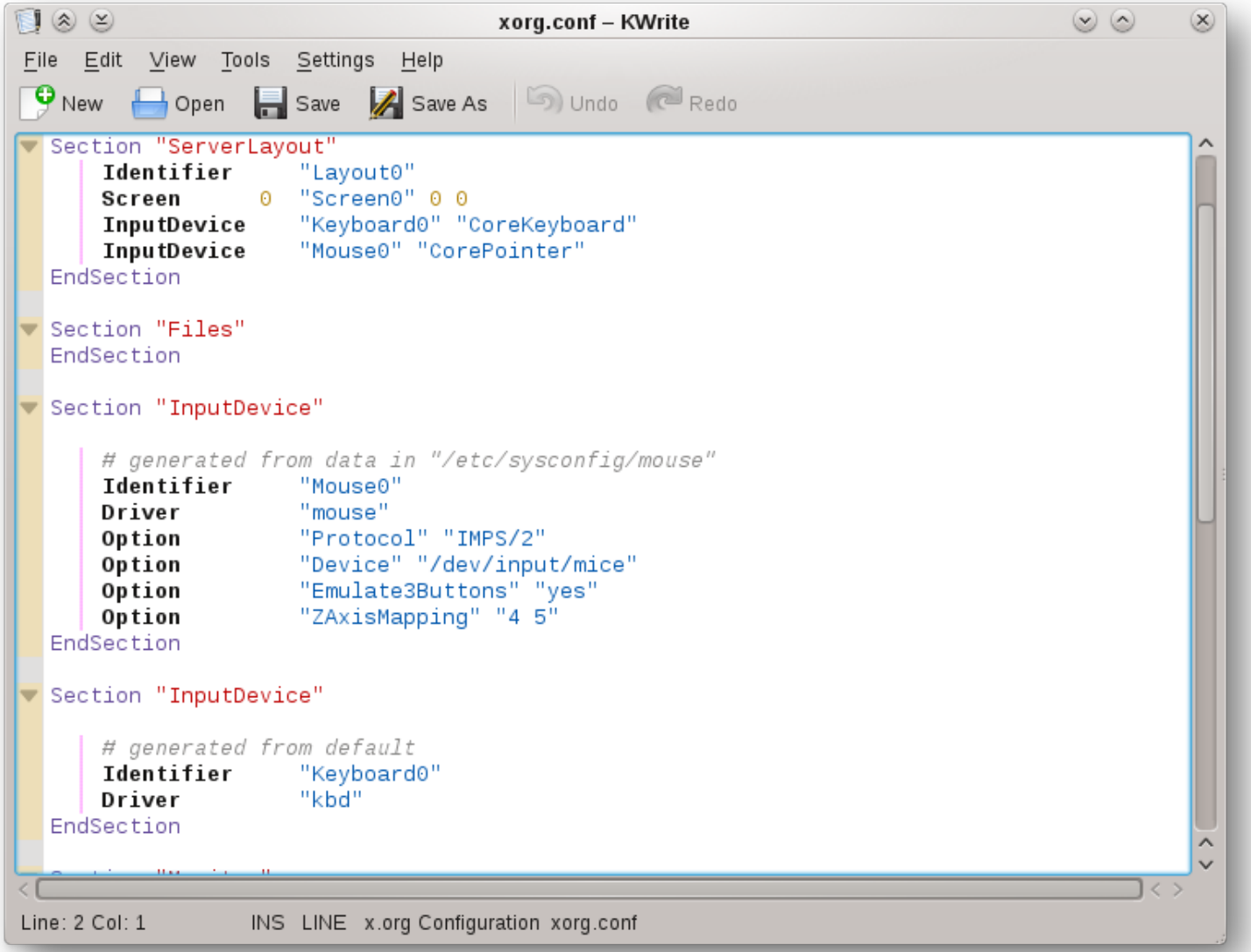
Abschlusstest: Ort und Zeit

- Ort: Rechnerraum F411, Hauptgebäude
  - dort haben auch die Tutorien stattgefunden
- Zeit: Do 25.2., Fr 26.2.
- 12 Gruppen
  - Rechnerraum, begrenzte Anzahl Plätze
  - daher verschiedene Gruppen und Zeiten
  - die Gruppeneinteilung werden wir in Kürze über Stud.IP mitteilen
- 5 Minuten vor Beginn vor Raum F411 sein
- Studierendenausweis und Personalausweis mitbringen

Programmieraufgaben am Rechner

- Abschlusstest an Linux-PCs
- wir erstellen Account für Prüfung
- Texteditor "KWrite" mit Syntax-Highlighting
- Kommandozeile mit C-Compiler (gcc),
  Java-Compiler (javac) und Java-Interpreter (java)
- Aufgabentext Deutsch
- Stift für Notizen erlaubt, keine weiteren Hilfsmittel erlaubt
- Notizen werden nicht mitbewertet
- Mobiltelefon ausschalten!

Editor: kWrite



Abschlusstest

- 60 Minuten Dauer, gesamte Zeit im Raum bleiben
- erfolgreiche Teilnahme an Abschlusstest
  - 2 Programmieraufgaben (mit je 2 Teilen) lösen
- eine Aufgabe C (1a, 1b), eine Aufgabe Java (2a, 2b)
- beide Aufgaben erfolgreich bearbeiten
- falls Bonus erlangt wurde, darf eine als Bonusaufgabe markierte Teilaufgabe weggelassen werden
- melden, wenn fertig, so dass wir Lösung überprüfen können

Name: _____ Matrikelnummer: _____

PC, Gruppe: _____ Unterschrift: _____

Von den Prüfern auszufüllen:	☐ bestanden	☐ nicht bestanden

Leibniz Universität Hannover

Deckblatt

Leibniz Universität Hannover
Fachgebiet Mensch-Computer-Interaktion

Programmieren 1
Dozent: Prof. Dr. Michael Rohs

Abschlusstest
25. Februar 2016, Gruppe 1

Dies ist ein **60**-minütiger Test. Tragen Sie bitte Ihren Namen, Ihre Matrikelnummer, die PC-Nummer und die Nummer der Gruppe auf diesem Blatt ein und unterschreiben Sie es. Legen Sie bitte Ihren Studierenden- und Personalausweis zur Anwesenheitskontrolle bereit. Es sind keine zusätzlichen Hilfen erlaubt. Schalten Sie bitte Ihr Mobiltelefon aus. Für Notizen können Sie diesen Zettel verwenden. Notizen auf diesem Zettel werden nicht mitbewertet.

Ablauf:

1. Bearbeiten Sie alle Aufgaben.
2. Wenn Sie einen Bonus erworben haben, dürfen Sie **eine** als Bonusaufgabe markierte Aufgabe auslassen.
3. Bitte melden Sie sich, wenn Sie fertig sind.
4. Legen Sie diesen Zettel ausgefüllt für uns zur Mitnahme bereit.
5. Nach der Bewertung bitte ausloggen, aber nicht ausschalten!
6. Verlassen Sie den Raum bitte nicht vor dem Ende des Tests.

Hinweise:

- Sie kommen in das Template-Verzeichnis, indem Sie ein Terminal öffnen (siehe Verknüpfung auf dem Desktop) und `cd Desktop/GruppeX` eingeben.
- Mit `ls` können Sie die Dateien im Verzeichnis auflisten.
- Als Tools dürfen Sie den Texteditor KWrite, den Java-Compiler (javac) und Java-Interpreter (java), sowie den C-Compiler (gcc), verwenden.
- Es empfiehlt sich nicht, Umlaute im Quelltext zu verwenden.
- Sofern in der Aufgabenstellung nicht explizit gefordert, brauchen Sie keine setter- oder getter-Methoden zu implementieren.
- Ihre Lösungen müssen mit den jeweiligen Testfällen funktionieren, müssen aber generelle Lösungen sein.

Seite 2

1. C
   (a) Implementieren Sie die Funktion...
   Hinweis: ...

   (b) *(bei Bonus: entweder 1b oder 2b auslassen)* Implementieren Sie die Funktion...
   Hinweis: ...

   Die Template-Datei für diese Aufgabe ist my_file.c. Anweisungen zum Kompilieren und Ausführen finden Sie in der Template-Datei.

2. Java
   (a) Implementieren Sie die Methode...
   Hinweis: ...

   (b) *(bei Bonus: entweder 1b oder 2b auslassen)* Implementieren Sie die Methode...
   Hinweis: ...

   Die Template-Datei für diese Aufgabe ist MyFile.java. Anweisungen zum Kompilieren und Ausführen finden Sie in der Template-Datei.

my_file.c

```
/*
Compile: make my_file
Run: ./my_file
Compile and run:
make my_file && ./my_file
*/

#include "base.h"
#include "string.h"

...

int main(void) {
    my_function_test();
    return 0;
}
```

MyFile.java

```
/*
Compile: javac -cp .:prog1javalib.jar MyFile.java
Run: java -cp .:prog1javalib.jar MyFile
Compile and run:
javac -cp .:prog1javalib.jar MyFile.java &&
                                java -cp .:prog1javalib.jar MyFile
*/

import prog1.base.Base;

public class MyFile {
    ...

    public static void main(String[] args) {
        test();
    }
}
```

Vorbereitung

- Vorlesungsfolien durchgehen
- Review-Fragen beantworten
- Übungen rekapitulieren
- Übungsfolien durchgehen

BEISPIELAUFGABE 1 (JAVA)

Beispielaufgabe (Java)

Implementieren Sie die Methode

`int countLeftSumLargerThanRightSum()`.

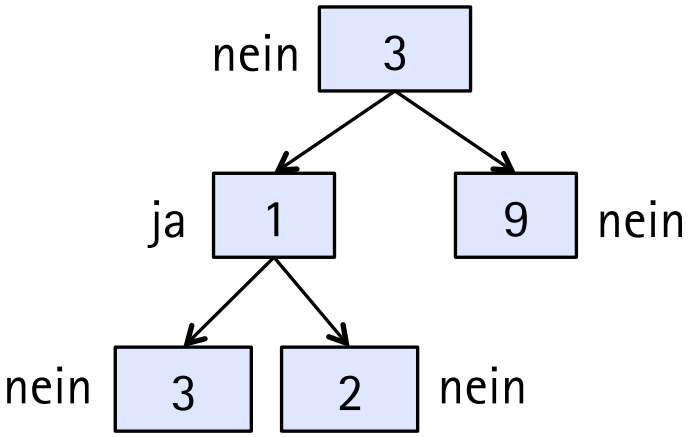
Diese soll die Anzahl der Knoten des Baums zurückgeben, für die die Summe der Werte im linken Unterbaum größer ist als die Summe der Werte im rechten Unterbaum.

Die Summe eines leeren Unterbaums ist 0.

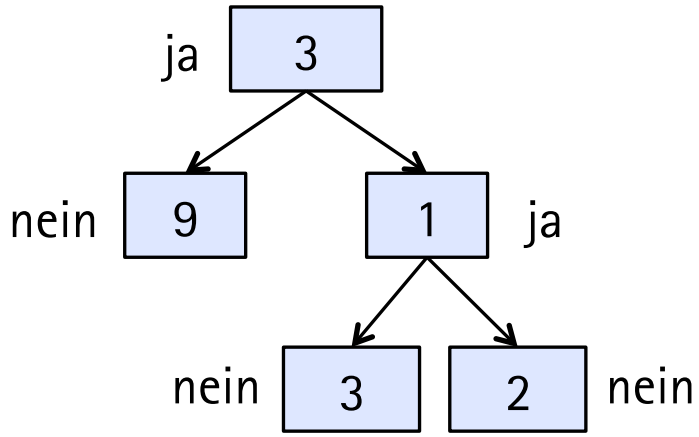
Sie dürfen eine Hilfsmethode implementieren.

Beispielbäume

- Summe der Werte im linken Unterbaum größer als Summe der Werte im rechten Unterbaum?
- Anzahl Knoten, die das Kriterium erfüllen?
- (Die Summe eines leeren Unter-baums ist 0.)



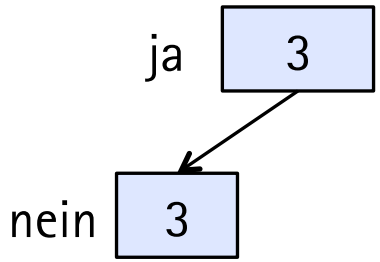
Anzahl der Knoten: 1

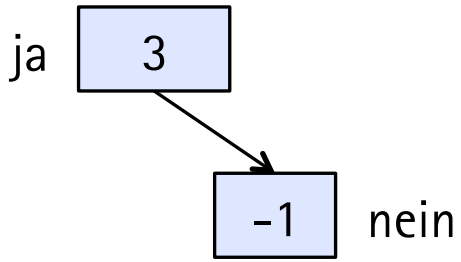


Anzahl der Knoten: 2

Beispielbäume

- Summe der Werte im linken Unterbaum größer als Summe der Werte im rechten Unterbaum?
- Anzahl Knoten, die das Kriterium erfüllen?
- (Die Summe eines leeren Unter-baums ist 0.)





Anzahl der Knoten: 1

Anzahl der Knoten: 1

Binary Tree Node

```
class Node {
    public int value;
    public Node left, right;
    public Node(Node l, int v, Node r) {
        this.left = l;
        this.value = v;
        this.right = r;
    }
    public int countLeftSumLargerThanRightSum() {
        return 0; // todo: implement
}}
```

How?

Implementing countLeftSumLargerThanRightSum?

Four cases depending on the available of left/right subtree:

```
public int countLeftSumLargerThanRightSum() {
    if (left == null && right == null) { ...
    } else if (left != null && right == null) { ...
    } else if (left == null && right != null) { ...
    } else /* left != null && right != null */ { ...
    }
}
```

Implementing countLeftSumLargerThanRightSum?

```
public int countLeftSumLargerThanRightSum() {
    if (left == null && right == null) {
        return 0;
    } else if (left != null && right == null) { ...
    } else if (left == null && right != null) { ...
    } else /* left != null && right != null */ { ...
    }
}
```

Implementing countLeftSumLargerThanRightSum?

```
public int countLeftSumLargerThanRightSum() {
    if (left == null && right == null) {
        return 0;
    } else if (left != null && right == null) {
            return left.countLeftSumLargerThanRightSum()
                + (left.sum() > 0 ? 1 : 0);
    } else if (left == null && right != null) { ...
    } else /* left != null && right != null */ { ...
    }
}
```

recursive call on the left subtree

sum() is a helper function (to be implemented)

Implementing countLeftSumLargerThanRightSum?

```
public int countLeftSumLargerThanRightSum() {
    if (left == null && right == null) {
        return 0;
    } else if (left != null && right == null) { ...
    } else if (left == null && right != null) {
        return right.countLeftSumLargerThanRightSum()
            + (right.sum() < 0 ? 1 : 0);
    } else /* left != null && right != null */ { ...
    }
}
```

Implementing countLeftSumLargerThanRightSum?

```
public int countLeftSumLargerThanRightSum() {
    if (left == null && right == null) {
        return 0;
    } else if (left != null && right == null) { ...
    } else if (left == null && right != null) { ...
    } else /* left != null && right != null */ {
        return left.countLeftSumLargerThanRightSum()
            + right.countLeftSumLargerThanRightSum()
            + (left.sum() > right.sum() ? 1 : 0);
    }
}
```

Implementing sum?

- Four cases depending on the available of left/right subtree

```
public int sum() {
    if (left == null && right == null) { ...
    } else if (left != null && right == null) { ...
    } else if (left == null && right != null) { ...
    } else /* left != null && right != null */ { ...
    }
}
```

Implementing sum?

- Four cases depending on the available of left/right subtree

```
public int sum() {
    if (left == null && right == null) {
        return value;
    } else if (left != null && right == null) {
        return left.sum() + value;
    } else if (left == null && right != null) { ...
        return value + right.sum();
    } else /* left != null && right != null */ { ...
        return left.sum() + value + right.sum();
}  }
```

General Strategy with Binary Trees

- Methods may be placed in class Node (as here) or in a class Tree
- List all cases for the structure of the tree (4 cases)
  - No children
  - Left child only
  - Right child only
  - Both left and right child
- Handle each case
- Handle subtrees in recursive calls
- Implement helper functions to compute required aspects

Leibniz Universität Hannover

BEISPIELAUFGABE 2 (JAVA)

Beispielaufgabe (Java)

a) Gegeben ist ein Suchbaum zum Speichern von ganzen Zahlen. Implementieren Sie die Methode void insert(int v) der Klasse Node, die den Wert v in den Suchbaum einfügt. Dabei soll die Suchbaumeigenschaft erhalten bleiben.

b) Implementieren Sie die Methode int depth(), die die Tiefe eines Binärbaums zurückgibt.

Search Tree

- Search tree criterion: For a node n with value v, all nodes in left subtree have values w with $w < v$ and all nodes in the right subtree have values w with $w \geq v$.
- Search tree insertion: If $w < v$ insert in n.left else insert in n.right. Special cases if n.left and/or n.right do not exist.

Search Tree Insertion

```
class Node {
    public int value;
    public Node left, right;
    public Node(Node l, int v, Node r) {
        this.left = l;
        this.value = v;
        this.right = r;
    }
    public Node(int v) {
        this.value = v;
        left = null;
        right = null;
    }
```

```
public void insert(int v) {
    if (v < value) {
        if (left != null) {
            left.insert(v);
        } else {
            left = new Node(v);
        }
    } else {
        if (right != null) {
            right.insert(v);
        } else {
            right = new Node(v);
}   }   }
```

Node Methods

```
public int depth() {
    if (left == null && right == null) {
        return 1;
    }
    else if (left != null && right == null) {
        return 1 + left.depth();
    }
    else if (left == null && right != null) {
        return 1 + right.depth();
    }
    else /* left != null && right != null */ {
        return 1 + Math.max(left.depth(), right.depth());
    }
}
```

General approach:
Look at all cases based on the structure of the tree.

Here: 4 cases derived from structure of tree

BEISPIELAUFGABE (C)

Beispielaufgabe (C)

Implementieren Sie die Funktion double **distSecondSmallestToSmallest(List *list)**.

Diese soll die absolute Differenz zwischen dem kleinsten Element und dem zweitkleinsten Element der Liste zurückgeben. Bei einer zu kurzen Liste soll der Wert 0 zurückgegeben werden.

List and Node Structures

```
typedef struct Node {
    double value;
    struct Node *next;
} Node;

typedef struct List {
    Node *first;
    Node *last;
} List;
```

double distSecondSmallestToSmallest(List *list)

```
// return 0 if list is too short (no list or zero or one elements)
if (list == NULL || list->first == NULL || list->first->next == NULL) {
    return 0.0;
}
// assert: list has at least 2 elements
...
```

double distSecondSmallestToSmallest(List *list)

- Need to identify smallest (min1) and second smallest (min2) element (min2 ≥ m1)
- Result is min2 - min1
- Need to look at each element of list to decide which one is smallest (and which one is second smallest)
- Use a loop
  - New element might be smaller than smallest (or second smallest) seen so far
  - Update smallest and/or second smallest if necessary

double distSecondSmallestToSmallest(List *list)

```
double min1 = ?;
double min2 = ?;
...
// need to look at each element
for (Node *node = list->first; node != NULL; node = node->next) {
    double v = node->value;
    // update min1 and/or min2 if necessary...
}
return min2 - min1; // result is min2 - min1
```

Subproblem: Find Smallest Element

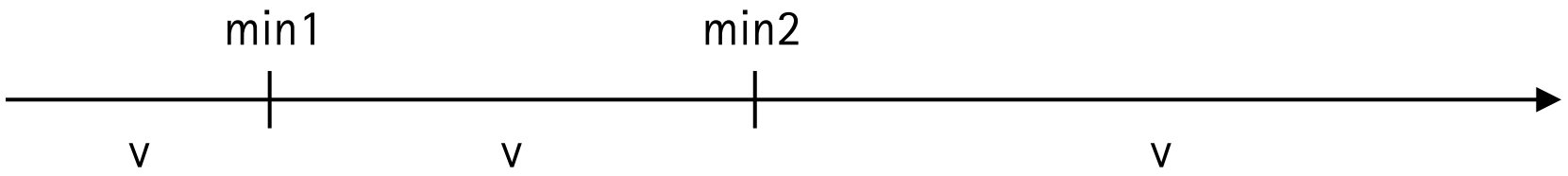
```
double min1 = ?;
// need to look at each element
for (Node *node = list->first; node != NULL; node = node->next) {
    double v = node->value;
    // update min1 if necessary...
    if (v < min1) {
        min1 = v;
    }
}
return min2 - min1; // result is min2 - min1
```

Subproblem: Find Smallest Element

```
double min1 = 0;
bool min1valid = false; // not valid yet
// need to look at each element
for (Node *node = list->first; node != NULL; node = node->next) {
    double v = node->value;
    // update min1 if necessary...
    if (v < min1 || !min1valid) {
        min1 = v;
        min1valid = true; // min1 is now valid
    }
}
```

min1 and min2

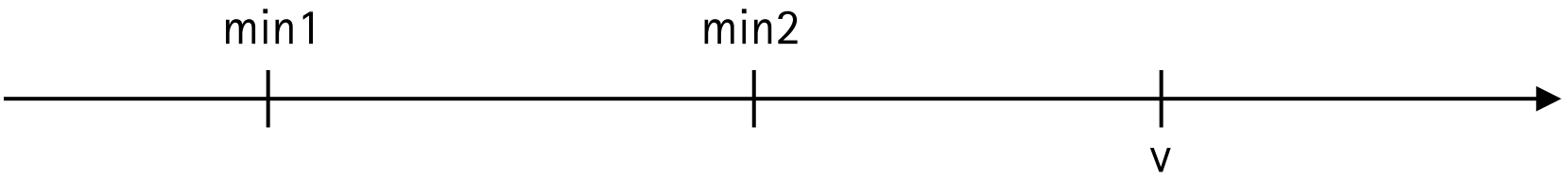
- min1: The smallest element seen so far
- min2: The second smallest element seen so far



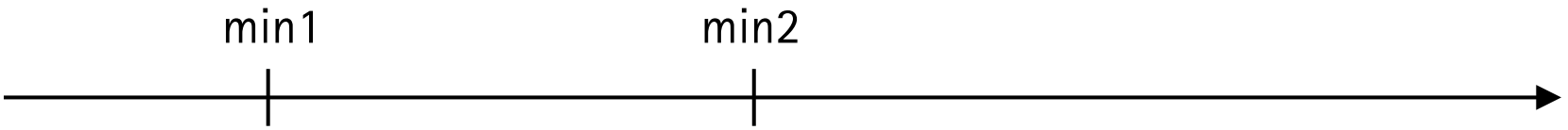
- 3 cases for next list element v
  - v ≥ min2 → no change
  - min1 ≤ v < min2 → min1 no change, min2 = v
  - v < min1 → min1 = v, min2 = old value of min1

min1 and min2

- min1: The smallest element seen so far
- min2: The second smallest element seen so far

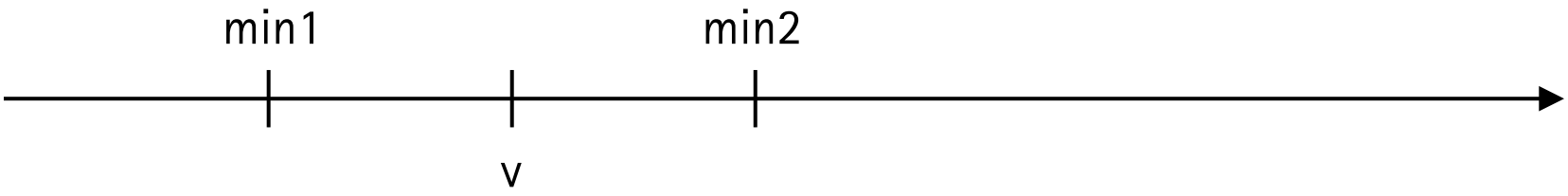


- Case 1: v ≥ min2 → no change

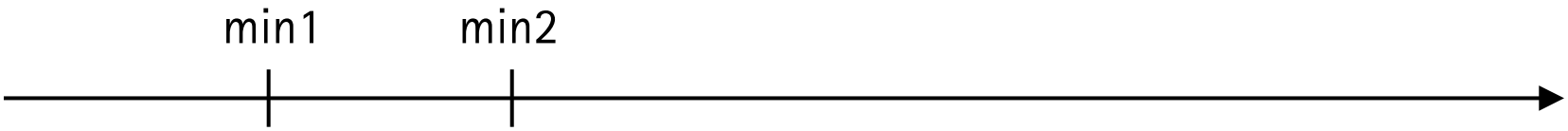


min1 and min2

- min1: The smallest element seen so far
- min2: The second smallest element seen so far

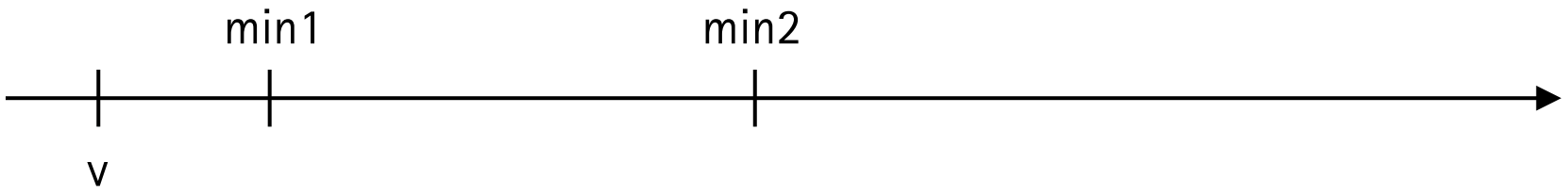


- Case 2: min1 ≤ v < min2 → min1 no change, min2 = v

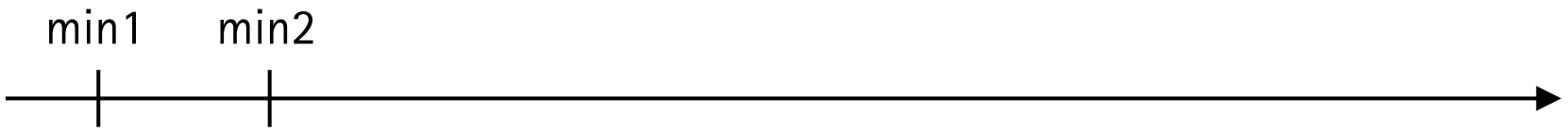


min1 and min2

- min1: The smallest element seen so far
- min2: The second smallest element seen so far



- Case 3: v < min1 → min1 = v, min2 = old value of min1



Find Smallest and Second Smallest Element

```
double min1 = 0;
bool min1valid = false; // not valid yet
double min2 = 0;
bool min2valid = false; // not valid yet
// need to look at each element
for (Node *node = list->first; node != NULL; node = node->next) {
    double v = node->value;
    // update min1 and/or min2 if necessary...
    ...
}
return min2 - min1; // result is min2 - min1
```

Find Smallest and Second Smallest Element

```
double v = node->value;
if (v < min1 || !min1valid) { // v less than min1
    min2 = min1; // old value of min1
    min1 = v;
    min2valid = min1valid; // old value of min1valid
    min1valid = true;
} else if (v < min2 || !min2valid) {// v between min1 and min2
    min2 = v;
    min2valid = true;
}
```